s

**INTERNATIONAL PROGRAMMERS TRAINING CENTER**

**SOFTECH APTECH**

**---- ----**

**EPROJECT: HOTEL MANAGEMENT**

**Lecturer:** Mr Nguyen Duc Thao

**Group Name:** 1

**Batch:** 146

**Semester:** 2

 **Members:**

1. Ngo Thien An

2. Nguyen Thanh Kien

3. Doan Thanh Nhan

**DANANG, VIETNAM   
APR 2019**

|  |
| --- |
| **SOFTECH APTECH** |
| **EProject Document** |
| **HOTEL MANAGEMENT** |
|  |
| |  |  | | --- | --- | | Group Name: Group 1 | | | Group members | 1. 1. Ngo Thien An 2. 2. Nguyen Thanh Kien ( C ) 3. 3. Doan Thanh Nhan | | Instructor | Mr.Nguyen Duc Thao | | Batch | 146 | | Semester | 2 | |
|  |

**Danang, Vietnam, 04/2019**

**TEACHER’S EVALUATION**

……………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

Table of Contents

[ACKNOWLEDGEMENTS 6](#_Toc524985327)

[ABSTRACT 7](#_Toc524985328)

[CHAPTER 1: INTRODUCTION 9](#_Toc524985329)

[1.1 HTLM5: 9](#_Toc524985330)

[1.2 CSS- CSS3: 10](#_Toc524985331)

[1.3. BOOSTRAP 3: 12](#_Toc524985332)

[1.4. ANGULARJS: 14](#_Toc524985333)

[CHAPTER 2: PROBLEM DEFINITION 19](#_Toc524985334)

[2.1. The Home Page should be created making use of Frames with a suitable logo: 19](#_Toc524985335)

[2.2. The look and feel of the project must be made user friendly and uniformity in the color combination must be maintained: 19](#_Toc524985336)

[2.3. About us, Contact Us (address and map) must be added. In the Contact Us page the website of all the shops location must be given: 19](#_Toc524985337)

[2.4. All the categories of the products like HAIR, SKIN, MAKE-UP KITS, NAIL PAINTS, NAIL ARTS, JEWELLRY, WEDDING KITS, UPCOMING PRODUCTS & so on. For each product the cost must be mentioned in the list: 19](#_Toc524985338)

[2.5. Feedback page must be added to take the valuable feedback from the users: 19](#_Toc524985339)

[2.6. Contact Us, About Us: 19](#_Toc524985340)

[2.7. Use GeoLocation API to show the address or location of SIMPLON’s Beauty care center on the maps: 20](#_Toc524985341)

[2.8. Gallery section can be included: 20](#_Toc524985342)

[2.9. Navigation of the project must be made user friendly: 20](#_Toc524985343)

[2.10. Customer must be allowed to place the order by selecting the product: 20](#_Toc524985344)

[2.11. While placing the dummy order, the total cost of the order must be calculated: 20](#_Toc524985345)

[CHAPTER 3: REQUIREMENT ATTRIBUTES 21](#_Toc524985346)

[3.1. Main Requirements: 21](#_Toc524985347)

[3.2. Critical Use Case Specification: 21](#_Toc524985348)

[3.3. Use case Diagram: 26](#_Toc524985349)

[CHAPTER 4: DESIGN PROTOTYPE 27](#_Toc524985350)

[4.1. Website mapping: 27](#_Toc524985351)

[4.2. Home page prototype: 28](#_Toc524985352)

[4.3. Products page prototype: 29](#_Toc524985353)

[4.4. Product detail page prototype: 30](#_Toc524985354)

[4.5. Contact us page prototype: 31](#_Toc524985355)

[4.6. About us page prototype: 32](#_Toc524985356)

[CHAPTER 5: TECHNICAL 33](#_Toc524985357)

[5.1. Bootstrap framework: 33](#_Toc524985358)

[5.2. AngularJS framework: 33](#_Toc524985359)

[5.3. Node JS: 33](#_Toc524985360)

[CHAPTER 6: THE PAGES BASIC OF WEBSITE: 34](#_Toc524985361)

[6.1. Home page screenshot: 34](#_Toc524985362)

[6.2. Products page screenshot: 35](#_Toc524985363)

[6.3. Product detail page screenshot: 36](#_Toc524985364)

[6.4. Checkout page screenshot: 37](#_Toc524985365)

[6.5. Contact us page screenshot: 38](#_Toc524985366)

[6.6. About us page screenshot: 39](#_Toc524985367)

[6.7. Payment page screenshot: 40](#_Toc524985368)

[CHAPTER 7: USER GUIDE 41](#_Toc524985369)

[7.1. Instructrion: 41](#_Toc524985370)

[7.2. Running step by step: 41](#_Toc524985371)

[7.2.1. Download project: 41](#_Toc524985372)

[7.2.1.1. Without GIT: 41](#_Toc524985373)

[7.2.1.2. Using GIT: 41](#_Toc524985374)

[7.2.2. Running server: 41](#_Toc524985375)

[7.2.3. Running web pages: 42](#_Toc524985376)

[7.3. Developer’s guide: 42](#_Toc524985377)

# ACKNOWLEDGEMENTS

In the process of completing this project, we have received great deal of helps, guidance and encouragements from teachers and friends.

First of all, I would like to express my deepest thanks to my instructor**, Mr.Nguyen Duc Thao** who given me suggestions on how to shape the study and always been most willing and ready to give me valuable advice, helpful comments as well as correction our project.

Next, we would like to express my gratitude to all teachers as well as other parts of Softech Aptech centers have created the best conditions to help me complete this project.

**LIST OF FIGURES**

[Figure 1.1: HTML5 Logo 9](#_Toc524816751)

[Figure 1.2: CSS 3 Logo 10](#_Toc524816752)

[Figure 1.3: Bootstrap 3 Logo 12](#_Toc524816753)

[Figure 1.4: AngularJS Logo 14](#_Toc524816754)

[Figure 1.5: MVC software Architecture Pattern 16](#_Toc524816755)

[Figure 1.6: MVC Example in AngularJS 17](#_Toc524816756)

[Figure 1.7: Relationships among Components 18](#_Toc524816757)

[Figure 3.1: User use case diagram 25](#_Toc524816758)

[Figure 4.1: Website mapping 26](#_Toc524816759)

[Figure 4.2: Homepage prototype 27](#_Toc524816760)

[Figure 4.3: Products page prototype 28](#_Toc524816761)

[Figure 4.4: Product detail page prototype 30](#_Toc524816762)

[Figure 4.5: Contact us page prototype 32](#_Toc524816763)

[Figure 4.6: About us page prototype 34](#_Toc524816764)

[Figure 5.1: Link bootstrap.min.js 35](#_Toc524816765)

[Figure 5.2: Link Angular.js 35](#_Toc524816766)

[Figure 5.3: Node JS version 36](#_Toc524816767)

[Figure 6.1: Homepage 37](#_Toc524816768)

[Figure 6.2: Products page 38](#_Toc524816769)

[Figure 6.3: Product detail page screenshot 40](#_Toc524816770)

[Figure 6.4: Checkout page screenshot 41](#_Toc524816771)

[Figure 6.5: Contact us page screenshot 43](#_Toc524816772)

[Figure 6.6: About us page screenshot 45](#_Toc524816773)

# CHAPTER 1: INTRODUCTION

## 1.1 JAVA:



***Figure 1.1: Java Logo***

Java is a programming language created by James Gosling from Sun Microsystems (Sun) in 1991. The target of Java is to write a program once and then run this program on multiple operating systems. The first publicly available version of Java (Java 1.0) was released in 1995. Sun Microsystems was acquired by the Oracle Corporation in 2010. Oracle has now the steermanship for Java. In 2006 Sun started to make Java available under the GNU General Public License (GPL). Oracle continues this project called OpenJDK.

Over time new enhanced versions of Java have been released. The current version of Java is Java 12 which is also known as Java 12.(2019)

Java is defined by a specification and consists of a programming language, a compiler, core libraries and a runtime (Java virtual machine) The Java runtime allows software developers to write program code in other languages than the Java programming language which still runs on the Java virtual machine. The Java platform is usually associated with the Java virtual machine and the Java core libraries.

The Java language was designed with the following properties:

* Platform independent: Java programs use the Java virtual machine as abstraction and do not access the operating system directly. This makes Java programs highly portable. A Java program (which is standard-compliant and follows certain rules) can run unmodified on all supported platforms, e.g., Windows or Linux.
* Object-orientated programming language: Except the primitive data types, all elements in Java are objects.
* Strongly-typed programming language: Java is strongly-typed, e.g., the types of the used variables must be pre-defined and conversion to other objects is relatively strict, e.g., must be done in most cases by the programmer.
* Interpreted and compiled language: Java source code is transferred into the bytecode format which does not depend on the target platform. These bytecode instructions will be interpreted by the Java Virtual machine (JVM). The JVM contains a so called Hotspot-Compiler which translates performance critical bytecode instructions into native code instructions.
* Automatic memory management: Java manages the memory allocation and de-allocation for creating new objects. The program does not have direct access to the memory. The so-called garbage collector automatically deletes objects to which no active pointer exists.

The Java syntax is similar to C++. Java is case-sensitive, e.g., variables called myValue and myvalue are treated as different variables.

## 1.2 JAVAFX:



***Figure 1.2: JavaFX Logo***

JavaFX is a set of Java graphics libraries for creating Java GUI applications, just like Java AWT and Swing.

JavaFX was originally targeted for Rich Interface Application (RIA, introduced in 2002), i.e., GUI webapp delivered thru a browser's plugin (competing with Adobe Flash, Microsoft Silverlight and Java Applets). However, the trend today is to use HTML5/JavaScript-based, instead of plug-in-based framework. Moreover, browsers (such as Firefox) has stopped supporting plug-ins (such as Java Plug-in for Applets).

Sun Microsystems created the Java Programming Language and presented JDK 1.0 in 1995/96. To support GUI programming, Java introduced AWT (Abstract Windowing Toolkit) in JDK 1.1 (1997), and Swing in JDK 1.2 (1998). But many developers felt Swing was over-complex and Java on the desktop never really took off as it did on the server.

Sun Microsystems tried several ways to make it easier to create Java GUI applications. One of these was a scripting language called JavaFX Script 1.0 (2008) which allows developers to build much more complex user Interfaces. But JavaFX Script was not Java. It is a totally new language and never really caught on with Java developers.

When Oracle acquired Sun Microsystems, they killed off JavaFX as a scripting language but added its functionality into the Java Language as JavaFX 2.0 (2011). They enhanced it as the new way to develop user interfaces, intended to replace Swing. Starting from JDK 8 (2014), JavaFX was part of JDK (as JavaFX 8).

Oracle will continue to maintain the Swing library but will not enhance it. Swing and JavaFX can be used together. But for writing new Java applications, JavaFX is recommended as it offers a much simpler way to create desktop applications, and you can write more powerful applications with much less code.

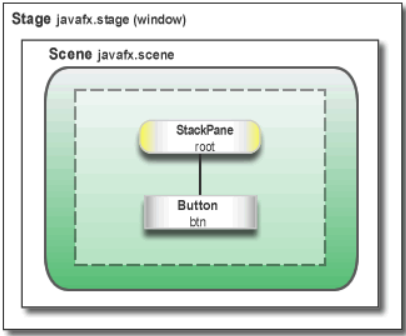
### 1.2.1. ****Architecture****:

**JavaFX uses hardware accelerated graphics pipeline for the rendering, known as Prism**. What’s more, to fully accelerate the graphics usage, it leverages either software or hardware rendering mechanism, by internally using DirectX and OpenGL.

**JavaFX has a platform dependent Glass windowing toolkit layer to connect to the native operating system**. It uses the operating system’s event queue to schedule thread usage. Also, it asynchronously handles windows, events, timers.

The Media and Web engines enable media playback and HTML/CSS support.

Let’s see what the [*main structure*](https://docs.oracle.com/javafx/2/get_started/img/helloworld_scenegraph.png) of a JavaFX application looks like:



***Figure 1.2.1: JavaFX*** [***main structure***](https://docs.oracle.com/javafx/2/get_started/img/helloworld_scenegraph.png)

Here, we notice two main containers:

* **Stage is the main container and the entry point of the application**. It represents the main window and passed as an argument of the start() method.
* Scene is a container for holding the UI elements, such as Image Views, Buttons, Grids, TextBoxes.

The Scene can be replaced or switched to another Scene. This represents a graph of hierarchical objects, which is known as a Scene Graph. Each element in that hierarchy is called a node. A single node has its ID, style, effects, event handlers, state.

Additionally, the Scene also contains the layout containers, images, media.

## 1.2.2. Threads:

At the system level,**the JVM creates separate threads for running and rendering the application**:

* Prism rendering thread – responsible for rendering the Scene Graph separately.
* Application thread – is the main thread of any JavaFX application. All the live nodes and components are attached to this thread.

## 1.2.3. Lifecycle:

The javafx.application.Application class has the following lifecycle methods:

* **init() – is called after the application instance is created**. At this point, the JavaFX API isn’t ready yet, so we can’t create graphical components here.
* start(Stage stage) – all the graphical components are created here. Also, **the main thread for the graphical activities starts here.**
* stop() – is called before the application shutdown; for example, when a user closes the main window. It’s useful to override this method for some cleanup before the application termination.

The static launch() method starts the JavaFX application.

## 1.2.4. FXML:

**JavaFX uses a special FXML markup language to create the view interfaces.**

This provides an XML based structure for separating the view from the business logic. XML is more suitable here, as it’s able to quite naturally represent a Scene Graph hierarchy.

Finally, to load up the .fxml file, we use the FXMLLoader class, which results in the object graph of the scene hierarchy.

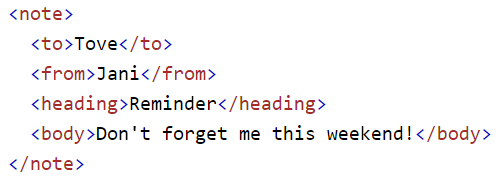
## 1.3. XML:

**1.3.1. What is XML:**

* XML stands for *e****X****tensible* ***M****arkup* ***L****anguage*
* XML is a markup language much like HTML
* XML was designed to store and transport data
* XML was designed to be self-descriptive
* XML is a W3C Recommendation

**1.3.2. XML does not do anything:**

Maybe it is a little hard to understand, but XML does not DO anything.



***Figure 1.3.2:*** *This note is a note to Tove from Jani* stored as XML

The XML above is quite self-descriptive:

* It has sender information.
* It has receiver information
* It has a heading
* It has a message body.

But still, the XML above does not DO anything. XML is just information wrapped in tags.

## 1.3.3. The difference between XML and HTML:

XML and HTML were designed with different goals:

* XML was designed to carry data - with focus on what data is
* HTML was designed to display data - with focus on how data looks
* XML tags are not predefined like HTML tags are

## 1.3.4. XML does not use predefined Tags:

The XML language has no predefined tags.

The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.

HTML works with predefined tags like <p>, <h1>, <table>, etc.

With XML, the author must define both the tags and the document structure.

## 1.3.5. XML is Extensible:

Most XML applications will work as expected even if new data is added (or removed).

Imagine an application designed to display the original version of note.xml (<to> <from> <heading> <body>).

Then imagine a newer version of note.xml with added <date> and <hour> elements, and a removed <heading>.

## 1.3.6. XML Simplifies Things:

* It simplifies data sharing
* It simplifies data transport
* It simplifies platform changes
* It simplifies data availability

Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost.

XML stores data in plain text format. This provides a software- and hardware-independent way of storing, transporting, and sharing data.

XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

With XML, data can be available to all kinds of "reading machines" like people, computers, voice machines, news feeds, etc.

## 1.4. SQL:

SQL is a standard language for accessing and manipulating databases.



***Figure 1.4: Microsoft SQL Server Logo***

**1.4.1. What is SQL?**

* SQL stands for Structured Query Language
* SQL lets you access and manipulate databases
* SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

**1.4.2. What can SQL do?**

* SQL can execute queries against a database
* SQL can retrieve data from a database
* SQL can insert records in a database
* SQL can update records in a database
* SQL can delete records from a database
* SQL can create new databases
* SQL can create new tables in a database
* SQL can create stored procedures in a database
* SQL can create views in a database
* SQL can set permissions on tables, procedures, and views

**1.4.3. SQL Server intructions:**

SQL Server is a relational database management system (RDBMS) developed by Microsoft. It is primarily designed and developed to compete with MySQL and Oracle database.

SQL Server supports ANSI SQL, which is the standard SQL (Structured Query Language) language. However, SQL Server comes with its own implementation of the SQL language, T-SQL (Transact-SQL).

**T-SQL** is a Microsoft propriety Language known as **Transact-SQL.** It provides further capabilities of declaring variable, exception handling, stored procedure, etc.

SQL Server Management Studio (SSMS) is the main interface tool for SQL Server, and it supports both 32-bit and 64-bit environments.

## 1.4.4. History SQL Server:

Microsoft and Sybase released version 1.0 in 1989. However, the partnership between these two ended in the early 1990s. Microsoft maintained ownership rights to the name SQL Server. Since the 1990s, subsequent versions of SQL Server have been released including SQL Server 2000, 2005, 2008, 2012, 2014, 2016 and 2017.

## 1.4.5. SQL Server Editions:

Following editions are available

**SQL Server Enterprise:**It is used in the high end, large scale and mission Critical business. It provides High-end security, Advanced Analytics, Machine Learning, etc.

**SQL Server Standard:**Itis suitable for Mid-Tier Application and Data marts. It includes basic reporting and analytics.

**SQL Server WEB:**It is designed for a low total-cost-of-ownership option for Web hosters. It provides scalability, affordability, and manageability capabilities for small to large scale Web properties.

**SQL Server Developer:**It is similar to an enterprise edition for the non-production environment. It is mainly used for build, test, and demo.

**SQL Server Express:**It is for small scale applications and free to use.

## 1.5. SQL Azure:

## 1.5.1. Introductions:

Cloud computing is a technology trend, that involves the delivery of software, platforms, and infrastructure as service through the Internet or networks. Windows Azure is a key offering in Microsoft’s suite of cloud computing products and services. The database functions of Microsoft’s cloud platform are provided by Windows Azure SQL Database, which is commonly known as SQL Azure.

SQL Azure can be used to store and manage data using queries and other functions that are similar to SQL Server2012. The data on SQL Azure does not have the constraint of being location-specific. This means that the data stored in SQL Azure can be viewed and edited from any location, as the entire data is stored on cloud storage platform.



***Figure 1.5.1: New Microsoft Azure Logo***

## 1.5.2. SQL Azure:

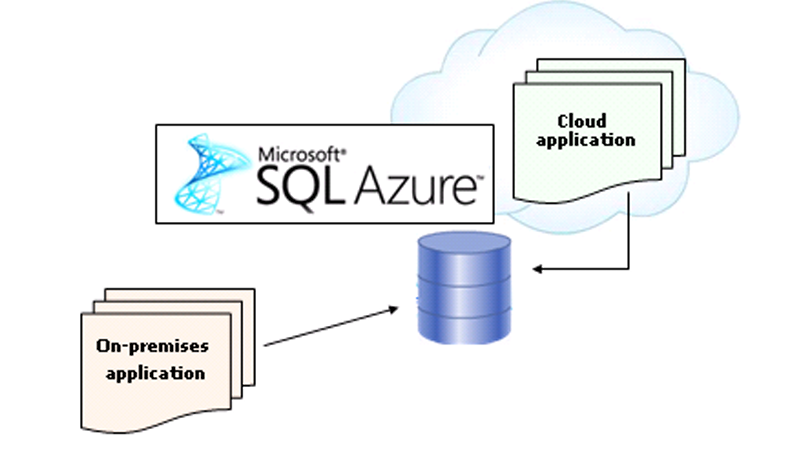
Consider a scenario of the Income Tax department. During the month of March, the department is flooded with heavy workload. During the rest of the year, the workload may be less. As a result, resources, server, and computing power are under-utilized during those months and over-utilized during peak periods. In such a scenario, using a cloud database service such as SQL Azure can help in optimal use of resources only as and when required.

SQL Azure is a cloud based relational database service that leverages existing SQL Server technologies. Microsoft SQL Azure extends the functionality of Microsoft SQL Server for developing applications that are Web-based, scalable, and distributed.

SQL Azure enables users to perform relational queries, search operations, and synchronize data with mobile users and remote back offices. SQL Azure can store and retrieve both structured and unstructured data.

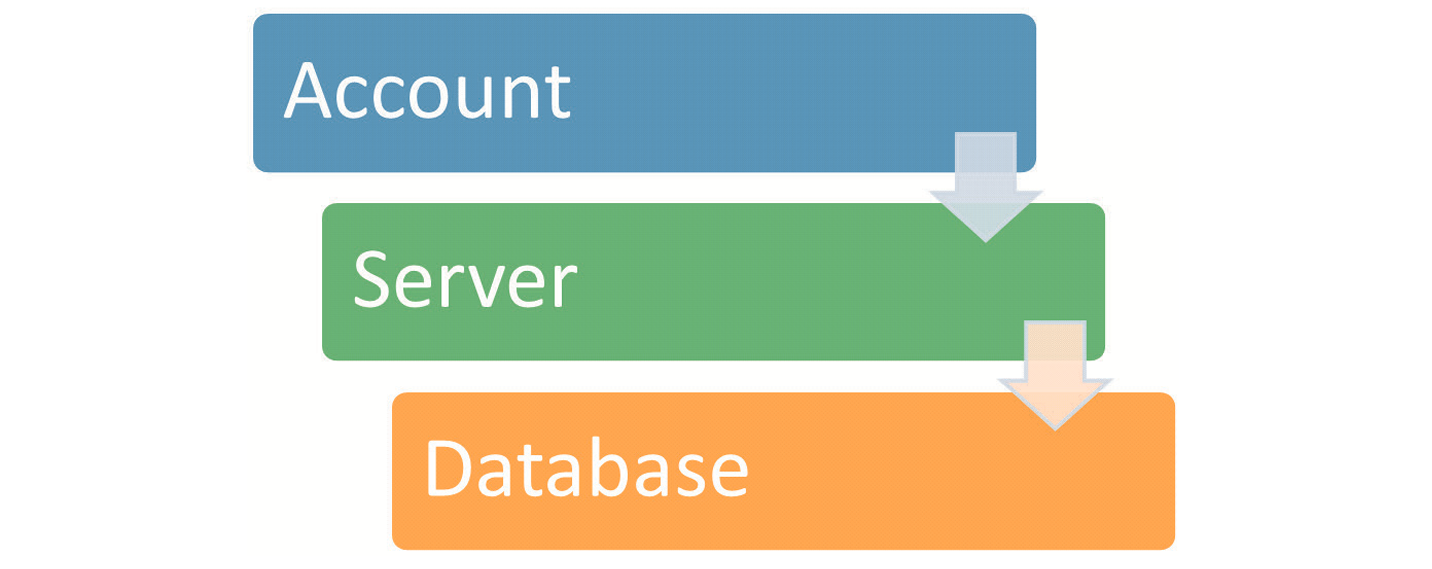
Both cloud based as well as on-premises applications can use the SQL Azure database.

Applications retrieve data from SQL Azure through a protocol known as Tabular Data Stream (TDS). This protocol is not new to SQL Azure. Whenever on-premises applications involve interaction with SQL Server Database Engine, this protocol is used by the client and the server.



***Figure 1.5.2.a: Simplified View of SQL Azure Architecture***

The process of SQL Azure operation is explained in the model as show in below figure:



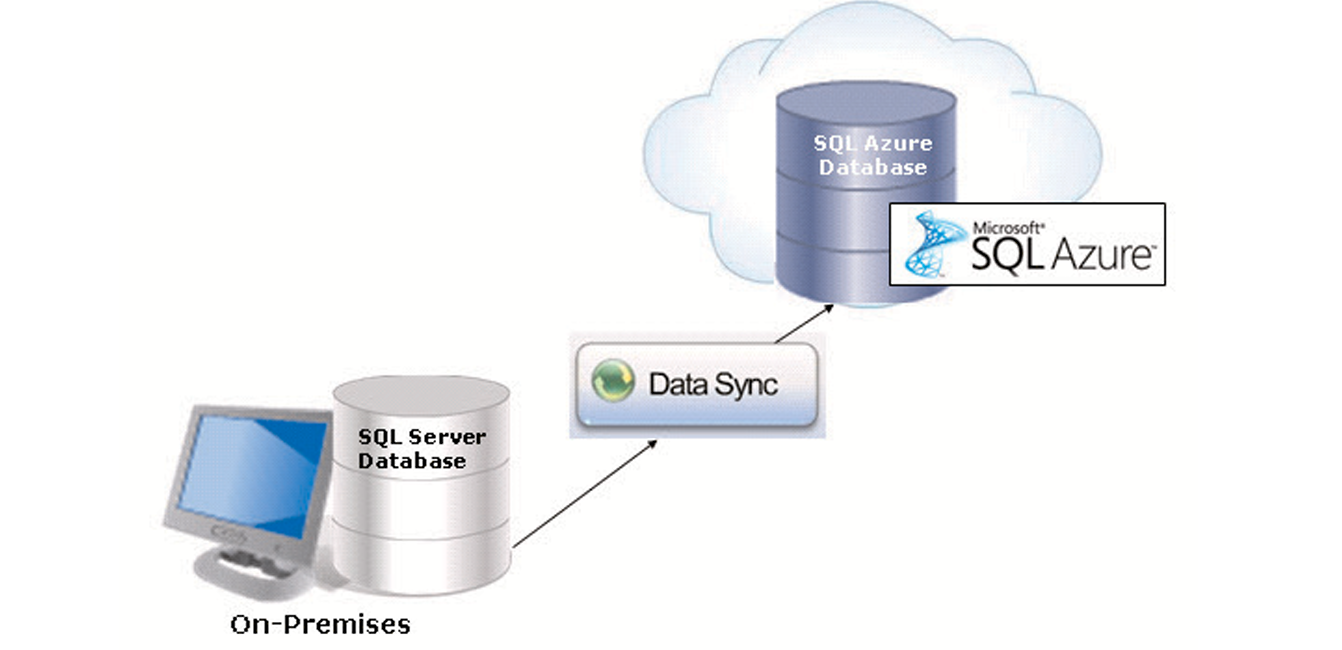
***Figure 1.5.2.b: SQL Azure Operation Model***

The three core objects in the SQL Azure operation model are as follows:

* + **Account** – An SQL Azure account must first be created before adding servers that will help to store and manage the data. This account is created for billing purposes. The subscription for an account is recorded and metered and an individual is charged according to the usage. To create an account, the credentials need to be provided. After the user account is created, the requirements need to be provided for the SQL Azure database. This includes the number of databases required, database size, and so on.
  + **Server** – The SQL Azure server is the object that helps to interact between the account and the database. After the account is registered, the databases are configured using the SQL Azure server. Other settings such as firewall settings and Domain Name System (DNS) assignment are also configured in the SQL Azure server.
  + **Database** – The SQL Azure database stores all the data in a similar manner as any on-premises SQL Server database would store the data. Though present on the cloud, the SQL Azure database has all functionalities of a normal RDBMS such as tables, views, queries, functions, security, settings, and so on.

In addition to these core objects, there is an additional object in SQL Azure. This object is the SQL Azure Data Sync technology. The SQL Azure Data Sync technology is built on Microsoft Sync Framework and SQL Azure database.

SQL Azure Data Sync helps to synchronize data on the local SQL Server with the data on SQL Azure as shown in figure below:



***Figure 1.5.2.c: Data Sync***

Data Sync also has data management capabilities that help to share data easily between different SQL Databases. Data Sync is not only used for synchronizing on-premises to SQL Azure, but also to synchronize on SQL Azure account to another.

## 1.5.3. Benefits of SQL Azure:

The benefits of using SQL Azure are as follows:

* + **Lower cost –** SQLAzure provides several functions similar to on-premises SQL Server at a lower cost when compared on-premises instances of SQL Server. Also, as SQL Azure is on the cloud platform, it can be accessed from any location. Hence, there is no additional cost required to develop a dedicated IT infrastructure and department to manage the databases.
  + **Usage of TDS –** TDS isused in on-premises SQL Server databases for client libraries. Hence, most developers are familiar with TDS and its uses. The same kind of TDS interface is used in SQL Azure to build client libraries. Hence, it is easier for developers to work on SQL Azure.
  + **Automatic failover measures –** SQL Azure stores multiple copies of data on different physical locations. Even if there is a hardware failure due to heavy usage or excessive load, SQL Azure helps to maintain the business operations by providing availability of data through other physical locations. This is done by using the automatic failover measures that are provided in SQL Azure.
  + **Flexibility in service usage –** Even small organizations can use SQL Azure as the pricing model for SQL Azure is based on the storage capacity that is used by an organizations. If the organization needs more storage, the price can be altered to suit the need. This helps organizations to be flexible in the investment depending on the service usage.
  + **Transact-SQL support** – As Azure is completely based on relational database model, it also supports Transact-SQL operations and queries. This concept is similar to the working of the on-premises SQL Servers. Hence, administrators do not need any additional training or support to use SQL Azure.

## 1.5.4. Different between SQL Azure and On-Premises SQL Server:

The major difference between SQL Azure and on-premises SQL Server is the presence of physical hardware and storage. Some other key distinctions between SQL Azure and on-premises SQL Server are as follows:

* + **Tools** – On-premises SQL Server provides a number of tolls for monitoring ad management. All these tools may not be supported by SQL Azure as there are a limited set of tools that are available in this version.
  + **Backup** – Backup and restore function must be supported in on-premises SQL Server for disaster recovery. For SQL Azure, as all the data is on the cloud platform, backup and restore is not required.
  + **USE Statement** – The USE statement is not supported by SQL Azure. Hence, the user cannot switch between databases in SQL Azure as compared to on-premises SQL Server.
  + **Authentication** – SQL Azure supports only SQL Server authentication and on-premises SQL Server supports both SQL Server authentication and Windows Authentication.
  + **Transact-SQL support** – Not all Transact-SQL functions are supported by SQL Azure.
  + **Accounts and Logins** – In SQL Azure, administrative accounts are created in the Azure management portal. Hence, there are no separate instance-level user logins.
  + **Firewalls** – Firewalls settings for allowed ports and IP addresses can be managed on physical servers for on-premises SQL Server. As an SQL Azure database is present on cloud, authentication through logins is the only method to verify the user.

# CHAPTER 2: PROBLEM DEFINITION

*This project has the following functionalities:*

## 2.1. The ‘Admin’ is the person who will be operating all functions of the application. Admin can register himself/herself with the application. This registration will enable him/her to log in to the system and use the application:

## Admin can control all functions of the application such as creating, editing, deleting,…

Admin must be create before log in.

## 2.2. Users who will be operating some functions of the application:

## Users can only use functions that are set by Admin

## 2.3. Customers are the people on whose behalf the ‘Admin/Retailer’ booking, check in, use services, check out and payment:

Customer contain some personal information such as name, birthday…

## 2.4. Employees who working at hotel:

Each employee will have an account to use the application, and this account can be create by admin or user who have permission to create new account.

**2.5. Department:**

Specialized functional area within an organization or a division, such as accounting, marketing, planning. Generally every department has its own manager and chain of command.

Department contains some roles for employees who will be set to the department.

**2.6. Roles:**

Have 2 cases:

- Group role (Department roles)

- User role (Employee roles)

All of these roles can be set dynamically.

**2.7. Rooms:**

Room can be create easily and contains many useful information.

**2.8. Room type:**

Don’t allow creating room type due to UI of the application, but room type can be edited some information such as price, discount…

**2.9. Service type:**

User can create, edit service type

**2.10. Service orders:**

User can create, edit service orders

**2.11. Service order detail:**

User can create, edit service orders

**2.12. Booking:**

Allow customer booking to reserve a room.

**2.13. Check in:**

Allow customer check in to stay.

**2.14. Check out:**

Allow customer check out and do payment.

**2.15. Bill:**

Contain information of bills.

**2.16. User logs:**

Recording all action of user like adding, editing, deleting…

**2.17. Application have basic CRUD functions**

**2.18. Sending user account and random password to the employee’s email**

**2.19. Creating and sending QR code and booking ID to customer’s email**

**2.20. Filtering function**

**2.21. Printing bill for customer after check out, and sending it to email if customer need**

**2.22. Can create database by the application if the database does not exist.**

**2.23. Allow choosing remote or local server.**

**2.24. Encrypt secret information such as password, secret questions…**

# CHAPTER 3: REQUIREMENT ATTRIBUTES

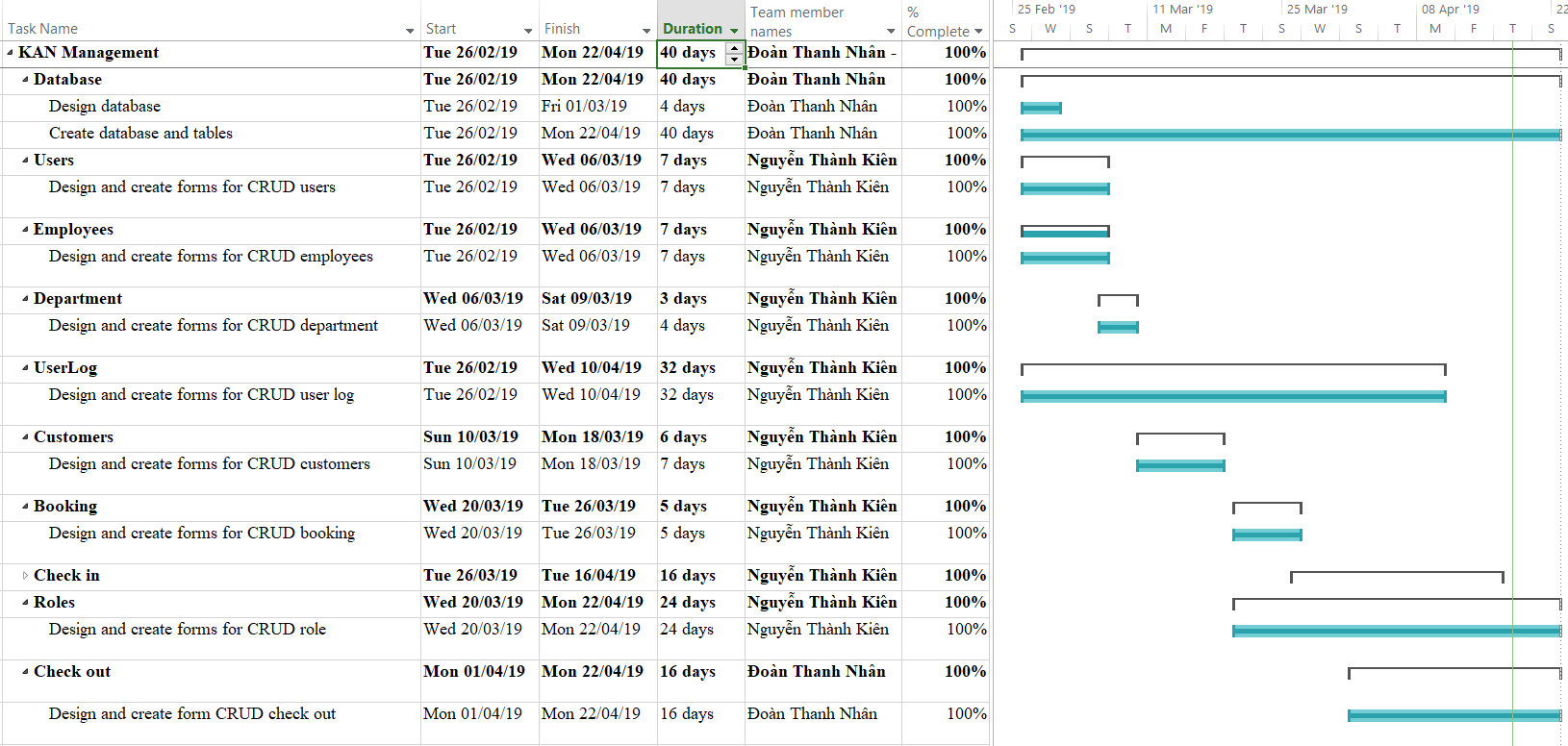
## 3.1. Use case Diagram:



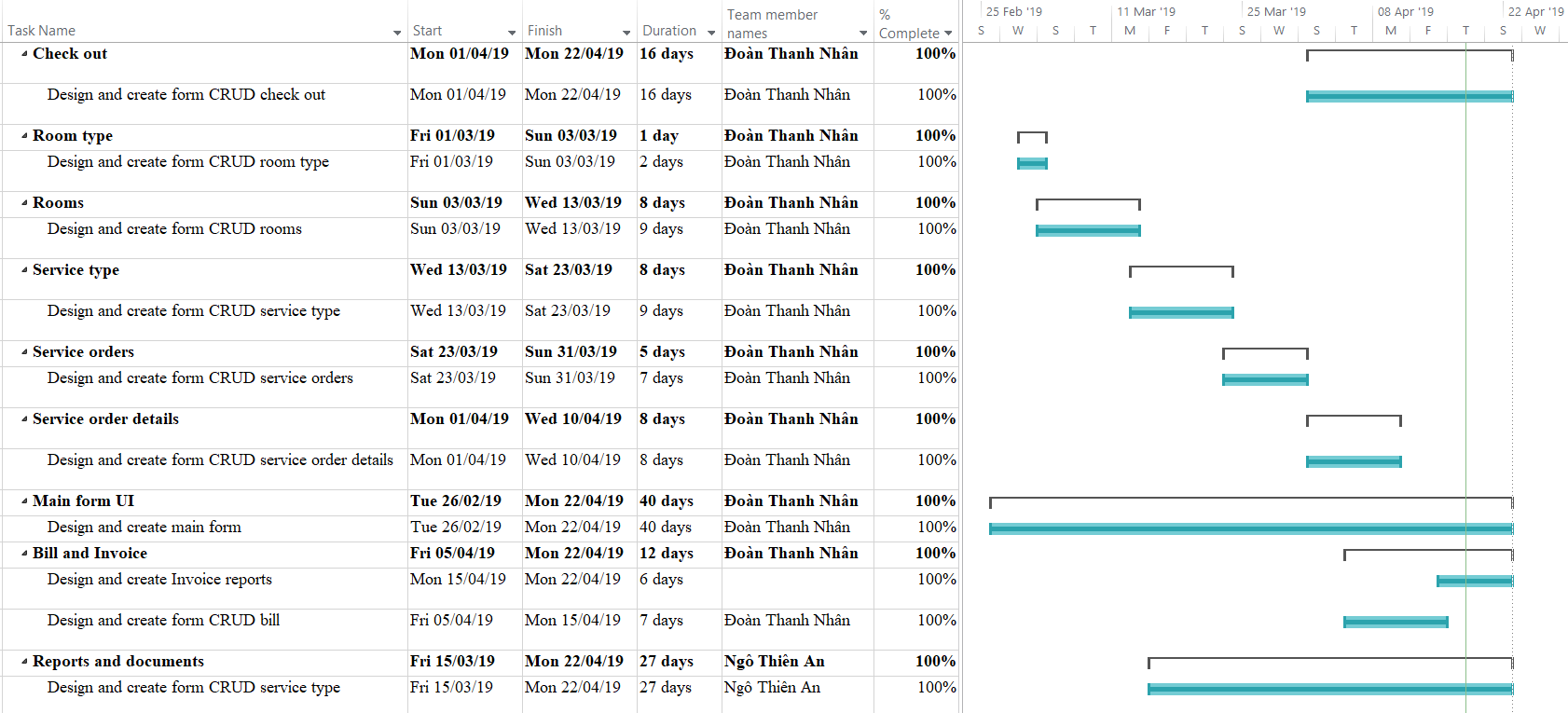
***Fig 3.1: User use case diagram***

# CHAPTER 4: PLAN AND TASK ASSIGNMENT

***Fig 4.1: Plan timeline***



***Fig 4.2: Plan and task assignment 1***



***Fig 4.2: Plan and task assignment 2***

# CHAPTER 5: DATABASE

## We use 1 database and 16 tables as below:

## 5.1. Bill table detail:

This table contain all information of the invoice.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | ID | int | 🞎 |  | PK, auto increase |
| 2 | RoomID | varchar(100) | 🞎 |  | FK, room ID |
| 3 | CustomerID | varchar(100) | 🞎 |  | FK, customer ID |
| 4 | UserName | varchar(100) | 🞎 |  | FK, user name |
| 5 | CheckInID | varchar(100) | 🞎 |  | FK, check in ID |
| 6 | CheckOutID | varchar(100) | 🞎 |  | FK, check out ID |
| 7 | CheckInDate | datetime | 🞎 |  | Check in date |
| 8 | CheckOutDate | datetime | 🞎 |  | Check out date |
| 9 | NoOfDay | int | 🞎 |  | Number of staying days |
| 10 | RoomPrice | decimal(18,3) | 🞎 |  | Price of room type |
| 11 | RoomCharge | decimal(18,3) | 🞎 |  | Room charge in total |
| 12 | ServiceCharge | decimal(18,3) | 🞎 |  | Services charge in total |
| 13 | RoomDiscount | decimal(18,3) | 🞎 |  | Room type discount |
| 14 | ServiceDiscount | decimal(18,3) | 🞎 |  | Service discount |
| 15 | CustomerDiscount | decimal(18,3) | 🞎 |  | Customer discount |
| 16 | TotalBillAmount | decimal(18,3) | 🞎 |  | Bill charge in total |
| 17 | VATAmount | decimal(18,3) | 🞎 |  | VAT charge in total |
| 18 | PayableAmount | decimal(18,3) | 🞎 |  | The money have to pay |
| 19 | CustomerGive | decimal(18,3) | 🞎 |  | Customer give money |
| 20 | CustomerChange | decimal(18,3) | 🞎 |  | Customer change |
| 21 | QRCode | varbinary(MAX) | 🗹 |  | QR code image |
| 22 | Active | bit | 🗹 |  | Active status of bill(default 1) |
| 23 | Note | nvarchar(1000) | 🗹 |  | Bill note |

## 5.2. BookingInfo table detail:

This table contain information of reservations.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | BookingID | varchar(100) | 🞎 |  | PK, booking ID |
| 2 | RoomID | varchar(100) | 🞎 |  | FK, room ID |
| 3 | CustomerID | varchar(100) | 🞎 |  | FK, customer ID |
| 4 | UserName | varchar(100) | 🞎 |  | FK, user name |
| 5 | Note | nvarchar(1000) | 🗹 |  | Booking note |
| 6 | NumberGuest | tinyint | 🞎 |  | Number of guests |
| 7 | DateBook | date | 🞎 |  | Booking date |
| 8 | DateLeave | datetime | 🗹 |  | Leaving date |
| 9 | CustomerName | nvarchar(100) | 🞎 |  | Customer’s full name |
| 10 | Active | bit | 🗹 |  | Booking active (default 1) |

## 5.3. CheckBlockMacAddress table detail:

This table contain information for blocking PC from using application.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | ID | int | 🞎 |  | PK, auto increase |
| 2 | MACaddress | varchar(30) | 🞎 |  | MAC address |
| 3 | Times | int | 🞎 |  | Number of login failed |
| 4 | Active | bit | 🞎 |  | Active status |

## 5.4. CheckInOrders table detail:

This table contain information of customer register checking in.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | CheckInID | varchar(100) | 🞎 |  | PK, check in ID |
| 2 | RoomID | varchar(100) | 🞎 |  | FK, room ID |
| 3 | CustomerID | varchar(100) | 🞎 |  | FK, customer ID |
| 4 | UserName | varchar(100) | 🞎 |  | FK, user name |
| 5 | BookingID | varchar(100) | 🞎 |  | FK, booking ID |
| 6 | CheckInType | varchar(20) | 🞎 |  | Check in type |
| 7 | NumberOfCustomer | tinyint | 🞎 |  | Number of customer |
| 8 | CheckInDate | datetime | 🞎 |  | Check in date |
| 9 | LeaveDate | datetime | 🞎 |  | Leaving date |
| 10 | CustomerPackage | nvarchar(200) | 🗹 |  | Customer packages |
| 11 | Active | bit | 🗹 |  | Active status (default 1) |

## 5.5. CheckOutOrders table detail:

This table contain information of customer register checking out.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | CheckOutID | varchar(100) | 🞎 |  | PK, check out ID |
| 2 | CheckInID | varchar(100) | 🞎 |  | FK, check in ID |
| 3 | RoomID | varchar(100) | 🞎 |  | FK, room ID |
| 4 | CustomerID | varchar(100) | 🞎 |  | FK, customer ID |
| 5 | UserName | varchar(100) | 🞎 |  | FK, user name |
| 6 | CheckInDate | datetime | 🞎 |  | Check in date |
| 7 | CheckOutDate | datetime | 🞎 |  | Check out date |
| 8 | CustomerPayment | nvarchar(50) | 🞎 |  | Customer payment |
| 9 | Discount | decimal(4,3) | 🗹 |  | Customer discount |
| 10 | Tax | decimal(6,3) | 🗹 |  | Tax charge |
| 11 | Active | bit | 🞎 |  | Active status (default 1) |

## 5.6. Customers table detail:

This table contain customer’s information.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | CustomerID | varchar(100) | 🞎 |  | PK, customer ID |
| 2 | UserName | varchar(100) | 🞎 |  | FK, user name |
| 3 | CustomerFirstName | nvarchar(20) | 🞎 |  | Customer’s first name |
| 4 | CustomerMidName | nvarchar(20) | 🗹 |  | Customer’s mid name |
| 5 | CustomerLastName | nvarchar(20) | 🞎 |  | Customer’s last name |
| 6 | CustomerBirthday | date | 🞎 |  | Customer’s birthday |
| 7 | CustomerPhoneNumber | varchar(20) | 🞎 |  | Customer’s P.Number |
| 8 | CustomerPassport | varchar(30) | 🗹 |  | Customer’s passport/ID |
| 9 | CustomerEmail | varchar(100) | 🗹 |  | Customer’s email |
| 10 | Discount | decimal(3,2) | 🗹 |  | Customer’s discount |
| 11 | Company | nvarchar(100) | 🗹 |  | Customer’s company |
| 12 | Active | bit | 🗹 |  | Active status (default 1) |
| 13 | Sex | bit | 🗹 |  | Customer gender |

## 5.7. Departments table detail:

This table contain department’s information and roles.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | DepartmentID | varchar(100) | 🞎 |  | PK, department ID |
| 2 | DepartmentName | nvarchar(100) | 🞎 |  | Department name |
| 3 | Active | bit | 🞎 |  | Active status (default 1) |
| 4 | Username | nvarchar(50) | 🞎 |  | User name |
| 5 | Employee\_View | bit | 🗹 |  | R employee table role |
| 6 | Employee\_Add | bit | 🗹 |  | C employee table role |
| 7 | Employee\_Edit | bit | 🗹 |  | U employee table role |
| 8 | Employee\_Delete | bit | 🗹 |  | D employee table role |
| 9 | User\_View | bit | 🗹 |  | R user table role |
| 10 | User\_Add | bit | 🗹 |  | C user table role |
| 11 | User\_Edit | bit | 🗹 |  | U user table role |
| 12 | User\_Delete | bit | 🗹 |  | D user table role |
| 13 | Booking\_View | bit | 🗹 |  | R Booking table role |
| 14 | Booking\_Add | bit | 🗹 |  | C Booking table role |
| 15 | Booking\_Edit | bit | 🗹 |  | U Booking table role |
| 16 | Booking\_Delete | bit | 🗹 |  | D Booking table role |
| 17 | CheckIn\_View | bit | 🗹 |  | R CheckIn table role |
| 18 | CheckIn\_Add | bit | 🗹 |  | C CheckIn table role |
| 19 | CheckIn\_Edit | bit | 🗹 |  | U CheckIn table role |
| 20 | CheckIn\_Delete | bit | 🗹 |  | D CheckIn table role |
| 21 | CheckOut\_View | bit | 🗹 |  | R CheckOut table role |
| 22 | CheckOut\_Add | bit | 🗹 |  | C CheckOut table role |
| 23 | CheckOut\_Edit | bit | 🗹 |  | U CheckOut table role |
| 24 | CheckOut\_Delete | bit | 🗹 |  | D CheckOut table role |
| 25 | Customer\_View | bit | 🗹 |  | R Customer table role |
| 26 | Customer\_Add | bit | 🗹 |  | C Customer table role |
| 27 | Customer\_Edit | bit | 🗹 |  | U Customer table role |
| 28 | Customer\_Delete | bit | 🗹 |  | D Customer table role |
| 29 | Department\_View | bit | 🗹 |  | R Department table role |
| 30 | Department\_Add | bit | 🗹 |  | C Department table role |
| 31 | Department\_Edit | bit | 🗹 |  | U Department table role |
| 32 | Department\_Delete | bit | 🗹 |  | D Department table role |
| 33 | Role\_View | bit | 🗹 |  | R Role table role |
| 34 | Role\_Add | bit | 🗹 |  | C Role table role |
| 35 | Role\_Edit | bit | 🗹 |  | U Role table role |
| 36 | Role\_Delete | bit | 🗹 |  | D Role table role |
| 37 | Room\_View | bit | 🗹 |  | R Room table role |
| 38 | Room\_Add | bit | 🗹 |  | C Room table role |
| 39 | Room\_Edit | bit | 🗹 |  | U Room table role |
| 40 | Room\_Delete | bit | 🗹 |  | D Room table role |
| 41 | SODetail\_View | bit | 🗹 |  | R SODetail table role |
| 42 | SODetail\_Add | bit | 🗹 |  | C SODetail table role |
| 43 | SODetail\_Edit | bit | 🗹 |  | U SODetail table role |
| 44 | SODetail\_Delete | bit | 🗹 |  | D SODetail table role |
| 45 | SOder\_View | bit | 🗹 |  | R SOder table role |
| 46 | SOder\_Add | bit | 🗹 |  | C SOder table role |
| 47 | SOder\_Edit | bit | 🗹 |  | U SOder table role |
| 48 | SOder\_Delete | bit | 🗹 |  | D SOder table role |
| 49 | SType\_View | bit | 🗹 |  | R SType table role |
| 50 | SType\_Add | bit | 🗹 |  | C SType table role |
| 51 | SType\_Edit | bit | 🗹 |  | U SType table role |
| 52 | SType\_Delete | bit | 🗹 |  | D SType table role |
| 53 | UserLog\_View | bit | 🗹 |  | R UserLog table role |
| 54 | UserLog\_Add | bit | 🗹 |  | C UserLog table role |
| 55 | UserLog\_Edit | bit | 🗹 |  | U UserLog table role |
| 56 | UserLog\_Delete | bit | 🗹 |  | D UserLog table role |
| 57 | ReActive\_View | bit | 🗹 |  | R ReActive table role |
| 58 | ReActive\_Add | bit | 🗹 |  | C ReActive table role |
| 59 | ReActive\_Edit | bit | 🗹 |  | U ReActive table role |
| 60 | ReActive\_Delete | bit | 🗹 |  | D ReActive table role |

## 5.8. Employees table detail:

This table contain employee’s information.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | EmployeeID | varchar(100) | 🞎 |  | PK, Employee ID |
| 2 | DepartmentID | varchar(100) | 🞎 |  | FK, department ID |
| 3 | EmployeeFirstName | nvarchar(15) | 🞎 |  | Employee’s first name |
| 4 | EmployeeMidName | nvarchar(25) | 🗹 |  | Employee’s mid name |
| 5 | EmployeeLastName | nvarchar(15) | 🞎 |  | Employee’s last name |
| 6 | Birthday | date | 🗹 |  | Employee’s birthday |
| 7 | PhoneNumber | varchar(20) | 🗹 |  | Employee’s P.Number |
| 8 | Address | nvarchar(100) | 🗹 |  | Employee’s address |
| 9 | IDNumber | varchar(20) | 🗹 |  | Employee’s ID number |
| 10 | HireDate |  |  |  |  |
| 11 | Job |  |  |  |  |
| 12 | EducatedLevel |  |  |  |  |
| 13 | Sex | bit | 🞎 |  | Employee gender |
| 14 | Salary | decimal(18,0) | 🗹 |  | Employee’s salary |
| 15 | Bonus | decimal(18,0) | 🗹 |  | Employee’s bonus |
| 16 | Comm | decimal(18,0) | 🗹 |  | Employee’s commission |
| 17 | Email | varchar(100) | 🗹 |  | Employee’s email |
| 18 | Active | bit | 🗹 |  | Active status (default 1) |
| 19 | Image | varbinary(max) | 🗹 |  | Employee’s image |
| 20 | ReActive | bit | 🗹 |  | ReActive status |

## 5.9. Role table detail:

This table contain role’s information.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | RoleID | int | 🞎 |  | PK, auto increase |
| 2 | EmployeeID | varchar(100) | 🞎 |  | FK, employee’s ID |
| 3 | Employee\_View | bit | 🗹 |  | R employee table role |
| 4 | Employee\_Add | bit | 🗹 |  | C employee table role |
| 5 | Employee\_Edit | bit | 🗹 |  | U employee table role |
| 6 | Employee\_Delete | bit | 🗹 |  | D employee table role |
| 7 | User\_View | bit | 🗹 |  | R user table role |
| 8 | User\_Add | bit | 🗹 |  | C user table role |
| 9 | User\_Edit | bit | 🗹 |  | U user table role |
| 10 | User\_Delete | bit | 🗹 |  | D user table role |
| 11 | Booking\_View | bit | 🗹 |  | R Booking table role |
| 12 | Booking\_Add | bit | 🗹 |  | C Booking table role |
| 13 | Booking\_Edit | bit | 🗹 |  | U Booking table role |
| 14 | Booking\_Delete | bit | 🗹 |  | D Booking table role |
| 15 | CheckIn\_View | bit | 🗹 |  | R CheckIn table role |
| 16 | CheckIn\_Add | bit | 🗹 |  | C CheckIn table role |
| 17 | CheckIn\_Edit | bit | 🗹 |  | U CheckIn table role |
| 18 | CheckIn\_Delete | bit | 🗹 |  | D CheckIn table role |
| 19 | CheckOut\_View | bit | 🗹 |  | R CheckOut table role |
| 20 | CheckOut\_Add | bit | 🗹 |  | C CheckOut table role |
| 21 | CheckOut\_Edit | bit | 🗹 |  | U CheckOut table role |
| 22 | CheckOut\_Delete | bit | 🗹 |  | D CheckOut table role |
| 23 | Customer\_View | bit | 🗹 |  | R Customer table role |
| 24 | Customer\_Add | bit | 🗹 |  | C Customer table role |
| 25 | Customer\_Edit | bit | 🗹 |  | U Customer table role |
| 26 | Customer\_Delete | bit | 🗹 |  | D Customer table role |
| 27 | Department\_View | bit | 🗹 |  | R Department table role |
| 28 | Department\_Add | bit | 🗹 |  | C Department table role |
| 29 | Department\_Edit | bit | 🗹 |  | U Department table role |
| 30 | Department\_Delete | bit | 🗹 |  | D Department table role |
| 31 | Role\_View | bit | 🗹 |  | R Role table role |
| 32 | Role\_Add | bit | 🗹 |  | C Role table role |
| 33 | Role\_Edit | bit | 🗹 |  | U Role table role |
| 34 | Role\_Delete | bit | 🗹 |  | D Role table role |
| 35 | Room\_View | bit | 🗹 |  | R Room table role |
| 36 | Room\_Add | bit | 🗹 |  | C Room table role |
| 37 | Room\_Edit | bit | 🗹 |  | U Room table role |
| 38 | Room\_Delete | bit | 🗹 |  | D Room table role |
| 39 | SODetail\_View | bit | 🗹 |  | R SODetail table role |
| 40 | SODetail\_Add | bit | 🗹 |  | C SODetail table role |
| 41 | SODetail\_Edit | bit | 🗹 |  | U SODetail table role |
| 42 | SODetail\_Delete | bit | 🗹 |  | D SODetail table role |
| 43 | SOder\_View | bit | 🗹 |  | R SOder table role |
| 44 | SOder\_Add | bit | 🗹 |  | C SOder table role |
| 45 | SOder\_Edit | bit | 🗹 |  | U SOder table role |
| 46 | SOder\_Delete | bit | 🗹 |  | D SOder table role |
| 47 | SType\_View | bit | 🗹 |  | R SType table role |
| 48 | SType\_Add | bit | 🗹 |  | C SType table role |
| 49 | SType\_Edit | bit | 🗹 |  | U SType table role |
| 50 | SType\_Delete | bit | 🗹 |  | D SType table role |
| 51 | UserLog\_View | bit | 🗹 |  | R UserLog table role |
| 52 | UserLog\_Add | bit | 🗹 |  | C UserLog table role |
| 53 | UserLog\_Edit | bit | 🗹 |  | U UserLog table role |
| 54 | UserLog\_Delete | bit | 🗹 |  | D UserLog table role |
| 55 | ReActive\_View | bit | 🗹 |  | R ReActive table role |
| 56 | ReActive\_Add | bit | 🗹 |  | C ReActive table role |
| 57 | ReActive\_Edit | bit | 🗹 |  | U ReActive table role |
| 58 | ReActive\_Delete | bit | 🗹 |  | D ReActive table role |

## 5.10. Rooms table detail:

This table contain room’s information.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | RoomID | varchar(100) | 🞎 |  | PK, room ID |
| 2 | CustomerID | varchar(100) | 🞎 |  | FK, customer ID |
| 3 | UserName | varchar(100) | 🞎 |  | FK, user name |
| 4 | RoomType | varchar(100) | 🞎 |  | FK, room type |
| 5 | PhoneNumber | varchar(20) | 🞎 |  | Room’s phone number |
| 6 | RoomOnFloor | tinyint | 🞎 |  | The no of floor |
| 7 | RoomArea | decimal(8,3) | 🞎 |  | Room’s area (m2) |
| 8 | RoomStatus | varchar(10) | 🞎 |  | Room status |
| 9 | DayRemaining | int | 🗹 |  | Day remaining |
| 10 | NoOfBooking | int | 🗹 |  | Number of reservations |
| 11 | NoOfGuests | int | 🗹 |  | Number of staying guests |
| 12 | BookingDate | datetime | 🗹 |  | Nearest booking date (from now) |
| 13 | CheckInDate | datetime | 🗹 |  | Check in date |
| 14 | LeaveDate | datetime | 🗹 |  | Nearest leave date (from now to the past) |
| 15 | Clean | bit | 🗹 |  | Clean status of bill(default 1) |
| 16 | Repaired | bit | 🗹 |  | Repairted status of bill(default 1) |
| 17 | InProgress | bit | 🗹 |  | Checking status of bill(default 0) |
| 18 | Active | bit | 🗹 |  | Active status of bill(default 1) |

## 5.11. RoomType table detail:

This table contain room type information.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | RoomType | varchar(100) | 🞎 |  | PK, room type |
| 2 | Price | decimal(18,3) | 🞎 |  | Room type price |
| 3 | Discount | decimal(3,2) | 🗹 |  | Room type discount |
| 4 | Active | bit | 🗹 |  | Active status |

## 5.12. ServicesOrderDetails table detail:

This table contain services order in detail.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | ID | int | 🞎 |  | PK, auto increase |
| 2 | OrderID | varchar(100) | 🞎 |  | FK, service order ID |
| 3 | ServiceID | varchar(100) | 🞎 |  | FK, service type ID |
| 4 | UserName | varchar(100) | 🞎 |  | FK, user name |
| 5 | ServiceQuantity | int | 🞎 |  | Service order quantity |
| 6 | Price | decimal(18,3) | 🞎 |  | Service type price |
| 7 | Discount | float | 🗹 |  | Service type discount |
| 8 | Finish | bit | 🗹 |  | SOD finished |
| 9 | CheckOut | bit | 🗹 |  | SOD checked out |
| 10 | Active | bit | 🞎 |  | Active status (default 1) |

## 5.13. ServicesOrders table detail:

This table contain service order information.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | OrderID | varchar(100) | 🞎 |  | PK, service order ID |
| 2 | CustomerID | varchar(100) | 🞎 |  | FK, customer ID |
| 3 | RoomID | varchar(100) | 🞎 |  | FK, room ID |
| 4 | UserName | varchar(100) | 🞎 |  | FK, user name |
| 5 | ServiceOrderDate | datetime | 🞎 |  | Service order date and time |
| 6 | ServiceNote | nvarchar(200) | 🗹 |  | Service order note |
| 7 | Finish | bit | 🗹 |  | SO finished |
| 8 | CheckOut | bit | 🗹 |  | SO checked out |
| 9 | Active | bit | 🞎 |  | Active status (default 1) |

## 5.14. ServiceType table detail:

This table contain service type information.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | ServiceID | varchar(100) | 🞎 |  | PK, service type ID |
| 2 | UserName | varchar(100) | 🞎 |  | FK, user name |
| 3 | ServiceName | nvarchar(100) | 🞎 |  | Service name |
| 4 | ServiceUnit | nvarchar(20) | 🞎 |  | Service unit |
| 5 | ServicePrice | decimal(18,3) | 🞎 |  | Service price |
| 6 | ServiceInventory | int | 🞎 |  | Service inventory |
| 7 | ImportQuantity | int | 🗹 |  | Service import quantity |
| 8 | ImportDate | datetime | 🗹 |  | Nearest Service import date |
| 9 | ExportQuantity | int | 🗹 |  | Service export quantity |
| 10 | ExportDate | datetime | 🗹 |  | Nearest Service export date |
| 11 | ServiceDescription | nvarchar(500) | 🗹 |  | Service description |
| 12 | Image | varbinary(max) | 🗹 |  | Service’s image |
| 13 | Active | bit | 🞎 |  | Active status (default 1) |

## 5.15. UserLogs table detail:

This table contain user log information.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | ID | int | 🞎 |  | PK, auto increase |
| 2 | UserName | varchar(100) | 🞎 |  | FK, user name |
| 3 | MACAddress | varchar(20) | 🗹 |  | MAC address |
| 4 | LogContent | nvarchar(1000) | 🞎 |  | Log content |
| 5 | LogTime | datetime | 🞎 |  | Log time |
| 6 | Active | bit | 🞎 |  | Active status (default 1) |

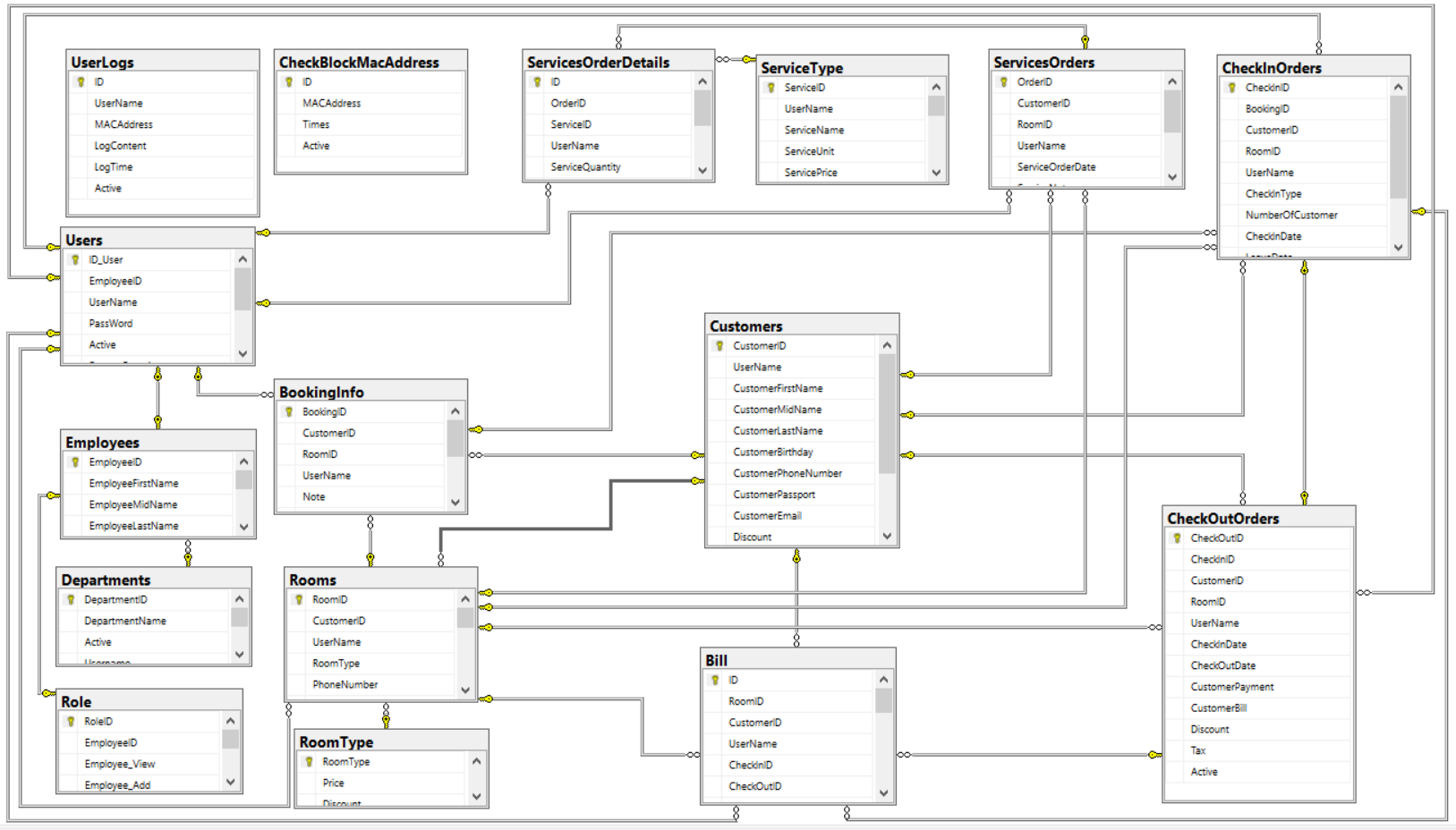
## 5.16. Users table detail:

This table contain user information.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Column name** | **Data type** | **Allow null** | **Key** | **Description** |
| 1 | ID | int | 🞎 |  | PK, auto increase |
| 2 | EmployeeID | varchar(100) | 🞎 |  | FK, user name |
| 3 | UserName | varchar(100) | 🗹 |  | User name |
| 4 | PassWord | varchar(255) | 🗹 |  | Password |
| 5 | Secret\_Question | nvarchar(100) | 🗹 |  | Secret question |
| 6 | Secret\_Answer | nvarchar(100) | 🗹 |  | Secret answer |
| 7 | Check\_Login | int | 🗹 |  | Check login times |
| 8 | Check\_Time | date | 🗹 |  | Check login date |
| 9 | Active | bit | 🗹 |  | Active status (default 1) |

## 5.17. Database ERD (Entity Relationship Diagram):

This table contain user information.

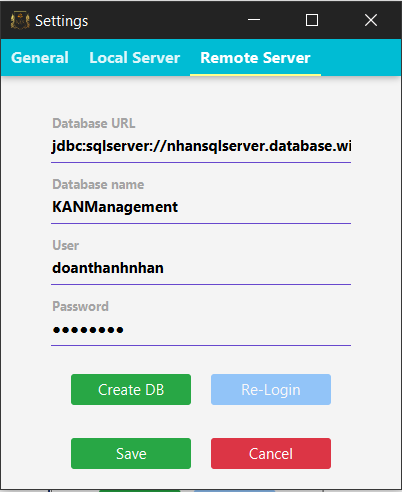
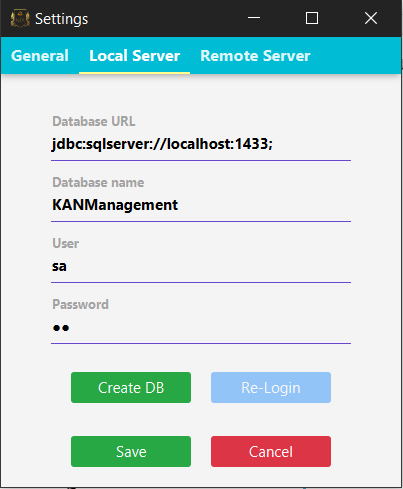
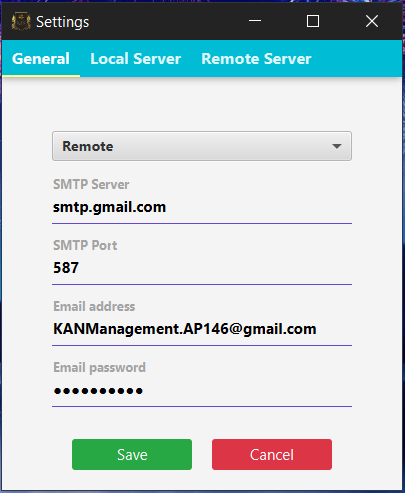


***Fig 5.17: Database ERD***

# CHAPTER 6: THE BASIC UI OF APPLICATION

## 6.1. Settings form:

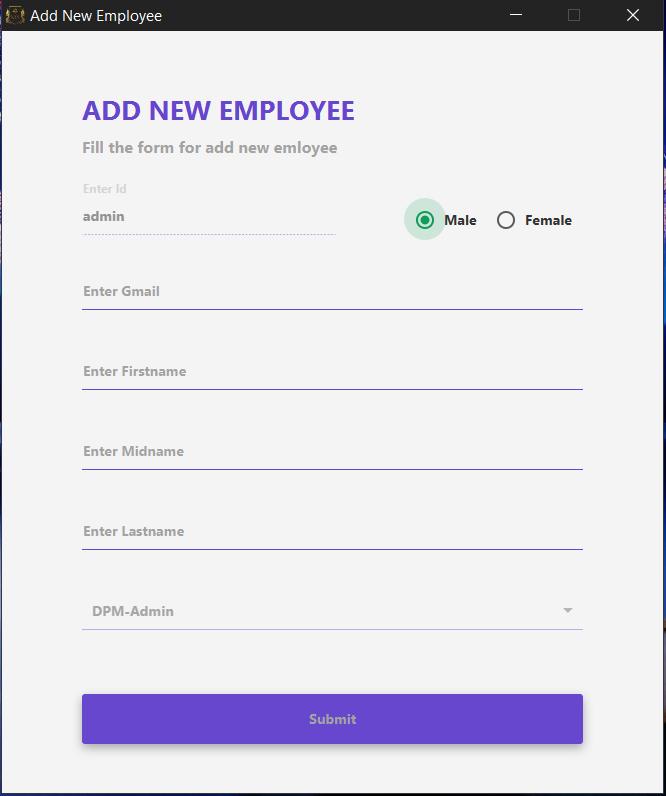
This form allow user input email, local server and remote server information. At the 1st time application run (database may be not configured) this form will appear and user must input information to create database and tables. (Click **Create DB** button)



***Figure 6.1: Settings form***

## 6.2. Add new Employee form:

If every things in 1st step are correct, the “**Add new Employee form**” will appear to create “**admin**” account, and this account don’t allow to change the ID like below figure:

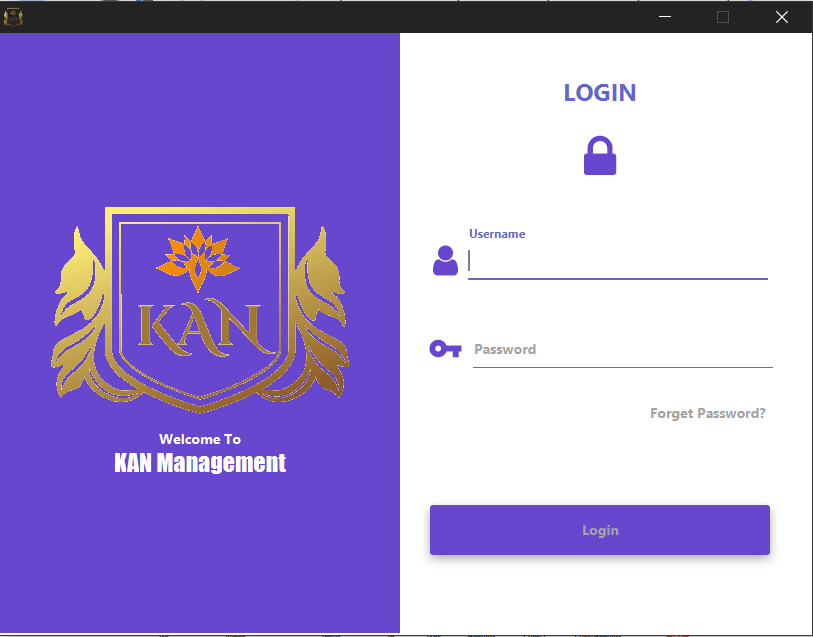


***Figure 6.2: Add new employee form***

After create “admin” account successful, the application will send user name and random password to admin email.

## 6.3. Login form:

After getting user and password you can login now via this form:

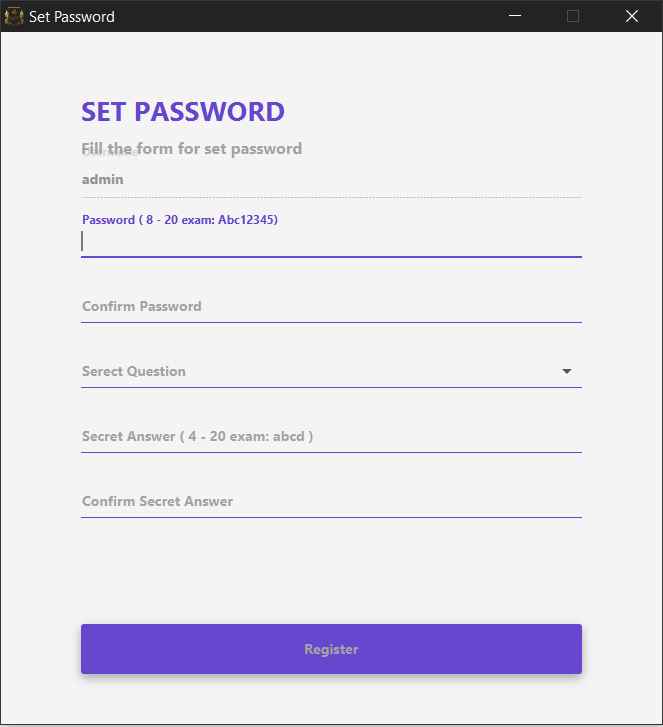


***Figure 6.3: Login form***

## 6.4. Checkout page screenshot:

In the 1st login of this account, “Set password form” will appear and allow user change their password for security purpose. And in this form you can choose secret question in case forgotten password in future.

All password, secret questions and answers are encrypted by MD5 **(Message-Digest algorithm 5).**

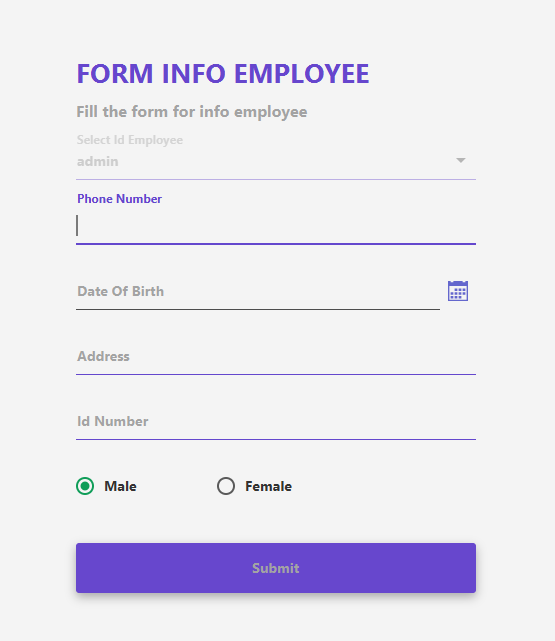


***Figure 6.4: Set password form***

After finishing input information (click Register button), the login form will appear again for login purpose. And user can use their account with their new password.

## 6.5. Employee’s information form:

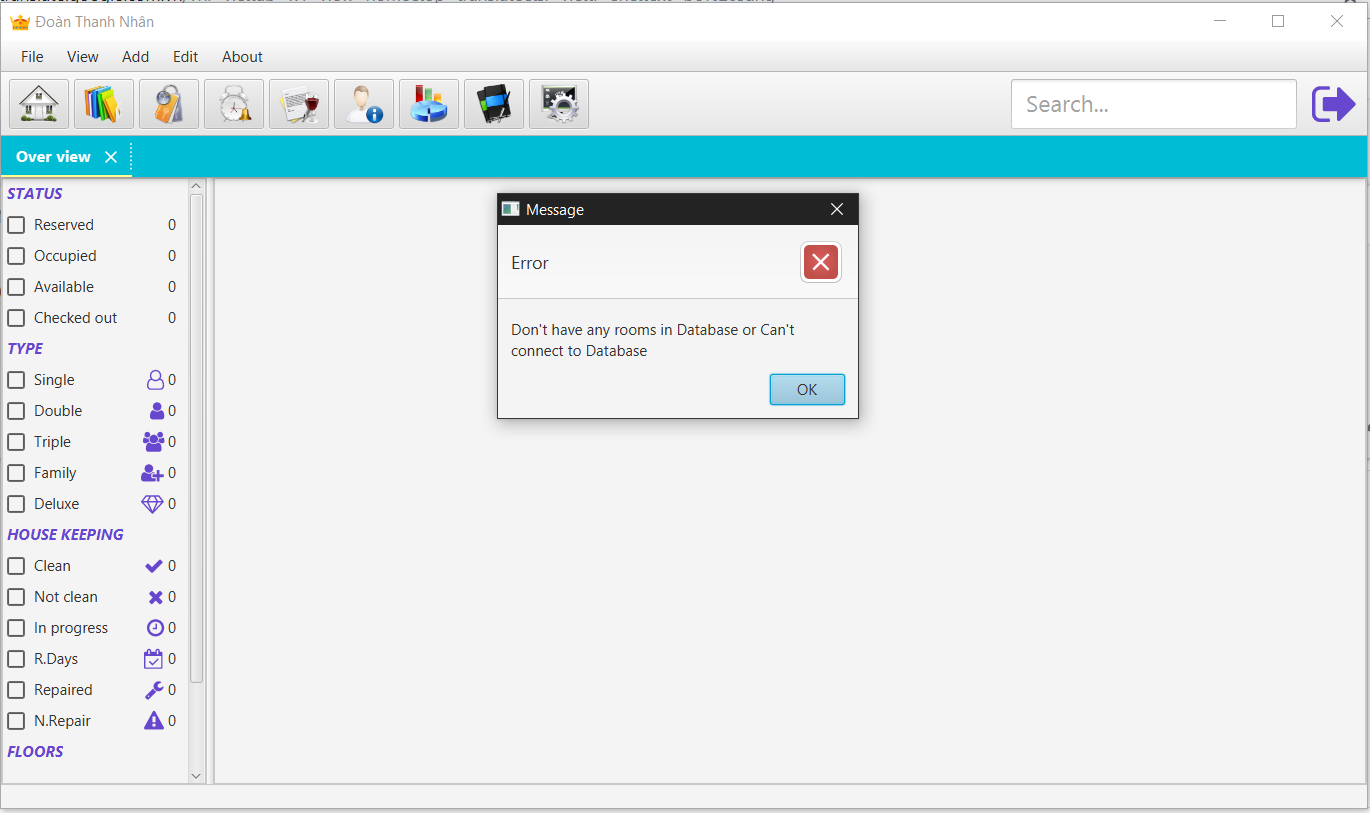
After login successfully, ***in the 1st time user must input additional information***, this form don’t allow to close (doesn’t have close button in the title bar), and if user try to close this form or another action try to close this form, in the next login this form will display again.



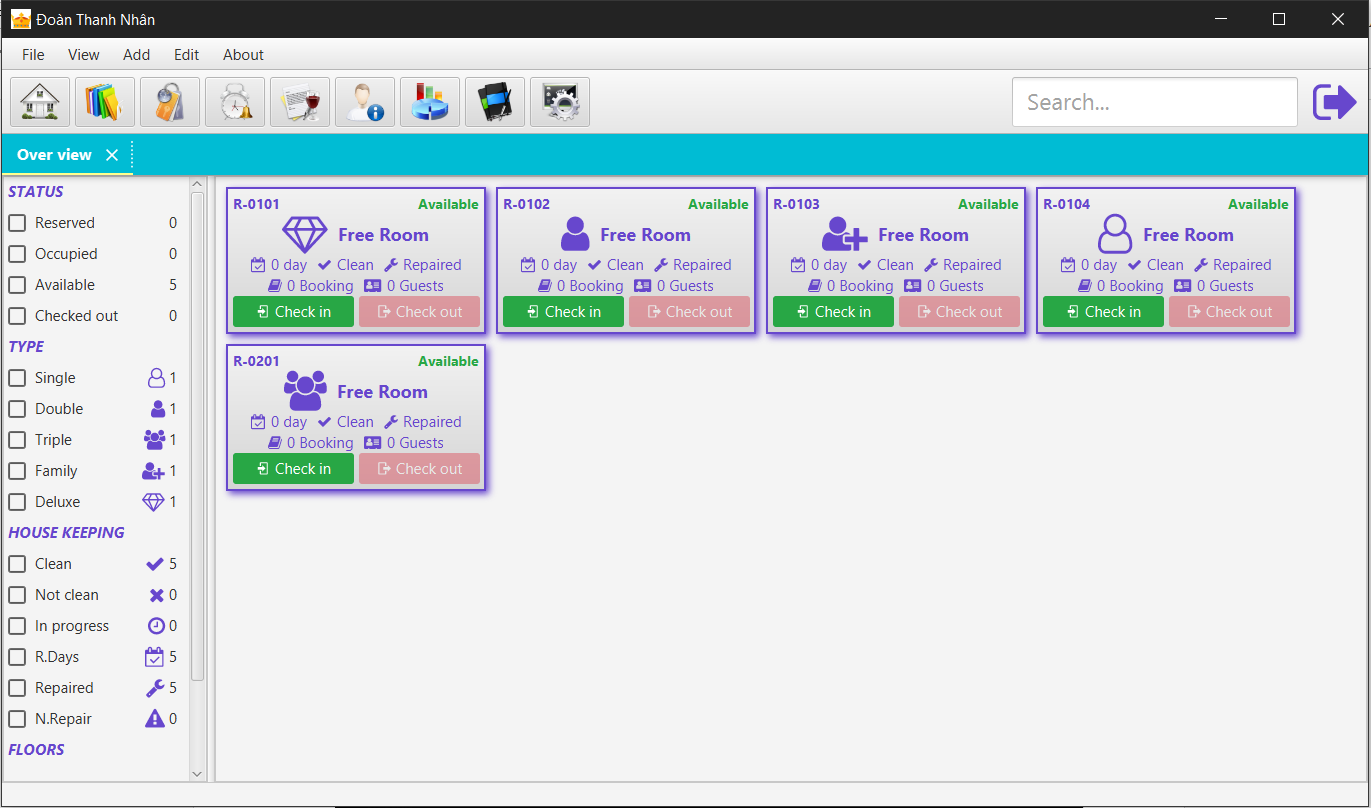
***Figure 6.5: Employee’s information form***

## 6.6. Main form:

After login successfully, “Main form” will appear and in the 1st login, database doesn’t have any data so the main form will display like this:



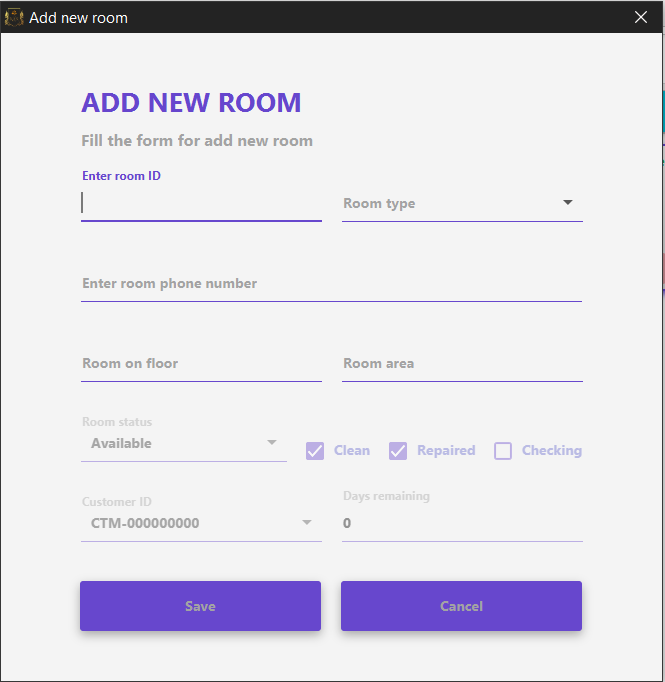
***Figure 6.6.a: Main form in the 1st login***



***Figure 6.6.b: Main form***

## 6.7. Add new room form:

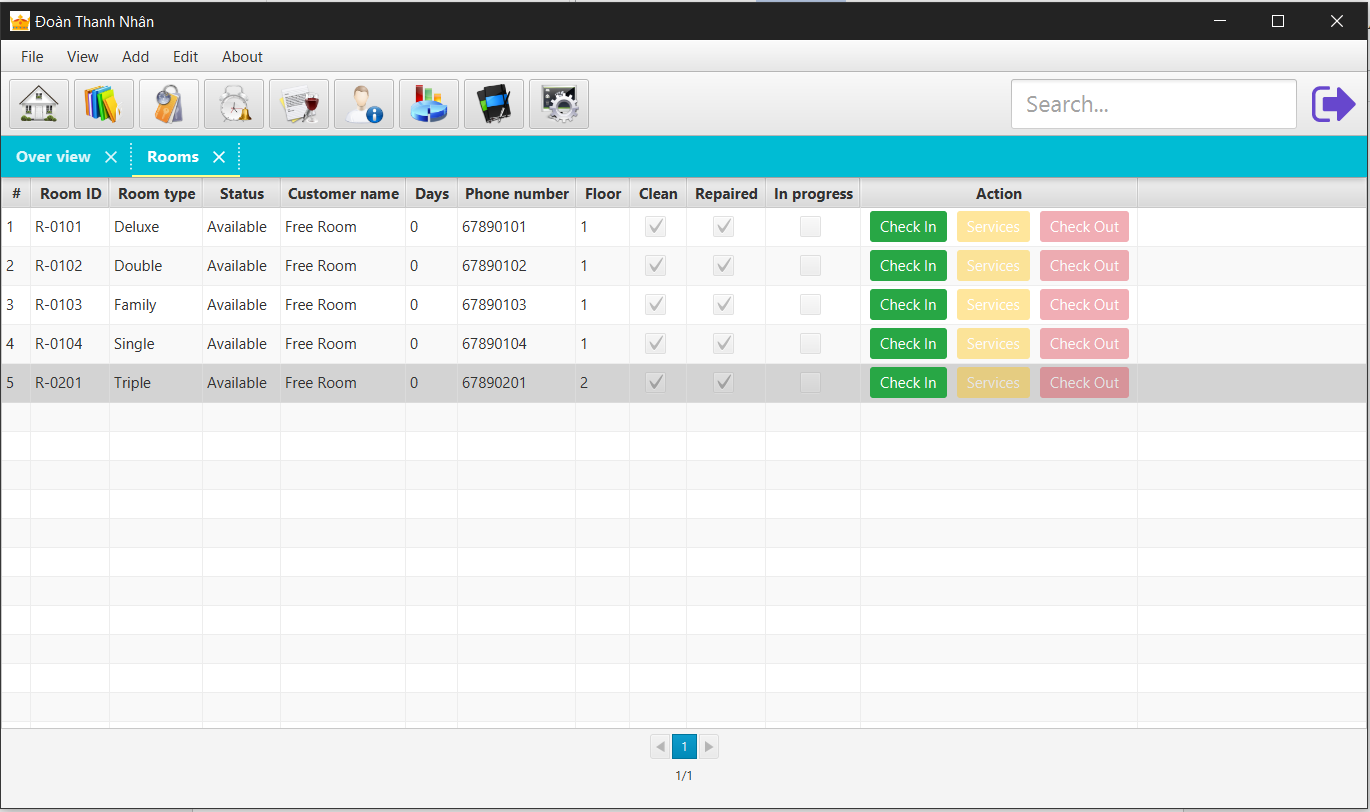
Now users can do anything up to their’s role. Next step will add new room via below form:



***Figure 6.7: Add new room form***

## 6.8. List room form:

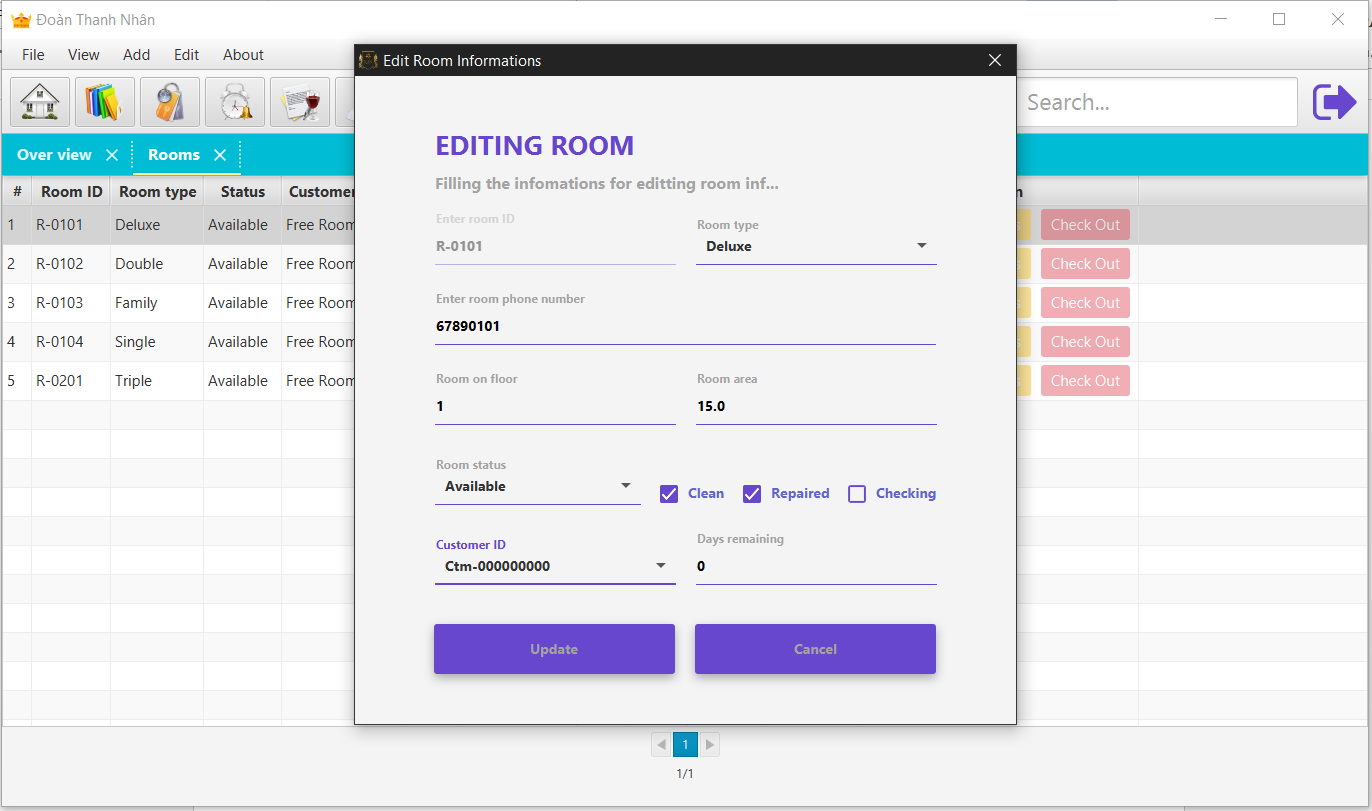
This form display all room.



***Figure 6.8: List room form***

## 6.9. Edit room:

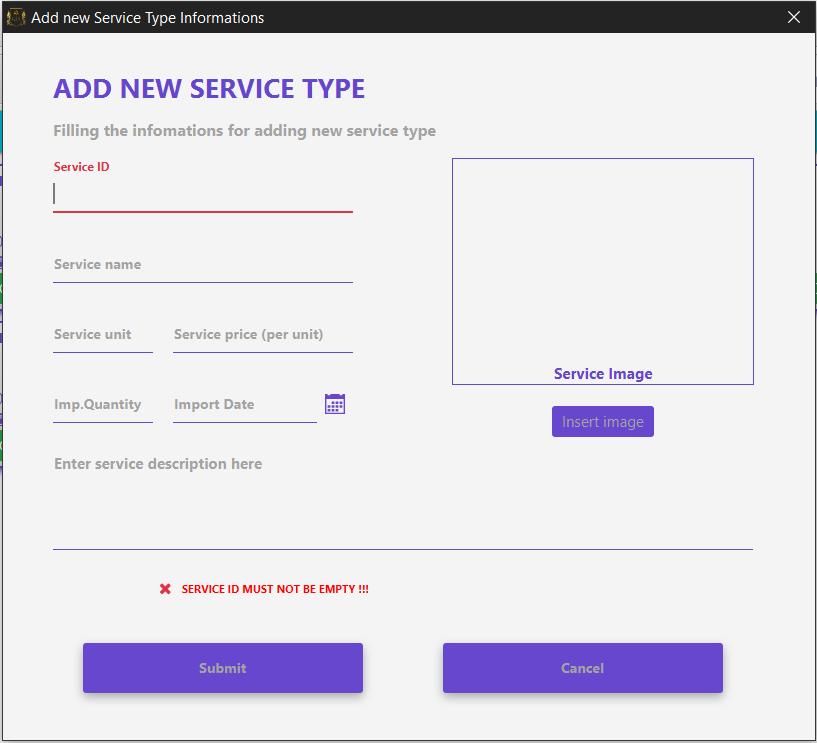
This form allow edditing room’s information:



***Figure 6.9: Edit room information form***

## 6.10. Add new service form:

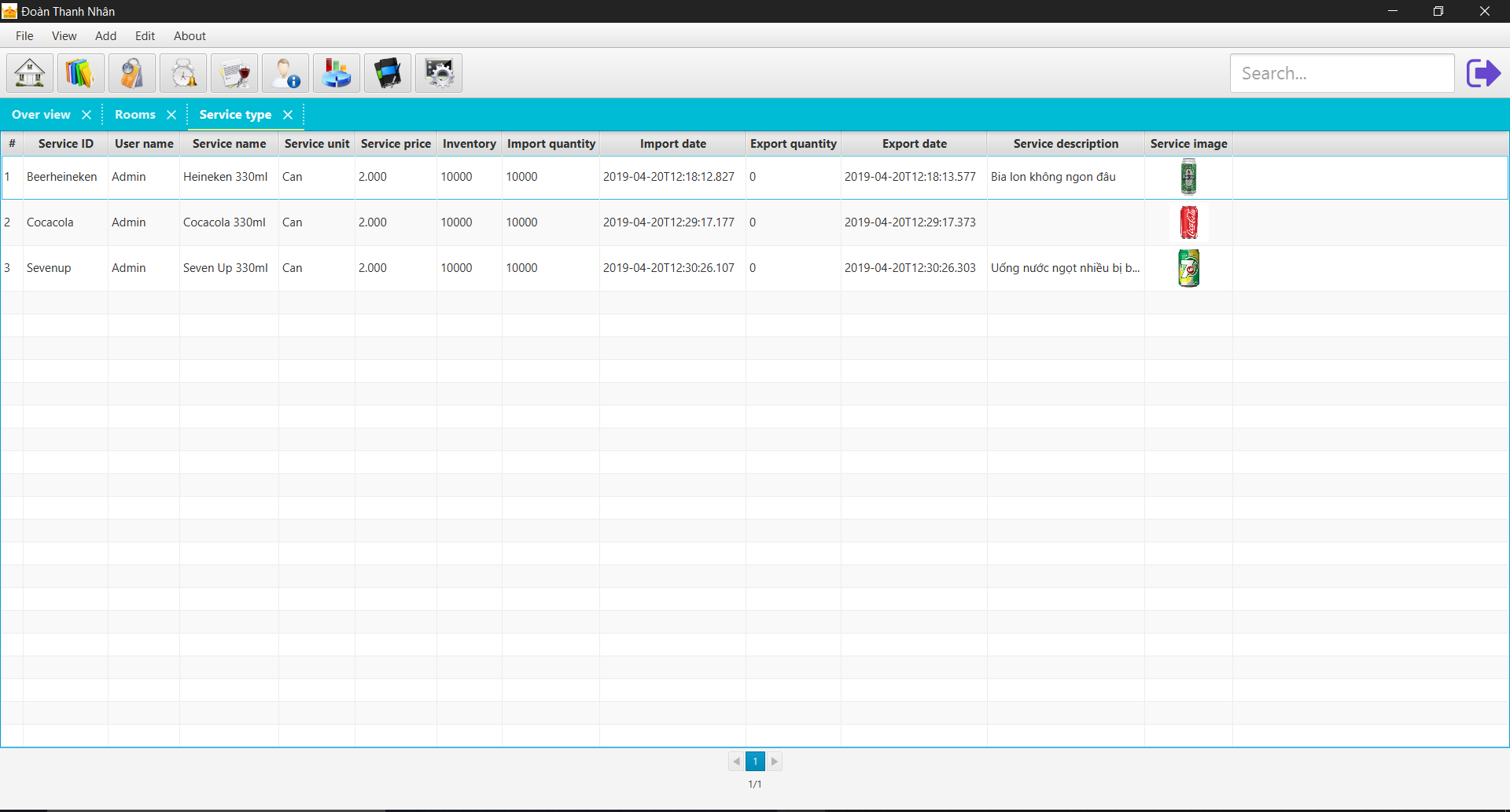
Next step will add new service for the hotel via below form:



***Figure 6.8: Add new service type***

## 6.11. List services form:

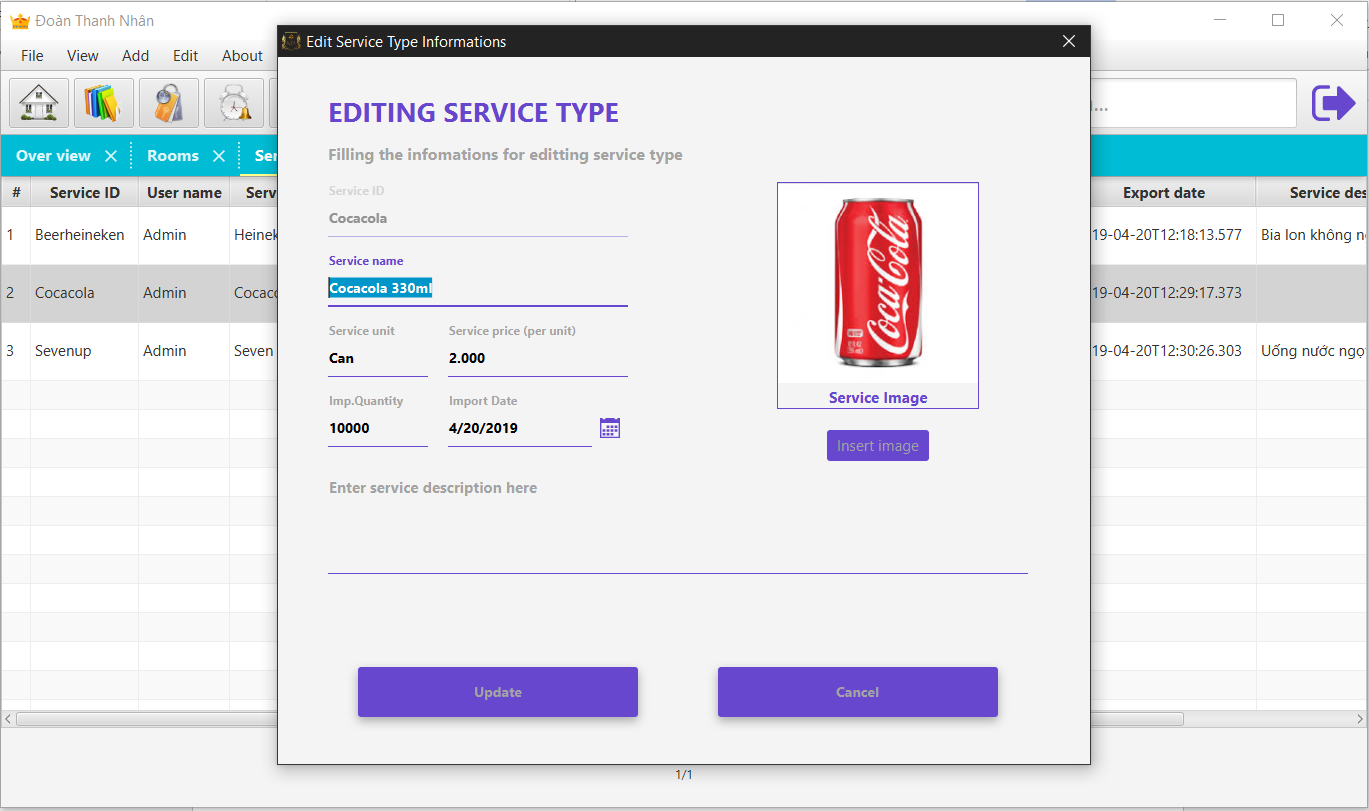
This form display all service in the hotel.



***Figure 6.11: List services***

## 6.12. Edit services form:

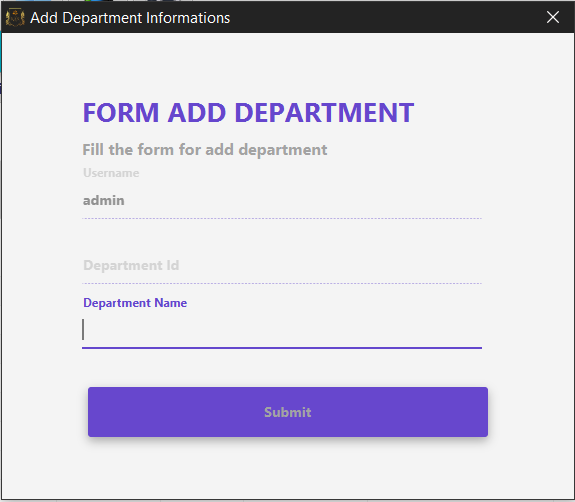
This form allow edditing service’s information:



***Figure 6.12: Edit service type form***

## 6.13. Add department form:

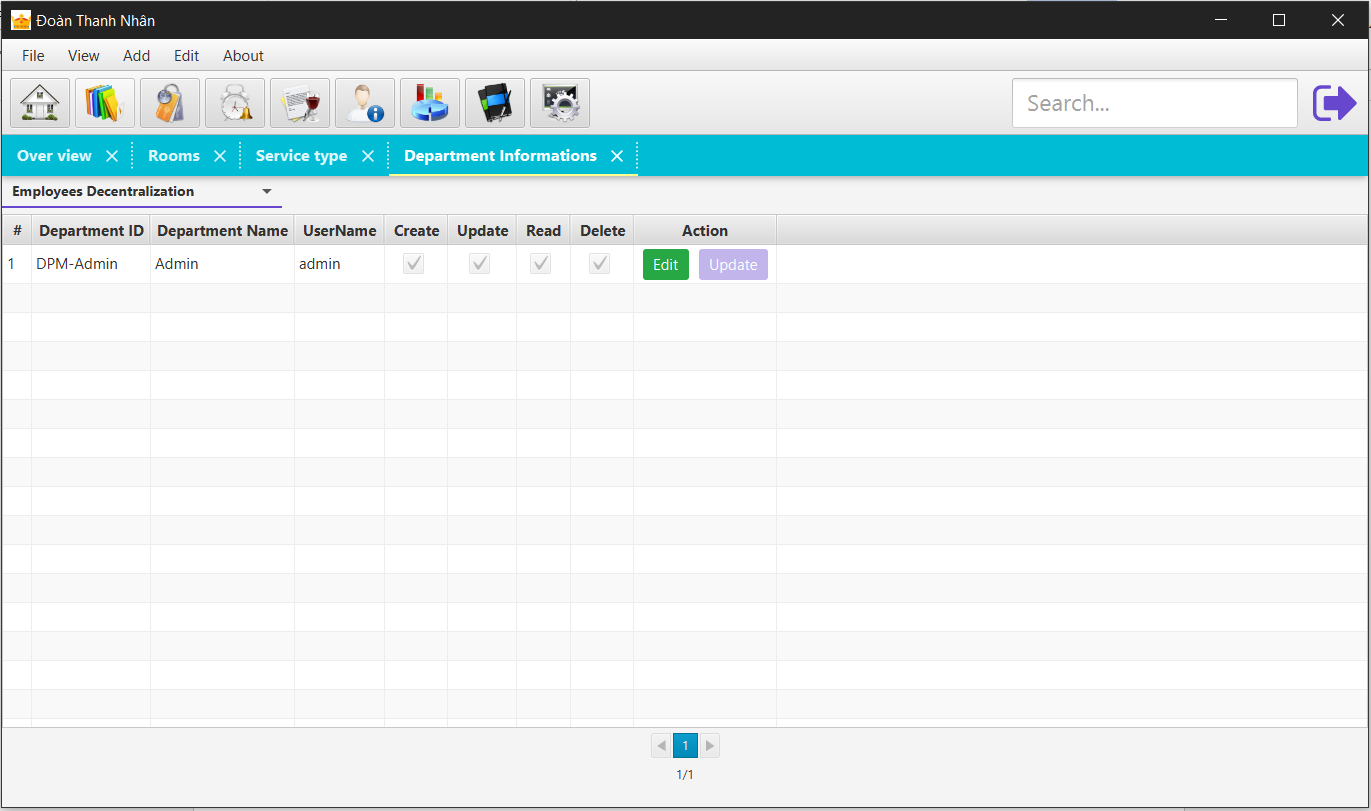
This form allow adding department’s information:



***Figure 6.13: Add department form***

## 6.14. List department form:

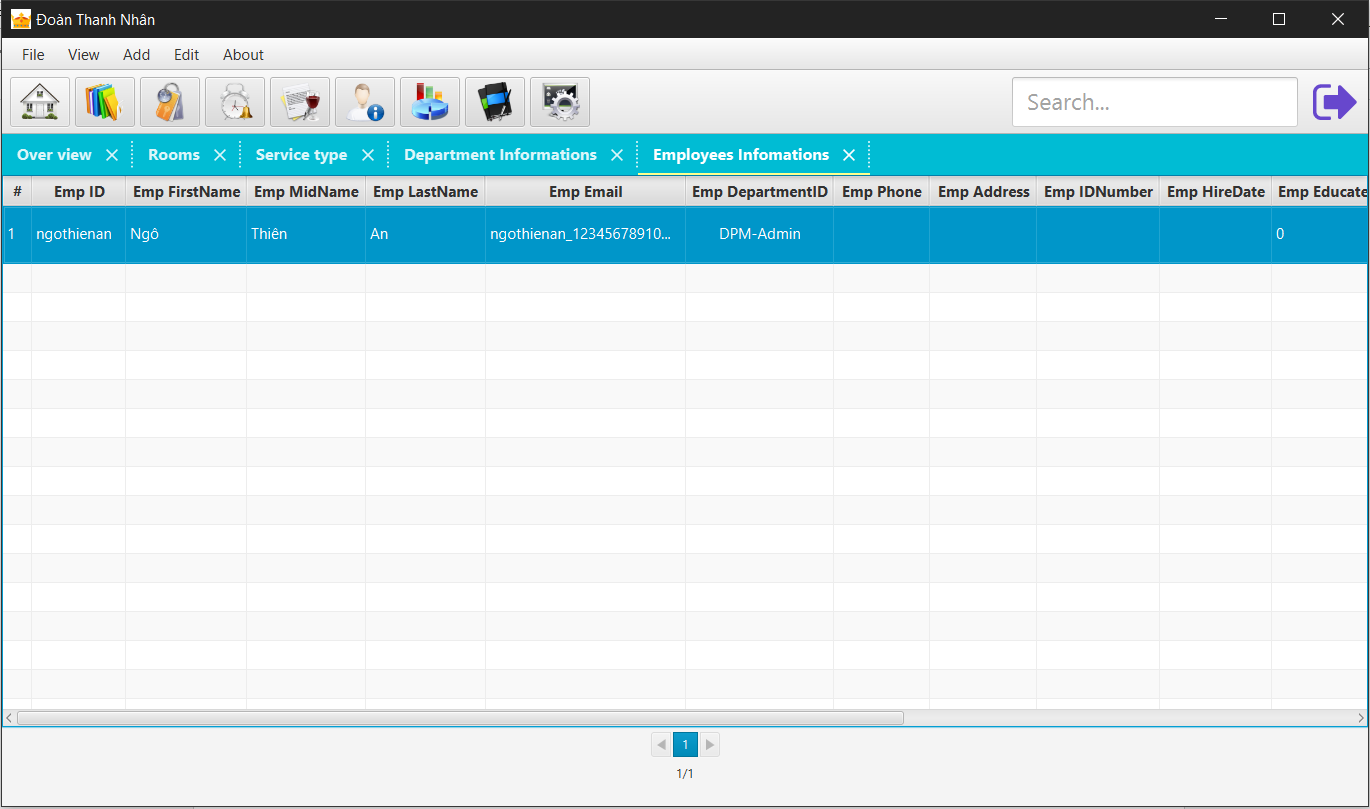
This form display all department’s information:



***Figure 6.14: List department form***

## 6.15. List employees form:

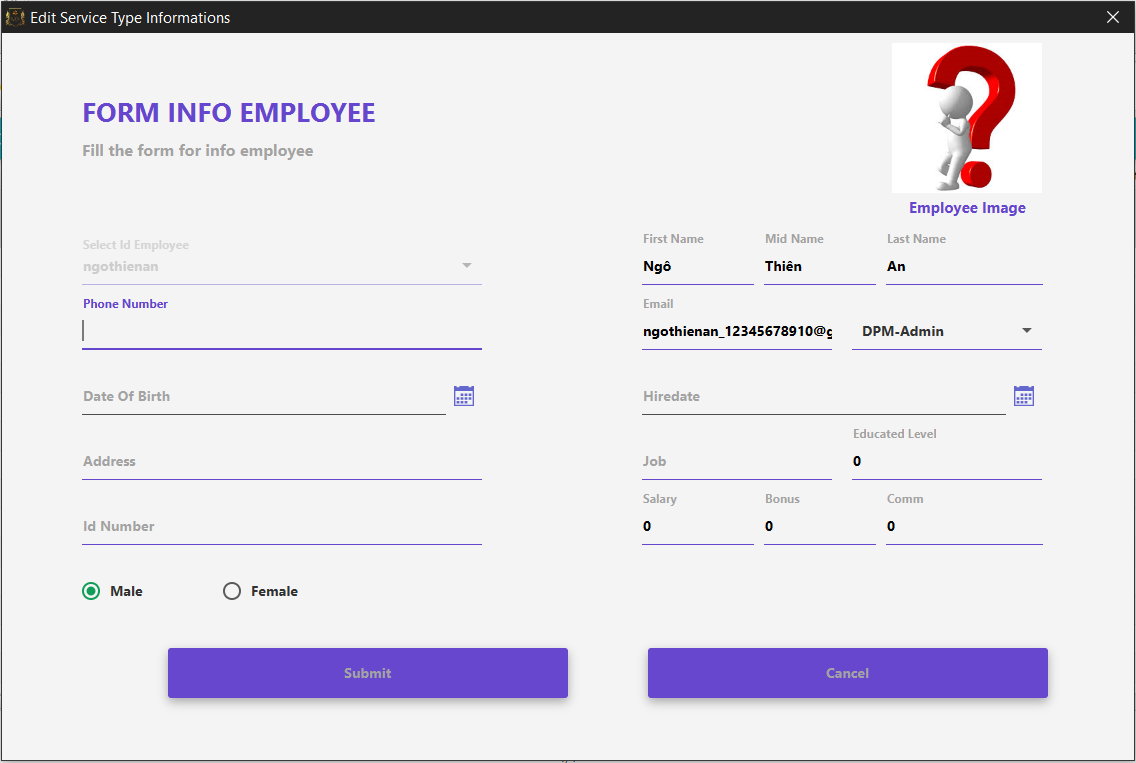
This form display all employees’s information:



***Figure 6.15: List employees form***

## 6.16. Editing employees form:

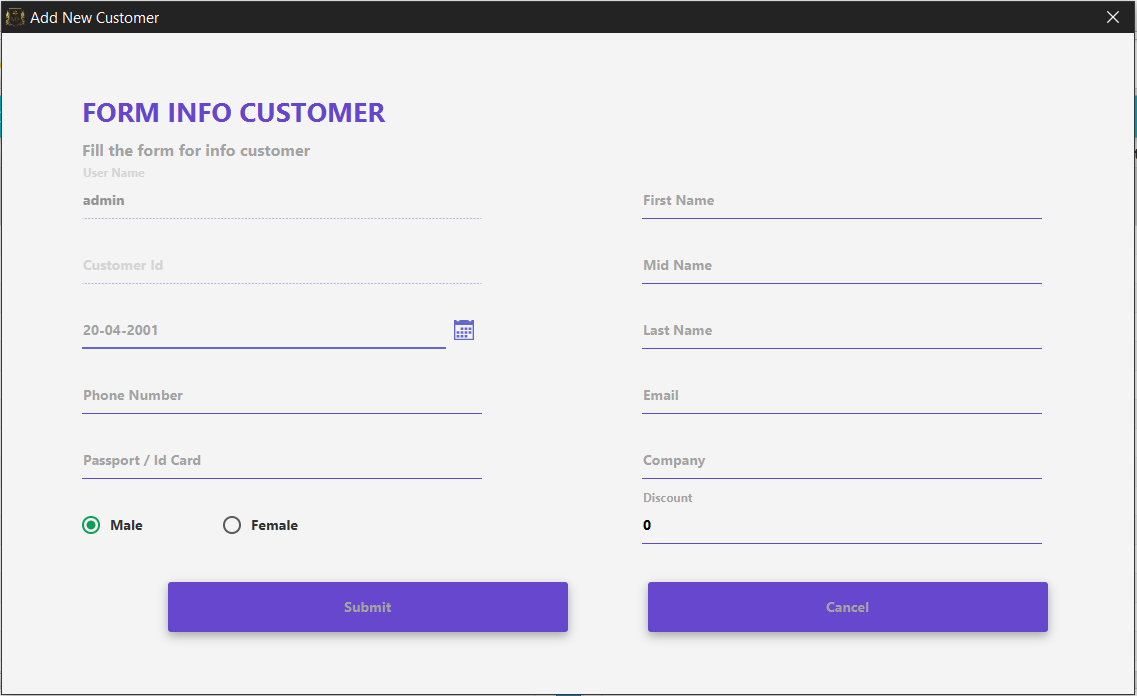
This form allow adding employees’s detail information:



***Figure 6.16: Edit employee’s information form***

## 6.17. Add customer form:

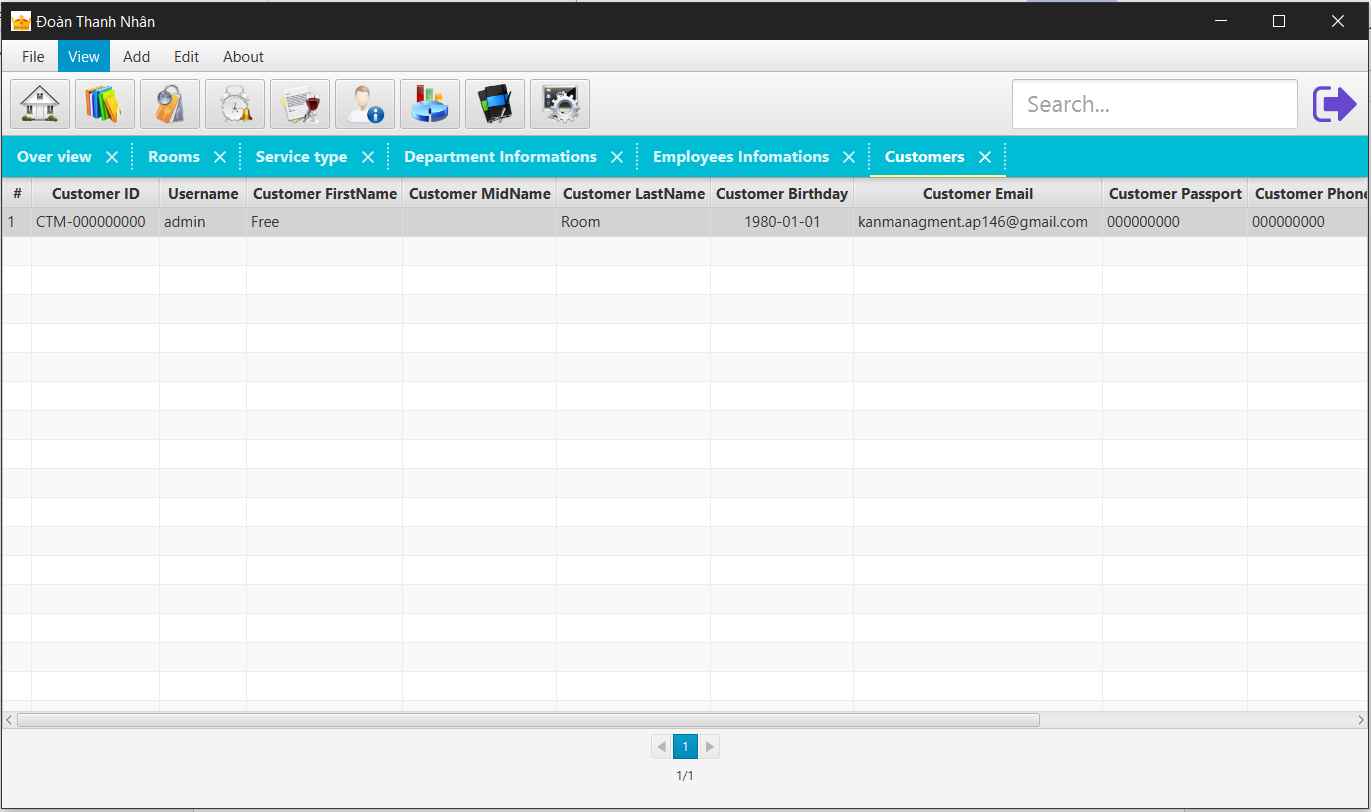
This form allow adding employees’s detail information:



***Figure 6.17: Edit employee’s information form***

## 6.18. List customers form:

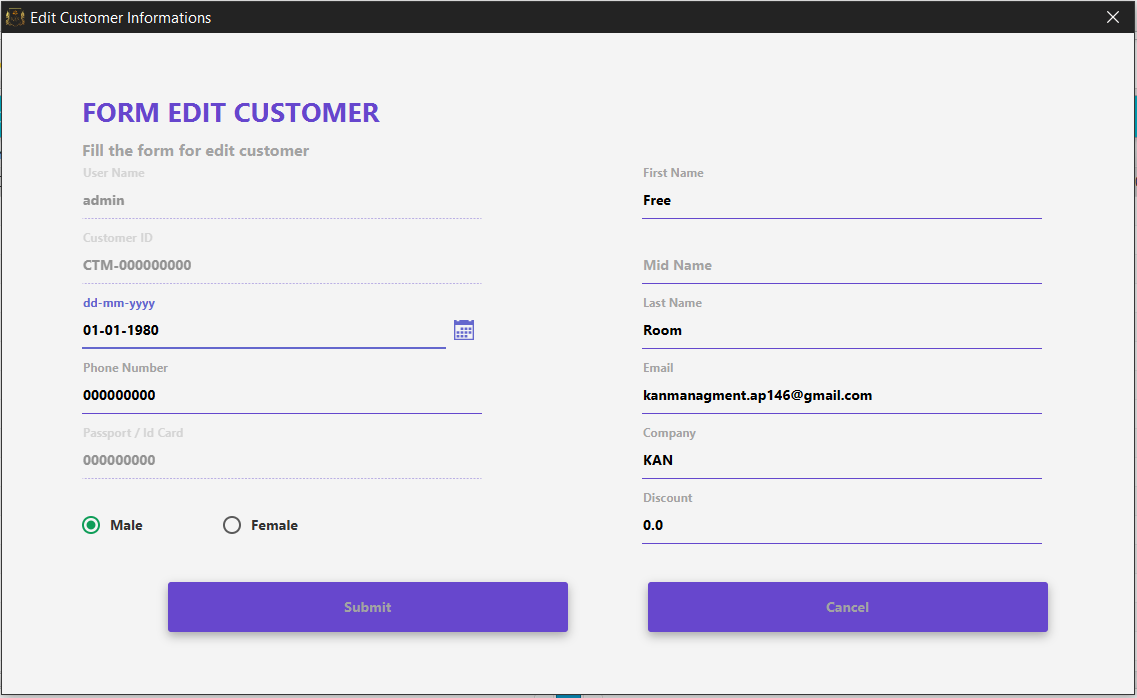
This form display all employees’s detail information:



***Figure 6.18: List customer’s information form***

## 6.19. Edit customers form:

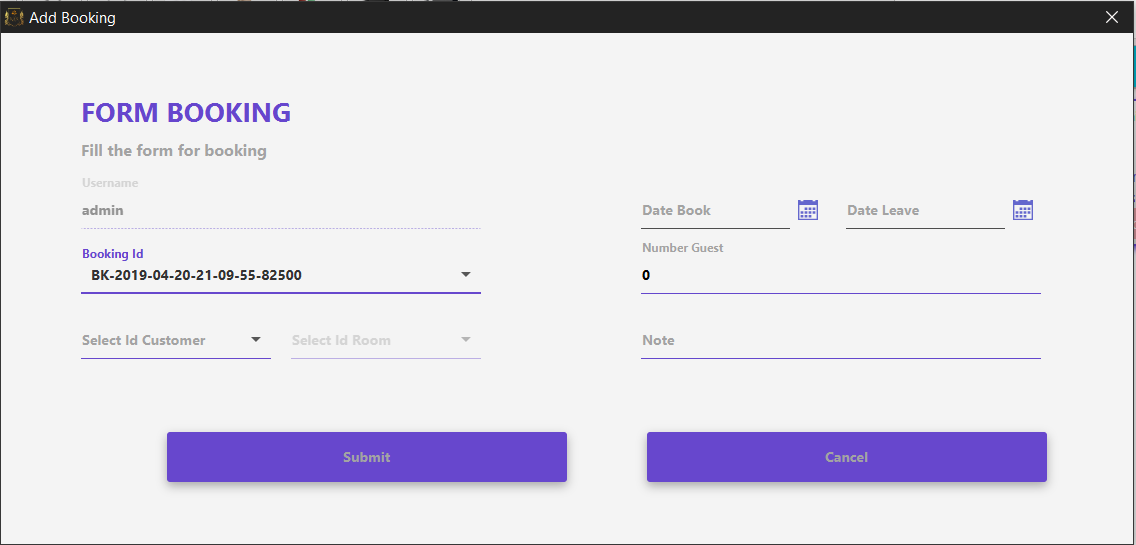
This form allow editting employees’s detail information:



***Figure 6.19: Edit customer’s information form***

## 6.20. Add booking form:

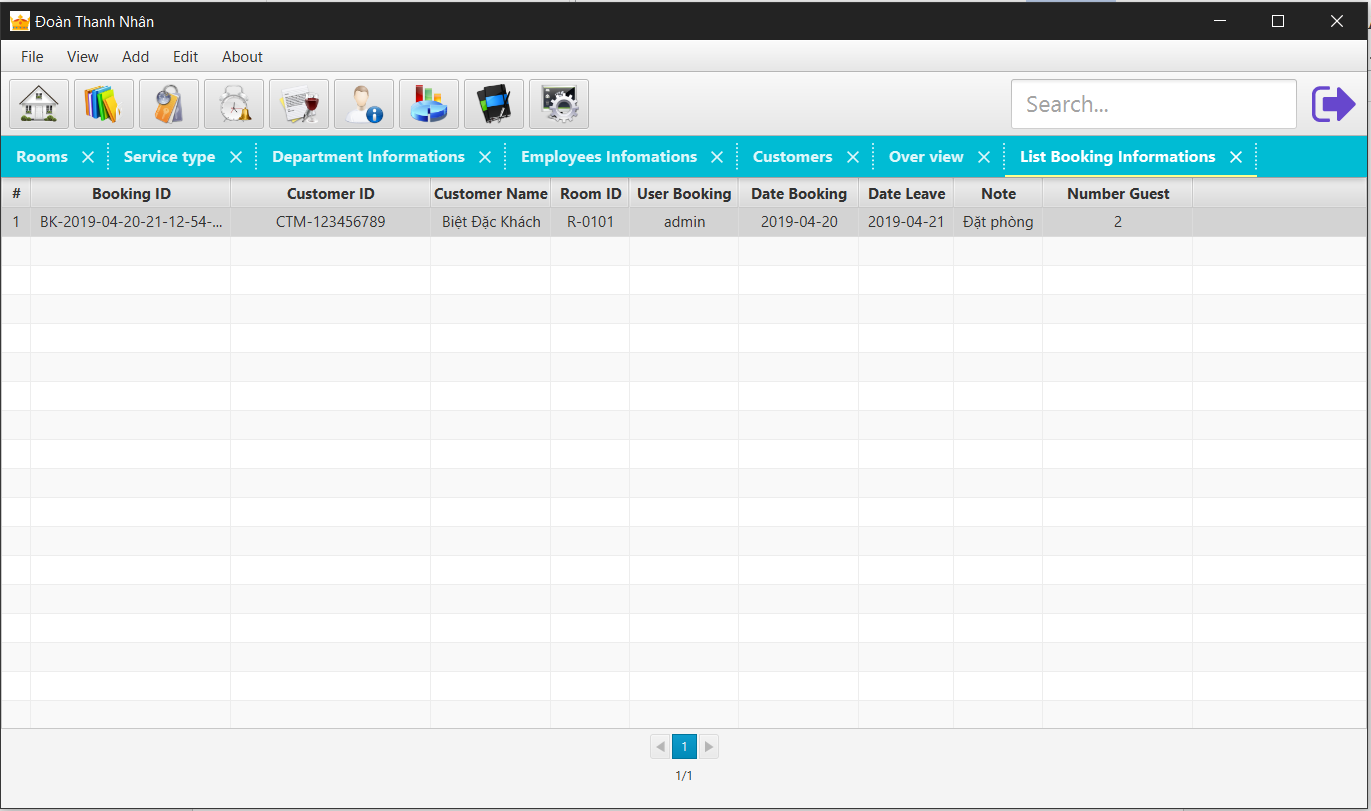
When having a customer wanna reserve a room, we have booking form like this :



***Figure 6.20: Add booking form***

## 6.21. List booking form:

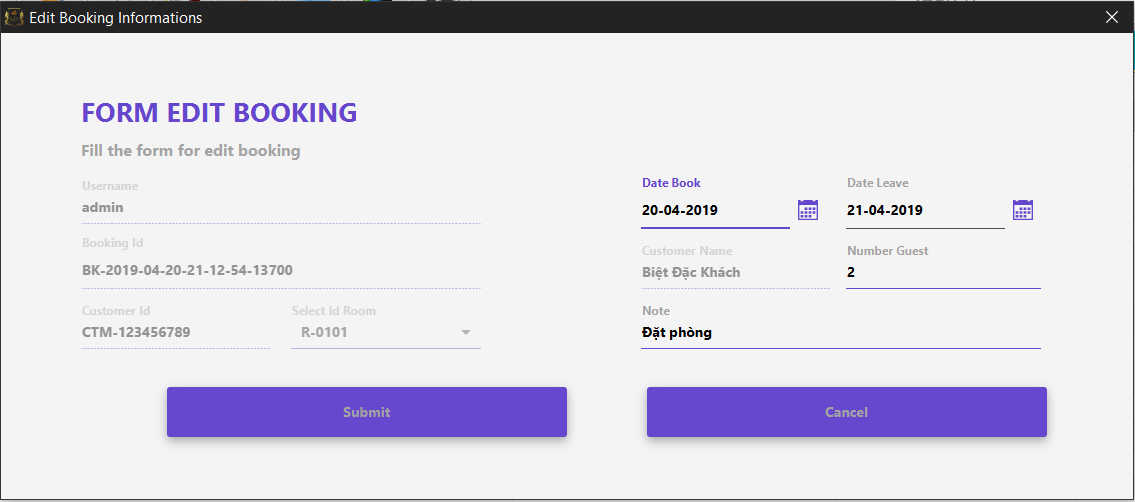
This form display all booking’s detail information:



***Figure 6.21: List booking form***

## 6.22. Edit booking form:

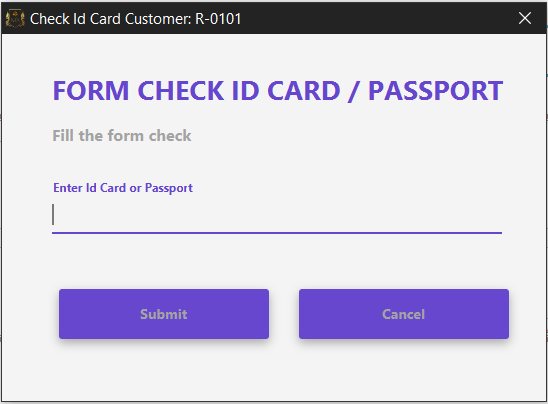
This form allow editing booking’s detail information:



***Figure 6.22: Edit booking form***

## 6.23. Checking customer’s ID card number form:

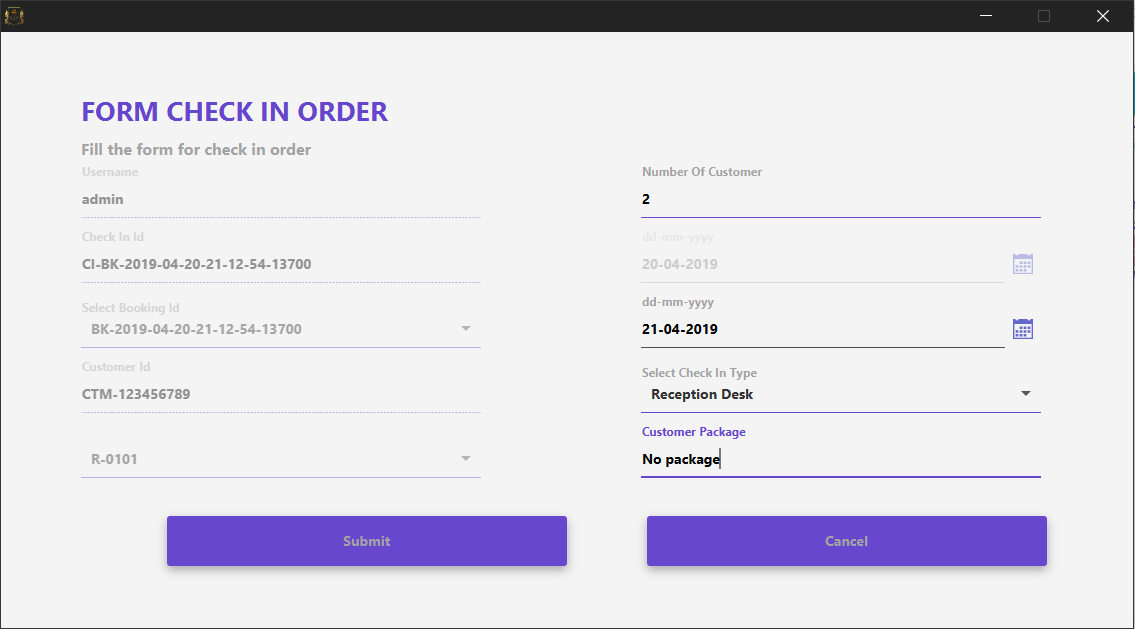
This form check customer ID (ID card number). If it is exist, that customer had stayed at hotel before and the “Add new Customer” form appear with customer’s information for receptionist checking again , if not the “Add new Customer” form will appear for adding new customer :



***Figure 6.23: Check customer ID card number form***

## 6.24. Check in form:

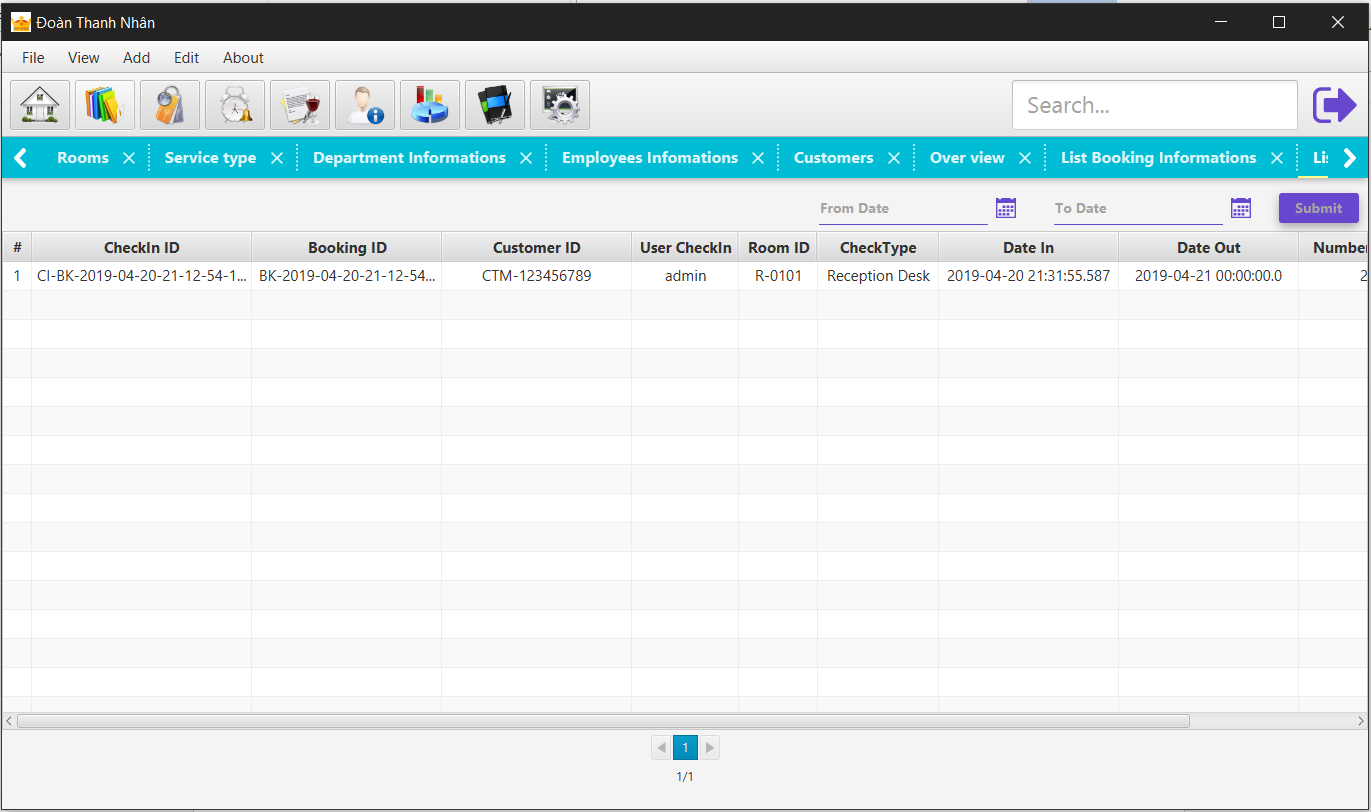
This form allow customer register check in room:



***Figure 6.23: Check in form***

## 6.25. Check in form:

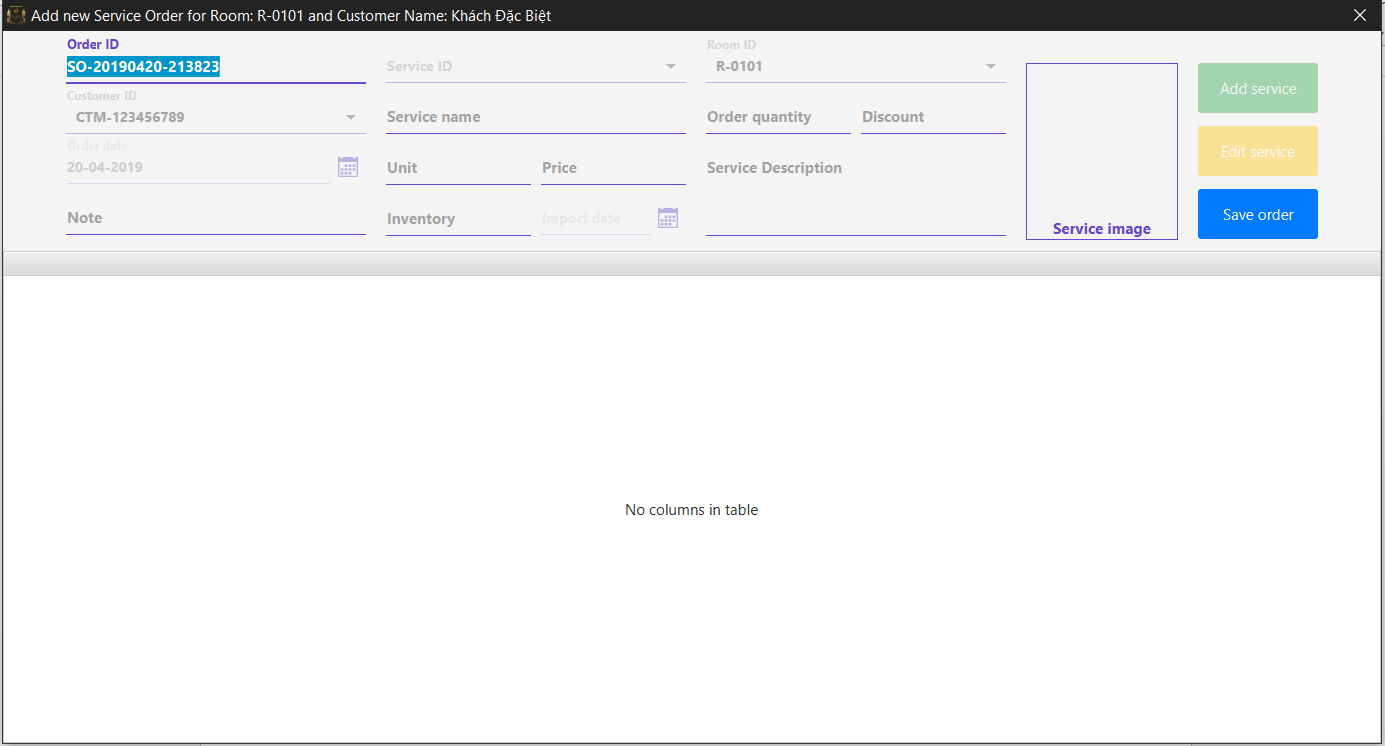
This form display all check in registrations:



***Figure 6.25: List check in form***

## 6.26. Service order form:

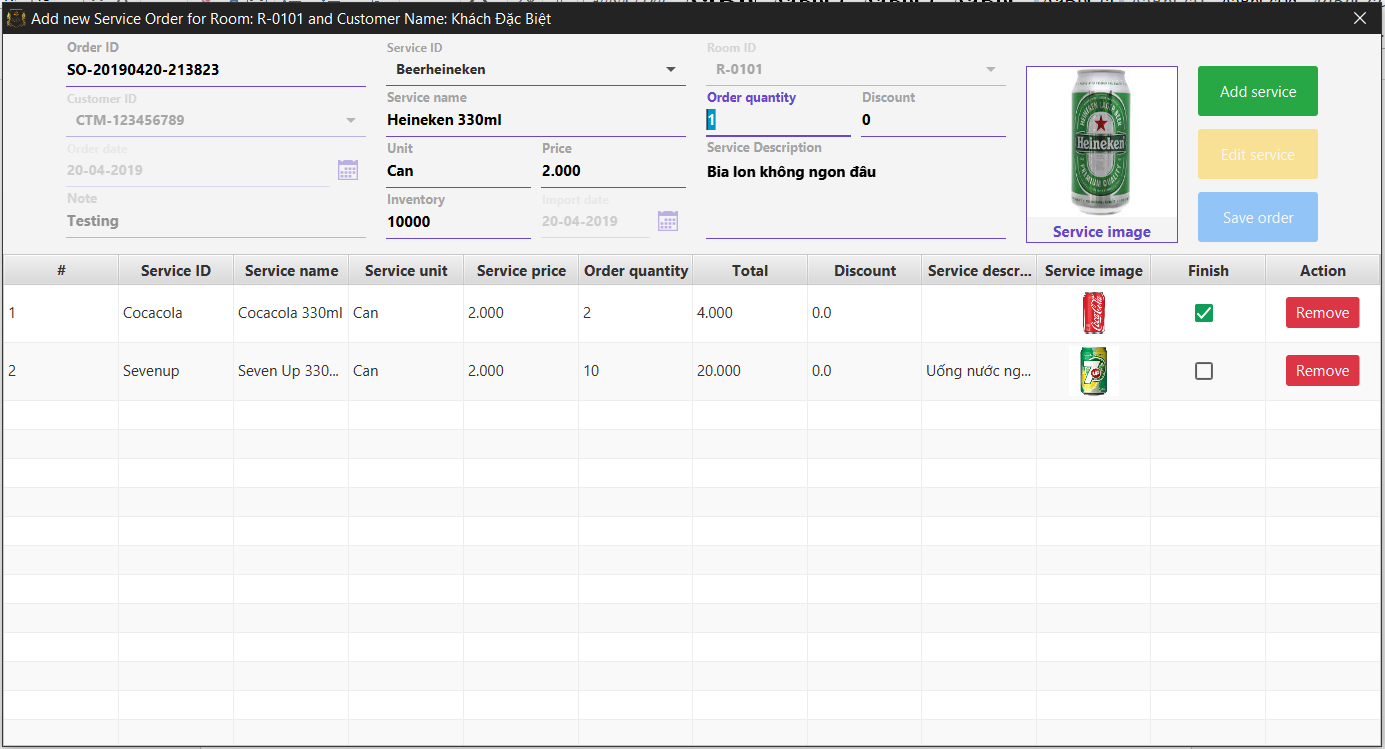
After registering check in finish, customer can order hotel’s services via this form:



***Figure 6.26: Add new service order form***

## 6.27. Service order detail form:

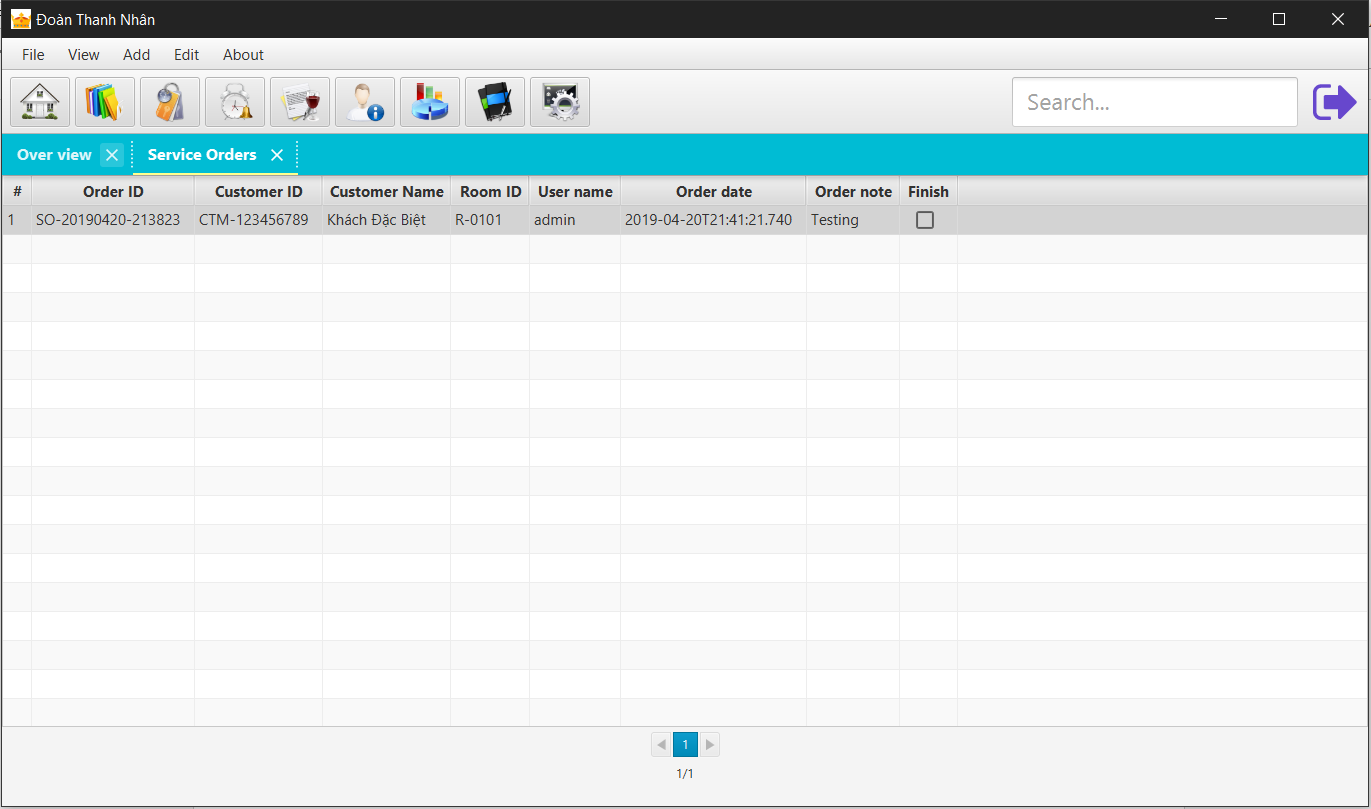
Adding service into service order:



***Figure 6.27: Add new service order detail form***

## 6.28. List Service order form:

This form display all service order’s information:



***Figure 6.28: List service order form***

## 6.29. List detail Service order form:

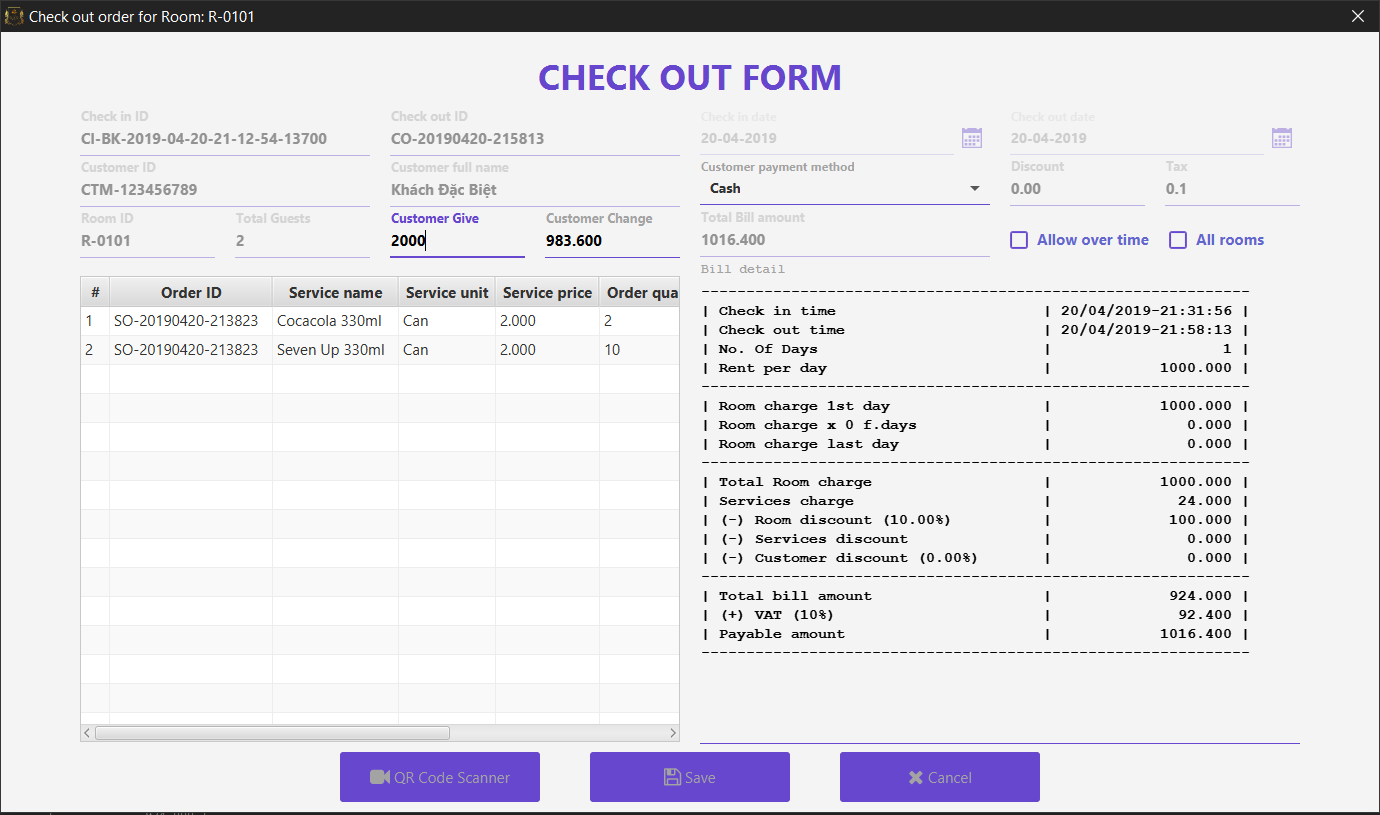
This form display all detail service order’s information:



***Figure 6.29: List detail service order form***

## 6.30. Check out form:

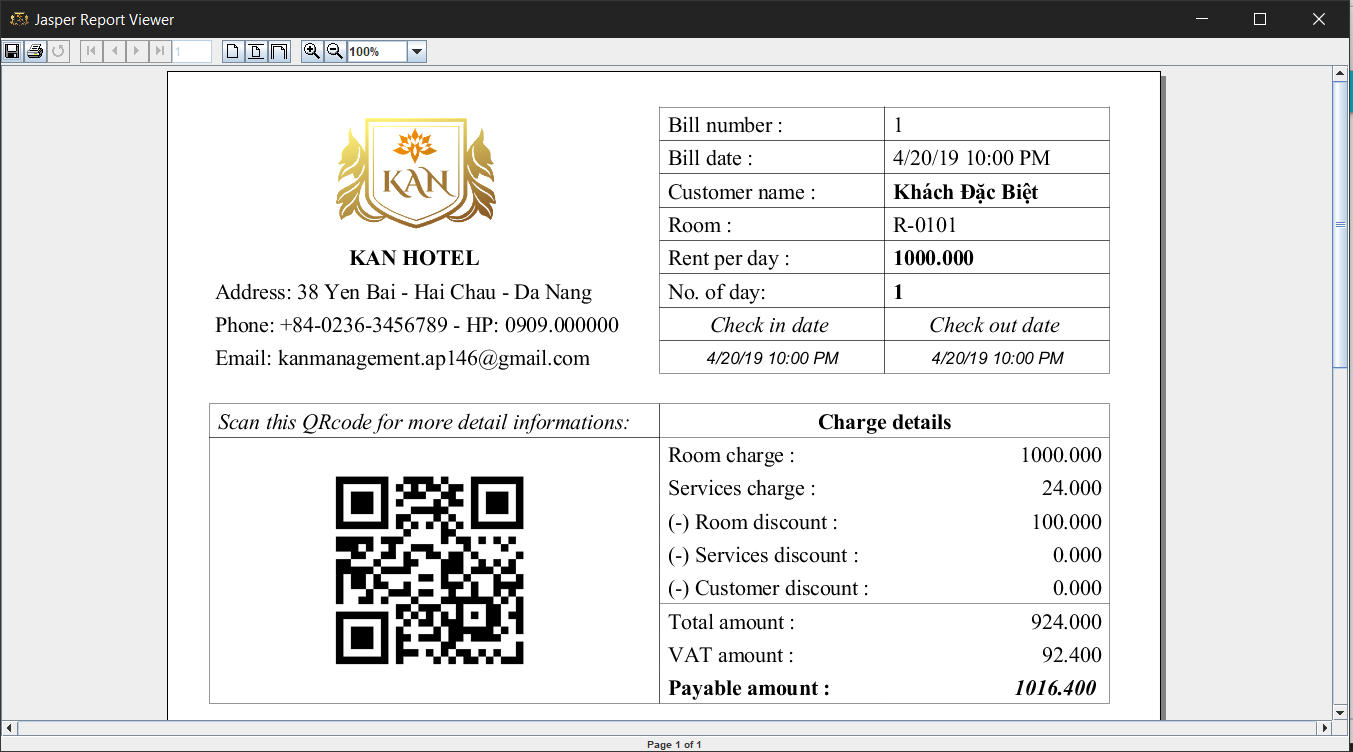
This form allow customer registering check out :



***Figure 6.30: Check out form***

## 6.31. Bill and Invoice:

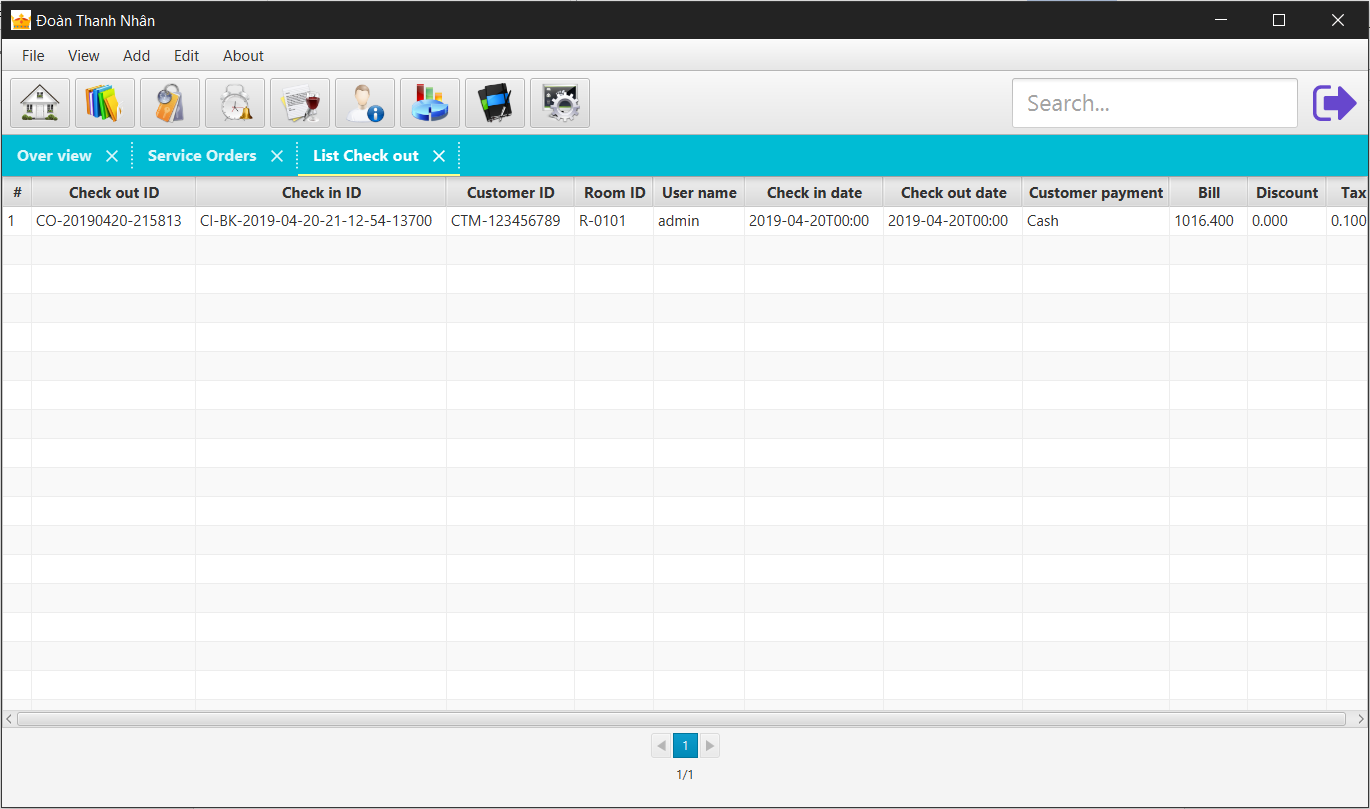
Display invoice information:



***Figure 6.31: Bill report form***

## 6.31. List check out form:

This form display all check out information :



***Figure 6.31: Bill report form***

## 6.32. List virtual booking form:

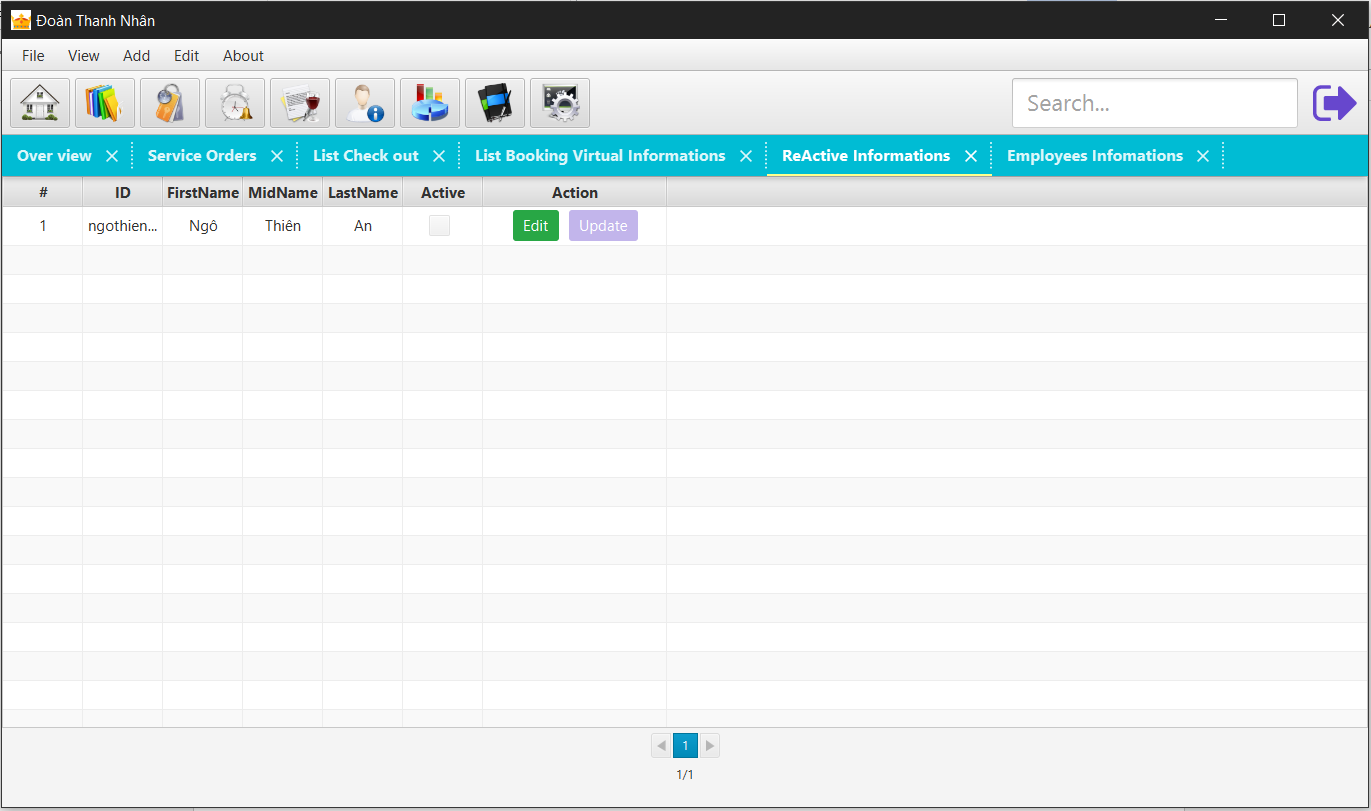
This form display all virtual booking (the booking but customer don’t come to register check in) information :



***Figure 6.32: List virtual booking form***

## 6.33. List ReActive Employee form:

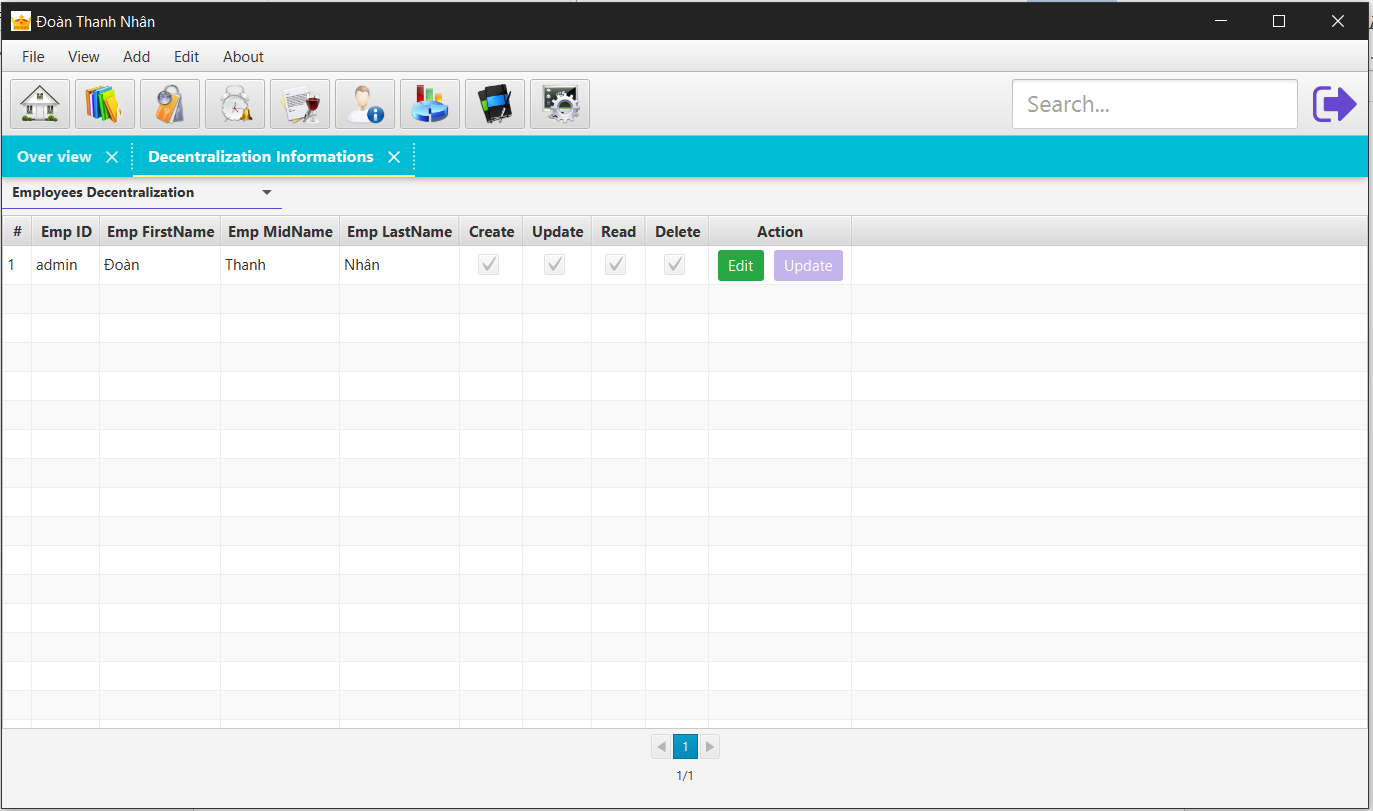
This form display all deleted employees for reactiving purpose information :



***Figure 6.33: List reactive employee form***

## 6.34. List Role form:

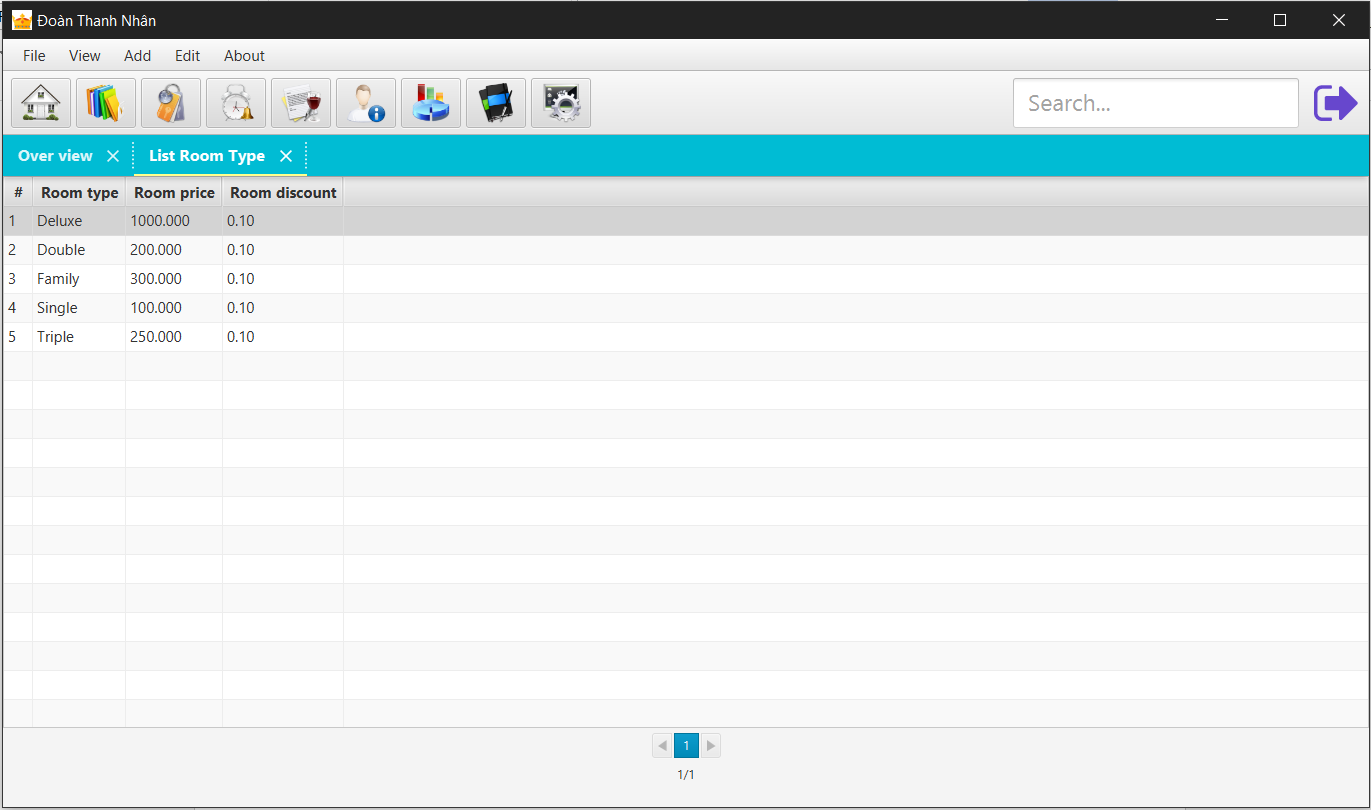
This form display all user’s role and allow editing role :



***Figure 6.34: List role form***

## 6.35. List Room type form:

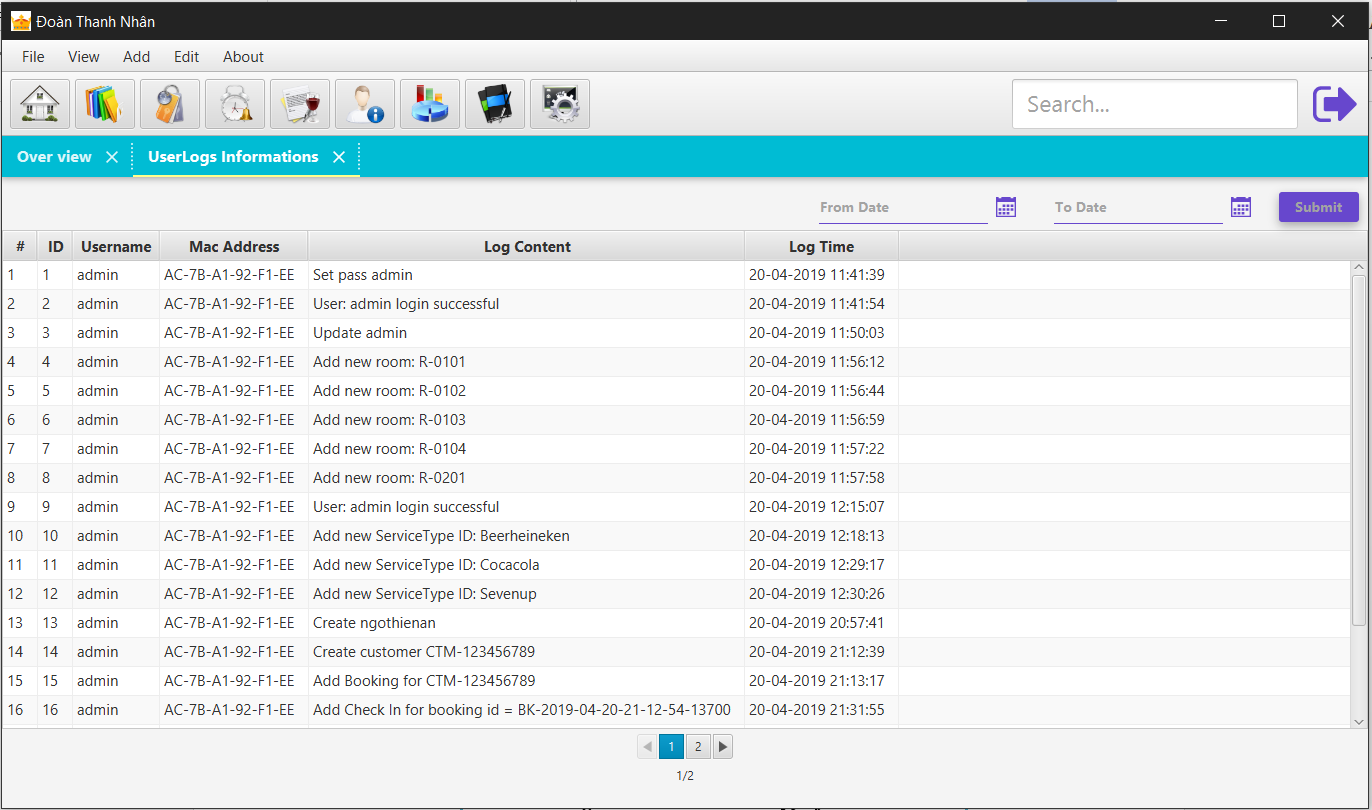
This form display all room type and allow editing room type :



***Figure 6.35: List room type form***

## 6.36. List User log form:

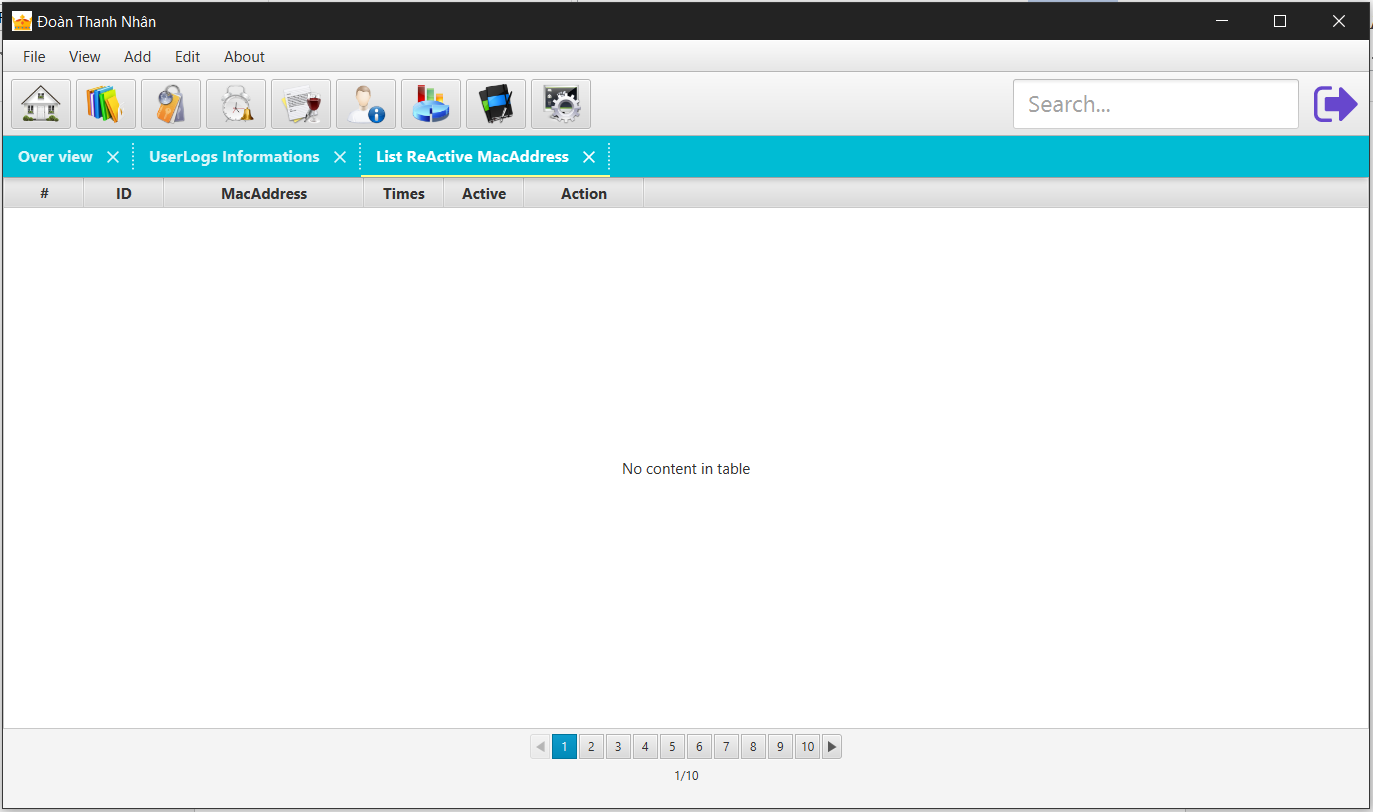
This form display all user’s role and allow editing role :



***Figure 6.36: List user logs form***

## 6.36. List Reactive MAC address form:

This form display all blocked MAC address and can reactive this here :



***Figure 6.36: List reactive MAC address form***

# CHAPTER 7: USER GUIDE

## 7.1. Instructrion:

We are using Node JS for reading and saving data so need install Node JS first.

## 7.2. Running step by step:

### 7.2.1. Download project:

We are using GIT and GitHub for remote repository. There 2 ways you can download this project:

### 7.2.1.1. Without GIT:

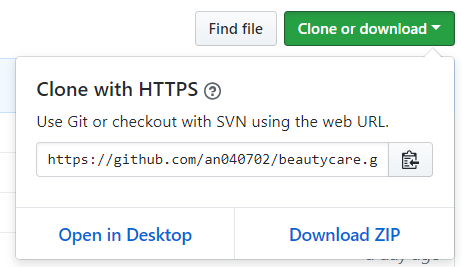
+ Go to <https://github.com/>

+ Enter: beautycare at Search Box (Search Github)

+ Then select:



+ Click “**Clone or Download**” button then click “**Download ZIP**”:



### 7.2.1.2. Using GIT:

Open cmd.exe or Git Bash then enter this command:

***Git clone*** [***https://github.com/an040702/beautycare.git***](https://github.com/an040702/beautycare.git)

### 7.2.2. Running server:

Go to ***beautycare*** folder, then use cmd.exe, and enter this command:

***Node server.js***

The path should look like this firgure:

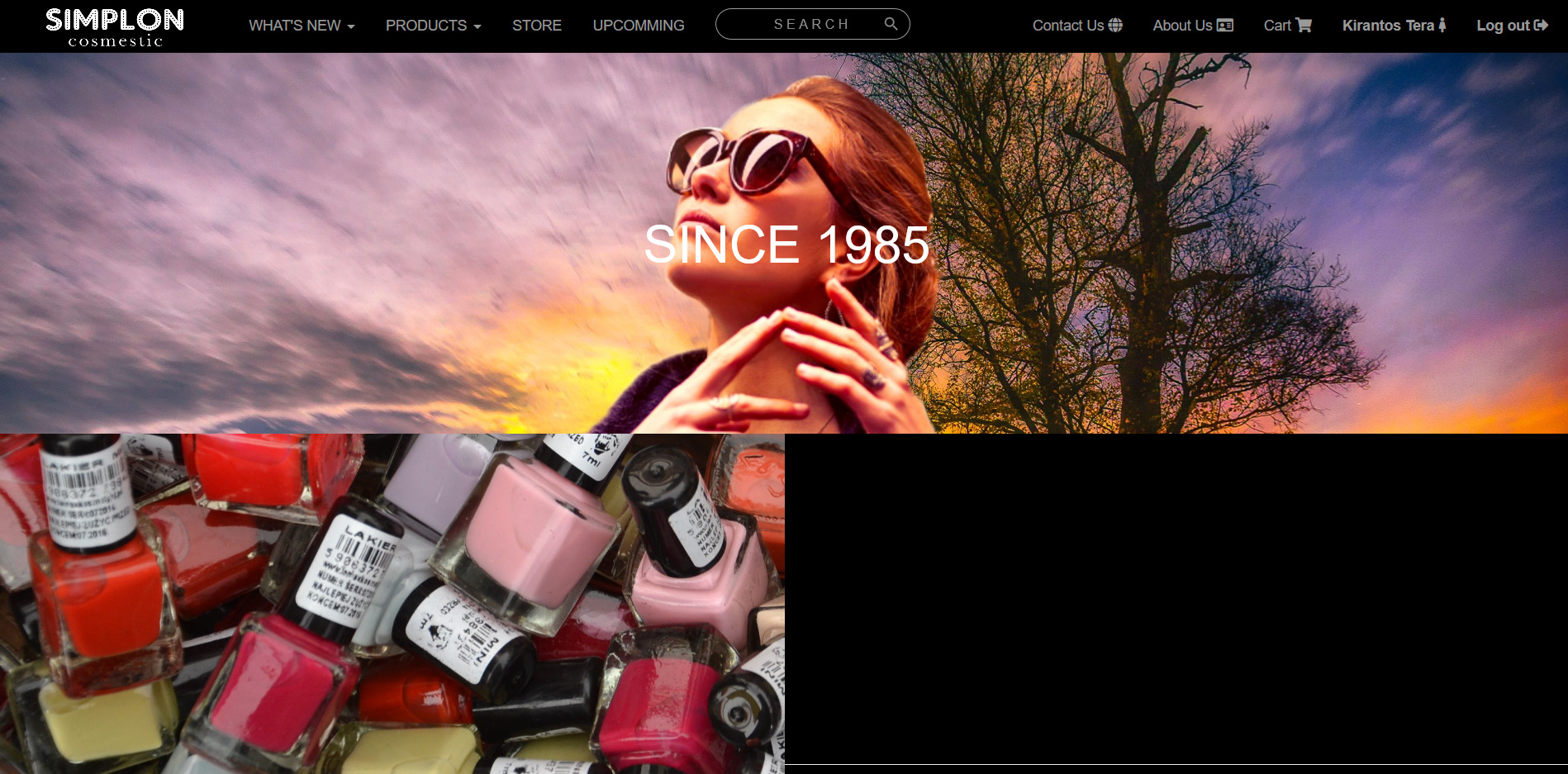


### 7.2.3. Running web pages:

Open browser, we use **Chrome** for example. At url input enter this:

***Localhost:8080***

Website will display like this:



Then you can view what ever you want via navigation bar.

## 7.3. Developer’s guide:

Our project on GitHub:

