

**Carti Hotel Management System**

**Use-case-Realization Specification: Manage Users**

**Version <1.0>**

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 24th Nov 2024 | Version 1.0 | Final version | Tran Quang Huy |
|  |  |  |  |

**Table of Contents**

[Revision History 2](#_heading=h.1fob9te)

[Use-Case-Realization Specification: Manage Users 4](#_heading=h.3znysh7)

[1.](#_heading=h.tyjcwt) Introduction 4

[1.1](#_heading=h.1t3h5sf) Purpose 4

[1.2](#_heading=h.2s8eyo1) Scope 4

[1.3](#_heading=h.3rdcrjn) Definitions, Acronyms, and Abbreviations 4

[1.4](#_heading=h.lnxbz9) References 4

[1.5](#_heading=h.1ksv4uv) Overview 4

[2.](#_heading=h.2jxsxqh) Flow of Events—Design 4

[2.1.](#_heading=h.3j2qqm3) Flow of events 4

[2.1.1.](#_heading=h.4i7ojhp) Add User 4

[2.1.2, Update User 4](#_heading=h.2xcytpi)

[2.1.3. Delete User 5](#_heading=h.1ci93xb)

[2.2.](#_heading=h.qsh70q) Sequence diagram 5

[2.2.1.](#_heading=h.3as4poj) Add User 5

[2.2.2.](#_heading=h.1pxezwc) Update User 6

[2.2.3.](#_heading=h.49x2ik5) Delete User 7

[3.](#_heading=h.147n2zr) Derived Requirements 8

# Use-Case-Realization Specification: Manage Users

## Introduction

### Purpose

This document provides an overview of how the "Manage Users" use case is implemented within the design model of the Carti Hotel Management System project, emphasizing the collaboration of objects involved in the process.

### Scope

This document applies to the Carti Hotel Management System.

### Definitions, Acronyms, and Abbreviations

None

### References

None

### Overview

The next section thoroughly covers the design of the use case and its related requirements. In particular, the flow of events-design portion incorporates the development of both the class diagram and sequence diagram for the use case, along with brief written explanations.

After presenting the design, the following section details the derived requirements that are vital for the successful execution of the use case. These requirements act as key guidelines to guarantee the proper implementation and functionality of the use case.

## Flow of Events—Design

### Flow of events

The diagrams presented illustrate the key classes and the order in which they interact to implement the "Manage Users" use case.

After a user logs in successfully as an admin, they are taken to the Admin Page. There, the Room Controller calls the loadRoom() method to fetch all room numbers and their availability statuses, displaying this information sequentially on the Admin Page. This setup allows the admin to effectively manage all users, including adding, updating, and deleting user accounts, by selecting the "Users" button located on the left side.

### Add User

When the admin selects the "Add User" button, the redirect() function is triggered, guiding the admin to the Add User Page. There, the admin enters the necessary information to create a new user and clicks the Save button. The checkIsUserExist() function is then called to verify the information. If the user does not already exist in the system, the User Controller adds the new user to the database using the addUser() function, and the showSuccessMessage() function is invoked to display a success message. If the user already exists, the showErrorMessage() function is called to notify the admin of the error.

### 2.1.2, Update User

After an admin successfully logs in, they are redirected to the Admin Page. The Room Controller then calls the loadRoom() method to fetch all room numbers and their availability statuses, which are displayed sequentially on the Admin Page.

When the admin selects the "User" button and then the "Update User" option, the loadUser() function is triggered to load all user information. Following this, the User Controller invokes the getListOfUsers() and showUserList() functions to retrieve and display the user data in a table format.

To update a user, the admin clicks on a specific user, which redirects them to the Update User Page. On this page, the loadUserInfo() and showUserInfo() functions are called to request and display the selected user's information from the database through the getUserInfo() function.

Next, the admin enters the necessary details to modify the user's information and clicks the "Save" button. The validateInformation() function is then called to ensure the data is valid. If the information passes validation, the User Controller updates the user in the database using the updateUser() function and triggers the showSuccessMessage() function to notify the admin of the successful update. If the information is invalid, the showErrorMessage() function is activated to inform the admin of the error.

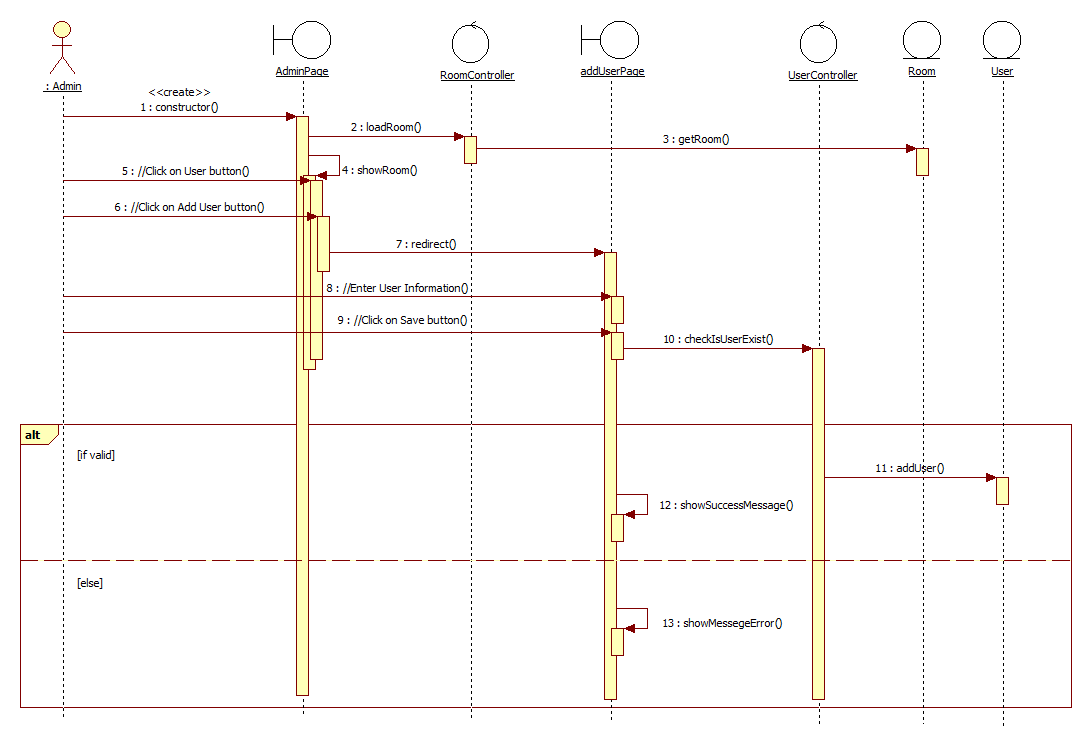
### 2.1.3. Delete User

When the admin selects the "Delete User" button, the redirect() function is triggered, sending the admin to the Delete User Page.

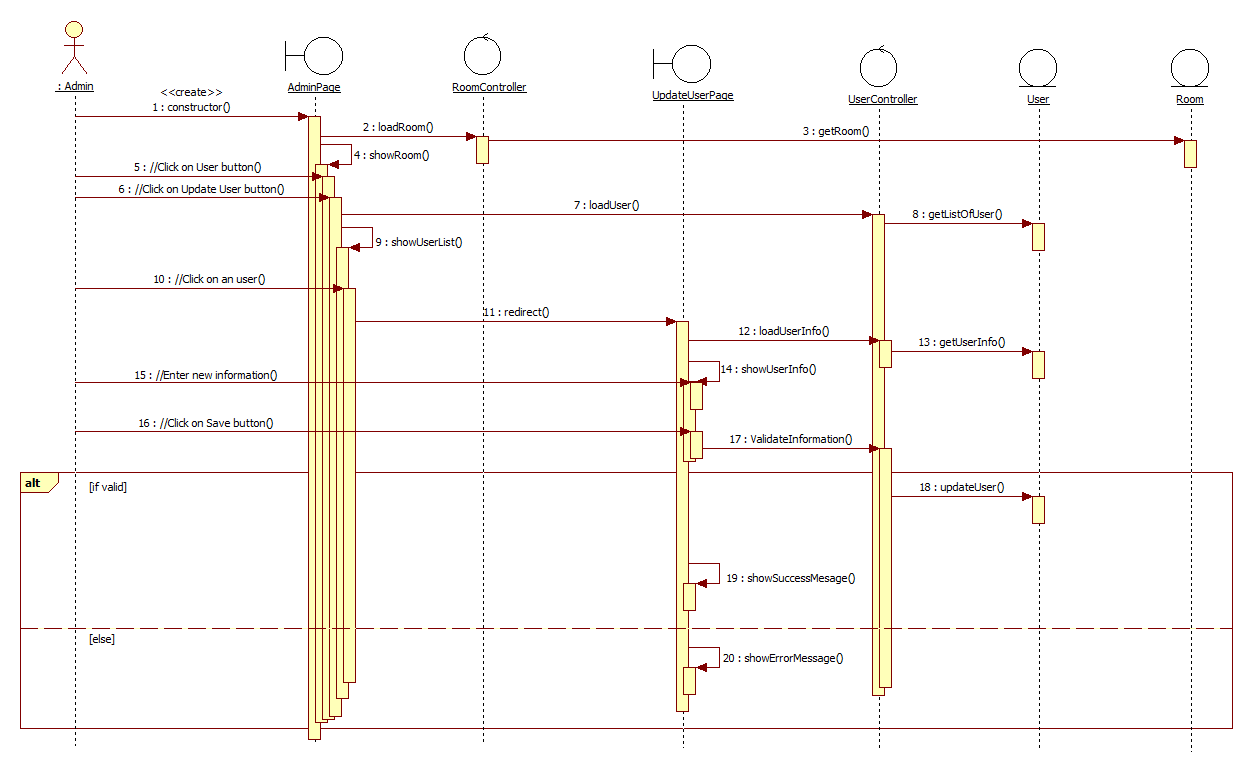
To remove a user, the admin clicks on a specific user row, which then calls the checkConfirm() function to display a confirmation dialog. If the admin confirms the deletion, the deleteUser() function within the User Controller is executed, removing the user's information from the database. If the admin does not confirm, no further action is taken.

### Sequence diagram

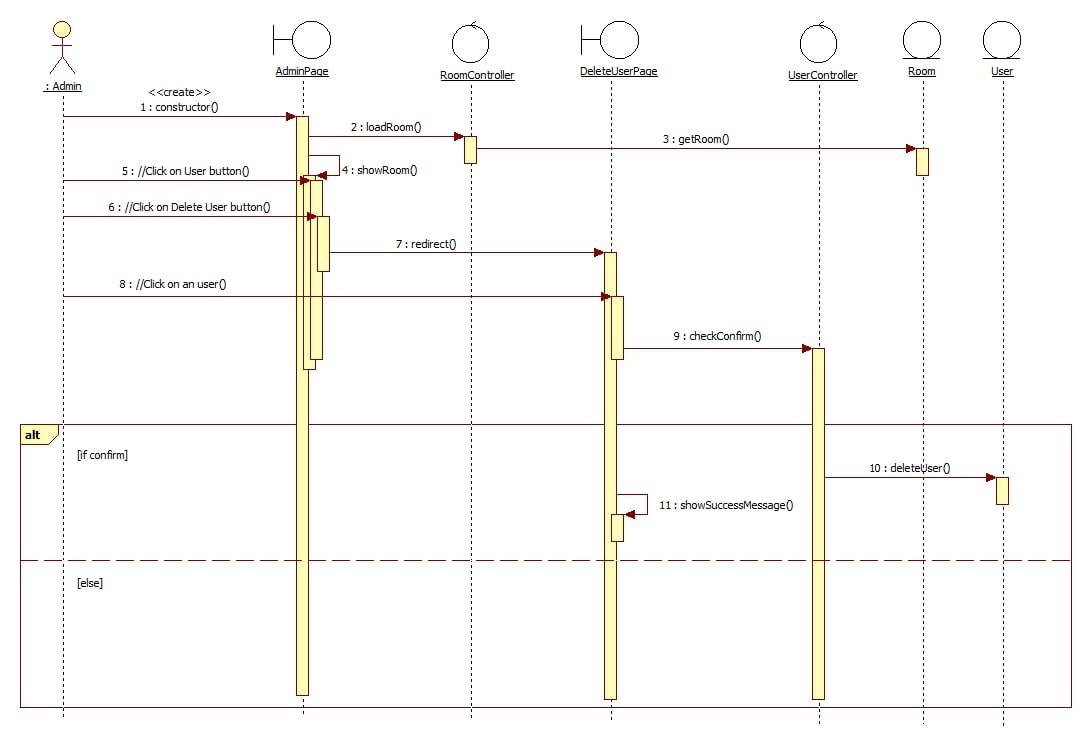
### Add User

****

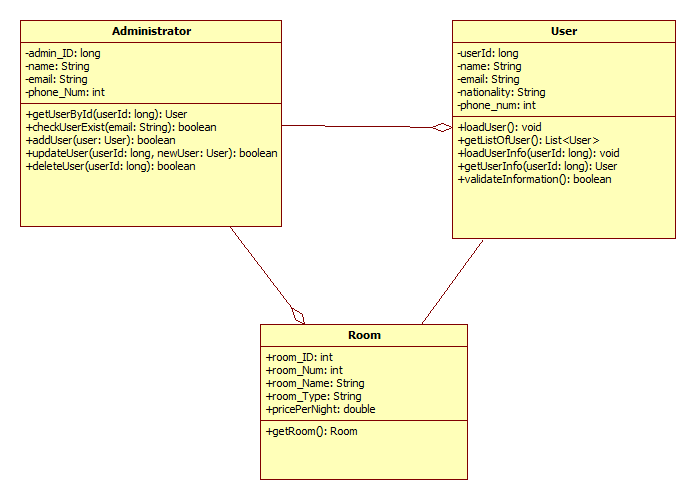
### Update User



### Delete User



**2.3 Class diagram**

****

## Derived Requirements

None