Version <1.0>

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 24/11/2024 | 1.0 | Final version | Dang Gia Nguyen |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 2

1.1 Purpose 2

1.2 Scope 2

1.3 Definitions, Acronyms, and Abbreviations 2

1.4 References 2

1.5 Overview 2

2. Flow of Events—Design 2

3. Derived Requirements 2

# Introduction

## Purpose

This document describes how the Manage Services use-case is realized within the design model of the Carti Hotel Management System projects, in terms of collaborating objects.

## Scope

This document applies to the Carti Hotel Management System.

## Definitions, Acronyms, and Abbreviations

None

## References

None

## Overview

The next section details the design of the use case and its corresponding requirements. Specifically, the Flow of Events-Design subsection includes the implementation of the class diagram and sequence diagram for the use case, complemented by brief textual descriptions.

Following this, the Derived Requirements section outlines the essential requirements needed to implement the use case. These requirements act as critical guidelines to guarantee the successful execution and functionality of the use case.

# Flow of Events—Design

### Flow of events

The diagrams below depict the key classes and the collaborative sequence necessary to implement the Manage Services use case.

Once the user logs in as a receptionist, they are navigated to the Receptionist Page. At this point, the Room Controller invokes the loadRoom() method to fetch all room numbers along with their availability statuses. These details are displayed sequentially on the Receptionist Page. From there, the receptionist can manage all services for a specific customer, including adding new services or removing existing ones.

### Add service

On the Receptionist Page, the receptionist enters a customer's name into the search bar and clicks the search button, triggering the loadCustomer() method. The Search Controller then calls the getListOfCustomer() function to retrieve all matching customers. The results are displayed on the screen using the showListCustomer() function.

When a customer is selected, the system redirects to the Reservation Detail Page. Here, the Reservation Controller invokes the getReservation() function to fetch the selected customer's information from the database. The customer's reservation history is then displayed using the showReservation() method.

Clicking the Service Listing Button navigates the receptionist to the Service Page, where the loadAllService() method is executed. The Service Controller calls the getAllServices() function to retrieve all services linked to the reservation. These services are displayed on the screen using the showServices() method.

To add a service, the receptionist selects a reservation and clicks the Add Service Button. After choosing the desired services, the receptionist clicks the Save Button, triggering the validateServiceInfo() function to validate the inputs. If the inputs are valid, the Service Controller executes the addService() function to store the services in the Service database. Simultaneously, the services are linked to the customer's reservation and bill using the addReservationService() and addBillService() functions, respectively.

Finally, the showSuccessMessage() method is called to notify the receptionist of the successful addition of the services.

### Delete service

On the Receptionist Page, the receptionist enters a customer's name into the search bar and clicks the search button, initiating the loadCustomer() method. The Search Controller then executes the getListOfCustomer() function to retrieve all matching customers. The results are displayed on the screen using the showListCustomer() function.

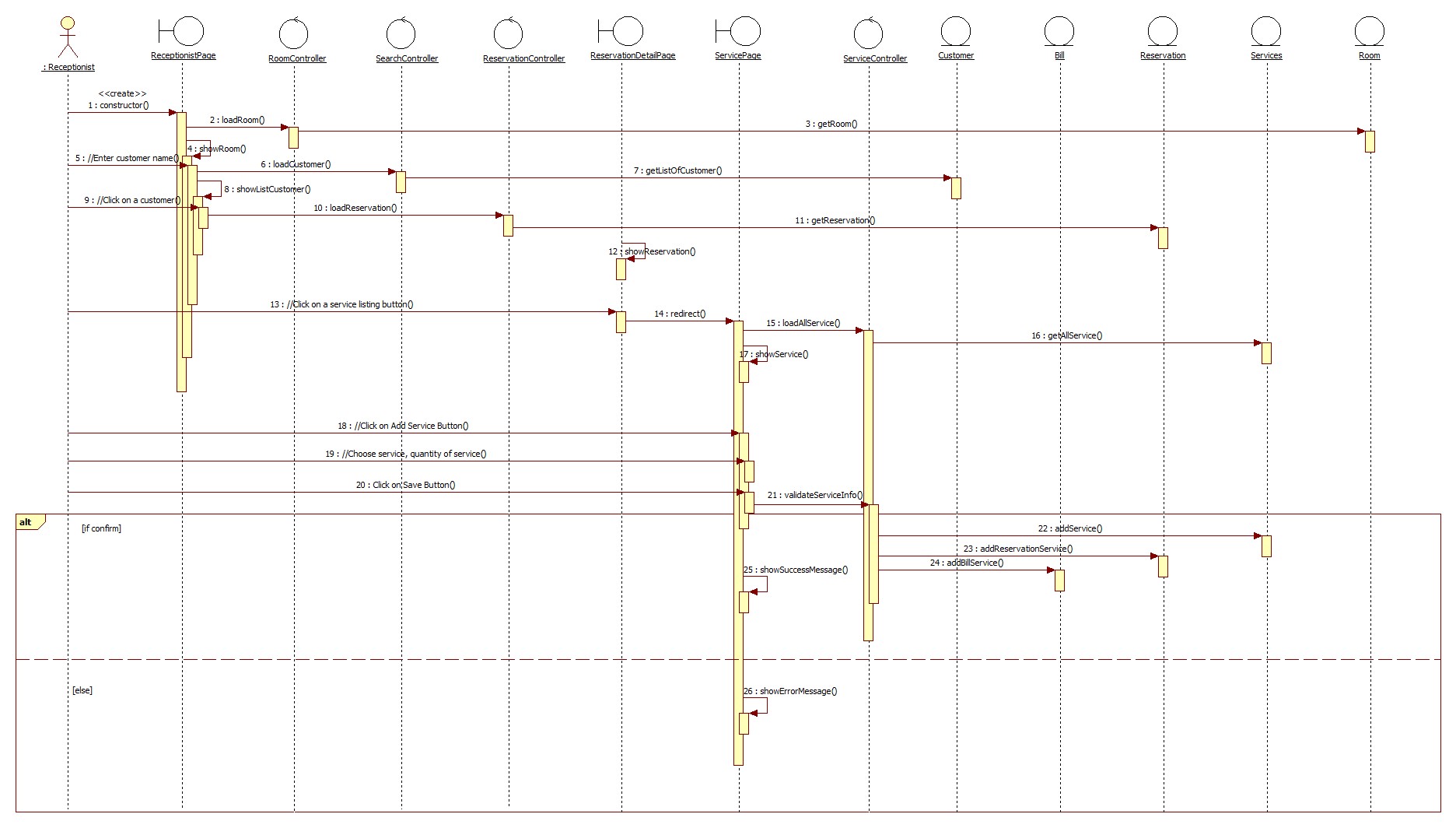
On the Reservation Detail Page, the receptionist clicks the Service Listing Button, which redirects to the Service Page. Here, the loadAllService() method is called, and the Service Controller invokes the getAllService() function to fetch all services associated with the selected reservation. These services are displayed on the screen using the showService() method.

To delete a service, the receptionist selects a service and clicks the Delete Service Button, prompting a confirmation box to appear. This triggers the confirmChecking() function to verify whether the receptionist confirms the deletion. If confirmed, the Service Controller executes the deleteService() function to remove the service from the database and calls the deleteBillService() function to update the customer's bill. Finally, the showSuccessMessage() function is invoked to inform the receptionist of the successful deletion.

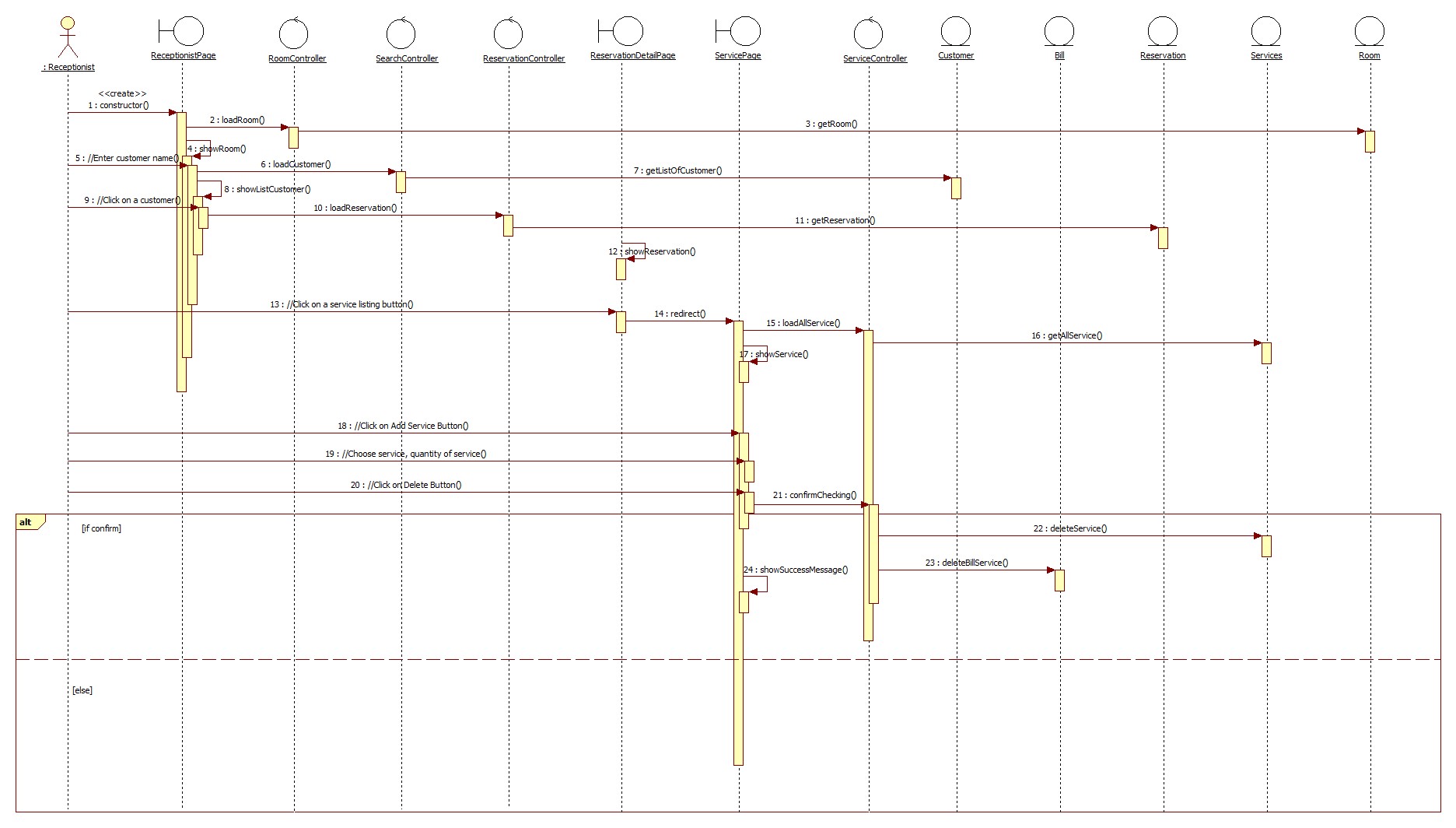
If the receptionist does not confirm the deletion, the Service Controller terminates the process by calling the cancelDeletingService() function.

## sequence diagram

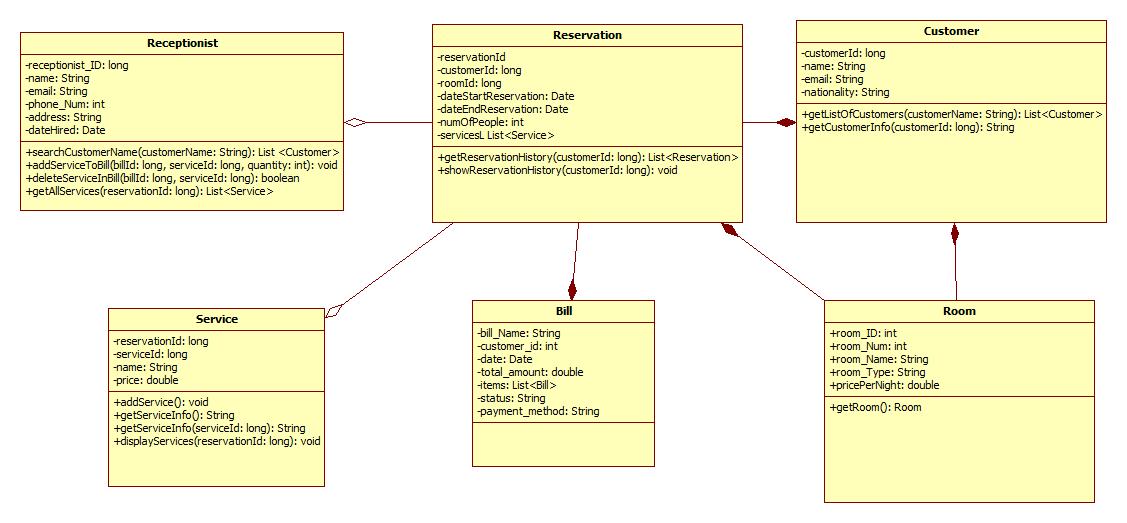
* + 1. **Add Service**



### **Delete Service**



## class diagram



# Derived Requirements

none