

Program ‘Til it Hertz

Felipe Corona

Jahiro Otzoy

Mauricio Lomeli

Brian Vu

Salman Shah

Hotel Management System

Software Requirements Specification

Version 1.0

CS 200 - Software Engineering

Instructor: B. Clarke

Spring 2014

Contents

[1. Introduction 11](#_Toc388446593)

[1.1 Purpose 13](#_Toc388446594)

[1.2 Scope 13](#_Toc388446595)

[1.3 Definitions 14](#_Toc388446596)

[2 General Description 22](#_Toc388446597)

[2.1 Product Perspective 22](#_Toc388446598)

[2.1.1 System Interfaces 22](#_Toc388446599)

[2.1.2 Interfaces 22](#_Toc388446600)

[2.1.3 Hardware Interfaces 23](#_Toc388446601)

[2.1.4 Software Interfaces 23](#_Toc388446602)

[2.1.5 Communication Interfaces 24](#_Toc388446603)

[2.1.6 Memory Constraints 25](#_Toc388446604)

[2.1.7 Operations 25](#_Toc388446605)

[2.1.8 Site Adaptation Requirements 26](#_Toc388446606)

[2.2 System Environment 26](#_Toc388446607)

[2.3 Product Functions 27](#_Toc388446608)

[2.4 User Characteristics 29](#_Toc388446609)

[2.5 Constraints 29](#_Toc388446610)

[2.6 Assumptions and Dependencies 30](#_Toc388446611)

[3 Specific Requirements 31](#_Toc388446612)

[3.2 Features 32](#_Toc388446613)

[3.2.1 Feature 1: Create New Reservations 33](#_Toc388446614)

[3.2.5 Feature 5: Create a Room 35](#_Toc388446615)

[3.2.6 Feature 6: Deleting a Room 37](#_Toc388446616)

[3.2.7 Feature 7: Modifying a Room 39](#_Toc388446617)

[3.2.8 Feature 8: Create a Room Type 41](#_Toc388446618)

[3.2.9 Feature 9: Delete a Room Type 43](#_Toc388446619)

[3.2.10 Feature 10: Modify a Room Type 45](#_Toc388446620)

[3.2.11 Feature 11: View Reports and Transactions 47](#_Toc388446621)

[3.2.12 Feature 12: Create a new account 49](#_Toc388446622)

[3.2.13 Feature 13: Delete an existing account 51](#_Toc388446623)

[3.2.14 Feature 14: Edit an existing account 53](#_Toc388446624)

[3.2.15 Feature 15: Manage Daily Rates 55](#_Toc388446625)

[3.2.16 Feature 16: Add a Discount Code 57](#_Toc388446626)

[3.2.17 Feature 17: Delete an Existing Discount Code 59](#_Toc388446627)

[3.2.19 Feature 19: Acquire and save credit card information 60](#_Toc388446628)

[3.3 Use Cases 62](#_Toc388446629)

[3.3.1 Use Case: Create a Reservation: Input Name 63](#_Toc388446630)

[3.3.2 Use Case: Create a Reservation: Input Address 64](#_Toc388446631)

[3.3.3 Use Case: Create a Reservation: Input City 65](#_Toc388446632)

[3.3.4 Use Case: Create a Reservation: Input State 66](#_Toc388446633)

[3.3.5 Use Case: Create a Reservation: Input Country 67](#_Toc388446634)

[3.3.6 Use Case: Create a Reservation: Input Phone Number 68](#_Toc388446635)

[3.3.7 Use Case: Create a Reservation: Input Email 69](#_Toc388446636)

[3.3.8 Use Case: Create a Reservation: Input Number of Adults 70](#_Toc388446637)

[3.3.9 Use Case: Create a Reservation: Input Number of Children 71](#_Toc388446638)

[3.3.10 Use Case: Create a Reservation: Input Arrival Date 72](#_Toc388446639)

[3.3.11 Use Case: Create a Reservation: Input Departure Date 73](#_Toc388446640)

[3.3.12 Use Case: Create a Reservation: Credit Card Type 74](#_Toc388446641)

[3.3.13 Use Case: Create a Reservation: Credit Card Number 75](#_Toc388446642)

[3.3.14 Use Case: Create a Reservation: Credit Card Expiration 76](#_Toc388446643)

[3.3.15 Use Case: Create a Reservation: Select Type of Room 77](#_Toc388446644)

[3.3.16 Use Case: Create a Reservation: Submit Reservation 78](#_Toc388446645)

[3.3.17 Use Case: Modify a Reservation: Modify Name 79](#_Toc388446646)

[3.3.18 Use Case: Modify a Reservation: Modify Address 80](#_Toc388446647)

[3.3.19 Use Case: Modify a Reservation: Modify City 81](#_Toc388446648)

[3.3.20 Use Case: Modify a Reservation: Modify State 82](#_Toc388446649)

[3.3.21 Use Case: Modify a Reservation: Modify Country 83](#_Toc388446650)

[3.3.22 Use Case: Modify a Reservation: Modify Phone Number 84](#_Toc388446651)

[3.3.23 Use Case: Modify a Reservation: Modify Email 85](#_Toc388446652)

[3.3.24 Use Case: Modify a Reservation: Modify Number of Adults 86](#_Toc388446653)

[3.3.25 Use Case: Modify a Reservation: Modify Number of Children 87](#_Toc388446654)

[3.3.26 Use Case: Modify a Reservation: Modify Arrival Date 88](#_Toc388446655)

[3.3.27 Use Case: Modify a Reservation: Modify Departure Date 89](#_Toc388446656)

[3.3.28 Use Case: Modify a Reservation: Credit Card Type 90](#_Toc388446657)

[3.3.30 Use Case: Modify a Reservation: Credit Card Expiration 91](#_Toc388446658)

[3.3.31 Use Case: Modify a Reservation: Room Type 92](#_Toc388446659)

[3.3.32 Use Case: Modify a Reservation: Submit Modifications 93](#_Toc388446660)

[3.3.33 Use Case: Add New Room Type: Input Name of Type 94](#_Toc388446661)

[3.3.34 Use Case: Add New Room Type: Input Room Type Code 95](#_Toc388446662)

[3.3.35 Use Case: Add New Room Type: Input Number of Beds 96](#_Toc388446663)

[3.3.36 Use Case: Add New Room Type: Input Rack Rate 97](#_Toc388446664)

[3.3.37 Use Case: Add New Room Type: Save New Room Type 98](#_Toc388446665)

[3.3.38 Use Case: Modify Room Type: Modify Name of Type 99](#_Toc388446666)

[3.3.39 Use Case: Modify Room Type: Modify Room Type Code 100](#_Toc388446667)

[3.3.40 Use Case: Modify Room Type: Modify Number of Beds 101](#_Toc388446668)

[3.3.41 Use Case: Modify Room Type: Modify Rack Rate 102](#_Toc388446669)

[3.3.42 Use Case: Modify Room Type: Save Modified Room Type 103](#_Toc388446670)

[3.3.43 Use Case: Delete Room Type: Select Room Type 104](#_Toc388446671)

[3.3.44 Use Case: Delete Room Type: Erase Selected Room Type 105](#_Toc388446672)

[3.3.45 Use Case: Add Room: Room Number 106](#_Toc388446673)

[3.3.46 Use Case: Add Room: Room Type 107](#_Toc388446674)

[3.3.51 Use Case: Add Account: Input Name 108](#_Toc388446675)

[3.3.52 Use Case: Add Account: Input Age 109](#_Toc388446676)

[3.3.53 Use Case: Add Account: Input Gender 110](#_Toc388446677)

[3.3.54 Use Case: Add Account: Input Position 111](#_Toc388446678)

[3.3.55 Use Case: Add Account: Input Access Level 112](#_Toc388446679)

[3.3.56 Use Case: Add Account: Input Phone Number 113](#_Toc388446680)

[3.3.57 Use Case: Add Account: Input Emergency Contact 114](#_Toc388446681)

[3.3.58 Use Case: Add Account: Input Email Address 115](#_Toc388446682)

[3.3.59 Use Case: Add Account: Input Address 116](#_Toc388446683)

[3.3.60 Use Case: Add Account: Input City 117](#_Toc388446684)

[3.3.61 Use Case: Add Account: Input Zip Code 118](#_Toc388446685)

[3.3.62 Use Case: Add Account: Input Country 119](#_Toc388446686)

[3.3.63 Use Case: Add Account: Input Additional Details 120](#_Toc388446687)

[3.3.64 Use Case: Modify Account: Modify Name 121](#_Toc388446688)

[3.3.65 Use Case: Modify Account: Modify Age 122](#_Toc388446689)

[3.3.66 Use Case: Modify Account: Modify Gender 123](#_Toc388446690)

[3.3.67 Use Case: Modify Account: Modify Position 124](#_Toc388446691)

[3.3.68 Use Case: Modify Account: Modify Account Access 125](#_Toc388446692)

[3.3.69 Use Case: Modify Account: Modify Phone Number 126](#_Toc388446693)

[3.3.70 Use Case: Modify Account: Modify Emergency Contact 127](#_Toc388446694)

[3.3.71 Use Case: Modify Account: Modify Email Address 128](#_Toc388446695)

[3.3.72 Use Case: Modify Account: Modify Address 129](#_Toc388446696)

[3.3.73 Use Case: Modify Account: Modify City 130](#_Toc388446697)

[3.3.74 Use Case: Modify Account: Modify Zip Code 131](#_Toc388446698)

[3.3.75 Use Case: Modify Account: Modify Country 132](#_Toc388446699)

[3.3.76 Use Case: Modify Account: Modify Additional Details 133](#_Toc388446700)

[3.3.77 Use Case: Delete Account: Select Account 134](#_Toc388446701)

[3.3.78 Use Case: Delete Account: Erase Selection 135](#_Toc388446702)

[3.3.79 Use Case: Change Daily Rate: View Rates (Open up the window) 136](#_Toc388446703)

[3.3.80 Use Case: Change Daily Rate: Select Start Date 137](#_Toc388446704)

[3.3.81 Use Case: Change Daily Rate: Select End Date 138](#_Toc388446705)

[3.3.82 Use Case: Change Daily Rate: Enter difference from Rack Rate 139](#_Toc388446706)

[3.3.83 Use Case: Change Daily Rate: Finalize Modification 140](#_Toc388446707)

[3.3.84 Use Case: Add Discount: Open Add Discount Window 141](#_Toc388446708)

[3.3.85 Use Case: Add Discount: Input Discount Code 142](#_Toc388446709)

[3.3.86 Use Case: Add Discount: Input Discount Rate 143](#_Toc388446710)

[3.3.87 Use Case: Add Discount: Input Discount Description 144](#_Toc388446711)

[3.3.88 Use Case: Add Discount: Finalize Discount 145](#_Toc388446712)

[3.3.89 Use Case: Modify Discount: Select Discount to Modify 146](#_Toc388446713)

[3.3.90 Use Case: Modify Discount: Open Selected Discount to Modify 147](#_Toc388446714)

[3.3.91 Use Case: Modify Discount: Modify Attributes 148](#_Toc388446715)

[3.3.92 Use Case: Modify Discount: Finalize Modifications 149](#_Toc388446716)

[3.3.93 Use Case: Delete Discount: Select Discount to Delete 150](#_Toc388446717)

[3.3.94 Use Case: Delete Discount: Review Deletion 151](#_Toc388446718)

[3.3.95 Use Case: Delete Discount: Confirm Deletion 152](#_Toc388446719)

[3.3.96 Use Case: Using the Interface: Login Menu 153](#_Toc388446720)

[3.3.97 Use Case: Using the Interface: Enter Username 154](#_Toc388446721)

[3.3.98 Use Case: Using the Interface: Enter Password 155](#_Toc388446722)

[3.3.99 Use Case: Using the Interface: Finalize Login 156](#_Toc388446723)

[3.4 Objects 157](#_Toc388446724)

[3.4.1 Object: Hotel 158](#_Toc388446725)

[3.4.2 Object: RoomType 165](#_Toc388446726)

[3.4.3 Object: Room 166](#_Toc388446727)

[3.4.4 Object: Accounts 168](#_Toc388446728)

[3.4.5 Object: AccountsEmployee 170](#_Toc388446729)

[3.4.6 Object: AccountsManager 171](#_Toc388446730)

[3.4.7 Object: Reservation 172](#_Toc388446731)

[3.4.8 Object GeneralInfo 174](#_Toc388446732)

[3.4.9 Object: RequiredInfo 177](#_Toc388446733)

[3.4.10 Object: ExtraFeatures 179](#_Toc388446734)

[3.4.11 Object : Discount 181](#_Toc388446735)

[3.4.12 Object: CreditAmericanExpress 183](#_Toc388446736)

[3.4.13 Object: CreditDiscover 186](#_Toc388446737)

[3.4.14 Object: CreditMastercard 189](#_Toc388446738)

[3.4.15 Object: CreditVisa 192](#_Toc388446739)

[3.4.16 Object: Date 195](#_Toc388446740)

[3.4.17 Object Relationships 197](#_Toc388446741)

[3.5 Non-functional Requirements 198](#_Toc388446742)

[3.5.1 Performance 198](#_Toc388446743)

[3.5.2 Reliability 198](#_Toc388446744)

[3.5.3 Availability 198](#_Toc388446745)

[3.5.4 Security 199](#_Toc388446746)

[3.5.5 Maintainability 199](#_Toc388446747)

[3.5.6 Portability 199](#_Toc388446748)

[3.6 Inverse Functions 200](#_Toc388446749)

[3.6.1 No Log of Transactions 200](#_Toc388446750)

[3.6.2 Data Backups must be Done Manually 200](#_Toc388446751)

[3.6.3 Lacks Options for Housekeeping 201](#_Toc388446752)

[3.7 Design Constraints 202](#_Toc388446753)

[3.8 Logical Database Requirements 203](#_Toc388446754)

[3.9 Other Requirements 204](#_Toc388446755)

[4 Analysis Models 205](#_Toc388446756)

[4.1 Sequence Diagrams 205](#_Toc388446757)

[4.2 Data Flow Diagrams 206](#_Toc388446758)

[4.2.1 DFD 1: Interactions between Reservation Components 206](#_Toc388446759)

[4.2.2 DFD 2: Assigning a Room Number inside Reservation 207](#_Toc388446760)

[4.2.3 DFD 3: Adding a New Hotel Employee Inside Accounts 208](#_Toc388446761)

[4.2.4 DFD 4: Updating Rates through Rate Management Settings 209](#_Toc388446762)

[4.3 State-Transitional Diagrams 209](#_Toc388446763)

[5 Graphical User Interface Menus 210](#_Toc388446764)

[5.1 Login Screen 210](#_Toc388446765)

[5.2 Settings: Inventory 211](#_Toc388446766)

[5.3 Settings: Repots: Accounting 212](#_Toc388446767)

[5.4 Settings: Reports: Accounts 213](#_Toc388446768)

[5.5 Settings: Reports: Rates 214](#_Toc388446769)

[5.6 Settings: Reports: Reservations 215](#_Toc388446770)

[5.7 Settings: Accounts 216](#_Toc388446771)

[5.8 Room Calendar 217](#_Toc388446772)

[6 Change Management Process 218](#_Toc388446773)

[Document Approvals 218](#_Toc388446774)

# 1. Introduction

Vacations, businesses, and family accommodations have been increasingly changing over the past decade. As a result, tourism has become an industry across the globe with increasing demand. The rate of growth has turned many hotelier’s over to automated processes for efficiency, accuracy, and reliability.

As hotels become more fascinating and elegant so too would their resources and ability to innovate. As there is much room for improvement in the chef’s kitchen, table service, cash register, management control such as the flow of control, and much more.

The local travel and tourism industry is crucial to the economy. To sustain the growth and development of the industry, we have to win the confidence of the international community by not only continuously improving professionalism and quality of service but also enhancing the standard of the integrity of the industry. Hoteliers, as one of the important industry practitioners, are expected to take the lead in making improvements to their software. One of the key dilemmas Hoteliers face today is allocating the complexity of the business as it upholds competitive standards.

In hotel operations, procurement of goods and services is the most vulnerable area which could lead to malpractice because hotels spend substantial amounts on goods (such as food and beverage, utensils, toiletries etc.) and services (such as cleaning and security services, group insurance services etc.). Besides making sure that purchases are value for money, it is important for the hotel management to establish a fair and competitive procurement system with sufficient safeguards. These systems are increasingly improved by newer technology maintenances with more accurate and concurrent software.

Sales and marketing activities in a hotel mainly refer to marketing and reservation of rooms and offering of special rates and complimentary rooms to customers, such as travel agencies, airline companies and corporate clients, etc. Such data is very extensive and requires reluctant filing to be finished.

Our team has worked elegantly and sophisticatedly to meet all these requirements and more. We have been at our toes developing new and innovative policies for richer documents structures. We ensure exotic visual components for user attentiveness. We provide fine attention to details in all of our expressions to unite goods and services with unmatched quality.

We hope you will be as pleased with your investment, as we are very fulfilled to present our most sophisticated platform.

-Program ‘til It Hurtz

## 1.1 Purpose

The purpose of the Software Requirements Specifications (SRS) is to thoroughly explain the requirements and functionality necessary to develop the Hotel Management System (HMS). This Software Requirements System will completely break down every aspect of the hotel management system software to make it understandable for both the software engineers and the hotel end users. For the software engineers, the Software Requirements Specifications will provide a set of guidelines to assure that the software is being developed to meet the hotel end users’ expectations. For the hotel end user, the Software Requirements Specifications will provide a checklist to guarantee the final product of the software will completely meet all the expectations that were agreed upon. Once the SRS is agreed upon by both parties, the software will be developed directly to what is included in this document. The Software Requirements Specifications may also be used as a template for further implementation of future versions of the current software.

## 1.2 Scope

The software product that will be produced is a Hotel Management System (HMS). This Hotel Management System will create a virtual environment to help manage and organize the hotel’s daily functions more efficiently. The software will primarily be operated by the hotel’s management and employees. Management will be allowed access to more functionality under the administrator account, while the employees will have a more basic employee account with less functionality.

The software will be split into several small subsystems. The reservation subsystem will be the main subsystem included in the software that will allow for the staff to create, delete, and manage reservations made at the hotel. The reservation subsystem will require for the end user to input the client’s general contact information along with their credit card information in order to make a reservation. The software will contain another subsystem that produces a grid that contains the hotel rooms. This grid will allow the end user to see the status of a certain room and the reservations on those rooms. There will also be a search subsystem that will be incorporated with the reservation subsystem. This will allow the end user to input parameters to search through the reservations database for a specific reservation. Other subsystems that this software will contain are employees, room service, and restaurant. The Employees subsystem will allow management to keep record of all the employees personal information, work schedule, and payroll. The Room Service subsystem will handle all charges booked to the particular room. The Restaurant subsystem will deal with all charges pertaining to the hotel restaurant.

## 1.3 Definitions

ADR (average daily rate)

*-* A measure of the average rate paid for rooms sold, calculated by dividing room revenue by rooms sold. (ADR = room revenue / rooms sold)

All-suite

- All guest rental units consist of one or more bedrooms and may include a separate living area. Many suites contain kitchenettes or mini-refrigerators. Suite hotels often have no integrated dining facilities available; however, many offer complimentary breakfast.

Availability

- The readiness of a system to deliver services when requested.

BAR

– Best Available Rate

Boutique

- These are hotels that appeal to their guests because of their unusual amenity and room configurations. They are normally independent and smaller than 200 rooms with a high rack rate.

C++

- An object-oriented programming language that is a superset of C.

Class Diagram

- A UML diagram types that shows the object classes in a system and their relationships.

Comp

– A complimentary stay.

Conference

- Lodging hotels that place major focus on conference operations. Hotels must meet guidelines of the International Association of Conference Centers.

Contract rooms

- Contract rooms are occupied at rates stipulated by contracts–such as for airline crews and permanent guests.

Controller

– The financial director of the hotel.

Convention

- Hotels with a minimum of 300 rooms and large meeting facilities (minimum of 20,000 Square feet) and not part of the conference center group.

Dependability

- The dependability of a system is an aggregate property that takes into account the

Dependability requirement

- A system requirement that is included to help achieve the required dependability for a system. Nonfunctional dependability requirements specify dependability attribute values system’s safety, reliability, availability, security, and other attributes.

Design pattern

- A well-tried solution to a common problem that captures experience and good practice in a form that can be reused.

Destination resorts

- These are hotels that appeal to the leisure traveler, are located in resort markets and are considered a destination in themselves. They have extensive amenities, multiple pools, multiple restaurants and normally a beautiful physical hotel. If a hotel has the word “resort’ in its name and is part of a luxury or upper-upscale chain, it is automatically considered a destination resort.

End User

- the client using the software.

Extended stay

- Extended stay hotels focus on attracting hotel guests for extended periods of time, typically more than 5 consecutive nights. These hotels quote weekly rates.

Food & beverage (F&B) revenue

- Revenues derived from the sale of food (including coffee, milk, tea and soft drinks), beverages (including, beer, wine and liquors), banquet beverages and other F&B sources. Other F&B sources include meeting room rentals, audio-visual equipment rentals, cover or service charges or other revenues within the food-and-beverage department (includes banquet services charges).

Front of House

- Reception, concierge and porters (see Back of the House)

Front Office

- reception

Full-service hotel

- Full-service hotels are generally mid-price, upscale or luxury hotels with a restaurant, lounge facilities and meeting space as well as minimum service levels often including bell service and room service. These hotels report food-and-beverage revenue.

GM

– General manager.

Group rooms

- Group rooms are sold simultaneously in blocks of a minimum of ten rooms or more (e.g. group tours, domestic and international groups, association, convention and corporate groups).

HMS

- Hotel Management System

Hotel types

- Hotel classifications are driven primarily by building structure and secondarily by service level. Chain management has provided us with hotel type classifications for a significant number of locations.

Hotel/Motel

- A standard hotel or motel operation.

House Count

- The occupancy of a hotel at any given time of day

House Manager

- Usually the number two to the General Manager, often responsible for an individual hotel where the General Manager covers more than one.

Information hiding

- Using programming language constructs to conceal the representation of data structures and to control external access to these structures.

Interface

- A specification of the attributes and operations associated with a software component. The interface is used as the means of accessing the component’s functionality

Limited-service hotel

 - Limited-service hotels have rooms-only operations, (i.e. without food-and-beverage service) or offer a bedroom and bathroom for the night, but very few other services and amenities. These hotels are often in the budget or economy group and do not report food-and-beverage revenue.

Maintenance

- The process of making changes to a system after it has been put into operation.

Mystery Guest

– Spottier

N/S

– No show.

Object-oriented (OO) development

- An approach to software development where the fundamental abstractions in the

Occupancy

 - Occupancy is the percentage of available rooms that were sold during a specified period of time. Occupancy is calculated by dividing the number of rooms sold by rooms available.

Operator

- The hotel management company which manages a hotel under a management agreement.

Owner

- The owner of a hotel business which is counterparty to the Operator under a management contract.

Paid out

- An amount included on a guest bill for an expense which the hotel has paid for the guest and passes on, for example taxis or theatre tickets

POS

– Point of Sale

Quality plan

- A plan that defines the quality processes and procedures that should be used. This involves selecting and instantiating standards for products and processes and defining the system quality attributes that are most important quantitatively as a probability of failure on demand or as the rate of occurrence of failure.

Reliability

- The ability of a system to deliver services as specified. Reliability can be specified

Requirement, functional

- A statement of some function or feature that should be implemented in a system.

Requirement, non-functional

- A statement of a constraint or expected behavior that applies to a system. This constraint may refer to the emergent properties of the software that is being developed or to the development process.

Reservation

– A room that is retained for future use.

Reservation

- The functionality handling the reservations clients make at the hotel.

Room night

- A room being occupied by one or more guests for one night.

Room revenue

 - Total room revenue generated from the sale or rental of rooms.

Rooms available (room supply)

 - The number of rooms in a hotel or set of hotels multiplied by the number of days in a specified time period.

Rooms sold (room demand)

 - The number of rooms sold in a specified time period (excludes complimentary rooms).

Sample

 - The number of hotels and rooms from which data is received.

Size

 - Based on physical guestroom count of the hotel.

Ski

- Guests have easy access to ski slopes.

Spa

- Hotel must have designated spa facility and offer treatments. Offering a sauna or hot tub/whirlpool would not qualify.

SRS

- Software Requirements Specifications

Structured method

- A method of software design that defines the system models that should be developed, the rules and guidelines that should apply to these models, and a process to be followed in developing the design.

Total revenue

 - Revenue from all hotel operations, including rooms sold, F&B, parking, laundry, phone, miscellaneous, etc.

Transient rooms

 - Include rooms occupied by those with reservations at rack, corporate, corporate negotiated, package, government, or foreign traveler rates. Also includes occupied rooms booked via third party web sites (exception: simultaneous bookings of 10 or more rooms which should be defined as group).

Unified Modeling Language (UML)

- A graphical language used in object-oriented development that includes several types of system models that provide different views of a system. The UML has become a de facto standard for object-oriented modeling.

User interface design

- The process of designing the way in which system users can access system functionality, and the way that information produced by the system is displayed.

Validation

- The process of checking that a system meets the needs and expectations of the

Verification

- The process of checking that a system meets its specification

Workflow

- A detailed definition of a business process that is intended to accomplish a certain task. The workflow is usually expressed graphically and shows the individual

# 2 General Description

General Descriptions are started with minimal details of each and every single aspect of the program. These general descriptions are subdivided into multiple subgroups.

## 2.1 Product Perspective

This section discusses and assesses what the total outlook will have. Simple examples include the look and feel of certain widgets and overall system management.

### 2.1.1 System Interfaces

System Interfaces defines the overall user interface as a whole. It explains what is included in the program. A frontier across which two, the management and time, can operate at optimal levels. An interface might be a hardware connector used to link to other devices, or it might be a convention used to allow communication between two software systems. This system replaces ordinary custom need of a balance book with automatic application of software. Items that are of basic use include: time and occupancy management, purchase transactions, report documentation, and strategic design.

### 2.1.2 Interfaces

Interfaces are varied by tabs. Each tab has its own main user specification. The reservation tab quickly manages guest maintenance and reservation addition. The reports tab acquires account information and arranges data to friendly easy to read information. It compiles all hotel accommodation and rates into summary notation. Objects include: accommodation suites, discounted rates, account of added packages, and fees. The accounts tab lists account details between users and guest relations. It handles information such as their names, access operations, resource handling, and profile information.

### 2.1.3 Hardware Interfaces

Hardware Interfaces defines the system requirements. Note: software will run only on Windows PC versions, not included for Mac IOS versions, preferably versions on Windows 7 or Windows 8. Updates will be added until May 21, 2014. Any further updates require purchase of ongoing development.

### 2.1.4 Software Interfaces

Button – activates the action to which the button describes. For example if button was “create”, it would make what its intended use was.

Tab – activates the switching from panel to panel. In this example it opens another panel to the reports tab where you can view all balance query and fees.

Time Bar – works the timeframe for a reservation. If double clicked, it will open a new window for the current editable reservation. Panning it left or right moves the bar to a selected date. Dragging the ends extends the duration of the reservation.

Time Tab – separates the reservation time zones. These tabs are separated by weeks. Each one expanding 7 days in the reservation calendar. The clicked week will be lit and centralized.

Room Tab – separates each reservation by room number. Each room number is separated by a lit room type. Room types will vary by the room properties in the manager’s settings.

Info Bar – contains easy to access information about the reservation highlighted or clicked. The info bar consists of basic ergonomic information which is always in use. Examples: name, reservation type, etc.

Search Bar – organizes information for easy discovery. The search could be used to find a certain reservation, date, name, account profile, or availability.

Add Reservation – makes it easy to quick start the reservation software. It is the start button for the main screen. Activating this grants an employee or manager the ability to continue with the further hotel registrations.

Settings – opens up variable integration. This robust system allows an administrator to remove or altar certain fields in the software. Examples: changing rates, room types, etc.

### 2.1.5 Communication Interfaces

Communication Interfaces make up the different parts of the program communicate with each other, however, through the operating system. When the software is used on a computer, data is stored on the computer and not shared on any other computer. The computers running the software are not required to be networked together, since the program is not able to support a networked database of information. Most of the communication is done through user and object tab interface which communicate from an arithmetic expression, into a casual interpretation. These interfaces comprise between timed functions (availability) with respect to number ratio (occupancy). Most of which make up the reservation calendar:

Bar lite – separates information between party size and special guests.

Lit buttons – create an interface between active and inactive icons.

Lit room buttons – separates information by room type.

Search bar – limits access difficulties.

Tabs – quick changing operations.

Info bar – lays out important common information.

Bar – communicates time in reference to date

Account integration – conveys a sense of security

### 2.1.6 Memory Constraints

This software will run on a Windows installation on a PC. The amount of memory that the initial software uses will be minimal, however, this amount will grow as more entries are populated into the database. The more reservations that there are will result in more memory being used.

Since the reservations grid actively displays current reservations over a period of time, this may become memory-intensive and cause the system to lag. The application will need to strike a balance between displaying all the relevant information in a view and being responsive and smooth in performance. The software should not cause any lag when there is a lot of data to be stored.

Other memory constraints consist of unlimited employees and overlapping reservations. First, the number of employee accounts is indefinitely infinite. Not removing most or all unemployed accounts will result to memory overflow. Therefore it is strictly advised to remove any unused accounts in the employment area. Second, overlapping reservations is not advised. Overlapping consistently could cause an overflow of “unknowing” reservations. Which entails memory allocation. If enough overlapping reservations are created, there may be a memory overflow.

### 2.1.7 Operations

Operations of the software involve interaction between guest and hotel and transfer of information to the software. Guests will communicate information across to an operative who which will then operate the software. The software will calculate and organize a purchase plan which will be communicated to the operative. The operative will have an ongoing relation with the guest to convey the right purchase decisions. Once a decision has been made, the software will conclude final results. The final results will report and asses the arrangements where then the software works a set of algorithms to organize and save the information into its database for updated usage.

### 2.1.8 Site Adaptation Requirements

Site Adaptation Requirements include basic computer knowledge and operation. The software mainly consists of textbox interfaces, clicker events, and scroll panning. The interaction of all the gadgets with these events is further explained throughout this document.

## 2.2 System Environment

There are several sets of interactions that exist in the system environment that involves the use of the Hotel Management Software.

Interactions between Employees and Customers

This demand driven interaction where customers approach a hotel receptionist seeking hotel rooms is the beginning of the set of interactions that involves the HMS.

Interactions between Employees and HMS

In order to meet the customers’ demands, an employee uses the reservation management feature in HMS to reserve rooms. Using the HSM, employees will be able to securely block off rooms with appropriate guest information. Employees can also check guests in and out of the HMS to keep track of the customers whose expectations have already been met and the customers who are still expecting service.

Interactions between Managers and HSM

Aside from the direct interactions between managers and employees and between managers and customers, managers may meet the customers’ needs by manipulating the inventory of the hotel and managing the rates to set them up for customers and employees. In addition, the account management feature of the HMS allows the manager to manage employee information for documentation.

Interactions between Managers and Employees

Managers are required to handle the accounts of different employees. Whenever an employee is recruited or let go, the manager is required to create an account for that employee so that the employees may use HMS to meet the business’s needs. If a worker is promoted from employee to manager, the employee must contact a manager in order to gain the necessary access to manage the property.

## 2.3 Product Functions

There are four primary functions of this software of which stakeholders can take advantage to manage the hotel.

Managing Reservations  
Employees will be able to meet the customer’s need for hotel rooms by creating and maintaining reservations for the requested types of hotel rooms with the features desired by the customers. When a hotel guests approaches a front desk clerk to reserve a room, this HMS makes it easy for the employee to note down the necessary information and block off the designated room for that particular guest.

Managing Inventory

Managers will be able to manipulate the types of hotel rooms available and the amenities provided in specific rooms through the inventory management feature provided in the software. This feature allows the hotel manager to set up for employees and customers the rooms that are available to be reserved or out of order. It also allows the manager to manipulate the rooming list in cases such as where the room has been extended to have additional amenities, combined with another room to increase spacing and furnishing, renovated, or demolished etc. Below are some diagrams of a few possible scenarios for the utmost use of inventory management.

Managing Rates

Managers will be able to manipulate the prices for different types of hotel rooms in order to maximize the business revenue. This allows a manager to increase prices during busier days, such as an event that attracts tourists, and decrease prices during slower days to control the amount of sales and profit for their hotel business. These predictions are made by the managers themselves and this HMS does not assist with any of these predictions. This HMS gives a manager full control

Analyzing Past Data

Managers will be able to use the reporting features to track their progress. This will give managers data (such as the amount of rooms sold in a certain month or number of reservations cancelled due to poor service etc.) with which to analyze their financial and reputational status, understand positive and negative consequences of actions to enhance their business and prevent harmful outcomes, create future goals etc.

## 2.4 User Characteristics

There are two classes of users of this hotel management software who are expected to attain two different levels of expertise in various subjects. The types of users and their required characteristics are listed below.

Employees

Employees are expected to have general knowledge of using a computer, including using a mouse and a keyboard to navigate through the interface, along with a basic understanding of hotel room reservation process. They must obtain this set of knowledge through training, previous work experience, or personal interest prior to working with this Hotel Management Software for the software is built upon these fundamental assumptions.

Managers

Managers are expected to also have a general knowledge of using a computer in order to use this software. In addition, *Managers* are required to have an understanding of inventory and rates management in order to get the most use out of the inventory and rates management features of the software.

## 2.5 Constraints

The only constraint that is put upon the Hotel Management System is the range of time for which reservations will be allowed. To save memory and allow better efficiently the Hotel Management System will only save three years’ worth of reservations. After that it will be saved into a database to be loaded at a later date if ever needed.

## 2.6 Assumptions and Dependencies

Below is a list of assumptions for designing HMS.

* HMS is not required to save any generated reports.
* HMS is not required to print directly to a printer.
* HMS will be run in a PC with Windows OS environment.

# 3 Specific Requirements

This section contains all the software requirements in enough detail for designers to design a system that satisfies all of those requirements, and testers to test that the system satisfies those requirements. To increase readability and use of comprehension, the requirements are divided into categories based on the subgroups in which they belong. These sections are labeled as following:

* External Interface Requirements
* Features (Functional Requirements)
* Use Cases
* Objects
* Non-Functional Requirements
* Inverse Requirements
* Design Constraints
* Logical Database Requirements
* Other Requirements

## 3.2 Features

Features (aka Functional Requirements) define the fundamental actions that must take place in HMS in accepting and processing the inputs and in processing and generating the outputs. They are functionalities that the system cannot exist without as opposed to functionalities that are designed to ease the software’s use. They are critical in the sense that the software cannot operate without these functionalities or the software is useless to the client once the product is finished for the software would lack the important functions required to for its intended use.

This section is an accumulation of all the functional requirements that exist in HMS.

<Insert relevant picture or diagram to illustrate the meaning of features>

### 3.2.1 Feature 1: Create New Reservations

This section discusses the methods and interactions concerning the creation of new reservations.

#### 3.2.1.1 Introduction

Reservations will be utilized by the staff (both management and employee) to record and organize the client’s personal information, credit card information, and reservation information. The client will need to provide their name, address, and credit card information in order to meet the minimum requirements to reserve a hotel room. Reservations will also have information regarding the type of room the client has requested and the client’s duration at the hotel. If the client has any special requests the staff will also have an option to enter these as well. Once the reservation has been made, it will be added to a linked list of reservations for future use.

#### 3.2.1.2 Inputs

There are several inputs required in order to create a reservation. The general information of a hotel guest is contained within an object called GeneralInfo, which includes the following basic information: name of the guest, the address, city, zip-code and country of the guest’s home, a contact number, and an email address. The necessary information required to allow a reservation to block off a room is contained in another object called RequiredInfo, which includes the following information: the arrival and departure dates, credit card information, a type of room. If the hotel guest has any special requests or if the hotel guests have a reservation for a big party, an object called ExtraFeatures.

#### 3.2.1.3 Processing

In order to create an object Reservation, the user will need to input the proper information to create objects of the type GeneralInfo, RequiredInfo, and ExtraFeatures. Once the Reservation object has been created, it will be added to a linked list of Reservations for future handling.

#### 3.2.1.4 Outputs

Once the Reservation object has been created, the software will print the reservation to the screen so the end user can confirm the information is correct. The staff will also be able to access all of the Reservation’s attributes from the reservations tab in the main software screen. The user will be able to see multiple reservations on a grid or choose to view an individual reservation at a time. Multiple reservations will have the viewing option of being sorted by date, type of room, or price.

#### 3.2.1.5 Error Handling

The software will verify that all the proper information was provided to create the GeneralInfo and RequiredInfo objects. If the proper information is not provided, the software will refrain from creating a Reservation object and prompt the user to do so. The software will also verify that the room being reserved isn’t already reserved for the specific date and time requested. If the user attempts to reserve a room that is already reserved for the date and time requested, the software will alert the user that the reservation cannot be created.

### 3.2.5 Feature 5: Create a Room

In order to increase flexibility of the hotel managers to manage their rooms, HMS allows users to add, delete, or modify rooms on their own, without having to contact the HMS developer or technical support. This section will discuss everything that is involved in creating a room.

#### 3.2.5.1 Introduction

There are many instances when rooms are added to hotels. If a property owner decides to purchase neighboring land and extend their facility to the new land, a great deal of new rooms becomes available for sale. These new rooms may be added through the Inventory Management portion of HMS.

#### 3.2.5.2 Inputs

There are several inputs needed to add a room to the rooming list. First, since each room must have its own unique room number, the user must input a room number that does not already exist in the rooming list (the rooming list is stored as a vector of Room pointers inside the Hotel object). Next, for each room, the user must specify the room type that room will represent, which the user is able to pick from a drop down menu (these room types are predefined by the user and manipulation of room types is discussed in sections 3.2.8 to 3.2.10). The user may or may not specify any features that are to be added to the room.

#### 3.2.5.3 Processing

Assuming the user has inputted all of the above information with a room number that does not already exist, the information is placed in a dynamically created Room object. A pointer to that Room object is then placed inside the vector of Room pointers inside the Hotel object. This will officially include the new user defined room in the hotel rooming list.

#### 3.2.5.4 Outputs

After the room has been added to the rooming list, this new room will become available to reserve. It will appear in the list of available rooms when creating a reservation and the features unique to this specific room will appear in the features window upon reservation creation or modification. The new room will also be added to the room calendar so reservations for that room can appear in the GUI.

#### 3.2.5.5 Error Handling

When the user attempts to input a room number that already exists in the rooming list, HMS will present an error message preventing the user from creating the room and hold off processing the information until a unique room number is given. This prevents the user from having duplicate rooms and, in some cases, accidentally inputting the wrong room number.

### 3.2.6 Feature 6: Deleting a Room

This section will discuss everything that is involved in deleting a room.

#### 3.2.6.1 Introduction

There are various reasons a manager may need to delete a room from HMS. If a section of the hotel is demolished and rebuilt for a different purpose, if a portion of the land is sold or given away, or if multiple rooms are combined into one, it becomes necessary to delete the room from HMS since it no longer exists in the inventory and is unavailable for sale.

#### 3.2.6.2 Inputs

All the rooms in the rooming list, which is stored as a vector of Room pointers inside of the Hotel object, are differentiated primarily through its room number (the room number is stored as a member variable inside individual Room objects). For this reason, the only input required to delete a room is the room number which is to be provided by the user. The user may select the room to delete through a drop down menu that numerically lists all the rooms at the hotel, or simply typing in the room number.

#### 3.2.6.3 Processing

HMS must first search through the rooming list in order to find the user provided room number. A room with the specified room number must exist in order for processing to take place. If it is not found, the situation is handled through the error handler (see section 3.2.6.5). If the room number is found, the compiler then proceeds to delete the pointer to the Room object from the vector of Room pointers in Hotel object.

#### 3.2.6.4 Outputs

Once the pointer has been deleted, it is no longer accessible by other objects. Therefore, it will not appear in the drop down menu of available rooms in when creating or modifying a reservation, nor will it appear in any menu that lists any rooms. The room will also be removed the main calendar and will no longer be viewable by the user. Any reservation originally intending to use that room will lose its value and will become a reservation with an unassigned room number. Any information about past reservations that used the deleted room, however, will continue to be displayed in the Reports sections of HMS.

#### 3.2.6.5 Error Handling

If the user provided room number is not found when searched through the rooming list, the processing part of this feature will not go through. Instead, the user will presented an error message indicating that no such room exists with the provided room number. No further action will be taken and all objects and functions of HMS will remain unaffected.

### 3.2.7 Feature 7: Modifying a Room

This section will discuss everything that is involved in modifying a room.

#### 3.2.7.1 Introduction

In case of renovation, upgrades, addition of features etc., a hotel room goes through a various changes. In order to advertise the new changes and to keep things consistent, a manager needs to be able to apply the same changes in the HMS. That is what the room modification feature of HMS allows a user to do.

#### 3.2.7.2 Inputs

The input for room modification is dependent on the information that the user would like to update. This input information may be a single, multiple, or all of the instances of the following: room number, room type, and room features. Thus, a user may modify anywhere between one to all of variables in a room by choosing the input to provide.

#### 3.2.7.3 Processing

Once the user finalizes all the inputs and orders HMS to update the room information, the system checks for any issues that may arise from the update, such as having duplicate room numbers or assigning nonexistent room types. These issues are managed by the error handler (see section 3.2.7.5). If no issues are likely to occur, the system will replace all of variables in the Room object with the user provided inputs.

#### 3.2.7.4 Outputs

After the update, the previous room information will no longer be available. Instead, all of the old information will be replaced by the new information, including the new set of modified room features included in the instance of the Features object inside of Room object. If the room type is changed, this room will present itself as the new room type to all of the objects that communicate with that Room object. If the room number is changed, the main calendar will also modify itself to match the values of the modified room. In case of any issues prior to the processing stage, the error handler will display a list of errors in a single window so the user may fix the necessary information.

#### 3.2.7.5 Error Handling

There are a few cases where updating information will not be allowed. First, no duplicates for room numbers will be allowed. Therefore, if the user attempts to modify the room number with one that already exists, the processing will not take place. Instead, the user will be prompted with an error message indicating that another room with that room number already exists. Second, if the user attempts to enter a nonexistent room type as an input, the processing will not take place and the user will be prompted with another error message.

### 3.2.8 Feature 8: Create a Room Type

This section will discuss everything that is involved in creating a room type.

#### 3.2.8.1 Introduction

In many occasions, owners of a hotel will introduce a new type of room that was not previously available. This is usually done to extend the options for a hotel customers, thus, allowing them to pick from a bigger variety to make choices that better fit their needs. When a room type is added inside a hotel, it must also be added in HMS to allow employees to offer that option to the customer. This is where creating a room type inside HMS becomes useful for managers.

#### 3.2.8.2 Inputs

All the room types, which are stored as a vector of pointers to RoomType objects, are differentiated through the name of the room type. Thus, in order to create a new room type, the manager must choose a room type name that is different from any other room type. This allows all the room types to be unique by preventing duplicates and prevents confusing in both code and HMS. Upon room type creation, the user must also provide the following information: A code to identify similarities between different room types, the number of beds provided in the specific types of rooms, the maximum number of adults allowed inside the room per reservation, the maximum number of children allowed per reservation, a small description that may help the employee to better understand the aspects of the specific room type, a base price for the room type (the rack rate) and a color for highlighting purposes.

#### 3.2.8.3 Processing

Once all the input has been placed, the processing part will insure the uniqueness of the room type name. If a duplicate occurs, the error handler will present the appropriate message to ask the user to provide a unique name instead. If the name is unique, the processing section will dynamically create a new object of RoomType and place one additional pointer pointing to the new object into the vector of room type pointers.

#### 3.2.8.4 Outputs

The new room type will now be available in several places. First, it will appear as an option upon creating a reservation for this new room type; it will appear in the drop down menu. Second, the new room type will be available in the rates section for rate modification. Third, it will also appear as an option upon a creation of a room.

#### 3.2.8.5 Error Handling

If the name provided by the user is not unique, then the user will be prompted an error message saying “The given room type name already exists. Please choose a different room type name.” Other than this, there should not be any possible error that might occur during the room type creation process.

### 3.2.9 Feature 9: Delete a Room Type

This section will discuss everything that is involved in room type deletion.

#### 3.2.9.1 Introduction

Due to business reasons, where a certain room type does not have a high enough demand to have a reason to be available, owners of the hotel may decide to remove a specific type of room. Whether the existing rooms of that type are to be converted into other room types or removed from the inventory is up to the general managers. However, in order to keep information consistent with HMS, whenever a hotel room type is erased from the business, it must also be removed from HMS to prevent confusion between employees and customers who may think that the room type is available when it has been removed.

#### 3.2.9.2 Inputs

There is only one input necessary for the deletion of a room type. In order to delete a room type, the user must choose the type to delete from a drop down list of all existing room types. Once the room type has been selected, and the users’ decision was confirmed, the room type object will be handled by the processing section. There is no possibility of false inputs because the user must choose from a drop down list which limits the user’s options to only inputs that are not likely to cause errors.

#### 3.2.9.3 Processing

Once the input is complete, the compiler will pick the room type object from the vector of existing room type pointers and delete that previously dynamically created object. The compiler will also search through all the rooms that have that room type assigned and will assign the default room type to those rooms. Those rooms that have been using that room type must have their types reassigned in order to be used again. Any reservation that was pointing to a room type of that deleted room type will have its room type assigned to the default one and will lose its room number value. This allows the manager to go through the rooms and reservations to make any necessary changes caused from the room type deletion.

#### 3.2.9.4 Outputs

The user will be prompted a message immediately after the room type deletion to confirm that it had been successful. However, there are several output consequences due to a room type deletion. Because HMS will no longer have that room type available in the inventory, it will no longer appear as an option upon room creation or modification. Also, the deleted room type will stop appearing in reservations as options. The rooms that had the deleted room type previously assigned to it will change their color due to the reassignment of room types to default type. Finally, the deleted room type will no longer be available for modification or deletion in the inventory management settings.

#### 3.2.9.5 Error Handling

Deleting room type must be handled carefully for it may have very undesirable consequences to the previously organized HMS program. Any reservation or room must be reassigned to default in order to prevent the program from crashing or causing false output. Close attention must be paid to insure that all objects pointing to that room type are reassigned before the object is deleted.

### 3.2.10 Feature 10: Modify a Room Type

This section will discuss everything that is involved in editing any specific room type.

#### 3.2.10.1 Introduction

Due to changes in customer demand and severe revenue expectations, there are a few occasions when a hotel owner decides to change aspects of a room type. It may be reducing the number of beds from 3 to 2 or allowing a higher number of adults into the room or any other changes relevant to room type variables. On these occasions, the object representing that room type in HMS must also be altered to maintain consistency between the HMS inventory and physical inventory. This prevents confusion between employees and customers and saves business hassles in future.

#### 3.2.10.2 Inputs

The user must first choose the room type to modify from a drop down menu of existing room types. The additional inputs that are required for a room type modification are the same inputs that were used upon its creation. The number of inputs to modify may vary from one to all. However, the compiler will not proceed to the processing section of modification until at least one input has been changed. If nothing has been changed, the output section will prompt the appropriate message to ask the user to provide at least one modified input. The inputs that may be changed are as follows: the name of the room type, the code that is used to identify similarities between room types, the number of beds provided that specific room type, the maximum number of adults allowed, the maximum number of children allowed, the small description that is intended to help the employee understand the aspects of that room type, the base prices for the room type, and the color for highlighting purposes.

#### 3.2.10.3 Processing

Once all the inputs have been identified, the compiler should then proceed to replace all the variables of that room type object. Any modified input provided by the user will replace the original value that existed in that room type. ­­­If the name is changed, the compiler must also go through every reservation object that uses that specific room type and change its type to match the name of the new name for the room type. For rooms, no modification is necessary since the objects are directly linked through a pointer. Thus, any change in a room type object will automatically change the names in the room objects.

#### 3.2.10.4 Output

There are several outputs that come as a result of room modification. First, immediately after the modification is complete, the user will prompted with a message confirming that room type had been successfully modified. If any errors occur, the appropriate error message, provided by the error handler, will appear as a result and the user will be notified that the room modification process had failed. Also, if either the room name or the room code is changed, all the objects that have been using the room type object will now contain the new name or new code. If the maximum number of adults or children allowed in that specific room type has been changed, then any warning messages that originally appeared due to having too many adults or children will output the messages after doing the proper calculations with the new value.

#### 3.2.10.5 Error Handling

Because room types are classified using their names, any modification to room type may cause issues with the reservation objects that are trying to call it using the old name. To prevent this, the compiler must go through each reservation and change the string in that reservation to match it with the modified name. Aside from this issue, there should not be anything else that is likely to cause errors.

### 3.2.11 Feature 11: View Reports and Transactions

This section will discuss everything involved in reports and transactions

#### 3.2.11.1 Introduction

In order to analyze progress, revenue, and to make future plans, the managers need data to track past progress. Tracking past progress allows them to understand and make necessary changes to their business approach in order to maximize profit and minimize complaints to preserve reputation. In addition, in a case of issues with finances or customers complaints, the manager is able to look back in the reports section to gather evidence and data to help make the appropriate decision. Also, reports can be used to settle disputes between employees or managers and resolve internal conflicts.

#### 3.2.11.2 Inputs

The only inputs required by the user are the dates for which they would like to view the reports. All other necessary inputs are saved in the database and will be retrieved as needed by the processing section.

#### 3.2.11.3 Processing

There are several classes of data that will be needed in order to display them in the output. They are listed as follows: Accounting, Rates, Reservations, Availability, Transactions, Credit Cards, and Accounts. All values relevant to the reports will be retrieved from the database by the system. This section will provide information regarding all the transactions that had taken place during the specified period of time, how many rooms of each room type are available on the specified days, the balances that originated in those days, the transactions taken place on those days, the reservations created and cancelled during those days, the accounts created or modified in those days, and rates for the different room types for those days. All of this information will be calculated properly and sent to the output for the manager to view and analyze.

#### 3.2.11.4 Output

The retrieved data will be displayed in an organized manner in the reports window. Things should be displayed in categories when they are easier to view and understand. Reservations will be categorized as “created” or “cancelled.” Accounts will be categorized as “created,” “deleted,” or “modified.” Availability will be organized based on dates. Credit cards, rates changes and accounting will all be categorized using dates.

#### 3.2.11.5 Error Handling

If the user inputs invalid dates for which there is no data, an error message will be prompted to the user notifying that invalid input. Because a great deal of data is retrieved from the database, careful attention should be paid to insure that no retrieved data causes issues.

### 3.2.12 Feature 12: Create a new account

#### 3.2.12.1 Introduction

In order to keep track of all the employees of the hotel and allow them to use the software, each employee must create an account in the HMS. Every employee will have a username and password to log in to the HMS. They will also be assigned an account type depending on their position at the hotel. The type may be manager, who is able to access the settings and reservations menu, or employee, who is only able to access the reservations menus. Since this feature is a part of the settings menu and requires increased access, only a user who is a manager is able to access this.

#### 3.2.12.2 Inputs

All the accounts in the list of accounts, which is stored as a vector of Account pointers inside of the Hotel object, are differentiated primarily through its usernames (the username is stored as a member variable inside individual Account objects). Therefore, all the attributes of the particular account will be considered an input. These include the username, password, and account type of the user. The username and password will be entered via text fields, and the account type will be selected via a drop-down list consisting of Manager or Employee.

#### 3.2.12.3 Processing

When the user has inputted the username, password, and chosen the account type of the user, the system will create an Account object. It will create either an AccountManager or an AccountEmployee object, depending on the account type chosen. A pointer to the Account object will be placed inside a vector of pointers inside the Hotel object.

#### 3.2.12.4 Outputs

When the Accounts object has been created, it will be viewable in the settings menu of the GUI. The manager will be able to view the information on the particular account including username, password, and account type.

#### 3.2.12.5 Error Handling

The system will ensure that an account type is selected when a new account is created. If the manager proceeds to create the account without first choosing an account type, an error message will be prompted. This will ensure that every account is assigned an account type. The system will also need to make sure that the username is not a duplicate. If it is already present in the system, the system will prompt an error message. This is to make sure that every username in the database is unique.

### 3.2.13 Feature 13: Delete an existing account

#### 3.2.13.1 Introduction

When managing a hotel, there will be occasions when managers will need to remove an existing account. Reasons for this may include employee layoff, termination, death, or other circumstances. When the account is removed, the system should no longer have any records of the username, password, or account type of the particular account. Since this feature is a part of the settings menu and requires increased access, only a user who is a manager is able to access this.

#### 3.2.13.2 Inputs

When deleting an account, the manager will be able to select the particular employee to be deleted via the GUI. The manager will need to only choose employee from the list of accounts, and then click the delete employee button to complete the task. This will ensure that only existing employees may be deleted.

#### 3.2.13.3 Processing

The HMS will search the vector of Accounts for the username of the particular employee object. Once found, the system will remove the particular Account object and then decrease the total amount of employees by one.

#### 3.2.13.4 Outputs

The system will have no records of the particular Account once it has been deleted. The list of Accounts will show one less employee, and that particular Account is no longer searchable. The GUI will also reflect this change.

#### 3.2.13.5 Error Handling

If there is no username matching the one that is being searched, the user account does not exist. The system will then prompt a message to the user after searching through the entire vector of Accounts.

### 3.2.14 Feature 14: Edit an existing account

#### 3.2.14.1 Introduction

One essential feature in managing employee accounts is the ability to change their information. The manager is able to change a particular account’s username and password. The manager is also able to change the account type of the particular account. This will come in handy when employees are promoted or demoted.

#### 3.2.14.2 Inputs

The input for updating an existing account may include the particular account’s username, password, or account type. This information is accessed and changed via the through text fields in the settings view of the GUI. The username and passwords are edited through text fields, while the account type may be changed via a drop-down list.

Only one employee may be edited at a time.

#### 3.2.14.3 Processing

When the input is recorded in the system, the HMS will search for the particular Account to be edited and change its attributes accordingly. The system will only be able to change the attributes of one object at a time. In the event that the account type of a particular is changed, it will create a new Account object (either accountEmployee or accountManager) and copy the username and password data. It will then delete the object with the previous account type. This allows for the system to easily allocate the new access level for the changed Account.

#### 3.2.14.4 Outputs

After the updates to the particular account are made, the changes will be displayed accordingly in the GUI and any reports that may be generated. The old information no longer exists as well. If all values are changed, then the system will display the new values for all of the attributes for the particular object. If only some values were edited, then the object will display a combination of both old and new values.

#### 3.2.14.5 Error Handling

When changing Account information, there are a few scenarios that might cause the HMS to work improperly. One scenario is if the new username that is chosen is already present in the system. In this case, the HMS will want to avoid having duplicate accounts, and therefore prompt an error message to the user. Another case is if the new password matches the old password. The HMS will also prompt an error message to the user, encouraging him/her to change the password to something not already used.

### 3.2.15 Feature 15: Manage Daily Rates

This section discusses everything that is involved in managing daily rates.

#### 3.2.15.1 Introduction

Because demand goes up and down in different periods of time, it is important for managers to change the rates based on the demand on that day to maximize profits overall. During special events, for instance, tourists may be travelling from overseas areas to attend in which case the prices can be driven up. During slower times when no one is looking for a hotel room, customers are less likely to come to a hotel to rent a room, thus, forcing the price to go down. In order to accomplish maximum profit, the rates management portion of HMS allows the managers to change room prices based on dates and room types.

#### 3.2.15.2 Inputs

In order to modify rates, the manager must provide a date range in which the modification will take place. Thus, first two inputs needed are the start date and the end date. The final input is a double value indicating the difference from the base rate, which may be positive or negative. The summation of the base price and the difference will be the final price that is applied to the room types for the selected dates. All three inputs are necessary in order to proceed with the change.

#### 3.2.15.3 Processing

After the inputs have been provided, the system will then proceed to add up the difference and the base rate. For each room type and date, the new value will added into the database. These changes are not reversible since the system overrides the previous value with the new value. The overwritten prices will be used by the system from then on.

#### 3.2.15.4 Output

Immediately after the modification, a message will appear notifying the user whether or not the changes were successfully saved. After the processing is finished, the new prices will now appear as the rates while making a reservation. Though existing reservations with set prices will not change, modification to a reservation should change price to the modified price if the dates have been changed. The rates management window will also present the new rates instead of the old rates.

#### 3.2.15.5 Error Handling

The user must provide all three inputs (the start date, the end date, and the price difference) in order to make a modification in the daily rates. If not all fields are filled in before the system is asked to move to the processing part, the user will be prompted with an error message asking to provide all the information in order to make changes. The original prices will be saved in a temporary data structure during the overwriting process in case an error causes the process to be halted or crash. In this case, the original prices are restored by copying them back from the temporary backup structure to prevent the database from having corrupt data.

### 3.2.16 Feature 16: Add a Discount Code

This section will discuss everything that is involved in creating discount objects.

#### 3.2.16.1 Introduction

Using discounts is a common way of promoting sales. In order to give reasons to customers to come to one hotel over another, managers may decide to promote different types of discounts. If a new promotion comes in or if a specific promotion expires, the manager will need to add or delete them based on their availability. HMS will allow the manager to create, delete, and edit different discount objects which may be applied to the prices upon reservation creation or modification.

#### 3.2.16.2 Inputs

There will be three inputs required from the user to create a discount object. First, the user must provide a code for the discount. This code will be used to differentiate between the discount objects. In order to prevent duplicates in the set of discount objects, the code provided by the user must be unique. In other words, it may code name data may not overlap with an existing discount object. If the user provides the duplicate input, the system will pass it to the error handler to take the appropriate action. Second, the user must input a small description explaining what the discount is given for. For instance, the description may be “Senior Citizen” for a discount designed for people who are 65 years or older. Finally, the user must input the percentage of discount to be given for that new promotion. This value may not go below 0 and must not exceed 100.

#### 3.2.16.3 Processing

Once the necessary inputs have been placed and they are verified by the system as valid, the system then proceeds to dynamically create that discount object and place a pointer pointing to that object inside the vector of all discounts. In addition, the information is saved in the database to be retrieved in the future by other objects that may need it.

#### 3.2.16.4 Outputs

Immediately after the object creation, the user will be notified through a message whether the object creation was successful or not. If successful, the new discount code will then appear as an option when creating or modifying any reservation.

#### 3.2.16.5 Error Handling

The error handler will only handle cases where the inputs are invalid. As long as the inputs are valid, the system will always be able to successfully create the discount object. If there another discount object with the same code already exists, the user will be prompted with an error message asking for a unique code. If any of the other inputs are invalid or left blank, an error message will ask the user to do fix the inputs.

### 3.2.17 Feature 17: Delete an Existing Discount Code

#### 3.2.17.1 Introduction

Occasionally managers will need to delete an existing discount code because the discount code has reached the expiration date or management decides to discontinue the discount code. If the manager doesn’t delete a discount it will remain in the discount and guests will remain with the option to use the discount code. When the discount code is deleted, the Discount object and all of its attributes will no longer be in the system and will be removed from the GUI.

#### 3.2.17.2 Inputs

In order to delete a discount, the user will need to select the desired discount to be deleted from the GUI. Once the desired discount is selected, the user then must press the Delete button to remove the Discount from the system.

#### 3.2.17.3 Processing

The software will search through a vector of discounts for the desired discount to be deleted. Once the correct discount has been found, the system will then proceed to delete the discount and rearrange the vector. Once the discount has been deleted, the system and GUI will no longer have access to the object and its attributes.

#### 3.2.17.4 Outputs

Deleting a discount will not provide any output. Once the discount is deleted, the HMS will no longer have any information regarding the discount. The discount is removed from the vector and the GUI.

#### 3.2.17.5 Error Handling

If the system searches through the vector and it doesn’t find the desired discount, it will return an error message explaining that it cannot delete a discount that doesn’t exist.

### 3.2.19 Feature 19: Acquire and save credit card information

#### 3.1.19.1 Introduction

The management of credit card information is crucial to any hotel operation. It is often used as the primary source of identification and of payment for hotel guests. The HMS will need to accurately appropriate a credit card object to each hotel guest. This will include allocating data for the credit card’s account number, expiration date, verification code, cardholder name, billing information, and credit card type. In addition, the HMS will check to see if the credit card is verified, by comparing the billing information of the credit card to the address of the guest.

#### 3.1.19.2 Inputs

There are many inputs, which the HMS shall record through text fields on the reservation screen. There will be a section devoted to credit card information for each guest. In this section will include text fields for credit card number, expiration date, and verification code, billing address, cardholder name, and a drop-down menu for the credit card type. Both employees and managers of the hotel may access and change this information.

#### 3.1.19.3 Processing

When a new reservation is created and the credit card fields have been populated, the system will create the appropriate credit card object depending on what credit card type was selected. For example, if the reservation is made with a Visa credit card, a CreditVisa object will be created, its attributes populated, and attached to the particular reservation. In the case that the reservation decides to use a different credit card than one already present in the system, the HMS will delete the old object and create a new appropriate object to associate with the guest.

#### 3.1.19.4 Outputs

The system will display the credit card information on the credit card section in the reservations view of the GUI. The text fields will display the correct credit card number, expiration date, verification code, billing address, and first and last name of the particular user’s credit card. It will also display the credit card type of the particular card (American Express, Discover, Mastercard, or Visa).

#### 3.1.19.5 Error Handling

Different credit card types have different nuances. For example, American Express cards have only 15 digit card numbers, while Discover, Mastercard, and Visa have 16 digit card numbers. When the particular credit card type is chosen, the system will check the number of digits of the credit card number to see if it matches the template. If it doesn’t, an error message is prompted. The HMS shall also if the credit card billing information matches that of the reservation. If it doesn’t an error message will be prompted. This ensures that guests are not using stolen credit cards and adds a layer of security to the program.

## 3.3 Use Cases

The purpose of including use cases is to describe to a designer the goal of the end-user of a program and to list the necessary steps required by the end-users to accomplish that goal. This allows designers to see the program through the stakeholders’ point of view and understand their motive behind taking the listed actions.

This section includes a series of use-cases to better allow the designer to understand a stakeholder’s expectations and to satisfy the specific requirements detailed in this SRS.

### 3.3.1 Use Case: Create a Reservation: Input Name

|  |
| --- |
| Section 3.3.1. Figure 1: Use case for inputting guest name into reservation. |
| |  |  | | --- | --- | | HMS: Inputting guest name upon creating reservation | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | In order to allow customers to reserve certain room, employees and managers must specify the name of the person under which the reservation is to be made | | Data | Name of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to access the reservation information. | |

### 3.3.2 Use Case: Create a Reservation: Input Address

|  |
| --- |
| Section 3.3.2. Figure 1: Use case for inputting the guest street address into reservation. |
| |  |  | | --- | --- | | HMS: Inputting guest street address upon creating reservation | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | In order to allow customers to reserve a certain room, employees and managers must specify the street address of the person under which the reservation is to be made | | Data | Street address of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to access the reservation information. | |

### 3.3.3 Use Case: Create a Reservation: Input City

|  |
| --- |
| Section 3.3.3. Figure 1: Use case for inputting the guest city of residence into reservation. |
| |  |  | | --- | --- | | HMS: Inputting guest city of residence upon creating reservation | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | In order to allow customers to reserve a certain room, employees and managers must specify the city of residence of the person under which the reservation is to be made | | Data | City of residence of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to access the reservation information. | |

### 3.3.4 Use Case: Create a Reservation: Input State

|  |
| --- |
| Section 3.3.4. Figure 1: Use case for inputting the guest state of residence into reservation. |
| |  |  | | --- | --- | | HMS: Inputting guest state of residence upon creating reservation | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | In order to allow customers to reserve a certain room, employees and managers must specify the state of residence of the person under which the reservation is to be made | | Data | State of residence of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to access the reservation information. | |

### 3.3.5 Use Case: Create a Reservation: Input Country

|  |
| --- |
| Section 3.3.5. Figure 1: Use case for inputting the guest country of residence into reservation. |
| |  |  | | --- | --- | | HMS: Inputting guest country of residence upon creating reservation | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | In order to allow customers to reserve a certain room, employees and managers must specify the country of residence of the person under which the reservation is to be made | | Data | Country of residence of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to access the reservation information. | |

### 3.3.6 Use Case: Create a Reservation: Input Phone Number

|  |
| --- |
| Section 3.3.6. Figure 1: Use case for inputting the guest phone number into reservation. |
| |  |  | | --- | --- | | HMS: Inputting guest phone number upon creating reservation | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | In order to allow customers to reserve a certain room, employees and managers must specify the phone number of the person under which the reservation is to be made | | Data | Phone number of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to access the reservation information. | |

### 3.3.7 Use Case: Create a Reservation: Input Email

|  |
| --- |
| Section 3.3.7. Figure 1: Use case for inputting the guest email into reservation. |
| |  |  | | --- | --- | | HMS: Inputting guest email upon creating reservation | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | In order to allow customers to reserve a certain room, employees and managers must specify the email of the person under which the reservation is to be made | | Data | Email of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to access the reservation information. | |

### 3.3.8 Use Case: Create a Reservation: Input Number of Adults

|  |
| --- |
| Section 3.3.8. Figure 1: Use case for inputting the amount of adult guests into reservation. |
| |  |  | | --- | --- | | HMS: Inputting amount of adult guests upon creating reservation | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | In order to allow customers to reserve a certain room, employees and managers must specify the amount of adult guests under which the reservation is to be made | | Data | Amount of adult guests | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to access the reservation information. | |

### 3.3.9 Use Case: Create a Reservation: Input Number of Children

|  |
| --- |
| Section 3.3.9. Figure 1: Use case for inputting the amount of children guests into reservation. |
| |  |  | | --- | --- | | HMS: Inputting amount of children guests upon creating reservation | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | In order to allow customers to reserve a certain room, employees and managers must specify the amount of children guests under which the reservation is to be made | | Data | Amount of children guests | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to access the reservation information. | |

### 3.3.10 Use Case: Create a Reservation: Input Arrival Date

|  |
| --- |
| Section 3.3.10. Figure 1: Use case for inputting the guest arrival date into reservation. |
| |  |  | | --- | --- | | HMS: Inputting guest arrival date upon creating reservation | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | In order to allow customers to reserve a certain room, employees and managers must specify the arrival date of the guest under which the reservation is to be made | | Data | Guest arrival date | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to access the reservation information. | |

### 3.3.11 Use Case: Create a Reservation: Input Departure Date

|  |
| --- |
| Section 3.3.11. Figure 1: Use case for inputting the guest departure date into reservation. |
| |  |  | | --- | --- | | HMS: Inputting guest departure date upon creating reservation | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | In order to allow customers to reserve a certain room, employees and managers must specify the departure date of the guest under which the reservation is to be made | | Data | Guest departure date | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to access the reservation information. | |

### 3.3.12 Use Case: Create a Reservation: Credit Card Type

|  |
| --- |
| Section 3.3.12. Figure 1: Use case for reserving accounts using a credit card type. |
| |  |  | | --- | --- | | HMS: Input a Credit Card Type | | | Actors | Manager/Employee, Hotel Management System (HMS) Database | | Description | Upon applying and reserving room at a hotel, the vendor may input the credit card type into the database for payment processing | | Data | Credit Card Type | | Stimulus | Mouse click by the manager or employee | | Response | None | | Comments | Only managers or employees may access options for payments. | |

### 3.3.13 Use Case: Create a Reservation: Credit Card Number

|  |
| --- |
| Section 3.3.13. Figure 1: Use case for reserving accounts using a credit card number. |
| |  |  | | --- | --- | | HMS: Input a Credit Card Number | | | Actors | Manager/Employee, Hotel Management System (HMS) Database | | Description | Upon applying and reserving room at a hotel, the vendor may input the credit card number into the database for payment processing | | Data | Credit Card Number | | Stimulus | Mouse click by the manager or employee | | Response | None | | Comments | Only managers or employees may access options for payments. | |

### 3.3.14 Use Case: Create a Reservation: Credit Card Expiration

|  |
| --- |
| Section 3.3.14. Figure 1: Use case for reserving accounts using a credit card expiration. |
| |  |  | | --- | --- | | HMS: Input a Credit Card Expiration | | | Actors | Manager/Employee, Hotel Management System (HMS) Database | | Description | Upon applying and reserving room at a hotel, the vendor may input the credit card expiration into the database for payment processing | | Data | Credit Card Expiration | | Stimulus | Mouse click by the manager or employee | | Response | None | | Comments | Only managers or employees may access options for payments. | |

### 3.3.15 Use Case: Create a Reservation: Select Type of Room

|  |
| --- |
| Section 3.3.15. Figure 1: Use case for reserving accounts by type of room. |
| |  |  | | --- | --- | | HMS: Type of Room | | | Actors | Manager/Employee, Hotel Management System (HMS) Database | | Description | Upon applying and reserving room at a hotel, the vendor may input the preferred type of room into the database based on availability. | | Data | Type of Room | | Stimulus | Mouse click by the manager or employee | | Response | None | | Comments | Only managers or employees may access options for room types. | |

### 3.3.16 Use Case: Create a Reservation: Submit Reservation

|  |
| --- |
| Section 3.3.16. Figure 1: Use case for reserving accounts by submission. |
| |  |  | | --- | --- | | HMS: Submit Reservation | | | Actors | Manager/Employee, Hotel Management System (HMS) Database | | Description | Upon applying at a hotel, the vendor may submit the preferred reservation into the database based on availability. | | Data | Submit Reservation | | Stimulus | Mouse click by the manager or employee | | Response | None | | Comments | Only managers or employees may access the submissions. | |

### 3.3.17 Use Case: Modify a Reservation: Modify Name

|  |
| --- |
| Section 3.3.17. Figure 1: Use case for modifying guest name in reservation. |
| |  |  | | --- | --- | | HMS: Modifying guest name after the reservation has been created | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the name of the guest on the currently existing reservation if a mistake was made. | | Data | Name of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the reservation information. | |

### 3.3.18 Use Case: Modify a Reservation: Modify Address

|  |
| --- |
| Section 3.3.18. Figure 1: Use case for modifying the guest street address in reservation. |
| |  |  | | --- | --- | | HMS: Modifying guest street address after the reservation has been created | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the street address of the guest on the currently existing reservation if a mistake was made. | | Data | Street address of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the reservation information. | |

### 3.3.19 Use Case: Modify a Reservation: Modify City

|  |
| --- |
| Section 3.3.19. Figure 1: Use case for modifying the guest city of residence in reservation. |
| |  |  | | --- | --- | | HMS: Modifying guest city of residence after the reservation has been created | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the city of residence of the guest on the currently existing reservation if a mistake was made. | | Data | City of residence of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the reservation information. | |

### 3.3.20 Use Case: Modify a Reservation: Modify State

|  |
| --- |
| Section 3.3.20. Figure 1: Use case for modifying the guest state of residence in reservation. |
| |  |  | | --- | --- | | HMS: Modifying guest state of residence after the reservation has been created | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the state of residence of the guest on the currently existing reservation if a mistake was made. | | Data | State of residence of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the reservation information. | |

### 3.3.21 Use Case: Modify a Reservation: Modify Country

|  |
| --- |
| Section 3.3.21. Figure 1: Use case for modifying the guest country of residence in reservation. |
| |  |  | | --- | --- | | HMS: Modifying guest country of residence after the reservation has been created | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the country of residence of the guest on the currently existing reservation if a mistake was made. | | Data | Country of residence of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the reservation information. | |

### 3.3.22 Use Case: Modify a Reservation: Modify Phone Number

|  |
| --- |
| Section 3.3.22. Figure 1: Use case for modifying the guest phone number in reservation. |
| |  |  | | --- | --- | | HMS: Modifying guest phone number after the reservation has been created | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the phone number of the guest on the currently existing reservation if a mistake was made. | | Data | Phone number of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the reservation information. | |

### 3.3.23 Use Case: Modify a Reservation: Modify Email

|  |
| --- |
| Section 3.3.23. Figure 1: Use case for modifying the guest email in reservation. |
| |  |  | | --- | --- | | HMS: Modifying guest email after the reservation has been created | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the email of the guest on the currently existing reservation if a mistake was made. | | Data | Email of the hotel guest | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the reservation information. | |

### 3.3.24 Use Case: Modify a Reservation: Modify Number of Adults

|  |
| --- |
| Section 3.3.24. Figure 1: Use case for modifying the amount of adult guests in reservation. |
| |  |  | | --- | --- | | HMS: Modifying amount of adult guests after the reservation has been created | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the amount of adult guests on the currently existing reservation if a mistake was made. | | Data | Amount of adult guests | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the reservation information. | |

### 3.3.25 Use Case: Modify a Reservation: Modify Number of Children

|  |
| --- |
| Section 3.3.25. Figure 1: Use case for modifying the amount of children guests in reservation. |
| |  |  | | --- | --- | | HMS: Modifying amount of children guests after the reservation has been created | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the amount of children guests on the currently existing reservation if a mistake was made. | | Data | Amount of children guests | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the reservation information. | |

### 3.3.26 Use Case: Modify a Reservation: Modify Arrival Date

|  |
| --- |
| Section 3.3.26. Figure 1: Use case for modifying the guest arrival date in reservation. |
| |  |  | | --- | --- | | HMS: Modifying guest arrival date after the reservation has been created | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the arrival date of the guest on the currently existing reservation if a mistake was made. | | Data | Guest arrival date | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the reservation information. | |

### 3.3.27 Use Case: Modify a Reservation: Modify Departure Date

|  |
| --- |
| Section 3.3.27. Figure 1: Use case for modifying the guest’s departure date in a reservation. |
| |  |  | | --- | --- | | HMS: Modifying a guest’s departure date | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the departure date of guests on the currently existing reservation if a mistake was made. | | Data | Guest departure date | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the departure information. | |

### 3.3.28 Use Case: Modify a Reservation: Credit Card Type

|  |
| --- |
| Section 3.3.28. Figure 1: Use case for modifying the guest’s credit card type in a reservation. |
| |  |  | | --- | --- | | HMS: Modifying a guest’s credit card type | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the credit card type of guests on the currently existing reservation if a mistake was made. | | Data | Guest credit card type | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the credit card type information. | |

### 3.3.30 Use Case: Modify a Reservation: Credit Card Expiration

|  |
| --- |
| Section 3.3.30. Figure 1: Use case for modifying the guest’s credit card expiration in a reservation. |
| |  |  | | --- | --- | | HMS: Modifying a guest’s credit card expiration | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the credit card expiration  of guests on the currently existing reservation if a mistake was made. | | Data | Guest credit card expiration | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the credit card expiration information. | |

### 3.3.31 Use Case: Modify a Reservation: Room Type

|  |
| --- |
| Section 3.3.31. Figure 1: Use case for modifying the guest’s room type in a reservation. |
| |  |  | | --- | --- | | HMS: Modifying a guest’s room type | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may modify the room type of a guest on the currently existing reservation if a mistake was made. | | Data | Guest’s room type | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Both the managers and the employees have permissions to modify the room type information. | |

### 3.3.32 Use Case: Modify a Reservation: Submit Modifications

|  |
| --- |
| Section 3.3.32. Figure 1: Use case for submitting the guest’s modified information in a reservation. |
| |  |  | | --- | --- | | HMS: Submit modifications | | | Actors | Employee, Manager, Hotel Management System (HMS) Database | | Description | Employees and managers may submit the modification of a guest on the currently existing reservation if a mistake was made. | | Data | Guest’s modified information | | Stimulus | Button event by user | | Response | None | | Comments | Both the managers and the employees have permissions to submit the modified information. | |

### 3.3.33 Use Case: Add New Room Type: Input Name of Type

|  |
| --- |
| Section 3.3.33. Figure 1: Use case for inputting name for room type. |
| |  |  | | --- | --- | | HMS: Inputting name upon room type creation | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon creating a new room type for a hotel, the manager may input the room type into the database to make it available to be reserved or to be selected from the user interface. | | Data | The name for the room type (Standard, Deluxe etc.) | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. | |

### 3.3.34 Use Case: Add New Room Type: Input Room Type Code

|  |
| --- |
| Section 3.3.34. Figure 1: Use case for inputting code for room type. |
| |  |  | | --- | --- | | HMS: Inputting code upon room type creation | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon creating a new room type for a hotel, the manager may input the room type into the database to make it available to be reserved or to be selected from the user interface. A code is short identifier used to refer to specific room types. These are the codes that are displayed from the drop down menu when creating a reservation or creating a new room | | Data | The desired code for the room type (May be up to 5 characters long) | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. | |

### 3.3.35 Use Case: Add New Room Type: Input Number of Beds

|  |
| --- |
| Section 3.3.35. Figure 1: Use case for inputting the number of beds for room type. |
| |  |  | | --- | --- | | HMS: Inputting number of beds upon room type creation | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon creating a new room type for a hotel, the manager may input the room type into the database to make it available to be reserved or to be selected from the user interface. The number of beds in a room type is directly related to the number of beds available at the hotel for that room type | | Data | The number of beds for the room type | | Stimulus | The number issued by the manager | | Response | None | | Comments | Only managers may access options for room type creation. This act is not permitted from an employee’s account. | |

### 3.3.36 Use Case: Add New Room Type: Input Rack Rate

|  |
| --- |
| Section 3.3.36. Figure 1: Use case for inputting the rack rate for room type. |
| |  |  | | --- | --- | | HMS: Inputting rack rate upon room type creation | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon creating a new room type for a hotel, the manager may input the room type into the database to make it available to be reserved or to be selected from the user interface. The rack rate is the starting price for a room of that type | | Data | The base price for the room type | | Stimulus | The number issued by the manager | | Response | None | | Comments | Only managers may access options for room type creation. This act is not permitted from an employee’s account. The base rate will vary depending on the date. | |

### 3.3.37 Use Case: Add New Room Type: Save New Room Type

|  |
| --- |
| Section 3.3.37. Figure 1: Use case for finalizing the process of adding new room type. |
| |  |  | | --- | --- | | HMS: Saving new room type | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon creating a new room type for a hotel, the manager may input the room type into the database to make it available to be reserved or to be selected from the user interface | | Data | The finalized room type object | | Stimulus | A mouse click issued by the manager | | Response | None | | Comments | Only managers may access options for room type creation. This act is not permitted from an employee’s account. | |

### 3.3.38 Use Case: Modify Room Type: Modify Name of Type

|  |
| --- |
| Section 3.3.38. Figure 1: Use case for modifying name of room type. |
| |  |  | | --- | --- | | HMS: Modify name of an existing room type | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | When management decides to change features of a certain room type in a hotel, officials also want to update the information in HMS to keep the information consistent and avoid confusion between a customer’s expectation and the actual product. | | Data | The new name for the room type | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may to access options for room type modification. This act is not permitted from an employee’s account. | |

### 3.3.39 Use Case: Modify Room Type: Modify Room Type Code

|  |
| --- |
| Section 3.3.39. Figure 1: Use case for modifying the code of existing room type. |
| |  |  | | --- | --- | | HMS: Modify code of an existing room type | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | When management decides to change features of a certain room type in a hotel, officials also want to update the information in HMS to keep the information consistent and avoid confusion between a customer’s expectation and the actual product. The code name is a string of characters used to refer to a specific room type. | | Data | The new code for the room type | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may to access options for room type modification. This act is not permitted from an employee’s account. | |

### 3.3.40 Use Case: Modify Room Type: Modify Number of Beds

|  |
| --- |
| Section 3.3.40. Figure 1: Use case for modifying the number of beds of room type. |
| |  |  | | --- | --- | | HMS: Modify number of beds of an existing room type | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | When management decides to change features of a certain room type in a hotel, officials also want to update the information in HMS to keep the information consistent and avoid confusion between a customer’s expectation and the actual product. Maintaining consistency with the number of beds that exist in a room type is especially important in maintain good customer service | | Data | The new number of beds for the room type | | Stimulus | Typed number issued by the manager | | Response | None | | Comments | Only managers may to access options for room type modification. This act is not permitted from an employee’s account. | |

### 3.3.41 Use Case: Modify Room Type: Modify Rack Rate

|  |
| --- |
| Section 3.3.41. Figure 1: Use case for inputting the new rack rate for a specific room type. |
| |  |  | | --- | --- | | HMS: Inputting new rack rate in room type modification | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | When modifying a room type, the base rate for that room type may be directly manipulated to affect it in the rates management window | | Data | The new rate for the specific room type | | Stimulus | The price value issued by the manager | | Response | None | | Comments | Only managers may access options for room type modification. This act is not permitted from an employee’s account. | |

### 3.3.42 Use Case: Modify Room Type: Save Modified Room Type

|  |
| --- |
| Section 3.3.42. Figure 1: Use case for finalizing modification of room type. |
| |  |  | | --- | --- | | HMS: Saving room type modifications | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | When management decides to change features of a certain room type in a hotel, officials also want to update the information in HMS to keep the information consistent and avoid confusion between a customer’s expectation and the actual product | | Data | All modified inputs | | Stimulus | A mouse click initiated by the user | | Response | A confirmation message verifying that the modifications were saved successfully | | Comments | Only managers may to access options for room type modification. This act is not permitted from an employee’s account. | |

### 3.3.43 Use Case: Delete Room Type: Select Room Type

|  |
| --- |
| Section 3.3.43. Figure 1: Use case for selecting the room type to delete. |
| |  |  | | --- | --- | | HMS: Selecting room type to delete | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | When management decides to stop providing a certain type of room to be rented, it is important to delete that room type from the database so that no reservations can come in for those room types | | Data | The selected room type | | Stimulus | A mouse click initiated by the user | | Response | None | | Comments | Only managers may to access options for room type deletion. This act is not permitted from an employee’s account. | |

### 3.3.44 Use Case: Delete Room Type: Erase Selected Room Type

|  |
| --- |
| Section 3.3.44. Figure 1: Use case for finalizing room type deletion. |
| |  |  | | --- | --- | | HMS: Finalizing deletion | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | When management decides to stop providing a certain type of room to be rented, it is important to delete that room type from the database so that no reservations can come in for those room types | | Data | The selected room type | | Stimulus | A mouse click initiated by the user | | Response | Confirmation verifying that the room type was deleted successfully | | Comments | Only managers may to access options for room type deletion. This act is not permitted from an employee’s account. | |

### 3.3.45 Use Case: Add Room: Room Number

|  |
| --- |
| Section 3.3.45. Figure 1: Use case for inputting a room number for the new room. |
| |  |  | | --- | --- | | HMS: Adding a room number | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | When management decides to provide a new room to be rented, it is important to add a room number to the database so that a reservation can exist for that room | | Data | Room number | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Only the managers have permission to add the room number information. | |

### 3.3.46 Use Case: Add Room: Room Type

|  |
| --- |
| Section 3.3.46. Figure 1: Use case for inputting a room type for the new room. |
| |  |  | | --- | --- | | HMS: Adding a room type | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | When management decides to provide a new room to be rented, it is important to add a room type to the database so that a reservation can exist for that room | | Data | Room type | | Stimulus | Typed text issued by the user | | Response | None | | Comments | Only the managers have permission to add the room type information. | |

### 3.3.51 Use Case: Add Account: Input Name

|  |
| --- |
| Section 3.3.51. Figure 1: Use case for inputting the name for account |

|  |  |
| --- | --- |
| HMS: Inputting name upon account creation | |
| Actors | Manager, Hotel Management System (HMS) Database |
| Description | Upon creating a new account, the manager may enter the name of the account into the database to make it available to be viewed from the user interface. |
| Data | Name of the account (first and last name) |
| Stimulus | Typed text issued by the manager |
| Response | None |
| Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. |

### 3.3.52 Use Case: Add Account: Input Age

|  |
| --- |
| Section 3.3.52. Figure 1: Use case for inputting age for the account |
| |  |  | | --- | --- | | HMS: Inputting age upon account creation | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon creating a new account, the manager may input the age of the user into the database. | | Data | Age of user | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. | |

### 3.3.53 Use Case: Add Account: Input Gender

|  |
| --- |
| Section 3.3.53. Figure 1: Use case for inputting gender for account |

|  |  |
| --- | --- |
| HMS: Inputting gender upon account creation | |
| Actors | Manager, Hotel Management System (HMS) Database |
| Description | Upon creating a new account, the manager may input the gender of the user into the database to make it available to be viewed from the user interface. |
| Data | Gender of the user |
| Stimulus | Typed text issued by the manager |
| Response | None |
| Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. |

### 3.3.54 Use Case: Add Account: Input Position

|  |
| --- |
| Section 3.3.54. Figure 1: Use case for inputting position for account |

|  |  |
| --- | --- |
| HMS: Inputting position upon account creation | |
| Actors | Manager, Hotel Management System (HMS) Database |
| Description | Upon creating a new account, the manager may input the position of the account of the user into the database to make it available to be viewed from the user interface. |
| Data | Position of the user |
| Stimulus | Typed text issued by the manager |
| Response | None |
| Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. |

### 3.3.55 Use Case: Add Account: Input Access Level

|  |
| --- |
| Section 3.3.55. Figure 1: Use case for inputting access level for account |

|  |  |
| --- | --- |
| HMS: Inputting access level upon account creation | |
| Actors | Manager, Hotel Management System (HMS) Database |
| Description | Upon creating a new account, the manager may enter the access level of the account into the database to make it available to be viewed from the user interface. |
| Data | Access level of the user(employee, manager) |
| Stimulus | Typed text issued by the manager |
| Response | None |
| Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. |

### 3.3.56 Use Case: Add Account: Input Phone Number

|  |
| --- |
| Section 3.3.56. Figure 1: Use case for inputting phone number for account |

|  |  |
| --- | --- |
| HMS: Inputting phone number upon account creation | |
| Actors | Manager, Hotel Management System (HMS) Database |
| Description | Upon creating a new account, the manager may input the phone number of the user into the database to make it available to be viewed from the user interface. |
| Data | Phone number of the account |
| Stimulus | Typed text issued by the manager |
| Response | None |
| Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. |

### 3.3.57 Use Case: Add Account: Input Emergency Contact

|  |
| --- |
| Section 3.3.57. Figure 1: Use case for inputting emergency contact information for an account. |
| |  |  | | --- | --- | | HMS: Inputting emergency contact upon account creation | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon creating a new account, the manager may input the emergency contact info of the user into the database to make it available to be viewed from the user interface. | | Data | Emergency Contact Info for the Account(name, phone number, and/or relationship) | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. | |

### 3.3.58 Use Case: Add Account: Input Email Address

|  |
| --- |
| Section 3.3.58. Figure 1: Use case for inputting name for email address for account. |
| |  |  | | --- | --- | | HMS: Inputting email address upon account creation | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon creating a new account, the manager may input the email address of the user into the database to make it available to be viewed from the user interface. | | Data | The email address for the Account | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. | |

### 3.3.59 Use Case: Add Account: Input Address

|  |
| --- |
| Section 3.3.59. Figure 1: Use case for inputting address for account |
| |  |  | | --- | --- | | HMS: Inputting address upon account creation | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon creating a new account, the manager may input the living address of the user into the database to make it available to be viewed from the user interface. | | Data | Address for the account (street number, street name, and/or apartment number) | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. | |

### 3.3.60 Use Case: Add Account: Input City

|  |
| --- |
| Section 3.3.60. Figure 1: Use case for inputting city for the account |
| |  |  | | --- | --- | | HMS: Inputting city upon account creation | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon creating a new account, the manager may input the city of the user’s place of residence into the database. | | Data | City of user | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. | |

### 3.3.61 Use Case: Add Account: Input Zip Code

|  |
| --- |
| Section 3.3.61. Figure 1: Use case for inputting zip code for account |

|  |  |
| --- | --- |
| HMS: Inputting zip code upon account creation | |
| Actors | Manager, Hotel Management System (HMS) Database |
| Description | Upon creating a new account, the manager may input the zip code of the user into the database to make it available to be viewed from the user interface. |
| Data | Zip code for the user’s residence |
| Stimulus | Typed text issued by the manager |
| Response | None |
| Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. |

### 3.3.62 Use Case: Add Account: Input Country

|  |
| --- |
| Section 3.3.62. Figure 1: Use case for inputting country for the account |
| |  |  | | --- | --- | | HMS: Inputting age upon account creation | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon creating a new account, the manager may input the country of the user’s residence into the database. | | Data | Country of user | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. | |

### 3.3.63 Use Case: Add Account: Input Additional Details

|  |
| --- |
| Section 3.3.63. Figure 1: Use case for inputting additional details for account |
| |  |  | | --- | --- | | HMS: Inputting additional details upon account creation | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon creating a new account, the manager may choose to input the any additional information into the database to make it available to be viewed from the user interface. | | Data | Any additional details for the account. | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may to access options for room type creation. This act is not permitted from an employee’s account. | |

### 3.3.64 Use Case: Modify Account: Modify Name

|  |
| --- |
| Section 3.3.64. Figure 1: Use case for replace name for an account. |
| |  |  | | --- | --- | | HMS: Modifying name upon account search | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon editing an account for a hotel, the manager/employee may modify the account from the database to make it the new justification available to begin real time adjustments. | | Data | The name for the account | | Stimulus | Typed text issued by the manager/employee | | Response | None | | Comments | Only managers or employees may access options account revision.  The manager is the only user who can modify any employee’s information. | |

### 3.3.65 Use Case: Modify Account: Modify Age

|  |
| --- |
| Section 3.3.65. Figure 1: Use case for replace age for an account. |
| |  |  | | --- | --- | | HMS: Modifying age upon account search | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon editing an account for a hotel, the manager/employee may modify the account from the database to make it the new justification available to begin real time adjustments. | | Data | The age for the account | | Stimulus | Typed text issued by the manager/employee | | Response | None | | Comments | Only managers or employees may access options account revision.  The manager is the only user who can modify any employee’s information. | |

### 3.3.66 Use Case: Modify Account: Modify Gender

|  |
| --- |
| Section 3.3.66. Figure 1: Use case for replacing gender for an account. |
| |  |  | | --- | --- | | HMS: Modifying gender upon account search | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon editing an account for a hotel, the manager/employee may modify the account from the database to make it the new justification available to begin real time adjustments. | | Data | The gender for the account | | Stimulus | Typed text issued by the manager/employee | | Response | None | | Comments | Only managers or employees may access options account revision.  The manager is the only user who can modify any employee’s information. | |

### 3.3.67 Use Case: Modify Account: Modify Position

|  |
| --- |
| Section 3.3.67. Figure 1: Use case for replacing position for an account. |
| |  |  | | --- | --- | | HMS: Modifying position upon account search | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon editing an account for a hotel, the manager may modify the account from the database to make it the new justification available to begin real time adjustments. | | Data | The position for the account | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may access options account revision.  The manager is the only user who can modify any employee’s information. | |

### 3.3.68 Use Case: Modify Account: Modify Account Access

|  |
| --- |
| Section 3.3.68. Figure 1: Use case for replacing account access for an account. |
| |  |  | | --- | --- | | HMS: Modifying account access upon account search | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon editing an account for a hotel, the manager may modify the account from the database to make it the new justification available to begin real time adjustments. | | Data | The account access for the account | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may access options account revision.  The manager is the only user who can modify any employee’s information. | |

### 3.3.69 Use Case: Modify Account: Modify Phone Number

|  |
| --- |
| Section 3.3.69. Figure 1: Use case for replacing phone number for an account. |
| |  |  | | --- | --- | | HMS: Modifying the phone number upon account search | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon editing an account for a hotel, the manager may modify the account from the database to make it the new justification available to begin real time adjustments. | | Data | The phone number for the account | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may access options account revision.  The manager is the only user who can modify any employee’s information. | |

### 3.3.70 Use Case: Modify Account: Modify Emergency Contact

|  |
| --- |
| Section 3.3.70. Figure 1: Use case for replacing emergency contact for an account. |
| |  |  | | --- | --- | | HMS: Modifying the emergency contact number upon account search | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon editing an account for a hotel, the manager may modify the account from the database to make it the new justification available to begin real time adjustments. | | Data | The emergency contact number for the account | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may access options account revision.  The manager is the only user who can modify any employee’s information. | |

### 3.3.71 Use Case: Modify Account: Modify Email Address

|  |
| --- |
| Section 3.3.71. Figure 1: Use case for replacing email address for an account. |
| |  |  | | --- | --- | | HMS: Modifying the email address upon account search | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon editing an account for a hotel, the manager may modify the account from the database to make it the new justification available to begin real time adjustments. | | Data | The email address number for the account | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may access options account revision.  The manager is the only user who can modify any employee’s information. | |

### 3.3.72 Use Case: Modify Account: Modify Address

|  |
| --- |
| Section 3.3.72. Figure 1: Use case for replacing address for an account. |
| |  |  | | --- | --- | | HMS: Modifying the address upon account search | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon editing an account for a hotel, the manager may modify the account from the database to make it the new justification available to begin real time adjustments. | | Data | The address number for the account | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may access options account revision.  The manager is the only user who can modify any employee’s information. | |

### 3.3.73 Use Case: Modify Account: Modify City

|  |
| --- |
| Section 3.3.73. Figure 1: Use case for replacing city for an account. |
| |  |  | | --- | --- | | HMS: Modifying the city upon account search | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon editing an account for a hotel, the manager may modify the account from the database to make it the new justification available to begin real time adjustments. | | Data | The city for the account | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may access options account revision.  The manager is the only user who can modify any employee’s information. | |

### 3.3.74 Use Case: Modify Account: Modify Zip Code

|  |
| --- |
| Section 3.3.74. Figure 1: Use case for replacing zip code for an account. |
| |  |  | | --- | --- | | HMS: Modifying the zip code upon account search | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon editing an account for a hotel, the manager may modify the account from the database to make it the new justification available to begin real time adjustments. | | Data | The zip code for the account | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may access options account revision.  The manager is the only user who can modify any employee’s information. | |

### 3.3.75 Use Case: Modify Account: Modify Country

|  |
| --- |
| Section 3.3.75. Figure 1: Use case for replacing country for an account. |
| |  |  | | --- | --- | | HMS: Modifying the country upon account search | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon editing an account for a hotel, the manager may modify the account from the database to make it the new justification available to begin real time adjustments. | | Data | The country for the account | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may access options account revision.  The manager is the only user who can modify any employee’s information. | |

### 3.3.76 Use Case: Modify Account: Modify Additional Details

|  |
| --- |
| Section 3.3.76. Figure 1: Use case for replacing additional details for an account. |
| |  |  | | --- | --- | | HMS: Modifying the additional details upon account search | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon editing an account for a hotel, the manager may modify the account from the database to make it the new justification available to begin real time adjustments. | | Data | The addition details for the account | | Stimulus | Typed text issued by the manager | | Response | None | | Comments | Only managers may access options account revision.  The manager is the only user who can modify any employee’s information. | |

### 3.3.77 Use Case: Delete Account: Select Account

|  |
| --- |
| Section 3.3.77. Figure 1: Use case for searching accounts. |
| |  |  | | --- | --- | | HMS: Searching an account | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon searching an account for a hotel, the manager may delete the account from the database to make real time adjustments. | | Data | The account | | Stimulus | Mouse click by the manager | | Response | None | | Comments | Only managers may access options for account revisions.  The manager is the only user who can delete any employee’s information. | |

### 3.3.78 Use Case: Delete Account: Erase Selection

|  |
| --- |
| Section 3.3.33. Figure 1: Use case for deselecting accounts. |
| |  |  | | --- | --- | | HMS: Deselect an account | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | Upon searching an account for a hotel, the manager may unselect the account from the database to make real time adjustments. | | Data | The account | | Stimulus | Mouse click by the manager | | Response | None | | Comments | Only managers may access options for account revisions.  The manager is the only user who can modify any employee’s information. | |

### 3.3.79 Use Case: Change Daily Rate: View Rates (Open up the window)

|  |
| --- |
| Section 3.3.79. Figure 1: Use case for opening the rates management window. |
| |  |  | | --- | --- | | HMS: Open rates management window | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager decides to change the daily rates for the hotel, the manager needs to press the rates button to open up the rates management window. | | Data | None | | Stimulus | Button is pressed by the user | | Response | Opens the rates management window | | Comments | The managers are the only employees with access to modifying the daily rates. | |

### 3.3.80 Use Case: Change Daily Rate: Select Start Date

|  |
| --- |
| Section 3.3.80. Figure 1: Use case for inputting the starting date of the new daily rates. |
| |  |  | | --- | --- | | HMS: Inputting starting date of the new daily rates | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager decides to change the daily rates for the hotel, the manager needs to input the starting date of the new daily rates to be applied to the rooms. | | Data | Starting date for the daily rate | | Stimulus | Typed text issued by the user | | Response | None | | Comments | The managers are the only employees with access to modifying the daily rates. | |

### 3.3.81 Use Case: Change Daily Rate: Select End Date

|  |
| --- |
| Section 3.3.81. Figure 1: Use case for inputting the ending date of the new daily rates. |
| |  |  | | --- | --- | | HMS: Inputting ending date of the new daily rates | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager decides to change the daily rates for the hotel, the manager needs to input the ending date of the new daily rates to be applied to the rooms. | | Data | Ending date for the daily rate | | Stimulus | Typed text issued by the user | | Response | None | | Comments | The managers are the only employees with access to modifying the daily rates. | |

### 3.3.82 Use Case: Change Daily Rate: Enter difference from Rack Rate

|  |
| --- |
| Section 3.3.82. Figure 1: Use case for inputting the difference from the rack rate. |
| |  |  | | --- | --- | | HMS: Inputting difference from Rack Rate | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager decides to change the daily rates for the hotel, the manager needs to input the difference from the Rack Rate. | | Data | Difference from the Rack Rates | | Stimulus | Typed text issued by the user | | Response | None | | Comments | The managers are the only employees with access to modifying the daily rates. | |

### 3.3.83 Use Case: Change Daily Rate: Finalize Modification

|  |
| --- |
| Section 3.3.83. Figure 1: Use case for Finalizing the modifications made to the daily rates. |
| |  |  | | --- | --- | | HMS: Inputting difference from Rack Rate | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager decides to change the daily rates for the hotel, the manager needs press the Finished button to finalize the desired changes made to the daily rates | | Data | None | | Stimulus | Button is pressed by the user | | Response | Stores desired changes made to the daily rates | | Comments | The managers are the only employees with access to modifying the daily rates. | |

### 3.3.84 Use Case: Add Discount: Open Add Discount Window

|  |
| --- |
| Section 3.3.84. Figure 1: Use case for pressing the Add Discount button to add a new discount to the HMS. |
| |  |  | | --- | --- | | HMS: Pressing the Add Discount button to add a new discount | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager needs to add a new discount to the Hotel Management System (HMS), he must first press the Add Discount button to open the ADD discount window | | Data | None | | Stimulus | Button is pressed by the user | | Response | Opens an Add Discount window | | Comments | The managers are the only employees with access to adding discounts | |

### 3.3.85 Use Case: Add Discount: Input Discount Code

|  |
| --- |
| Section 3.3.85. Figure 1: Use case for inputting the discount code. |
| |  |  | | --- | --- | | HMS: Inputting the discount code | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager needs to add a new discount to the Hotel Management System (HMS), he must input the discount code for the object Discount. | | Data | Discount code for the discount | | Stimulus | Typed text issued by the user | | Response | None | | Comments | The managers are the only employees with access to adding discounts | |

### 3.3.86 Use Case: Add Discount: Input Discount Rate

|  |
| --- |
| Section 3.3.86. Figure 1: Use case for inputting the discount rate. |
| |  |  | | --- | --- | | HMS: Inputting the discount rate | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager needs to add a new discount to the Hotel Management System (HMS), he must input the discount rate for the object Discount. | | Data | Discount rate for the discount | | Stimulus | Typed text issued by the user | | Response | None | | Comments | The managers are the only employees with access to adding discounts | |

### 3.3.87 Use Case: Add Discount: Input Discount Description

|  |
| --- |
| Section 3.3.87. Figure 1: Use case for inputting the discount description. |
| |  |  | | --- | --- | | HMS: Inputting the discount description | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager needs to add a new discount to the Hotel Management System (HMS), he must input the discount description for the object Discount. | | Data | Discount code for the discount | | Stimulus | Typed text issued by the user | | Response | None | | Comments | The managers are the only employees with access to adding discounts | |

### 3.3.88 Use Case: Add Discount: Finalize Discount

|  |
| --- |
| Section 3.3.88. Figure 1: Use case for pressing the Finished button. |
| |  |  | | --- | --- | | HMS: Pressing the Finished button | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager needs to add a new discount to the Hotel Management System (HMS), he must press the Finished button to save the newly created object Discount and add it to the HMS. | | Data | None | | Stimulus | Pressing the Finished button | | Response | Save the newly created Discount object | | Comments | The managers are the only employees with access to adding discounts | |

### 3.3.89 Use Case: Modify Discount: Select Discount to Modify

|  |
| --- |
| Section 3.3.89. Figure 1: Use case for selecting the discount to be modified. |
| |  |  | | --- | --- | | HMS: Selecting the discount to be modified | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager needs to modify an existing discount in the Hotel Management System (HMS), he must select the desired discount to be modified | | Data | Desired discount to be modified | | Stimulus | User selection through the mouse | | Response | Highlights the selected discount. | | Comments | The managers are the only employees with access to modifying the discounts. | |

### 3.3.90 Use Case: Modify Discount: Open Selected Discount to Modify

|  |
| --- |
| Section 3.3.90. Figure 1: Use case for opening selected discount to modify. |
| |  |  | | --- | --- | | HMS: Opening selected discount to modify | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | After the manager has selected the Discount he/she desires to modify, he/she must press the modify button to open the window which will allow you to modify the Discount. | | Data | None | | Stimulus | Pressing the Modify button | | Response | Opens the window which allows the user to modify the Discount | | Comments | The managers are the only employees with access to modifying the discounts. | |

### 3.3.91 Use Case: Modify Discount: Modify Attributes

|  |
| --- |
| Section 3.3.91. Figure 1: Use case for modifying the attributes of the Discount. |
| |  |  | | --- | --- | | HMS: Modifying the desired attributes | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager needs to modify an existing discount in the Hotel Management System (HMS), he may modify the desired attributes of the Discount object | | Data | Correct data pertaining to the attributes being modified | | Stimulus | Typed text issued by the user | | Response | None | | Comments | The managers are the only employees with access to modifying the discounts. | |

### 3.3.92 Use Case: Modify Discount: Finalize Modifications

|  |
| --- |
| Section 3.3.92. Figure 1: Use case for finalizing the modifications |
| |  |  | | --- | --- | | HMS: Finalizing the modifications | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager needs to modify an existing discount in the Hotel Management System (HMS), he must press the Finished button to save the modifications and add the object to the system | | Data | None | | Stimulus | Pressing the finished button | | Response | Saves the modifications made | | Comments | The managers are the only employees with access to modifying the discounts. | |

### 3.3.93 Use Case: Delete Discount: Select Discount to Delete

|  |
| --- |
| Section 3.3.93. Figure 1: Use case for selecting the Discount to be deleted. |
| |  |  | | --- | --- | | HMS: Selecting the discount to be deleted | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager needs to delete an existing discount in the Hotel Management System (HMS), he must select the desired discount to be deleted | | Data | Discount to be deleted | | Stimulus | Selecting a discount using the mouse left-click | | Response | Highlights the selected discount. | | Comments | The managers are the only employees with access to deleting the Discounts | |

### 3.3.94 Use Case: Delete Discount: Review Deletion

|  |
| --- |
| Section 3.3.94. Figure 1: Use case for pressing the button to initiate deletion process |
| |  |  | | --- | --- | | HMS: Selecting the discount to be deleted | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager needs to delete an existing discount in the Hotel Management System (HMS), he must press the Delete button. The HMS will then bring up a window with a warning message. | | Data | None | | Stimulus | Pressing the Delete button | | Response | Opens a window containing a warning message | | Comments | The managers are the only employees with access to deleting the Discounts | |

### 3.3.95 Use Case: Delete Discount: Confirm Deletion

|  |
| --- |
| Section 3.3.95. Figure 1: Use case for pressing the button to confirm/cancel deletion. |
| |  |  | | --- | --- | | HMS: Selecting the button to confirm/cancel deletion | | | Actors | Manager, Hotel Management System (HMS) Database | | Description | If the manager needs to delete an existing discount in the Hotel Management System (HMS), he must either press the Confirm button to confirm and delete the Discount or the Cancel button to cancel the deletion | | Data | Button pressed | | Stimulus | Pressing the Delete button | | Response | Either deletes the Discount or cancels the process | | Comments | The managers are the only employees with access to deleting the Discounts | |

### 3.3.96 Use Case: Using the Interface: Login Menu

|  |
| --- |
| Section 3.3.96. Figure 1: Use case for starting HMS program to bring up the login screen. |
| |  |  | | --- | --- | | HMS: The Log In process | | | Actors | Manager/Employee, Hotel Management System (HMS) Database | | Description | All managers and employees have unique user names and passwords to allow them to log in to HMS. The access level is defined by the user’s job title specified in the account | | Data | The file to start HMS | | Stimulus | A mouse click issued by the user | | Response | Outputs the Login Screen | | Comments | All users have access to the Login screen, but only those with an appropriate username and password combination to an account may gain access to the program. | |

### 3.3.97 Use Case: Using the Interface: Enter Username

|  |
| --- |
| Section 3.3.97. Figure 1: Use case for entering the user name in the login screen. |
| |  |  | | --- | --- | | HMS: Entering a user name | | | Actors | Manager/Employee, Hotel Management System (HMS) Database | | Description | Users must provide a username associated with an account to gain access to HMS | | Data | A valid username | | Stimulus | Typed text provided by user | | Response | None | | Comments | All users have access to the Login screen, but only those with an appropriate username and password combination to an account may gain access to the program. | |

### 3.3.98 Use Case: Using the Interface: Enter Password

|  |
| --- |
| Section 3.3.98. Figure 1: Use case for entering a password in the login screen. |
| |  |  | | --- | --- | | HMS: Entering a password | | | Actors | Manager/Employee, Hotel Management System (HMS) Database | | Description | Users must provide a password associated with an account to gain access to HMS | | Data | A valid password correlated with the previously typed user name | | Stimulus | Typed text provided by user | | Response | None | | Comments | All users have access to the Login screen, but only those with an appropriate username and password combination to an account may gain access to the program. | |

### 3.3.99 Use Case: Using the Interface: Finalize Login

|  |
| --- |
| Section 3.3.99. Figure 1: Use case for finalizing the log in. |
| |  |  | | --- | --- | | HMS: Entering a password | | | Actors | Manager/Employee, Hotel Management System (HMS) Database | | Description | After providing the username and password, HMS will decide if the combination is correct before granting access to HMS features. | | Data | The user provided input for username and password | | Stimulus | A mouse click initiated by user | | Response | Access to HMS functionalities | | Comments | All users have access to the Login screen, but only those with an appropriate username and password combination to an account may gain access to the program. | |

## 3.4 Objects

In object-oriented development, an object is a set of attributes and functions contained in a class that is designed for a specific operation. Variables and methods inside objects may be contained within the object for data collection or shared with other objects. The organized set of interactions leads the system to behave in an orderly manner allowing humans to use it as a program.

This section is a collection of all the objects required to fulfill the specific requirements detailed in this SRS.

### 3.4.1 Object: Hotel



**Section 3.4.1 Figure 1:** A UML diagram for object Hotel. Part A

#### 

**Section 3.4.1 Figure 2:** A UML diagram for object Hotel. Part B

#### ***Attributes***

* allRoomTypes : A vector containing all the different room types in memory.
* allRooms : A vector containing all the different rooms in memory
* allAccounts : A vector containing all the Accounts in memory
* allReservations: A vector containing all Reservations in memory
* allFeatures: A vector containing all Features in memory
* allDiscounts: A vector containing all Discount objects in memory
* calendarArray[]: An array of lists of Reservation pointers holding 16 weeks of reservation data. This is used in the GUI only
* tax : a double holding the tax amount
* db : a pointer to manipulate the database

#### Functions

* Hotel(): A constructor for the hotel class. It initializes everything and uploads all data from the database
* accountExists(string): A function for checking the existence of an account
* accountExistsByUsername(string): A function for finding reservation by username
* addAccount():A function for adding an account in database and memory
* addAccount(Account): A function overloaded for easier manipulation
* addDiscount(string, string, double): A function for adding discount objects
* addReservation(Reservation): A function for adding reservation objects
* addRoom(int, string): A function for adding room objects
* addRoomFeature(int, string, string): A function for adding feature objects
* addRoomType(string, string): A function a function for adding RoomType objects
* assignRoomNumber(Reservation, int): A function for assigning room numbers to specified reservation
* calculateDateIndex(Date): A function for finding the index in the calendar array
* cancelRes(Reservation): a function that cancels a reservation
* changeRoomFeature(int, string): A function that changes features of a specified room
* checkInRes(Reservation): A function that checks in a reservation into a room
* checkOutRes(Reservation): A function that checks out a reservation from a room
* deleteAccount(string): A function that deletes an account
* deleteDiscount(string): A function that deletes a discount object
* deleteReservation(int): A function that deletes a reservation from memory only
* deleteReservationFromDB(int): A function that deletes a reservation from database only
* deleteRoom(int): A function that deletes Room objects from both memory and database
* deleteRoomFeature(int, string): A function that deletes a feature from room objects
* deleteRoomType():A function that deletes a room type object
* discountExists(string): A function that checks if a discount object exists. Used for traversing through the vector and for checking purposes
* featureExists(string): A function that checks if a specific feature object exists in memory
* findAccount(string): A function that finds an account object in memory
* findAccountByUsername(string): A function that finds an account by specified username
* findDiscount(string): A function that finds a discount object
* findReservation(int, int): A function that finds a reservation object. Used by GUI.
* findReservation(int): A function overloaded that is simpler and easier to use and requires less information.
* findRoom(int): A function that finds a Room object in memory
* findRoomType(string): A function that finds a roomtype object in memory
* getAccountsBegin(): a function that marks the beginning of a container of accounts
* getAccountsEnd():A function that marks the end of a container of accounts
* getCalendarArray(): A function that returns a pointer to the array so the data can be accessed inside the array
* getCurrentDate():A function that returns the current date
* getGrandTotal(string, string): A function that returns the grand total price for a reservation
* getPrice(string) : A function that returns the price of a reservation by looking at an object
* getRackRate(string) : A function that returns the rack rate of a room type
* getReportsForAccountingDetails(): A function that returns a string listing the details of revenue and accounting information
* getReportsForAccountingRevenue(): A function that returns the revenue in reports
* getReportsForAccounts(): A function that returns a string of details about accounts
* getReportsForCreditCards(): A function that reports the details of credit cards
* getReportsForRatesDiscounts(): A function that is used to list the discounts available
* getReportsRackRates(): A function that lists the rack rates for different room types
* getReportsReservationsCancelled(): A function that reports on the reservations that had been cancelled
* getReportsReservationsCheckedIn(): A function that reports on the reservations that had been checked in
* getReportsReservationsCheckedOut(): A function that reports on the reservations that had been checked out
* getReportsReservationsNoShows(): A function that reports on the reservations that had been no shows
* getReportsReservationsReserved():A function that reports on the reservations that had been reserved
* getReservationFromDB(int): A function that returns a reservation object from the database
* getSubTotal(string, string) : A function that returns the sub total for a reservation
* getTaxRate(): A function that returns the tax amount for that hotel business
* getTotalNights(Date \*, Date\*): A function that returns the total number of nights by subtracting the arrival from the departure date
* modifyAccount(): A function that is used to edit information in account objects
* modifyAccount(Account) : A function overloaded for easier account modification
* modifyDiscount(string, string) : A function that modifies a discount object
* modifyReservation(Reservation) : A function that is used to change things around in a reservation
* modifyReservation(): A function overloaded to make things simpler when changing the reservation. Some parts use this function, others use the other function
* modifyRoom(int, int string): : A function that is used to modify room objects.
* modifyRoomType(string) : A function that is used to modify room type objects
* noShowRes(Reservation) : A function that marks a reservation as a no show and processes the necessary information
* populateArray(): A function that feeds in information from database to the calendar array.
* printAccount(): A function that is used primary to test account objects in memory and in database
* printDiscount(): A function that used primarily to test the discount objects both in memory and in database
* printReservationArray(): A function that prints out all the reservations for 16 weeks that are stored in memory
* printRoomsFeatures(): A function that prints all the room features available
* removeRoomAssignment(Reservation) : A function that removes a room number from a reservation
* reserve(Reservation) : A function that is used for putting reservations back from cancelled or checked out state. Must meet certain conditions
* roomNumberExists(int) : A function that is used for checking if the room number exists
* roomTypeExists(string) : A function for checking if a certain room type exists
* setCurrentDate(Date) : A function for changing the current date in hotel
* setRackRate(string, double) : A function that is used to change rack rates for different room type obejcts
* setTaxRate(double) : A setter for the tax variable
* ~Hotel(): Destructor for the Hotel object

This object is the primary class that will be used by all other objects. This object creates a hotel in memory, fills it in with all the necessary information from the database, and runs the rest in memory.

### 3.4.2 Object: RoomType



**Section 3.4.2 Figure 1:** A UML diagram for object RoomType.

#### ***Attributes***

* typeName: A string that holds the name of the room type.
* numberOfBeds: An integer that holds the number of beds available in that RoomType.

#### Functions

* getRoomType: A getter for the name of the room type. It returns a string.
* getNumberOfBeds: A getter for the number of beds in the room type. It returns an integer.
* setTypeName: A setter for the name of the room type.
* setNumberOfBeds: A setter for the number of Beds in the room type.
* print: A function that prints the name and the number of beds in the room type. This function is used for testing and displaying purposes.

This object is used as a template for the different types of rooms in HMS. Each room type must have its own name to make it unique and the number of beds in each room type is dependent on the number of physical beds available at the hotel for that specific room type. It is used by the following objects (separated by commas): Room, Hotel

### 3.4.3 Object: Room



**Section 3.4.3 Figure 1:** A UML diagram for object Room.

#### Attributes

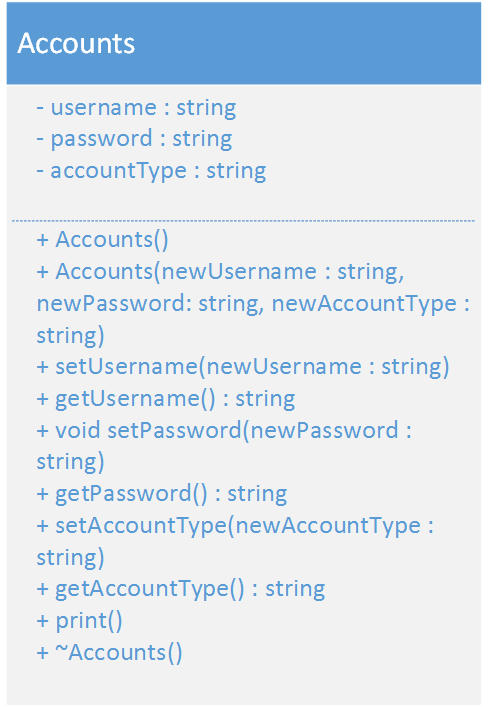
* roomNumber: An integer that holds the room number of the hotel room.
* type: A pointer variable pointing to an object RoomType that indicates the type of the hotel room.
* roomFeatures: A variable of an object Features that keeps track of all the features available in the hotel room.

#### Functions

* getRoomNumber: A getter for the roomNumber.
* setRoomNumber: A setter for the roomNumber.
* getRoomType: A getter for the RoomType pointer.
* setRoomType: A setter for the RoomType pointer.
* addFeature: A function that adds a user defined string to the object Features.
* deleteFeature: A function that deletes a specific feature from the list of existing features.
* changeFeature: A function that replaces an existing feature with the new feature.
* print: A function that prints all of the information an instance of an object Room contains. This function is used for testing and displaying purposes.

This object acts as a hotel room. Each instance of this object is stored dynamically in a vector of Room object pointers. Each room can be differentiated by its room number and rooms with an identical number will not be allowed inside the vector of Room object pointers. Each Room object contains its own instance of a Features object which allows the user to add any feature unique to that room inside its individual Feature object as a list of strings. It is used by the following objects (separated by commas): Hotel, Reservation.

### 3.4.4 Object: Accounts



**Section 3.4.4 Figure 1:** A UML diagram for object Accounts

#### Attributes

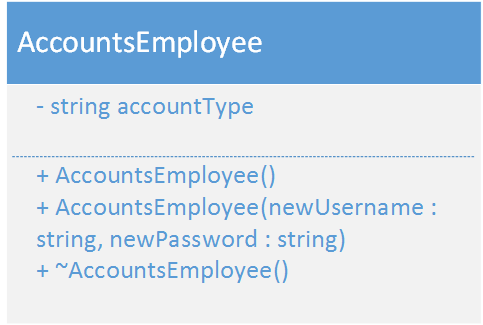
* username: A string that holds the name of the user
* password: A string that holds the password for the particular user
* accountType: A string that holds the account type of the particular user. Must be either “employee” or “manager”.

#### Functions

* Accounts(): Default constructor that does not initialize the variables.
* Accounts(string, string): Overloaded constructor that initializes the variables.
* setUsername(): A setter for the username of a particular user.
* getUsername(): A getter for the username of a particular user. Returns a string.
* setPassword(): A setter for the password of a particular user.
* getPassword(): A getter for the password of a particular user. Returns a string.
* setAccountType(): A setter for the account type of a particular user. Must be either “employee” or “manager”
* getAccountType(): A getter for the account type of a particular user. Must be either “employee” or “manager”. Returns a string.
* setNumberOfBeds(): A setter for the number of Beds in the room type.
* Print(): A function that prints the username, password, and account type of a particular user. This function is used for testing and displaying purposes.

This object is used as a template for the two types of users in HMS. It serves as the parent class for the AccountsEmployee and the AccountsManager classes. Each user must have a unique username, and a secure password. Each user is also assigned an account type, which will determine his/her level of access in HMS.

### 3.4.5 Object: AccountsEmployee



**Section 3.4.5 Figure 1:** A UML diagram for object AccountsEmployee.

#### Attributes

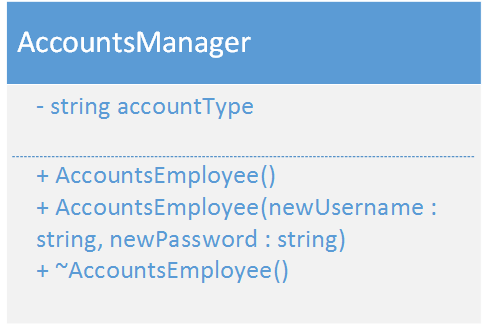
* accountType: A string that holds the account type of the particular user. Must be either “employee” or “manager”.

#### Functions

* Accounts(): Default constructor that does not initialize the variables.
* Accounts(string, string): Overloaded constructor that initializes the variables.

This object is used by one of the two users in HMS. It serves as the child class of the Accounts class and inherits all of its functions and attributes. By using this object, the account type is defaulted to employee, , which can save time in creating the object, and prevent any changes in account type for security purposes. Each user must have a unique username, and a secure password. By being assigned account type “employee”, the user will have a more restricted level of access in the program compared to the “manager” type.

### 3.4.6 Object: AccountsManager



**Section 3.4.6 Figure 1:** A UML diagram for object AccountsManager

#### Attributes

* accountType: A string that holds the account type of the particular user. Must be either “employee” or “manager”.

#### Functions

* Accounts(): Default constructor that does not initialize the variables.
* Accounts(string, string): Overloaded constructor that initializes the variables.

This object is used by one of the two users in HMS. It serves as the child class of the Accounts class and inherits all of its functions and attributes. By using this object, the account type is defaulted to manager, which can save time in creating the object and prevent any changes in account type for security purposes. Each user must have a unique username, and a secure password. By being assigned account type “manager”, the user will have increased levels of access in the program compared to the “employee” type.

### 3.4.7 Object: Reservation



**Section 3.4.7 Figure 1:** A UML diagram for object Reservation.

#### ***Attributes***

* generalInfo: A GeneralInfo object that holds the general information of the client reserving a room.
* requiredInfo: A RequiredInfo object that holds the required information for the client reserving a room.
* extraInfo: A ExtraFeatures object that holds any special requests made by the client reserving a room.
* reservations: A node type that points to all the currently existing reservations.

#### Functions

* Reservation: A constructor that initiates all attributes to the default settings.
* Reservation(overloaded): An overloaded constructor that initiates all attributes to the user’s input.
* setGeneralInfo: A setter for the general information of the reservation type.
* getGeneralInfo: A getter for the general information of the reservation type. It returns a GeneralInfo object.
* setRequiredInfo: A setter for the required information in the reservation type.
* getRequiredInfo: A getter for the required information of the reservation type. It returns a RequiredInfo object.
* setExtraInfo: A setter for the extra information of the reservation type.
* getExtraInfo: A getter for the extra information of the reservation type. It returns a ExtraFeatures object.
* setNode: A setter for the location of the reservation being added to the list of reservations.
* getNode: A getter for the location of the currently existing reservations. It returns a pointer to the location.
* Operator=: Overloads the “=” operator of the Reservation type to make implementation easier.
* ~Reservation: A destructor that resets all the attributes of the reservation type to the default settings.

This object is used to represent reservations that will be made at the hotel. The reservation object will be split into three different categories (GeneralInfo, RequiredInfo, ExtraFeatures) to hold all the necessary information. All the information is required with the exception of the ExtraFeatures object being optional.

### 3.4.8 Object GeneralInfo



**Section 3.4.8 Figure 1:** A UML diagram for object GeneralInfo.

#### ***Attributes***

* firstName: A string that holds the first name of the client.
* lastName: A string that holds the last name of the client.
* address: A string that holds the street address of the client.
* city: A string that holds the city where the client resides.
* state: A string that holds the state where the client resides.
* country: A string that holds the country where the client resides.
* zipCode: A string that holds the zip code of the client.
* email: A string that holds the client’s email.
* phoneNumber: A string that holds the client’s phone number.

#### Functions

* GeneralInfo: A constructor that initiates all attributes to the default settings.
* GeneralInfo(overloaded): An overloaded constructor that initiates all attributes to the user’s input.
* setFirstName: A setter for the first name of the generalInfo type.
* getFirstName: A getter for the first name of the generalInfo type. It returns a string.
* setLastName: A setter for the last name in the generalInfo type.
* getLastName: A getter for the last name of the generalInfo type. It returns a string.
* setAddress: A setter for the address of the generalInfo type.
* getAddress: A getter for the address of the generalInfo type. It returns a string.
* setCity: A setter for the city of the generalInfo type.
* getCity: A getter for the city of the generalInfo type. It returns a string.
* setState: A setter for the state of the generalInfo type.
* getState: A getter for the state of the generalInfo type. It returns a string.
* setCountry: A setter for the country of the generalInfo type.
* getCountry: A getter for the country of the generalInfo type. It returns a string.
* setZipCode: A setter for the zip code of the generalInfo type.
* getZipCode: A getter for the zip code of the generalInfo type. It returns a string.
* setEmail: A setter for the email of the generalInfo type.
* getEmail: A getter for the email of the generalInfo type. It returns a string.
* setPhoneNumber: A setter for the phone number of the generalInfo type.
* getPhoneNumber: A getter for the phone number of the generalInfo type. It returns a string.
* Operator=: Overloads the “=” operator of the discount type to make implementation easier.
* ~GeneralInfo: A destructor that resets all attributes to the default settings.

This object will hold all of the general contact information of the client making a reservation at the hotel. The majority, if not all, of this information will be mandatory in order for the client to be able to reserve a room.

### 3.4.9 Object: RequiredInfo



**Section 3.4.9 Figure 1:** A UML diagram for object RequiredInfo.

#### ***Attributes***

* arrivalDate: An integer that holds the arrival date of the client.
* departureDate: A integer that holds the departure date of the client.
* roomType: A string that holds the type of room being reserved.

#### Functions

* RequiredInfo: A constructor that initiates all attributes to the default settings.
* RequiredInfo(overloaded): An overloaded constructor that initiates all attributes to the user’s input.
* setArrivalDate: A setter for the arrival date of the RequiredInfo type.
* getArrivalDate: A getter for the arrival date of the RequiredInfo type. It returns a string.
* setDepartureDate: A setter for the departure date in the RequiredInfo type.
* getDepartureDate: A getter for the departure date of the RequiredInfo type. It returns a string.
* setRoomType: A setter for the type of room of the RequiredInfo type.
* getRoomType: A getter for the type of room of the RequiredInfo type. It returns a string.
* Operator=: Overloads the “=” operator of the roomType to make implementation easier.
* ~RequiredInfo: A destructor for the requiredInfo object that resets all attributes to the default settings.

This object is used to hold all personal information necessary to complete a reservation. This includes arrival and departure dates for the client. It will also include what type of room the client is reserving. It will also include the credit card information that will be put on file in order to make the reservation.

### 3.4.10 Object: ExtraFeatures



**Section 3.4.10 Figure 1:** A UML diagram for object ExtraFeatures.

#### Attributes

The object ExtraFeatures has 2 attributes:

* userText: A string that holds the special requests made by the client.
* groupId: An integer that holds the group identification number for a group reservation.

#### Functions

The object ExtraFeatures has 8 functions:

* ExtraFeatures: A constructor that initiates all attributes to the default settings.
* ExtraFeatures(overloaded): An overloaded constructor that initiates all attributes to the user’s input.
* setUserText: A setter for the special requests made by the client reserving the room.
* getUserText: A getter for the special requests made by the client reserving the room. It returns a string.
* setGroupId: A setter for the group identification number of the ExtraFeatures type.
* getGroupId: A getter for the group identification number of the ExtraFeatures type. It returns an integer.
* Operator=: Overloads the “=” operator of the discount type to make implementation easier.
* ~ExtraFeatures: A destructor that resets all the attributes of the ExtraFeatures object to the default settings.

This object is used to hold information on any special requests made by the client reserving a room. It will also hold a group identification number if the reservation is for a big party rather than an individual reservation.

### 3.4.11 Object : Discount



**Section 3.4.11 Figure 1:** A UML diagram for object Discount.

Attributes

* Description: A string that holds the description of the discount.
* Code: A string that holds the code for the designated discount.
* Rate: A double that holds the rate of discount.

#### Functions

* Discount: A constructor that initiates all attributes to the default settings.
* Discount(overloaded): An overloaded constructor that initiates all attributes to the user’s input.
* setDescription: A setter for the description of the discount type.
* getDescription: A getter for the description of the discount type. It returns a string.
* setCode: A setter for the discount code in the discount type.
* getCode: A getter for the discount code of the discount type. It returns a string.
* setRate: A setter for the rate of the discount type.
* getRate: A getter for the rate of the discount type. It returns a double.
* Operator=: Overloads the “=” operator of the discount type to make implementation easier.
* ~Discount: A destructor that will reset all attributes of the Discount object to the default settings.

This object is used to represent promotional discounts that are currently ongoing at the hotel. Each discount will contain a description, discount code, and rate. The discount will be applied to the Reservations class.

### 3.4.12 Object: CreditAmericanExpress



**Section Figure 1:** A UML diagram for object CreditAmericanExpress

#### Attributes

* firstName: A string that holds the first name of the client
* lastName: A string that holds the last name of the client
* address: A string that holds the street number, street name, and/or apartment number which the client resides
* state: A string that holds the state which the client resides. There are 2 characters.
* zipCode: A string that holds the zipCode which the client resides.
* creditCardNumber: A string that holds the credit card number of the client’s credit card. There are 15 digits.
* verificationCode: A string that holds the verification code of the client’s credit card. May be 3 or 4 digits.
* expirationDate: A string that holds the expiration date of the client’s credit card. There are 4 digits. In MMYY form.

#### Functions

* CreditAmericanExpress(): Default constructor that does not initialize the variables.
* getFirstName(): A getter for the first name of the client. Returns a string.
* setFirstName(): A setter for the first name of the client.
* getLastName(): A getter for the first name of the client. Returns a string.
* setLastName(): A setter for the last name of the client.
* getAddress(): A getter for the address of the client. Returns a string.
* setAddress(): A setter for the address that the client resides. May include street number, street name, and/or apartment number.
* getCity(): A getter for the city that the client resides. Returns a string.
* setCity(): A setter for the city that the client resides.
* getState(): A getter for the state that the client resides. Returns a string of two characters.
* setState(): A setter for the state that the client resides.
* getZipCode(): A getter for the zip code of the client. Returns a string.
* setZipCode(): A setter for the zip code that the client resides.
* getCreditCardNumber(): A getter for the credit card number of the client’s credit card. Returns a string of 15 digits.
* setCreditCardNumber(): A setter for the credit card number of the client’s credit card.
* getVerificationCode(): A getter for the verification code of the client’s credit card. Returns a string of either 3 or 4 digits.
* setVerificationCode(): A setter for the verification code of the client’s credit card.
* getExpirationDate(): A getter for the expiration date of the client’s credit card. Returns a string of 4 digits in MMYY format, where MM is the month, and YY is the last two digits of the year.
* setExpirationDate(): A setter for the expiration date of the client’s credit card.
* creditCardVerification(): Compares the client’s information on the credit card with the reservation’s information. Returns true if all fields match.
* ~CreditAmericanExpress(): Destructor.
* Print(): A function that prints the client’s information and credit card information. This function is used for testing and displaying purposes.

This object will hold information on a client’s American Express credit card. It will be associated with a particular client only when the credit card information is of the type American Express. The credit card numbers will have only 15 digits, and the verification code may have 3 or 4 digits

### 3.4.13 Object: CreditDiscover



**Section 3.4.13 Figure 1:** A UML diagram for object CreditDiscover

#### Attributes

* firstName: A string that holds the first name of the client
* lastName: A string that holds the last name of the client
* address: A string that holds the street number, street name, and/or apartment number which the client resides
* state: A string that holds the state which the client resides. There are 2 characters.
* zipCode: A string that holds the zipCode which the client resides.
* creditCardNumber: A string that holds the credit card number of the client’s credit card. There are 16 digits.
* verificationCode: A string that holds the verification code of the client’s credit card. There are 3 digits.
* expirationDate: A string that holds the expiration date of the client’s credit card. There are 4 digits. In MMYY form.

#### Functions

* CreditDiscover(): Default constructor that does not initialize the variables.
* getFirstName(): A getter for the first name of the client. Returns a string.
* setFirstName(): A setter for the first name of the client.
* getLastName(): A getter for the first name of the client. Returns a string.
* setLastName(): A setter for the last name of the client.
* getAddress(): A getter for the address of the client. Returns a string.
* setAddress(): A setter for the address that the client resides. May include street number, street name, and/or apartment number.
* getCity(): A getter for the city that the client resides. Returns a string.
* setCity(): A setter for the city that the client resides.
* getState(): A getter for the state that the client resides. Returns a string of two characters.
* setState(): A setter for the state that the client resides.
* getZipCode(): A getter for the zip code of the client. Returns a string.
* setZipCode(): A setter for the zip code that the client resides.
* getCreditCardNumber(): A getter for the credit card number of the client’s credit card. Returns a string of 16 digits.
* setCreditCardNumber(): A setter for the credit card number of the client’s credit card.
* getVerificationCode(): A getter for the verification code of the client’s credit card. Returns a string of 3 digits.
* setVerificationCode(): A setter for the verification code of the client’s credit card.
* getExpirationDate(): A getter for the expiration date of the client’s credit card. Returns a string of 4 digits in MMYY format, where MM is the month, and YY is the last two digits of the year.
* setExpirationDate(): A setter for the expiration date of the client’s credit card.
* creditCardVerification(): Compares the client’s information on the credit card with the reservation’s information. Returns true if all fields match.
* ~CreditDiscover(): Destructor.
* Print(): A function that prints the client’s information and credit card information. This function is used for testing and displaying purposes.

This object will hold information on a client’s Discover credit card. It will be associated with a particular client only when the credit card information is of the type Discover. The credit card numbers will have only 16 digits, and the verification code will have 3 digits.

### 3.4.14 Object: CreditMastercard



**Section 3.4.14 Figure 1:** A UML diagram for object CreditMasterCard

#### Attributes

* firstName: A string that holds the first name of the client
* lastName: A string that holds the last name of the client
* address: A string that holds the street number, street name, and/or apartment number which the client resides
* state: A string that holds the state which the client resides. There are 2 characters.
* zipCode: A string that holds the zipCode which the client resides.
* creditCardNumber: A string that holds the credit card number of the client’s credit card. There are 16 digits.
* verificationCode: A string that holds the verification code of the client’s credit card. There are 3 digits.
* expirationDate: A string that holds the expiration date of the client’s credit card. There are 4 digits. In MMYY form.

#### Functions

* CreditMastercard(): Default constructor that does not initialize the variables.
* getFirstName(): A getter for the first name of the client. Returns a string.
* setFirstName(): A setter for the first name of the client.
* getLastName(): A getter for the first name of the client. Returns a string.
* setLastName(): A setter for the last name of the client.
* getAddress(): A getter for the address of the client. Returns a string.
* setAddress(): A setter for the address that the client resides. May include street number, street name, and/or apartment number.
* getCity(): A getter for the city that the client resides. Returns a string.
* setCity(): A setter for the city that the client resides.
* getState(): A getter for the state that the client resides. Returns a string of two characters.
* setState(): A setter for the state that the client resides.
* getZipCode(): A getter for the zip code of the client. Returns a string.
* setZipCode(): A setter for the zip code that the client resides.
* getCreditCardNumber(): A getter for the credit card number of the client’s credit card. Returns a string of 16 digits.
* setCreditCardNumber(): A setter for the credit card number of the client’s credit card.
* getVerificationCode(): A getter for the verification code of the client’s credit card. Returns a string of 3 digits.
* setVerificationCode(): A setter for the verification code of the client’s credit card.
* getExpirationDate(): A getter for the expiration date of the client’s credit card. Returns a string of 4 digits in MMYY format, where MM is the month, and YY is the last two digits of the year.
* setExpirationDate(): A setter for the expiration date of the client’s credit card.
* creditCardVerification(): Compares the client’s information on the credit card with the reservation’s information. Returns true if all fields match.
* ~CreditMastercard(): Destructor.
* Print(): A function that prints the client’s information and credit card information. This function is used for testing and displaying purposes.

This object will hold information on a client’s Mastercard credit card. It will be associated with a particular client only when the credit card information is of the type Mastercard. The credit card numbers will have only 16 digits, and the verification code will have 3 digits.

### 3.4.15 Object: CreditVisa



**Section 3.4.15 Figure 1:** A UML diagram for object RoomType.

#### Attributes

* firstName: A string that holds the first name of the client
* lastName: A string that holds the last name of the client
* address: A string that holds the street number, street name, and/or apartment number which the client resides
* state: A string that holds the state which the client resides. There are 2 characters.
* zipCode: A string that holds the zipCode which the client resides.
* creditCardNumber: A string that holds the credit card number of the client’s credit card. There are 15 digits.
* verificationCode: A string that holds the verification code of the client’s credit card. May be 3 or 4 digits.
* expirationDate: A string that holds the expiration date of the client’s credit card. There are 4 digits. In MMYY form.

#### Functions

* CreditVisa(): Default constructor that does not initialize the variables.
* getFirstName(): A getter for the first name of the client. Returns a string.
* setFirstName(): A setter for the first name of the client.
* getLastName(): A getter for the first name of the client. Returns a string.
* setLastName(): A setter for the last name of the client.
* getAddress(): A getter for the address of the client. Returns a string.
* setAddress(): A setter for the address that the client resides. May include street number, street name, and/or apartment number.
* getCity(): A getter for the city that the client resides. Returns a string.
* setCity(): A setter for the city that the client resides.
* getState(): A getter for the state that the client resides. Returns a string of two characters.
* setState(): A setter for the state that the client resides.
* getZipCode(): A getter for the zip code of the client. Returns a string.
* setZipCode(): A setter for the zip code that the client resides.
* getCreditCardNumber(): A getter for the credit card number of the client’s credit card. Returns a string of 15 digits.
* setCreditCardNumber(): A setter for the credit card number of the client’s credit card.
* getVerificationCode(): A getter for the verification code of the client’s credit card. Returns a string of either 3 or 4 digits.
* setVerificationCode(): A setter for the verification code of the client’s credit card.
* getExpirationDate(): A getter for the expiration date of the client’s credit card. Returns a string of 4 digits in MMYY format, where MM is the month, and YY is the last two digits of the year.
* setExpirationDate(): A setter for the expiration date of the client’s credit card.
* creditCardVerification(): Compares the client’s information on the credit card with the reservation’s information. Returns true if all fields match.
* ~CreditVisa(): Destructor.
* Print(): A function that prints the client’s information and credit card information. This function is used for testing and displaying purposes.

This object will hold information on a client’s American Express credit card. It will be associated with a particular client only when the credit card information is of the type American Express. The credit card numbers will have only 15 digits, and the verification code may have 3 or 4 digits.

### 3.4.16 Object: Date



**Section 3.4.16 Figure 1:** A UML diagram for object Date.

#### ***Attributes***

* year: An integer that holds the year of time.
* month: An integer that holds the month of time.
* day: An integer that holds the day of time.
* leapYear(): An integer that holds the leap year where time resides.
* monthDays(): An integer that holds the total days in a month where time resides.
* monthDays(int): An integer that holds the number of days in a given month, where time resides.

#### Functions

* Date: A constructor that initiates all attributes to the default settings.
* Date(overloaded): An overloaded constructor that initiates all attributes to the user’s input.
* setDate: A setter for the day of the date type.
* getDate: A getter for the day of the date type. It returns an integer.
* setleapYear: A setter for the leap year of the date type.
* getleapYear: A getter for the leap year of the date type. It returns an integer.
* setMonthDays: A setter for the month days of the date type.
* getMonthDays: A getter for the month days of the date type. It returns an integer.
* setMonthDays(int): A setter for the month days of the date type.
* getCountry: A getter for the country of the date type. It returns an integer.
* setZipCode: A setter for the zip code of the date type.
* getZipCode: A getter for the zip code of the date type. It returns an integer.
* setEmail: A setter for the email of the date type.
* getEmail: A getter for the email of the date type. It returns an integer.
* setPhoneNumber: A setter for the phone number of the Date type.
* getPhoneNumber: A getter for the phone number of the Date type. It returns an integer.
* Operator=: Overloads the “=” operator of the discount type to make implementation easier.
* ~ Date: A destructor that resets all attributes to the default settings.

This object will hold all of the general contact information of time making a reservation at the hotel. The majority, if not all, of this information will be mandatory in order for time to be able to reserve a room.

### 3.4.17 Object Relationships



Figure 3.4.17: A UML diagram illustrating the relationships between all the objects in HMS.

## 3.5 Non-functional Requirements

### 3.5.1 Performance

This section is intentionally left blank.

### 3.5.2 Reliability

HMS is has moderate reliability in the hotel data since the database is saved in the system and not in a server. Backups of the data may easily be created upon the user’s wish by duplicating the database files and storing it either in a different hard disk or another storage device. Backups must be created by the user manually; HMS will not create backups for the user.

The data themselves, however, will not be affected if HMS or the system crashes due to any technical issue. Thus, any data already saved will be secure and available in the database in the storage device regardless of the status of HMS or the system.

### 3.5.3 Availability

HMS will be available in any system that has the software installed. The data stored in the database, unique to that system, will be available in the hard disks of the software. The database may be moved from any storage device to another for moving purposes to be made available elsewhere.

### 3.5.4 Security

This section is intentionally left blank.

### 3.5.5 Maintainability

HMS will be fully tested before being released and will not release any updates for maintenance. There will only be one version of HMS released. Any undetected bug leading to nonfunctioning features that causes severe drawback during its normal operation will require the software to be brought back to the developer to be fixed.

### 3.5.6 Portability

HMS is designed to only be used in one system at a time. It will not have functionalities that allow data to be moved around and manipulated in separate systems simultaneously. However, because all the data is saved in a very organized manner in a database, it will indeed be possible to copy the data from one system to another, though manipulation of that data in on system will not affect the data on the other system due to the fact that they two databases will not be connected.

## 3.6 Inverse Functions

Inverse functions are functions that the end user may expect the software to have but the software does not provide. This section will discuss most of the common inverse functions.

### 3.6.1 No Log of Transactions

Due to the amount of the data HMS deals with, it is necessary to keep track of all transactions that occur during interactions between the customers and employees. If the customer makes a purchase but loses all records of that purchase, it is expected from the business to keep a log to record the transaction in case of any errors or discrepancies.

Because of its importance, the user may expect HMS to be able to record all the transactions and save them in the database. HMS does not have this feature. HMS will not record any transactions done from any aspect. It will only store and display data in the proper manner to assist in running a hotel business.

### 3.6.2 Data Backups must be Done Manually

It is important of have backups of data to prevent data loss or corruption. If a guest had reserved a room and expects the reservation to be there when he arrives, it will be extremely inconvenient for the guest if the reservation is lost due to system crash or corruption. This may lead to customer dissatisfaction, a drop in reviews, a reduction in the number of future customers and future revenue.

In order to prevent data loss, it’s ideal to create backups of the database and storing them in multiple places (preferably in other systems for additional safety). The user may expect this feature to be included inside of HMS. However, HMS will not create backups for its data. All backups must be done by the user manually by creating duplicates of the database and storing it in another system or storage device.

### 3.6.3 Lacks Options for Housekeeping

When managing a hotel, it is helpful for the manager to have a tool to organize information about a room’s status and maid’s assignment. When dealing with a few hundred rooms and fifty or so maids, it would become difficult for everyone to keep track of the maid assignments and the status of the rooms. A computer designed tool may be optimal for these types of task organization, in order to keep track of which maid is working on which room or which rooms still haven’t been cleaned, which stay over guests need room service and which guests do not etc.

The user may expect a version of this tool to be integrated into HMS. However, HMS will not handle any housekeeping operations. All housekeeping information must be done externally, either through different software or by hand.

## 3.7 Design Constraints

The hotel management system is an independent system running on the Windows operating system and it is optimized for Windows XP and Windows 7. The system is developed in C++ using Microsoft Visual Studio. The architecture and implementation of the software will be developed with object-oriented design principles under C++ programming standards. All aspects of the software will be modularized as much as possible with these object-oriented design principles. Any modifications of the source code must be made using the Visual Studio 2010 platform toolset v100. The GUI is modeled using Adobe Photoshop 5. The graphical content of the application shall be in the form of .gif or .jpeg files. All persistent data will be stored in relation databases using SQLite v 3.0. The software will be designed so that it is accessible for hotel employees who may have basic computer experience. The software must flow in a reasonable manner, and be simple to use, yet effective in achieving its purpose.

## 3.8 Logical Database Requirements

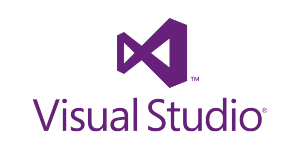
The HMS will maintain a relational database to store persistent information. The database will be using SQLite. The HMS will directly access the database, and processes on the database will include pushing/pulling information, updating information, and creating/deleting information. The HMS will primarily rely on the database to perform its duties. Therefore, the database will be accessed very frequently. The following major data elements (along with their attributes and other information) will be stored in the database:

1. Reservations
2. Rooms
3. Room Type
4. Hotel
5. Features
6. Accounts
7. Discounts
8. Dates

## 3.9 Other Requirements

The main part of the software was coded using Visual Studio 2010 and Visual Studio 2013. The GUI was created using Adobe Photoshop 5 and Adobe Photoshop 6. The relational databases were created using the SQLite software library. The software was created on computers using Microsoft Windows and Apple OSX operating systems.







# 4 Analysis Models

Analysis models are provided in order to analyze data and allow a designer to understand the way many of the functionalities of this software. This section is devoted to analysis models.

## 4.1 Sequence Diagrams

This section is intentionally left blank.

## 4.2 Data Flow Diagrams

### 4.2.1 DFD 1: Interactions between Reservation Components

New Reservation

Get Required Information

Required Information

Get General Information

General Information

Get Extra Information

Extra Information

Store in Database

Finalized Reservation

**Figure 4.2.1.1:** A data flow diagram demonstrating the data process of creating a reservation.

Creating a hotel reservation requires a great deal of information to be obtained before it can be created. Among these include name, address, contact info, credit card information, arrival and departure date, any special requests and more. In order to organize all of this information for reservation, it is ideal to separate them into their own object instances.

When creating a reservation, three main pieces of information are required to successfully create a reservation: a fully completed RequiredInfo object, and partially completed GeneralInfo and ExtraFeatures objects. All of this information must be obtained before the reservation can be completed. Once they have been obtained from the user, the data is accumulated into one single object of Reservation. Finally, the data is sent over to the database for storage. If the dates for that reservation lie between the following 16 weeks, then the data is also saved in memory to depict it in the Main Calendar.

### 4.2.2 DFD 2: Assigning a Room Number inside Reservation

Room Type

String

Find Room Type Object

Room Type  
Object

Calculate  
Room Numbers

Room Numbers

Find Room  
Object

Room Object

Change Status

New Room  
Status

**Figure 4.2.2.1:** A data flow diagram demonstrating the data process of assigning a room number.

When creating a reservation, two of the main pieces of information required to put that reservation to use is the room type and the room number. Since there are only limited number of rooms available of certain room types, the number of rooms open must be calculated before a room number is assigned in the reservation. Once assigned, the status of that room object must be changed so that no other reservations can be created for that room.

The data-flow diagram above illustrates the data movement from one object to another and calculations that take place along the way. The only variable that the Reservation object has at hand is a string representing a certain RoomType object. In order to find rooms of that type that are available, the system then proceeds to find all the room numbers that take the identity of the specified RoomType. Once that information has been processed, the Room object’s status is changed to make it unavailable to other reservations for the dates specified.

### 4.2.3 DFD 3: Adding a New Hotel Employee Inside Accounts

Username

String

Check for Duplicates

Accounts

Object

Add to Vector of Accounts

Accounts

Vector

Add to Database

Accounts Table in DB

**Figure 4.2.3.1:** A data flow diagram demonstrating the data process of adding hotel employees into the system

The username of a new employee account is unique for every account. This restraint enables the system to differentiate between employees and allow for a higher level of security. The system first checks for duplicates among usernames in the vector of Accounts currently in the system. Once this is complete and the username is verified as being unique, all of the information inputted in the fields is aggregated into an Accounts Object. This object will then be placed in the vector of accounts. The new account will also be added to the database, which gives it a permanent space in the HMS.

### 4.2.4 DFD 4: Updating Rates through Rate Management Settings

Rate Change Amount

Double

Find Room Type Objects

Room Type  
Object

Find Base Rates

Base Rate

Calculate New Rate

New Rate

Rate Change Amount

Double

Figure 4.2.4: A data flow diagram for data process of updating rates through rate management settings

When the user wants to change the room rates of the hotels, he will need one vital piece of information. This is the amount that he wants the rates to change from their base rates.

Since all of the rates of the rooms are stored as a base rate variable in the database, changes to the rates are made via modifiers. For example, if the day’s rates require a lower price by $10, this will be the modifier to be entered into the system. The system will find the base rate for the particular room type, apply the rate change modifier, and return the new rate for the day.

The data-flow diagram above illustrates the data movement from one object to another and calculations that take place along the way. The only variable that the RoomType object has at hand is a double signifying the base rate. The only information that the user needs to change the daily rate is the modifying amount.

## 4.3 State-Transitional Diagrams

This section is intentionally left blank.

# 5 Graphical User Interface Menus

## 5.1 Login Screen

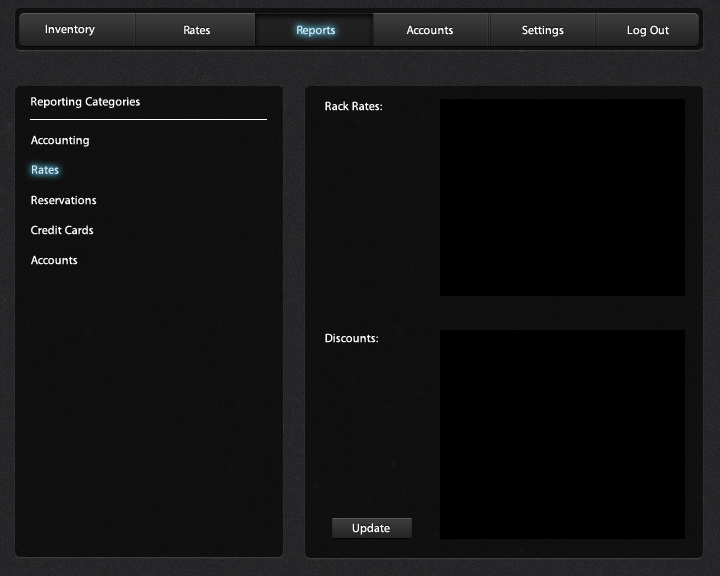


## C:\Users\Valkyrie\Downloads\Add-Add.jpg5.2 Settings: Inventory

## C:\Users\Valkyrie\Downloads\Reports-Accounting-New.jpg5.3 Settings: Repots: Accounting

## C:\Users\Valkyrie\Downloads\Reports-accounts-New.jpg5.4 Settings: Reports: Accounts

## 5.5 Settings: Reports: Rates



## C:\Users\Valkyrie\Downloads\Reports-Reservations-NEW.jpg5.6 Settings: Reports: Reservations

## C:\Users\Valkyrie\Downloads\Accounts.jpg5.7 Settings: Accounts

## C:\Users\Valkyrie\Downloads\Main-Menu-New6.jpg5.8 Room Calendar

# 6 Change Management Process

Any changes to this document may be made after approval from the project manager. The changes must be communicated to the project manager, and then to the rest of the team. Evaluations of the SRS will be done every week as changes in design and requirements progress.

For any changes requested by stakeholders, there is a special approval process involved. The stakeholder must communicate the requested change to the project manager. This can be done via email or an in person meeting. The manager will then assess the changes with the team and a consensus must be reached for the changes to be approved. Once approved, the manager must then sign off the changes to confirm the approval. Only when both signatures are present on the change request form will the changes become approved.

## Document Approvals

Stakeholder

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project Manager

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_