EECE 419 - Pod 1

Hotel Bookings Requirements Specification

November 29, 2009

Sean Clark	36538056
Yang Gao	52588050
Shen Li	65962060
Neil Gentleman	62973029
Arash Malekzadeh	33685058
Michael Tando	79529061
Wei-Chen Wang	64341043

Table of Contents

Illustration Index

Glossary

AJAX   An approach that is used to design and implement web applications

Apache   A web server software that provides great environment and features for web designing

Client   An interface that accesses a remote service

Firefox   An open source web browser that is used to display web pages

Server   A machine on a network that is used to connect participants together

User Interface   An interface that allows user to interact with computer device, system, or program

1.0 Introduction

1.1 Purpose

The purpose of this document is to explain different aspects of software requirements regarding a hotel reservation software. The scope of this document is limited to the main functional and non-functional requirements of the system. These functions are defined as those required to have a stable and usable hotel reservation system. Extra features designed for different hotels will not be mentioned in this document.

1.2 System Overview

X-Reserve is a hotel reservation system that allows hotels to provide guests with a web-based reservation interface. There are a number of features available to both clients and the hotel management or staff. A client can create an account and use it to view available rooms for different dates. The room search function provides users the ability to find a room based on the price, date, room layout, capacity and other options. As for the management side, there are features designed for managers, staff and maids. Hotel managers can view reports and graphs on reservation trends, and be able to maximize their profits. They can add customized room layouts to the database. The hotel staff can make reservations for guests, add chargeable items to rooms, and set special prices ratest. Rooms will automatically be marked so that maids can keep track of which room is to be occupied and needs cleaning.

1.3 Project Scope

X-Reserve is to be designed as a web-based application which will only operate with one specific database. This software can be deployed for different hotels, and It will allow users around the world to make reservations. The goal of this software is to provide a reliable and fully customizable reservation system for any type of hotel within Canada and United States.

2.0 Assumptions

In order to simplify development and testing, we have decided to only support Firefox version 3.5 and up. Users of our software should be familliar with the navigation of websites and simple web applications, such as Google Maps or Gmail. Our interface will be designed around the current UI norms and should be intuitive enough to use without any instruction. Installation of X-Reserve is too technical for normal users, but we will provide enough documentation to make the process easy for people familiar with installing server-side software. To ease installation, our software will useable without any external database or servlet container. It will also be able to take advantage of pre-installed servlet containers and database servers, such as Apache Tomcat and MySQL, to enhance scalability. We will do our best to minimize resource usage of X-Reserve, and it will be usable on limited hardware with a small number of users.

3.0 System Features

3.1 Account Management and Security (Access Control)

X-Reserve is designed to enable role-based access control to organize privileges. It provides authentication and authorization of multiple customizable roles such as administrator role, staff role, and user role. This provides a secure and effective way to manage access to different information and resources.

3.2 AJAX Usability Enhancement

X-Reserve is further enhanced using AJAX approach. Using this approach, X-Reserve is able to provide improved user-experience by presenting web content dynamically without the need of page refresh. This decreases the bandwidth consumption and user delay, thus significantly improves the efficiency of work flow and overall productivity of the users.

3.3 Reservation Management

X-Reserve allows certain roles such as administrator and staff to manage customer reservations efficiently. The administrator and staff can easily view current reservations, modify or delete reservations. They can also manage customer check-in and check-out.

3.4 Business Report and Analysis

X-Reserve allows the administrator to generate business reports easily and quickly. The administrator can analyze reports and graphs on reservation trends, and be able to maximize their profits. For example, the administrator can generate a room type statistic report that presents most recent data on room type preferences from latest customer reservations.

3.5 Easy Hotel Facility Management

X-Reserve provides hotel administrators an easy way to manage hotel facilities. They can add customized room types with different room layouts and attributes, such as number of beds, to the database. The hotel staff can make reservations for guests, add chargeable items to rooms, and adjust room rates depending on the season.

3.6 Web-based, No Installation Required

X-Reserve is designed as a web-based application. No installation is required in order to run X-Reserve; however, an internet connection is required. X-Reserve can be up and running 24/7, providing services to customer and hotel management staffs.

3.7 Advanced Search with Attributes

Using X-Reserve, customers will be able to easily search room facilities according to their preferences. Customers can quickly search for rooms by specifying the type of room, the price range, and start/end date of their stay. Furthermore, with the advanced search, customers can, for example, specify the number of beds, number of bathrooms, smoking/non smoking, and etc. The advanced search will provide the customer with utmost satisfaction.

4.0 System Architecture

4.1 Design

X-Reserve is using server-client architecture. The client of X-Reserve should be any browser that supports the latest standard from W3C. However, due to the limited resource we had during the project, we only tested X-Reserve using Firefox 3.5. The following will assume a compatible browser is given, and it only focuses on the design of the server.

4.2 MVC

The server is designed using a popular architectural pattern called the Model-View-Controller (MVC). The MVC pattern isolates the business logic from the presentation layer, and allows independent development and maintenance. The MVC pattern uses three basic components, as explained below.

Model represents data that is specific to the domain model. For example, a reservation can be represented as a model.

View represents the Hypertext Markup Language (HTML) pages that a user can see. For example, a page displaying information about a particular reservation can be a view.
Controller represents all the control logics in the application. In general, a controller receives a request; it invokes some methods on the model, and responds with a different view.
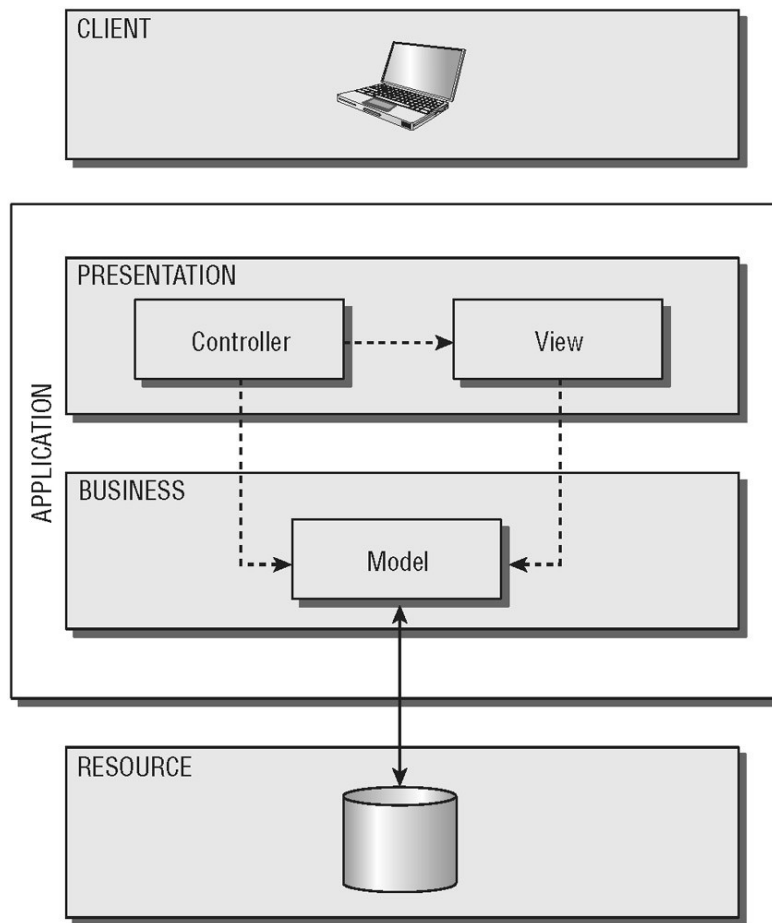
Figure 1: System Architecture

4.3 Spring MVC

Spring is a java based frame work that provides a lot of services for enterprise web applications. Spring MVC provides all the fundamental infrastructures to allow easy implementation of the MVC pattern.

The DispatcherServelet is the first place where all the HTTP requests will get handled. The DispatcherServlet will use the prefix of the request path to determine which Controller should handle the request, and this is done by delegating the responsibility to a SimpleURLHandlerMapping handler. The SimpleURLHandlerMapping will look up the Controller based on a pre-configured map.

Once the HTTP request reaches the corresponding Controller, a collection of Validators will be called to verify the correctness of the request. After the request is validated, the controller will manipulate the models, and returns a view. A ViewResolver is then called to lookup the view. Once the correct View is found, the View will render itself using the models given, and return a complete HTML page.

6

The relations between various components in the Spring MVC are shown below.
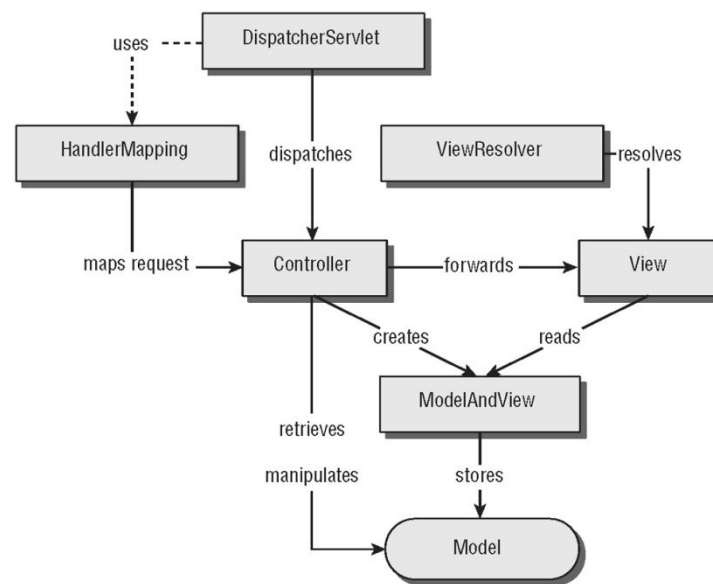
Figure 2: Spring MVC

4.4 Request Flow

There are two types of requests served by the server, static requests and servlet request. For a static request, the static content is returned directly without involving any Controllers. This type of request is useful for static contents such as, CSS, JavaScript files, and images. All static requests starts with "/static". The other type of request is the servlet request, which represents all requests that return dynamic pages. For example, a request to make a reservation will be a servlet request. Servlet requests always starts with "/entityname". For example, a request to make a reservation will start with "/reservation". By using this convention, a SimpleURLHandlerMapping handler can be used to look up the right controller.

 For the complete request flow, see the diagram below.
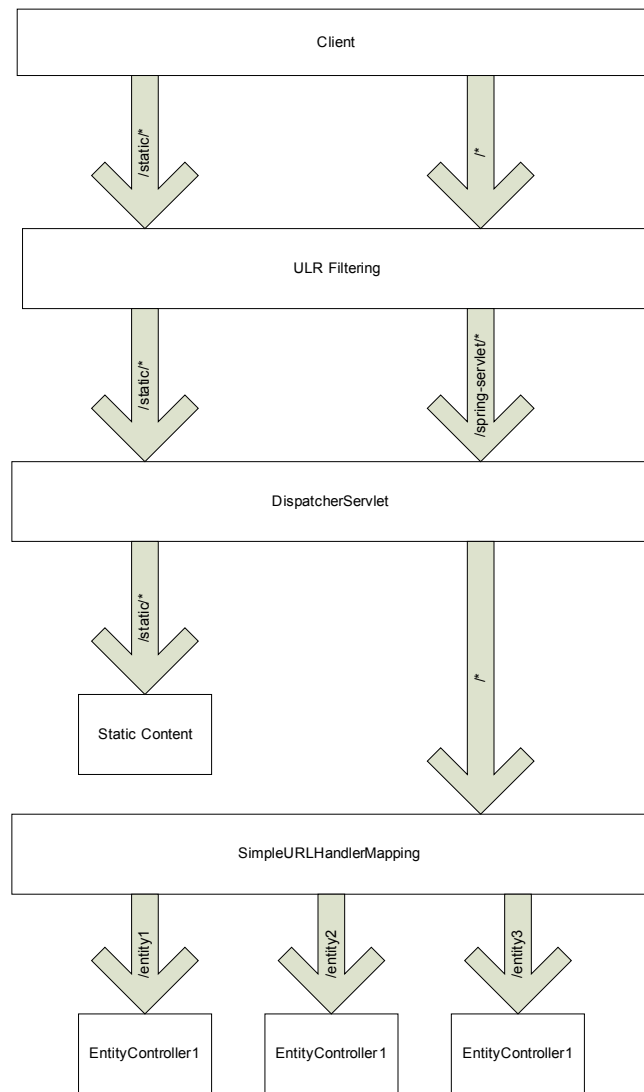
Figure 3: Request Flow

4.5 Input Validation

Any system that is as robust and reliable as X-Reserve will implement some mechanism to validate user input before persisting them into the database. With the Spring MVC, this can be easily achieved by adding some Validators to the Controller.

The Validator checks the input, and returns a list of errors. Each error contains a description of the error, and it can be used as an error message to the user. Based on the output of the Validator, the Controller will decide which View to return.
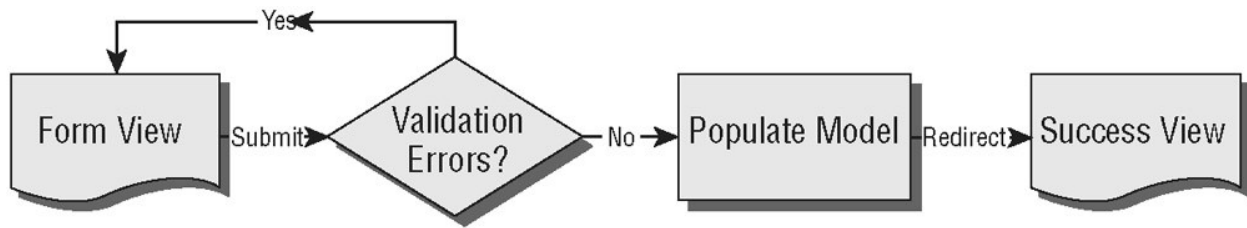
Figure 4: Validation Flow

4.6 Persistence

X-Reserve uses the Java Persistence API (JPA) to persist all the objects. JPA automatically manages the mapping between Java objects and the relational database, and it is the standard to persistence mechanism used in enterprise Java applications. JPA can translate Plain Old Java Objects (POJO) into Structured Query Language (SQL). The JPA provider used by X-Reserve is Hibernate.
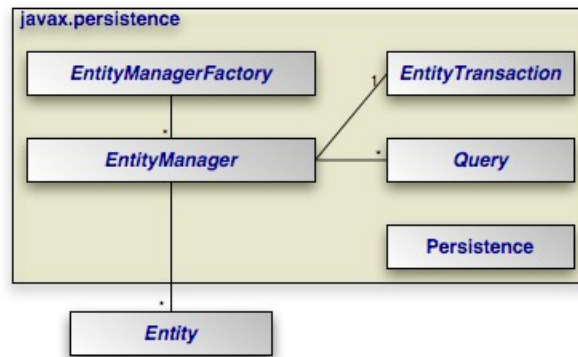
Figure 5: Persistence API

5.0 Non-Functional Requirements

The following section will describe our non-functional requirements in detail. They are broken down into five sections -- usabillity, scalability, security, portability and performance. Our key constraints are usability, security and performance.

5.1 Usability

The target clients of the software are hotel managers, receptionists, maids, and hotel customers. Most of the clients may know little about computers and software systems. As a result, the software interface takes a very important role. The software has to provide flexible and easy-to-use interfaces so that our clients can achieve reasonable success with little training. The software is a web-based application running on Mozilla Firefox 3.5, and the interface will be designed and constructed following the Nielsen's Ten Usability Heuristics. The software will expose different functionality depending on different roles of users. For example, if the login user is identified as a manager, the manager will see an interface with the viewing report option. The details of these interfaces will be discussed further in the functional requirements. Moreover, all these different interfaces will be consistent and will follow the same standards. The goal of these interfaces is to minimize user confusion and increase the usability of our software.

5.2 Scalability

X-Reserve is only intended to manage one hotel, not a chain, so our ultimate scalability goals are limited. The system will, however scale to the largest of single hotels, with hundreds of of rooms. In this process, customer volumes are expected to scale to several thousands customers. The objective for this requirement is to cater for future growth in the business. The system also needs to provide access for a number of terminals that can be used by the hotel personnel. Eventually, there could be over twenty terminals in a hotel and thus the system should be able to handle over twenty simultaneous requests. This requirement is to allow the system to be used across the whole hotel by the hotel personnel at any time. The potential for database sharding or application server clustering will not be explored.

5.3 Security

Since our software will be storing sensitive personal information, like credit card information and addresses, security is essential. We are building on top of the Spring framework, which provides a secure base to work from. It provides authentication that will be used throughout the software to hide information from the wrong users. We will also be actively looking for bugs that could cause a security hole or information leak.

5.4 Portability

To ease installation and broaden our customer base, our software will be portable across a variety of operating systems, including Linux, OS X, and Windows. To simplify development, we will only be testing on Linux and Windows; however, our software should run without problems on any system with Java installed.

5.5 Performance

Performence is an important requirement for X-Reserve, especially due to its scalability. A system with our minimum requirements (1Ghz processor, 1GB of RAM) will be able to support up to 10 simultaneous users. The actual performance varies depending on a number of factors, including network latency and the capabilities of our user's computers. On an internal network, with relatively new computers (at least 600MHz with 256MB of RAM), X-Reserve will perform quite well. Normal operations and page refreshes will complete inside of 1 second, and more complicated operations (like statistical reports) will take at most 5 seconds.

5.6 Reliability

Since our software will be operating 24 hours a day and 7 days a week, reliability is a very important requirement to our system. All our system's data will be stored into a database to prevent data corruption or data loss due to computer failure. In addition, to increase the reliability of our system, we will perform a lot of testing on the software. We will perform testing on each of the system features with both valid inputs and invalid inputs. During all the testing phases, software bugs will be discovered, corrected, and re-tested. As a result of all this testing, our system is aiming to a very small failure rate or even a zero failure rate.

5.7 Robustness

Users' inputs are always unpredictable and uncontrollable. They may enter invalid inputs to our system. As a result, our software has to validate all inputs to prevent system failure from unexpected inputs. That is, our software will provide robustness. All the input fields will be validated and if an unexpected input is encountered, a warning message will be displayed.

5.8 Efficiency

Our software will be very efficient in term of resource utilization. Our system uses the Jetty web server, which is a very light weight web server. It is much smaller and more efficient than other web servers such as Tomcat. In addition, we choose Spring as our framework for developing, which is also a light weight framework. It is very easy for programmers to develop. Since this software is a web based application, as a user, everything he/she needs is a browser and he/she can run the application immediately.

5.9 Adaptability

Our software is implemented in Java and has been tested on Windows, Linux and Mac OS X. Also, our software is developed with Spring framework. One of its advantages is dependency injection, which is the ability of providing an external dependency to the software. It provides flexibility and adaptability to our software because it can create alternative services via a configuration file instead of changing our software.

6.0 Key Constraints

X-Reserve is developed as a reliable and fully customizable reservation system; however, there are some key constraints that must be followed. In order to run this software, users require the following applications: Java 1.6 and FireFox 3.5. Since this software is a web-based application, it requires users to have internet access. X-Reserve is our first prototype; it has not been modified to handle large load; in other words, it cannot be clustered. In term of security, even though we do provide a secure on top of Spring framework base to work from, we are unable to prevent any brute force attack.

7.0 Functional Requirements

This section contains descriptions and diagrams of our man functional requirements. There are also detailed use cases and sequence diagrams, which describe our user interactions and the sequence of messages needed to complete them.

7.1 Class Diagram

The class diagram, shown in Figure 6, shows the relationship between all of the classes we will be using to implement our software.

7.2 Test Diagram

The test diagram, shown in Figure 7, illustrates the relationships between Mock objects and our test controllers.

7.3 Controller Diagram

The controller diagram shown in Figure 8, shows all of our controllers and how they relate to each other.

7.4 Repository Diagram

This section provides an overview of a GenericDao class that implements GenericRepository. The extended GenericDao class forms a JDBC-abstraction layer that removes the need to do tedious JDBC coding and parsing of database-vendor specific error codes. Figure 6 below illustrates the functions associated with the GenericDao class and its subclasses.
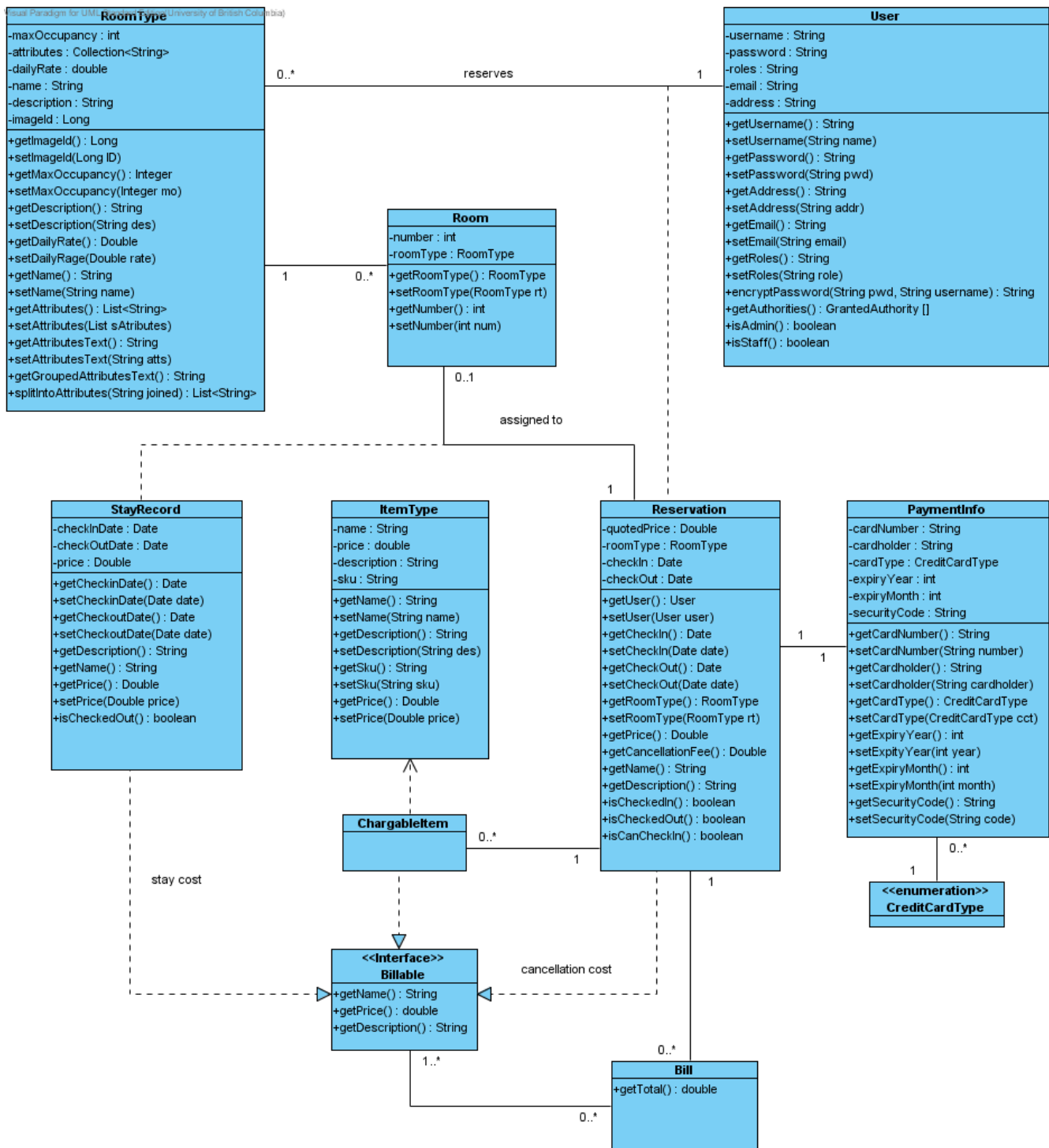
RoomType

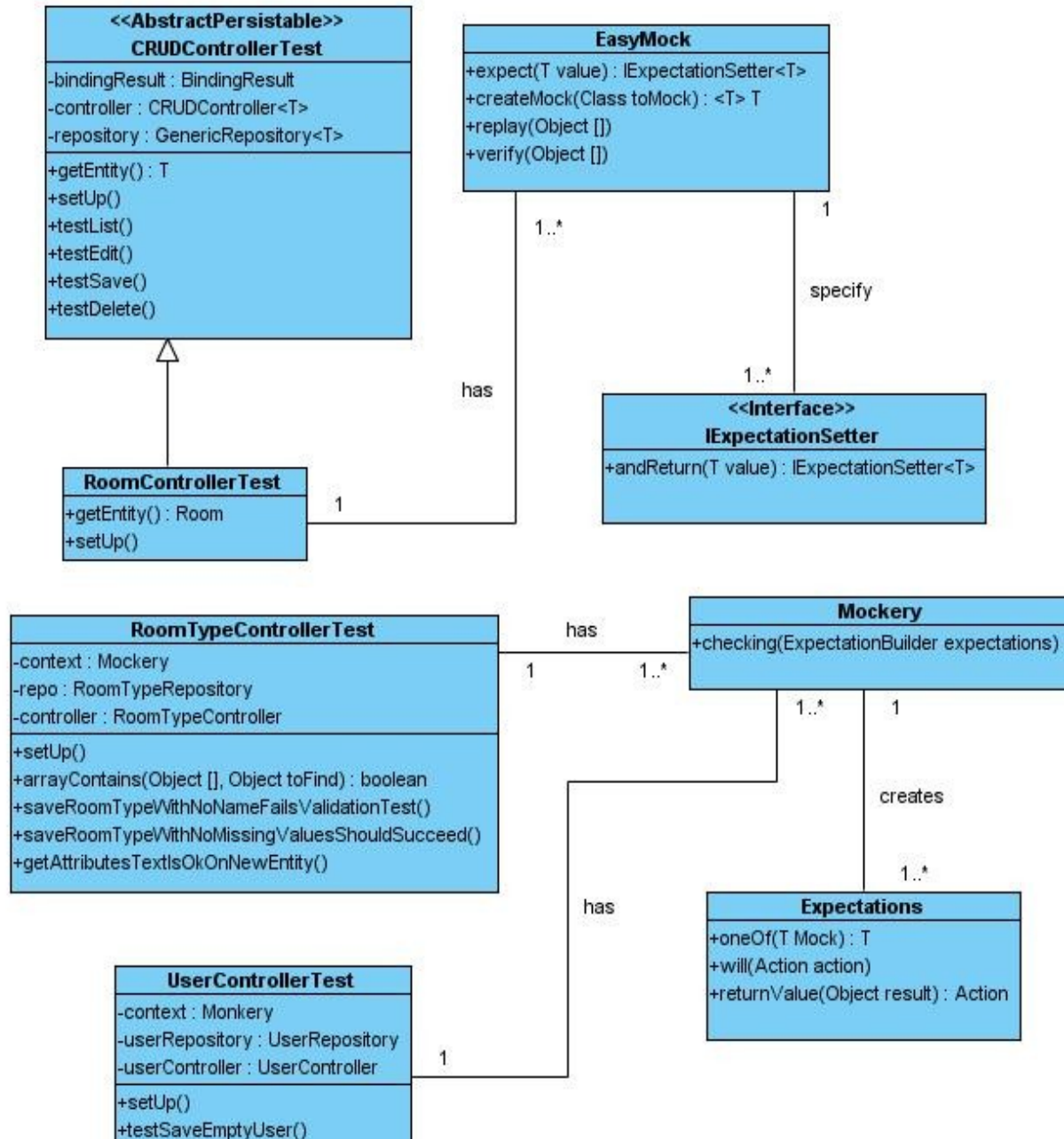-maxOccupancy : int
-attributes : Collection<String>
-dailyRate : double
-name : String
-description : String
-imageId : Long

+getImageId() : Long
+setImageId(Long ID)
+getMaxOccupancy() : Integer
+setMaxOccupancy(Integer mo)
+getDescription() : String
+setDescription(String des)
+getDailyRate() : Double
+setDailyRage(Double rate)
+getName() : String
+setName(String name)
+getAttributes() : List<String>
+setAttributes(List sAtributes)
+getAttributesText() : String
+setAttributesText(String atts)
+getGroupedAttributesText() : String
+splitIntoAttributes(String joined) : List<String>

User

-username : String
-password : String
-roles : String
-email : String
-address : String

+getUsername() : String
+setUsername(String name)
+getPassword() : String
+setPassword(String pwd)
+getAddress() : String
+setAddress(String addr)
+getEmail() : String
+setEmail(String email)
+getRoles() : String
+setRoles(String role)
+encryptPassword(String pwd, String username) : String
+getAuthorities() : GrantedAuthority []
+isAdmin() : boolean
+isStaff() : boolean

0..*   reserves   1

Room

-number : int
-roomType : RoomType

+getRoomType() : RoomType
+setRoomType(RoomType rt)
+getNumber() : int
+setNumber(int num)

1   0..*

0..1

assigned to   1

StayRecord

-checkInDate : Date
-checkOutDate : Date
-price : Double

+getCheckinDate() : Date
+setCheckinDate(Date date)
+getCheckoutDate() : Date
+setCheckoutDate(Date date)
+getDescription() : String
+getName() : String
+getPrice() : Double
+setPrice(Double price)
+isCheckedOut() : boolean

ItemType

-name : String
-price : double
-description : String
-sku : String

+getName() : String
+setName(String name)
+getDescription() : String
+setDescription(String des)
+getSku() : String
+setSku(String sku)
+getPrice() : Double
+setPrice(Double price)

Reservation

-quotedPrice : Double
-roomType : RoomType
-checkIn : Date
-checkOut : Date

+getUser() : User
+setUser(User user)
+getCheckIn() : Date
+setCheckIn(Date date)
+getCheckOut() : Date
+setCheckOut(Date date)
+getRoomType() : RoomType
+setRoomType(RoomType rt)
+getPrice() : Double
+getCancellationFee() : Double
+getName() : String
+getDescription() : String
+isCheckedIn() : boolean
+isCheckedOut() : boolean
+isCanCheckIn() : boolean

PaymentInfo

-cardNumber : String
-cardholder : String
-cardType : CreditCardType
-expiryYear : int
-expiryMonth : int
-securityCode : String

+getCardNumber() : String
+setCardNumber(String number)
+getCardholder() : String
+setCardholder(String cardholder)
+getCardType() : CreditCardType
+setCardType(CreditCardType cct)
+getExpiryYear() : int
+setExpityYear(int year)
+getExpiryMonth() : int
+setExpiryMonth(int month)
+getSecurityCode() : String
+setSecurityCode(String code)

1
1

ChargableItem

0..*

1

stay cost

cancellation cost

<<Interface>>
Billable

+getName() : String
+getPrice() : double
+getDescription() : String

1..*

0..*

Bill

+getTotal() : double

0..*

0..*

1

<<enumeration>>
CreditCardType

Figure 6: Class Diagram

14

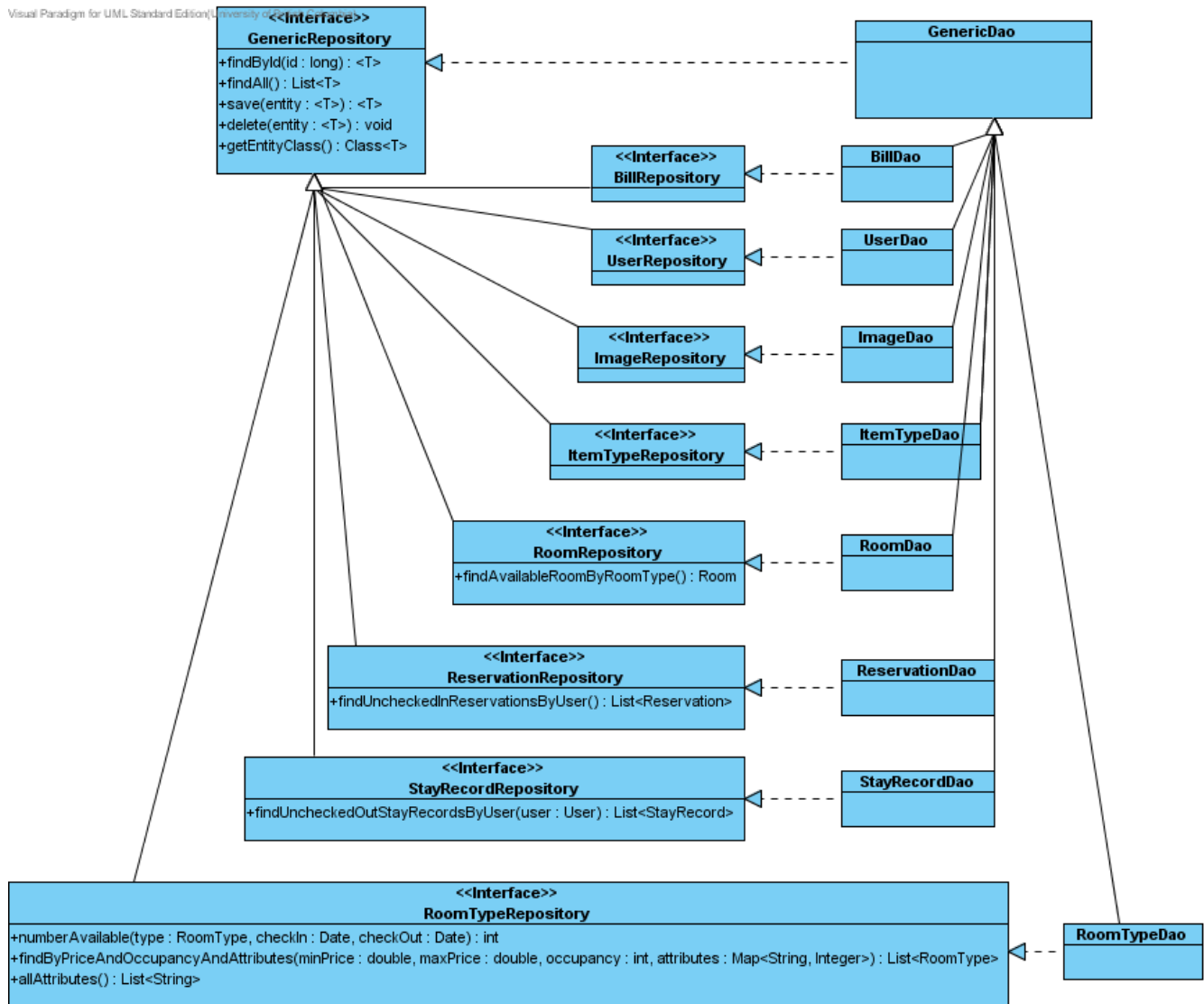*Figure 7: Test Diagram*

Figure 8: Controller Diagram

<<AbstractPersistable>>
CRUDController
+CRUDController()
+getNewEntity() : T
+getRepository() : abstract GenericRepository<T>
+getEntityName() : Class<T>
+exposeCurrentUser() : User
+index() : ModelAndView
+list() : ModelAndView
+edit(Long id) : ModelAndView
+editView(T entity) : ModelAndView
+redirectToListView() : ModelAndView
+setValidators(List validators) : void
+getValidators() : List<Validator>
+addValidator(Validator validator) : void
+save(T bound, BindingResult errors, String view) : ModelAndView
+delete(Long id) : ModelAndView
+hasError(T bound, BindingResult errors) : boolean

AJAXController
+exposeCurrentUser() : user
+listImage() : ModelAndView

ChartController
+index() : ModelAndView

PropertyEditor
+getAsText()
+setAsText()

EntityEditor
+EntityEditor(GenericRepository repo)
+getAsText() : string
+setAsText(String text) : void

<<Interface>>
Controller
+AbstractController()
+AbstractFormController()
+AbstractWizardFormController()

ItemTypeController
+ItemTypeController()
+getRepository() : ItemTypeRepository

WelcomeController
+handleRequestInternal(HttpServletRequest request, HttpServletResponse response) : ModelAndView

ImageController
+addByteArrayBinder(WebDataBinder binder) : void
+getRepository() : GenericRepository<Image>
+view(Long id, ServletResponse response) : void

CheckOutController
+postProcessPage(HttpServletRequest request, Object command, Errors errors, int page)
+CheckOutController()
+handleRequest(HttpServletRequest request, HttpServletResponse response) : ModelAndView
+processFinish2(HttpServletRequest request, HttpServletResponse response, Object command, BindException errors) : ModelAndView
+validatePage(Object command, Errors errors, int page) : void
+processFinish(HttpServletRequest request, HttpServletResponse response, Object command, BindException errors) : ModelAndView

UserController
+UserController()
+delete(Long id) : ModelAndView
+save(User bound, BindingResult errors, String view) : ModelAndView
+edit(Long id) : ModelAndView
+getRepository() : UserRepository
+login(Integer error) : ModelAndView
+register(User user, BindingResult errors) : ModelAndView
+registerForm(User user) : ModelAndView

CheckInController
+CheckInController()
+handleRequest(HttpServletRequest request, HttpServletResponse response) : ModelAndView
+postProcessPage(HttpServletRequest request, Object command, Errors errors, int page) : ModelAndView
+validatePage(Object command, Errors errors, int page) : void
+setStayRecordRepository(StayRecordRepository stayRecordRepository) : void
+setUserRepository(UserRepository userRepository) : void
+setRoomRepository(RoomRepository roomRepository) : void
+setReservationRepository(ReservationRepository reservationRepository) : void

RoomController
+RoomController()
+registerRoomTypeEditor(WebDataBinder binder) : void
+exposeRoomTypes() : List<RoomType>
+getNewEntity() : room
+getRepository() : RoomRepository
+edit(Long id) : ModelAndView
+delete(Long id) : ModelAndView
+save(Room bound, BindingResult errors, String view) : ModelAndView

BaseWizardFormController
+handleRequest(HttpServletRequest request, HttpServletResponse response) : ModelAndView

RoomTypeController
+RoomTypeController()
+getRepository() : RoomTypeRepository
+Search() : static class
+search(Search search, Errors errors) : ModelAndView
+edit(Long id) : ModelAndView
+delete(Long id) : ModelAndView
+save(RoomType bound, BindingResult errors, String view) : ModelAndView

ReservationWizardController
+ReservationWizardController()
+handleRequest(HttpServletRequest request, HttpServletResponse response) : ModelAndView
+bindGetSearchParams(HttpServletRequest request, ReservationForm reservationForm) : void
+referenceData(HttpServletRequest request, Object command, Errors errors, int page) : Map<String, Object>
+validatePage(Object reservationData, Errors errors, int page) : void
+getViewName(HttpServletRequest request, Object command, int page) : String
+processFinish(HttpServletRequest request, HttpServletResponse response, Object command, BindException errors) : ModelAndView
+processCancel(HttpServletRequest request, HttpServletResponse response, Object command, BindException errors) : ModelAndView

ReservationController
+ReservationController()
+getRepository() : ReservationRepository
+ReservationForm() : static class
+reserve(ReservationForm reservationForm, Errors errors) : ModelAndView
+delete(Long id) : ModelAndView
+edit(Long id) : ModelAndView
+save(Reservation bound, BindingResult errors, String view) : ModelAndView

Figure 9: Repository Diagram

7.5 Use Cases

This section describes the use cases for our project. The use case diagram in Figure 10 shows the use cases and their relation to the different types of users of our project. The rest of this section explains the use cases in detail, with sequence diagrams explaining the flow of messages.
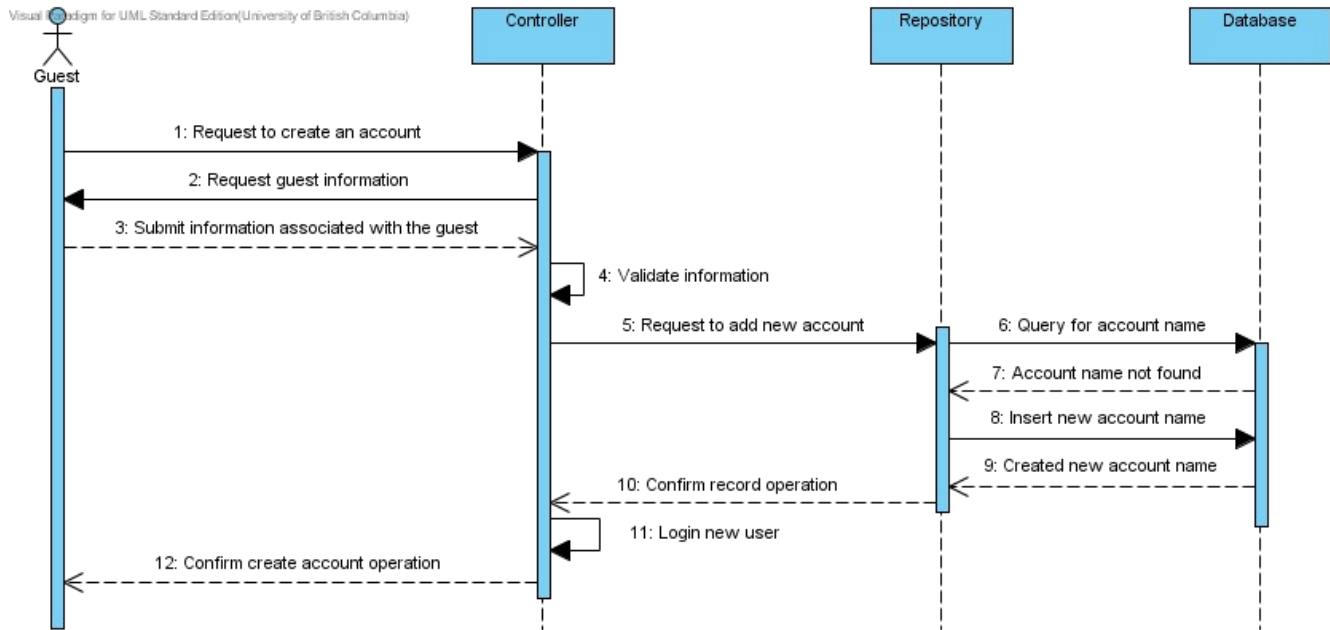

Figure 10: Use Case Diagram

Figure 11: UC1 - Create an Account

Create an account

Use case number: UC1

1.Guest enters his email address, password, credit card information and other relevant information

2.System validates all the information is correct

3.System records all the information and logins the guest
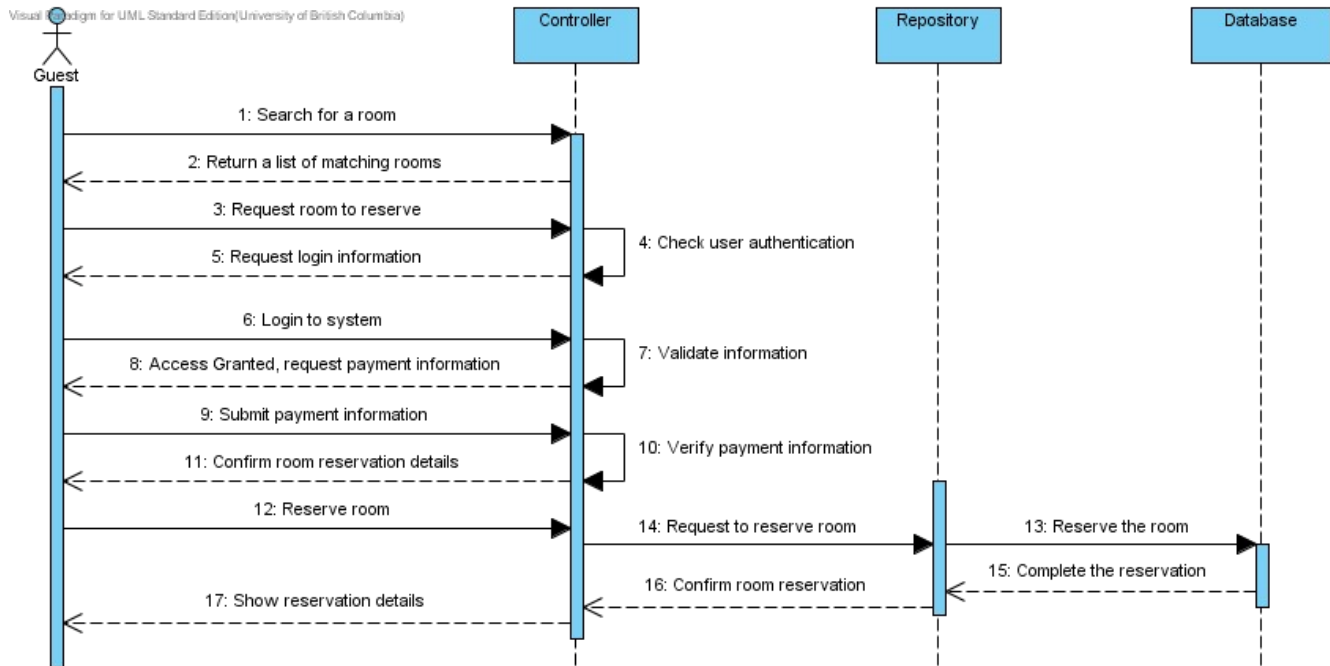
Figure 12: UC2 - Make a Reservation

Make a reservation

Use case Number: UC2

1. Guest or Staff enters the check-in and check-out dates, room capacity, and price range to the system.

2. The system will return a list of rooms.

3. User selects a room from this list to reserve.

4. If guest, guest logs on to retrieve her already stored customer information.

5. If guest does not have an user account, include UC1

6. User selects the payment type (Visa or MasterCard)

7. User confirms the reservation

8. The system generates a confirmation number and a reservation summary
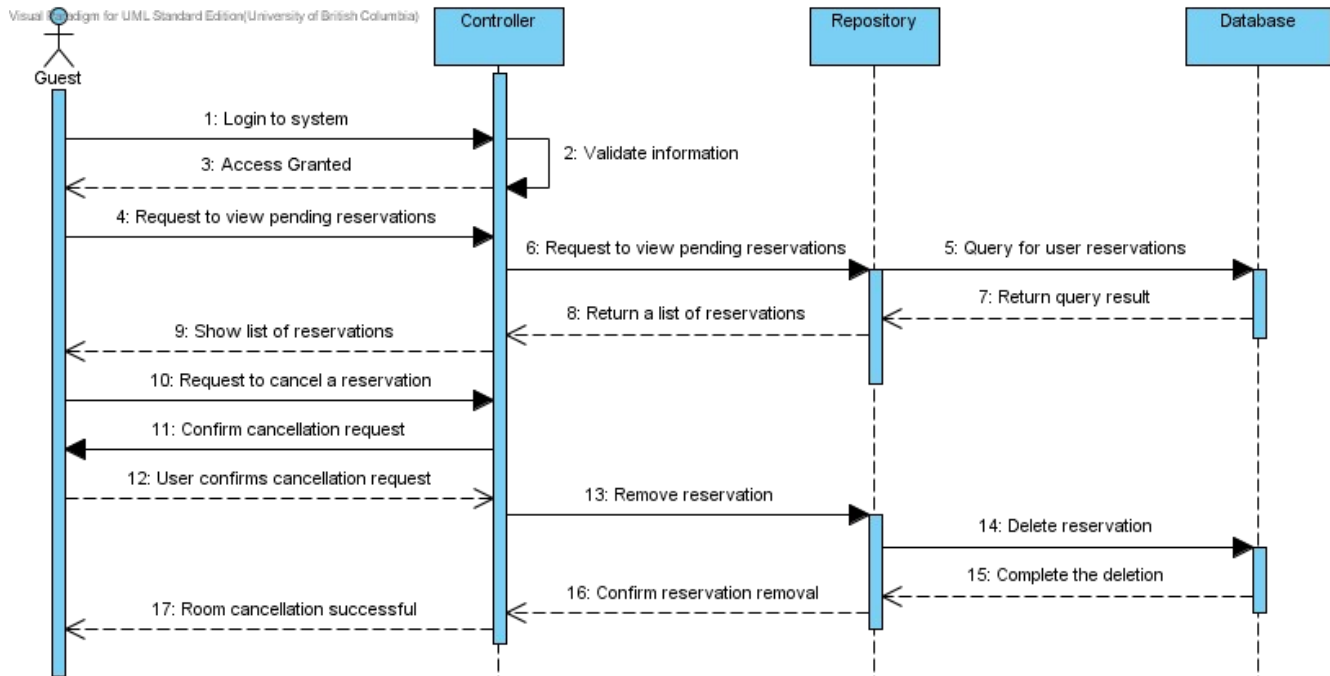
Figure 13: UC3 - Cancel a Reservation

Cancel a reservation

Use case Number: UC3

1. Guest logins to his account

2. Guest requests to view all the reservations

3. System responds with a list of reservations

4. Guest requests to cancel one of listed reservation

5. System validates that reservation can be cancelled

    5.1. System will display an error if the reservation cannot be cancelled

6. System displays any cancellation fees to the guest

7. Guest confirms to proceed with the cancellation

8. System confirms the reservation has been cancelled
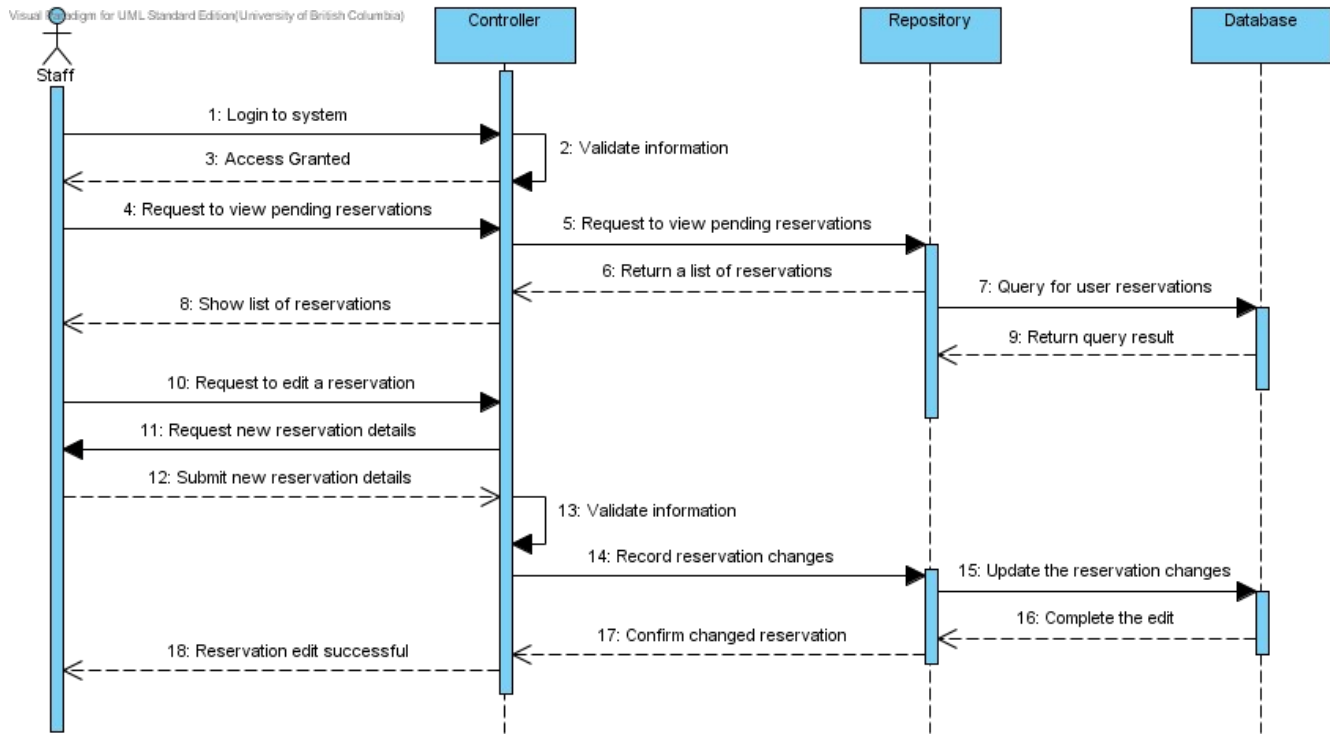
Figure 14: UC4 - Edit a Reservation

Edit a reservation

Use case Number: UC4

1. Guest logins to his account

2. Guest requests to view all the reservations

3. System responds with a list of reservations

4. Guest requests to edit one of listed reservations

5. System prompts the user to enter new reservation details

6. Guests enter new reservation details

7. System validates the new input
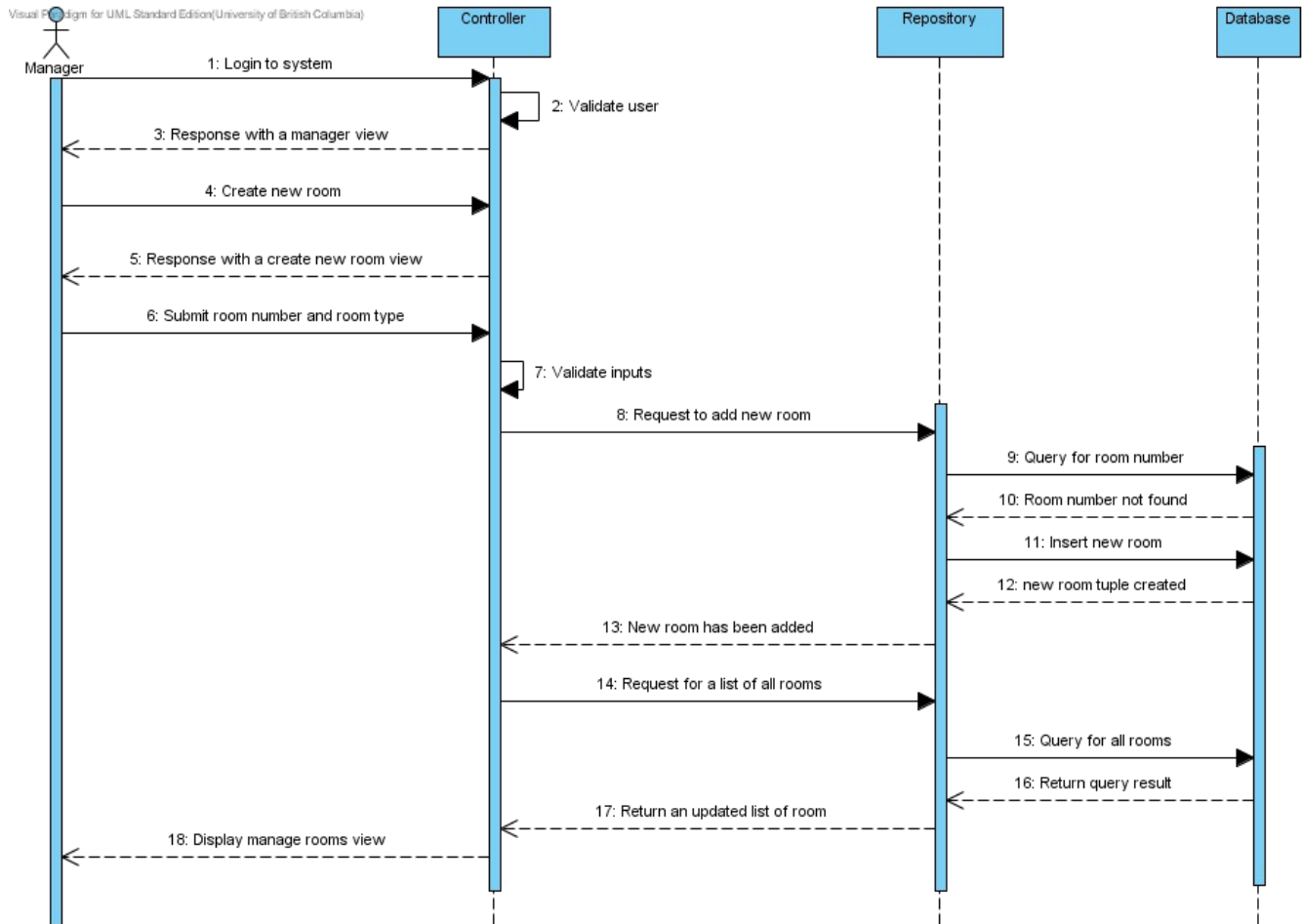
Figure 15: UC5 - Create New Room

Create new room

Use case Number: UC5

1. Manager logs into the system using his or her account

2. Manager requests to create a new room

3. Manager specifies a room number and room type for the new room

4. System validates all the information entered is correct

   4.1. System displays an error message if the information is incorrect

5. System adds the new room to the database
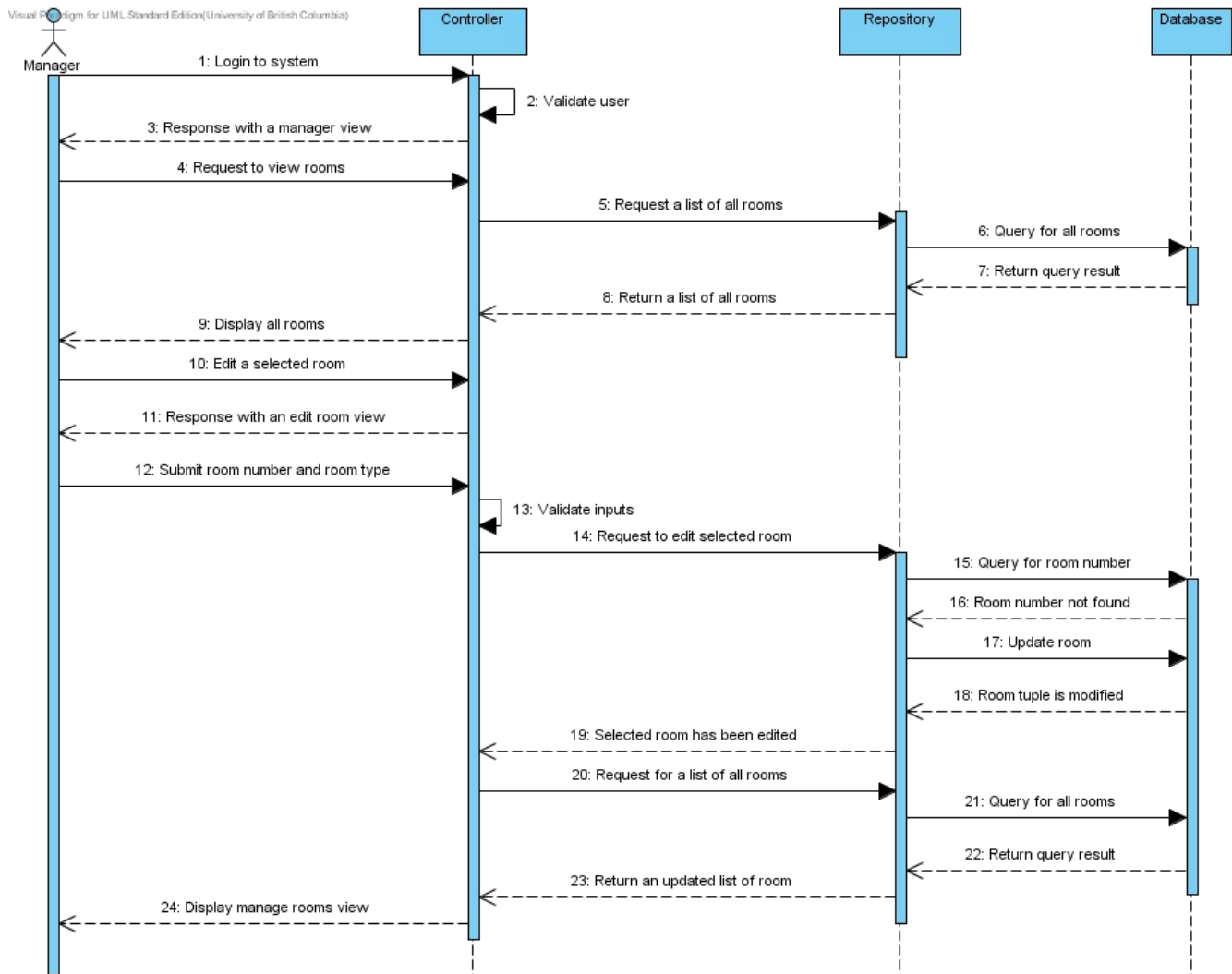
Figure 16: UC6 - Edit Room

Edit room

Use case Number: UC6

1. Manager logs into the system using his or her account

2. System displays list of all existing rooms

3. Manager selects a room to edit from the list of all existing rooms

4. Manager can change room type, room number and any information that should overwrite the defaults

5. System validates all the information entered is correct

    5.1. System displays an error message if the information is incorrect

6. System saves the changes to the room in the database
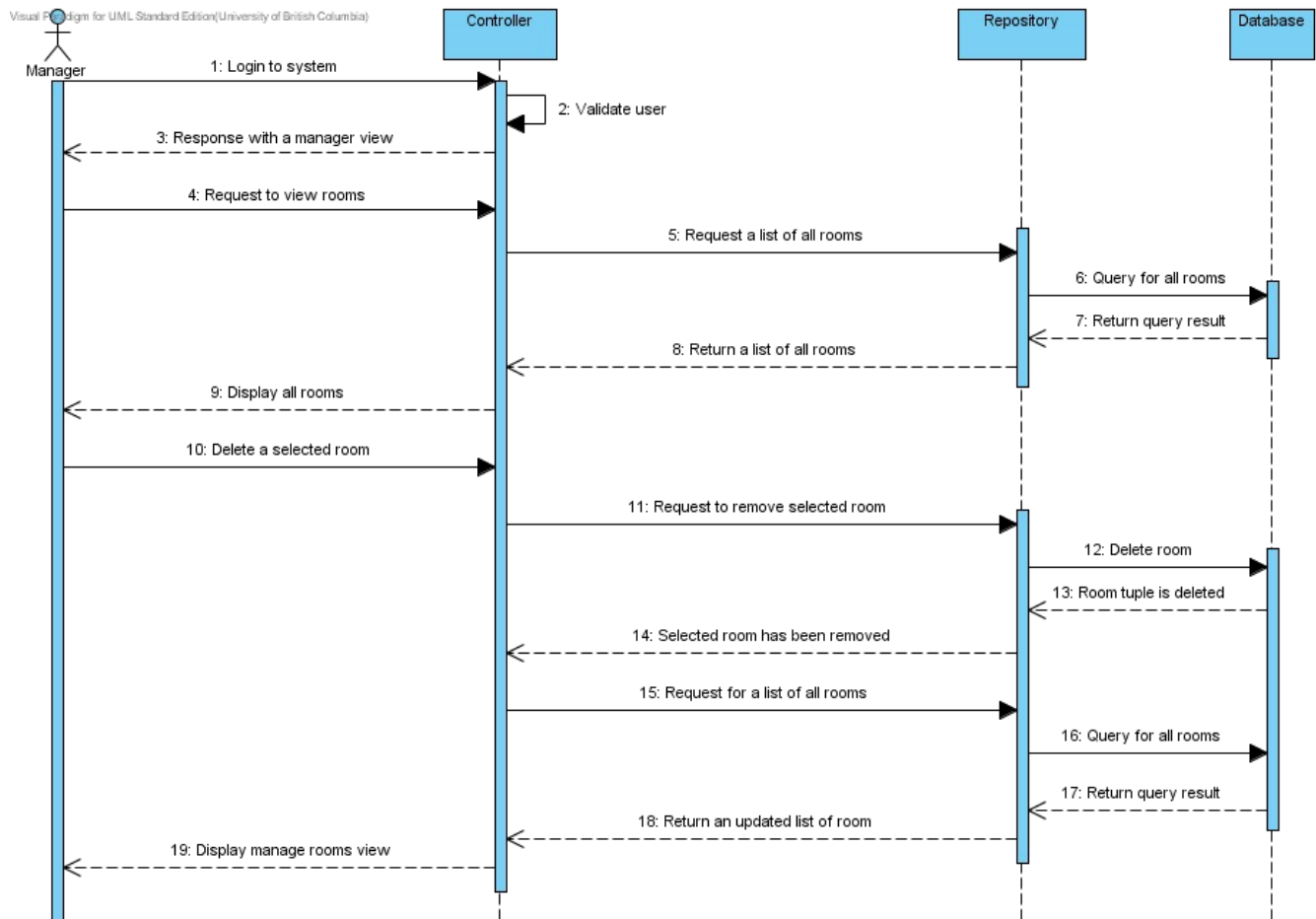
Figure 17: UC7 - Delete Room

Delete room

Use case Number: UC7

1. Manager logs into the system using his or her account

2. System displays list of all existing rooms

3. Manager selects a room to delete from the list of all existing rooms

4. System removes selected room from database
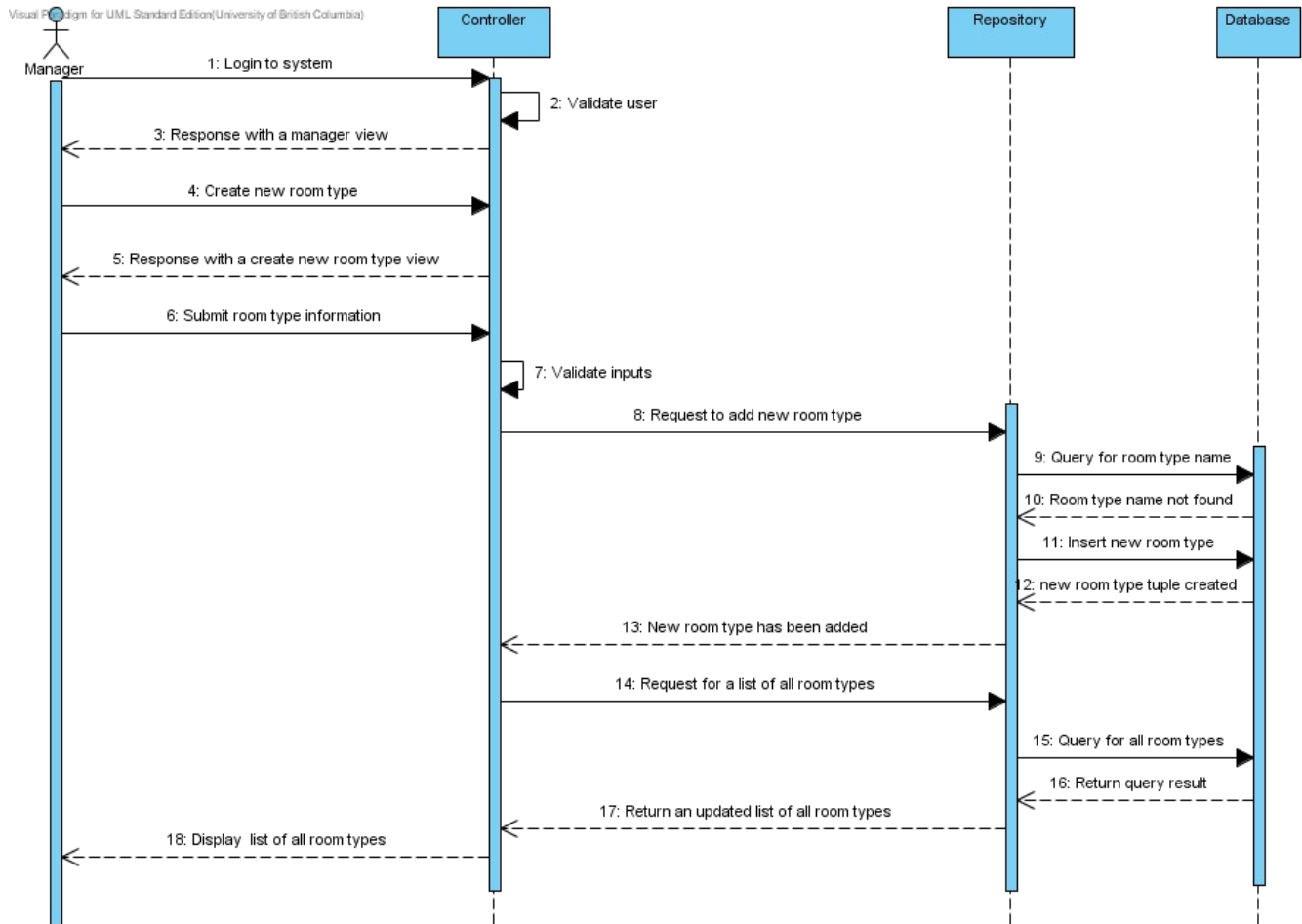
Figure 18: UC8 - Create New Room Type

Create new room type

Use case Number: UC8

1. Manager logs into the system using his or her account

2. Manager selects to create a new room type

3. Manager inputs name of type, beds, kitchenette, etc

4. System validates all the information entered is correct

    4.1. System displays an error message if the information is incorrect

5. System adds the new room type to the database
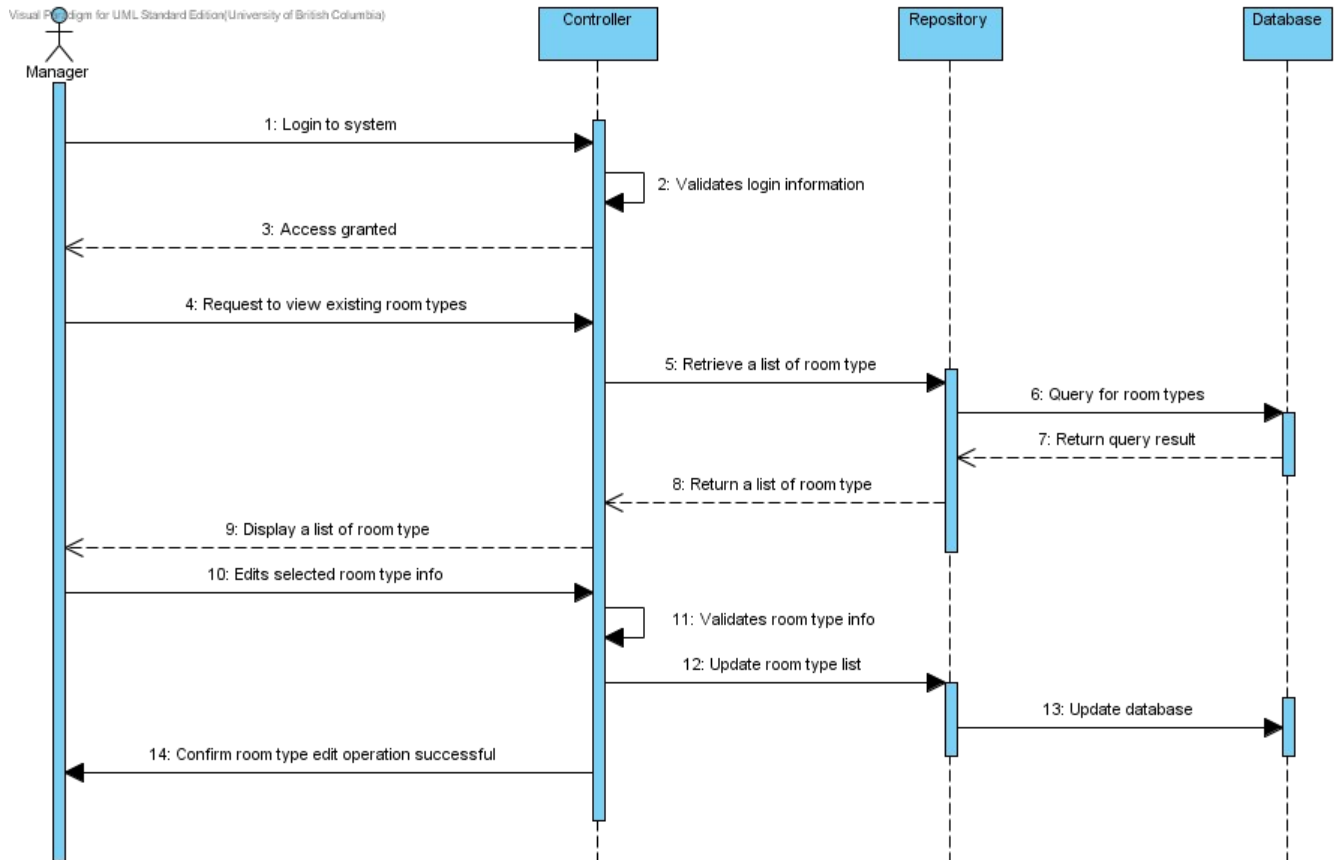
Figure 19: UC9 - Edit Room Type

Edit room type

Use case Number: UC9

1. Manager logs into the system using his or her account

2. Manager requests to view existing room types

3. After viewing a list of room type, Manager selects a room to edit

4. Manager enters new room type information (room type name, price rate, etc.)

5. System validates all the information entered is correct

    5.1. System displays an error message if the information is incorrect

6. System confirms the room type edit operation is completed
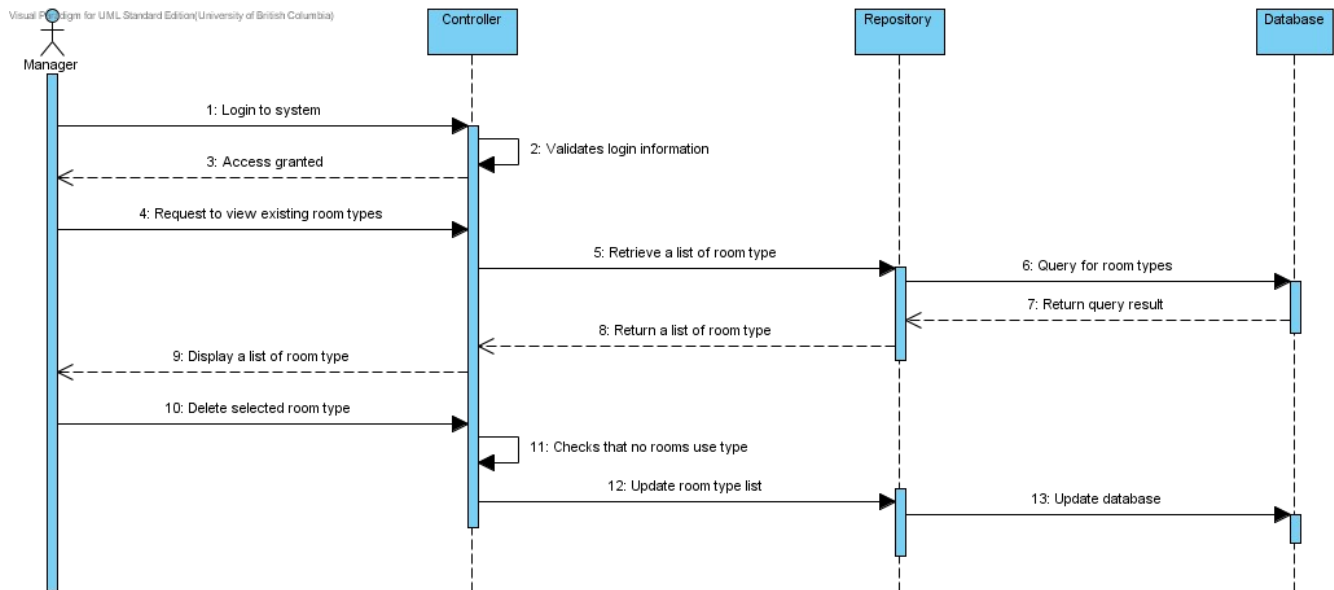
Figure 20: UC10 - Delete Room Type

Delete Room Type

Use case Number: UC10

1. Manager logs into the system using his or her account

2. Manager requests to view existing room types

3. After viewing a list of room type, manager selects a room type to delete

4. System checks that no rooms use that type

    4.1. If rooms use this type, show an error message
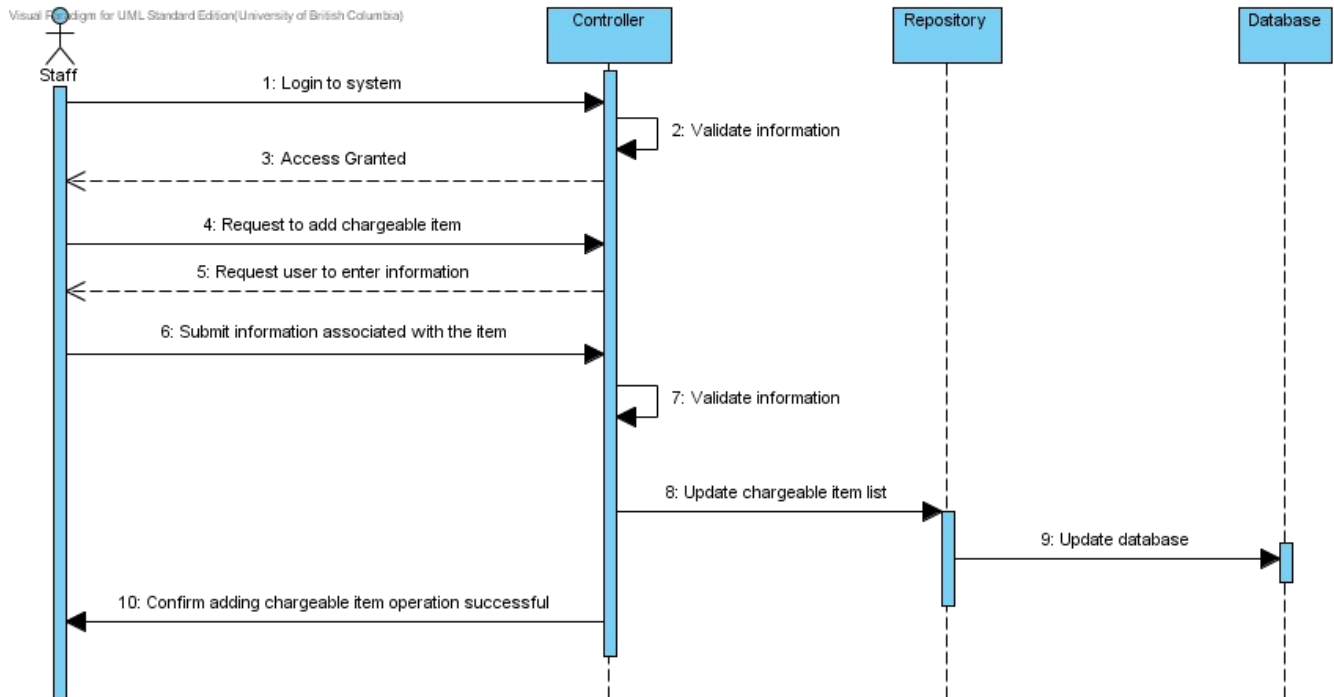
Figure 21: UC11 - Add Chargeable Item

Add chargeable item(s) [elided: edit chargeable item, delete chargeable item]

Use case Number: UC11

1. Hotel staff logs into the system using their account

2. Staff requests to add chargeable item

3. Staff enters all the information about the new item such as name and price

4. System validates all the information entered is correct

    4.1. System displays an error message if the information is incorrect

5. System confirms a new item is added
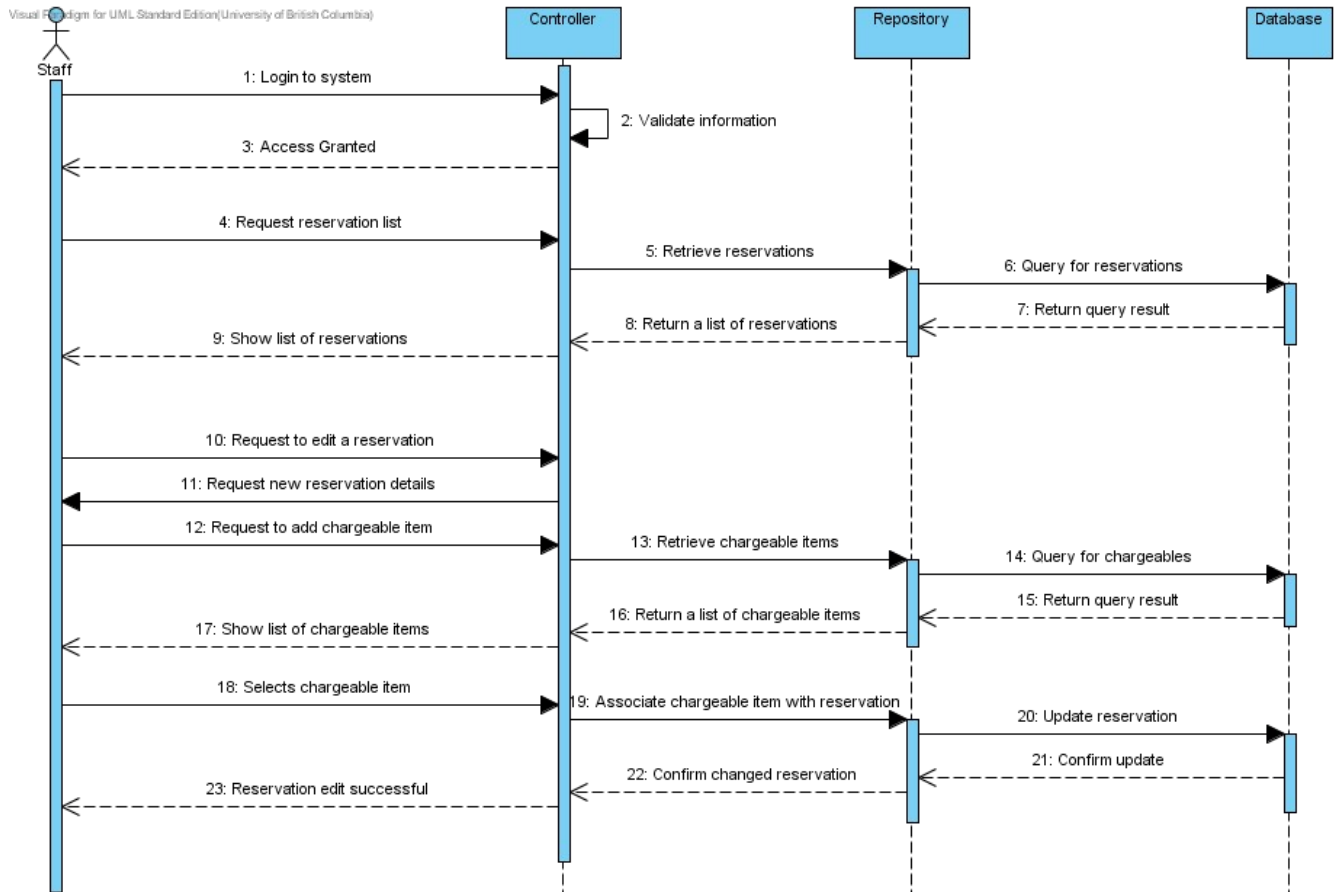
Figure 22: UC12 - Charge Chargeable Item

Charge chargeable item(s)

Use case Number: UC12

1. Hotel staff logs into the system using their account

2. Staff navigates to reservation listings

3. Staff selects a reservation to edit

4. Staff clicks "Add chargeable item" link

5. Staff selects a chargeable item

6. System validates all the information entered is correct

    6.1. System displays an error message if the information is incorrect

7. System confirms the association between the chargeable item and the guest is established
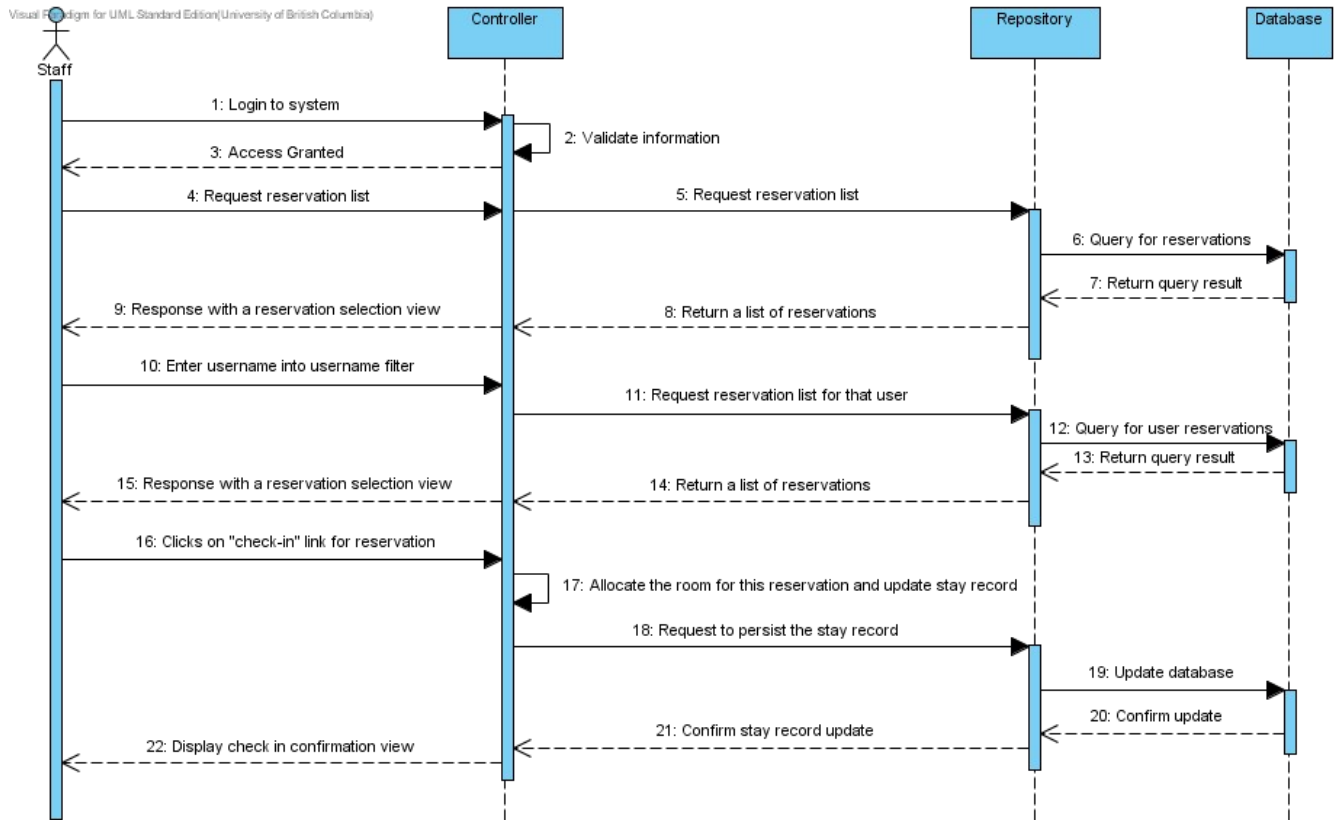
Figure 23: UC13 - Check-in

Check-in

Use Case Number: UC13

1. Hotel staff logs into his or her account

2. Hotel staff navigates to reservation listing

3. Receptionist enters customer username in the reservation filter

4. System displays all reservations for the given user

5. Receptionist clicks on "check-in" link for the desired reservation

6. System confirms check-in
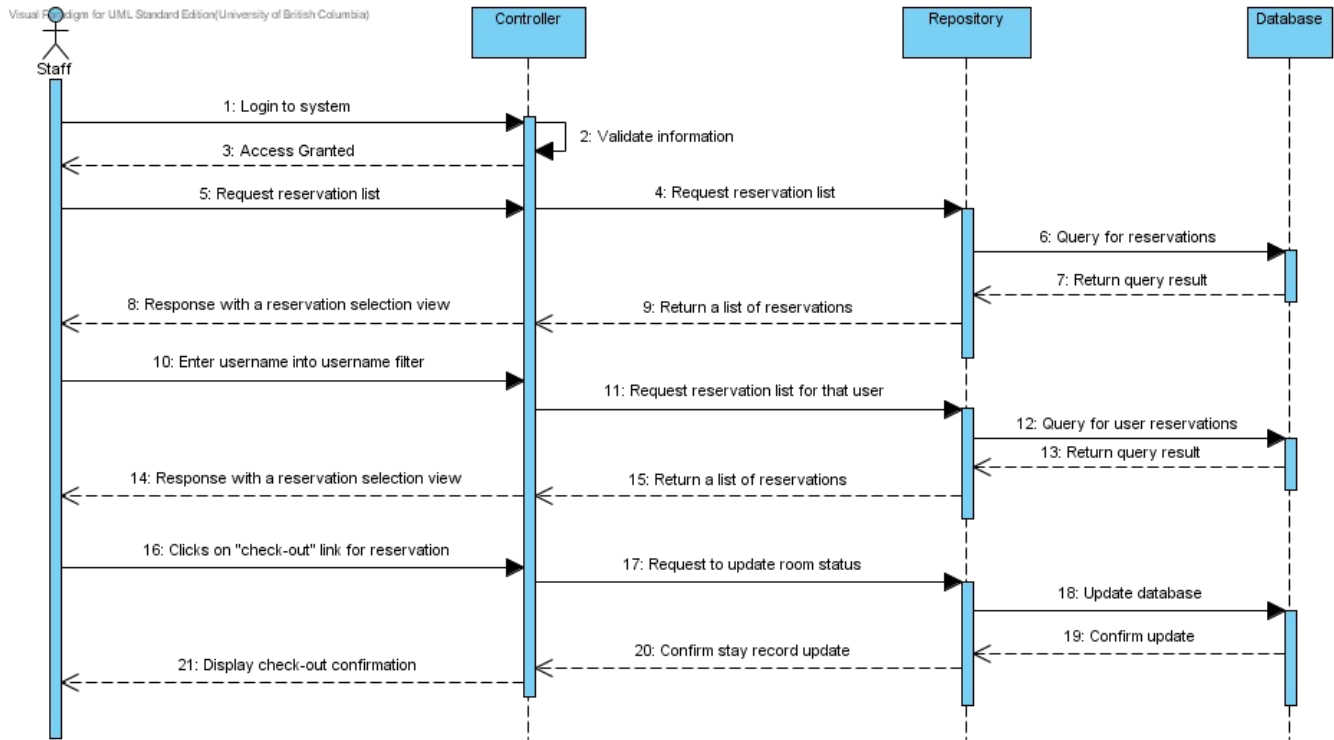
Figure 24: UC14 - Check-out

Check-out

Use Case Number: UC14

1. Hotel Staff logs into to his or her account

2. Hotel staff navigates to reservation listing

3. Receptionist enters customer username in the reservation filter

4. System displays an error message if the corresponding customer is not found

5. System displays all reservations for the given user

6. Receptionist clicks on "check-out" link for the desired reservation

7. System displays a summary of the stay and a detailed bill

8. Guest and hotel staff validate the correctness of the summary and bill

9. System charges the guest's credit card

10. System confirms check-out
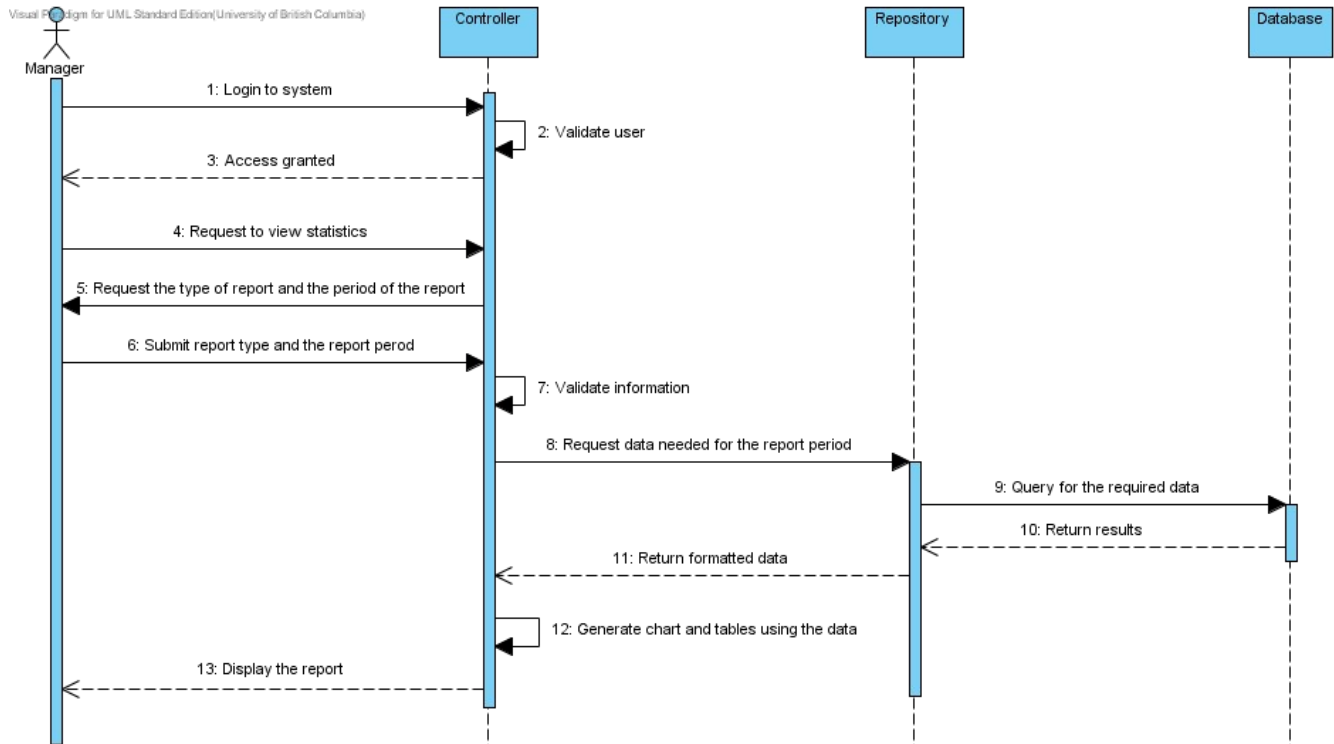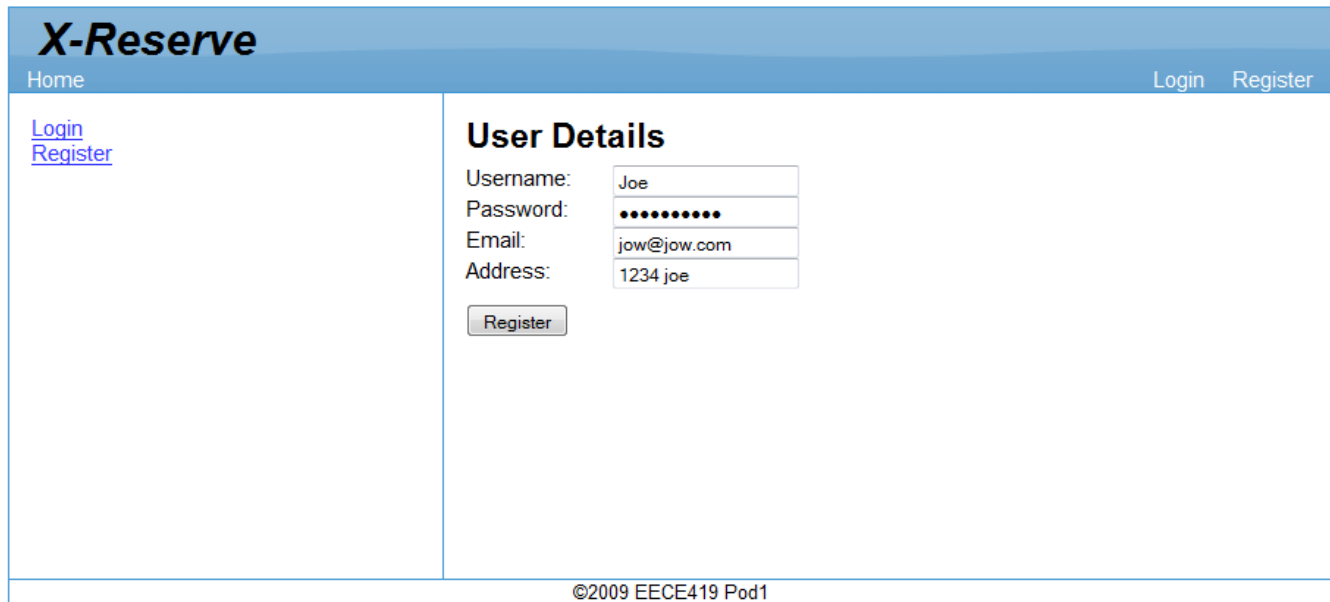
Figure 25: UC15 - View Statistic Report

View Statistic Report

Use Case Number: UC15

1. Manager logins his account

2. Manager requests to view reports

3. Manager enters the type of report (statistical analysis room occupancy, most popular type of room reservation, real-time view of occupied/free status) and the period of the report

4. System displays the corresponding report

Appendix A: User Interface Mockups

Figure 26: Create Account

Figure 27: Make a Reservation (1)

Figure 28: Make a Reservation (2)

Figure 29: Make a Reservation (3)

Figure 30: Search for Rooms

X-Reserve

Home   Rooms   Users   Reports   Reservations   admin   Edit Profile   Logout

Manage Rooms

View Rooms
Create Room

View Room Types
Create Room Type

View Chargeable Items
Create Chargeable Item

View Room Images
Upload New Image

All Room Types

4 room types found.

Room Type: Penthouse Suite
Occupancy: 12
Rate: 750.0
The ultimate in comfort.
Attributes: 3×Double, Kitchen
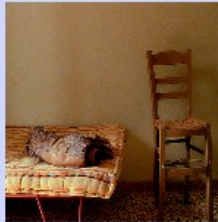
Room Type: Bachelor Dive
Occupancy: 1
Rate: 20.0
Perfect for starving students.
Attributes: Shared Bathroom, Single

Room Type: Corporate Econo-box
Occupancy: 2
Rate: 80.0
Comfort befitting Middle America.
Attributes: Double

Room Type: Double Econo-box
Occupancy: 4
Rate: 120.0
Twice the fun of a Corporate Econo-box.
Attributes: 2×Double

©2009 EECE419 Pod1

Figure 31: Manage Rooms

Figure 32: View Future Reservations

Figure 33: View Reports