Term Project

Hotel Room Booking System

CSIS 3275-001 – Software Engineering

Submission Date: 09, April 2021

Submitted To:

Reza Ghaeli

Professor at:

Douglas College,

New Westminster,

BC

Submitted By: (Team 8)

Heena Kashyap - 300305305
Sukhleen Kaur - 300310052
Sehajpreet Singh Kingra - 300304180
Kunal Ajaykumar Jeshang - 300328339

1. Introduction

1.1 Executive Summary

Repository Link: https://github.com/kjeshang/CSIS3275-TermProject

1.1.1 System Overview Summary

Our team has created a Hotel Room Booking System. It is a desktop application meant to be utilized as proprietary software for hotels our team is associated with (such as Goldman Suites). It is a simple & easy-to-use application that allows hotel personnel or administration (admin) to view, add, & edit hotel guest information and reservations. In addition, a hotel guest is also able to add their information and make a reservation themselves.

1.1.2 Team Information

Heena Kashyap - 300305305
Sukhleen Kaur - 300310052
Sehajpreet Singh Kingra - 300304180
Kunal Ajaykumar Jeshang - 300328339

2. General Overview and Design Guidelines/Approach

2.1 General Overview

Views & APIs
The Java Swing framework is used to create the user interface for hotel personnel (admin) and prospective hotel guests. As the hotel booking system is quite intricate, the WindowBuilder tool from the Eclipse Marketplace is utilized to make creating the user interface easier as well as more manageable. It is our team's plan to incorporate at least three main user interfaces using WindowBuilder; there would be a user interface screen for a login, hotel guest to enter information & book a room, and for the admin to search all the room bookings to check, add, change, or cancel. Other user interface elements would be handled by simply incorporating JOptionPanes.

Models & Services
The application has a model package that contains entities for login information for both hotel & admin, hotel guest information, hotel personnel (admin) information, and the hotel guest's room booking. The entities are instantiated in the user interfaces where input is entered by the user (i.e. admin or hotel guest), then taken by the model entities for further use. That being said, the model entities also implement certain business rules based on the input shown on the user interface; e.g. if the hotel guest requests for lunch & dinner meals, the total cost calculated by the guest booking entity will reflect that. The 'models' and 'views' classes handle the business logic; e.g. if the length of stay of the hotel guest is negative

(according to respective model entity), meaning that the check-out date is earlier than the check-in date, then the user of the application will be made aware of that by the user interface. The model entities are also responsible for processing the data in preparation for database interaction.

Controllers & Data Access Objects

As the application is desktop-based, the team opted to combine the 'controllers' and 'data access objects' (dao), which makes managing packages and numerous classes easier. Thus, the application takes on more of an MVC structure rather than a POJO structure; this is further supported by the fact that we are not using Spring Boot. All controller and data access object (dao) related elements are within the classes of the Control package, and instantiated in the user interface. These classes contain specific methods to create & open the database and collections for interaction. For example, a method exists to insert information into the guest collection which holds a hotel guest's background information; the parameter would be a guest information entity.

2.2 System Constraints/Dependencies/Risks

2.2.1 Constraints

- The app will be deployed on Hotel Admin management computers and guest check in computers.
- The app can only be accessed within hotel permies and can be installed only on Hotel's own computing devices.
- The app can be deployed on any Operating system Mac or Windows computer.

2.2.2 Dependencies

- In order to access the app both the admin and the guest need to create an account.
- Admin accounts can only be created by senior managers whereas the guest account can be created by anyone visiting the hotel.
- Once a guest has made a booking only an admin can edit it or delete.
- Once an admin chooses to update a guest personal or booking information the check in and out dates need to be updated every time.

2.2.3 Risks

- Admins can accidentally delete, update a guest information
- The app deployed on the guest kiosks have a risk of being hacked by hackers to get access of admin accounts
- The computers with the management app installed need to protected by a security measure to ensure credibility of the apps

3. Design Considerations

3.1 Goals and Guidelines

3.2 Operational Environment

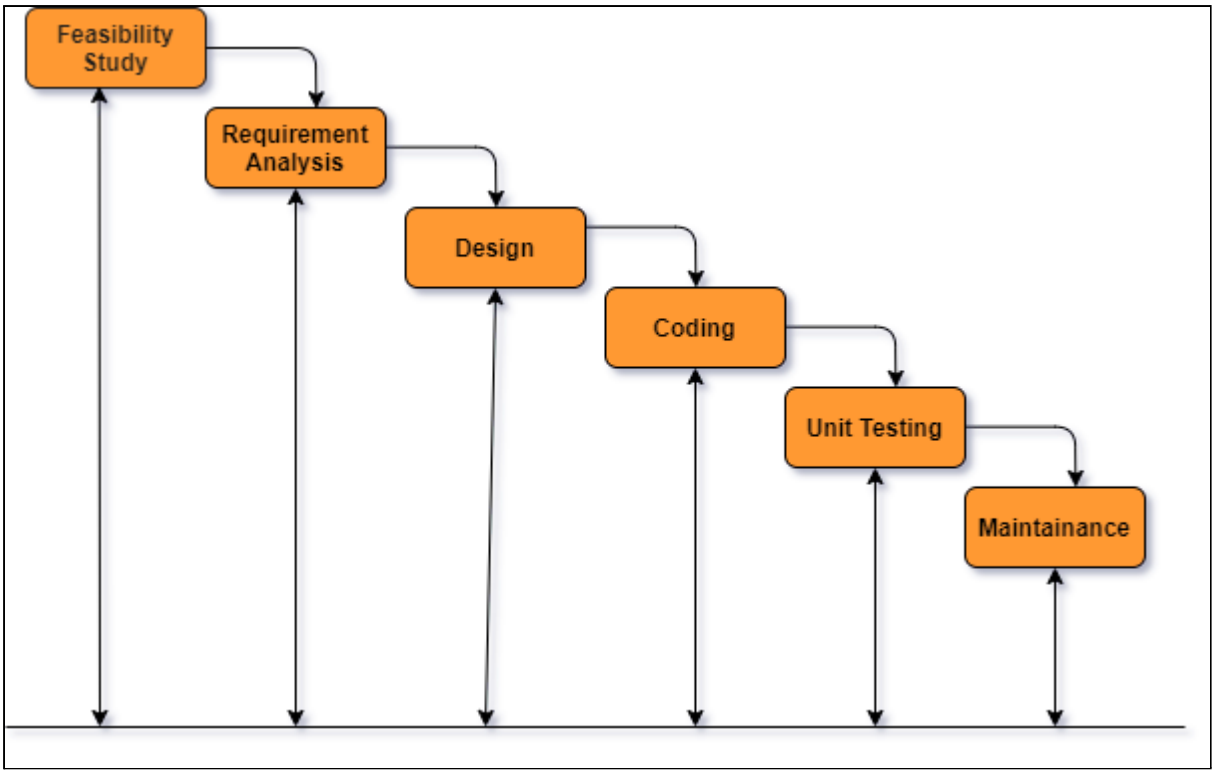
- Language: Java
- Version Control Software: Git
- Software: Eclipse
- Database tools: MongoDB

3.3 Development Methods

- Application structure & approach: MVC (Model-View-Controller)
- Framework: Java Swing
- Graphic User Interface (GUI) designer: WindowBuilder
- Plugins:
    - JCalender 1.4 JAR driver: utilized in order to incorporate a date chooser for hotel guest check-in & check-out date
    - Mongo Java Driver 3.12.8 JAR driver: utilized in order to incorporate the MongoDB database to handle the POST, GET, PUT, & DELETE commands, in turn, turning the application into a multi-layer application.
- Testing: JUnit 5 (Jupiter)

3.4 Project Development Scheduling

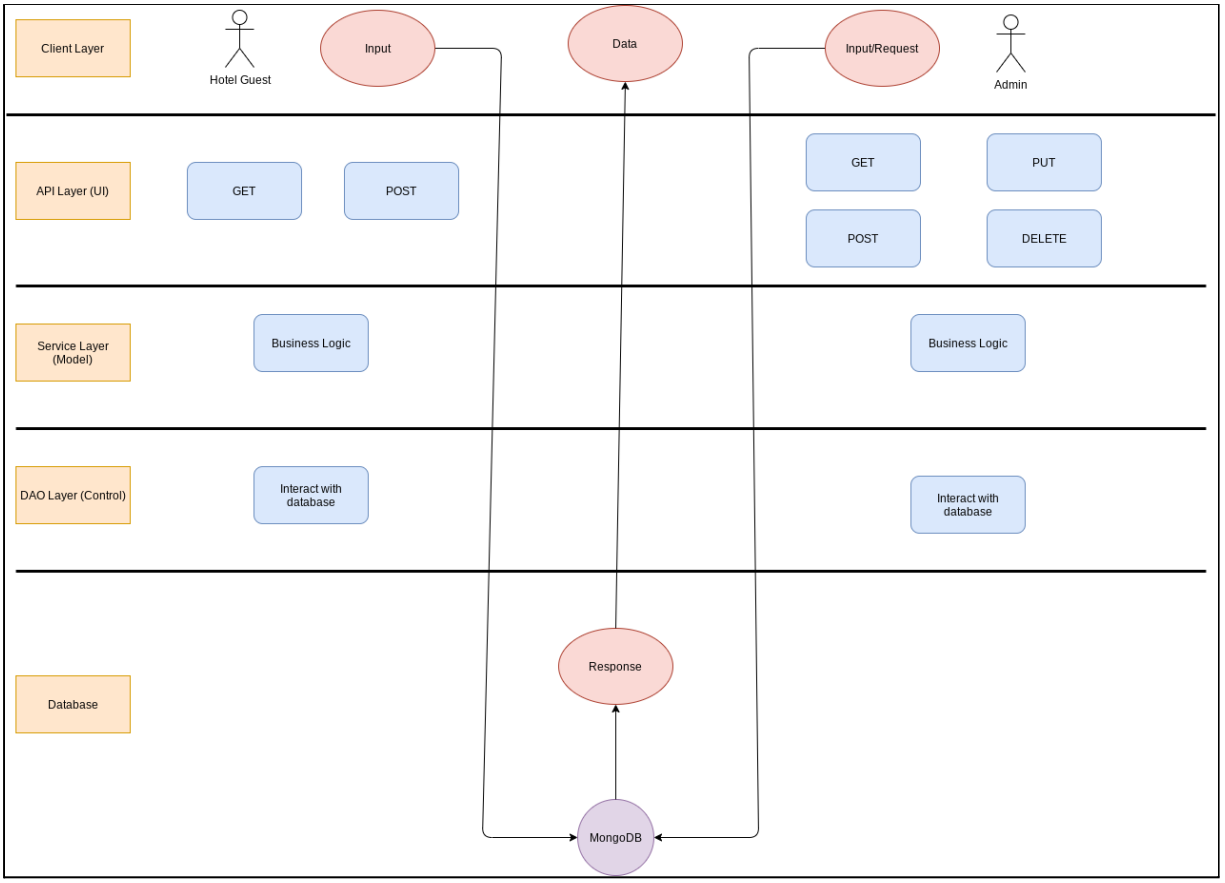
- Hotel Room Booking System is developed in four phases that include analysis, design, coding and testing.
- The Iterative Waterfall model is used to best represent the scenario as this model allows us to step back to the previous development phase of the life cycle.



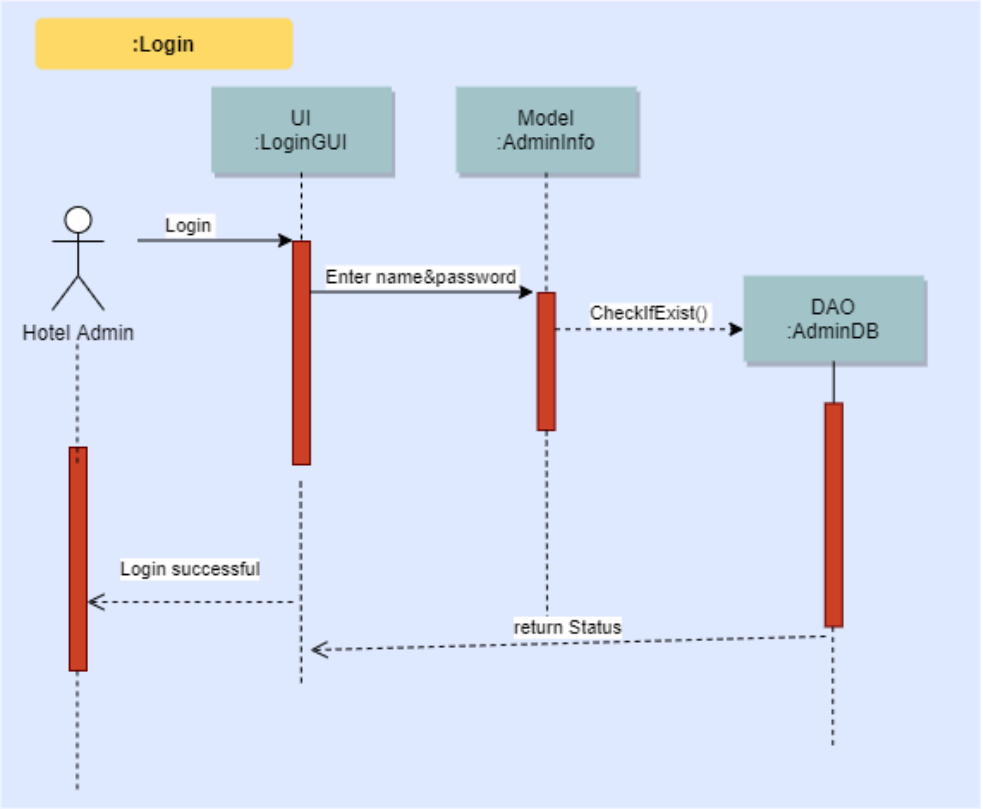
4. System Architecture and Architecture Design

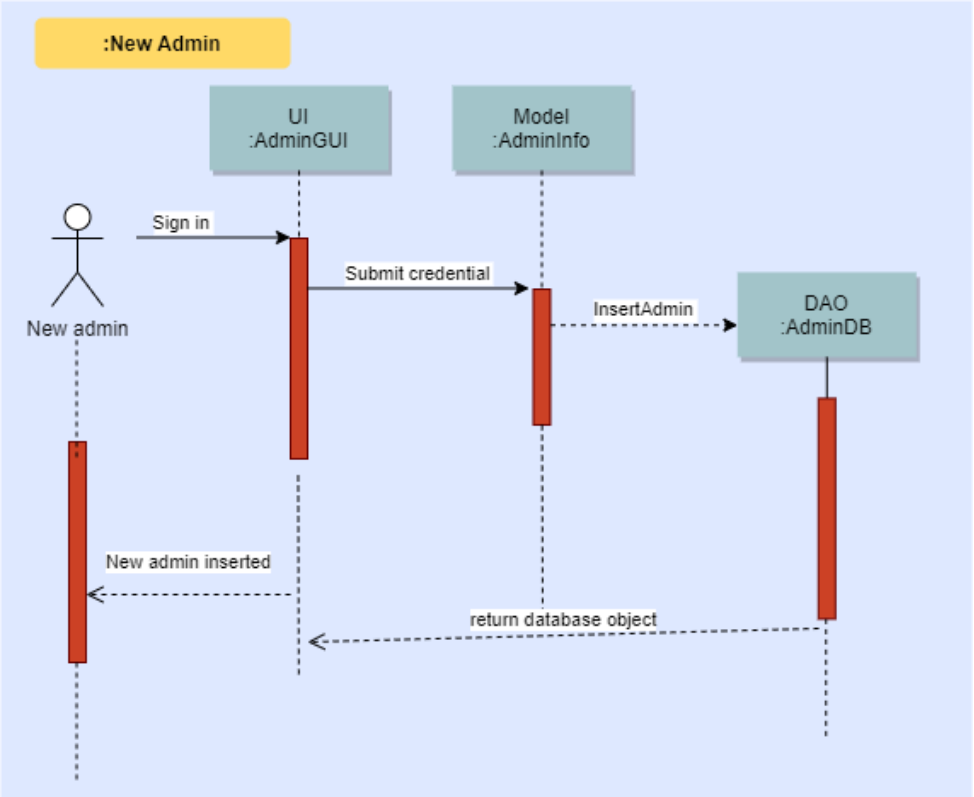
4.1 System Architecture Diagrams

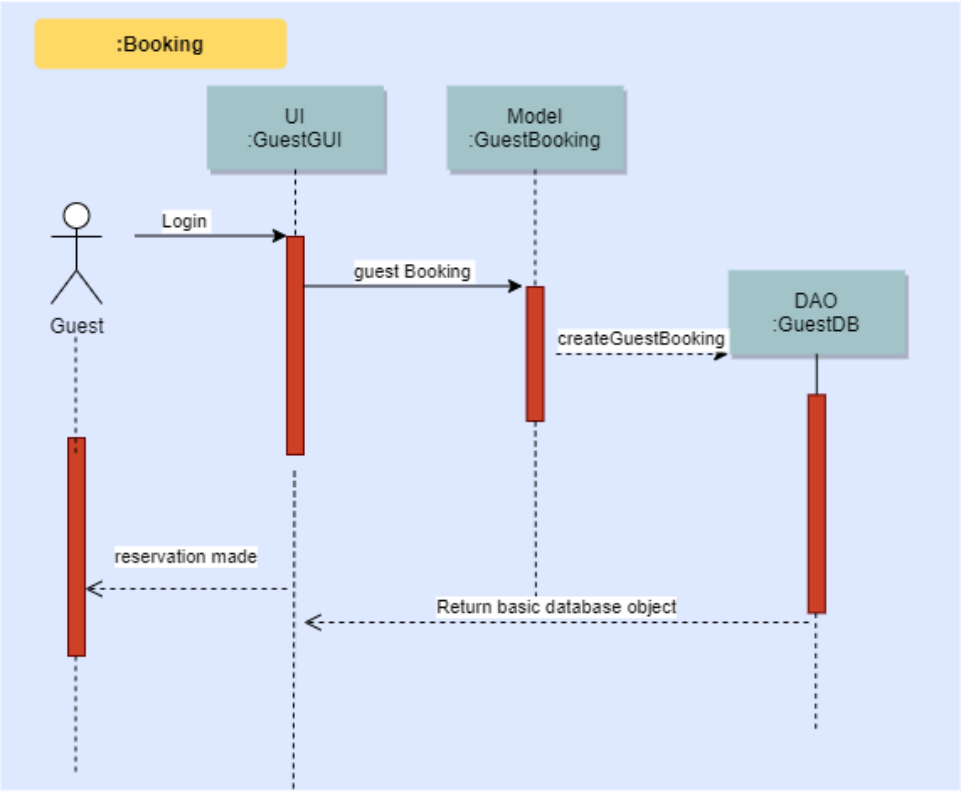
4.1.1 Functional Diagram

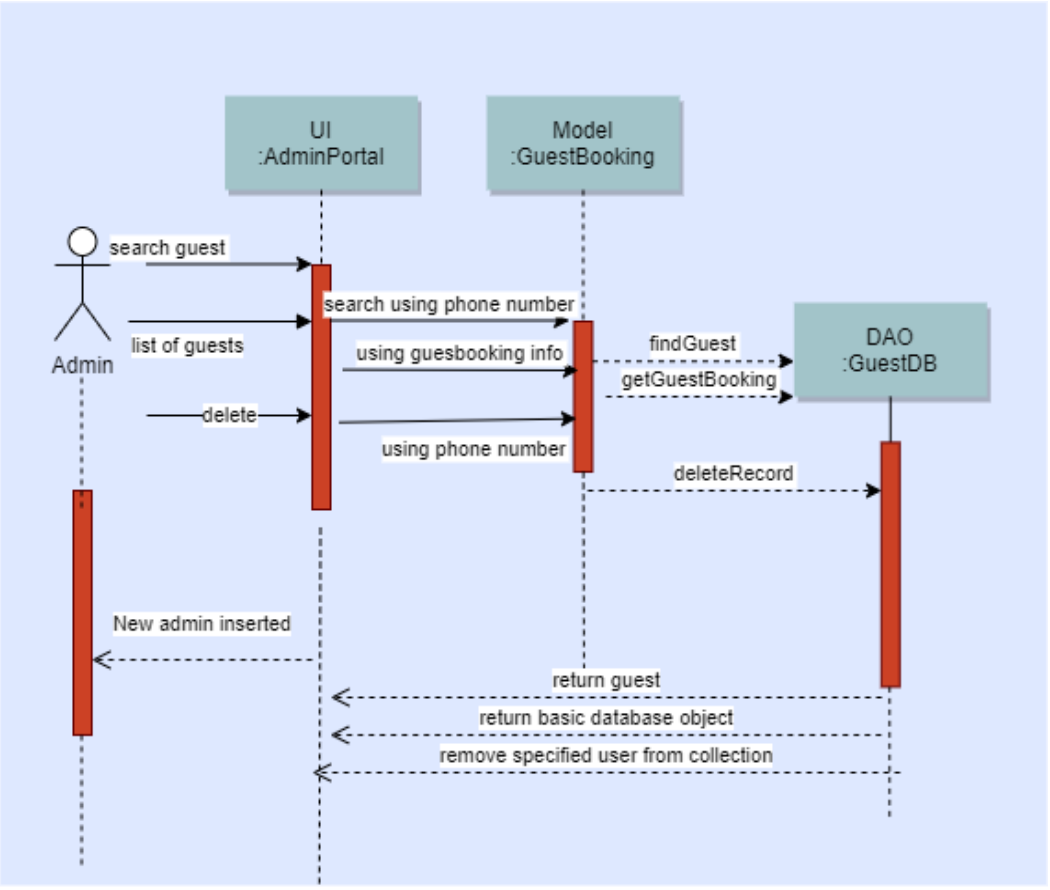


4.1.2 Sequence diagram

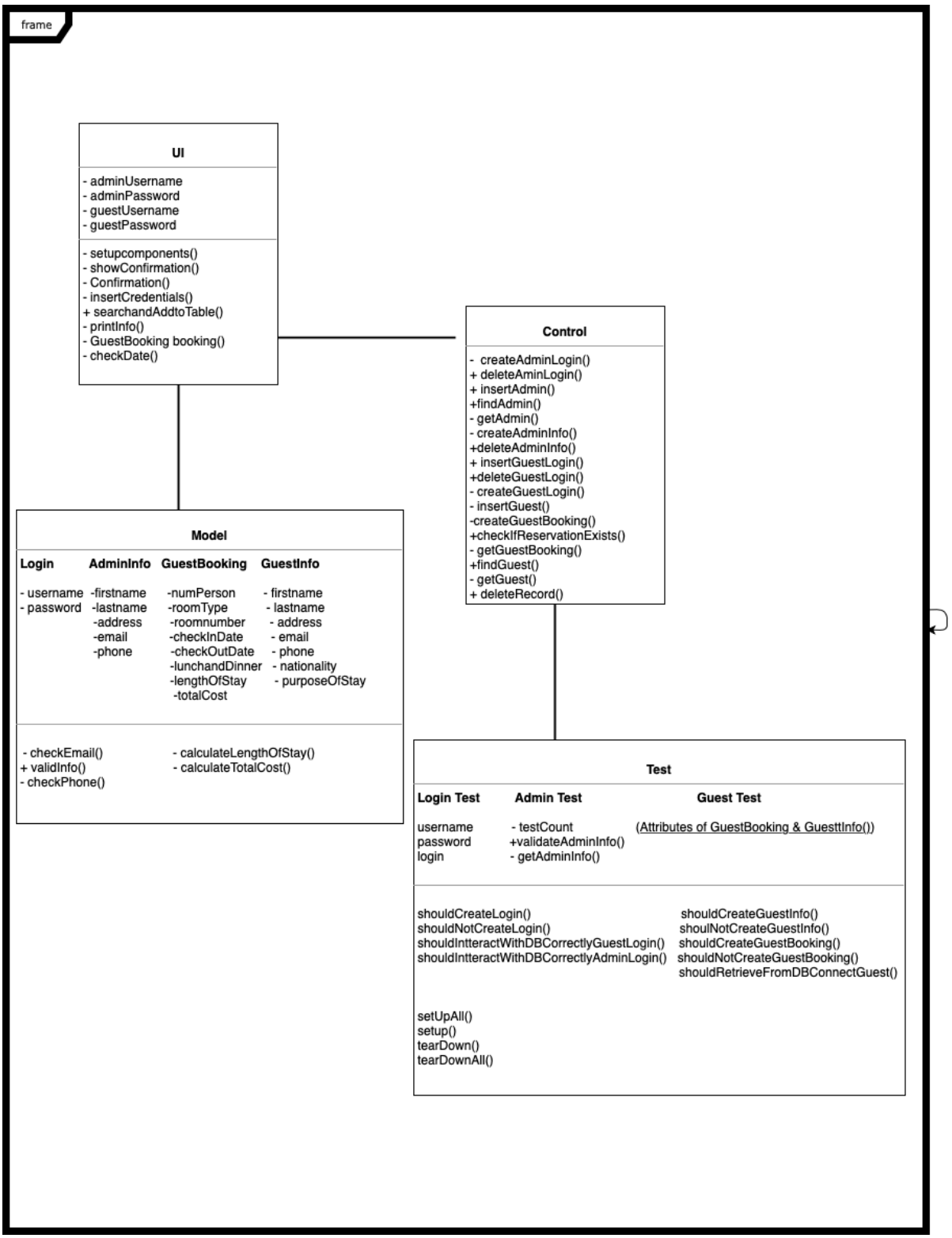



:New Admin
UI
:AdminGUI
Model
:AdminInfo
Sign in
New admin
Submit credential
InsertAdmin
DAO
:AdminDB
New admin inserted
return database object

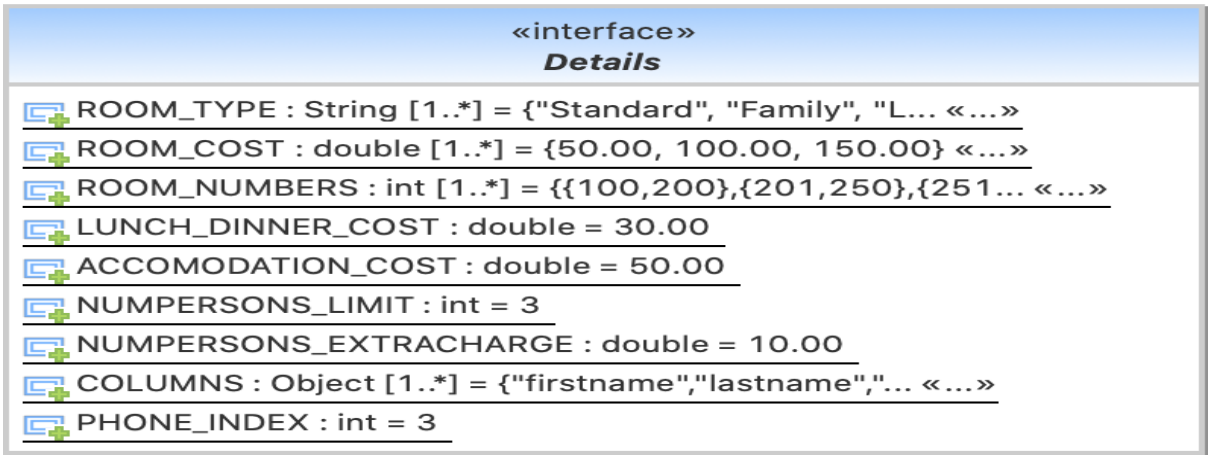

:Booking
UI
:GuestGUI
Model
:GuestBooking
Login
Guest
guest Booking
DAO
:GuestDB
createGuestBooking
reservation made
Return basic database object


UI
:AdminPortal
Model
:GuestBooking
DAO
:GuestDB
Admin
search guest
list of guests
delete
search using phone number
using guesbooking info
using phone number
findGuest
getGuestBooking
deleteRecord
New admin inserted
return guest
return basic database object
remove specified user from collection

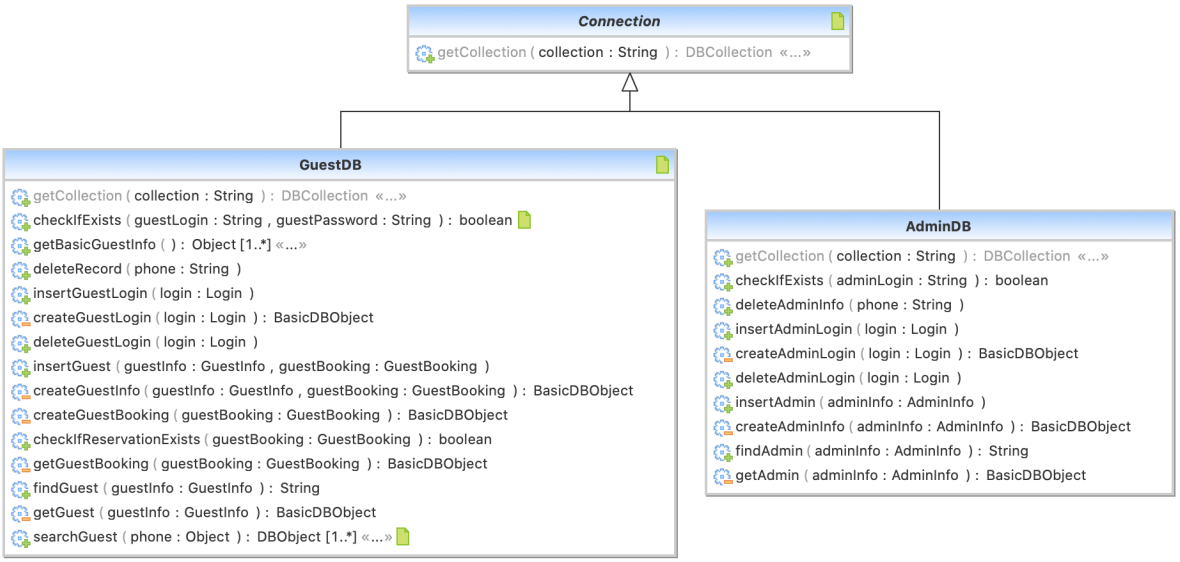
4.1.3 UML Diagram



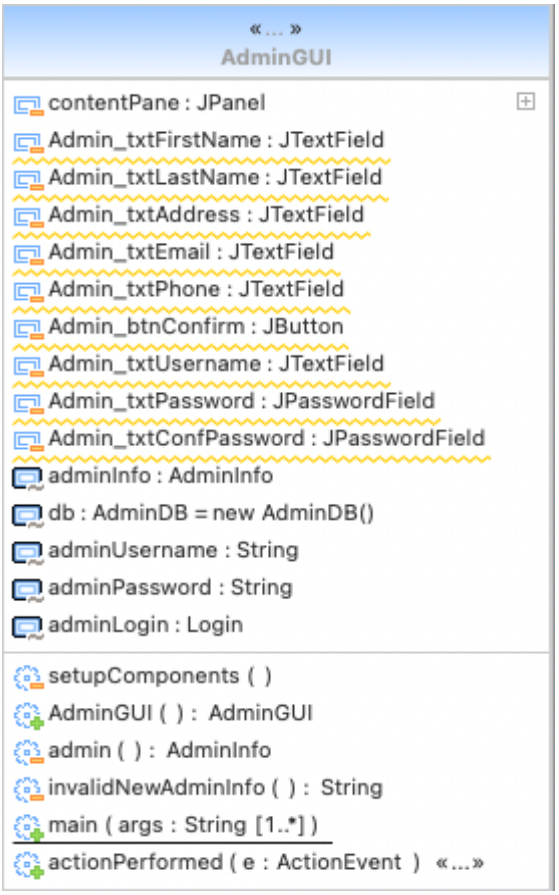
4.1.3.1. Model Package UML Objects

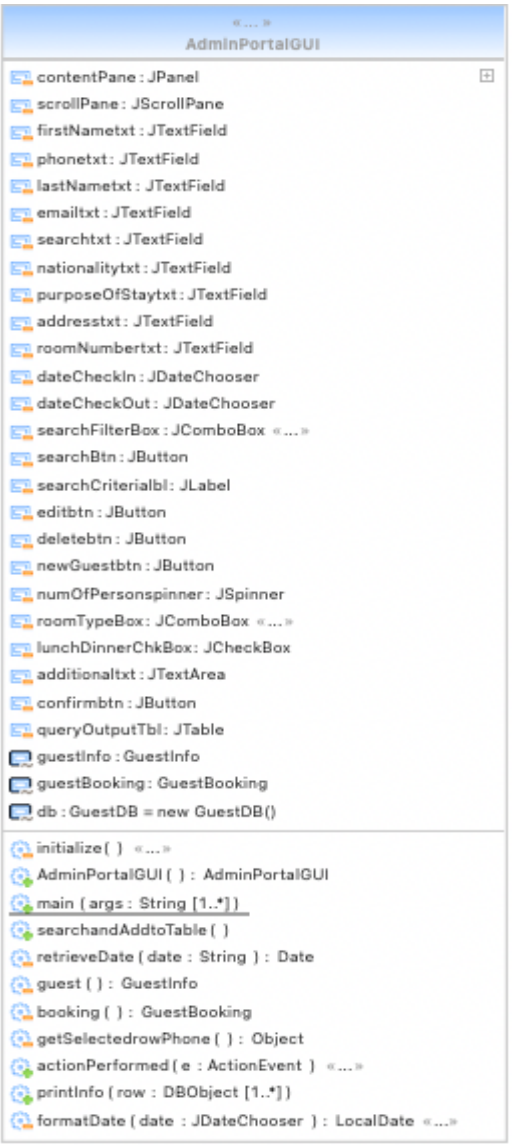


4.1.3.2. Control Package UML Objects



4.1.3.3. UI Package UML Objects

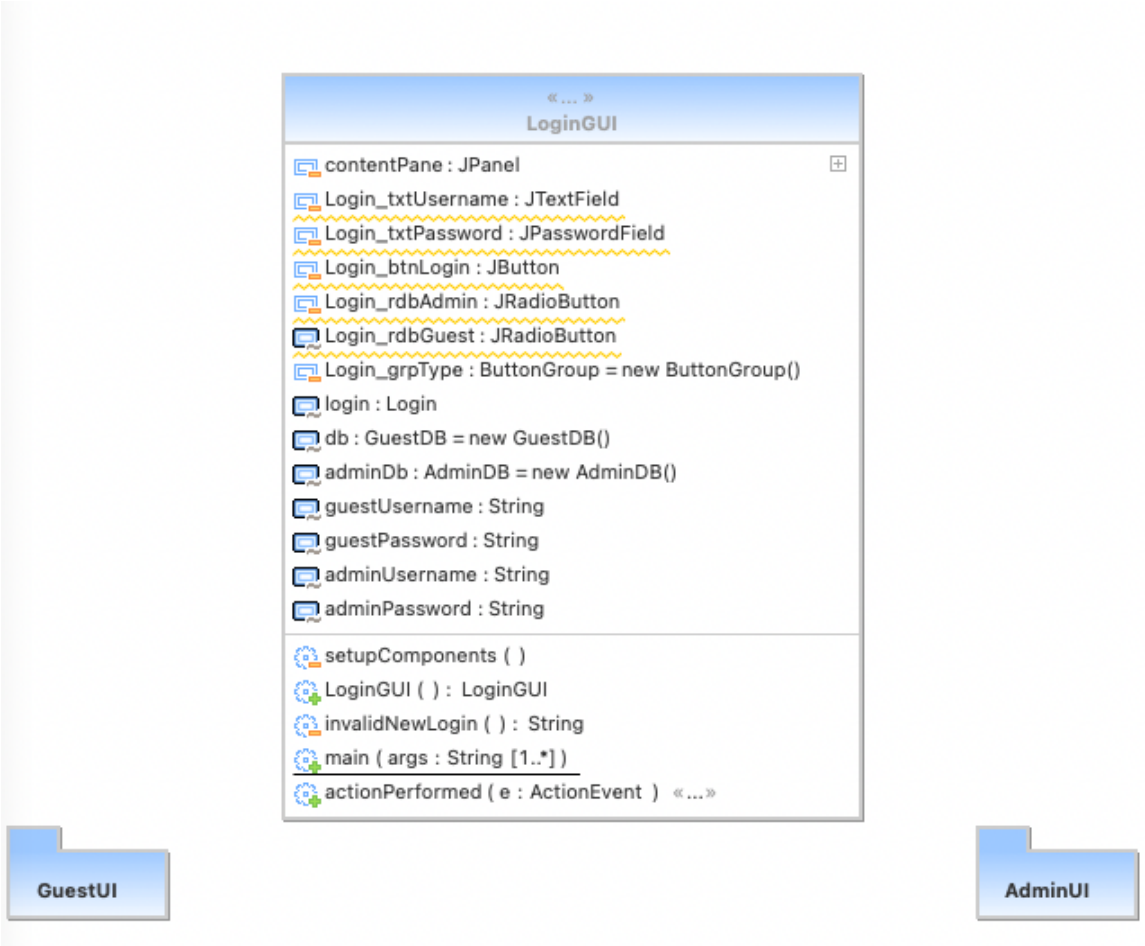




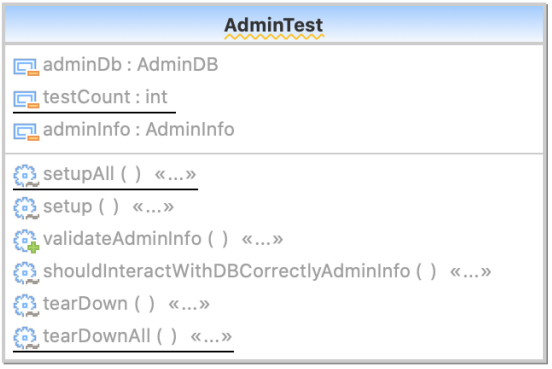
«...»
GuestGUI

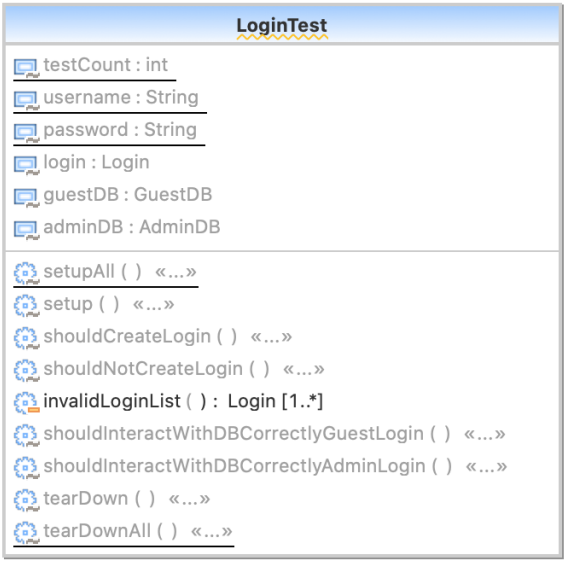
contentPane : JPanel
Guest_txtFirstName : JTextField
Guest_txtLastName : JTextField
Guest_txtAddress : JTextField
Guest_txtEmail : JTextField
Guest_txtPhone : JTextField
Guest_txtNationality : JTextField
Guest_txtPurposeOfStay : JTextField
Guest_btnConfirm : JButton
Guest_btnBookRoom : JButton
Guest_spNumPersons : JSpinner
Guest_cbxRoomType : JComboBox «...»
Guest_txtRoomNumber : JTextField
Guest_dateCheckIn : JDateChooser
Guest_dateCheckOut : JDateChooser
Guest_chkLunchDinner : JCheckBox
Guest_txtAddAccomodation : JTextArea
guestInfo : GuestInfo
guestBooking : GuestBooking
db : GuestDB = new GuestDB()

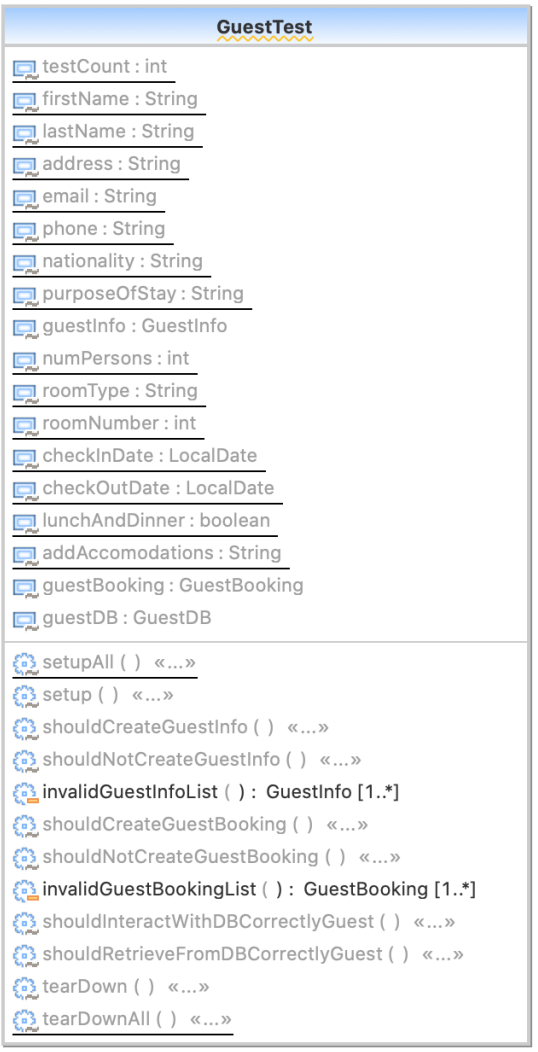
setupComponents ( ) «...»
setVisibility (status : boolean)
GuestGUI ( ) : GuestGUI
guest ( ) : GuestInfo
invalidNewGuestInfo ( ) : String
booking ( ) : GuestBooking
invalidNewGuestBooking ( ) : String
showConfirmation ( )
main (args : String [1..*])
confirmation (guestInfo : GuestInfo) : String
actionPerformed (e : ActionEvent) «...»
formatDate (date : JDateChooser) : LocalDate «...»


«...»
LoginGUI
contentPane : JPanel
Login_txtUsername : JTextField
Login_txtPassword : JPasswordField
Login_btnLogin : JButton
Login_rdbAdmin : JRadioButton
Login_rdbGuest : JRadioButton
Login_grpType : ButtonGroup = new ButtonGroup()
login : Login
db : GuestDB = new GuestDB()
adminDb : AdminDB = new AdminDB()
guestUsername : String
guestPassword : String
adminUsername : String
adminPassword : String
setupComponents ( )
LoginGUI ( ) : LoginGUI
invalidNewLogin ( ) : String
main (args : String [1..*])
actionPerformed (e : ActionEvent) «...»
GuestUI
AdminUI

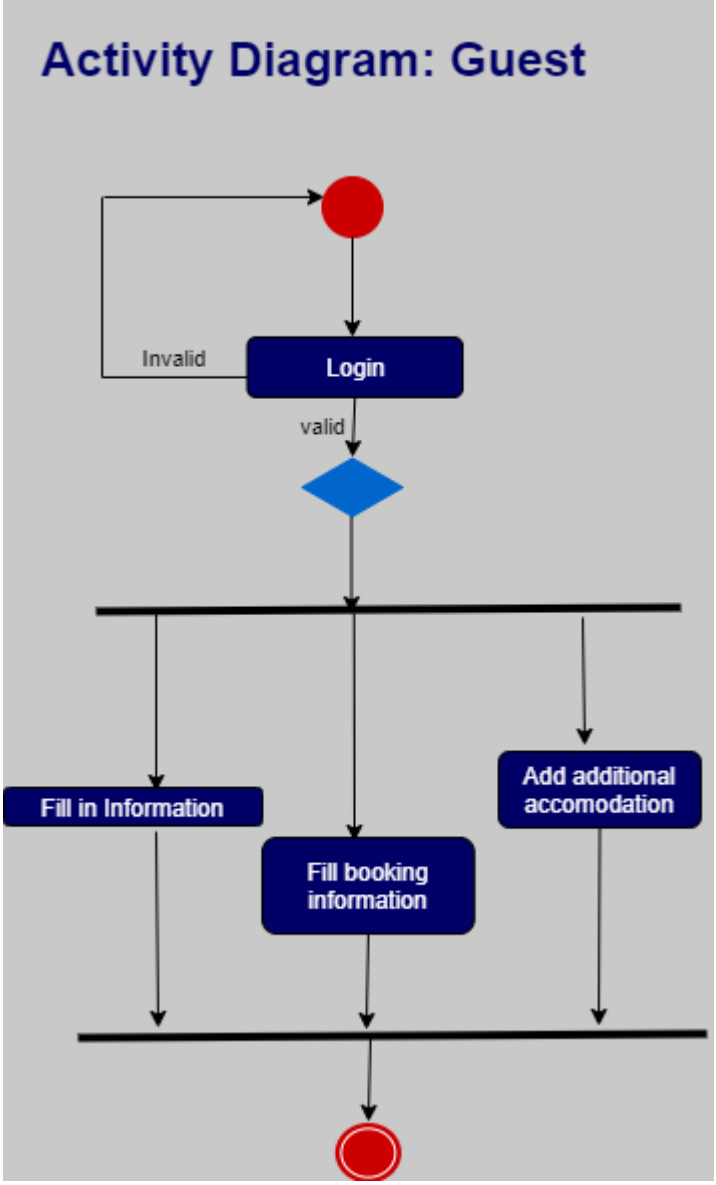
4.1.3.4. Tests Package UML Objects

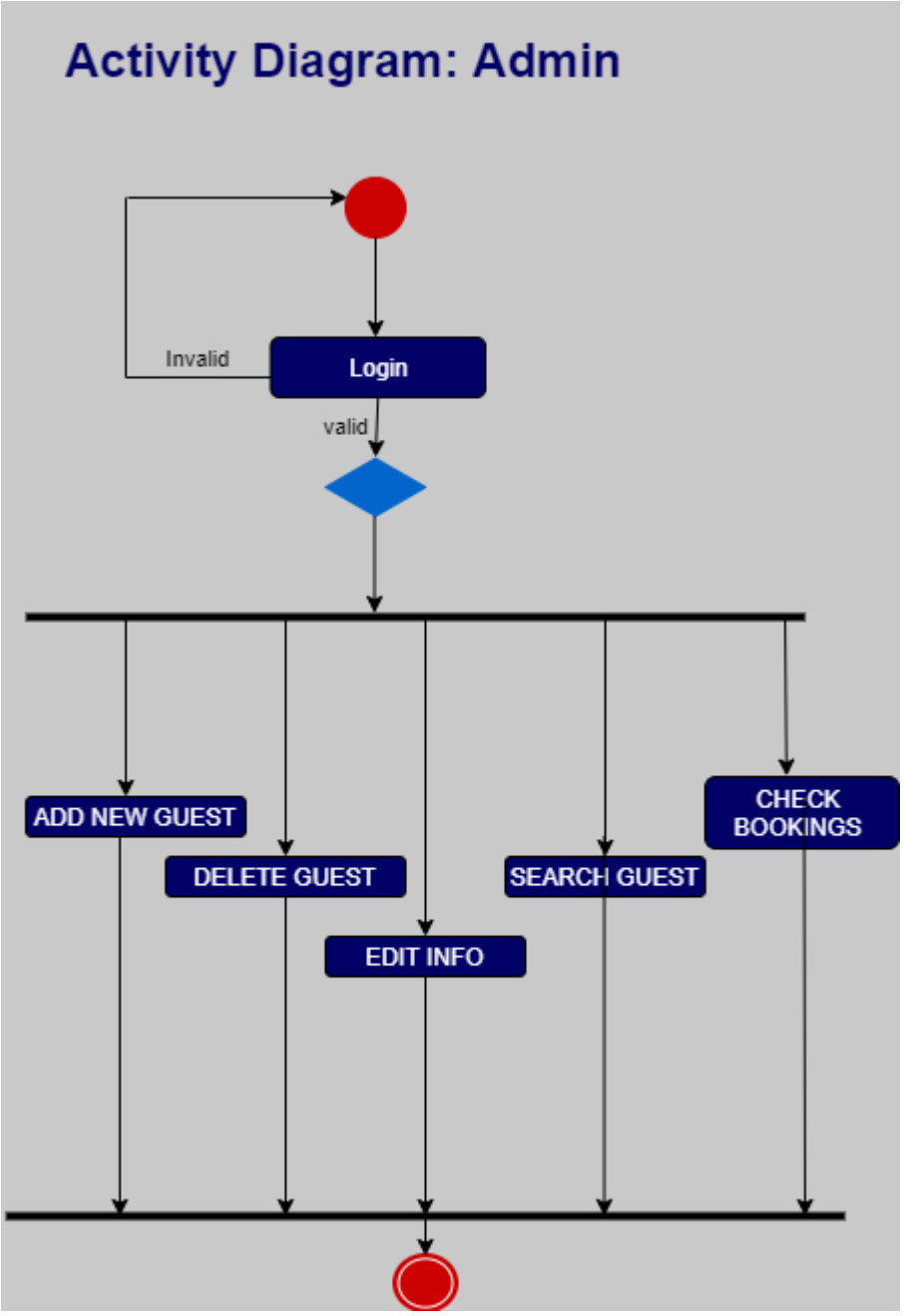




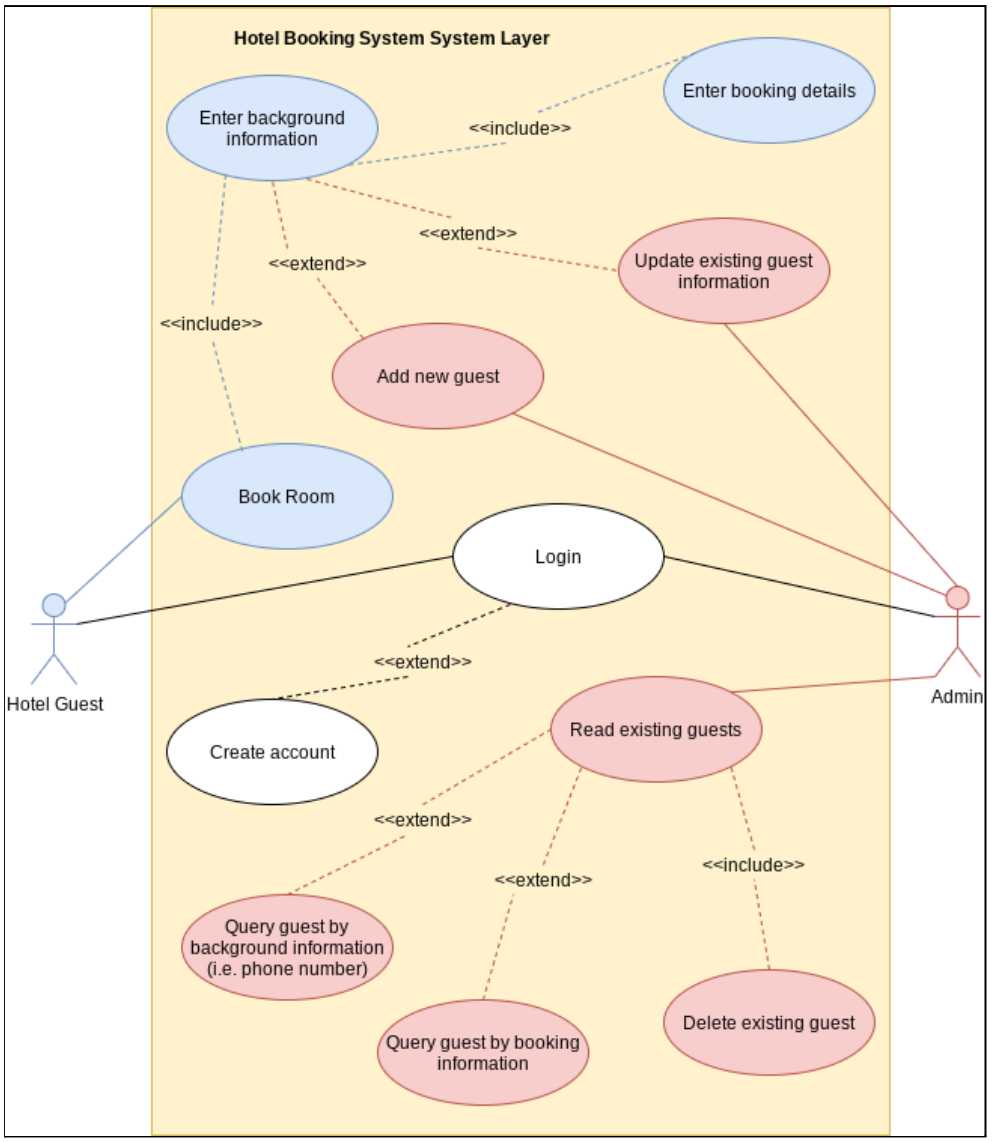


4.1.4 Activity Diagram




Activity Diagram: Admin
Invalid
Login
valid
ADD NEW GUEST
DELETE GUEST
EDIT INFO
SEARCH GUEST
CHECK BOOKINGS

4.1.5 Use Case Diagram



5. System Design

5.1 Business Requirements

5.1.1 Priorities

Hotel Guest Perspective:

- Guests can login to/register on the hotel application; all valid guests require a username and password.
- The guests provide their background information.
    - Name
    - Nationality
    - Address
    - Phone

    - Email
    - Purpose of stay
- Guests are able to find out about room availability and total cost by providing the following details.
    - Room type
    - Room number
    - Number of persons accompanying them to stay at the hotel
    - Check-in date
    - Check-out date
- Guests must also inform if they require lunch & dinner, as well as any additional accommodations for their stay; total cost would be incremented accordingly.
- If a guest wants to change or cancel their room booking, then they would have to contact the hotel admin to handle the procedure.

Administration (Admin)/Hotel Personnel Perspective:

- Admin log into/register on the hotel application; all valid admin require a username and password.
- Admin have full access to the details of current hotel guests and their booking information.
- Admin has the ability to override or change room reservation information as well as hotel guest details.

5.2 Database Design

5.2.1 Data Objects and Resultant Data Structures

Database: Hotel (working name is “CSIS3275”)

Guest Login Collection (name: Guest_Login)

Key	Type
Username	Value
Password	Value

Guest Collection (name: Guest)

Key	Type
First name	Value
Last name	Value
Address	Value
Email	Value
Phone	Value

Nationality	Value
Purpose of stay	Value
Booking	Embedded document Key: Type Number of persons: Numerical value Room type: Value Check-In date: Date value Check-Out date: Date value Lunch and Dinner: Value Additional Accommodations: Value Length of stay: Numerical value Total cost: Numerical value

Admin Login Collection

Key	Type
Username	Value
Password	Value

Admin Collection

Key	Value
First Name	Value
Last Name	Value
Address	Value
Email	Value
Phone	Value

5.2.2 File and Database Structures

Guest_Login Collection:

```
{
    "_id" : ObjectId("6067ced51296600b983a5505"),
    "username" : "guest",
    "password" : "guest123"
}
```

Guest Collection:

```
{
    "_id" : ObjectId("6067e6d64ce67c1581526781"),
    "firstName" : "Peter",
    "lastName" : "Parker",
    "address" : "Queens, New York, New York",
    "email" : "peter.parker@dailybugle.com",
    "phone" : "1234567890",
    "nationality" : "American",
    "purposeOfStay" : "Business",
    "booking" : {
        "numPersons" : 1,
        "roomType" : "Standard",
        "roomNumber" : 100,
        "checkInDate" : ISODate("2021-04-02T00:00:00.000Z"),
        "checkOutDate" : ISODate("2021-04-03T00:00:00.000Z"),
        "lunchAndDinner" : true,
        "addAccomodations" : "Extra Pillows",
        "lengthOfStay" : 1,
        "totalCost" : 130.0
    }
}
```

Admin_Login Collection

```
{
    "_id" : ObjectId("6067e3e5fb9fcf511e794134"),
    "username" : "admin",
    "password" : "admin123"
}
```

Admin Collection

```
{
    "_id" : ObjectId("6067e3e5fb9fcf511e794132"),
    "firstName" : "Steve",
    "lastName" : "Rogers",
    "address" : "Brooklyn, New York, New York",
    "email" : "steve.rogers@shield.com",
    "phone" : "1231231234"
}
```

5.3 User Interface

5.3.1 Inputs

Login GUI:

- Hotel Guest
    - Username
    - Password
- Admin
    - Username
    - Password

Hotel Guest GUI:

- Hotel Guest background information
    - First name
    - Last name
    - Address
    - Email
    - Phone
    - Nationality
    - Purpose of stay
- Hotel Guest booking information
    - Number of persons checking into room
    - Room type (Standard, Family, Luxury)
    - Check-in date
    - Check-out date
    - Option to request lunch & dinner meals in addition to complimentary breakfasts
    - Option to request any additional accommodations (e.g. extra pillows, sofa bed so more people can stay in a single room)

Admin GUI

- Admin/Hotel Personnel background information
    - First Name
    - Last Name
    - Address
    - Phone
    - Email
    - Username
    - Password

AdminPortalGUI

- Search all guests
    - Based upon first name, last name, email , phone or none.
- Delete guest = selected user from table results
- Update User = selected user from the table results
- Insert new user = entering the information in the given JFrame entities
- The information that can be Inserted/updated/deleted from Admin Portal:

Guest personal Information	Guest Booking Information
1. First name 2. Last name 3. Address 4. Email 5. Phone 6. Nationality 7. Purpose of stay	8. Number of persons checking into room 9. Room type (Standard, Family, Luxury) 10. Check-in date 11. Check-out date 12. Option to request lunch & dinner meals 13. Option to request any additional accommodations

5.3.2 Outputs

Login GUI:

- Login entity created from Hotel Guest username & password, and then insertion into Guest_Login collection.

Hotel Guest GUI

- GuestInfo entity created from Hotel Guest background information.
- GuestBooking entity created from Hotel Guest booking information; length & total cost of stay is calculated and displayed.
- GuestInfo and GuestBooking entities are inserted as a single document into the Guest collection where GuestBooking is joined to GuestInfo as an embedded document.
- Hotel Guest receives a final confirmation output of showing their background & booking information from the database.

Admin GUI

- Login entity created from Admin username & password, and then insertion into Admin_Login collection.
- Admininfo entity created from Admin background information, and then insertion into Admin collection.

AdminPortal GUI:

- Search all guests = output would be a scrollable table list of hotel guests; columns of table would be first name, last name, phone number, and email
- Delete guest = output would be that the particular hotel guest that was deleted will not appear in the scrollable table if the application user searches for all guests, or specifies first name/last name/phone number/ email of the deleted guest
- Edit guest by searching based on first name, last name, phone number, or email = output would be that hotel guest's background information & booking information in the appropriate editable fields

5.4 Code Structure

5.4.1 Classes

- Control package
    - Connection Java Class (creates/opens database)
    - GuestDB Java Class (creates/interacts with guest related collections in database)
- Model package
    - Guest sub-package
        - GuestBooking Java Class (guest booking information)
        - GuestInfo Java Class (guest background information)
    - Details Java Interface (commonly reused hotel information)
    - Login Java Class (username & password for application user)
- Test package
    - AdminTest Java Class
    - GuestTest Java Class
    - LoginTest Java Class
    - TestAll Java Class (Combines and runs all other JUnit test cases)
- UI package
    - Admin UI sub-package
        - Admin GUI Java Class (admin account registration & background information)
        - Admin GUI Portal Java Class (admin portal to insert/cancel/change hotel guest background information & room bookings)
    - Guest UI sub-package
        - GuestGUI (user interface for hotel guest)
    - Login GUI (user interface for both admin and hotel guest)

5.4.2 Pseudocode

5.4.2.1 Hotel Guest

1. Open application
2. Login Screen open
3. Enter username and password, then click login
4. Application checks if username and password exists in database
    a. If username and password do not exist in database, the user has the option to create the database based on the displayed option
        i. Selecting yes, creates a new user and inserts the login details into database (need to click login again to proceed to next screen)
    b. Username and password already exists in database
5. Login screen close, and Guest screen open
6. User needs to enter the requested background information, then click confirm
7. Application checks if all background information is provided
    a. If all information is not provided, the application will display a message stating that all requested information must be provided; then they will have to fill in remaining information before clicking confirm again

    b. If all information provided, a message will be displayed with all provided background information
        i. Selecting yes, reveals next set of information user must provide
8. Application reveals booking information that needs to be provided
9. User enters booking related information then clicks book room
    a. If all information is not provided, the application will display a message that all information must be provided; then they will have to fill in remaining information before clicking book room again
    b. If check-in date is a later date than check-out date, the application will not let the user proceed until the check-in & check-out dates are changed
    c. If the user selects a room type, room number, check-in & check-out date that is the same as another hotel guest, which already exists in the database, then the application will inform the user that they must alter the provided booking information
    d. If the user provides booking information that does not already exist in the database, meaning they make a booking that is available, the application will ask to confirm the booking; selecting yes displays all background & booking information provided by the user
        i. Clicking okay stops displaying aforementioned information
10. Guest screen close, login screen open

5.4.2.2 Hotel Admin

1. Open application
2. Login screen appears
3. Click on the admin radio button and enter credentials
4. Application checks if username and password exists in database
    a. If username and password do not exist in the database, the user has the option to create the admin.
    b. If the user selects Yes, a new Admin info screen appears where the user can create an admin account.
        i. After entering all the information the user hits create button from a new admin account is added into the database
    c. If the user selects No, the user will be prompted back to the same Login screen where they can enter different user credentials.
    d. The above process is repeated until the user enters a valid admin username and password that exists in the database.
5. After the admin credentials are verified the Admin is brought to Admin Portal
6. In Admin Portal
    a. Admin can Search for guest information based upon any of the following attributes- first name, last name, phone, email.
    b. Admin can update almost every information of any guest just by searching them .
    Note: Checking IN and Checking OUT needs to be updated every time an Admin tries to update a guest account.
    c. Admin can delete any guest entry present in the database
    d. Admin can create a new user by entering the information.

7. After performing all the operations Admin can close the Portal and it will log out of the system.

6. Implementation

6.1 Planning, Workflow Map, & Weekly meetings

** Updated from the plan shown in proposal **

Application development: ● Login user interface & related classes ● Hotel guest user interface & related classes ● Database interaction for hotel guest	Kunal (February 27, 2021 - March 5, 2021)
Virtual stand-up 1: Review of current development progress, and future planning ● Demonstration & explanation ● Allocation of work of respective team members	Team (March 6, 2021)
System Analysis & Design	Kunal (March 8, 2021) ● Define components of system ● Defining the database & schemas (hotel guest) ● Descriptions of system architecture (diagram) ● Project plan Heena (March 9, 2021 - March 12, 2021) ● Development of use case diagram ● Development of Sequence Diagram ● Defining the database & schemas (admin) Sukhleen (March 9, 2021 - March 12, 2021) ● Description of system architecture ● Defining the database & schemas (admin) Sehajpreet (March 9, 2021 - March 12, 2021) ● Description of system architecture ● Defining the database & schemas (admin) Team review (March 12, 2021)
Application development: ● Admin user interface & related classes	Sehajpreet(March 9, 2021 - March 18, 2021)

● Database interaction for admin ● Login user interface updated to accommodate for admin ● Admin Portal UI and functioning ● Database methods for searching, updating, deleting and retrieving a guest account.	
Virtual stand-up 2: Review of current development progress, and future planning ● Demonstration & explanation ● Allocation of work of respective team members	Team (March 19, 2021)
Application development: ● Continue to make progress in coding project and fulfill the basic requirements discussed in proposal ● Fix code for any errors ● Make changes based on virtual stand-up 2 ● Enhance application to look more visually appealing if time permits	Sehajpreet Singh Kingra (March 20, 2021 - March 28, 2021)
Features development and programming	Sukhleen ● User Stories ● Acceptance Tests ● User-interface diagram ● Sequence Diagram ● Activity Diagram Sehajpreet ● Defining classes Heena ● UML diagrams Kunal ● Introduction, general overview, and design considerations section of report ● Use Case diagram ● System Design sections of report related to hotel guest and some related to admin ● Planning development tasks and prototypes in report Team review (March 26, 2021)
Virtual stand-up 3: Review of current development progress, and future planning ● Demonstration & explanation ● Allocation of work of respective	Team (March 26, 2021)

team members	
Programming and development	Team (March 28, 2021 - April 5, 2021) ● Programming and wiring modules and layers Validation ● Heena (GuestInfo, AdminInfo) ● Kunal (Login, GuestInfo,GuestBooking, AdminInfo) Unit Testing ● Kunal (Login, GuestInfo,GuestBooking) ● Testing section of report Sehajpreet ● Adding sample data to application database upon fresh startup ● Pseudocode (Admin) ● Defining System Constraints/ Dependencies/Risks Prospective submission (April 8, 2021) Prospective presentation (April 9, 2021)
Application development: ● Fix date error in admin portal ● Created unit testing for admininfo and adminportal ● Added validations ● Enhance application to look more visually appealing if time permits	Kunal (April 6 - 7, 2021)
Project Presentation	Sehajpreet and Kunal (April 7,2021)

6.2 User Stories

As a *guest*

I want to be able to choose the days from calendar to make reservation for room.
And specify number of guests.

so that I would easily specify date I want to reserve room for.

Acceptance criteria

Given *guest logged in.*

When guest filled in all the credentials and specify dates to reserve room for.

Then system makes reservation by processing credentials.

As a *guest.*

I want to be able to make additional requests.

so that I can specify things I might require while booking.

Acceptance criteria

Given *guest logged in.*

When guest fills the guest information and guest booking details

Then system should have section for customers to make requests for additional accommodations they require.

As a *hotel admin*

I want to add new guest, delete, and update guest information.

so that I can perform those actions whenever required.

Acceptance criteria

Given *admin logged in.*

When admin add, delete and update information.

Then system reflects changes based on the action performed and update database.

As a *hotel admin*

I want to perform search operation on the guest list.

so that details of required guest could be accessed easily without having to go through whole list of the guests.

Acceptance criteria

Given *admin logged in.*

When admin perform search operation using name.

Then system shows the result for searched guest.

6.3 Prototype

Iteration/Prototype	Date	Link	Description
1.0	March 5, 2021	https://github.com/kjeshang/CSIS3275-TermProject/commit/558afaf9a8e643f78292c45b15069f3c3df01d92	Login user interface and hotel guest interface, as well as related database components & java classes
2.0	March 20, 2021	https://github.com/kjeshang/CSIS3275-TermProject/commit/74608b27b4fe270309e7965baa265e9ae28e66ba	Login user interface and admin user interface, as well as related database components & java classes
3.0	March 27, 2021	https://github.com/kjeshang/CSIS3275-TermProject/commit/0ea28e27028c25659979e5785cf5ab639b18a5e4	Login user interface and admin user interface, as well as related database components & java classes (** fixed bugs existing in previous version **)
4.0	Apr 1, 2021	https://github.com/kjeshang/CSIS3275-TermProject/commit/dc9c33ee543e58dbab89004cd084d8c43edd7b62	Initial validations added to GuestInfo and AdminInfo classes
5.0	Apr 2, 2021	https://github.com/kjeshang/CSIS3275-TermProject/commit/962d107fe1c890603d6480c054eb000313748e55	Updated validations added to GuestInfo and AdminInfo classes. Created validations for Login & GuestBooking. Implemented testing for admin/guest login, guest (& related classes) and database interaction
6.0	Apr 7, 2021	https://github.com/kjeshang/CSIS3275-TermProject/commit/5e21cf03d1ed3a521fa32502e606d73bae	Added another test case in GuestTest

		0e22d4	
6.1	Apr 7, 2021	https://github.com/kjeshang/CSIS3275-TermProject/commit/48cfdc31baf07d2e98eab5f8a7ebb6f012ff7402	Updated the code and structure for AdminGUI, AdminGUIPortal, & AdminDB, as well as added code for AdminTest

7. Testing

LoginTest

This class tests the validity of the parameters from the Login class. The parameters are username and password for either a patient or admin user. This class also tests if database interaction with the Login class is occurring successfully. The following static parameters were utilized to create a Login entity for this test class, and was consistently utilized for various tests.

- Username = testuser
- Password = testuser123

Although, when testing for invalid Login parameters, new Login entities were created for convenience and maintaining the integrity of the above described Login entity. When testing the above described Login entity in regards to database interaction, the GuestDB and AdminDB classes were instantiated in order to initialize the Guest_Login and Admin_Login collections, respectively.

Testing Summary

Test method	Requirement checks	Executed Test Cases	Results from Testing
shouldCreateLogin()	Login.getUsername() = username Login.getPassword() = password Login.validLogin() = true	AssertTrue for Login entity where ● Username = testuser ● Password = testuser123	Should create login entity as both username & password have a valid format, and prove its validity to be true
shouldNotCreateLogin()	Login.validLogin() = false	Return list of following test cases from invalidLoginList() method, where each list item is a new Login entity or the	Prove invalidity of logins due to username & password not being in the appropriate format

		already existing Login entity with changed parameter/s, and then AssertFalse: ● Username = empty string, password = empty string ● Username = a, password = a ● Username = testuser, password = tes ● Username = testuser, password = 123 ● Username = testuser, password = test ● Username = tes, password = testuser123	
shouldInteractWithDBCorrectlyGuestLogin()	guestDB.checkIfExists(login.getUsername(), login.getPassword()) = false guestDB.insertGuestLogin(login); guestDB.checkIfExists(login.getUsername(), login.getPassword()) = true guestDB.deleteGuestLogin(login)	AssertFalse when parameters from Login entity does not exist (i.e. not inserted or already deleted) in Guest_Login collection, where ● Username = testuser ● Password = testuser123 AssertTrue when parameters from Login entity does exist (i.e. already inserted or not	Should correctly check whether or not guest login exists in database, successfully insert guest login if it does not exist, and then successfully delete the aforementioned guest login from database

	guestDB.checkIfExists(login.getUsername(), login.getPassword()) = false	deleted) in Guest_Login collection, where ● Username = testuser ● Password = testuser123	
shouldInteractWithDBCorrectlyAdminLogin()	adminDB.checkIfExists(login.getUsername(), login.getPassword()) = false adminDB.insertGuestLogin(login); adminDB.checkIfExists(login.getUsername(), login.getPassword()) = true adminDB.deleteGuestLogin(login) adminDB.checkIfExists(login.getUsername(), login.getPassword()) = false	AssertFalse when parameters from Login entity does not exist (i.e. not inserted or already deleted) in Admin_Login collection, where ● Username = testuser ● Password = testuser123 AssertTrue when parameters from Login entity does exist (i.e. already inserted or not deleted) in Admin_Login collection, where ● Username = testuser ● Password = testuser123	Should correctly check whether or not admin login exists in database, successfully insert admin login if it does not exist, and then successfully delete the aforementioned guest login from database

<u>GuestTest</u>

This class tests the validity of the parameters from the GuestInfo & GuestBooking classes. In other words, tests the validity of prospective hotel guest's background information and room booking. This class also tests the database interaction of GuestInfo & GuestBooking classes with the Guest collection. The following static parameters were utilized to create a GuestInfo entity to be used on multiple testing methods.

- First name = Peter
- Last name = Parker
- Address = Queens, New York, New York
- Email = peter.parker@dailybugle.com
- Phone number = 1234567890
- Nationality = American

- Purpose of stay = Business

In addition, the following static parameters were utilized to create a GuestBooking entity to be used on multiple testing methods.

- Number of guests = 1
- Room type = Standard
- Room number = 100
- Check-in date = today
- Check-in date = tomorrow
- Lunch & dinner meal request = yes
- Additional accommodations = none

It should be noted that when testing for invalid parameters, some new GuestInfo and GuestBooking entities were created to maintain the integrity of the above described entities.

Testing Summary

Test method	Requirement checks	Executed test cases	Results from Testing
shouldCreateGuestInfo()	guestInfo.getFirstName() = first name guestInfo.getLastName() = last name guestInfo.getAddress() = address guestInfo.getEmail() = email guestInfo.getPhone() = phone number guestInfo.getNationality() = nationality guestInfo.getPurposeOfStay() = purpose of stay guestInfo.validInfo() = true	AssertTrue for GuestInfo entity, where • First name = Peter • Last name = Parker • Address = Queens, New York, New York • Email = peter.parker@dailybugle.com • Phone number = 1234567890 • Nationality = American • Purpose of stay = Business	Should create entity of guest background information with all valid parameters, and prove validity to be true
shouldNotCreateGuestInfo()	guestInfo.validInfo() = false	Return list of following test cases from invalidGuestInfoList() method, where each list item is a new GuestInfo entity or the already existing GuestInfo entity with changed parameter/s, and	Prove invalidity of guest background information due to incorrect format of required parameter

		then AssertFalse: ** Please refer to GuestTest Java class to see each test case**	
shouldCreateGuestBooking()	guestBooking.getNumPersons() = number of guests guestBooking.getRoomType() = Standard guestBooking.getRoomNumber() = 100 guestBooking.getCheckInDate() = check-in date guestBooking.getCheckOutDate() = check-out date guestBooking.isLunchAndDinner() = lunch and dinner request guestBooking.getAddAccomodations() = additional accommodations guestBooking.getLengthOfStay() = check-out date - check-in date guestBooking.getTotalCost() = total cost of stay guestBooking.validBooking() = true	AssertTrue for GuestBooking entity, where ● Number of guests = 1 ● Room type = Standard ● Room number = 100 ● Check-in date = today ● Check-in date = tomorrow ● Lunch & dinner meal request = yes ● Additional accommodations = none ● Length of stay = 1 ● Total cost of stay = 50	Should create entity of guest booking information with all valid parameters, and prove validity to be true
shouldNotCreateGuestBooking()	guestBooking.validBooking() = false	Return list of following test cases from invalidGuestBookingList() method, where each list item is a new GuestBooking	Prove invalidity of guest booking information due to incorrect format of required parameters

		entity or the already existing GuestBooking entity with changed parameter/s, and then AssertFalse: ** Please refer to GuestTest Java class to see each test case**	
shouldInteractWithDBCorrectlyGuest()	guestDB.checkIfReservationExists(guestBooking) guestDB.insertGuest(guestInfo, guestBooking) guestDB.checkIfReservationExists(guestBooking) guestDB.deleteRecord(guestInfo.getPhone()) guestDB.checkIfReservationExists(guestBooking)	AssertFalse when parameters from GuestInfo & GuestBooking entities does not exist (i.e. not inserted or already deleted) in Guest collection AssertTrue when parameters from GuestInfo & GuestBooking entities does exist (i.e. already inserted or not deleted) in Guest collection	Should correctly check whether or not guest (i.e. guest background information & reservation) exists in database, successfully insert guest if it does not exist, and delete successfully from database
shouldRetrieveFromDBCorrectlyGuest()	guestDB.findGuest(guestInfo) equals empty string = true guestDB.insertGuest(guestInfo, guestBooking) guestDB.findGuest(guestInfo) equals empty string = false	AssertTrue when parameters from guestInfo & guestBooking are not retrieved from Guest collection, in turn an empty string is retrieved AssertFalse when parameters from guestInfo & guestBooking are retrieved from Guest collection, in turn a string is retrieved with all parameters from GuestInfo and	Should correctly interact with guest collection in terms of data retrieval

		GuestBooking entities	

AdminTest

This class tests the validity of the parameters from the AdminInfo class. In other words, tests the validity of prospective admin's background information. Multiple different AdminInfo entities are utilized for various test cases of validity and invalidity. This class also tests the database interaction of the AdminInfo class with the Admin collection.

Testing Summary

Test method	Requirement checks	Executing test cases	Results from testing
validateAdminInfo()	adminInfo.validInfo() = false adminInfo.validInfo() = true	Each new AdminInfo entity within the method incrementally adds a valid parameter for testing until all parameters are valid with AssertTrue; otherwise the assertion is false	Validate Admin Info
shouldInteractWithDBCorrectlyAdminInfo()	adminDb.insertAdmin(getAdminInfo()) adminDb.findAdmin(getAdminInfo()).contains(getAdminInfo().getEmail())	Return AdminInfo entity from getAdminInfo(), insert parameters in Admin collection, then retrieve the appropriate record from the database, then assertTrue if stated admin email exists in the record	Successfully insert admin Info and verify it

TestAll

This test class is responsible for chaining all the test methods from LoginTest, GuestTest, and AdminTest together. If the TestAll test is initialized, all the test cases from the aforementioned classes run altogether. As this project was predominantly coded in Eclipse IDE, this Test was imperative to test all the test cases together and individually per class. If this project is run in IntelliJ IDE, then the TestAll class will stop the application from running due to a lack of the required dependencies. This is probably due to the fact that the Jupiter test case is in-built in the Eclipse and not in IntelliJ. It should be noted that IntellIJ allows for more than one test class to be run individually without issues. In conclusion, if one is using IntelliJ to run the application, the code within the TestAll class should be commented out.

8. Report on Experience

8.1. Lessons learned regarding technical and project management

- Choose a database that would be better suited and integrated with the IDE and language.
- Team Members should discuss and come up with a decision of using the same IDE taking into consideration ease and reliability of the IDE.
- Organize weekly meetings to keep track of the things to do, examine features that are implemented and discuss problem members are facing.
- Follow a unified approach and coding standard by using meaningful names and meaningful structure.
- Improvement in constructing easily-decipherable code and providing documentation.
- Improving debugging skills (i.e. software deconstruction/destruction)
- Planning and scheduling. Always estimate the project length and make it double, as a developer person might encounter unexpected problems.
- Doing Regression testing makes sure that changes do not affect current functionality.
- During addition of new features to the existing project, understand the existing one before changing pieces of code that might disintegrate the existing functionality.
- Following a particular development life cycle, improves the understandability of the system.

8.2. Lessons learned for future projects

- Instead of using Java Swing (which is very old), we would take the time to learn JavaFX to utilize in a similar project
- Improve Git and GitHub skills to maintain a simultaneous project workflow
- Planning and anticipating for development roadblocks in order to stay-on-course with project development schedule
- Communicate if you hit a roadblock and reach out for help
- Make logs to keep track of each feature, and distribute tasks among team members and set deadlines for those tasks.
- Conceiving a project that also takes on a test-driven approach, that allows us to identify issues faster.
- Make a rough sketch of the UI before actually designing it. Doing so will allow us to come up with new ideas.
- Do a survey in each meeting to mark things that went right, things that went wrong and things that need to be improved.
- Always do the pseudocode, to plan the algorithm before actual coding takes place which would eventually save lots of time as it would allow designers to focus on main logic without being distracted by syntax errors.
- Make the right selection of software development models by learning about Software Development Life Cycle (SDLC) models first and then assessing the needs of the stakeholders.
- Do MoSCoW analysis i.e. M(must), S(should), C(could) and W(Won’t)

9. References

1BestCsharp blog. (2019, May 15). Java Project Tutorial - How To Make a Hotel Management System Project In Java NetBeans | Part 1/10. YouTube. https://www.youtube.com/watch?v=VxrzKcBAAI4&list=PLFDH5bKmoNqxxu_coQDXHhUMJoCZu6C9q&index=1.

Amigoscode. (2021, March 28). Software Testing Tutorial - Learn Unit Testing and Integration Testing. YouTube. https://www.youtube.com/watch?v=Geq60OVyBPg&list=WL&index=9.

Anandyita, Gaby, P, M., Gupta, L., Warfront1, Pawa, … Mar. (2020, December 25). JUnit 5 Test Suites Examples. HowToDoInJava. https://howtodoinjava.com/junit5/junit5-test-suites-examples/.

Anil, Chandrakanth, Laxmi, Shekhar, Gla, Ranjan, S., … Lerin, M. (2020, December 25). MongoDB find document examples. HowToDoInJava. https://howtodoinjava.com/mongodb/mongodb-find-documents/.

Building Your First Junit 5 Test - Java Unit Testing with JUnit 5. Educative. (n.d.). https://www.educative.io/courses/java-unit-testing-with-junit-5/B892KY261z2.

Create today, tomorrow and yesterday date. Java Date Time - Create today, tomorrow and yesterday date. (n.d.). http://www.java2s.com/Tutorials/Java_Date_Time/Example/Date/Create_today_tomorrow_and_yesterday_date.htm.

Dehghani, A. (2021, February 12). Guide to JUnit 5 Parameterized Tests. Baeldung. https://www.baeldung.com/parameterized-tests-junit-5.

freeCodeCamp.org, & Programming Techie. (2021, January 12). Java Testing - JUnit 5 Crash Course. YouTube. https://www.youtube.com/watch?v=flpmSXVTqBI&list=LL&index=3.

harshaharsha 111 gold badge11 silver badge11 bronze badge, Shinwar ismailShinwar ismail 23522 silver badges66 bronze badges, Waruna WijesundaraWaruna Wijesundara 1111 bronze badge, Ole V.V.Ole V.V. 63.3k1111 gold badges9090 silver badges111111 bronze badges, Lakshitha rasangaLakshitha rasanga 111 bronze badge, & Shehan DeemanthaShehan Deemantha 111 bronze badge. (2018, March 1). How to set date TO JDATECHOOSER in java? https://stackoverflow.com/questions/44054813/how-to-set-date-to-jdatechooser-in-java/47927726.

How to Work with Embedded Document in MongoDB Collection. ObjectRocket How to Work with Embedded Document in MongoDB Collection Comments. (0AD).

https://kb.objectrocket.com/mongo-db/how-to-work-with-embedded-document-in-mongodb-collection-379.

how to wrap large message of JOptionPane.showConfirmDialog in scrollbar ... how to wrap large message of JOptionPane.showConfirmDialog in scrollbar ... (Swing / AWT / SWT forum at Coderanch). (0AD). https://coderanch.com/t/339970/java/wrap-large-message-JOptionPane-showConfirmDialog.

Java String contains() Method: Check Substring with Example. Guru99. (n.d.). https://www.guru99.com/string-contains-method-java.html#:~:text=The%20Java%20String%20contains(),used%20inside%20the%20if%20statement.

LANDCARZY, D. J. O. (2020, May 17). How to Create Hotel Management System with SQL Database in Java using Eclipse - Full Tutorial. YouTube. https://www.youtube.com/watch?v=DdAeTUpnCRs.

MacKay, J. (2018, January 11). The Ultimate Guide to Implementing Agile Project Management (and Scrum). Planio. https://plan.io/blog/what-is-agile-project-management/#step-1-set-your-project-vision-and-scope-with-a-planning-meeting.

MongoDB Driver Quick Start. Quick Start. (0AD). https://mongodb.github.io/mongo-java-driver/3.4/driver/getting-started/quick-start/.

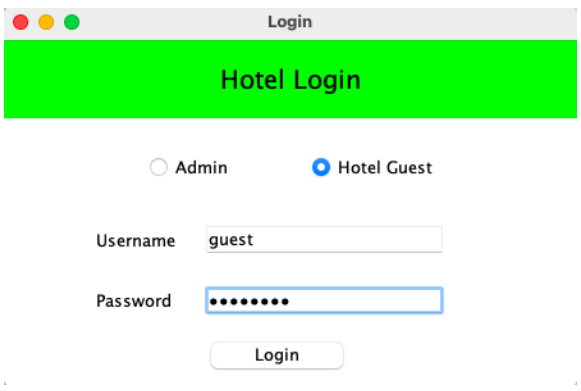
Person. (2014, August 7). Getting started with mongodb and java: Part i: Mongodb blog. https://www.mongodb.com/blog/post/getting-started-with-mongodb-and-java-part-i.

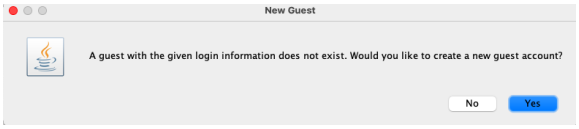
Person. (2014, August 14). Getting Started with MongoDB and Java: Part II: MongoDB Blog. MongoDB. https://www.mongodb.com/blog/post/getting-started-with-mongodb-and-java-part-ii.

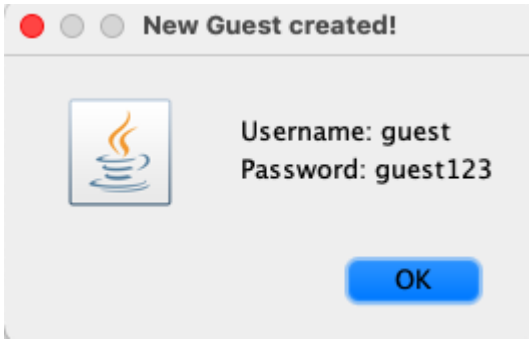
Spacey, J. (2017, May 23). 7 examples of software components. https://simplicable.com/new/software-components#:~:text=Software%20components%20are%20parts%20of,its%20implementation%20behind%20an%20interface.

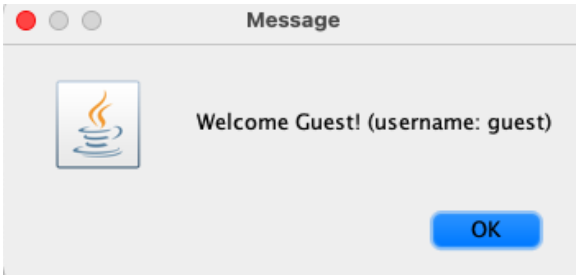
10. Appendices

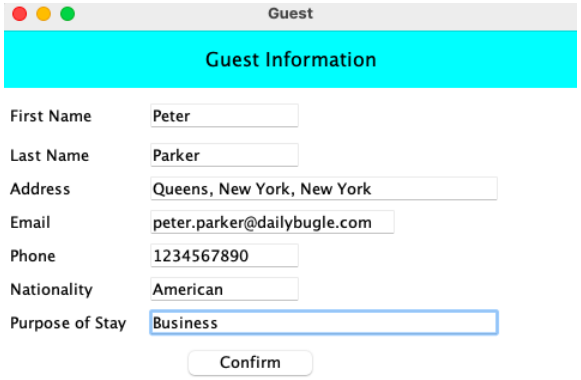
10.1. Hotel Guest Screens

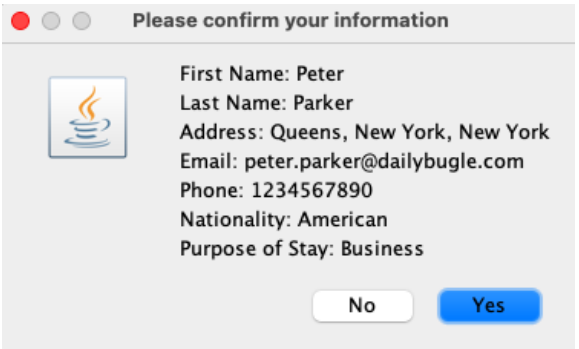


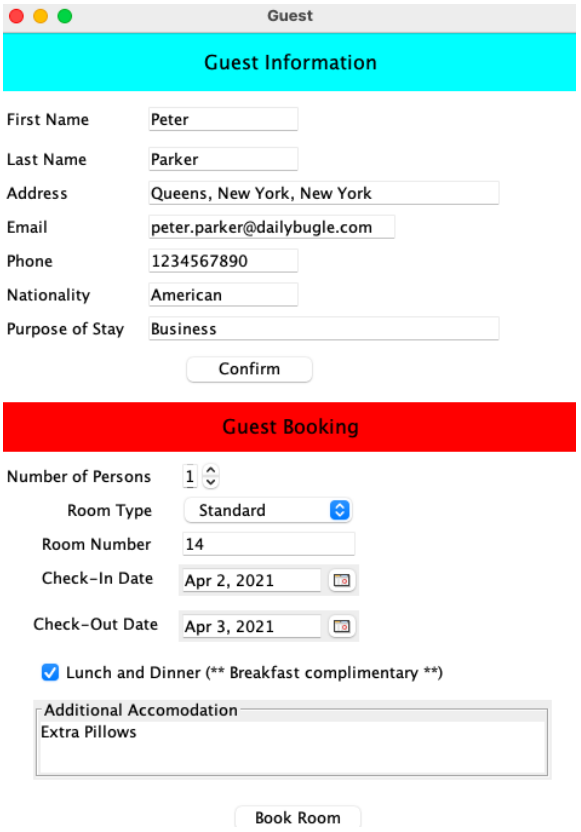


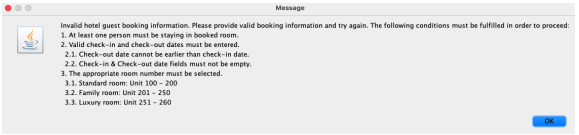


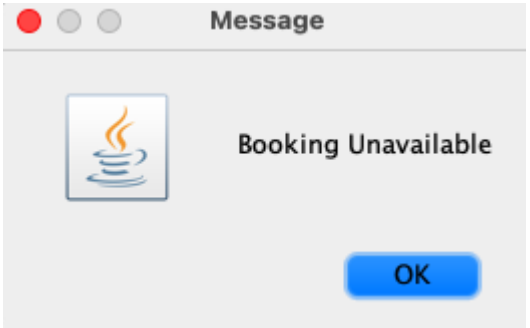


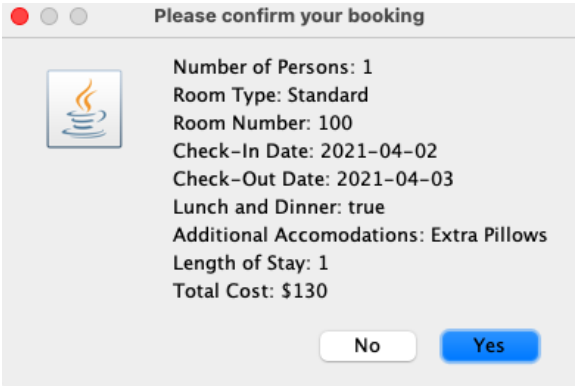


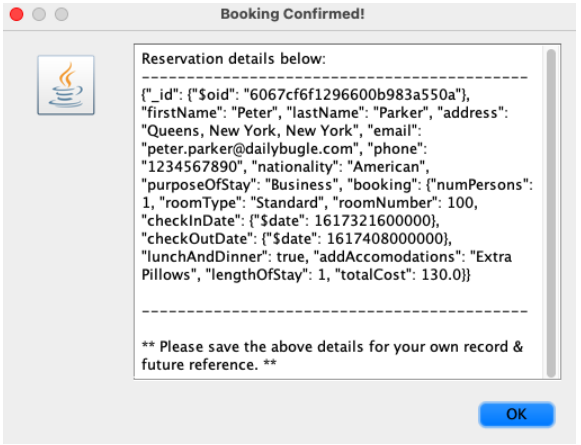

Please confirm your information
First Name: Peter
Last Name: Parker
Address: Queens, New York, New York
Email: peter.parker@dailybugle.com
Phone: 1234567890
Nationality: American
Purpose of Stay: Business
No
Yes


Guest
Guest Information
First Name
Peter
Last Name
Parker
Address
Queens, New York, New York
Email
peter.parker@dailybugle.com
Phone
1234567890
Nationality
American
Purpose of Stay
Business
Confirm
Guest Booking
Number of Persons
1
Room Type
Standard
Room Number
14
Check-In Date
Apr 2, 2021
Check-Out Date
Apr 3, 2021
Lunch and Dinner (** Breakfast complimentary **)
Additional Accomodation
Extra Pillows
Book Room


Message
Invalid hotel guest booking information. Please provide valid booking information and try again. The following conditions must be fulfilled in order to proceed:
1. At least one person must be staying in booked room.
2. Valid check-in and check-out dates must be entered.
2.1. Check-out date cannot be earlier than check-in date.
2.2. Check-in & Check-out date fields must not be empty.
3. The appropriate room number must be selected.
3.1. Standard room: Unit 100 - 200
3.2. Family room: Unit 201 - 250
3.3. Luxury room: Unit 251 - 260
OK


Message
Booking Unavailable
OK

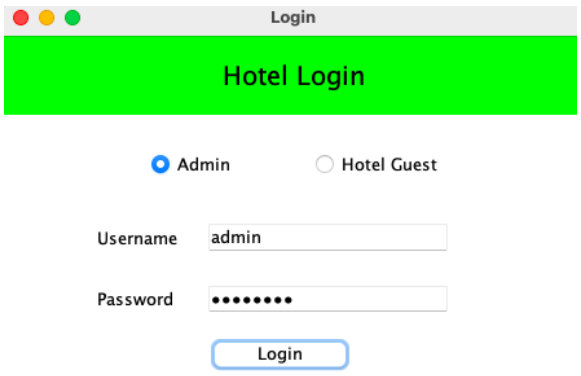

Please confirm your booking
Number of Persons: 1
Room Type: Standard
Room Number: 100
Check-In Date: 2021-04-02
Check-Out Date: 2021-04-03
Lunch and Dinner: true
Additional Accomodations: Extra Pillows
Length of Stay: 1
Total Cost: $130
No
Yes

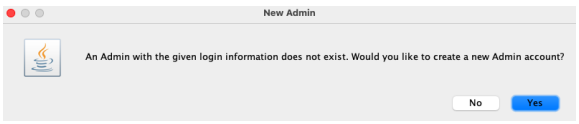

Booking Confirmed!
Reservation details below:

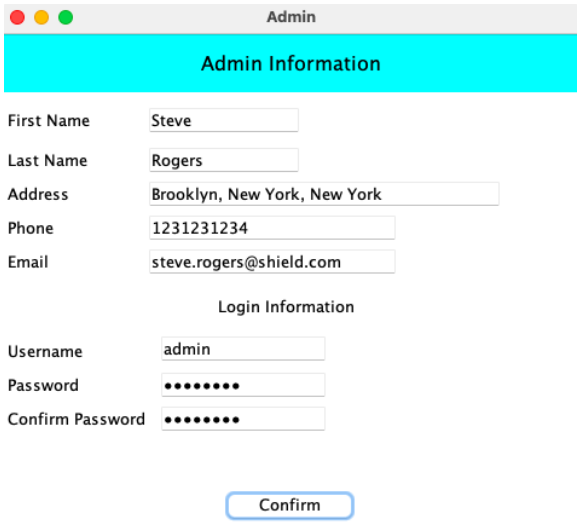
{"_id": {"$oid": "6067cf6f1296600b983a550a"}, "firstName": "Peter", "lastName": "Parker", "address": "Queens, New York, New York", "email": "peter.parker@dailybugle.com", "phone": "1234567890", "nationality": "American", "purposeOfStay": "Business", "booking": {"numPersons": 1, "roomType": "Standard", "roomNumber": 100, "checkInDate": {"$date": 1617321600000}, "checkOutDate": {"$date": 1617408000000}, "lunchAndDinner": true, "addAccomodations": "Extra Pillows", "lengthOfStay": 1, "totalCost": 130.0}}

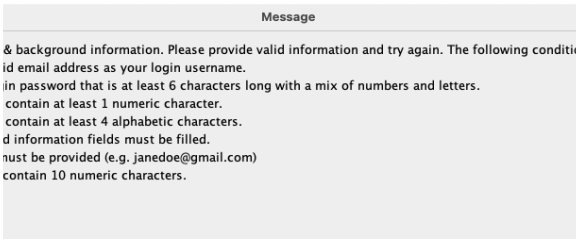
** Please save the above details for your own record & future reference. **
OK

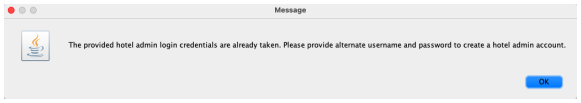
10.2. Admin Screens

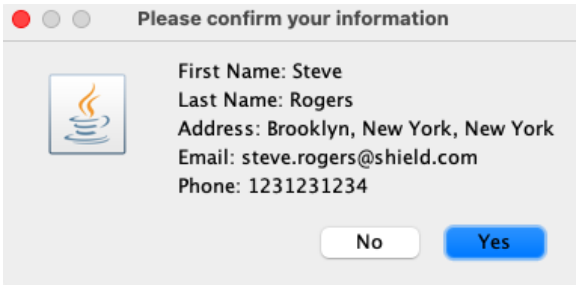


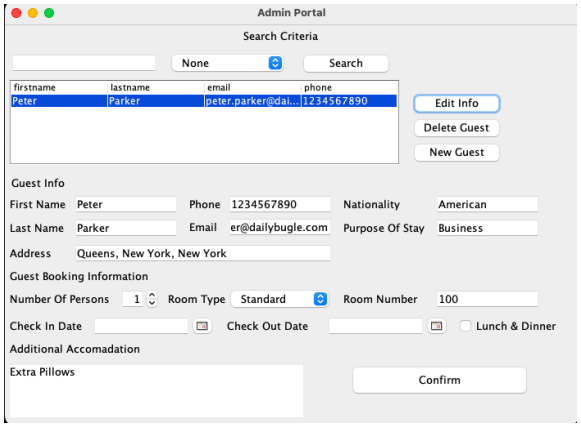


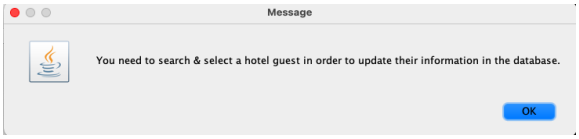


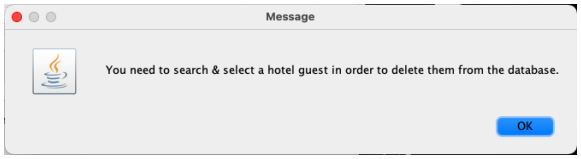




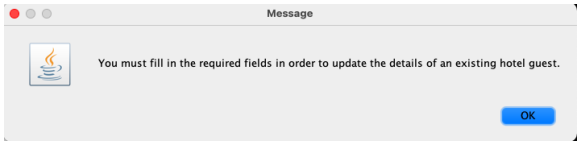



Admin Portal
Search Criteria
None
Search
firstname
lastname
email
phone
Peter
Parker
peter.parker@dai...
1234567890
Edit Info
Delete Guest
New Guest
Guest Info
First Name
Peter
Phone
1234567890
Nationality
American
Last Name
Parker
Email
er@dailybugle.com
Purpose Of Stay
Business
Address
Queens, New York, New York
Guest Booking Information
Number Of Persons
1
Room Type
Standard
Room Number
100
Check In Date
Check Out Date
Lunch & Dinner
Additional Accomadation
Extra Pillows
Confirm


Message
You need to search & select a hotel guest in order to update their information in the database.
OK


Message
You need to search & select a hotel guest in order to delete them from the database.
OK


Message
You must fill in the required fields in order to insert a new hotel guest.
OK


Message
You must fill in the required fields in order to update the details of an existing hotel guest.
OK