IT-SEP1X-A15

Hotel booking system

Group 5:	Martin Tomko, student number 239847
	Daniel Hamarik, student number 239857
	Atanas Latinov, student number 239841
	Stefan Alexiev, student number 240278
Supervisors:	Michael Vilhelmsen
	Mona Wendel Andersen

[15 December 2015]

VIA UC HORSENS

CONTENTS

LIST OF FIGURES AND TABLES

ABSTRACT

Our customer is the proprietor of a hotel with a long history and traditions that stretch back to the 15th century. His desire is to replace the old ledger that is used to contain the guests' information and bookings with an automated booking system as he sees that avoiding technology might hold back his business.

The system should enable its users to book rooms, check in and check out guests, as well as save that information and then be able to display it depending on the criteria selected by the user. Furthermore, editing reservations and adding expenses should be possible and an automated system for making rooms free should be created. These functionalities, along with a search bar that is able to give results depending on different search criteria and a user-friendly interface, are the core features that the system should contain.

The latest tests of the program have proven that it is a fully operational software product that satisfies all the requirements and demands and show that the program is capable of facilitating the room and guest management and providing assistance in the work of the receptionist.

I. INTRODUCTION

Over the past several decades the technological progress that has taken place, has influenced almost every aspect of our lives (McGrath, 2015) from the various industries to science and even the way we communicate. Without a doubt, information technology has radically altered the basis of business competition (Baines, 1998). We are witnessing a huge transition towards automated systems and technologies that stems from the aspiration to harness the power of the computer and use that power to boost the productivity and increase the efficiency in a certain business environment. Incorporating today's technology in business realizes its full potential and opens the way to countless new opportunities.

The Deer Alley Hotel, albeit small, is an establishment with a rich culture and a strong emphasis on tradition all the way from the rooms to the exterior. It offers accommodation to guests from all over the world who come to the hotel because of its rich cultural background and history stretching back to the 15th century. For this reason the owners have been hesitant to use modern equipment like computers and modern conveniences.

However, seeing how the business world is evolving and is incorporating technology more and more, the current owner realizes that if the hotel is to continue to be a successful business endeavor, technology must be applied in the form of a booking system. By replacing the old ledger with a booking system, the owner wants to be able to store the guest's information, book, check in or check out as well as look for rooms by either a date, number, guest name or availability. The program should also provide its users with the ability to see the daily schedule of check-ins and check-outs and also add additional expenses depending on what the guests use or require.

II. ANALYSIS

These analyses are derived from the interview with the proprietor of the hotel that has been given.

1. REQUIREMENTS

The requirements elicited from the interview are divided in two sections – functional and non-functional. The first section contains the services the system should be able to do, its interactivity with the user and also its range of capabilities whereas the other – the limitations on the functionality of the product.

1.1 FUNCTIONAL REQUIREMENTS

1. The system allows the receptionist to store the guest's name, date of birth, address, passport number, phone number, email, arrival date and date of departure and use this information for the booking as well as for check-in process.
2. The receptionist can check in or check out a guest or cancel a booking at any time and get familiar with the total cost of the stay for a specific number of days without additional expenses.
3. The receptionist can look for a room by room number, name of a guest and availability.
4. The receptionist can check a daily schedule that shows who is checking in or checking out.
5. The system automatically frees a room if the guest does not show up before 18:00.
6. The receptionist can reduce the price, add an extra bed and also add prices for room service, laundry and mini bar.
7. The system incorporates a user-friendly interface controlled by a mouse and keyboard.

1.2 NON-FUNCTIONAL REQUIREMENTS

1. The system should be developed using Java.
2. The system should be working offline.
3. The backend of the system should be file based.

2. USE CASE MODELING

The use cases used in the system are elicited from the requirements. They provide an overview of the main functions the program offers. There are use case descriptions that show the interactivity in a more detailed fashion.

2.1 USE CASE DIAGRAM

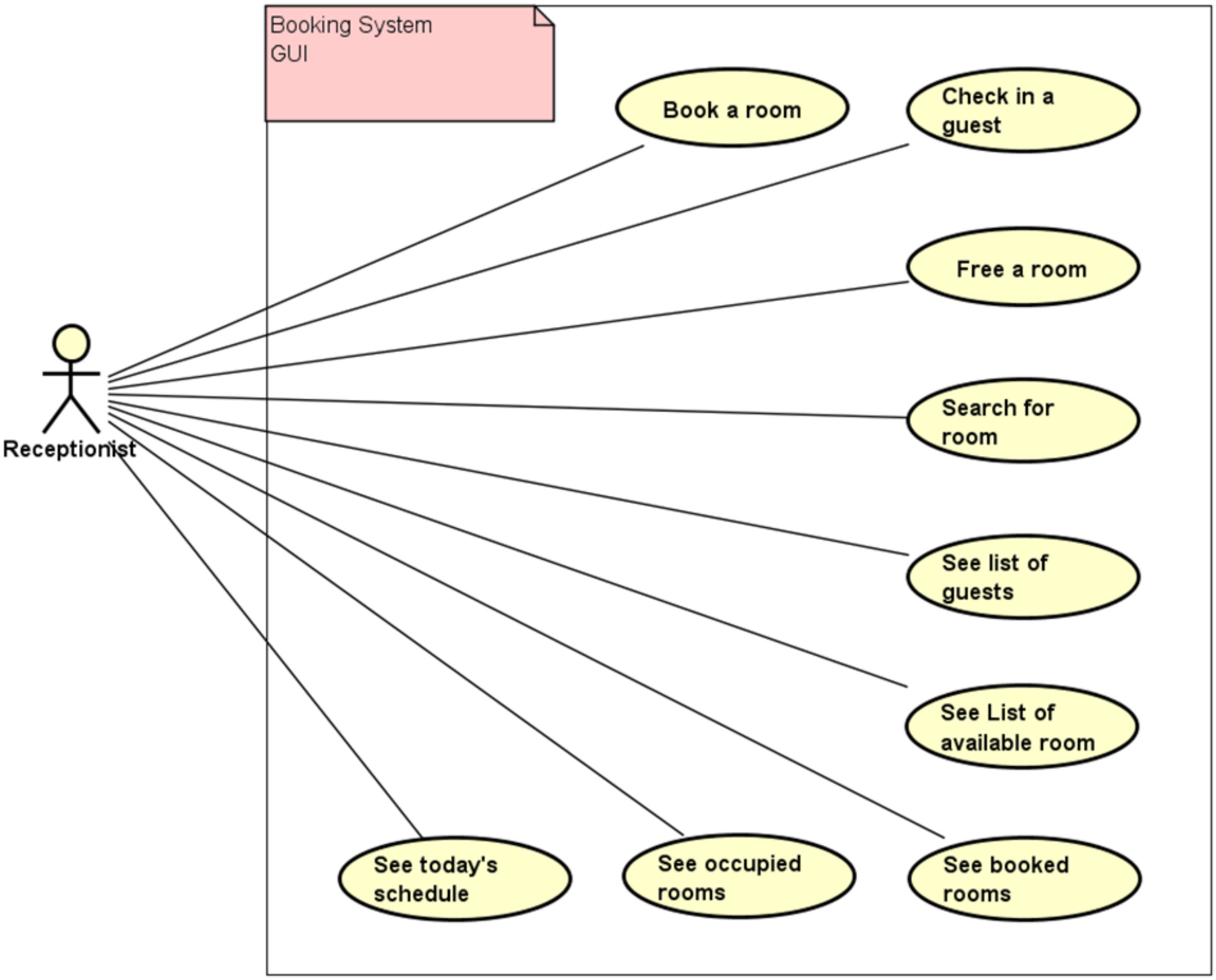
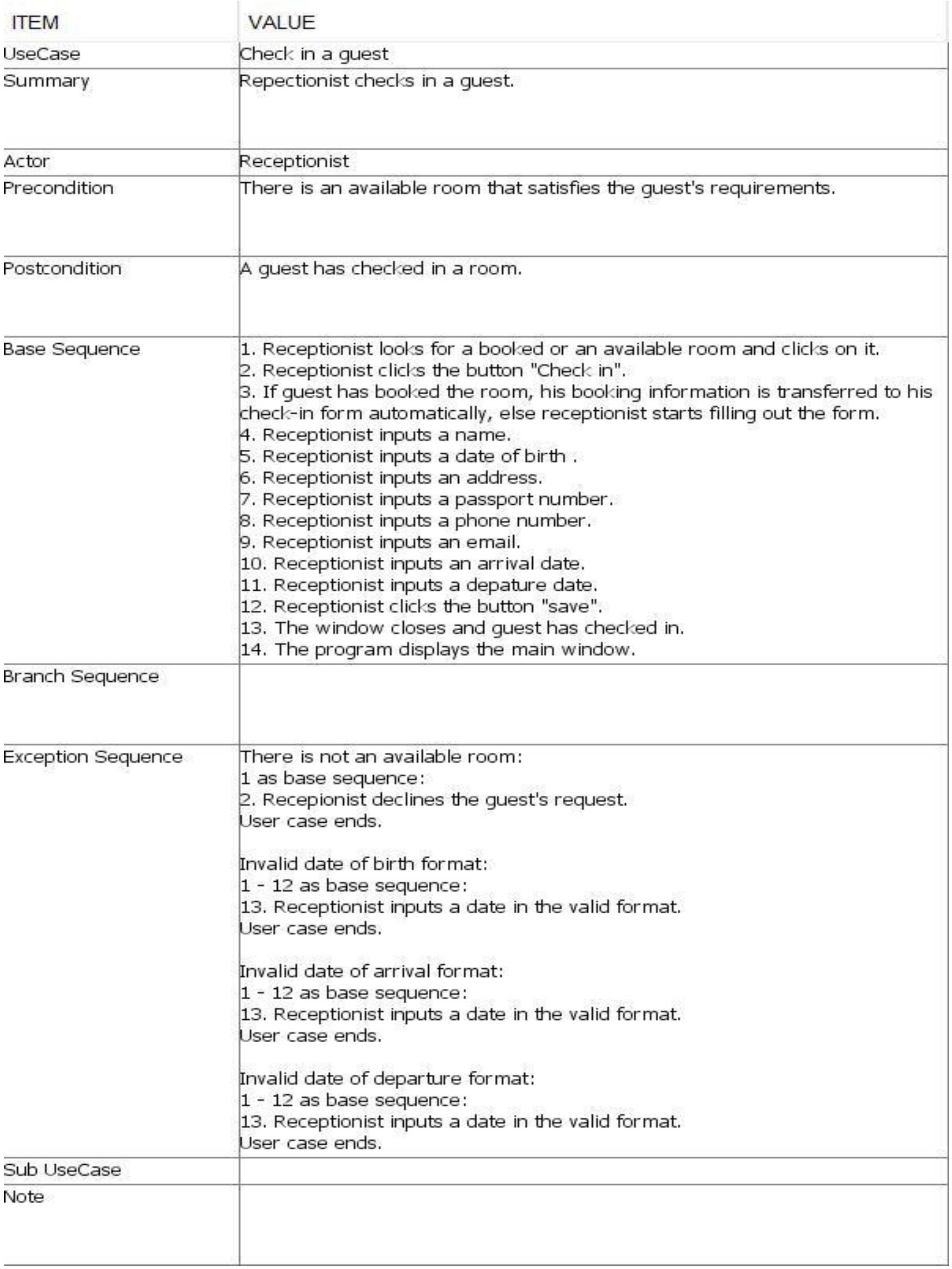


Figure 1. Hotel system use case diagram

2.2 USE CASE DESCRIPTION

Table 1: Use case description for Check in a guest



ITEM	VALUE
UseCase	Check in a guest
Summary	Repectionist checks in a guest.
Actor	Receptionist
Precondition	There is an available room that satisfies the guest's requirements.
Postcondition	A guest has checked in a room.
Base Sequence	1. Receptionist looks for a booked or an available room and clicks on it. 2. Receptionist clicks the button "Check in". 3. If guest has booked the room, his booking information is transferred to his check-in form automatically, else receptionist starts filling out the form. 4. Receptionist inputs a name. 5. Receptionist inputs a date of birth . 6. Receptionist inputs an address. 7. Receptionist inputs a passport number. 8. Receptionist inputs a phone number. 9. Receptionist inputs an email. 10. Receptionist inputs an arrival date. 11. Receptionist inputs a depature date. 12. Receptionist clicks the button "save". 13. The window closes and guest has checked in. 14. The program displays the main window.
Branch Sequence	
Exception Sequence	There is not an available room: 1 as base sequence: 2. Recepionist declines the guest's request. User case ends. Invalid date of birth format: 1 - 12 as base sequence: 13. Receptionist inputs a date in the valid format. User case ends. Invalid date of arrival format: 1 - 12 as base sequence: 13. Receptionist inputs a date in the valid format. User case ends. Invalid date of departure format: 1 - 12 as base sequence: 13. Receptionist inputs a date in the valid format. User case ends.
Sub UseCase	
Note	

2.3 ACTOR'S DESCRIPTION

The hotel booking system has only one actor – the User.

User: The user can be the proprietor of the hotel or the receptionist.

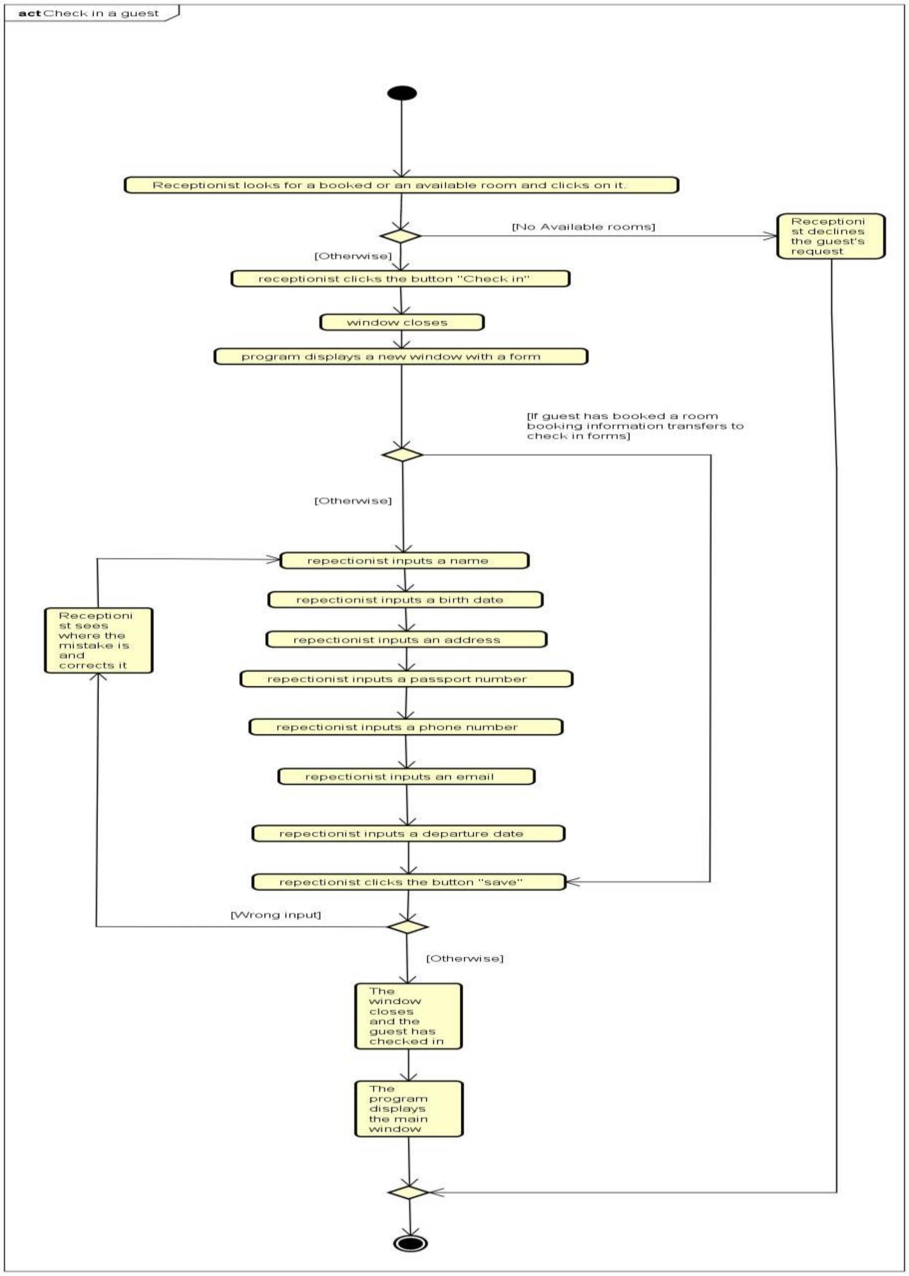

act Check in a guest
Receptionist looks for a booked or an available room and clicks on it.
[No Available rooms]
Receptionist declines the guest's request
[Otherwise]
receptionist clicks the button "Check in"
window closes
program displays a new window with a form
[If guest has booked a room booking information transfers to check in forms]
[Otherwise]
repectionist inputs a name
repectionist inputs a birth date
Receptionist sees where the mistake is and corrects it
repectionist inputs an address
repectionist inputs a passport number
repectionist inputs a phone number
repectionist inputs an email
repectionist inputs a departure date
repectionist clicks the button "save"
[Wrong input]
[Otherwise]
The window closes and the guest has checked in
The program displays the main window

Figure 2: Activity diagram

4. DESIGN

During the design stage the main focus is the class diagram as it describes the structure of a system by showing the system's classes, their attributes, methods, and the relationships among the objects.

4.1 CLASS DIAGRAM

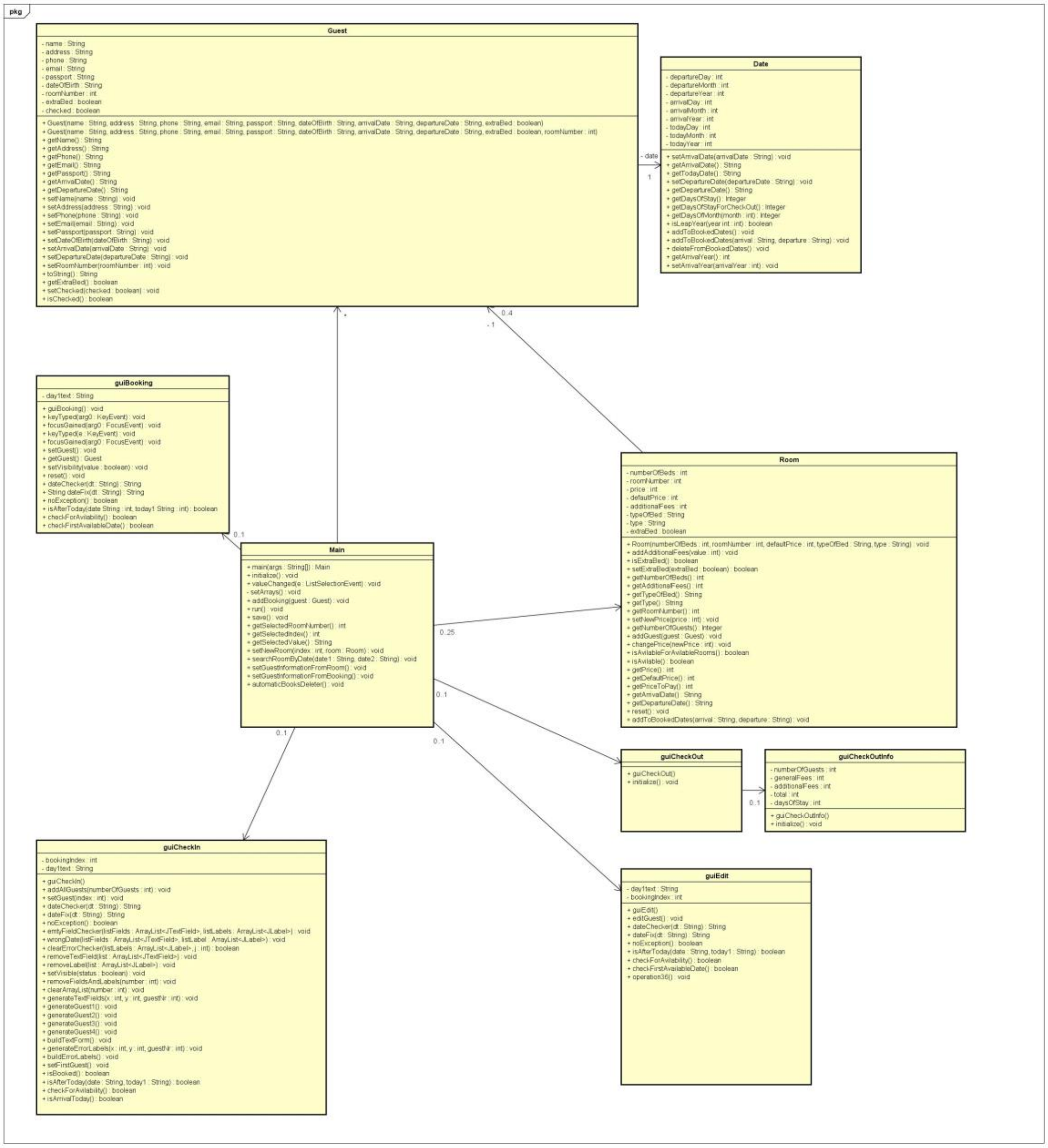

pkg
Guest
- name : String
- address : String
- phone : String
- email : String
- passport : String
- dateOfBirth : String
- roomNumber : int
- extraBed : boolean
- checked : boolean
+ Guest(name : String, address : String, phone : String, email : String, passport : String, dateOfBirth : String, arrivalDate : String, departureDate : String, extraBed : boolean)
+ Guest(name : String, address : String, phone : String, email : String, passport : String, dateOfBirth : String, arrivalDate : String, departureDate : String, extraBed : boolean, roomNumber : int)
+ getName() : String
+ getAddress() : String
+ getPhone() : String
+ getEmail() : String
+ getPassport() : String
+ getArrivalDate() : String
+ getDepartureDate() : String
+ setName(name : String) : void
+ setAddress(address : String) : void
+ setPhone(phone : String) : void
+ setEmail(email : String) : void
+ setPassport(passport : String) : void
+ setDateOfBirth(dateOfBirth : String) : void
+ setArrivalDate(arrivalDate : String) : void
+ setDepartureDate(departureDate : String) : void
+ setRoomNumber(roomNumber : int) : void
+ toString() : String
+ getExtraBed() : boolean
+ setChecked(checked : boolean) : void
+ isChecked() : boolean
- date
1
Date
- departureDay : int
- departureMonth : int
- departureYear : int
- arrivalDay : int
- arrivalMonth : int
- arrivalYear : int
- todayDay : int
- todayMonth : int
- todayYear : int
+ setArrivalDate(arrivalDate : String) : void
+ getArrivalDate() : String
+ getTodayDate() : String
+ setDepartureDate(departureDate : String) : void
+ getDepartureDate() : String
+ getDaysOfStay() : Integer
+ getDaysOfStayForCheckOut() : Integer
+ getDaysOfMonth(month : int) : Integer
+ isLeapYear(year int : int) : boolean
+ addToBookedDates() : void
+ addToBookedDates(arrival : String, departure : String) : void
+ deleteFromBookedDates() : void
+ getArrivalYear() : int
+ setArrivalYear(arrivalYear : int) : void
*
0..4
- 1
guiBooking
- day1text : String
+ guiBooking() : void
+ keyTyped(arg0 : KeyEvent) : void
+ focusGained(arg0 : FocusEvent) : void
+ keyTyped(e : KeyEvent) : void
+ focusGained(arg0 : FocusEvent) : void
+ setGuest() : void
+ getGuest() : Guest
+ setVisibility(value : boolean) : void
+ reset() : void
+ dateChecker(dt : String) : String
+ String dateFix(dt : String) : String
+ noException() : boolean
+ isAfterToday(date String : int, today1 String : int) : boolean
+ checkForAvilability() : boolean
+ checkFirstAvailableDate() : boolean
0..1
Main
+ main(args : String[]) : Main
+ initialize() : void
+ valueChanged(e : ListSelectionEvent) : void
- setArrays() : void
+ addBooking(guest : Guest) : void
+ run() : void
+ save() : void
+ getSelectedRoomNumber() : int
+ getSelectedIndex() : int
+ getSelectedValue() : String
+ setNewRoom(index : int, room : Room) : void
+ searchRoomByDate(date1 : String, date2 : String) : void
+ setGuestInformationFromRoom() : void
+ setGuestInformationFromBooking() : void
+ automaticBooksDeleter() : void
0..25
0..1
0..1
0..1
Room
- numberOfBeds : int
- roomNumber : int
- price : int
- defaultPrice : int
- additionalFees : int
- typeOfBed : String
- type : String
- extraBed : boolean
+ Room(numberOfBeds : int, roomNumber : int, defaultPrice : int, typeOfBed : String, type : String) : void
+ addAdditionalFees(value : int) : void
+ isExtraBed() : boolean
+ setExtraBed(extraBed : boolean) : boolean
+ getNumberOfBeds() : int
+ getAdditionalFees() : int
+ getTypeOfBed() : String
+ getType() : String
+ getRoomNumber() : int
+ setNewPrice(price : int) : void
+ getNumberOfGuests() : Integer
+ addGuest(guest : Guest) : void
+ changePrice(newPrice : int) : void
+ isAvilableForAvilableRooms() : boolean
+ isAvilable() : boolean
+ getPrice() : int
+ getDefaultPrice() : int
+ getPriceToPay() : int
+ getArrivalDate() : String
+ getDepartureDate() : String
+ reset() : void
+ addToBookedDates(arrival : String, departure : String) : void
guiCheckOut
+ guiCheckOut()
+ initialize() : void
0..1
guiCheckOutInfo
- numberOfGuests : int
- generalFees : int
- additionalFees : int
- total : int
- daysOfStay : int
+ guiCheckOutInfo()
+ initialize() : void
guiCheckIn
- bookingIndex : int
- day1text : String
+ guiCheckIn()
+ addAllGuests(numberOfGuests : int) : void
+ setGuest(index : int) : void
+ dateChecker(dt : String) : String
+ dateFix(dt : String) : String
+ noException() : boolean
+ emtyFieldChecker(listFields : ArrayList<JTextField>, listLabels : ArrayList<JLabel>) : void
+ wrongDate(listFields : ArrayList<JTextField>, listLabel : ArrayList<JLabel>) : void
+ clearErrorChecker(listLabels : ArrayList<JLabel>, j : int) : boolean
+ removeTextField(list : ArrayList<JTextField>) : void
+ removeLabel(list : ArrayList<JLabel>) : void
+ setVisible(status : boolean) : void
+ removeFieldsAndLabels(number : int) : void
+ clearArrayList(number : int) : void
+ generateTextFields(x : int, y : int, guestNr : int) : void
+ generateGuest1() : void
+ generateGuest2() : void
+ generateGuest3() : void
+ generateGuest4() : void
+ buildTextForm() : void
+ generateErrorLabels(x : int, y : int, guestNr : int) : void
+ buildErrorLabels() : void
+ setFirstGuest() : void
+ isBooked() : boolean
+ isAfterToday(date : String, today1 : String) : boolean
+ checkForAvilability() : boolean
+ isArrivalToday() : boolean
guiEdit
- day1text : String
- bookingIndex : int
+ guiEdit()
+ editGuest() : void
+ dateChecker(dt : String) : String
+ dateFix(dt : String) : String
+ noException() : boolean
+ isAfterToday(date : String, today1 : String) : boolean
+ checkForAvilability() : boolean
+ checkFirstAvailableDate() : boolean
+ operation36() : void

Figure 3. Class diagram

4.2 DESIGN

The system is designed in such a way that at almost all times the user has only to interact with the main window as all the information is available there. The user is not sent to another window when looking for a room or for the information about the guest in a specific room. This makes the interface extremely user-friendly and intuitive.

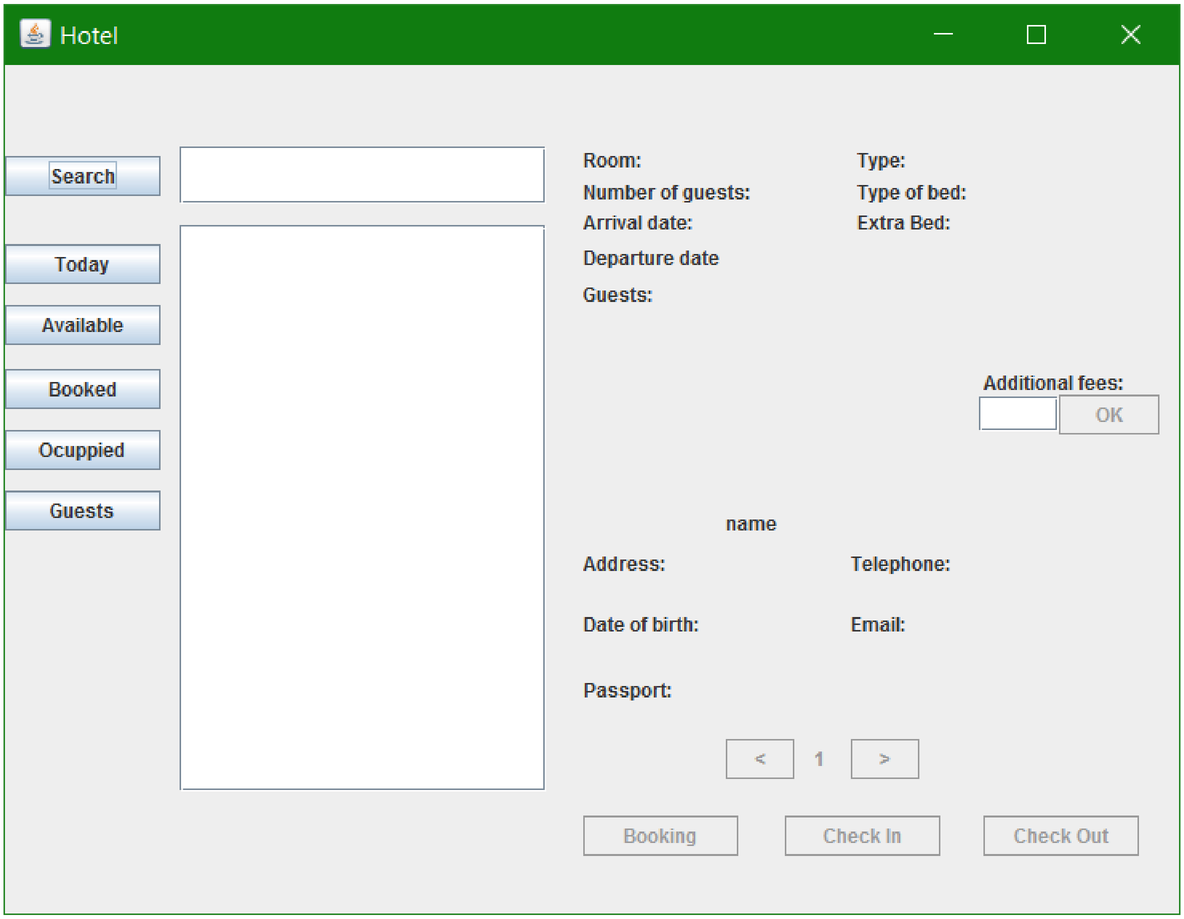
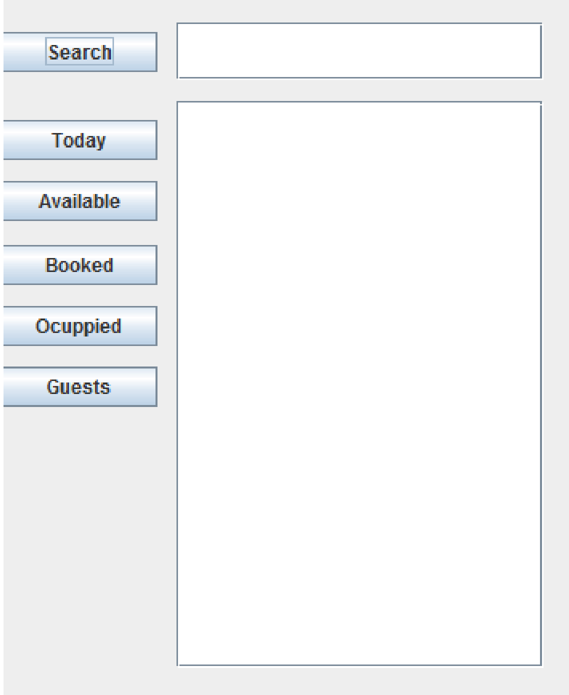


Figure 4. Main window

The left part of the main window is designed to contain the main functions of the system with the search bar and the result tab right next to the buttons.



.

Figure 5. Navigation & results

The right part of the main window displays details about the room that was clicked on the information tab and also information about the guest who can be up to 4 people. In addition the user can add supplementary fees if necessary.

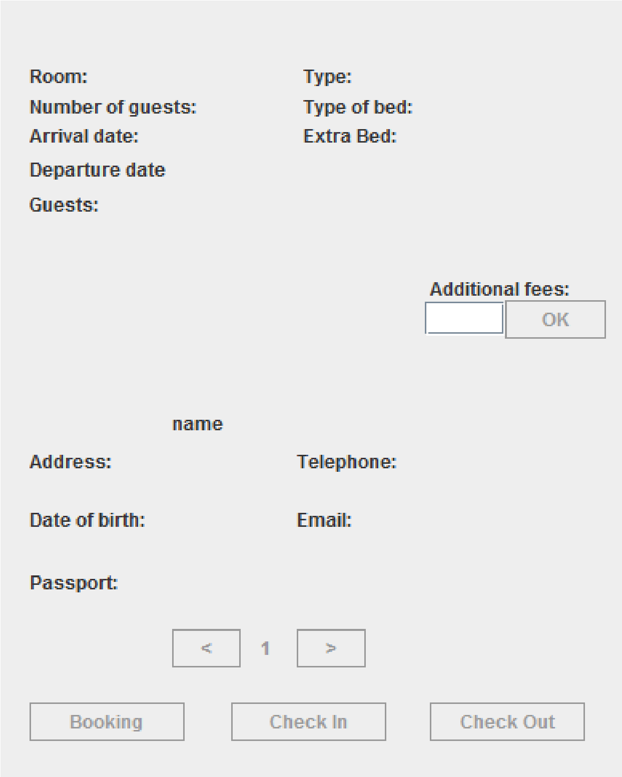


Figure 6. Information tab

5. TEST AND RESULT

Tests were carried out on each use case as well as the automated delete feature and the smaller functions such as input validations and overlapping date periods checks. Several guests were saved in the system with some bookings and check-ins and coinciding date ranges were set to check whether rooms can be booked by two guests on the same dates. The results are summarized in the table below.

Table 2. Test results

Use cases	Result
See a list of available rooms	Successful
See a list of available rooms	Successful
See a list of booked rooms	Successful
See a list of occupied rooms	Successful
See a list of the guests	Successful
Search by room number, guest name, arrival or departure date	Successful
Validation of guest information on input	Successful
Adding additional fees	Successful
Preventing check-ins or bookings on overlapping date periods	Successful
Automated freeing of rooms	Successful
Information about room	Successful
Delete bookings	Successful
Edit booking information	Successful
Information about guest	Successful

6. DISCUSSION

The important thing with such projects is to expand and build upon the requirements and demands of the client and not to alter or add unnecessary functionalities. The feature that we are most proud of is the search bar simply because it follows this rule. It augments the user experience tremendously because it expedites the work process and increases productivity but does not include any undesired features. The great thing about the search bar is that a search could be made with one of four parameters: a guest name, a room number, an arrival date or a departure date. The input restrictions are less and the user could type in the bar a parameter and receive a group of results that satisfy the criteria e.g. a time slot in which several rooms are booked and the result is a list of those rooms. This feature along with the stability of the program and the user friendly interface make it really stand out as a great software product that really deserves the users' approval.

7. CONCLUSION

The aim of the project was to design and create a dynamic hotel booking system that would improve the check-in and booking processes overall. The system should enable its users to store the guests' information when booking or checking in as well as automating processes such as validation of input and prevention of bookings with overlapping date periods. Furthermore the user can search for a specific room, add additional expenses, edit information and see a daily schedule of the upcoming guest arrivals or departures. The end result is an extremely stable application that satisfies all the requirements, offers a great deal of functionality and has a very user-friendly interface that complies with the latest software standards. In conclusion, the goals of the project are met and all the demands are fulfilled.

8. LIST OF REFERENCES

[1] ("Technology and tourism" - Work Study, Baines, Anna 1998.)

[2] (“Effects of Technology on Business" - HowStuffWorks, McGrath, Jane, December 2015.)