Linnéuniversitetet
Kalmar Växjö

Task 2:Re-engineering Legacy Software

Software Engineering - Design

- 2DV603

Member: Chaoran Meng
Member: Meng Li
Email: cm222in@student.lnu.se
Email: ml223tu@student.lnu.se

Contents

Design Document

This document will describe the basic principles, solutions, and core functions of the design concept of the Linnaeus Hotel Management program, as well as the important components of the program.

1 Purpose

“Linnaeus Hotel Management” is the system that manages reception activities at “Linnaeus Hotel”. According to this characteristic, our system will have the functions that the general hotel system has. That is the reservations, check-in and check-out. We will record the guests' personal information as the initial function.
Because a hotel has many different room types, room quality, room requirements. We will add these features to our program to improve our program. And all of this will be stored in the database, making it easy to call.And considering that this system will be used in two places together, we will consider using a networked database to store this data.Hotel staff will be able to view the customer's personal information, room number, room type, check-in time, check-out time, etc. through the system.
To avoid guests standing in a line while waiting for the checkout, the whole procedure must take in average less than 60 seconds to complete.

2 General priorities

One of our highest priorities is simplicity. Not only is the program easy to understand, the program's GUI should also look simple and easy to use.Because of some time requirements, we will design GUI options very easy to use, all options are very clear to see. This will enable the staff to increase work efficiency and meet the time requirements.
Next, we consider the reusability and maintainability of the program. Because some functions need to be used repeatedly, we will classify the methods one by one, so that you only need to change the interface to use it. This can also improve work efficiency, and when you find that some method is wrong, you only need to find the class of that method and correct it. This is very easy to maintain our program.

Afterwards, we will consider the scalability of the system. When the demand changes, we can also add features to avoid the impact of the whole body. We may only need to add some features without breaking the original system architecture. We will also consider the generality of the system. Our program will be written and developed using java. Because of the versatility of the Java language, our program can work on all the required platforms.

3 Outline of the design

After reviewing the legacy system, we first think that the legacy system does not have a good architectural model. The structure is very confusing and reading code is very inconvenient. We can see the picture " jhotel uml class diagram".This diagram shows us the uml class of legacy system, and we can see that this diagram is very large, complex and confusing. Because there is no good architectural pattern, we have found that the code is very messy to write together, some methods and data written in the same class, which is very unreasonable. Because these important data are not encapsulated. if you encapsulate this data, then users who use this class cannot easily manipulate the data structure directly, but can only execute data that the class allows to expose. This can avoid the external influence. On the internal data and improve the maintainability of the program.

Second, And as you can see from “jhotel uml class diagram”, because the legacy system does not have a good architectural model, it is difficult for us to modify the legacy system. This means that legacy system do not consider future updates. This is very detrimental to the program's development and updateability.

Third, when we run code, the program reacts very slowly because the legacy system writes methods, data, and GUI together, causing many methods and data not to be reused. Failure to increase the reusability of programs is a failure of the legacy system.

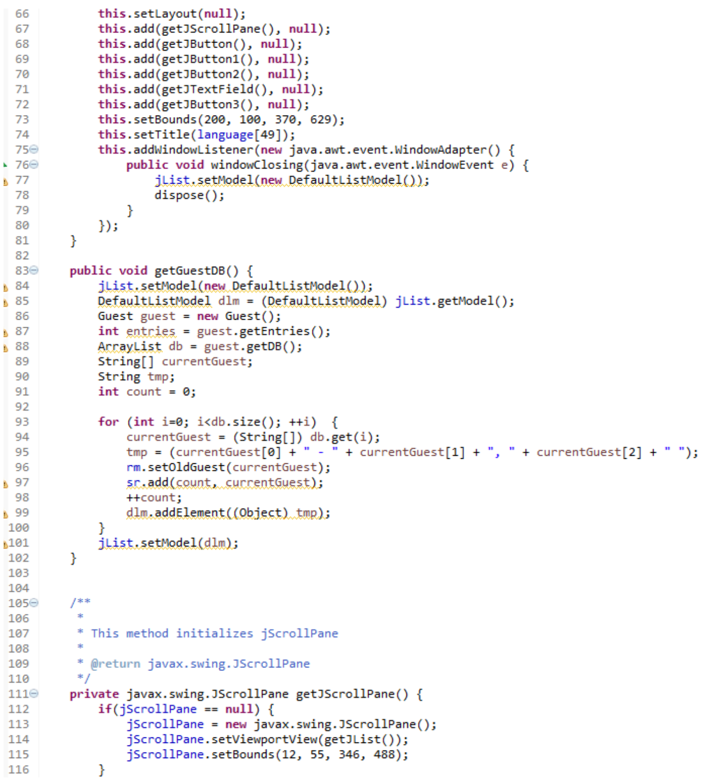
Finally, We can see figure 2， we think that the legacy system's GUI is very complex and unfriendly. And, because of the method, the data and the GUI cannot be separated, resulting in a GUI interface that cannot be easily modified.
Because the jhotel uml class diagram is too large, I stored the picture in the diagram folder (picture name is jhotel uml class diagram)

Figure3 is a uml class diagram presented based on MVC pattern. As can be seen from this diagram, Model is represented the application core, such as a list of data records, the controller controls the operation of the whole program. The MVC pattern enabled them to improve system flexibility and reusability.

Next, I'll cover some benefits of MVC pattern.
(1) a model provides different representations of multiple views and can create new views for a model without rewriting the model. Once the data of the model changes, the model notifies the relevant views, and each view refreshes itself accordingly.
(2) the model can be reused. Because the model is view-independent, you can port a model to work independently on a new platform.
(3) improve development efficiency. When developing the display part of the interface, you only need to consider how to layout a good user interface. When developing a model, you only need to consider the business logic and data maintenance, which enables developers to focus on one aspect of development and improve development efficiency.

That's why we decided to use the MVC architectural pattern.

Figure 2



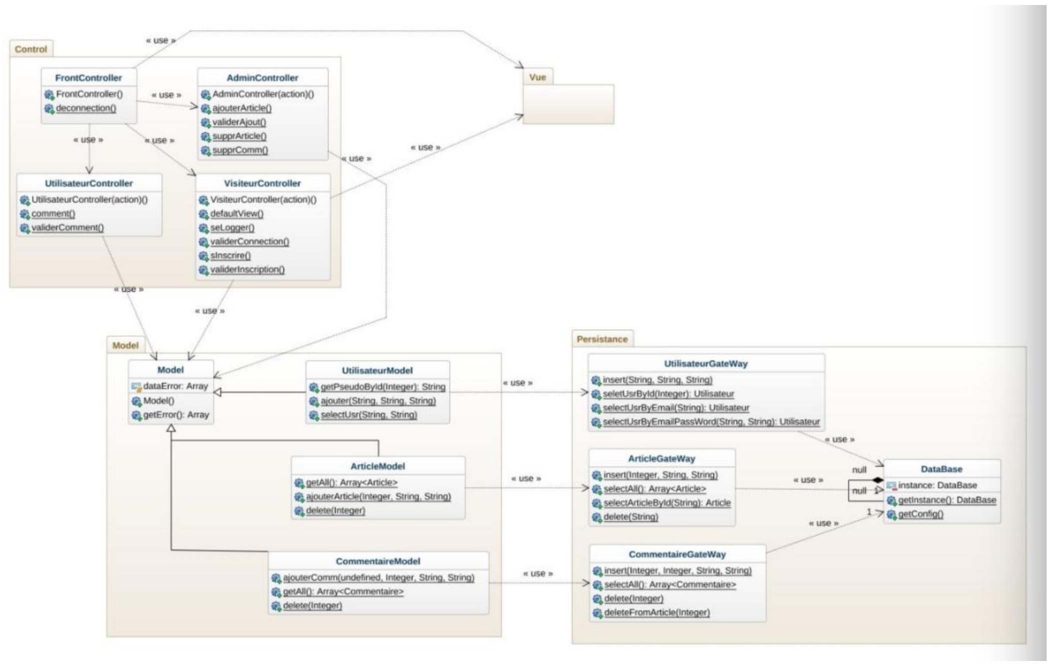
```
        this.setLayout(null);
        this.add(getJScrollPane(), null);
        this.add(getJButton(), null);
        this.add(getJButton1(), null);
        this.add(getJButton2(), null);
        this.add(getJTextField(), null);
        this.add(getJButton3(), null);
        this.setBounds(200, 100, 370, 629);
        this.setTitle(language[49]);
        this.addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                jList.setModel(new DefaultListModel());
                dispose();
            }
        });
    }

    public void getGuestDB() {
        jList.setModel(new DefaultListModel());
        DefaultListModel dlm = (DefaultListModel) jList.getModel();
        Guest guest = new Guest();
        int entries = guest.getEntries();
        ArrayList db = guest.getDB();
        String[] currentGuest;
        String tmp;
        int count = 0;

        for (int i=0; i<db.size(); ++i)  {
            currentGuest = (String[]) db.get(i);
            tmp = (currentGuest[0] + " - " + currentGuest[1] + ", " + currentGuest[2] + " ");
            rm.setOldGuest(currentGuest);
            sr.add(count, currentGuest);
            ++count;
            dlm.addElement((Object) tmp);
        }
        jList.setModel(dlm);
    }


    /**
     *
     * This method initializes jScrollPane
     *
     * @return javax.swing.JScrollPane
     */
    private javax.swing.JScrollPane getJScrollPane() {
        if(jScrollPane == null) {
            jScrollPane = new javax.swing.JScrollPane();
            jScrollPane.setViewportView(getJList());
            jScrollPane.setBounds(12, 55, 346, 488);
        }
```

Figure 3


Control
FrontController
FrontController()
deconnection()
AdminController
AdminController(action)()
ajouterArticle()
validerAjout()
supprArticle()
supprComm()
UtilisateurController
UtilisateurController(action)()
comment()
validerComment()
VisiteurController
VisiteurController(action)()
defaultView()
seLogger()
validerConnection()
sInscrire()
validerInscription()
Vue
« use »
Model
Model
dataError: Array
Model()
getError(): Array
UtilisateurModel
getPseudoById(Integer): String
ajouter(String, String, String)
selectUsr(String, String)
ArticleModel
getAll(): Array<Article>
ajouterArticle(Integer, String, String)
delete(Integer)
CommentaireModel
ajouterComm(undefined, Integer, String, String)
getAll(): Array<Commentaire>
delete(Integer)
Persistance
UtilisateurGateWay
insert(String, String, String)
seletUsrById(Integer): Utilisateur
selectUsrByEmail(String): Utilisateur
selectUsrByEmailPassWord(String, String): Utilisateur
ArticleGateWay
insert(Integer, String, String)
selectAll(): Array<Article>
selectArticleById(String): Article
delete(String)
CommentaireGateWay
insert(Integer, Integer, String, String)
selectAll(): Array<Commentaire>
delete(Integer)
deleteFromArticle(Integer)
DataBase
instance: DataBase
getInstance(): DataBase
getConfig()
null
null
1
1

4 Major design issues

The problem we encountered was how to solve the connection problem of network data,How should we share data between two hotels. Upon request, we know that this plan will be used for two buildings (one on the Växjö campus and one on the Kalmar campus). If the guests on the Växjö campus and Kalmar campus choose Room A at the same time, problems will arise. The same room cannot be used by two guests at the same time. In general, we will store data in a database so that we can easily call it. However, according to the situation we encountered, if we use a general database (such as a local database), the data cannot be shared, resulting in the inability of hotels in different regions to update the data when using our program. Therefore, using a local database is not possible. After our discussion, we decided to use the SQL database as our

database. When the guests of the Växjö campus chose room A and the data was stored in the SQL database, the staff at the Kalmar campus found that the room could not be selected to avoid the above error.
We downloaded MySQL and MySQL Workbench from the Internet. Using these two softwares, we can store data in a SQL database.

5 Design details

5.1 Reference Architectural Pattern

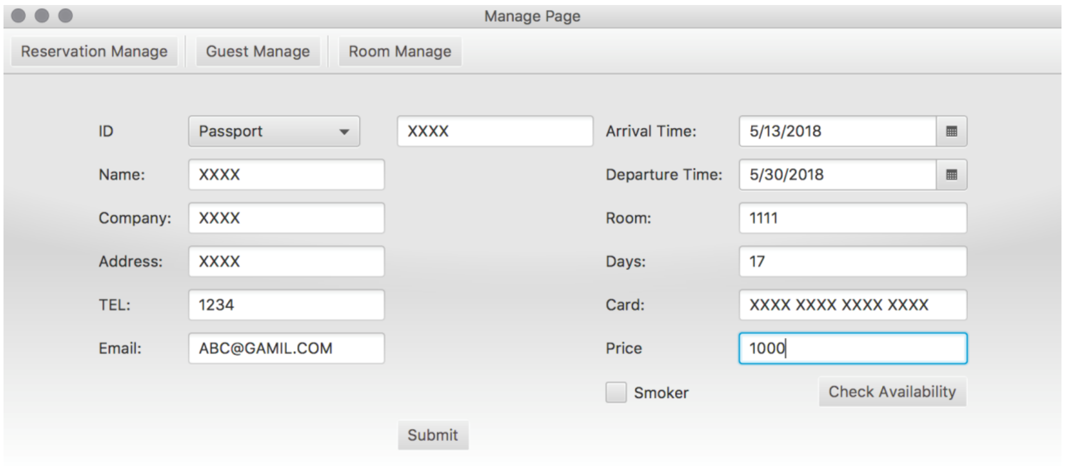
For Architectural Pattern, we refer to the MVC pattern. Not only have we learned this pattern, but the benefits of this pattern are numerous.The purpose of the MVC pattern is to implement a dynamic program design that simplifies the subsequent modification and expansion of the program and makes it possible to reuse a certain part of the program. In addition, this model makes the program structure more intuitive by simplifying the complexity. The software system, by separating itself from the basic part, also gives the functions that each basic part should have.And as described above, we can easily test our program, and increase the program's scalability.

5.2 The set of components

5.2.1 Discuss both requires and provides interface

We will discuss through four major functions. The four functions are check-in, check-out, reservations and edit rooms respectively.

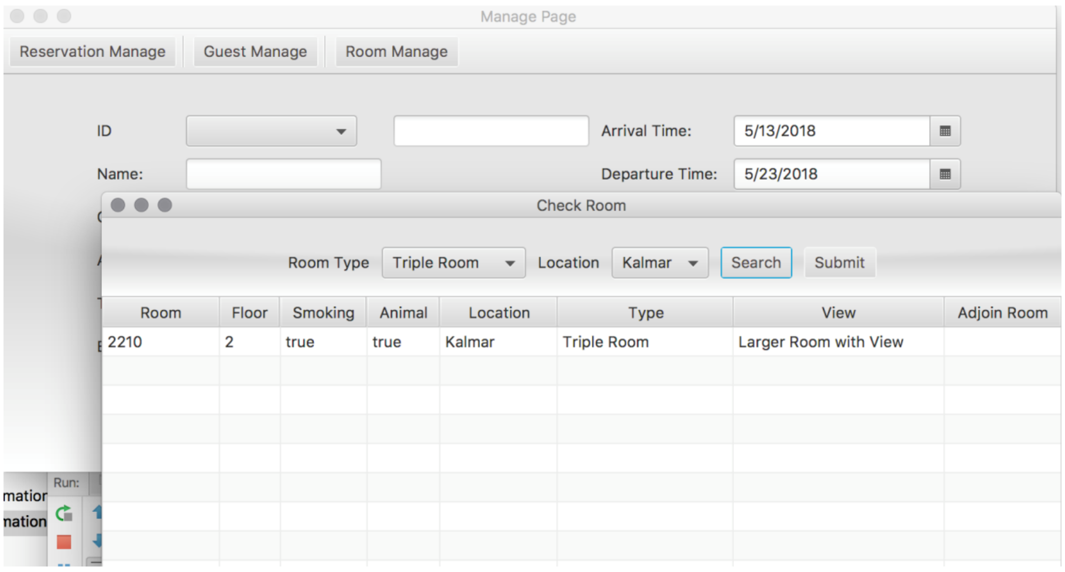
1.making a reservation.We record the personal information of the guests,room requirements, arrival time and departure time etc. To meet our requirements.Here you can choose different IDs. And enter the price of the room.


Manage Page

Reservation Manage | Guest Manage | Room Manage

ID	Passport	XXXX	Arrival Time:	5/13/2018
Name:	XXXX		Departure Time:	5/30/2018
Company:	XXXX		Room:	1111
Address:	XXXX		Days:	17
TEL:	1234		Card:	XXXX XXXX XXXX XXXX
Email:	ABC@GAMIL.COM		Price	1000

The space in the room may or may not be filled in, because here we consider that if the guest has a special request for the room, it needs the staff's own consideration. If you do not fill in, directly check, pop up the type of room you want to choose.


Manage Page

Reservation Manage | Guest Manage | Room Manage

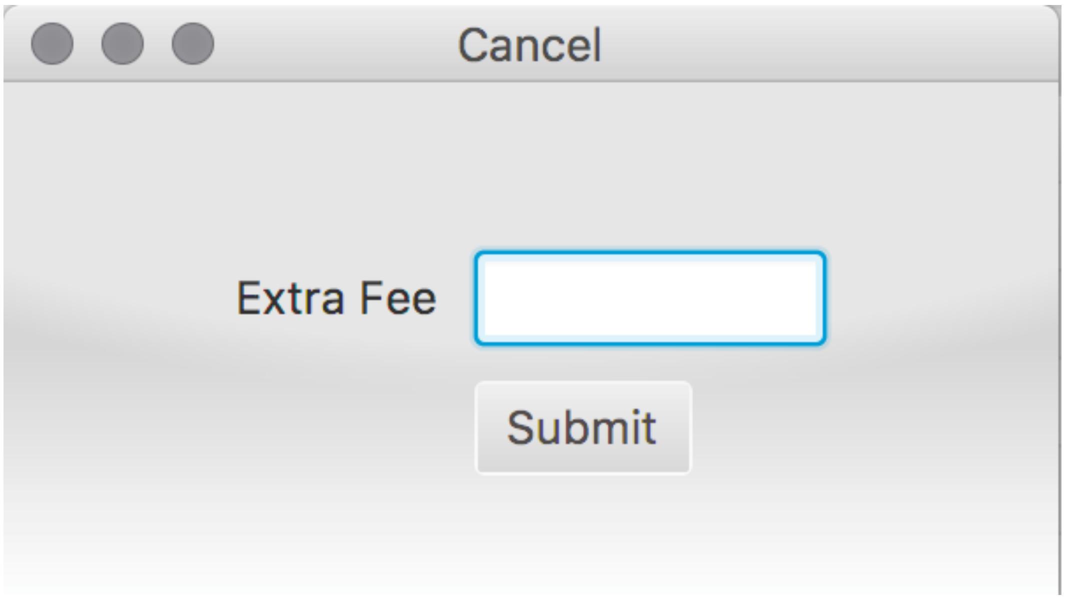
ID		Arrival Time:	5/13/2018
Name:		Departure Time:	5/23/2018

Check Room

Room Type: Triple Room — Location: Kalmar

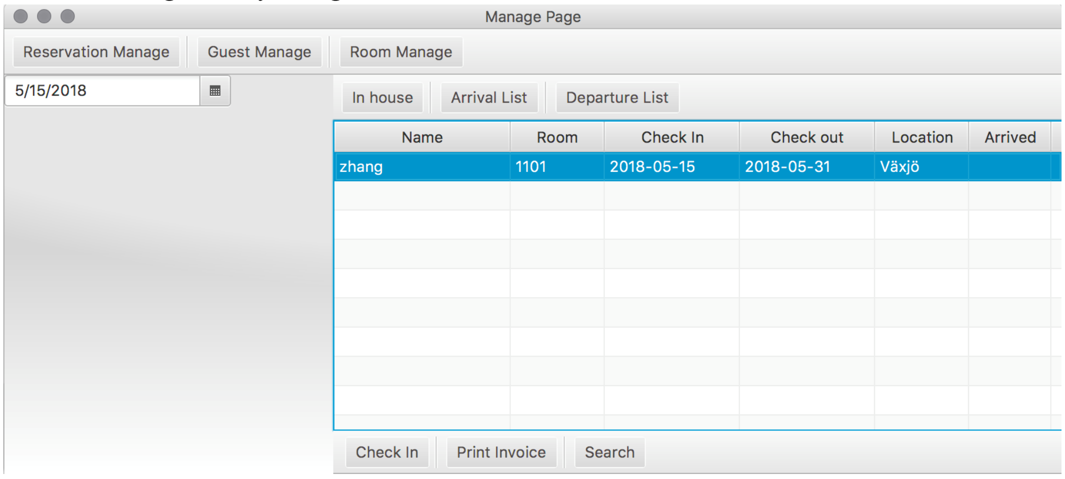
Room	Floor	Smoking	Animal	Location	Type	View	Adjoin Room
2210	2	true	true	Kalmar	Triple Room	Larger Room with View	

After you fill in the information, you can check the type of room and the location of the hotel. By selecting these options, you can see the type of room, quality(e.g. you can see the larger room with View and Adjoin room).And this side shows whether it can bring animals. This is one of the special requirements. It depends on whether managers meet customer requirements.When you have successfully confirmed all the information, the booking will be successful.



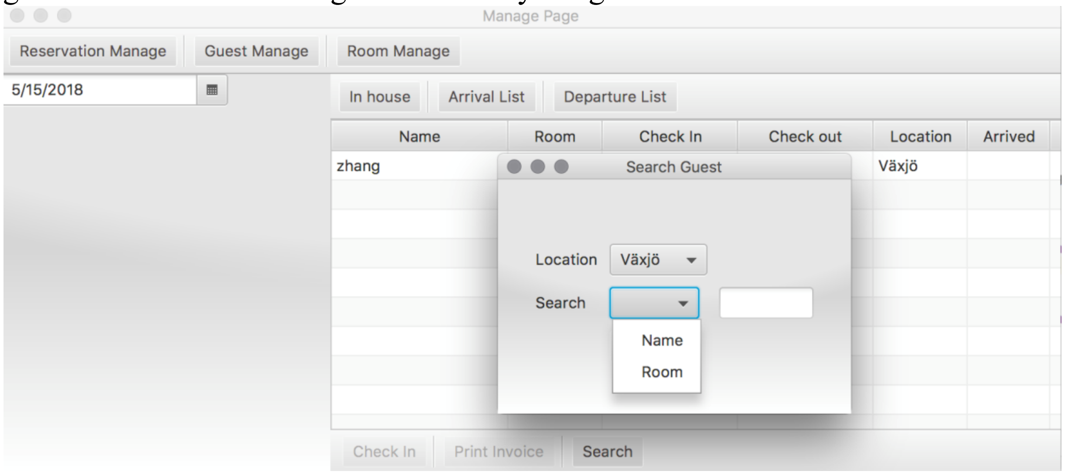
If the customer cancels after the reservation or does not arrive within the specified time, the hotel will charge part of the extra fee. This fee is filled in by management personnel.

2. check-in. When the guest wants to check-in, we must first confirm the time. If the list of guests is not too much, we can help him check-in directly based on the information given by the guests.



Name	Room	Check In	Check out	Location	Arrived
zhang	1101	2018-05-15	2018-05-31	Växjö	

If the number of people scheduled is too large to directly find the customer, then we will use the search method to find the guest's room number after checking the guest's name. Or find the guest's name by the guest's room number.



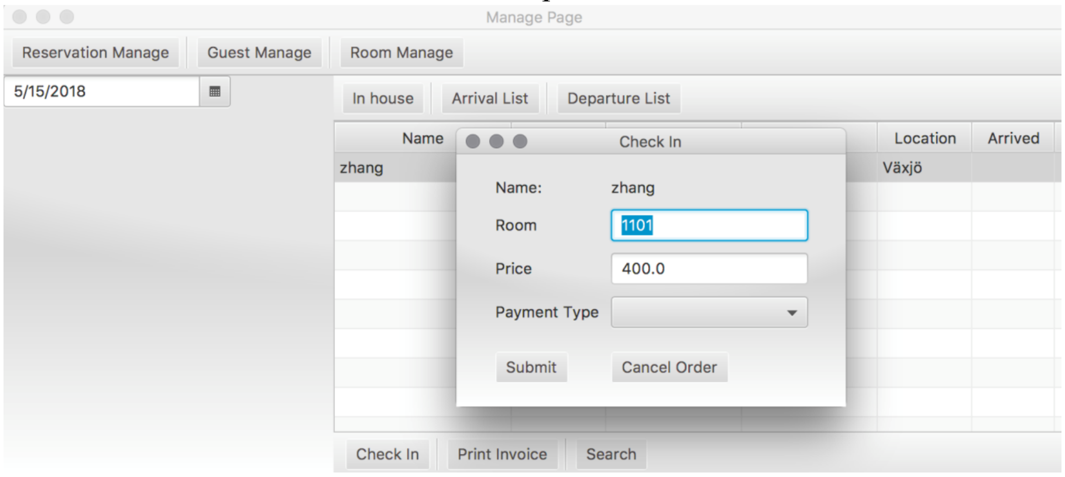
5/15/2018

Name	Room	Check In	Check out	Location	Arrived
zhang				Växjö	

Location: Växjö

Search:

Need to understand that check-in must be paid in advance.



5/15/2018

Name	Location	Arrived
zhang	Växjö	

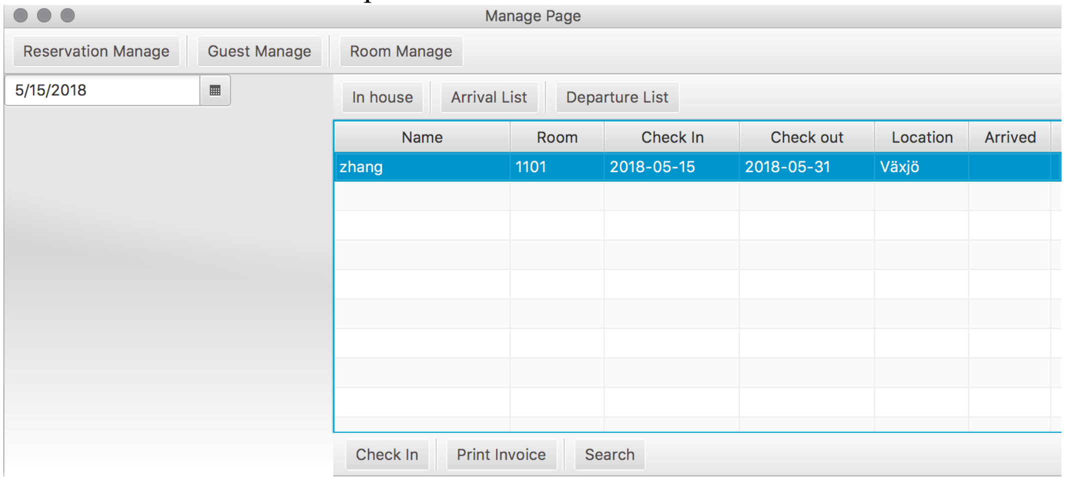
Name: zhang

Room: 1101

Price: 400.0

Payment Type:

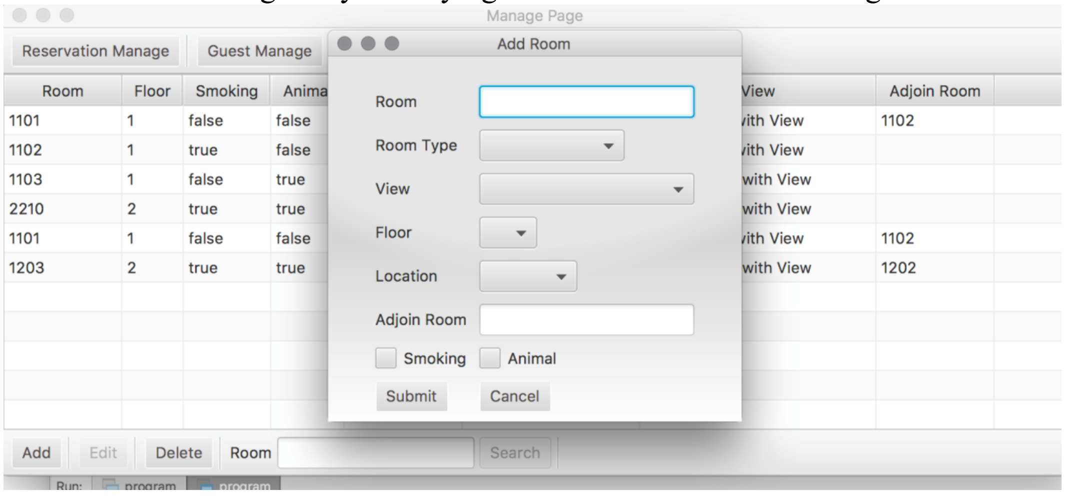
3. check-out. As with check-in, look for the time first, then select the information for selecting guests in the list. Finally click on the "Print Invoice" button to help the customer check-out and print the bill.



5/15/2018

Name	Room	Check In	Check out	Location	Arrived
zhang	1101	2018-05-15	2018-05-31	Växjö	

4. edit rooms.This one way is to meet the management staff to manage the room.You can manage it by modifying room information and adding rooms.


Manage Page

Reservation Manage | Guest Manage

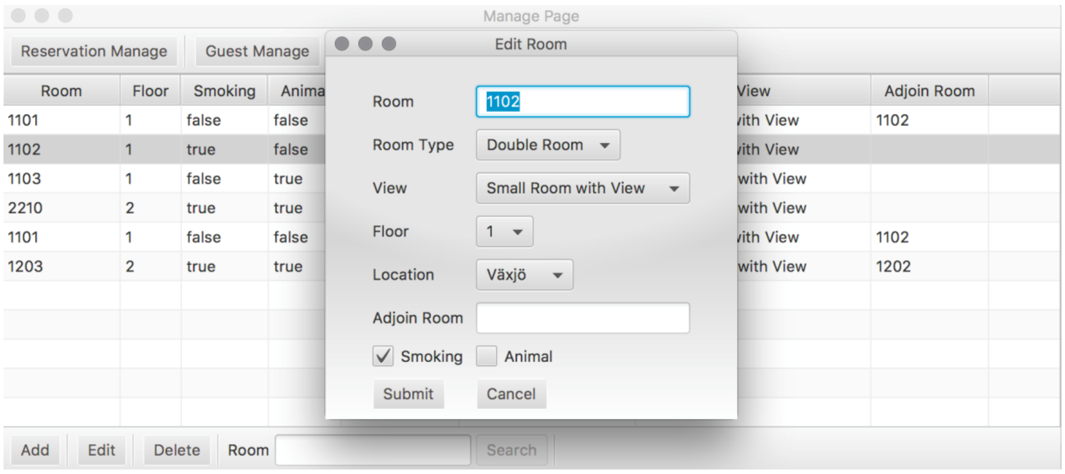
Room	Floor	Smoking	Anima	View	Adjoin Room
1101	1	false	false	ith View	1102
1102	1	true	false	ith View	
1103	1	false	true	with View	
2210	2	true	true	with View	
1101	1	false	false	ith View	1102
1203	2	true	true	with View	1202

Add Room

Room	
Room Type	
View	
Floor	
Location	
Adjoin Room	
☐ Smoking	☐ Animal

Submit | Cancel

Add | Edit | Delete | Room | Search


Manage Page

Reservation Manage | Guest Manage

Room	Floor	Smoking	Anima	View	Adjoin Room
1101	1	false	false	ith View	1102
1102	1	true	false	ith View	
1103	1	false	true	with View	
2210	2	true	true	with View	
1101	1	false	false	ith View	1102
1203	2	true	true	with View	1202

Edit Room

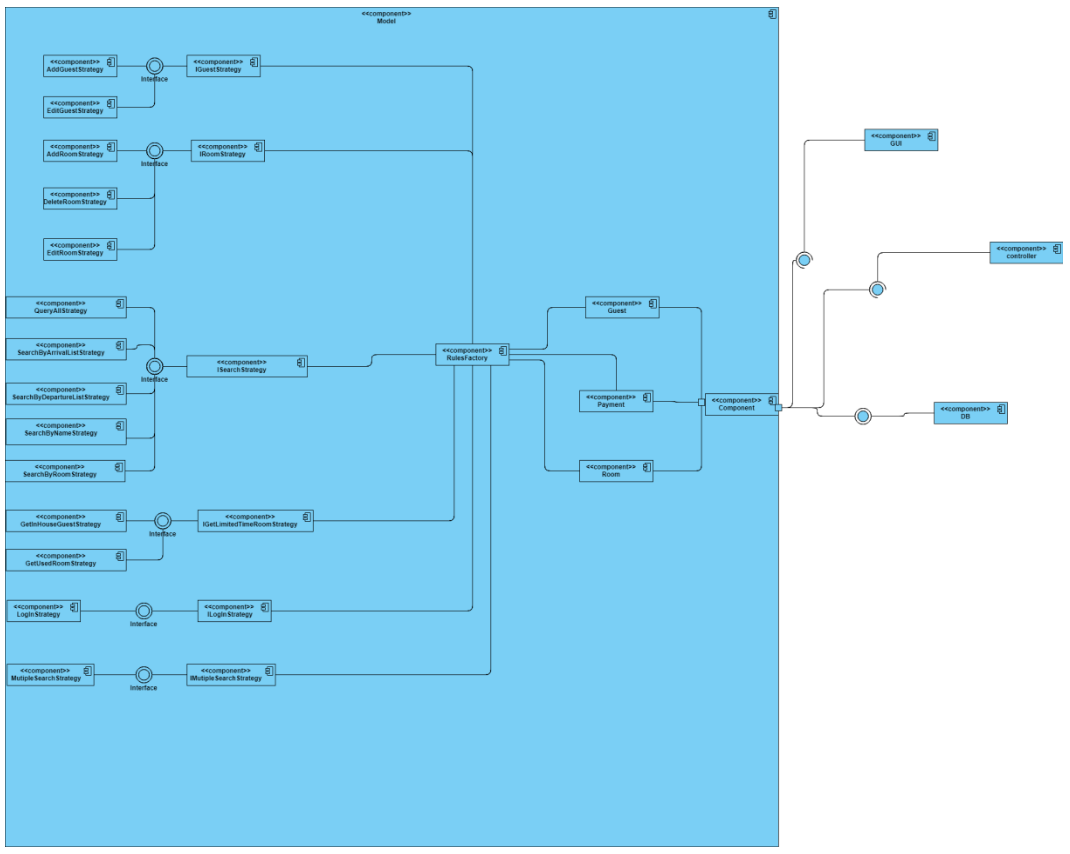
Room	1102
Room Type	Double Room
View	Small Room with View
Floor	1
Location	Växjö
Adjoin Room	
✓ Smoking	☐ Animal

Submit | Cancel

Add | Edit | Delete | Room | Search

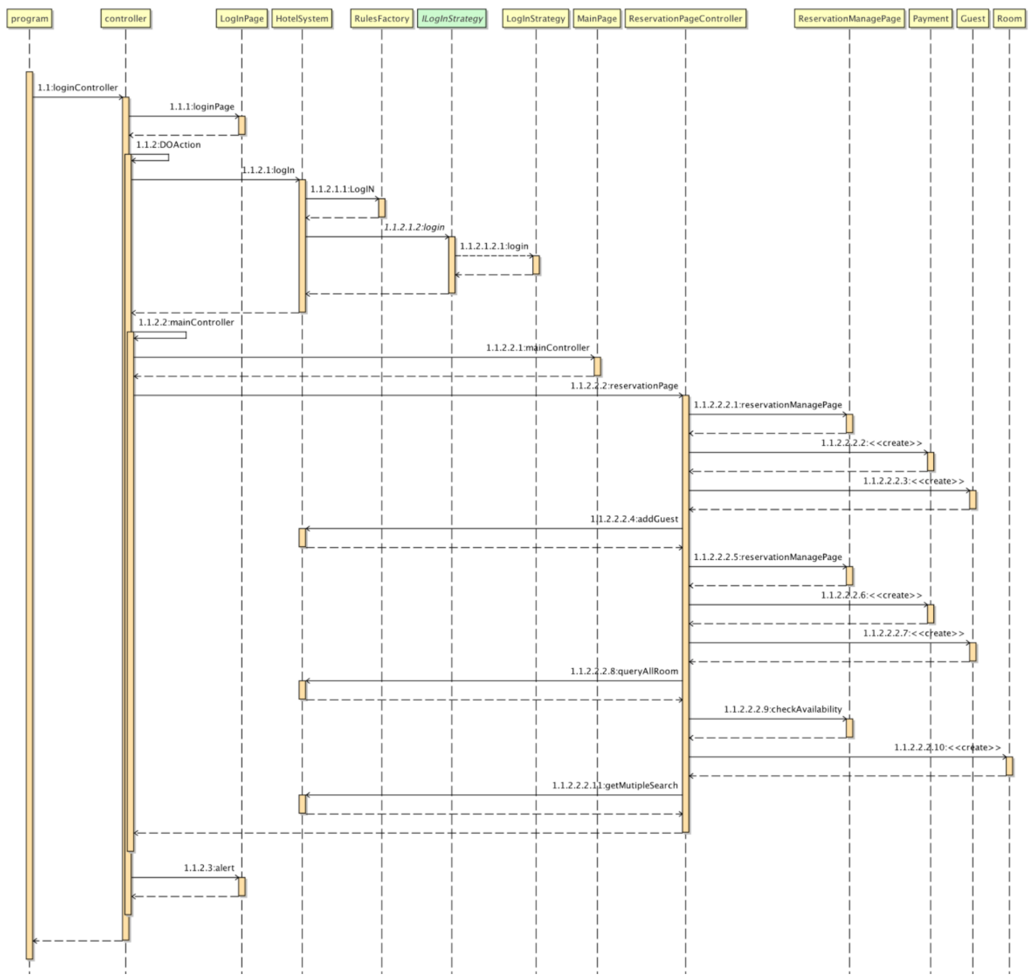
5.2.2 The UML Component Diagram of the architecture

Because our diagram is not only very large but also complicated. So we put all the diagrams in the Diagrams folder.

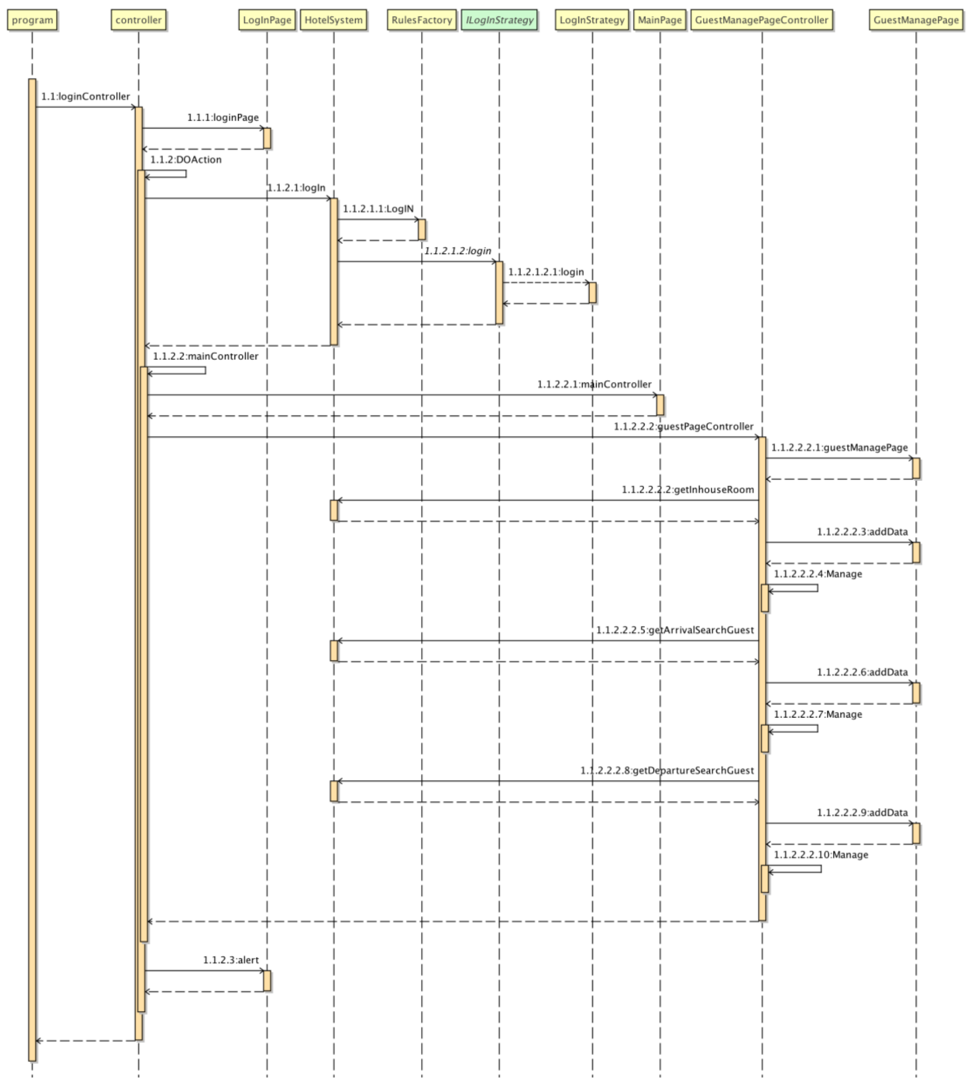

<<component>> Model
<<component>> AddGuestStrategy
<<component>> EditGuestStrategy
Interface
<<component>> IGuestStrategy
<<component>> AddRoomStrategy
<<component>> DeleteRoomStrategy
<<component>> EditRoomStrategy
Interface
<<component>> IRoomStrategy
<<component>> QueryAllStrategy
<<component>> SearchByArrivalListStrategy
<<component>> SearchByDepartureListStrategy
<<component>> SearchByNameStrategy
<<component>> SearchByRoomStrategy
Interface
<<component>> ISearchStrategy
<<component>> GetInHouseGuestStrategy
<<component>> GetUsedRoomStrategy
Interface
<<component>> IGetLimitedTimeRoomStrategy
<<component>> LogInStrategy
Interface
<<component>> ILogInStrategy
<<component>> MutipleSearchStrategy
Interface
<<component>> IMutipleSearchStrategy
<<component>> RulesFactory
<<component>> Guest
<<component>> Payment
<<component>> Room
<<component>> Component
<<component>> GUI
<<component>> controller
<<component>> DB

5.2.3 Sequence diagram for four major function

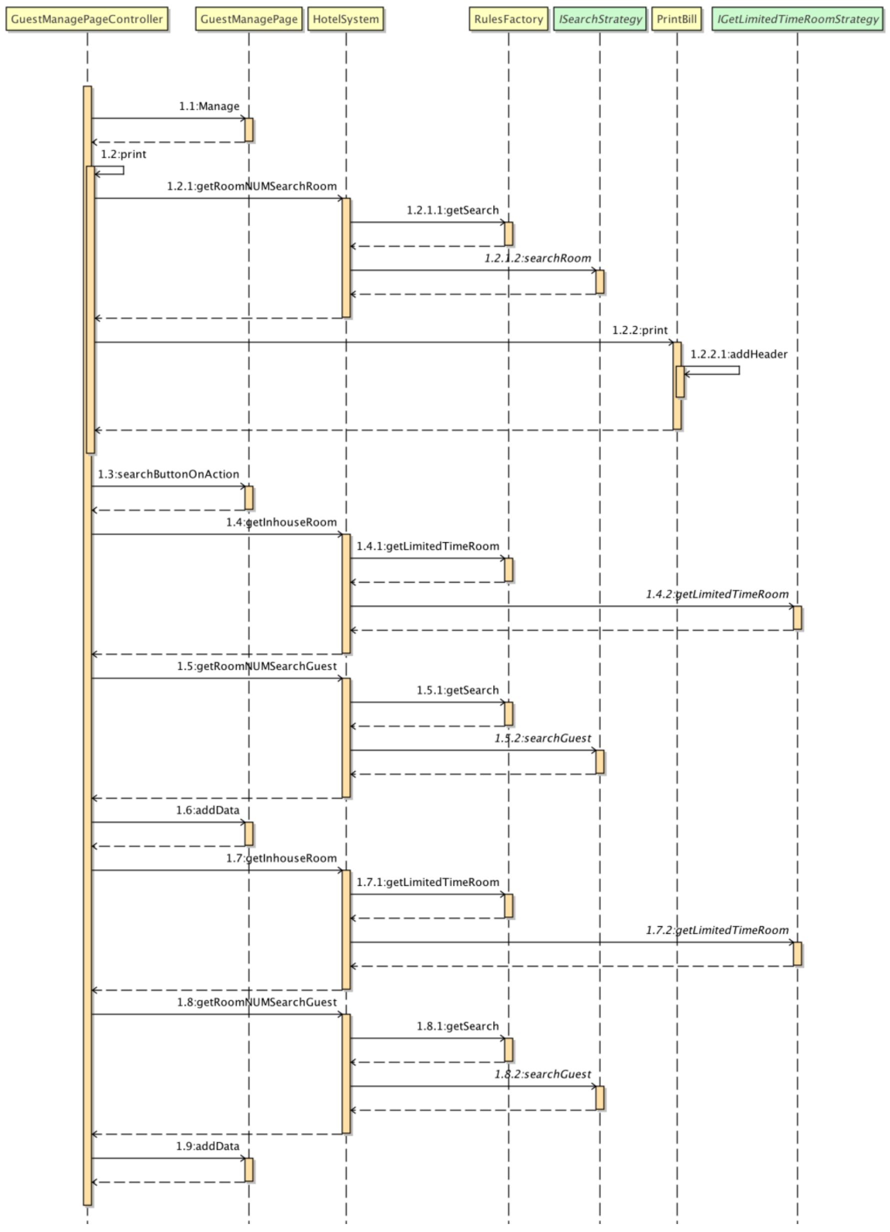
Because our diagram is not only very large but also complicated. So we put all the diagrams in the Diagrams folder.



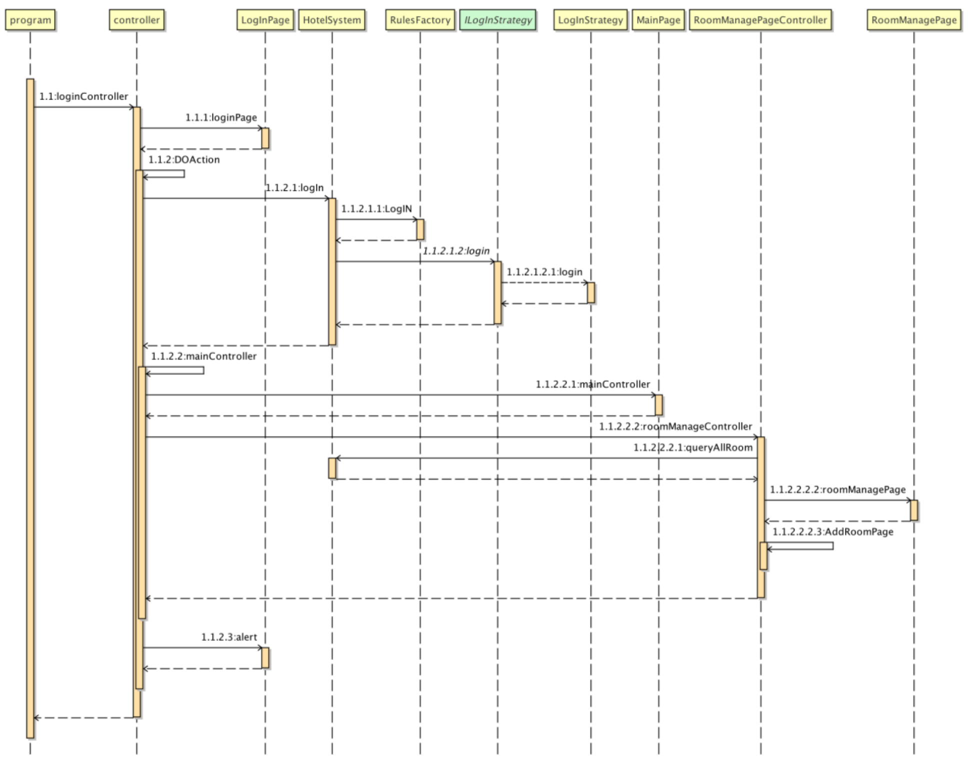
Reservation Sequence Diagram



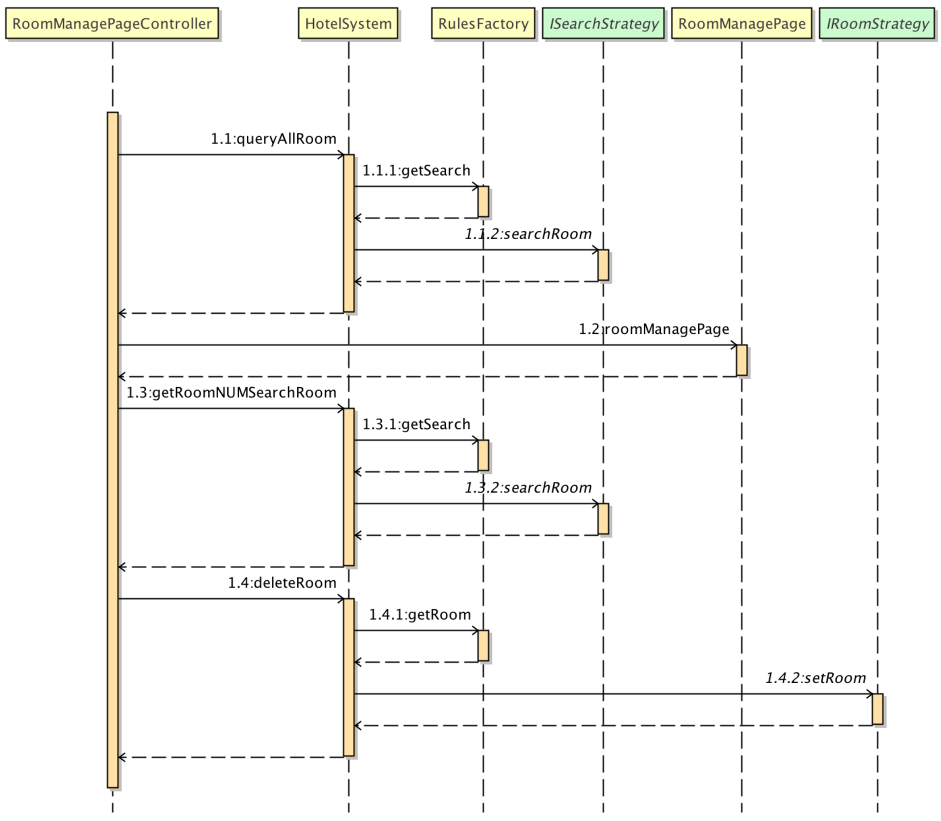
Check-in Sequence Diagram



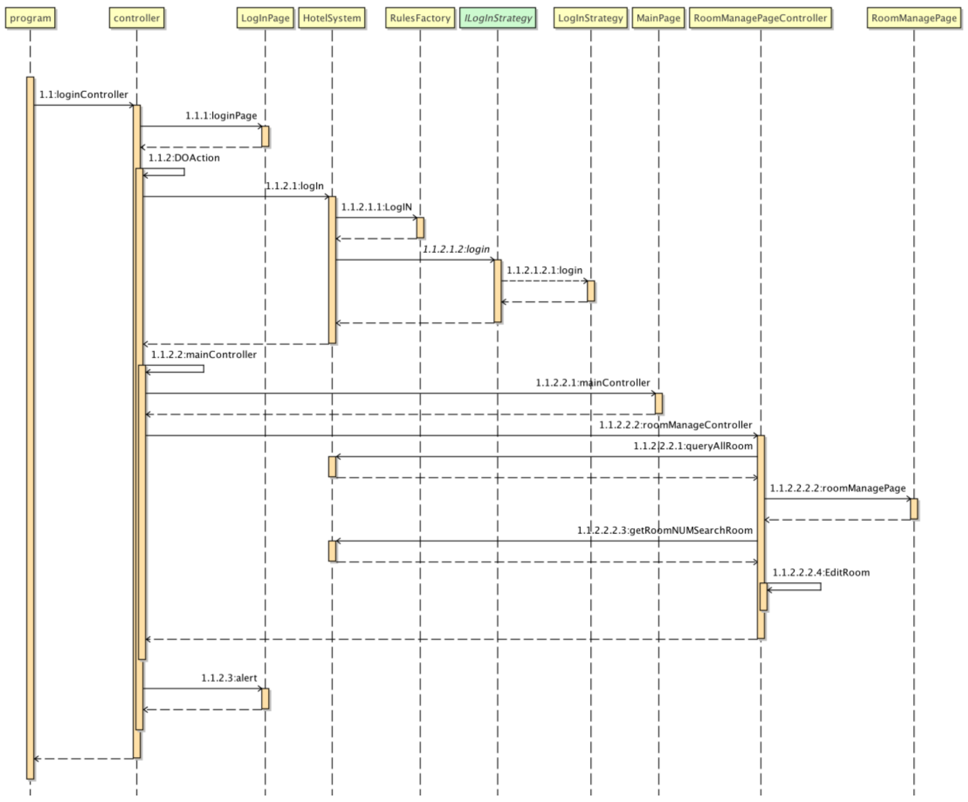
Check-out Sequence Diagram



Add Room Sequence Diagram



Delete Room Sequence Diagram



Edit Room Sequence Diagram

5.3 For each component identified in Software Architecture

5.3.1 Design Principles addressed

Object-oriented (OO): It is based on the concept of object, object-oriented, class and inheritance as the construction mechanism, making full use of interface and polymorphism to provide flexibility, understanding, understanding and characterization of the objective world and designing and constructing corresponding software system.
Object-oriented has four characteristics: abstraction, encapsulation, inheritance, polymorphism.

In order to meet these characteristics, we use the factory method pattern.

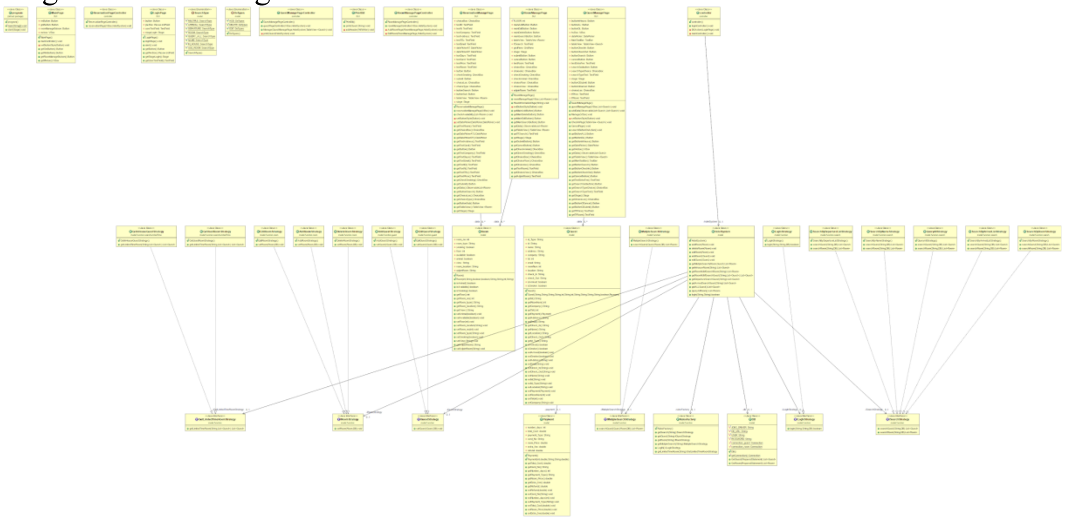
Factory method mode is also called factory mode, also called Virtual Constructor mode or Polymorphic Factory mode. It belongs to class creation mode. In the factory method mode, the factory parent is responsible for defining the public interface for creating the product object, and the factory subclass is responsible for generating the specific product object. The purpose of this is to delay the instantiation of the product class to the factory subclass. That is, through the factory subclass to determine which specific product class should be instantiated.

In the factory method pattern, the factory method is used to create the product that the customer needs, and at the same time hides the details of which specific

product class will be instantiated to the customer. The user only needs to care about the factory corresponding to the desired product and does not need to care about creating. Details do not even need to know the class name of a specific product class.
The polymorphism design based on factory roles and product roles is the key to the factory method model. It enables the factory to determine which product object to create, and the details of how to create this object are completely encapsulated within the specific factory. The factory method pattern is also called the polymorphic factory pattern because all concrete factory classes have the same abstract parent class.
Another advantage of using the factory method pattern is that when a new product is added to the system, there is no need to modify the interface provided by the abstract factory and the abstract product. There is no need to modify the client, and it is not necessary to modify other specific factories and specific products. Instead, only add a specific factory. And specific products on it. In this way, the scalability of the system has become very good, in full compliance with the "open and close principle."

5.3.2 The UML Class Diagram

Because our diagram is not only very large but also complicated. So we put all the diagrams in the Diagrams folder.



TimeLog

Activities	Estimate	Use time	End Date
Design Document	3 hours	1 hours 30 mins	2018/5/12

Purpose	half hours	20mins	2018/5/12
General priorities	45 mins	50 mins	2018/5/12
Outline of the design	1 hour	45 mins	2018/6/3
Major design issues	50 mins	30 mins	2018/6/4
Design details	18 hours	15 hours	2018/5/13
Reference Architectural Pattern	1 hour	1 hour	2018/5/11
The set of components	4 hours	2 hours 30 mins	2018/5/11
Discuss both requires and provides interface	2 hours	2 hours	2018/5/29
The UML Component Diagram of the architecture	2 hours	5 hours	2018/5/12
Sequence diagram for four major function	2 hours	30 mins	2018/5/13
For each component identified in Software Architecture	3 hours	2 hours	2018/5/13
Design Principles addressed	1 hours	1 hours	2018/5/13
The UML Class Diagram	2 hours	1 hours	2018/5/13
TimeLog	10 mins	10 mins	20185/13