THE POSTS AND TELECOMMUNICATIONS INSTITUTE OF TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY 1

INTRODUCTION TO SOFTWARE ENGINEERING

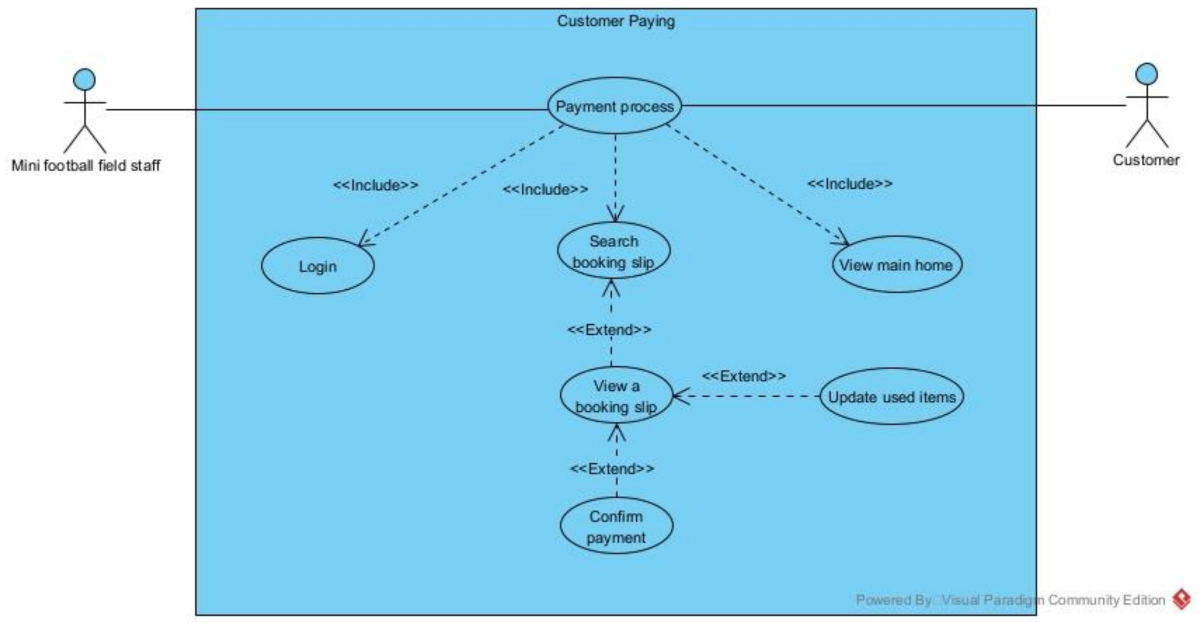
Class	**:**	**E22CQCN03-B**
Course group	**:**	**Group 6**
Group topic	**:**	**Mini football field management**
Project group	**:**	**Group 12**
Group members	**:**	trần minh hiếu – B22DCCN320 **Nguyễn Việt Hoàng – B22DCVT214** nguyễn tuấn minh – B22DCKH077 bùi minh tùng – B22DCDT293

MODULE: CUSTOMER PAYING

REPORT: SUMMARY REPORT

Hà Nội - 2025

1. Detail Use Case diagram for module 3: Customer paying



- Description:
  - Login: This use case let the mini football field staff log in to the system
  - View main home page: This use case let the mini football field staff choose the function that the staff can access
  - Payment process: This use case let the mini football field staff use the payment process function
  - Search booking slip: This use case let the mini football field staff search the booking slip by the current customer information
  - View a booking slip: This use case let the staff view the correct booking slip with the provided customer’s information
  - Update used items: This use case let the mini football field staff change or update the detailed list of the used items in case the customer complains
  - Confirm payment: This use case let the mini football field staff confirm that the payment for the booking slip has been made

2. Standard scenario of the module

Scenario	Update Used Items of the rental session
Actor(s)	Receptionist, Customer
Pre-condition	Receptionist has a receptionist account
Post-condition	- The payment for the session is saved in the database - The detailed list of the used items of the session is updated if the customer has a complain
Main-events	1. The receptionist A open the app to process the payment for the rental session of the customer B 2. The system display a login interface with a field to enter username, a field to enter password and a Login button 3. The receptionist enter username = “a”, password = “a” and click Login 4. The system display the main interface with the option to process payment 5. The receptionist click on the option to process payment 6. The system displays the search for customer with the input of customer name 7. The receptionist ask the customer about their name 8. The customer answer that their name is An 9. The receptionist enter the name “an” + click search 10. The system show the result interface

Search Customer [Go Back]

Customer name: [An] [Search]

Id	Name	Address	Phone	Email	Note
1	An	Huynh Thuc Khang	9172386245	an@gmail.com	
2	Anh	Lang Ha	1234685647	anh@gmail.com	

11. The receptionist ask the customer about their phone number

12. The customer answer with 9172386245

13. The receptionist click on the row of the correct customer

Booking Tickets for Customer: An [Go Back]

id	dateOfContract	depositAmount
1	30/03/2024	0
3	20/06/2024	0

14. The interface shows a list of booking ticket with that customer name

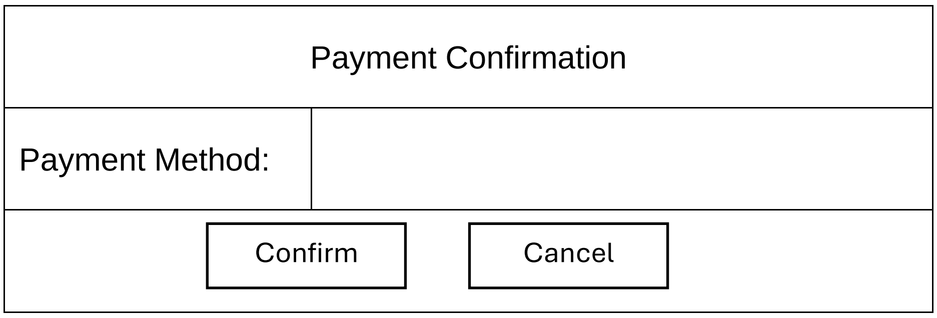
15. The receptionist ask the customer which booking slip they want to make payment for

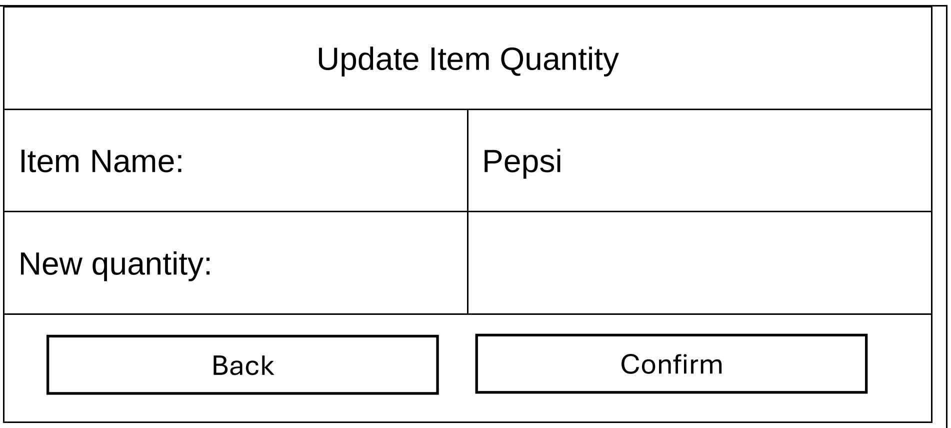
16. The customer answer the one with date of contract of 30/03/2024

17. The receptionist clicks the row of the booking ticket with the correct date of contract

18. The system shows the interface

Invoice		Confirm	Go Back
Customer Information			
Name:	An		
Phone:	9172386245		
Email:	an@gmail.com		
Field Cost			
Court	Court type	Sessions	Total Cost
1	single	13	6,500,000 VND
2	single	1	500,000VND
Items used			
Session	Item Name	Quantity	Price
1	Pepsi	10	100000
1	Revive	5	50000
2	Revive	5	50000
		Field Cost	7,000,000 VND
		Items Cost	200,000VND
		Total	7,200,000VND

	19. The customer confirm the information in the invoice, paid the money and the receptionist click the confirm button 20. The system display a confirm box, with a field to enter payment method, a confirm button and a cancel button  Payment Confirmation Payment Method: Confirm Cancel 21. The receptionist enter the payment method and click confirm button 22. The system display a success alert, confirming that the payment have been successfully save into the database and then returns to the main interface
Exception	4. The system display a “Login failed” alert and a Confirm button 4.1. The receptionist click the Confirm button 4.2. The system show the login interface 4.3. The receptionist enter the username and password correctly 4.4. The system display the main interface 19. The customer has a complain about this list of items used in the session 19.1. The receptionist click on an item that the customer complain about 19.2. The system show a pop-up interface with the name of the item, a field to enter that item quantity, and a Confirm button

	 Update Item Quantity Item Name: Pepsi New quantity: Back Confirm 19.3. The receptionist enter the number that item base on the customer’s complain and click Confirm 19.4. The system close the pop-up interface, and the invoice is update with the item’s new quantity 19.5. The receptionist then repeat step 19.1. to 19.4. for all the item that the customer complain about

3. Entity class diagram of analysis phase of the module

- Step 1: Describe the system in a paragraph or a scenario + exceptions

- Step 2+3: Extract and evaluate nouns in Step 1

+ Receptionist => **Need to be managed** => Class User: fullName, username, password, role

+ Customer => **Need to be managed** => Class Customer: name, address, phone, email, note

+ Account => Too generic => Eliminate

+ Session => **Need to be managed** => Class Session: receptionTime, returnTime, duration, status, note

+ Database => Too generic => Eliminate

+ Item => **Need to be managed** => Class Item: name, price, amount,

unit, category

+ Invoice => **Need to be managed** => Class Invoice: paymentDate,

paymentMethod, totalAmount

+Court => **Need to be managed** => Class Court: type, price

+ System => Too generic => Eliminate

+ Interface => Too generic => Eliminate

+ Button => Too generic => Eliminate

+ Rental period => Too generic => Eliminate

+ Booking ticket => **Need to be managed** => Class BookingTicket: dateOfContract, depositAmount, startHour, endHour

+ Alert => Too generic => Eliminate

=> Class extracted: User, Customer, Session, BookingTicket, Invoice, Item, Court

- Step 4: Consider quantities relationship among classes

+ 1 booking ticket has 1 customer, 1 customer can have 0 or many booking ticket => 1-n relationship

+ 1 session can have 0 or many items used, 1 item can be used in 0 or many session => n-n relationship => Create class UsedItem

+ 1 user can make 0 or many invoice, 1 invoice can be made by only 1 user => 1-n relationship

+ 1 booking ticket has 1 invoice, 1 invoice belongs to 1 booking ticket => 1-1 relationship

+ 1 booking ticket can be made by 1 user, 1 user can make many booking ticket => 1-n relationship

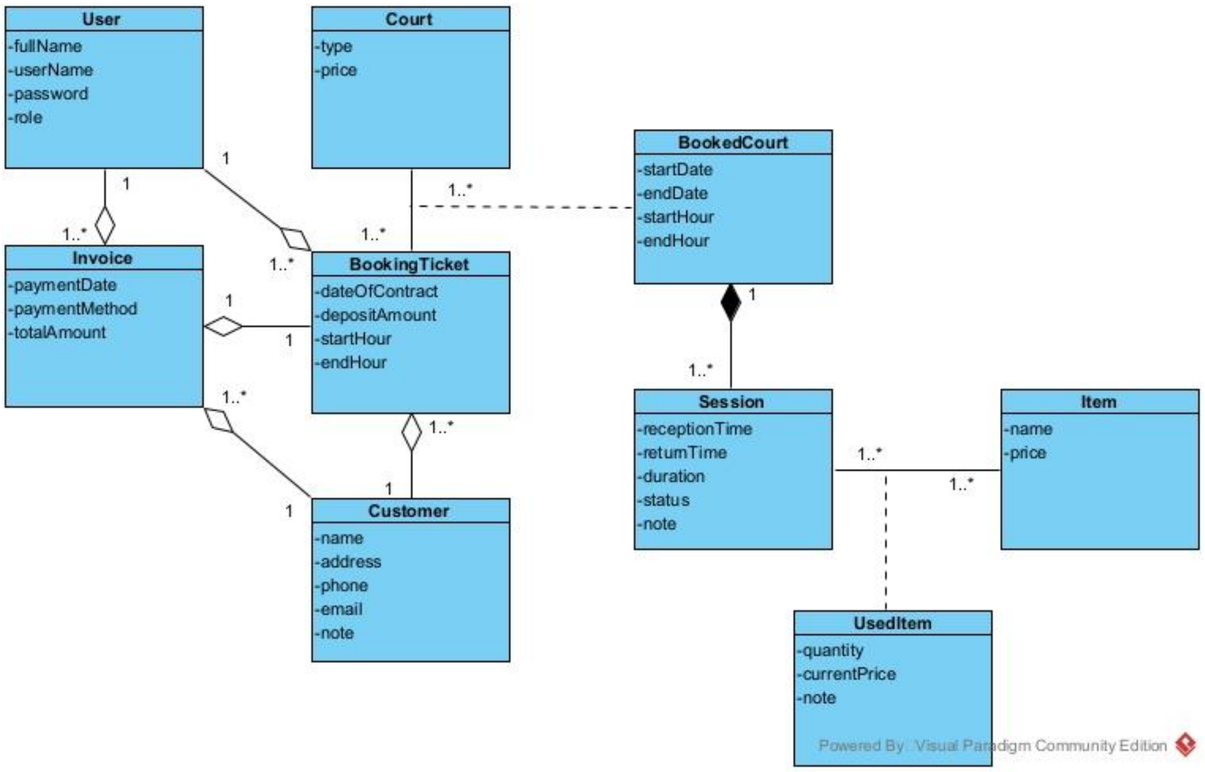
+ 1 booking ticket can book many courts, 1 court can be booked in many booking ticket => n-n relationship => Create class BookedCourt

+ 1 booked court can has many session, 1 session belongs to 1 booked court => 1-n relationship

+ 1 invoice can only have 1 customer, 1 customer can have many invoice => 1-n relationship

+ 1 BookedCourt can have many session, 1 session can only belong to 1 bookedCourt => 1-n relationship

- Step 5: Determine the object relationship among entity classes



4. Full class diagram of analysis phase of the module

- When the receptionist open the app, the system display a login interface => LoginView:

+ Input for username: inUsername

+ Input for password: inPassword

+ Login button: subLogin

- When click the login button, the system check username and password => a function:

+ Name: checkLogin()

+ Input: username, password

+ Output: boolean

+ Owner class: User

- The system displays the main interface => Class MainHomeView:

+ With at least an option Customer paying => subCustomerPaying

- Click on Customer Paying, the interface to search for customer by name appears => Class SearchCustomerView:

+ Input customer name: inName

+ Search button: subSearch

+ List of customer: outsubListCustomer

- Click on search button to search customer by name => a function:

+ Name: searchCustomer()

+ Input: Customer name

+ Output: List of customer

+ Owner class: Customer

- Click on a customer to show their booking ticket => a function:

+ Name: getBookingTickets()

+ Input: An object of customer

+ Output: List of booking ticket from a customer

+ Owner class: BookingTicket

- Click on a customer, the interface to show the list of booking ticket appears => Class BookingTicketView:

+ List of booking ticket: outsubListBookingTicket

- Click on 1 booking ticket, the interface with the invoice with full customer information and detailed list of items used in the session appear => Class InvoiceView :

+ Confirm the payment: subConfirm

+ List of used item that can be changed: outsubListUsedItem

+ Invoice with full customer information: outInvoice

- Click on 1 item, the interface to change item quantity appear => Class ChangeItemNumberView:

+ Input : inNumber

+ Confirm button: subConfirm

+ Cancel button: subCancel

- When click confirm to change item quantity, the system update the invoice in the system => a function:

+ Name: updateUsedQuantity()

+ Input: The number of an item, an object of item

+ Output: Void

+ Owner class: Session

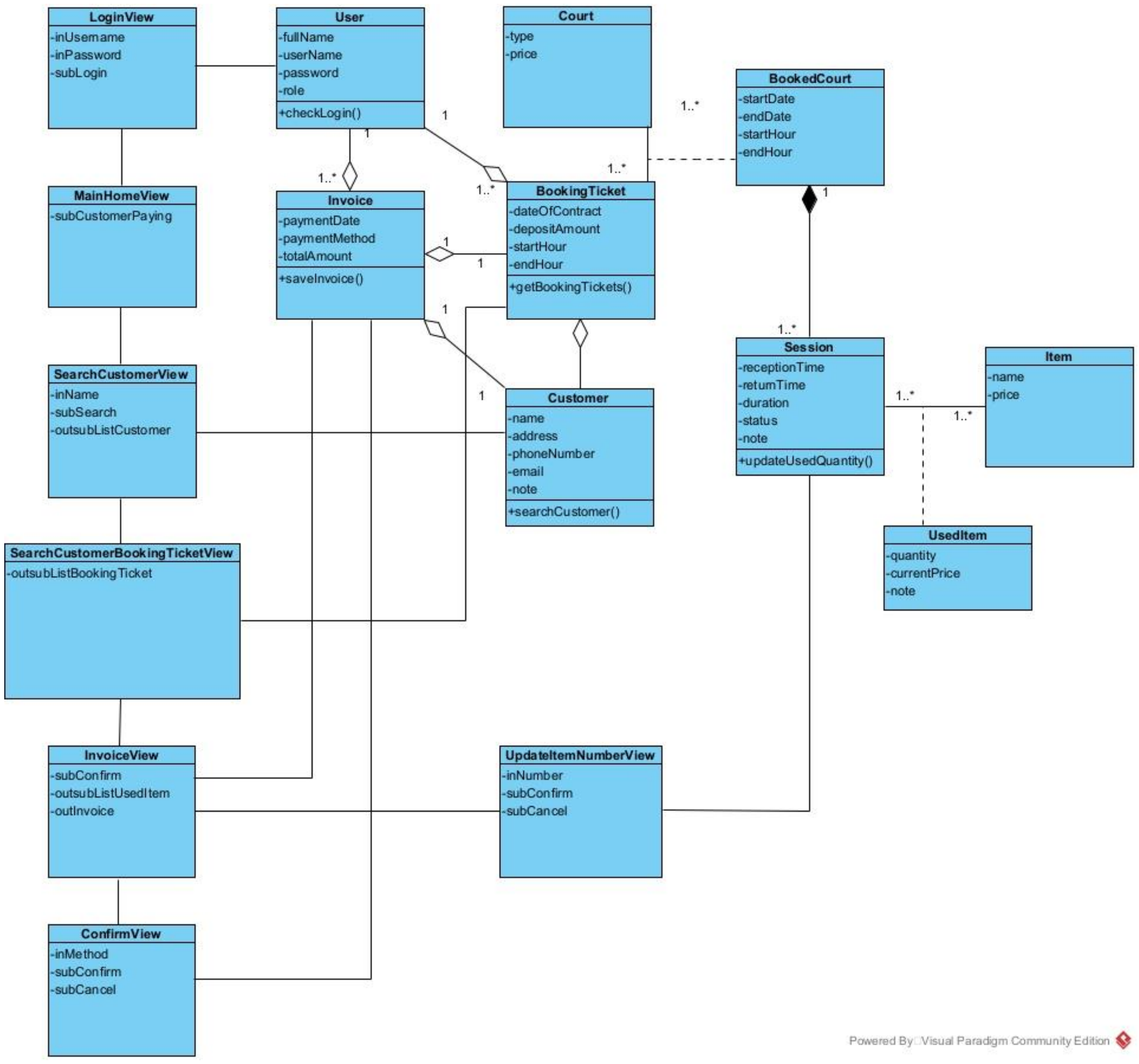
- Click on confirm to update the payment => a function:

+ Name: saveInvoice()

+ Input: an object of invoice, a string for payment method, an object of user

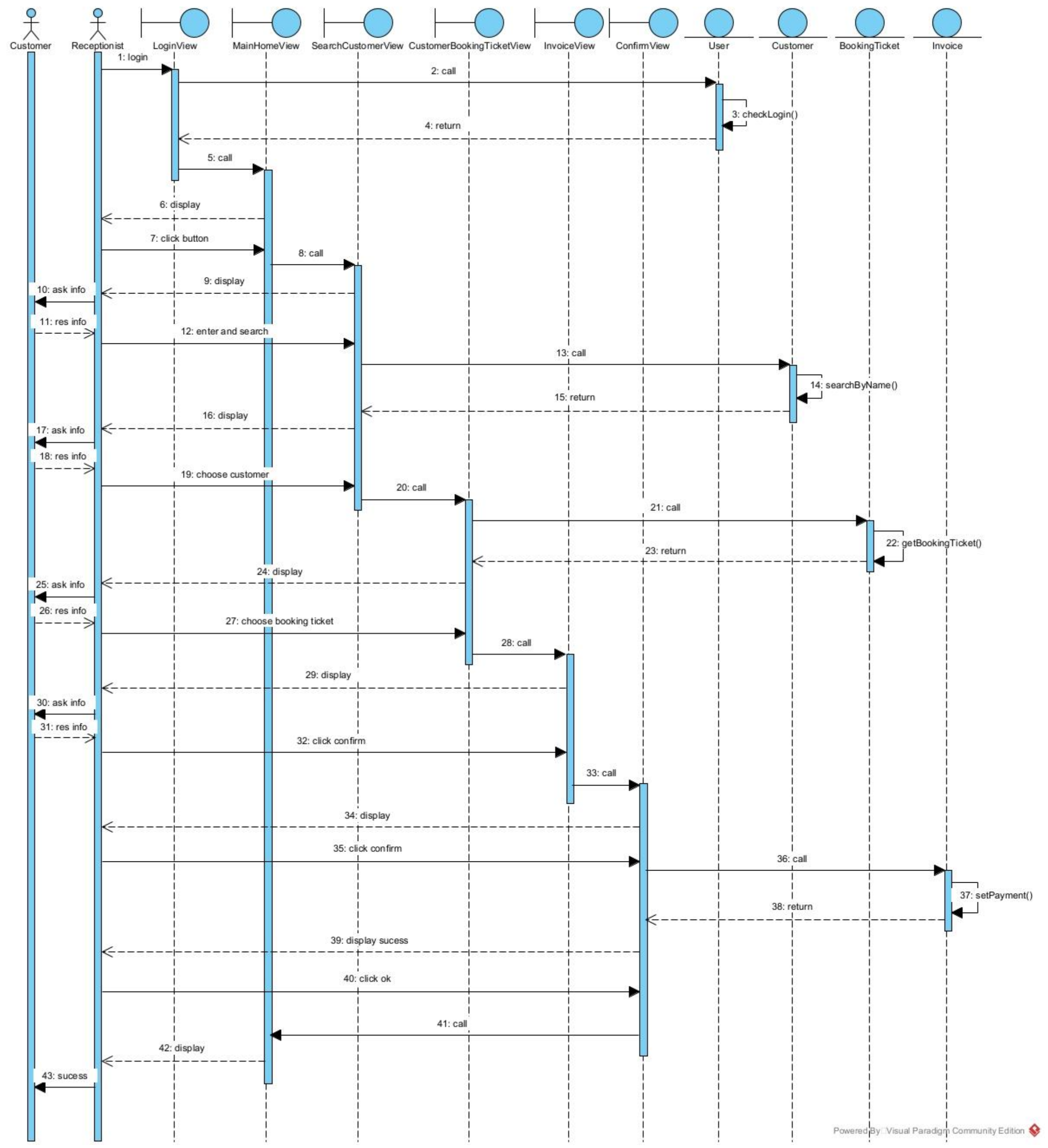
+ Output: boolean

+ Owner Class: Invoice



5. Sequence diagram of analysis phase of the module

1. Receptionist open the app, input username and password and click login in class LoginView
2. Class LoginView calls class User
3. Class User use checkLogin() to verify the log in. Log in success
4. Class User return the result to class LoginView
5. Class LoginView calls class MainHomeView
6. Class MainHomeView displays itself to the receptionist
7. The receptionist clicks subCustomerPaying button to use function “Customer paying”
8. Class MainHomeView calls class SearchCustomerView
9. Class SearchCustomerView displays itself to the receptionist
10. The receptionist asks the customer for their name
11. The customer says their name
12. The receptionist enter the customer name and click search
13. The class SearchCustomerView calls class Customer
14. Class Customer use searchCustomer() to find the list of customer
15. Class Customer return the result to the class SearchCustomerView
16. Class SearchCustomerView displays the list of customer to the receptionist
17. The receptionist asks the customer for their phone number
18. The customer says their phone number
19. The receptionist finds and clicks on the customer with the correct information
20. The class SearchCustomerView calls the class BookingTicketView
21. The CustomerBookingTicketView call the class BookingTicket
22. The class BookingTicket use function getBookingTickets() to find the list of a customer booking tickets
23. The class BookingTicket return the result to the BookingTicketView class
24. The class CustomerBookingTicketView displays itself along with the list of booking tickets from a customer to the receptionist
25. The receptionist ask the customer which booking ticket they want to make payment for
26. The customer response with the period of the correct booking ticket
27. The receptionist clicks on the correct booking tickets from the list
28. The class CustomerBookingTicket calls the class InvoiceView
29. The class InvoiceView display itself along with the invoice
30. The receptionist then asks the customer to confirm the invoice and make payment
31. The customer confirms and pays for the invoice
32. The receptionist then clicks confirm
33. The class InvoiceView calls class ConfirmView
34. The class ConfirmView displays itself to the receptionist
35. The receptionist enter payment method and clicks Confirm
36. The class ConfirmView calls class Invoice
37. The class Invoice use function saveInvoice()
38. The class Invoice return to the class ConfirmView
39. The class ConfirmView displays a success message to the receptionist
40. The receptionist click “OK” in the message
41. The class ConfirmView call class MainHomeView
42. The class MainHomeView displays itself to the receptionist
43. The receptionist informs the successful payment to the customer



Exception:

3. The method checkLogin() return false:

3.1. Class User return the result to LoginView

3.2. LoginView show the alert “Wrong user name or password”

3.3. The receptionist enter user name and password correctly and click Login button

3.4. The MainHomeView is shown to the receptionist

30. The customer want to change an item quantity

30.1. The receptionist click on the item used in the session that the customer complain

30.2. UpdatedItemNumberView is shown to the receptionist

30.3. The receptionist enter the new number and click OK button

30.4. The InvoiceView is shown to the receptionist

6. Entity class design of the module

- Step 1: Add the id attribute for the classes which DO NOT inherit from other class Court, BookedCourt, BookingTicket, Customer, User, Session, UsedItem, Item, Invoice.

- Step 2: Add the type of each attribute in all classes:

- User:
  - fullName: String
  - username: String
  - password: String
  - role: String
- Invoice:
  - paymentDate: Date
  - paymentMethod: String
  - totalAmount: int
- BookedCourt:
  - startDate: Date
  - endDate: Date
  - startHour: Time
  - endHour: Time
- Court:
  - type: String
  - price: int
- BookingTicket:
  - dateOfContract: Date
  - depositAmount: int
- Customer:
  - name: String
  - address: String
  - phone: String
  - email: String
  - note: String
- Session:
  - receptionTime: Time
  - returnTime: Time
  - duration : Time

    - date: Date
- Item:
    - name: String
    - price: int
- UsedItem:
    - quantity: int
    - currentPrice: int

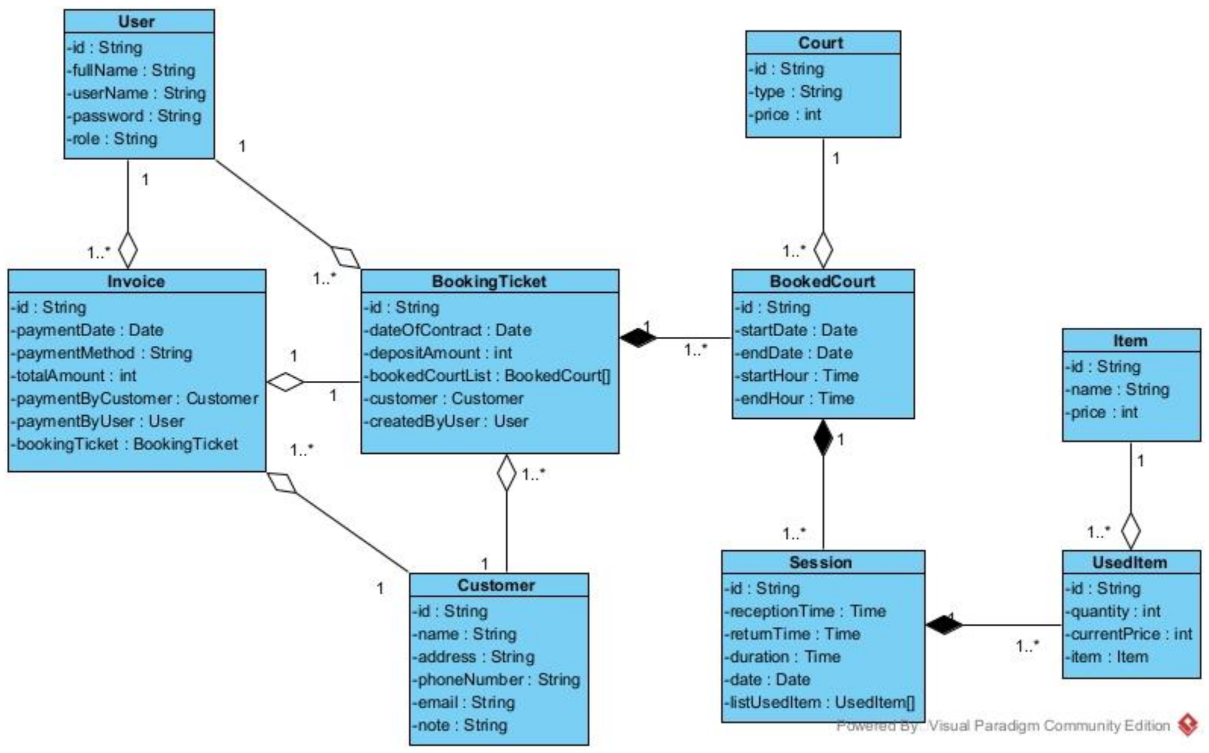
- Step 3: Convert all association relationships to correspond aggregation/ composition relationships:

BookingTicket + Court-> BookedCourt is converted to: Court is a component of BookedCourt, BookedCourt is a component of BookingTicket.

Session + Item-> UsedItem is converted to: Item is a component of UsedItem, UsedItem is a component of Session.

- Step 4: Add the object attributes which correspond to the aggregation/composition relationships:

- Court is a component of BookedCourt, of type 1-n -> BookedCourt has a Court
- BookedCourt is a component of BookingTicket, of type n-1 -> BookingTicket has a BookedCourt
- Customer is a component of BookingTicket, of type 1-n -> BookingTicket has a Customer
- Customer is a component of Invoice, of type 1-n -> Invoice has a Customer
- User is a component of BookingTicket, of type 1-n -> BookingTicket has an User
- User is a component of Invoice, of type 1-n -> Invoice has an User
- BookingTicket is a component of Invoice, of type 1-1 -> Invoice has an BookingTicket
- Item is a component of UsedItem, of type 1-n -> UsedItem has a Item
- UsedItem is a component of Session, of type n-1 -> Session has a list of UsedItem
- Session is a component of BookedCourt, of type n-1 -> BookedCourt has a list of Session



7. Database design of the module

- Step 1: For each entity class, create a corresponding table.

User -> tblUser

BookedCourt -> tblBookedCourt

Court -> tblCourt

Invoice -> tblInvoice

BookingTicket -> tblBookingTicket

Session -> tblSession

Item -> tblItem

Customer -> tblCustomer

UsedItem -> tblUsedItem

- Step 2: For each entity class, transfer all non-object attributes to contribute as the columns of the corresponding table.

- **tblUser:**
    - id: varchar(50)

    - fullName: varchar(50)
    - userName: varchar(50)
    - password: varchar(50)
    - note: varchar(50)
- **tblBookedCourt:**
    - id: varchar(50)
    - startDate: date
    - endDate: date
    - startHour: String
    - endHour: String
- **tblCourt:**
    - id: varchar(50)
    - type: varchar(50)
    - price: int
- **tblInvoice:**
    - id: varchar(50)
    - paymentDate: date
    - paymentMethod: varchar(50)
    - totalAmount: int
- **tblBookingTicket:**
    - id: varchar(50)
    - dateOfContract: date
    - depositAmount: int
- **tblSession:**
    - id: varchar(50)
    - receptionTime: varchar(50)
    - returnTime: varchar(50)
    - duration: varchar(50)
    - date: date
- **tblItem:**
    - id: varchar(50)
    - name: varchar(50)
    - price: int
- **tblCustomer:**
    - id: varchar(50)
    - name: varchar(50)
    - phoneNumber: varchar(50)
    - email: varchar(50)
    - note: varchar(50)
- **tblUsedItem:**
    - id: varchar(50)
    - quantity: int
    - currentPrice: int
    - note: varchar(50)

- Step 3: Consider the quantity relationships among entity classes. These relationships will be those among corresponding tables:

1 User – n BookingTicket

1 User – n Invoice

1 Customer – n BookingTicket

1 Customer – n Invoice

1 Invoice – 1 BookingTicket

1 Court – n BookedCourt

1 BookingTicket – n BookedCourt

1 BookingTicket – n Session

1 Session – n UsedItem

1 Item – n UsedItem

- Step 4: Configure the key column for tables.

- **tblUser:**
  - id: primary key
- **tblBookedCourt:**
  - id: primary key
  - The relationship between tblCourt and tblBookedCourt is 1-n -> tblBookedCourt will save the foreign key of CourtID references to tblCourt.
  - The relationship between tblBookingTicket and tblBookedCourt is 1-n -> tblBookedCourt will save the foreign key of BookingTicketID references to tblBookingTicket.
- **tblCourt:**
  - id: primary key
- **tblInvoice:**
  - id: primary key
  - The relationship between tblUser and tblInvoices is 1-n -> tblInvoices will save the foreign key of UserID references to tblUser.
  - The relationship between tblCustomer and tblInvoices is 1-n -> tblInvoices will save the foreign key of CustomerID references to tblCustomer.
  - The relationship between tblBookingTicket and tblInvoices is 1-1 -> tblInvoices will save the foreign key of BookingTicketID references to tblBookingTicket.
- **tblBookingTicket:**
  - id: primary key
  - The relationship between tblUser and tblBookingTicket is 1-n -> tblBookingTicket will save the foreign key of UserID references to tblUser.

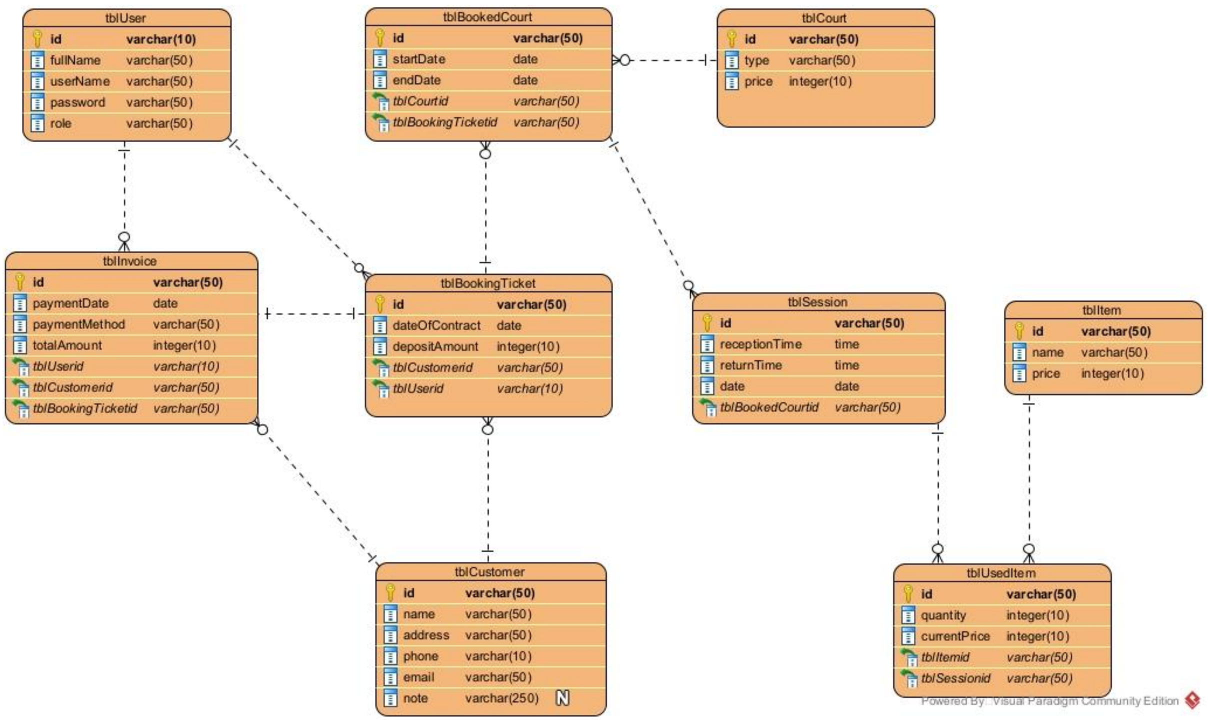
- The relationship between tblCustomer and tblBookingTicket is 1-n -> tblBookingTicket will save the foreign key of CustomerID references to tblCustomer.

- **tblSession:**
  - id: primary key
  - The relationship between tblBookedCourt and tblSession is 1-n -> tblSession will save the foreign key of BookedCourtId references to tblBookedCourt.
- **tblItem:**
  - id: primary key
- **tblCustomer:**
  - id: primary key
- **tblUsedItem:**
  - id: primary key
  - The relationship between tblItem and tblUsedItem is 1-n -> tblUsedItem will save the foreign key of ItemID references to tblItem.

The relationship between tblSession and tblUedItem is 1-n -> tblUsedItem will save the foreign key of ItemID references to tblItem.

- Step 5: Delete the derived attribute and abundant attribute

Abundant attribute: duration in in tblSession.



8. Interface design and full class diagram of the module

Interface design

In this module, the interface includes 7 main interface:

+ Login Page

+ MainHome Page

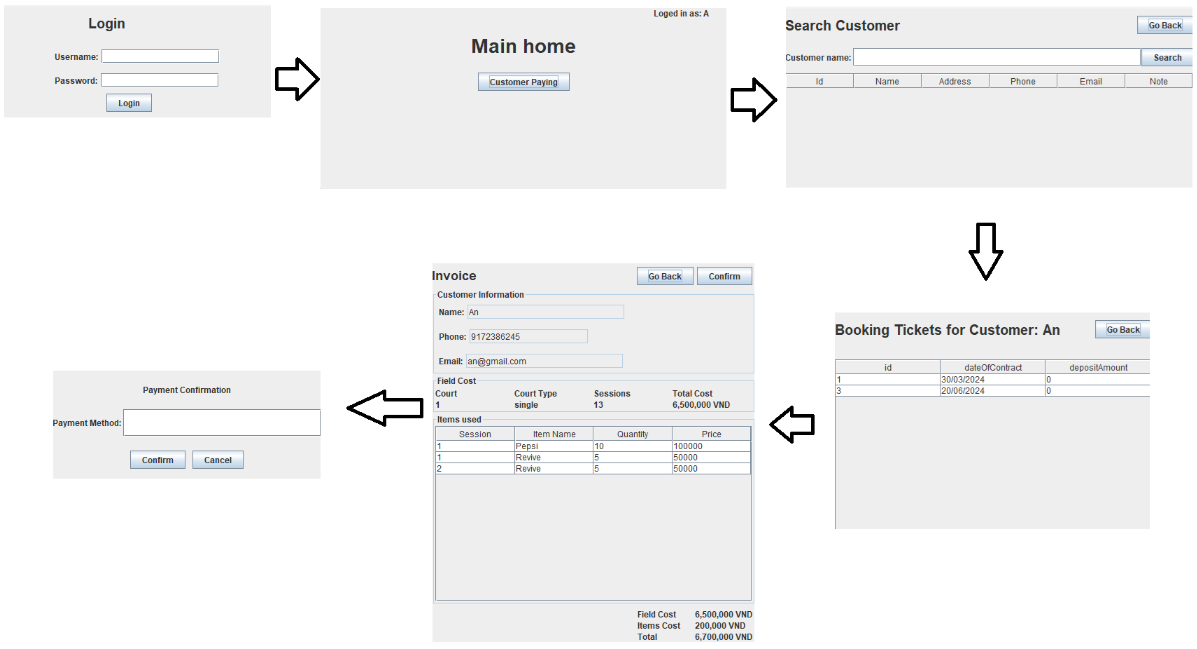
+ SearchCustomer Page

+ SearchBookingTicket Page

+ Invoice Page

+ Confirm Page.

+ UpdateItemNumber Page



id	dateOfContract	depositAmount
1	30/03/2024	0
3	20/06/2024	0

Customer Information

Name: An

Phone: 9172386245

Email: an@gmail.com

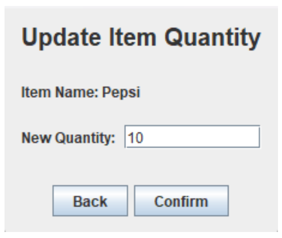
Field Cost

Court	Court Type	Sessions	Total Cost
1	single	13	6,500,000 VND

Items used

Session	Item Name	Quantity	Price
1	Pepsi	10	100000
1	Revive	5	50000
2	Revive	5	50000

Field Cost	6,500,000 VND
Items Cost	200,000 VND
Total	6,700,000 VND



Item Name: Pepsi

New Quantity: 10

Full class diagram

View class

- LoginFrm is the interface to log in. It needs a text field to enter the username, a text field to enter a password, and a button to log in.
- MainHomeFrm is the home interface of the employee. It needs at least a button to go to the receptionist payment function.

- SearchCustomerFrm is the interface of searching customer. It includes a search button function, a text field to enter customer name and a table displaying list of all customer.
- SearchBookingTicketFrm is the interface of searching booking ticket. It includes a table displaying all booking ticket of a customer.
- InvoiceFrm is the interface of displaying invoice. It includes a button to confirm, a button to go back and a table displaying the list of used item.
- ConfirmFrm is the interface of confirm payment. It includes a text field to enter the payment method, a button to confirm and a button to cancel.
- UpdateItemNumberFrm is the interface of updating item number. It needs the input text field to enter the number of the item, a button to confirm the update and a cancel button.

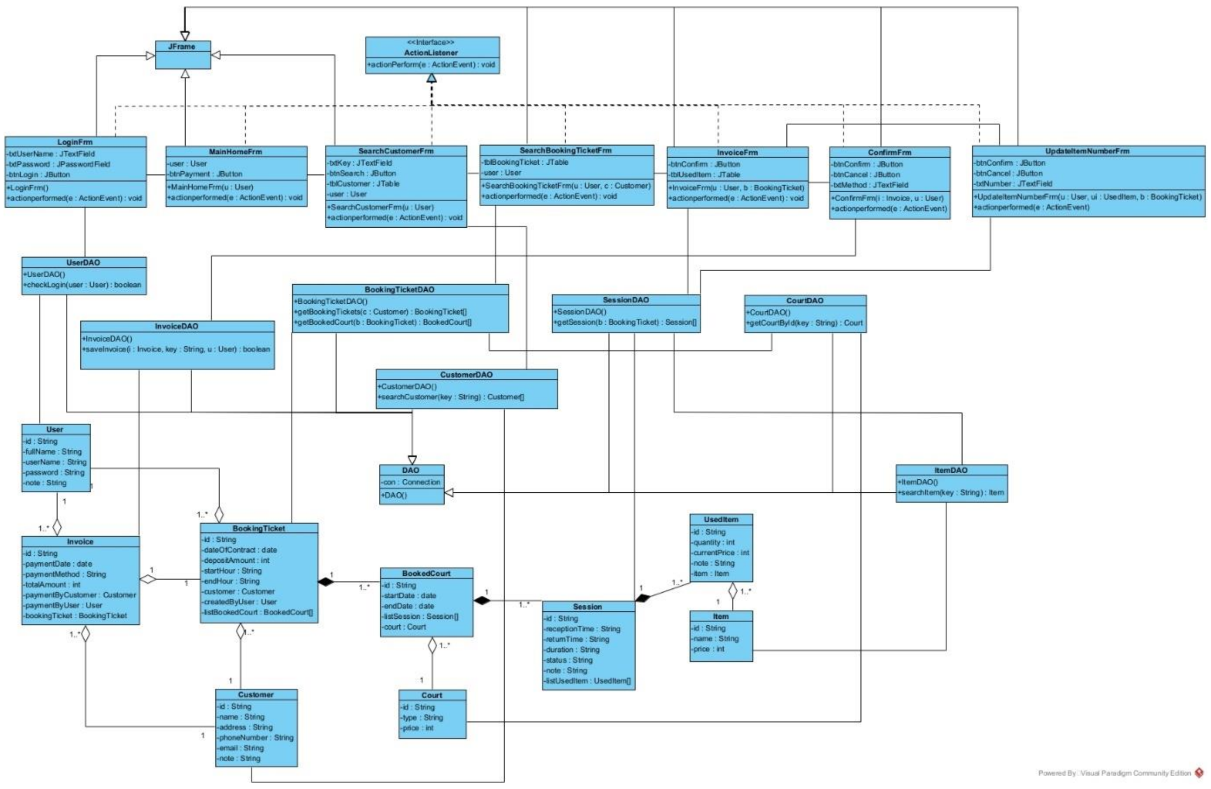
Control(DAO) class:

- DAO is a general class of DAO. It has only the construction to connect to the DB and provides the common connection for all inherited DAO classes in the system.
- UserDAO is the class for manipulating with DB related to the User object. In this module, it needs a method:
    - checkLogin(): to verify whether the login information is correct or not.
        - Input: User, ensure encapsulation
        - Output: Boolean, check if the login is valid, if not return false
- CustomerDAO is the class for manipulating with DB related to the Customer object. In this module, it’s need a method:
    - searchCustomer(): to search all customer whose name contains the keyword.
        - Input: String, the information is given by the customer for the receptionist to enter
        - Output: Customer[], there can be many customer with the same name
- BookingTicketDAO: is the class for manipulating with DB related to the booking ticket object. In this module, it’s need a method:
    - getBookingTickets(): to search all booking tickets which belong to a customer.
        - Input: Customer, ensure encapsulation

        - Output: BookingTicket[], one customer can have many booking tickets
    - getBookedCourt(): to to get all bookedCourt object belong to a BookingTicket object
        - Input: BookingTicket, ensure encapsulation
        - Output: BookedCourt[], 1 BookingTicket can have many BookedCourt
- InvoiceDAO: is the class for manipulating with DB related to the Invoices Object. In this module, it’s need 2 method:
    - saveInvoice(): to retrieve all information of the invoice
        - Input: Invoice, User, string for method, ensure encapsulation
        - Output: boolean, return true if the Invoice is saved successfully, otherwise false
- CourtDAO: is the class for manipulating with DB related to theCourt Object. In this module, it’s need a method:
    - getCourtById(): to retrieve information of a court
        - Input: String, the information is saved in tblBookedCourt
        - Output: Court, ensure encapsulation
- ItemDAO: is the class for manipulating with DB related to the Item object. In this module, it’s need a method:
    - searchItem(): to retrieve information of an item
        - Input: String, the information is saved in tblUsedItem.
        - OutPut: Item, ensure encapsulation
- SessionDAO: is the class for manipulating with DB related to the Session object. In this module, it’s need 2 method:
    - getSession(): to retrieve sessions of a BookedCourt object
        - Input: BookedCourt, ensure encapsulation
        - Output: Session[], a BookedCourt object can have many Session object
    - getUsedItem(): to retrieve all item used and quantity in a session
        - Input: Session, ensure encapsulation

- Output: UsedItem[], a session can have many used item

Entity classes: User, Invoices, BookingTicket, Session, Customer, Item, UsedItem

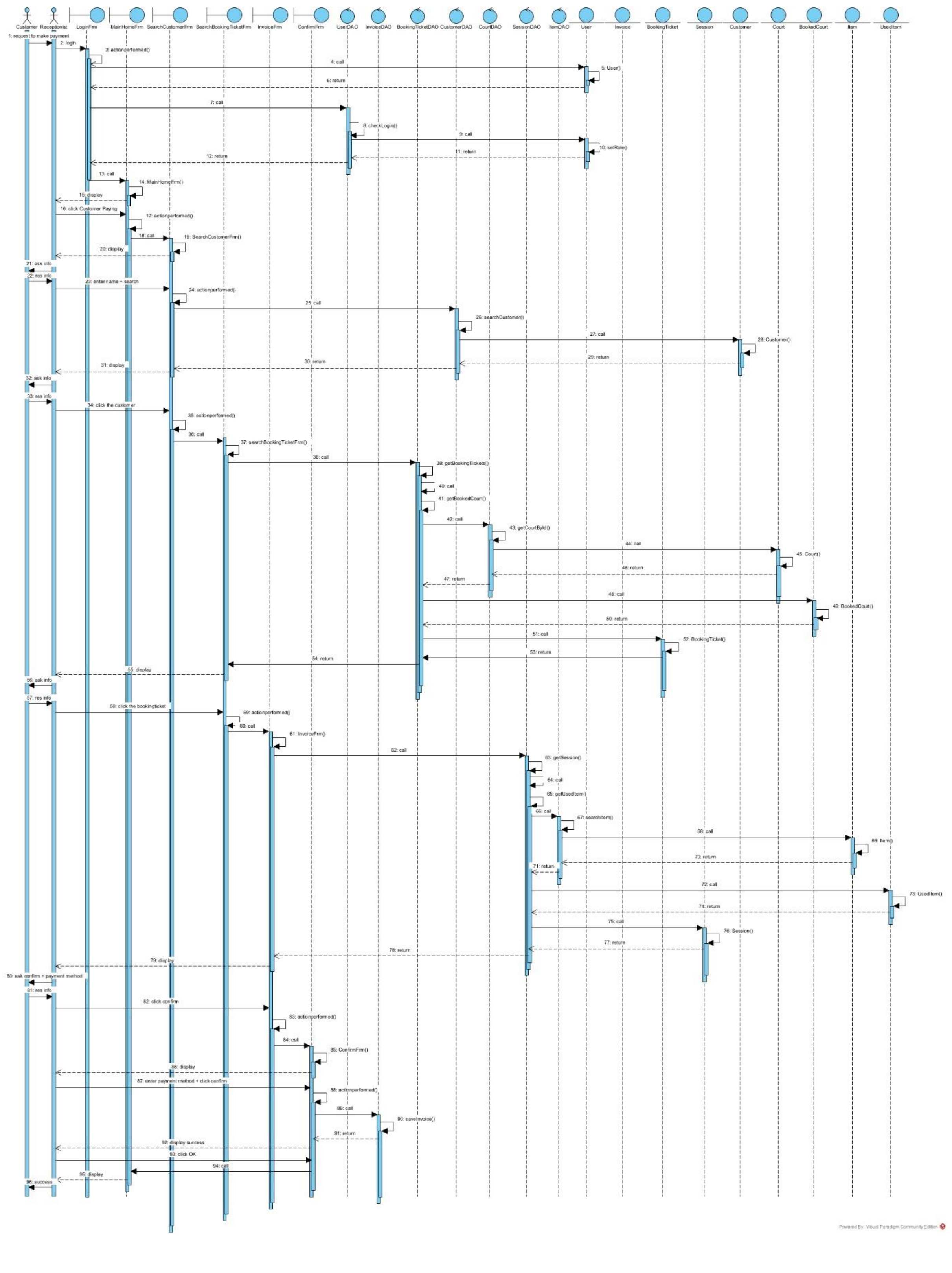


9. Sequence diagram of design phase of the module

1. A customer come to make payment
2. The receptionist enter username, password and click login button on LoginFrm
3. The method actionperformed() of LoginFrm() is called
4. The method actionperformed() calls User to create an User object
5. The class User packs the information into an User object
6. The class User returns User object to the method actionperformed().
7. The method actionperformed() calls method checkLogin() of the class UserDAO
8. The method checkLogin() checks the login information
9. The method checkLogin() calls the class User set two more attributes: name, role
10. The class User calls its method setName(), setRole()
11. The class User returns the User object to the method checkLogin()
12. The method checkLogin() returns the results to the actionPerformed()
13. The method actionPerformed() calls the class MainHomeFrm
14. The constructor MainHomeFrm() is called
15. The interface MainHomeFrm is shown to the receptionist
16. The receptionist clicks on the customer paying button
17. The method actionperformed() of the class MainHomeFrm is called
18. The method actionPerformed() calls the class SearchCustomerFrm
19. The constructor SearchCustomerFrm() is called

20. The interface SearchCustomerFrm is shown to the receptionist
21. The receptionist asks the customer about their name
22. The customer responds with their name
23. The receptionist enter their name into the name field and click search
24. The method actionperformed() is called
25. The method actionPerformed() calls the method searchCustomer() of the class CustomerDAO
26. The method searchCustomer() is executed
27. The method searchCustomer() calls the class Customer to pack the results
28. The class Customer pack each result into a Customer object
29. The class Customer returns the object to the method searchCustomer()
30. The method searchCustomer() returns the list of Customer object to the method actionperformed()
31. The method actionperformed() displays the results on the interface SearchCustomerFrm to the receptionist
32. The receptionist asks for the customer phone number
33. The customer responds with their phone number
34. The receptionist clicks on the correct customer
35. The method actionperformed() of the class SearchCustomerFrm is called
36. The method actionperformed() calls the constructor SearchBookingTicketFrm() of the class SeacrhBookingTicket
37. The constructor SearchBookingTicketFrm() is called
38. The constructor SearchBookingTicketFrm() calls the method getBookingTickets() of the class BookingTicketDAO
39. The method getBookingTickets() of the class BookingTicketDAO is executed
40. The method getBookingTickets() call the method getBookedCourt()
41. The method getBookedCourt() is executed
42. The method getBookedCourt() call the method getCourtById() of the class CourtDAO
43. The method getCourtById() is executed
44. The method getCourtById() call the class Court to pack the result
45. The class Court pack the result to a Court object
46. The class Court return the Court object to the method getCourtById()
47. The method getCourtById() return the Court object to the method getBookedCourt()
48. The method getBookedCourt() call the class BookedCourt to pack the results
49. The class BookedCourt pack each results into a BookedCourt object
50. The class BookedCourt return the BookedCourt object to the method getBookedCourt()
51. The method getBookingTicket() calls the class BookingTicket() to pack the results
52. The class BookingTicket pack each result into a BookingTicket object
53. The class BookingTicket returns the BookingTicket object to the method getBookingTicket()
54. The method getBookingTicket() returns list of BookingTicket object to the contructor SearchBookingTicketFrm()
55. The constructor SearchBookingTicketFrm() displays the results on the interface SearchBookingTicketFrm to the receptionist
56. The receptionist then asks the customer which booking ticket they want to make payment for

57. The customer responds with the booking ticket they want to make payment for
58. The receptionist click the correct booking ticket
59. The method actionperformed() of the class SearchBookingTicketFrm is called
60. The method actionperformed() call the constructor InvoiceFrm() of the class InvoiceFrm
61. The contructor InvoiceFrm() is called
62. The contructor InvoiceFrm() call the method getSession() of the class SessionDAO()
63. The method getSession() is executed
64. The method getSession() call the method getUsedItem()
65. The method getUsedItem() is called
66. The method getUsedItem() call the method searchItem() of the class ItemDAO
67. The method searchItem() is executed
68. The method searchItem() call the class Item to pack the result
69. The class Item pack the result into an Item object
70. The class Item return the Item object to the method searchItem()
71. The method searchItem() return the result to the getUsedItem() method
72. The method getUsedItem() call the class UsedItem to pack the result
73. The class UsedItem pack each result into a UsedIem object
74. The class UsedItem return the UsedItem object to the method getUsedItem()
75. The method getSession() call the class Sesion to pack the result
76. The class Session pack each result into a Session object
77. The class Session return the Session object to the method getSession()
78. The method getSession() return a list of Session object to the constructor InvoiceFrm()
79. The constructor InvoiceFrm() displays the results on the interface InvoiceFrm to the receptionist
80. The receptionist asks the customer for the confirmation and payment method
81. The customer confirms and responds with the payment method
82. The receptionist clicks confirm
83. The method actionperformed() of the class InvoiceFrm is executed
84. The method actionperformed() calls the class ConfirmFrm
85. The constructor ConfirmFrm() is called
86. The interface ConfirmFrm is shown to the receptionist
87. The receptionist enter the payment method and clicks confirm
88. The method actionperformed() of the class ConfirmFrm is called
89. The method actionperformed() call the method saveInvoice() of the class InvoiceDAO
90. The method saveInvoice() is executed
91. The method saveInvoice() return the result to the method actionperformed()
92. The method actionperformed() display a success message to the receptionist
93. The receptionist clicks OK button
94. The method actionPerformed() calls the interface MainHomeFrm
95. The interface MainHomeFrm is shown to the Customer
96. The receptionist confirms the successful payment to the customer



10. Test plan and standard testcase for blackbox testcase of the module

Test plan

No.	**Module**	**Test case**
1	Customer paying	Customer exists, booking ticket exists
2		Customer exists, booking ticket does not exist
3		Customer does not exist, booking ticket exists
4		Customer does not exist, booking ticket does not exist
5		Make payment 2 consecutive times for 1 booking ticket

Test case No. 1

The database before testing:

tblUser:

id	username	password	name	role
1	a	a	A	receptionist

tblCustomer:

id	name	address	phone	email	note
1	An	Huynh Thuc Khang	9172386245	an@gmail.com	
2	Anh	Lang Ha	1234685647	anh@gmail.com	
3	B	Yen Hoa	1234145613	b@gmail.com	

tblBookingTicket:

id	dateOfContract	depositAmount	CustomerId	UserId
1	30/03/2024	0	1	1
2	01/04/2024	0	2	1
3	20/06/2024	0	1	1

tblCourt:

id	type	price
1	single	500.000
2	single	500.000

tblBookedCourt:

id	startDate	endDate	startHour	endHour	dayOfWeek	CourtId	BookingTicketIId
1	01/04/2024	30/06/2024	20:00	22:00	Monday	1	1
2	01/05/2024	31/08/2024	14:00	16:00	Friday	2	3
3	01/07/2024	30/09/2024	09:00	11:00	Saturday	1	2

tblRentalSession:

id	receptionTime	returnTime	date	BookedCourtId
1	20:00	22:00	01/04/2024	1

2	20:00	22:00	08/04/2024	1
3	20:00	22:00	15/04/2024	1
4	20:00	22:00	22/04/2024	1
5	20:00	22:00	29/04/2024	1
6	20:00	22:00	06/05/2024	1
7	20:00	22:00	13/05/2024	1
8	20:00	22:00	20/05/2024	1
9	20:00	22:00	27/05/2024	1
10	20:00	22:00	03/06/2024	1
11	20:00	22:00	10/06/2024	1
12	20:00	22:00	17/06/2024	1
13	20:00	22:00	24/06/2024	1
14	14:00	16:00	03/05/2024	2
15	14:00	16:00	10/05/2024	2
16	14:00	16:00	17/05/2024	2
17	14:00	16:00	24/05/2024	2
18	14:00	16:00	31/05/2024	2
19	14:00	16:00	07/06/2024	2
20	14:00	16:00	14/06/2024	2
21	14:00	16:00	21/06/2024	2
22	14:00	16:00	28/06/2024	2
23	14:00	16:00	05/07/2024	2
24	14:00	16:00	12/07/2024	2
25	14:00	16:00	19/07/2024	2
26	14:00	16:00	26/07/2024	2
27	14:00	16:00	02/08/2024	2
28	14:00	16:00	09/08/2024	2
29	14:00	16:00	16/08/2024	2
30	14:00	16:00	23/08/2024	2
31	14:00	16:00	30/08/2024	2
32	9:00	11:00	06/07/2024	3
33	9:00	11:00	13/07/2024	3
34	9:00	11:00	20/07/2024	3
35	9:00	11:00	27/07/2024	3
36	9:00	11:00	03/08/2024	3
37	9:00	11:00	10/08/2024	3
38	9:00	11:00	17/08/2024	3
39	9:00	11:00	24/08/2024	3
40	9:00	11:00	31/08/2024	3
41	9:00	11:00	07/09/2024	3
42	9:00	11:00	14/09/2024	3
43	9:00	11:00	21/09/2024	3
44	9:00	11:00	28/09/2024	3

tblItem:

id	name	price
1	Pepsi	10.000
2	Revive	10.000

tblUsedItem:

id	quantity	currentPrice	RentalSessionId	ItemId
1	10	10.000	1	1
2	5	10.000	1	2
3	5	10.00	2	2

tblInvoice:

id	paymentDate	paymentMethod	totalAmount	UserId	CustomerId	BookingTicketId

Testing scenario and expected results:

Scenario	**Expected result**
1. Login Enter username="a"	Main home appear

Password="a" Click Login	
2. Click Customer Paying	The interface to search customer appear with a text field and a search button
3. Enter Customer name = ''An" Click Search	Interface show list of customer

Customer name = An				Search
No.	Name	Address	Phone	Email
1	An	Huynh Thuc Khang	9172386245	an@gmail.com
2	Anh	Lang Ha	1234685647	anh@gmail.com

4. Click first row	Interface show list of booking ticket of that cutomer

Id	Contract Date	Deposit
1	30/03/2024	0
2	01/04/2024	0

5. Click first row	Interface show invoice

Invoice		Confirm	Go Back
Customer Information			
Name:	An		
Phone:	9172386245		

Email:	an@gmail.com		
Field Cost			
Court	Court type	Sessions	Total Cost
1	single	13	6,500,000 VND
2	single	1	500,000VND
Items used			
Session	Item Name	Quantity	Price
1	Pepsi	10	100000
1	Revive	5	50000
2	Revive	5	50000
		Field Cost	7,000,000 VND
		Items Cost	200,000VND
		Total	7,200,000VND

6. Click confirm	A message box appear with field to enter payment method, a cancel and a confirm button

Payment Confirmation	
Payment Method:	
Confirm	Cancel

7. Enter Payment method = “Cash” Click confirm	A success message box appear with Ok button
8. Click Ok	Return to the main home interface

The database after testing:

Only one tables change:

tblInvoice:

id	paymentDate	paymentMethod	total	UserId	CustomerId	BookingTicketId
1	01/07/2024	Cash	7,200,000	1	1	1