Assignment 1

Professor Sujith

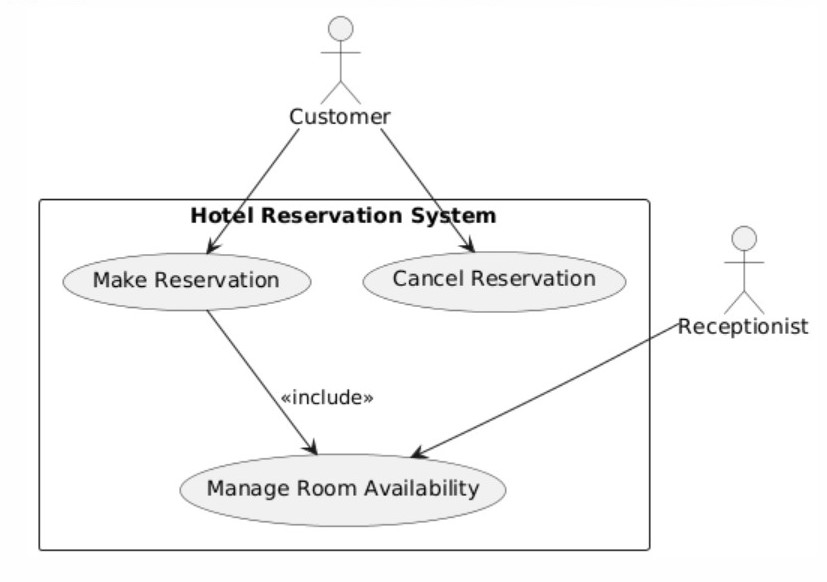
ICS220 Programming Fund.

Zayed University

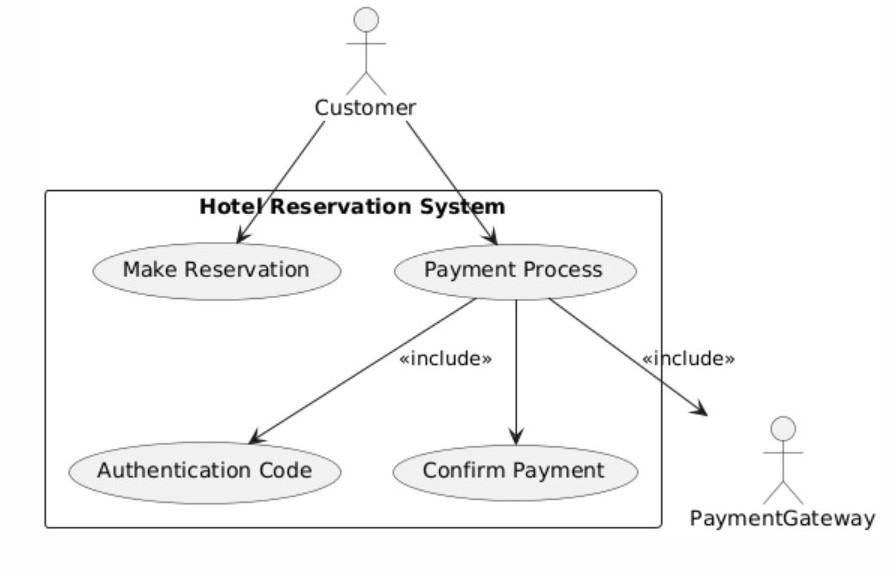
Al Hamzah Zohdi Nassar

9/30/2024

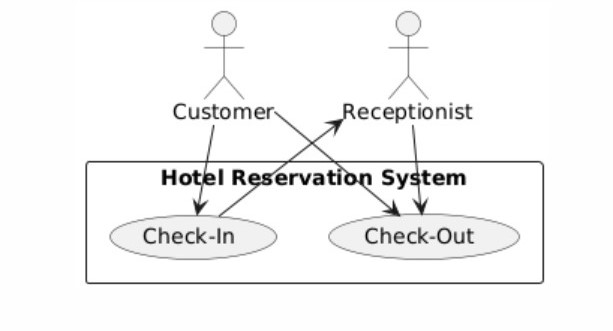
**Use Case Diagram 1: “Make Reservation and Room Management”**



**Use Case Diagram: “Payment Process”**



**Use Case Diagram 3: “Check-In/Check-Out Process”**



**Use Case Descriptions:**

**Use Case Description 1: Customer – Make Reservation**

|  |  |
| --- | --- |
| Use Case | Make Reservation |
| Actor | Customer |
| Trigger | Customer Decides on booking a room with the hotel |
| Preconditions | The customer must have access to the system of reservation. |
| Main Scenario | 1. Customer Provides necessary information required to book such as dates, number of guests, etc. 2. The system checks for the availability of the rooms with said preferences. 3. The system process the reservation and sends the customer a confirmation. |
| Exceptions | 1. There are no room available, so the system suggests a different room type or different dates. 2. If the details are invalid or incomplete, the system requests the missing information. |

#### Use Case Description 2: Receptionist – Manage Room Availability

|  |  |
| --- | --- |
| Use Case | Manage Room Availability |
| Actor | Receptionist |
| Trigger | Receptionist Updates the system with the room availability after reservation is complete. |
| Preconditions | The system must be updated to the latest reservation data. |
| Main Scenario | 1. The receptionist logs onto the reservation system 2. The reservation system displays room availability 3. The receptionist updates the room status after reservation booking or cancellation is complete. |
| Exceptions | 1. The system is not up to date and some incorrect information may be displayed. 2. The receptionist needs to correct any error for in data entries. |

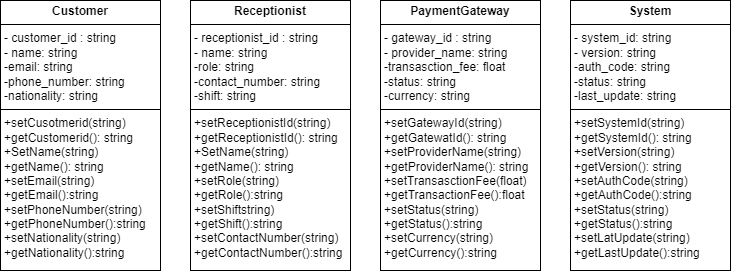
#### Use Case Description 3: Payment Gateway – Process Payment

|  |  |
| --- | --- |
| Use Case | Process Payment |
| Actor | Payment Gateway |
| Trigger | Customer completes the reservation and is then sent to the reservation payment to complete the payment. |
| Preconditions | A valid reservation should exist |
| Main Scenario | 1. The payment gateway processes the customer’s payment. 2. The system displays that the reservation is paid and complete. 3. Receipt is created and sent to the customer via email or SMS. |
| Exceptions | 1. If the payment fails, the system either asks the customer to retry the payment or cancels the payment. 2. The system reserves the room for the customer incase the payment gateway is down. |

#### Use Case Description 4: System – Authentication Code

|  |  |
| --- | --- |
| Use Case | Authentication Code |
| Actor | System |
| Trigger | The system requires authentication for certain actions to be made. |
| Preconditions | The customer must be registered and authenticated. |
| Main Scenario | 1. Customer receives an authentication code via email or SMS. 2. The customer copies the code into the system for authentication. 3. The system validates the code and then allows the customer to proceed with the requested action. |
| Exceptions | 1. If authentication fails, the customer is provided with another code. 2. The system blocks the user from preforming actions for a certain amount of time. |

**Classes:**

****

**Python Codes:**

***Customer*:**

class Customer:

"""

A class representing a customer making a hotel reservation.

Attributes:

\_customer\_id (str): The unique identifier for the customer.

\_name (str): The name of the customer.

\_email (str): The email of the customer.

\_phone\_number (str): The phone number of the customer.

\_nationality (str): The nationality of the customer.

"""

def \_\_init\_\_(self, customer\_id, name, email, phone\_number, nationality):

"""Initialize the customer with the provided details."""

self.\_customer\_id = customer\_id

self.\_name = name

self.\_email = email

self.\_phone\_number = phone\_number

self.\_nationality = nationality

# Getter and Setter methods

def set\_customer\_id(self, customer\_id):

"""Set the customer ID."""

self.\_customer\_id = customer\_id

def get\_customer\_id(self):

"""Get the customer ID."""

return self.\_customer\_id

def set\_name(self, name):

"""Set the customer's name."""

self.\_name = name

def get\_name(self):

"""Get the customer's name."""

return self.\_name

def set\_email(self, email):

"""Set the customer's email."""

self.\_email = email

def get\_email(self):

"""Get the customer's email."""

return self.\_email

def set\_phone\_number(self, phone\_number):

"""Set the customer's phone number."""

self.\_phone\_number = phone\_number

def get\_phone\_number(self):

"""Get the customer's phone number."""

return self.\_phone\_number

def set\_nationality(self, nationality):

"""Set the customer's nationality."""

self.\_nationality = nationality

def get\_nationality(self):

"""Get the customer's nationality."""

return self.\_nationality

# Example usage of Customer class

customer = Customer("C001", "Al Hamzah Nassar", "202234086@zu.ac.ae", "0506164061", "UAE")

print(f"Customer Name: {customer.get\_name()}")

print(f"Email: {customer.get\_email()}")

print(f"Phone Number: {customer.get\_phone\_number()}")

print(f"Nationality: {customer.get\_nationality()}")

customer.set\_name("alhamzahnassar")

print(f"Updated Name: {customer.get\_name()}")

**Python Codes:**

***Receptionist*:**

class Receptionist:

"""

A class representing a hotel receptionist.

Attributes:

\_receptionist\_id (str): The unique identifier for the receptionist.

\_name (str): The name of the receptionist.

\_role (str): The role of the receptionist.

\_shift (str): The shift of the receptionist.

\_contact\_number (str): The contact number of the receptionist.

"""

def \_\_init\_\_(self, receptionist\_id, name, role, shift, contact\_number):

“””Initialize the receptionist with the provided details.”””

self.\_receptionist\_id = receptionist\_id

self.\_name = name

self.\_role = role

self.\_shift = shift

self.\_contact\_number = contact\_number

# Getter and Setter methods

def set\_receptionist\_id(self, receptionist\_id):

"""Set the receptionist ID."""

self.\_receptionist\_id = receptionist\_id

def get\_receptionist\_id(self):

"""Get the receptionist ID."""

return self.\_receptionist\_id

def set\_name(self, name):

"""Set the receptionist's name."""

self.\_name = name

def get\_name(self):

"""Get the receptionist's name."""

return self.\_name

def set\_role(self, role):

"""Set the receptionist's role."""

self.\_role = role

def get\_role(self):

"""Get the receptionist's role."""

return self.\_role

def set\_shift(self, shift):

"""Set the receptionist's shift."""

self.\_shift = shift

def get\_shift(self):

"""Get the receptionist's shift."""

return self.\_shift

def set\_contact\_number(self, contact\_number):

"""Set the receptionist's contact number."""

self.\_contact\_number = contact\_number

def get\_contact\_number(self):

"""Get the receptionist's contact number."""

return self.\_contact\_number

# Example usage of Receptionist class

receptionist = Receptionist("R001", "Al Hamzah Nassar", "Front Desk", "Evening", "0506164061")

print(f"Receptionist Name: {receptionist.get\_name()}")

print(f"Role: {receptionist.get\_role()}")

print(f"Shift: {receptionist.get\_shift()}")

print(f"Contact: {receptionist.get\_contact\_number()}")

receptionist.set\_shift("Evening")

print(f"Updated Shift: {receptionist.get\_shift()}")

**Python Codes:**

***PaymentGateway*:**

class PaymentGateway:

"""

A class representing the payment gateway used to process transactions.

Attributes:

\_gateway\_id (str): The unique identifier for the payment gateway.

\_provider\_name (str): The name of the payment provider.

\_transaction\_fee (float): The fee charged per transaction.

\_status (str): The current status of the payment gateway (e.g., active, inactive).

\_currency (str): The currency used for payments.

"""

def \_\_init\_\_(self, gateway\_id, provider\_name, transaction\_fee, status, currency):

“””Initialize the payment gateway with the provided details.”””

self.\_gateway\_id = gateway\_id

self.\_provider\_name = provider\_name

self.\_transaction\_fee = transaction\_fee

self.\_status = status

self.\_currency = currency

# Getter and Setter methods

def set\_gateway\_id(self, gateway\_id):

"""Set the payment gateway ID."""

self.\_gateway\_id = gateway\_id

def get\_gateway\_id(self):

"""Get the payment gateway ID."""

return self.\_gateway\_id

def set\_provider\_name(self, provider\_name):

"""Set the provider name."""

self.\_provider\_name = provider\_name

def get\_provider\_name(self):

"""Get the provider name."""

return self.\_provider\_name

def set\_transaction\_fee(self, fee):

"""Set the transaction fee."""

self.\_transaction\_fee = fee

def get\_transaction\_fee(self):

"""Get the transaction fee."""

return self.\_transaction\_fee

def set\_status(self, status):

"""Set the status of the payment gateway."""

self.\_status = status

def get\_status(self):

"""Get the status of the payment gateway."""

return self.\_status

def set\_currency(self, currency):

"""Set the currency used by the payment gateway."""

self.\_currency = currency

def get\_currency(self):

"""Get the currency used by the payment gateway."""

return self.\_currency

# Example usage of PaymentGateway class

payment\_gateway = PaymentGateway("PG001", "Visa", 5, "Active", "AED")

print(f"Provider Name: {payment\_gateway.get\_provider\_name()}")

print(f"Transaction Fee: {payment\_gateway.get\_transaction\_fee()}%")

print(f"Currency: {payment\_gateway.get\_currency()}")

payment\_gateway.set\_status("Inactive")

print(f"Updated Status: {payment\_gateway.get\_status()}")

**Python Codes:**

***System*:**

class System:

"""

A class representing the hotel management system.

Attributes:

\_system\_id (str): The unique identifier for the system.

\_version (str): The current version of the system.

\_auth\_code (str): The current authentication code in use.

\_status (str): The operational status of the system (e.g., online, offline).

\_last\_update (str): The timestamp of the last system update.

"""

def \_\_init\_\_(self, system\_id, version, auth\_code, status, last\_update):

“”” Initialize the system with the provided details. """

self.\_system\_id = system\_id

self.\_version = version

self.\_auth\_code = auth\_code

self.\_status = status

self.\_last\_update = last\_update

# Getter and Setter methods

def set\_system\_id(self, system\_id):

"""Set the system ID."""

self.\_system\_id = system\_id

def get\_system\_id(self):

"""Get the system ID."""

return self.\_system\_id

def set\_version(self, version):

"""Set the system version."""

self.\_version = version

def get\_version(self):

"""Get the system version."""

return self.\_version

def set\_auth\_code(self, auth\_code):

"""Set the authentication code."""

self.\_auth\_code = auth\_code

def get\_auth\_code(self):

"""Get the authentication code."""

return self.\_auth\_code

def set\_status(self, status):

"""Set the system status."""

self.\_status = status

def get\_status(self):

"""Get the system status."""

return self.\_status

def set\_last\_update(self, last\_update):

"""Set the last update timestamp."""

self.\_last\_update = last\_update

def get\_last\_update(self):

"""Get the last update timestamp."""

return self.\_last\_update

# Example usage of System class

system = System("S001", "1.0", "123456", "Online", "2024-09-30")

print(f"System Version: {system.get\_version()}")

print(f"Auth Code: {system.get\_auth\_code()}")

print(f"System Status: {system.get\_status()}")

print(f"Last Update: {system.get\_last\_update()}")

system.set\_version("2.0")

print(f"Updated Version: {system.get\_version()}")

GitHub: “

**Summary of learnings:**

Throughout this project, I gained a deeper understanding of class diagrams and their significance in system design. Working with UML class diagrams helped me see how the system is put together, focused on the important properties and methods for the actors that I used: Customer, Receptionist, PaymentGateway, and System. This process helped me understand how objects work and interact in an object-oriented environment. It also gave me a good idea of what each class does and how it connects to other classes in the system. It was helpful to learn how to use diagram.io to create and the designs. Because the platform's layout was easy to use, it was simple to make diagrams that looked professional. I could alter and change almost everything about my Use Case Diagrams, making sure that the diagrams were clear and well-organized. I also learned more about important Object-Oriented Programming (OOP) ideas through this project. Through encapsulation, I learned how to use private variables with getter and setter methods to keep data safe. I was able to make a better design because abstraction helped me focus on the most important parts of each class. Another important idea was modularity, which made sure that each class worked as a separate unit that added to the system's general functionality. I got better at both UML modeling and OOP design overall because of this experience. It taught me how to quickly turn an idea for a system into a visual model and how to put that model into action using well-structured code. I also learned how to use diagram.io to make the system come to life in a clear and organized way.