CAR RENTAL SYSTEM
PROJECT REPORT

TEAM MEMBERS:
DHARSHANI MURUGAIYAN
SOBHIKA KAMALAVASAN
VINODHA R

ABSTRACT:

The Car Rental System is an extensive, object-oriented software application aimed at simplifying and mechanizing the car rental process for customers and administrators alike. The system is built with Python and adopts a modular framework that includes packages including entity, dao, service, exception, and util, exemplifying strong observance of software engineering concepts such as abstraction, encapsulation, and concerns separation. The backend connectivity is provided through MySQL, which facilitates the smooth storage and retrieval of vehicle, customer, lease, and payment information.

At the heart of this system is a cleanly defined database schema with CRUD support for major entities like Car, Customer, Lease, and Payment. The system provides efficient transaction processing for leasing cars, their availability tracking, lease ID generation, and customer management with validations and custom exceptions. The system keeps clean error handling and secure database interactions through user-defined exceptions and utility classes.

The business logic like checking car availability, lease duration and charges calculation, and lease duration history maintenance is implemented by the service layer. DAO interfaces and implementations allow for a neat separation of business logic and database interactions, keeping the system very much maintainable and scalable to integrate with web or mobile platforms in the future.

This Car Rental System project is a typical example of an actual software application that not only solidifies object-oriented programming and database design concepts but also lays down the groundwork for a professional-level rental service portal. It is specifically ideal for learning purposes and can be further developed into a full-stack web application using frameworks such as FastAPI or Flask for the backend and React or Angular for the frontend.

INTRODUCTION:

In the modern transportation landscape, the need for flexible and accessible mobility solutions has led to the widespread adoption of car rental services. A Car Rental System plays a vital role in enabling customers to lease vehicles for short or long durations without the responsibilities of ownership. As digital transformation accelerates in the automotive industry, automating the car rental process has become essential for improving customer experience, operational efficiency, and real-time management of vehicle resources.

The Car Rental System developed in this project is a console-based application built using Python with MySQL database connectivity. It is structured using object-oriented programming principles and follows a layered architecture with clear separation of concerns across packages such as entity, dao, service, exception, and util. This system provides core functionalities such as registering customers, listing available vehicles, creating and managing lease records, and handling payment details, making it a compact and efficient solution for small and medium-sized rental agencies.

The database schema is designed to store key details related to cars, customers, leases, and payments, enabling efficient tracking and management of rental operations. The system uses Data Access Object (DAO) patterns to interact with the backend, ensuring data consistency and ease of maintenance. With built-in validation and exception handling, the application ensures robust and secure operations throughout the rental process.

This project serves as a practical implementation of software development concepts, combining backend data handling with business logic execution. It can be further extended into a web-based or mobile application, making it highly adaptable for real-world deployment. The system not only simplifies rental workflows but also offers a solid learning experience for students and developers aiming to understand scalable application development using Python and databases.

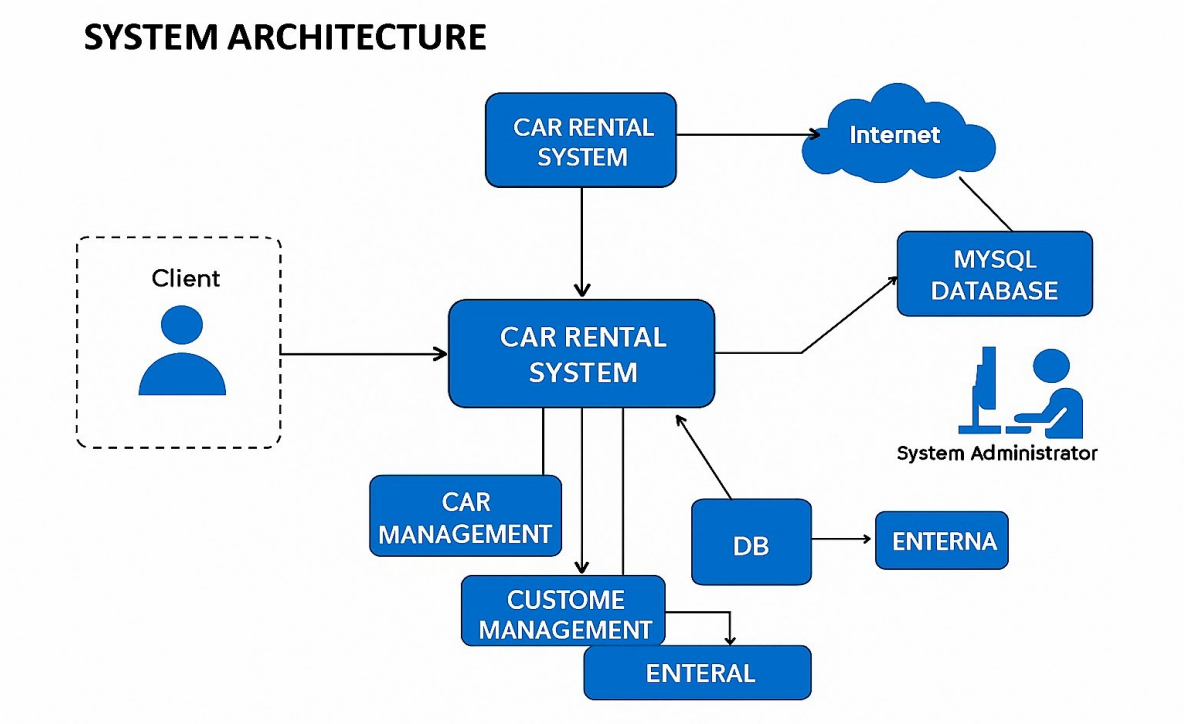
SYSTEM ANALYSIS:

The Car Rental System is designed to streamline the process of renting vehicles by automating customer registration, vehicle listing, lease management, and payment tracking. The system targets common challenges in manual rental operations, such as data inconsistencies, delayed lease processing, and inefficient customer management. Through this analysis, the system's functionalities, user interactions, and technical framework are evaluated to assess its effectiveness and efficiency.

The system identifies three primary actors: the Administrator, who manages the fleet of vehicles and overall operations; the Customer, who registers and rents vehicles; and the System, which ensures communication between the database and application layers. It supports functionalities like vehicle availability checks, rental period calculations, and payment verification. Each module is designed with minimal coupling and high cohesion, ensuring the system is modular and scalable.

From a technical perspective, the application is built using Python for logic and MySQL for persistent storage. The system follows an MVC-style separation using structured packages: entity for models, dao for data access, service for business logic, and exception for custom error handling. This approach enhances maintainability, debugging, and future feature expansion. The use of JDBC-like connectivity ensures real-time data flow between the UI and the database.

In conclusion, the system is capable of addressing core rental operations while minimizing manual intervention and errors. It is ideal for academic demonstration and also serves as a prototype for developing a more comprehensive web or mobile-based car rental platform. The design ensures a user-friendly experience, improved resource utilization, and reliable transaction processing.



SYSTEM DESIGN:

The system design of the Car Rental System follows a modular and layered architecture to ensure maintainability, scalability, and reusability. The architecture is structured into distinct layers—Presentation Layer, Business Logic Layer, Data Access Layer, and Database Layer—which work together to deliver core rental functionalities efficiently. Each layer is developed with specific responsibilities to ensure a clean separation of concerns.

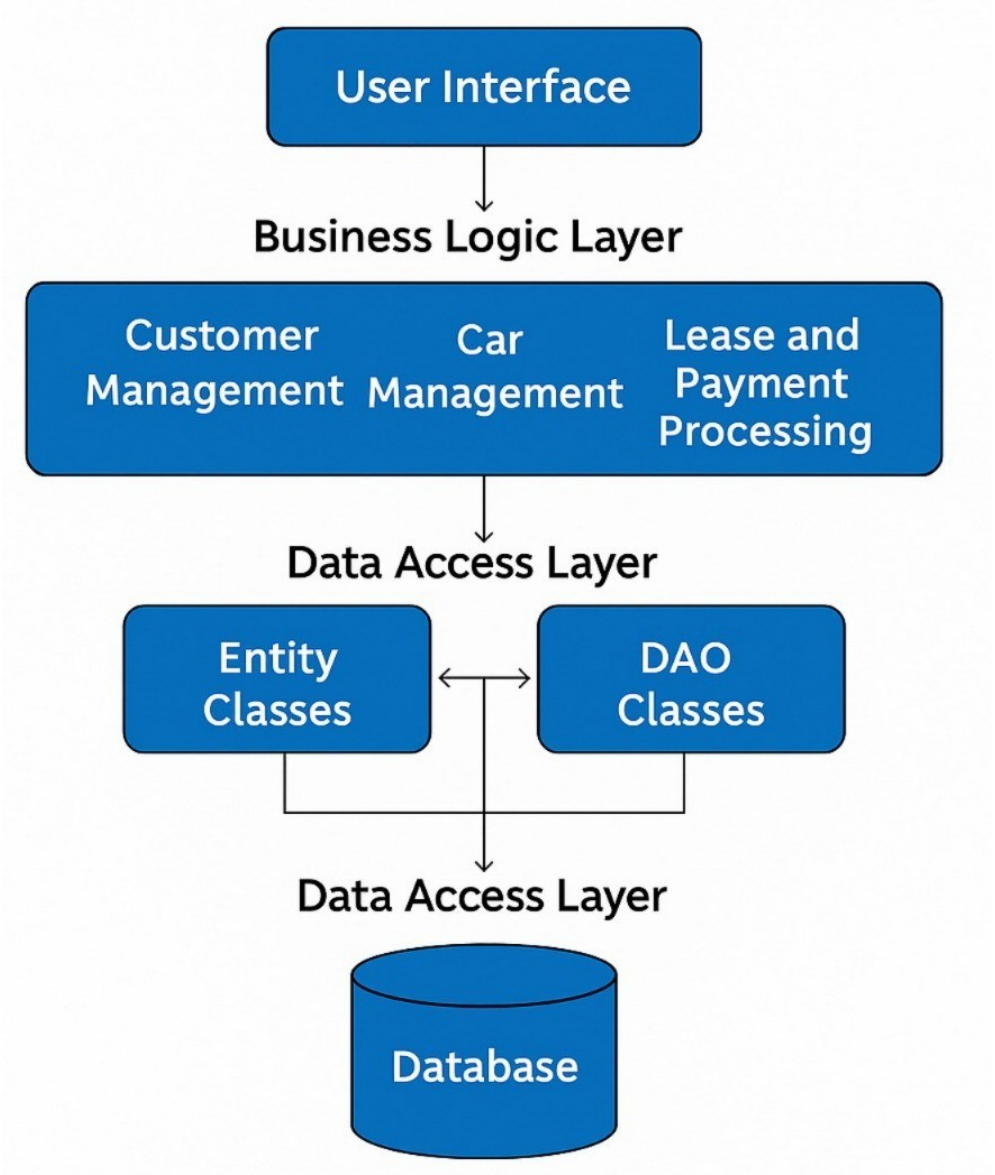
At the heart of the system lies the Entity Layer (entity package), which defines the real-world business objects like Customer, Vehicle, Lease, and Payment. Each class contains private attributes with getter/setter methods, default and parameterized constructors, and overrides of basic object methods if necessary. These entities serve as data carriers between different layers, representing tables from the underlying MySQL database.

The Data Access Layer (DAO) is responsible for all interactions with the database. It includes interfaces and implementation classes such as ICarLeaseRepositoryImpl, which handle operations like inserting customer records, retrieving available vehicles, managing lease records, and processing payments. The DAO layer abstracts the database logic and ensures that the rest of the application remains unaffected by changes in the storage layer.

The Business Logic Layer acts as a bridge between the DAO and the UI. This layer implements validation logic, lease duration computation, availability checks, and total cost calculation. It also handles user-defined exceptions like CustomerNotFoundException, VehicleNotAvailableException, and others to make error handling more structured and user-friendly. This encapsulation of logic supports modular testing and easier debugging.

The User Interface Layer, typically the main package, facilitates user interaction via a command-line interface. Users can log in, register, search vehicles by type or availability, book rentals, and view payment information. This layer gathers input, invokes services, and displays processed output, ensuring a smooth user experience. In a real-world extension, this layer could be adapted to a web-based or mobile interface.

Lastly, the Database Layer is represented by a well-normalized MySQL database schema that includes tables like customers, vehicles, leases, and payments. Relationships are properly established using foreign keys, and primary keys are auto-incremented for unique identification. The schema is designed to reduce redundancy and support quick query execution through indexing and proper relational mapping.


User Interface
Business Logic Layer
Customer Management
Car Management
Lease and Payment Processing
Data Access Layer
Entity Classes
DAO Classes
Data Access Layer
Database

BUSINESS LOGIC:

The business logic of the Car Rental System is designed to streamline the core operations of vehicle leasing by managing customers, vehicles, leases, and payments efficiently. At its core, the system ensures that only available vehicles are booked, lease durations are respected, and accurate financial transactions are recorded for both the business and the customer. The logic is encapsulated in service classes that bridge the gap between user actions and database operations.

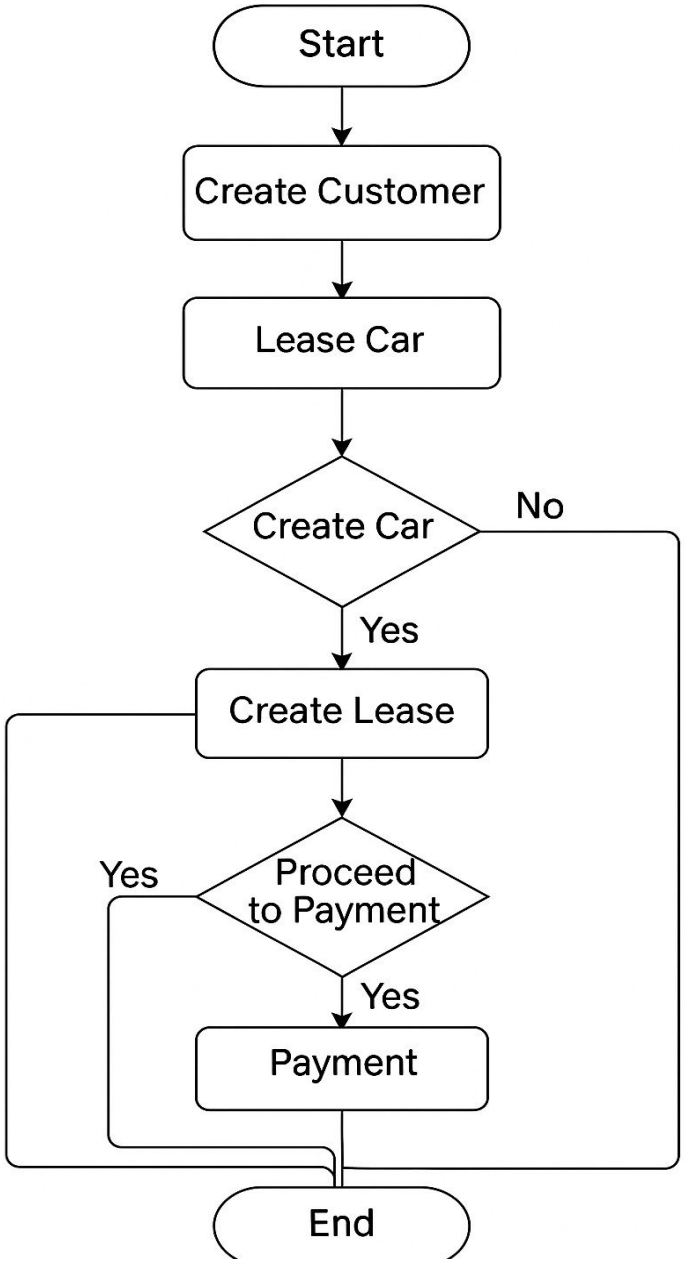
When a customer requests to lease a vehicle, the system first checks for vehicle availability through the Vehicle entity and its associated DAO implementation. If the vehicle is available and the customer is valid, a lease is created with a start date, expected return date, and lease amount based on the vehicle's rental rate and the duration. This computation forms the crux of the business logic, as it directly influences revenue and availability tracking.

In terms of customer management, the system ensures each customer record is uniquely identified and stored with essential information such as name, driving license number, and contact details. Before initiating a lease, the business logic validates that the customer exists in the system. If not, the customer is automatically registered, reducing friction in the transaction process. These customer validation and registration rules are enforced in the service layer, ensuring data consistency and reducing manual entry errors.

Another critical part of the business logic is vehicle status management. Once a vehicle is leased, its status is marked as "unavailable," preventing double bookings or over-leasing. Upon return of the vehicle, the system updates its availability and, if needed, adjusts the return date or any penalties for late return. This feature plays a vital role in fleet management and ensures that vehicles are optimally utilized without operational conflicts.

Payment processing in the system is tightly coupled with the lease lifecycle. When a lease is finalized, the expected rental fee is computed, and the customer must complete the payment process to confirm the transaction. The payment entity captures transaction details like amount, payment date, and payment method, which are then stored and can be retrieved for future audits or customer support. This part of the logic ensures financial transparency and helps businesses manage their revenue streams effectively.

Finally, the system is built with scalability in mind. Although it currently supports basic business operations, the layered architecture and modular service interfaces allow for the addition of advanced features such as promotional pricing, customer loyalty points, insurance integration, and driver assignment. These business rules can be incorporated into the existing logic with minimal disruption, making the Car Rental System a future-ready platform suitable for real-world deployment in vehicle leasing companies.



IMPLEMENTATION:

Setup MySQL Database:

- Create database and tables (Vehicle, Customer, Lease, Payment) using provided SQL scripts.

1. Vehicle Table

```
CREATE TABLE vehicle (
    vehicleID INT AUTO_INCREMENT PRIMARY KEY,
    make VARCHAR(50) NOT NULL,
```

```
    model VARCHAR(50) NOT NULL,
    year INT NOT NULL,
    dailyRate DECIMAL(10,2) NOT NULL,
    status INT NOT NULL,
    passengerCapacity INT NOT NULL,
    engineCapacity INT NOT NULL
);
```

2. Customer Table

```
CREATE TABLE customer (
    customerID INT AUTO_INCREMENT PRIMARY KEY,
    firstName VARCHAR(50) NOT NULL,
    lastName VARCHAR(50) NOT NULL,
    email VARCHAR(100),
    phoneNumber VARCHAR(15)
);
```

3. Lease Table

```
CREATE TABLE lease (
    leaseID INT AUTO_INCREMENT PRIMARY KEY,
    carID INT,
    customerID INT,
    startDate DATE,
    endDate DATE,
    leaseType VARCHAR(20),
    FOREIGN KEY (carID) REFERENCES vehicle(vehicleID),
    FOREIGN KEY (customerID) REFERENCES customer(customerID)
);
```

4. Payment Table

```
CREATE TABLE payment (
    paymentID INT AUTO_INCREMENT PRIMARY KEY,
    leaseID INT,
    paymentDate DATE,
    amount DECIMAL(10,2),
    FOREIGN KEY (leaseID) REFERENCES lease(leaseID)
);
```

Create Entity Classes:

- Classes like Vehicle, Customer, Lease and Payment are defined with private attributes, constructors, and getter/setter methods.

Vehicle Class:

```
class Vehicle:
    def __init__(self, vehicleID=None, make=None, model=None, year=None, dailyRate=None, status=None,
                 passengerCapacity=None, engineCapacity=None):
        self.vehicleID = vehicleID
        self.make = make
        self.model = model
        self.year = year
        self.dailyRate = dailyRate
        self.status = status
        self.passengerCapacity = passengerCapacity
        self.engineCapacity = engineCapacity
```

Customer Class :

```
class Customer:
    def __init__(self, customerID=None, firstName=None, lastName=None, email=None, phoneNumber=None):
        self.customerID = customerID
        self.firstName = firstName
        self.lastName = lastName
        self.email = email
        self.phoneNumber = phoneNumber
```

Lease Class :

```
class Lease:
    def __init__(self, leaseID, carID, customerID, startDate, endDate, leaseType):
        self.leaseID = leaseID
        self.carID = carID
        self.customerID = customerID
        self.startDate = startDate
        self.endDate = endDate
        self.leaseType = leaseType
```

Payment class :

```
class Payment:
    def __init__(self, paymentID=None, leaseID=None, paymentDate=None, amount=None):
        self.paymentID = paymentID
        self.leaseID = leaseID
        self.paymentDate = paymentDate
        self.amount = amount
```

Develop DAO Layer:

- Interface ICarLeaseRepository is defined with all required methods.

```python
from abc import ABC, abstractmethod

class ICarLeaseRepository(ABC):
    @abstractmethod
    def addCar(self, car): pass
    @abstractmethod
    def removeCar(self, carID): pass
    @abstractmethod
    def listAvailableCars(self): pass
    @abstractmethod
    def listRentedCars(self): pass
    @abstractmethod
    def findCarById(self, carID): pass

    @abstractmethod
    def addCustomer(self, customer): pass
    @abstractmethod
    def removeCustomer(self, customerID): pass
    @abstractmethod
    def listCustomers(self): pass
    @abstractmethod
    def findCustomerById(self, customerID): pass

    @abstractmethod
    def createLease(self, customerID, carID, startDate, endDate): pass
    @abstractmethod
    def returnCar(self, leaseID): pass
    @abstractmethod
    def listActiveLeases(self): pass
    @abstractmethod
    def listLeaseHistory(self): pass

    @abstractmethod
    def recordPayment(self, leaseID, amount): pass
```

- Implementation class CarLeaseRepositoryImpl performs all SQL operations.

```python
class CarLeaseRepositoryImpl(ICarLeaseRepository):
    def __init__(self):
        self.conn = getConnection()
        self.cursor = self.conn.cursor()

    def addCar(self, Vehicle):
        sql = "INSERT INTO Vehicle (make, model, year, dailyRate, status, passengerCapacity, engineCapacity)" \
        " VALUES (%s,%s,%s,%s,%s,%s,%s)"
        val = (car.make, car.model, car.year, car.dailyRate, car.status, car.passengerCapacity, car.engineCapacity)
        self.cursor.execute(sql, val)
        self.conn.commit()

    def removeCar(self, carID):
        self.cursor.execute("DELETE FROM Vehicle WHERE vehicleID = %s", (carID,))
        self.conn.commit()

    def listAvailableCars(self):
        self.cursor.execute("SELECT * FROM Vehicle WHERE status = 1")
        return self.cursor.fetchall()

    def listRentedCars(self):
        self.cursor.execute("SELECT * FROM Vehicle WHERE status = 0")
        return self.cursor.fetchall()

    def findCarById(self, carID):
        self.cursor.execute("SELECT * FROM Vehicle WHERE vehicleID = %s", (carID,))
        result = self.cursor.fetchone()
        if not result:
            raise CarNotFoundException("Car ID not found!")
        return result

    def addCustomer(self, customer):
        sql = "INSERT INTO Customer (firstName, lastName, email, phoneNumber) VALUES (%s, %s, %s, %s)"
        val = (customer.firstName, customer.lastName, customer.email, customer.phoneNumber)
        self.cursor.execute(sql, val)
        self.conn.commit()
```

```python
    def listCustomers(self):
        self.cursor.execute("SELECT * FROM Customer")
        return self.cursor.fetchall()

    def findCustomerById(self, customerID):
        self.cursor.execute("SELECT * FROM Customer WHERE customerID = %s", (customerID,))
        result = self.cursor.fetchone()
        if not result:
            raise CustomerNotFoundException("Customer ID not found!")
        return result

    def createLease(self, customerID, carID, startDate, endDate):
        self.findCustomerById(customerID)
        self.findCarById(carID)
        sql = "INSERT INTO lease (carID, customerID, startDate, endDate, leaseType) VALUES (%s, %s, %s, %s, %s)"
        val = (carID, customerID, startDate, endDate, "Daily")
        self.cursor.execute(sql, val)
        self.conn.commit()
        self.cursor.execute("UPDATE vehicle SET status=%s WHERE vehicleID = %s", (STATUS_NOT_AVAILABLE, carID))
        self.conn.commit()

        self.cursor.execute("SELECT * FROM Lease ORDER BY leaseID DESC LIMIT 1")
        result = self.cursor.fetchone()
        return Lease(*result)

    def returnCar(self, leaseID):
        self.cursor.execute("SELECT * FROM Lease WHERE leaseID = %s", (leaseID,))
        lease = self.cursor.fetchone()
        if not lease:
            raise LeaseNotFoundException("Lease ID not found!")
        self.cursor.execute("UPDATE vehicle SET status=%s WHERE vehicleID = %s", (STATUS_AVAILABLE, lease[1]))
        self.conn.commit()
        return Lease(*lease)
```

```
def listActiveLeases(self):
    today = date.today()
    self.cursor.execute("SELECT * FROM Lease WHERE endDate > %s", (today,))
    return self.cursor.fetchall()

def listLeaseHistory(self):
    self.cursor.execute("SELECT * FROM Lease")
    return self.cursor.fetchall()

def recordPayment(self, leaseID, amount):
    today = date.today()
    sql = "INSERT INTO Payment (leaseID, paymentDate, amount) VALUES (%s, %s, %s)"
    self.cursor.execute(sql, (leaseID, today, amount))
    self.conn.commit()
```

Add Utility Classes:

- PropertyUtil reads database config from a file.
- DBConnection uses the config to establish a live connection to MySQL.

Define Exception Classes:

- VehicleNotFoundException, CustomerNotFoundException, LeaseNotFoundException are created for structured error handling.

```
class CarNotFoundException(Exception):pass

class CustomerNotFoundException(Exception): pass

class LeaseNotFoundException(Exception): pass
```

Build Main Application:

- The user interacts with the system using a menu-driven console in MainModule.py.

```python
from dao.CarLeaseRepositoryImpl import CarLeaseRepositoryImpl
from entity.Car import Car
from entity.Customer import Customer
from datetime import datetime
import sys

STATUS_AVAILABLE = 'available'
STATUS_NOT_AVAILABLE = 'notAvailable'
repo = CarLeaseRepositoryImpl()

def get_date_input(prompt):
    while True:
        try:
            return datetime.strptime(input(prompt + " (YYYY-MM-DD): "), "%Y-%m-%d").date()
        except ValueError:
            print(" Invalid date format. Please enter in YYYY-MM-DD.")

while True:
    print("\n--- CAR RENTAL SYSTEM ---")
    print("1. Add Car")
    print("2. Add Customer")
    print("3. Create Lease")
    print("4. List Available Cars")
    print("5. Find Car by ID")
    print("6. List All Customers")
    print("7. Find Customer by ID")
    print("8. Exit")

    try:
        choice = int(input("Enter choice: "))
    except ValueError:
        print(" Invalid input. Enter a number.")
        continue
```

```python
if choice == 1:
    print("\n--- Add Car ---")
    make = input("Enter make: ")
    model = input("Enter model: ")
    year = int(input("Enter year: "))
    rate = float(input("Enter daily rate: "))
    status = input("Enter status (available / notAvailable): ").strip()
    passenger_capacity = int(input("Enter passenger capacity: "))
    engine_capacity = float(input("Enter engine capacity (in cc): "))

    car = Car(None, make, model, year, rate, status, passenger_capacity, engine_capacity)
    repo.addCar(car)
    print(" Car added.")

elif choice == 2:
    print("\n--- Add Customer ---")
    fname = input("Enter first name: ")
    lname = input("Enter last name: ")
    email = input("Enter email: ")
    phone = input("Enter phone number: ")

    customer = Customer(None, fname, lname, email, phone)
    repo.addCustomer(customer)
    print(" Customer added.")

elif choice == 3:
    print("\n--- Create Lease ---")
    try:
        cust_id = int(input("Enter customer ID: "))
        car_id = int(input("Enter car ID: "))
        start_date = get_date_input("Enter start date")
        end_date = get_date_input("Enter end date")

        lease = repo.createLease(cust_id, car_id, start_date, end_date)
        print(" Lease created:", lease.__dict__)
    except Exception as e:
        print(f" Failed to create lease: {e}")
```

```
elif choice == 4:
    print("\n--- Available Cars ---")
    try:
        cars = repo.listAvailableCars()
        if cars:
            for car in cars:
                print(car)
        else:
            print("No cars available.")
    except Exception as e:
        print(f" Error fetching available cars: {e}")

elif choice == 5:
    print("\n--- Find Car by ID ---")
    try:
        car_id = int(input("Enter Car ID: "))
        car = repo.findCarById(car_id)
        print("Car Found:", car)
    except Exception as e:
        print(f" {e}")

elif choice == 6:
    print("\n--- All Customers ---")
    try:
        customers = repo.listCustomers()
        if customers:
            for cust in customers:
                print(cust)
        else:
            print("No customers found.")
    except Exception as e:
        print(f" Error fetching customers: {e}")
```

```
elif choice == 7:
    print("\n--- Find Customer by ID ---")
    try:
        cust_id = int(input("Enter Customer ID: "))
        customer = repo.findCustomerById(cust_id)
        print("Customer Found:", customer)
    except Exception as e:
        print(f" {e}")

elif choice == 8:
    print(" Exiting... Goodbye!")
    sys.exit()

else:
    print(" Invalid choice. Try again.")
```

OUTPUT :

CHOICE 1 :

```
--- CAR RENTAL SYSTEM ---
1. Add Car
2. Add Customer
3. Create Lease
4. List Available Cars
5. Find Car by ID
6. List All Customers
7. Find Customer by ID
8. Exit
Enter choice: 1

--- Add Car ---
Enter make: BMW
Enter model: 3 series
Enter year: 2023
Enter daily rate: 60
Enter status (available / notAvailable): 1
Enter passenger capacity: 5
Enter engine capacity (in cc): 2499
Car added.
```

CHOICE 2 :

```
--- CAR RENTAL SYSTEM ---
1. Add Car
2. Add Customer
3. Create Lease
4. List Available Cars
5. Find Car by ID
6. List All Customers
7. Find Customer by ID
8. Exit
Enter choice: 2

--- Add Customer ---
Enter first name: Michael
Enter last name: Davis
Enter email: michael@example.com
Enter phone number: 555-873-8797
Customer added.
```

CHOICE 3 :

```
--- CAR RENTAL SYSTEM ---
1. Add Car
2. Add Customer
3. Create Lease
4. List Available Cars
5. Find Car by ID
6. List All Customers
7. Find Customer by ID
8. Exit
Enter choice: 3

--- Create Lease ---
Enter customer ID: 7
Enter car ID: 4
Enter start date (YYYY-MM-DD): 2023-07-01
Enter end date (YYYY-MM-DD): 2023-07-10
Failed to create lease: Customer ID not found!
```

CHOICE 4 :

```
--- CAR RENTAL SYSTEM ---
1. Add Car
2. Add Customer
3. Create Lease
4. List Available Cars
5. Find Car by ID
6. List All Customers
7. Find Customer by ID
8. Exit
Enter choice: 4

--- Available Cars ---
(2, 'Honda', 'Civic', 2023, Decimal('45.00'), 1, 7, 1500)
(4, 'Nissan', 'Altima', 2023, Decimal('52.00'), 1, 7, 1200)
(5, 'Chevrolet', 'Malibu', 2022, Decimal('47.00'), 1, 4, 1800)
(24, 'TestMake', 'TestModel', 2024, Decimal('3000.00'), 1, 5, 1500)
(25, 'TestMake', 'TestModel', 2024, Decimal('3000.00'), 1, 5, 1500)
(26, 'TestMake', 'TestModel', 2024, Decimal('3000.00'), 1, 5, 1500)
(27, 'TestMake', 'TestModel', 2024, Decimal('3000.00'), 1, 5, 1500)
(28, 'TestMake', 'TestModel', 2024, Decimal('3000.00'), 1, 5, 1500)
(29, 'BMW', '3 series', 2023, Decimal('60.00'), 1, 5, 2499)
```

CHOICE 5 :

```
--- CAR RENTAL SYSTEM ---
1. Add Car
2. Add Customer
3. Create Lease
4. List Available Cars
5. Find Car by ID
6. List All Customers
7. Find Customer by ID
8. Exit
Enter choice: 5

--- Find Car by ID ---
Enter Car ID: 2
Car Found: (2, 'Honda', 'Civic', 2023, Decimal('45.00'), 1, 7, 1500)
```

CHOICE 6 :

```
--- CAR RENTAL SYSTEM ---
1. Add Car
2. Add Customer
3. Create Lease
4. List Available Cars
5. Find Car by ID
6. List All Customers
7. Find Customer by ID
8. Exit
Enter choice: 6

--- All Customers ---
(1, 'John', 'Doe', 'johndoe@example.com', '555-555-5555')
(2, 'Jane', 'Smith', 'janesmith@example.com', '555-123-4567')
(3, 'Robert', 'Johnson', 'robert@example.com', '555-789-1234')
(4, 'Sarah', 'Brown', 'sarah@example.com', '555-456-7890')
(5, 'David', 'Lee', 'david@example.com', '555-987-6543')
(20, 'Michael', 'Davis', 'michael@example.com', '555-873-8797')
```

CHOICE 7 :

```
--- CAR RENTAL SYSTEM ---
1. Add Car
2. Add Customer
3. Create Lease
4. List Available Cars
5. Find Car by ID
6. List All Customers
7. Find Customer by ID
8. Exit
Enter choice: 7

--- Find Customer by ID ---
Enter Customer ID: 4
Customer Found: (4, 'Sarah', 'Brown', 'sarah@example.com', '555-456-7890')
```

CHOICE 8 :

```
--- CAR RENTAL SYSTEM ---
1. Add Car
2. Add Customer
3. Create Lease
4. List Available Cars
5. Find Car by ID
6. List All Customers
7. Find Customer by ID
8. Exit
Enter choice: 8
Exiting... Goodbye!
PS C:\Users\murug\Downloads\car_rental_system - Final\car_rental_system - Final\car_rental_system>
```

SOFTWARE TESTING:

```python
import unittest
from datetime import date
from dao.CarLeaseRepositoryImpl import CarLeaseRepositoryImpl
from entity.Car import Car
from entity.Customer import Customer
from myexceptions.CarNotFoundException import CarNotFoundException
from myexceptions.CustomerNotFoundException import CustomerNotFoundException
from myexceptions.LeaseNotFoundException import LeaseNotFoundException

STATUS_AVAILABLE = 1
STATUS_NOT_AVAILABLE = 0
class TestCarRentalSystem(unittest.TestCase):

    def setUp(self):
        self.repo = CarLeaseRepositoryImpl()


    def test_add_car(self):
        car = Car(None, "TestMake", "TestModel", 2024, 3000, STATUS_AVAILABLE, 5, 1500)
        self.repo.addCar(car)
        cars = self.repo.listAvailableCars()
        #print("Available cars:",cars)
        self.assertTrue(any(c[1] == "TestMake" for c in cars))

    def test_create_lease(self):
        lease = self.repo.createLease(1, 1, date(2025, 6, 21), date(2025, 6, 25))
        self.assertIsNotNone(lease)
        self.assertEqual(lease.carID, 1)
        self.assertEqual(lease.customerID, 1)

    def test_list_active_leases(self):
        leases = self.repo.listActiveLeases()
        self.assertIsInstance(leases, list)
        self.assertTrue(len(leases) >= 0)
```

```python
    def test_find_car_not_found_exception(self):
        with self.assertRaises(CarNotFoundException):
            self.repo.findCarById(99999)

    def test_find_customer_not_found_exception(self):
        with self.assertRaises(CustomerNotFoundException):
            self.repo.findCustomerById(99999)

    def test_return_car_lease_not_found_exception(self):
        with self.assertRaises(LeaseNotFoundException):
            self.repo.returnCar(99999)

if __name__ == '__main__':
    unittest.main()
```

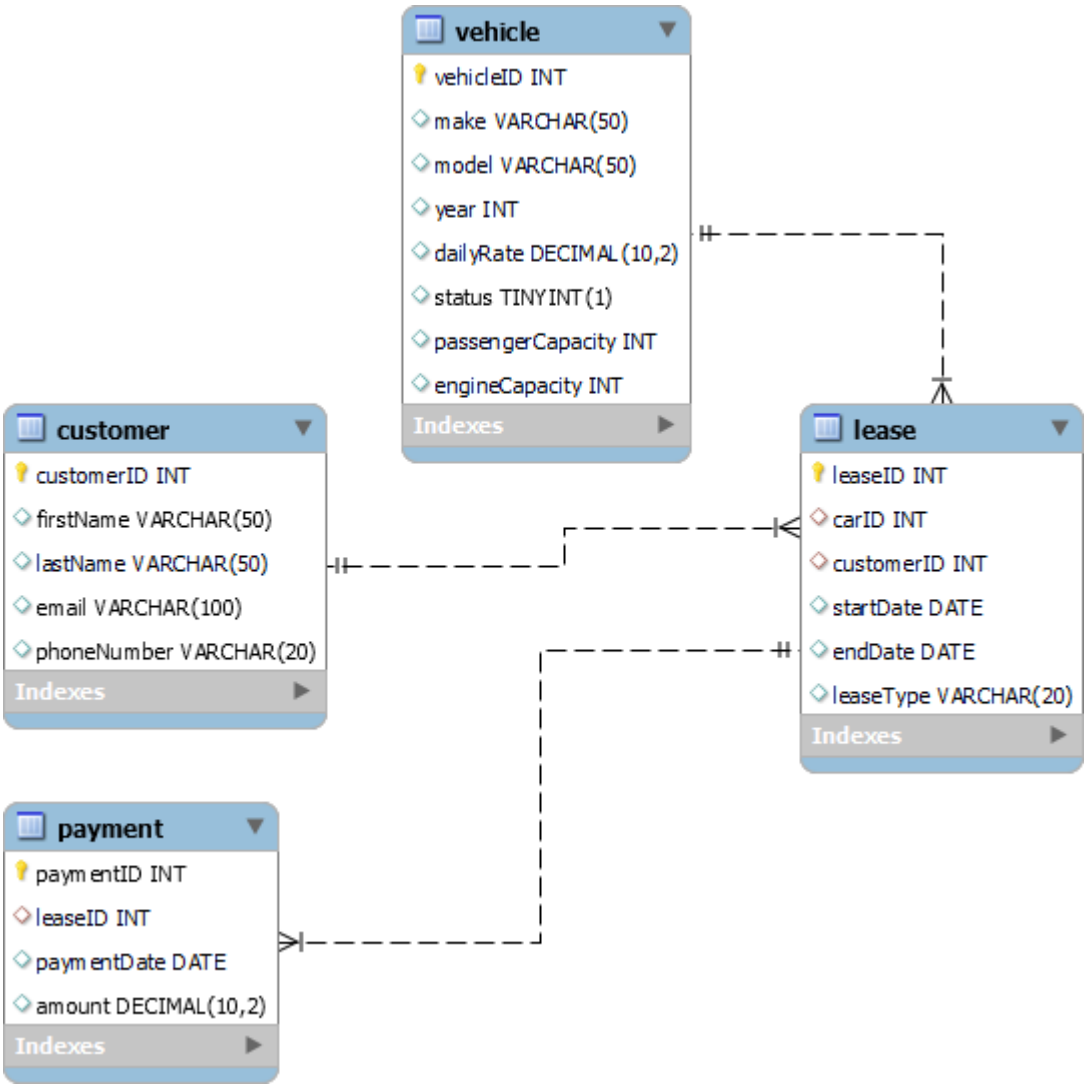
OUTPUT :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\murug\Downloads\car_rental_system - Final\car_rental_system - Final
>>
......
----------------------------------------------------------------------
Ran 6 tests in 0.117s

OK
PS C:\Users\murug\Downloads\car_rental_system - Final\car_rental_system - Final
```

ER - DIAGRAM:



CONCLUSION:

The development of the Car Rental System marks a significant achievement in simplifying and automating the traditional vehicle rental process. By leveraging object-oriented programming principles and layered architecture, the system delivers a user-friendly and robust platform that efficiently handles customer registration, vehicle inventory management, lease transactions, and payment tracking. The thoughtful integration of a MySQL database ensures data consistency, reliability, and security across all transactions.

Throughout the implementation, the modular design approach facilitated separation of concerns, making the system scalable and easy to maintain. The use of packages like entity, dao, util, exception, and main helped organize the codebase effectively, supporting better readability and debugging. Additionally, custom exceptions enhanced error handling, contributing to a seamless user experience and improved system stability during edge-case operations.

One of the strengths of the system is its extensibility. While the current version focuses on core functionalities-such as customer booking, vehicle availability, lease management, and payment processing-it lays the foundation for future enhancements. Features like online booking, driver assignment, vehicle condition tracking, or integration with GPS systems could be added to improve operational efficiency and provide a more comprehensive rental solution.

The system also adheres to good programming practices such as encapsulation, abstraction, and interface-driven development. By using interface classes to define the contract between the DAO and service layers, the system remains adaptable to technology changes. For example, the backend could later transition from a MySQL database to a cloud-based NoSQL database without rewriting business logic or presentation components extensively.

From an academic and industry perspective, this Car Rental System serves as an ideal case study of how software engineering principles and database-driven applications can be applied to solve real-world problems. It not only emphasizes technical implementation but also showcases the importance of system planning, user interaction design, and error management in delivering a successful software solution.

In conclusion, the Car Rental System is a practical, scalable, and modular application that effectively meets the core needs of a vehicle rental business. With minor enhancements and interface redesigns, it holds strong potential for deployment in actual commercial environments, paving the way for automation in transportation services.