++++A Project Report

On

PARKING MANAGEMENT SYSTEM

Submitted in partial fulfillment of the

BACHELOR OF COMPUTER APPLICATION

By

Md Zishan Zia

AJU/221354

Under the esteemed guidance of

Miss Moushmi

&

Dr. Arvind Kumar Pandey

Dean

School of Engineering and IT

DEPARTMENT OF COMPUTER SCIENCE & IT

ARKA JAIN UNIVERSITY, JHARKHAND

2022-2025

A PROJECT REPORT ON

PARKING MANAGEMENT SYSTEM

IN PARTIAL FULFILLMENT OF REQUIREMENT

OF

BACHELOR OF COMPUTER APPLICATION

BATCH 2022-2025

UNDER THE GUIDANCE OF:

DR. ARVIND KUMAR PANDEY
DEAN
SCHOOL OF ENGINEERING AND IT

PREPARED BY:

Md Zishan Zia

SUBMITTED TO

DEPARTMENT OF COMPUTER SCIENCE & IT

ARKA JAIN UNIVERSITY, JHARKHAND

ARKA JAIN University
Jharkhand

NAAC GRADE A
ACCREDITED UNIVERSITY

CERTIFICATE

This is to certify that the project entitled, **PARKING MANAGEMENT SYSTEM** is bonafide work of **Md Zishan Zia** bearing Enrollment No **AJU/221354** under the guidance of Dean, School of Engineering and IT, **Dr. Arvind Kumar Pandey** submitted in partial fulfillment of the requirements for the award of degree of BACHELOR OF COMPUTER APPLICATION from ARKA JAIN UNIVERSITY, JHARKHAND during the academic year 2024-2025.

Internal Guide
School of Engineering & IT

Dr. Arvind Kumar Pandey
Dean
ARKA JAIN UNIVERSITY, Jharkhand

University seal

Date:

COMPANY INTERNSHIP CERTIFICATE

ARKA JAIN University Jharkhand

NAAC GRADE A ACCREDITED UNIVERSITY

ABSTRACT

The **Parking Management System** is a comprehensive web-based application designed to automate and streamline parking operations for various facilities, including commercial lots, residential complexes, and public parking spaces. The system aims to enhance efficiency, improve user experience, reduce operational overhead, and maximize revenue generation. The project leverages a mordern technology stack (MongoDB, Node.js, Express.js, React.js) to deliver a scalable, secure, and user-friendly solution for parking administrators and customers alike. The primary goal of the system is to provide real-time visibility into parking space availability, simplify the parking process for users, and offer robust management tools for administrators. By moving away from manual processes, the system intends to minimize errors, reduce wait times, and optimize the utilization of parking resources. The backend of the system utilizes MongoDB to provide a flexible and scalable database to store information on parking spots, vehicles, customers, transactions, and system users. The frontend, built with React.js, offers intuitive interfaces for both end-users and administrators. Registered customers can easily search for available parking spots based on location and vehicle type, make online reservations, manage their vehicle profiles, and securely process payments. The system incorporates a secure login system for customers, providing them the ability to manage their accounts and view their parking history. The administrator dashboard offers a centralized platform where parking managers can view real-time parking occupancy, manage parking spot details (including rates and availability), handle reservations, monitor transactions, generate comprehensive reports on occupancy rates and revenue, and manage system users. The system provides functionalities for dynamic pricing, integration with payment gateways, and potential integration with external navigation systems. Furthermore, the application includes a robust reporting system, offering insights into parking trends, peak hours, and revenue streams, facilitating data-driven decision-making. Designed with a responsive and intuitive user interface, the application ensures

accessibility across various devices, including desktops, tablets, and smartphones. A key advantage of this system is its scalability and maintainability, employing a modular architecture that allows for future enhancements and integrations.

Key Features:

1. **Customer User Interface:**

   * Real-time availability of parking spots on a map or list view.

   * Advanced search filters based on location, spot type, and availability.

   * Option to make, modify, and cancel parking reservations.

   * Secure online payment integration with multiple payment options.

   * User profile management for saved vehicles and payment methods.

   * Push notifications and SMS alerts for reservation confirmations and reminders.

2. **Administrator Staff Interface:**

   * Real-time dashboard displaying parking occupancy and key metrics.

   * Comprehensive management of parking spots, including adding, editing, and deactivating spots.

   * Tools for managing parking rates and implementing dynamic pricing strategies.

   * Functionality to view, manage, and cancel reservations.

   * Secure management of customer accounts and vehicle information.

   * Generation of detailed reports on occupancy, revenue, and transaction history.

   * User management for adding, editing, and deactivating administrator accounts.

3. **System Architecture & Technologies:**

* MongoDB: Provides a flexible NoSQL database for storing diverse data types.

* Node.js and Express.js: Offer a robust and scalable backend framework for handling API requests and business logic.

* React.js: Enables the development of a dynamic, responsive, and user-friendly frontend interface that works seamlessly across devices.

* Secure API design and implementation to ensure data integrity and security.

* Cloud-based deployment for high availability and scalability.

JGi | ARKA JAIN University Jharkhand | NAAC GRADE A ACCREDITED UNIVERSITY

ACKNOWLEDGEMENT

After completion of my final year project, I would like to take this chance to express my sincere gratitude to my project guide and Dean, School of Engineering and IT, **Dr. Arvind Kumar Pandey** who has guided me a lot throughout my project development. Without him, I think I could not have finished the project on time. In addition, while I met some logic problem or design problem, he was always the one who gave me useful and logical answers.

I would like to thank **Miss Moushmi** for one more time for sharing his experience with me so that I could get more logical understanding on how to develop chat application which is suitable for current society.

Finally, I want to thank to all my friends and teachers, who helped and co-operated with me directly or indirectly in the accomplishment of this project.

.

JGi | ARKA JAIN University Jharkhand | NAAC GRADE A ACCREDITED UNIVERSITY

DECLARATION

I **Md Zishan Zia** hereby declare that the project entitled, **PARKING MANAGEMENT SYSTEM** done at **ARKA JAIN UNIVERSITY**, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfillment of the requirements for the award of degree of BACHLEOR OF COMPUTER APPLICATION to be submitted as final semester project as part of our curriculum.

Md Zishan Zia

AJU/221354

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Overview

This project focuses on developing a simplified Parking Management System using Python. The system will be terminal-based, providing a command-line interface for managing parking slots. This system aims to demonstrate the core functionalities of a parking management system, including tracking parking availability, vehicle entry and exit, and basic fee calculation**.**

1.2 OBJECTIVE

1.2 Objectives and Features

The objectives of this simplified system are to:

 * Track the availability of parking slots.
 * Record vehicle entry and exit.
 * Calculate parking fees based on time.
 * Provide a user-friendly terminal interface.

Key features include:

 * Slot Management: Adding and removing parking slots.
 * Vehicle Tracking: Recording vehicle entry and exit times.
 * Fee Calculation: Calculating parking fees based on the duration of stay.
 * Reporting: Displaying available slots and vehicle information.

Chapter 2

REQUIREMENTS AND ANALYSIS

2.1 Software Requirement Specification

*** Functional Requirements:**

* The system shall allow the user to add parking slots.

* The system shall allow the user to mark a slot as occupied upon vehicle entry.

* The system shall record the entry time of a vehicle. * The system shall display the availability of parking

* The system shall allow the user to mark a slot as vacant upon vehicle exit.

* The system shall record the exit time of a vehicle.

* The system shall calculate the parking fee based on the entry and exit time.

slots.

*** Non-Functional Requirements**:

* The system shall be easy to use through a terminal interface.

* The system shall provide clear and concise output.

2.2 Data Gathering

For this simplified system, the requirements were gathered through:

* Basic understanding of parking lot operations: Identifying the core functions of entry, exit, and fee calculation.

*** Focus on terminal-based interaction: Prioritizing command-line usability.**

2.3 Feasibility Study

*** Technical Feasibility**: Python is well-suited for this project due to its simplicity and ease of use for developing terminal applications**.**

*** Economic Feasibility**: The system is developed using open-source technologies (Python), minimizing costs**.**

*** Operational Feasibility:** The terminal-based interface is straightforward for parking attendants to use with minimal training**.**

*** Schedule Feasibility:** The system can be developed within a short timeframe due to its limited scope.

2.4 Hardware Requirements

***** A computer with a terminal or command-line interface.

2.5 Software Requirements

* Python 3.x

2.6 Justification of Selection of Technology

2.6.1 (USER INTERFACE)

* The terminal interface was chosen for its simplicity and ease of implementation in Python. It provides a basic but functional way to interact with the system.

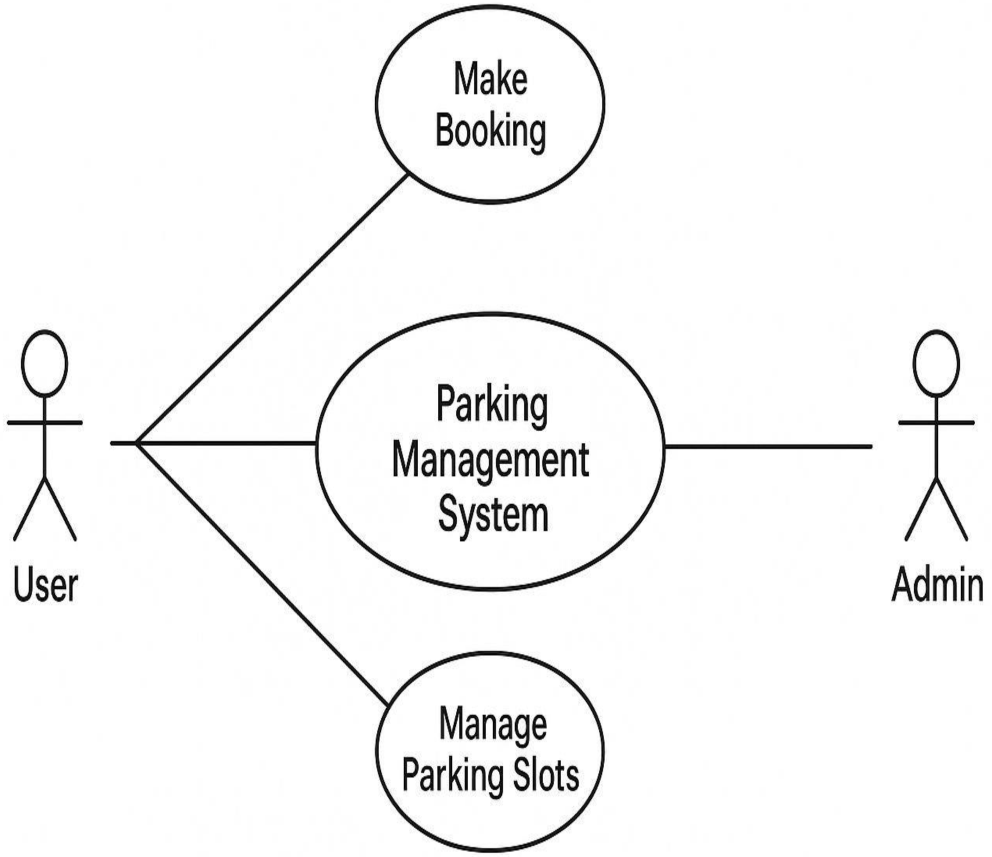
2.6.2 (SERVER-SIDE LOGIC)+

capabilities. It is ideal for creating the logic for managing parking slots, vehicles, and * Python was selected for its clear syntax, extensive libraries, and rapid development fees**.**

2.6.3 (DATABASE)

* For this simplified version, data is stored in Python data structures (lists and dictionaries) to avoid the complexity of setting up a separate database. This is suitable for demonstrating the core functionality

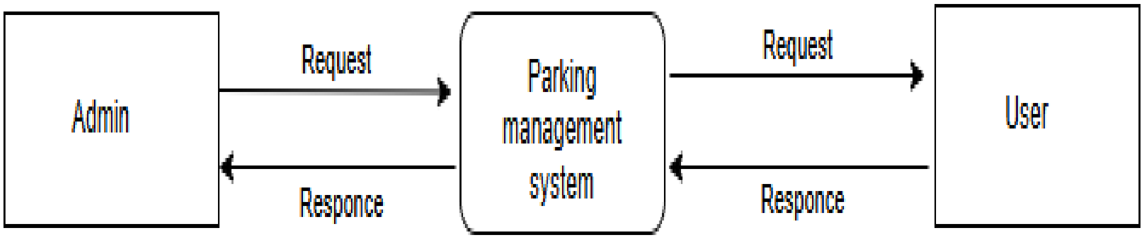
(Simplified Use Case Diagram - Imagine this as a simple diagram)



Parking Management System

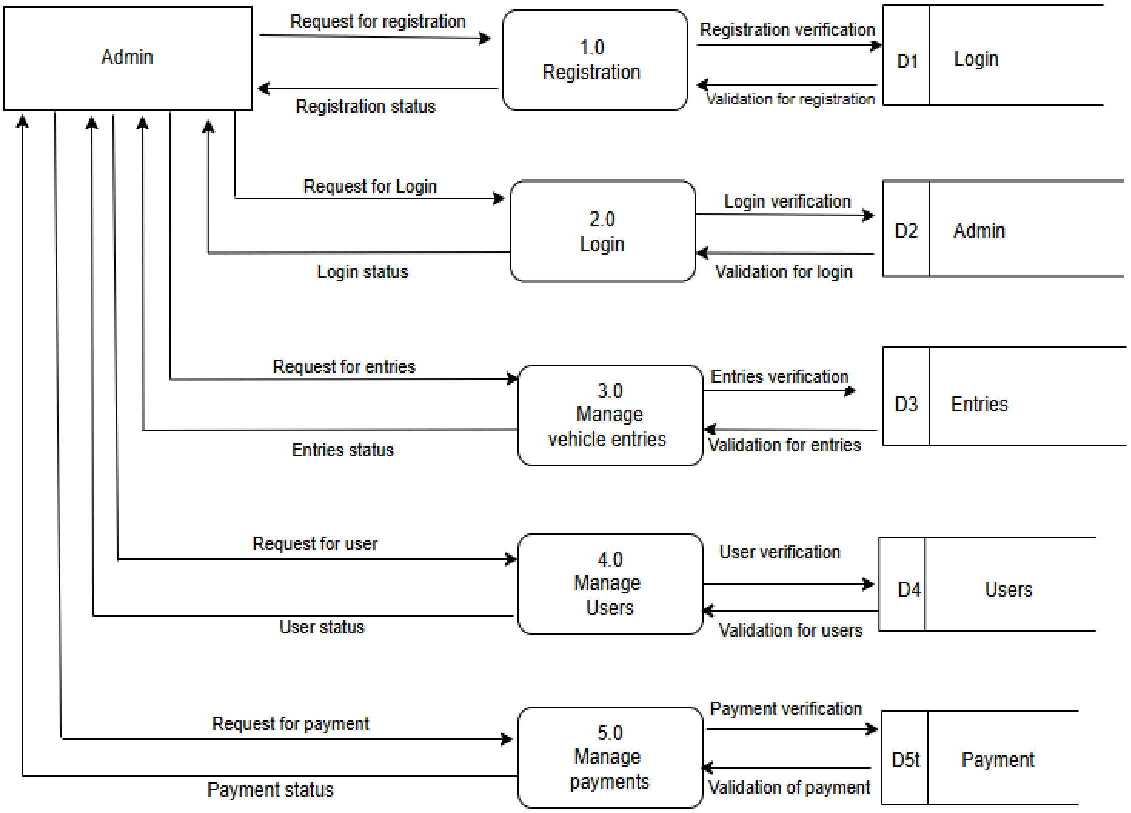
2.8 Data Flow Diagram (DFD)

2.8.1 Context level DFD – 0 level

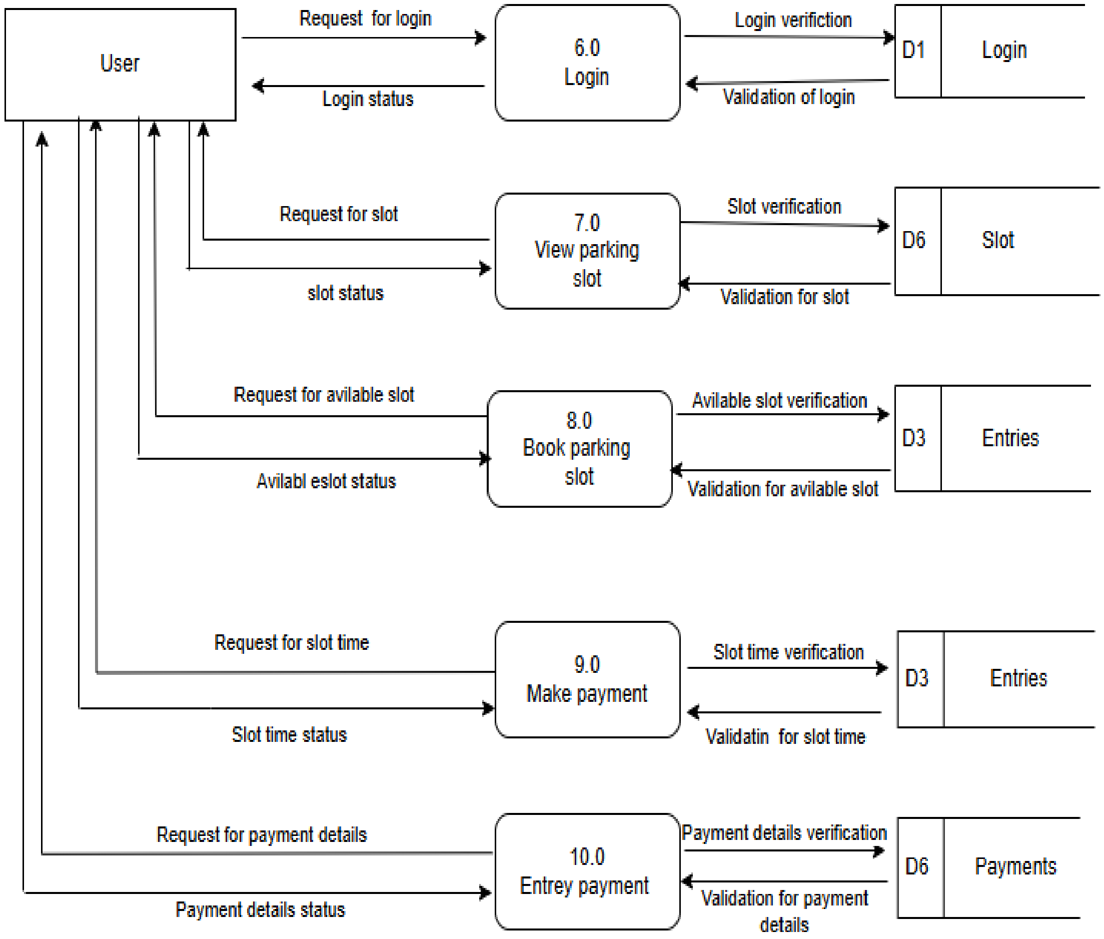


2.8.2 DFD level 1

DFD LEVEL 1 ADMIN

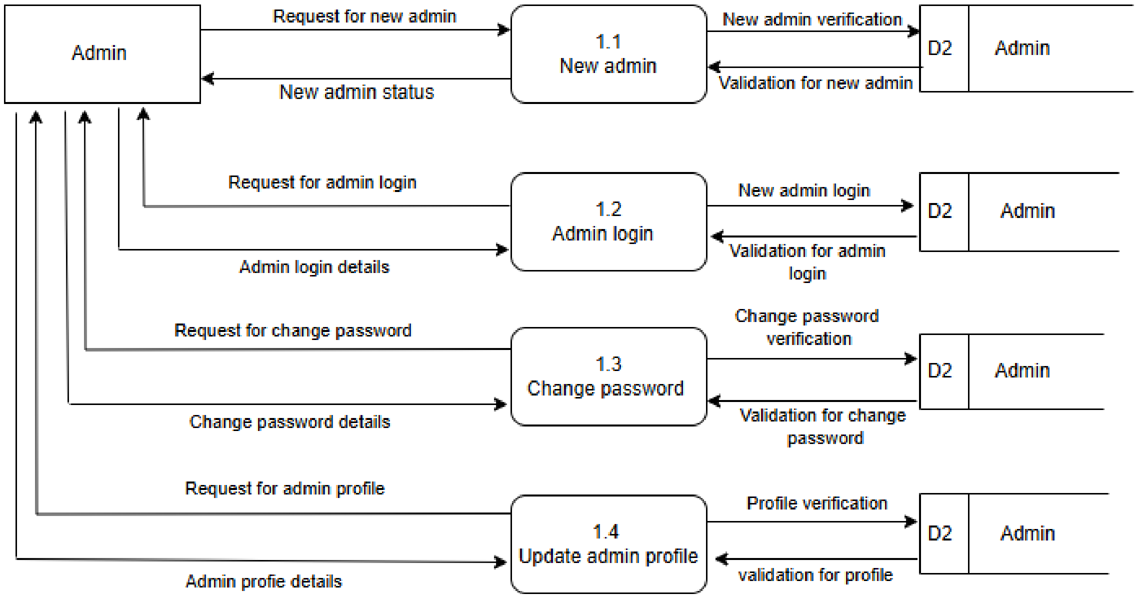


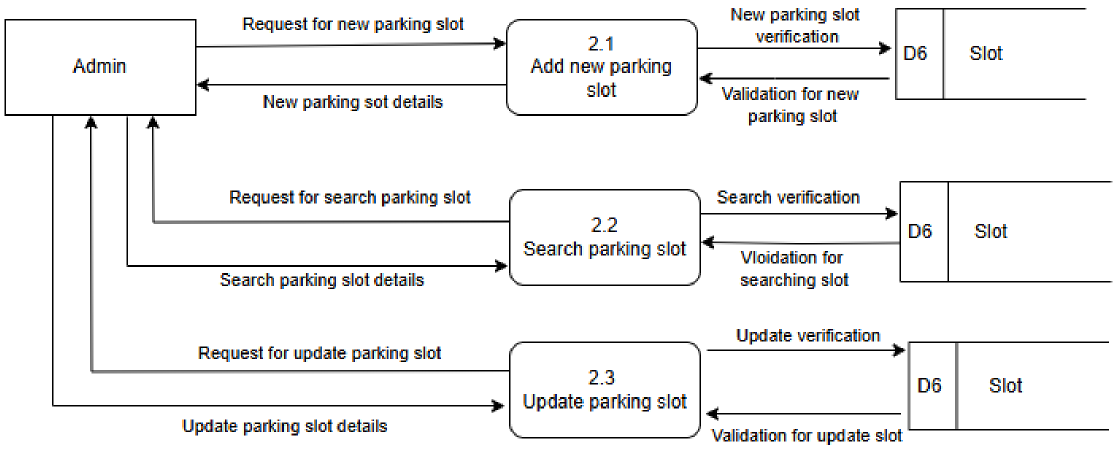
DFD LEVEL 1 - USER

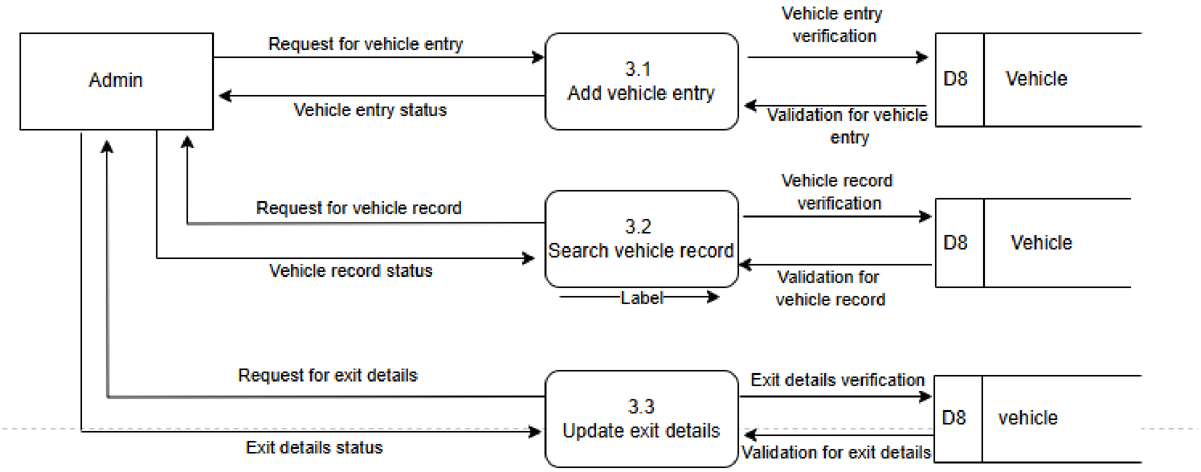


2.8.3 DFD level 2

DFD LEVEL 2 - ADMIN

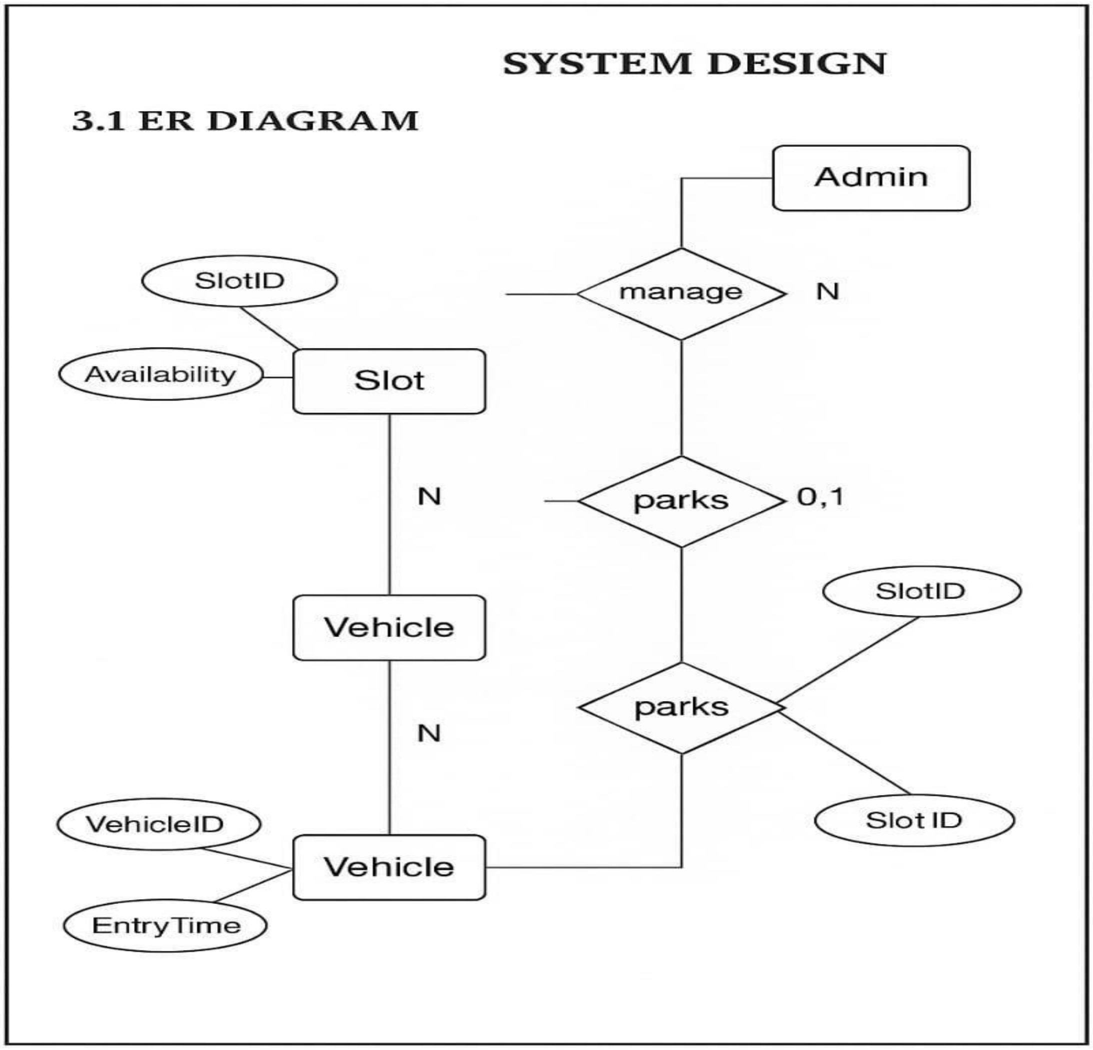



Admin
Request for new parking slot
New parking sot details
2.1
Add new parking slot
New parking slot verification
Validation for new parking slot
D6
Slot
Request for search parking slot
Search parking slot details
2.2
Search parking slot
Search verification
Vloidation for searching slot
D6
Slot
Request for update parking slot
Update parking slot details
2.3
Update parking slot
Update verification
Validation for update slot
D6
Slot


Admin
Request for vehicle entry
Vehicle entry status
3.1
Add vehicle entry
Vehicle entry verification
Validation for vehicle entry
D8
Vehicle
Request for vehicle record
Vehicle record status
3.2
Search vehicle record
Label
Vehicle record verification
Validation for vehicle record
D8
Vehicle
Request for exit details
Exit details status
3.3
Update exit details
Exit details verification
Validation for exit details
D8
vehicle

Chapter 3

3.1 ER DIAGRAM



3.2 DATA NORMALIZATION

The needed data for this Parking Management System are as follows:

Column	Type
Spot ID	Integer
Spot Number	Varchar
Spot Type	Varchar
Hourly Rate	Decimal
Is Occupied	Boolean
Vehicle ID	Integer
License Plate	Varchar
Vehicle Type	Varchar
Customer ID	Integer
Customer Name	Varchar
Customer Email	Varchar
Customer Phone	Varchar
Customer Address	Varchar
Entry Time	DateTime
Exit Time	DateTime
Duration (Minutes)	Integer
Amount Paid	Decimal
Payment Status	Varchar
Admin ID	Integer
Admin Username	Varchar
Admin Password	Varchar

After applying 1NF the result is:

- **Parking_Spot Table**

Columns	Type	Unique
spot_id	Integer	Yes
spot_number	Varchar	Yes
spot_type	Varchar	No
hourly_rate	Decimal	No
is_occupied	Boolean	No

- **Vehicle Table**

Columns	Type	Unique
vehicle_id	Integer	Yes
license_plate	Varchar	Yes
vehicle_type	Varchar	No
customer_id	Integer	No

- **Customer Table**

Columns	Type	Unique
customer_id	Integer	Yes
customer_name	Varchar	No
customer_email	Varchar	No
customer_phone	Varchar	No
customer_address	Varchar	No

- **Parking_Transaction Table**

Columns	Type	Unique
transaction_id	Integer	Yes
spot_id	Integer	No
vehicle_id	Integer	No
Entry _ time	Date Time	No

Columns	Type	Unique
exit_time	DateTime	No
duration_minutes	Integer	No
amount_paid	Decimal	No
payment_status	Varchar	No

- **Parking_Admin Table**

Columns	Type	Unique
admin_id	Integer	Yes
admin_username	Varchar	Yes
admin_password	Varchar	No

3.3 DATA DICTIONARY

Table: Parking_Admin

Attribute	Data Type	Description	Constraints
admin_id	Integer	Unique identifier of the admin	Primary Key, Not Null
Username	Varchar	Username for the admin	Not Null
Password	Varchar	Password for the admin	Not Null

Table: Parking_Spot

Attribute	Data Type	Description	Constraints
spot_id	Integer	Unique identifier of the parking spot	Primary Key, Not Null
spot_number	Varchar	Number or identifier of the spot	Not Null, Unique

Attribute	Data Type	Description	Constraints
spot_type	Varchar	Type of parking spot (e.g., Car, Bike, Truck)	Not Null
is_occupied	Boolean	Indicates if the spot is currently occupied (True/False)	Not Null
hourly_rate	Decimal (8, 2)	Hourly parking rate for this spot type	Not Null

Table: Vehicle

Attribute	Data Type	Description	Constraints
vehicle_id	Integer	Unique identifier of the vehicle	Primary Key, Not Null
license_plate	Varchar	Vehicle's license plate number	Not Null, Unique
vehicle_type	Varchar	Type of vehicle (e.g., Car, Bike, Truck)	Not Null
customer_id	Integer	Foreign key referencing the Customer table	Foreign Key (references Customer.customer_id), Not Null

Table: Customer

Attribute	Data Type	Description	Constraints
customer_id	Integer	Unique identifier of the customer	Primary Key, Not Null
Name	Varchar	Name of the customer	Not Null
Email	Varchar	Customer's email address	Not Null
phone_number	Varchar	Customer's contact phone number	Not Null
Address	Varchar	Customer's address	Nullable

Table: Parking_Transaction

Attribute	Data Type	Description	Constraints
transaction_id	Integer	Unique identifier of the parking transaction	Primary Key, Not Null
spot_id	Integer	Foreign key referencing the Parking_Spot table	Foreign Key (references Parking_Spot.spot_id), Not Null
vehicle_id	Integer	Foreign key referencing the Vehicle table	Foreign Key (references Vehicle.vehicle_id), Not Null
entry_time	DateTime	Date and time when the vehicle entered the parking spot	Not Null
exit_time	DateTime	Date and time when the vehicle exited the parking spot	Nullable
duration_minutes	Integer	Total parking duration in minutes	Nullable
amount_paid	Decimal (10, 2)	Total amount paid for parking	Nullable
payment_status	Varchar	Status of the payment (e.g., Paid, Pending)	Not Null

Chapter 4

PROGRAM CODE AND TESTING

4.1 CODING

```
"""


import datetime
import time
import uuid
import json
import os
from enum import Enum, auto

# --- Configuration ---
DATABASE_FILE = "parking_data.json"
FEE_STRUCTURE = {
    "car": {"hourly_rate": 20.0, "daily_cap": 150.0},
    "bike": {"hourly_rate": 10.0, "daily_cap": 75.0},
    "truck": {"hourly_rate": 30.0, "daily_cap": 250.0},
    "default": {"hourly_rate": 15.0, "daily_cap": 100.0},
}
REPORT_DIRECTORY = "parking_reports"
USER_DATABASE_FILE = "users.json"

# Ensure report directory exists
```

```
os.makedirs(REPORT_DIRECTORY, exist_ok=True)

# --- Enumerations ---
class VehicleType(Enum):
    CAR = "car"
    BIKE = "bike"
    TRUCK = "truck"
    OTHER = "other"

class ParkingSpotStatus(Enum):
    VACANT = "vacant"
    OCCUPIED = "occupied"
    RESERVED = "reserved"
    OUT_OF_SERVICE = "out_of_service"

class PaymentMethod(Enum):
    CASH = "cash"
    CARD = "card"
    UPI = "upi"
    ONLINE = "online"

class ParkingTicketStatus(Enum):
    ACTIVE = "active"
    PAID = "paid"
    LOST = "lost"

# --- Data Models ---
```

```
class ParkingSpot:
    def __init__(self, spot_id: str, spot_type: VehicleType, status:
ParkingSpotStatus = ParkingSpotStatus.VACANT):
        self.spot_id = spot_id
        self.spot_type = spot_type
        self.status = status
        self.current_vehicle = None
        self.reserved_for = None

    def to_dict(self):
        return {
            "spot_id": self.spot_id,
            "spot_type": self.spot_type.value,
            "status": self.status.value,
            "current_vehicle": self.current_vehicle,
            "reserved_for": self.reserved_for
        }

    @classmethod
    def from_dict(cls, data):
        spot = cls(
            spot_id=data["spot_id"],
            spot_type=VehicleType(data["spot_type"]),
            status=ParkingSpotStatus(data["status"])
        )
        spot.current_vehicle = data["current_vehicle"]
        spot.reserved_for = data["reserved_for"]
        return spot

class ParkingTicket:
    def __init__(self, ticket_id: str, spot_id: str, vehicle_number: str,
                 vehicle_type: VehicleType, entry_time: datetime.datetime):
        self.ticket_id = ticket_id
        self.spot_id = spot_id
        self.vehicle_number = vehicle_number
        self.vehicle_type = vehicle_type
        self.entry_time = entry_time
        self.exit_time = None
        self.amount_paid = 0.0
        self.payment_method = None
        self.status = ParkingTicketStatus.ACTIVE

    def calculate_fee(self) -> float:
        if self.exit_time is None:
            exit_time = datetime.datetime.now()
        else:
            exit_time = self.exit_time

        duration = exit_time - self.entry_time
        hours = duration.total_seconds() / 3600

        vehicle_type_str = self.vehicle_type.value
        fee_info = FEE_STRUCTURE.get(vehicle_type_str,
FEE_STRUCTURE["default"])
```

```
        fee = min(fee_info["hourly_rate"] * hours, fee_info["daily_cap"])
        return round(fee, 2)

    def to_dict(self):
        return {
            "ticket_id": self.ticket_id,
            "spot_id": self.spot_id,
            "vehicle_number": self.vehicle_number,
            "vehicle_type": self.vehicle_type.value,
            "entry_time": self.entry_time.isoformat(),
            "exit_time": self.exit_time.isoformat() if self.exit_time else None,
            "amount_paid": self.amount_paid,
            "payment_method": self.payment_method.value if
self.payment_method else None,
            "status": self.status.value
        }

    @classmethod
    def from_dict(cls, data):
        ticket = cls(
            ticket_id=data["ticket_id"],
            spot_id=data["spot_id"],
            vehicle_number=data["vehicle_number"],
            vehicle_type=VehicleType(data["vehicle_type"]),
            entry_time=datetime.datetime.fromisoformat(data["entry_time"])
        )
        if data["exit_time"]:
            ticket.exit_time =
datetime.datetime.fromisoformat(data["exit_time"])
        ticket.amount_paid = data["amount_paid"]
        if data["payment_method"]:
            ticket.payment_method =
PaymentMethod(data["payment_method"])
        ticket.status = ParkingTicketStatus(data["status"])
        return ticket

# --- Database Handler ---
class ParkingDatabase:
    def __init__(self, db_file: str = DATABASE_FILE):
        self.db_file = db_file
        self.spots = []
        self.tickets = []
        self.load_data()

    def load_data(self):
        try:
            if os.path.exists(self.db_file):
                with open(self.db_file, "r") as f:
                    data = json.load(f)
                    self.spots = [ParkingSpot.from_dict(spot) for spot in
data.get("spots", [])]
                    self.tickets = [ParkingTicket.from_dict(ticket) for ticket in
data.get("tickets", [])]
```

```
            else:
                self.spots = []
                self.tickets = []
        except Exception as e:
            print(f"Error loading database: {e}")
            self.spots = []
            self.tickets = []

    def save_data(self):
        try:
            data = {
                "spots": [spot.to_dict() for spot in self.spots],
                "tickets": [ticket.to_dict() for ticket in self.tickets]
            }
            with open(self.db_file, "w") as f:
                json.dump(data, f, indent=2)
        except Exception as e:
            print(f"Error saving database: {e}")

    def add_spot(self, spot: ParkingSpot):
        self.spots.append(spot)
        self.save_data()

    def add_ticket(self, ticket: ParkingTicket):
        self.tickets.append(ticket)
        self.save_data()
    def find_spot(self, spot_id: str) -> ParkingSpot:
        for spot in self.spots:
            if spot.spot_id == spot_id:
                return spot
        return None

    def find_ticket(self, ticket_id: str) -> ParkingTicket:
        for ticket in self.tickets:
            if ticket.ticket_id == ticket_id:
                return ticket
        return None

    def find_ticket_by_vehicle(self, vehicle_number: str) ->
ParkingTicket:
        for ticket in self.tickets:
            if ticket.vehicle_number == vehicle_number and ticket.status ==
ParkingTicketStatus.ACTIVE:
                return ticket
        return None

    def update_ticket(self, ticket: ParkingTicket):
        for i, t in enumerate(self.tickets):
            if t.ticket_id == ticket.ticket_id:
                self.tickets[i] = ticket
                self.save_data()
                return True
        return False
```

```
# --- Parking Manager ---
class ParkingManager:
    def __init__(self):
        self.db = ParkingDatabase()

    def initialize_parking_spots(self, spots_config):
        """Initialize parking spots based on configuration"""
        for spot_id, spot_type in spots_config.items():
            if not self.db.find_spot(spot_id):
                vehicle_type = VehicleType(spot_type)
                new_spot = ParkingSpot(spot_id, vehicle_type)
                self.db.add_spot(new_spot)

    def assign_parking_spot(self, vehicle_number: str, vehicle_type:
VehicleType) -> ParkingTicket:
        """Assign a parking spot to a vehicle and generate a ticket"""
        # Find available spot
        for spot in self.db.spots:
            if (spot.status == ParkingSpotStatus.VACANT and
                spot.spot_type == vehicle_type):

                # Create ticket
                ticket_id = str(uuid.uuid4())
                entry_time = datetime.datetime.now()
                ticket = ParkingTicket(ticket_id, spot.spot_id, vehicle_number,
vehicle_type, entry_time)
```

```
                # Update spot
                spot.status = ParkingSpotStatus.OCCUPIED
                spot.current_vehicle = vehicle_number

                # Save changes
                self.db.add_ticket(ticket)
                self.db.save_data()

                return ticket

        raise Exception("No available parking spots for this vehicle type")

    def release_parking_spot(self, ticket_id: str, payment_method:
PaymentMethod) -> float:
        """Release a parking spot and calculate the fee"""
        ticket = self.db.find_ticket(ticket_id)
        if not ticket or ticket.status != ParkingTicketStatus.ACTIVE:
            raise Exception("Invalid or inactive ticket")

        spot = self.db.find_spot(ticket.spot_id)
        if not spot:
            raise Exception("Parking spot not found")

        # Calculate fee
        ticket.exit_time = datetime.datetime.now()
        fee = ticket.calculate_fee()
        # Update ticket
        ticket.amount_paid = fee
        ticket.payment_method = payment_method
        ticket.status = ParkingTicketStatus.PAID

        # Update spot
        spot.status = ParkingSpotStatus.VACANT
        spot.current_vehicle = None

        # Save changes
        self.db.update_ticket(ticket)
        self.db.save_data()

        return fee

    def generate_report(self, report_type: str = "daily"):
        """Generate reports (daily, weekly, monthly)"""
        now = datetime.datetime.now()
        filename =
f"{REPORT_DIRECTORY}/{report_type}_report_{now.strftime('%Y%
m%d_%H%M%S')}.txt"

        try:
            with open(filename, "w") as f:
                if report_type == "daily":
                    f.write(f"Daily Parking Report - {now.date()}\n")
                    f.write("="*40 + "\n")
                    today_tickets = [t for t in self.db.tickets
                                   if t.entry_time.date() == now.date()]
                    f.write(f"Total vehicles today: {len(today_tickets)}\n")

                    total_revenue = sum(t.amount_paid for t in today_tickets if
t.status == ParkingTicketStatus.PAID)
                    f.write(f"Total revenue: ${total_revenue:.2f}\n")

                    # Vehicle type breakdown
                    f.write("\nVehicle Type Breakdown:\n")
                    type_counts = {}
                    for vt in VehicleType:
                        type_counts[vt.value] = len([t for t in today_tickets if
t.vehicle_type == vt])

                    for vt, count in type_counts.items():
                        f.write(f"{vt.capitalize()}: {count}\n")

                elif report_type == "occupancy":
                    f.write("Current Parking Occupancy Report\n")
                    f.write("="*40 + "\n")
                    total_spots = len(self.db.spots)
                    occupied_spots = len([s for s in self.db.spots if s.status ==
ParkingSpotStatus.OCCUPIED])
                    f.write(f"Occupancy Rate: {occupied_spots}/{total_spots}
({occupied_spots/total_spots:.1%})\n")
```

```
                    f.write("\nSpot Type Breakdown:\n")
                    for vt in VehicleType:
                        spots_of_type = [s for s in self.db.spots if s.spot_type ==
vt]
                        occupied_of_type = len([s for s in spots_of_type if s.status
== ParkingSpotStatus.OCCUPIED])
                        f.write(f"{vt.value.capitalize()}:
{occupied_of_type}/{len(spots_of_type)}\n")

                return filename
        except Exception as e:
            print(f"Error generating report: {e}")
            return None

# --- Main Application ---
def main():
    print("Parking Management System")
    print("=" * 30)

    # Initialize system
    manager = ParkingManager()

    # Sample parking spots configuration
    spots_config = {
        "A1": "car",
        "A2": "car",
    "A3": "car",
    "B1": "bike",
    "B2": "bike",
    "C1": "truck",
    "D1": "other"
}

manager.initialize_parking_spots(spots_config)

while True:
    print("\nMenu:")
    print("1. Park Vehicle")
    print("2. Exit Vehicle")
    print("3. Generate Report")
    print("4. Exit System")

    choice = input("Enter your choice: ")

    try:
        if choice == "1":
            # Park vehicle
            vehicle_number = input("Enter vehicle number: ")
            print("Vehicle types:")
            for i, vt in enumerate(VehicleType, 1):
                print(f"{i}. {vt.value.capitalize()}")

            vt_choice = int(input("Select vehicle type (1-4): ")) - 1
```

```
                vehicle_type = list(VehicleType)[vt_choice]

                ticket = manager.assign_parking_spot(vehicle_number, vehicle_type)
                print(f"Vehicle parked successfully. Ticket ID: {ticket.ticket_id}")
                print(f"Spot assigned: {ticket.spot_id}")
                print(f"Entry time: {ticket.entry_time}")

            elif choice == "2":
                # Exit vehicle
                ticket_id = input("Enter ticket ID: ")
                print("Payment methods:")
                for i, pm in enumerate(PaymentMethod, 1):
                    print(f"{i}. {pm.value.capitalize()}")

                pm_choice = int(input("Select payment method (1-4): ")) - 1
                payment_method = list(PaymentMethod)[pm_choice]

                fee = manager.release_parking_spot(ticket_id, payment_method)
                print(f"Payment successful. Amount paid: ${fee:.2f}")

            elif choice == "3":
                # Generate report
                print("Report types:")
                print("1. Daily report")
                print("2. Occupancy report")
                report_choice = input("Select report type (1-2): ")

                if report_choice == "1":
                    report_file = manager.generate_report("daily")
                elif report_choice == "2":
                    report_file = manager.generate_report("occupancy")
                else:
                    print("Invalid choice")
                    continue

                if report_file:
                    print(f"Report generated: {report_file}")

            elif choice == "4":
                print("Exiting system...")
                break

            else:
                print("Invalid choice. Please try again.")

        except Exception as e:
            print(f"Error: {e}")

if __name__ == "__main__":
    main()
```

4.2 TESTING APPROACH

The following testing approach was used:

* Unit Testing: Individual functions within the ParkingSystem class were tested. For example:

  * Testing find_available_slot() to ensure it correctly identifies available slots.

  * Testing calculate_fee() with different durations to verify fee calculation.

* Integration Testing: Testing the interaction between different functions. For example:

  * Testing park_vehicle() followed by unpark_vehicle() to ensure vehicle parking and unparking are correctly handled.

  * Testing add_slot() and remove_slot() to check if slot management works as expected.

* System Testing: Testing the entire system through the terminal interface. This involves manually executing different scenarios:

  * Adding and removing slots.

  * Parking and unparking vehicles.

  * Checking available slots.

  * Verifying fee calculations.

  * Handling invalid inputs (e.g., trying to unpark a non-existent vehicle).

Chapter 5

RESULTS AND DISCUSSION

5.1 OUTPUT SCREENS

(Example Terminal Output)

```
Enter the total number of parking slots: 5



Parking Management System

1. Add Slot

2. Remove Slot

3. Park Vehicle

4. Unpark Vehicle

5. Display Available Slots

6. Display Parked Vehicles

7. Exit

Enter your choice: 3

Enter vehicle ID: ABC-123
```

```
 vehicle ABC-123 parked in Slot 1


Parking Management System

...

Enter your choice: 5

Available Slots:

Slot 2

Slot 3

Slot 4

Slot 5


Parking Management System

...

Enter your choice: 4

Enter vehicle ID to unpark: ABC-123
```

```
Vehicle ABC-123 unparked from Slot 1. Parking fee: $10.00



Parking Management System

...

Enter your choice: 7

Exiting...
```

5.2 LIMITATION* Simplified Data Storage: Uses in-memory data structures (lists, dictionaries) instead of a persistent database. Data is lost when the program terminates.

* Basic Terminal Interface: Lacks a graphical user interface (GUI).

* Limited Features: Does not include advanced features like reservation systems, payment processing integration, or real-time sensor data.

* No Error Handling: Basic error handling is implemented, but it could be more robust.

5.3 FUTURE SCOPE

* Database Integration: Implement a database (e.g., SQLite, MySQL) to store data persistently.

* GUI Development: Develop a graphical user interface using a library like Tkinter or PyQt.

* Advanced Features: Add features such as:

  * Parking reservation system.

  * Payment gateway integration.

  * License plate recognition.

  * Integration with parking sensors.

* Improved Error Handling: Implement more comprehensive error handling and input validation.

* Multi-user Support: Allow multiple users (e.g., parking attendants) to interact with the system concurrently.

Chapter 6

CONCLUSION

This project demonstrates the fundamental principles of a Parking Management System using Python and a terminal-based interface. While it is a simplified version, it illustrates the core functionalities of slot management, vehicle tracking, and fee calculation. The system provides a foundation for future development and the addition of more advanced features**.**

Chapter 7

REFERENCES

***** Python Documentation: https://docs.python.org/3/

* (Any other resources you used)