RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI
ENGINEERING COLLEGE

CB23332
SOFTWARE ENGINEERING LAB

Laboratory Record Note Book

Name : . . . . . .

Year / Branch / Section : . . . . . .

Register No. : . . . . . .

Semester : . . . . . .

Academic Year : . . . . . .

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

RAJALAKSHMI NAGAR, THANDALAM – 602-105

BONAFIDE CERTIFICATE

NAME:_____**REGISTER NO.:** _____

ACADEMIC YEAR: 2024-25 **SEMESTER:** III **BRANCH:**_____B.E/B.Tech

This Certification is the bonafide record of work done by the above student in the

CB23332-SOFTWARE ENGINEERING - Laboratory during the year 2024 – 2025.

Signature of Faculty -in – Charge

Submitted for the Practical Examination held on _____

Internal Examiner

External Examiner

INDEX

S.No.	Name of the Experiment	Expt. Date	Faculty Sign
1.	Preparing Problem Statement		
2.	Software Requirement Specification (SRS)		
3.	Entity-Relational Diagram		
4.	Data Flow Diagram		
5.	Use Case Diagram		
6.	Activity Diagram		
7.	State Chart Diagram		
8.	Sequence Diagram		
9.	Collaboration Diagramt		
10.	Class Diagram		

EX NO:1	WRITE THE COMPLETE PROBLEM STATEMENT
DATE:	

AIM:

To prepare PROBLEM STATEMENT for any project.

ALGORITHM:

1. The problem statement is the initial starting point for a project.
2. A problem statement describes what needs to be done without describing how.
3. It is basically a one-to-three-page statement that everyone on theproject agrees with that describes what will be done at a high level.
4. The problem statement is intended for a broad audience and shouldbe written in non- technical terms.
5. It helps the non-technical and technical personnel communicate byproviding a description of a problem.
6. It doesn't describe the solution to the problem.

INPUT:

1. The input to requirement engineering is the problem statement prepared by customer.
2. It may give an overview of the existing system along with broad expectations from the new system.
3. The first phase of requirements engineering begins with requirements elicitation i.e. gathering of information about requirements.
4. Here, requirements are identified with the help of customer and existing system processes.

Problem:

The current parking management system at many urban establishments, such as malls, universities, and office complexes, is plagued with inefficiencies that lead to congestion, mismanagement of parking spaces, and frustration among users. Complaints often arise from long wait times, unclear parking availability, and frequent instances of unauthorized parking. To address these issues, there is an urgent need to modernize the parking management system to enhance user experience and optimize space utilization.

Background:

Many establishments continue to rely on manual or semi-automated processes for managing parking spaces. These outdated systems are prone to errors, such as inaccurate tracking of occupied or available spaces and ineffective monitoring of unauthorized parking. Additionally, the lack of real-time information for users often results in unnecessary delays and confusion. These challenges are compounded during peak hours, leading to inefficiencies and dissatisfaction among stakeholders.

Relevance:

Effective parking management is essential for ensuring smooth traffic flow, minimizing congestion, and improving user satisfaction. A poorly managed system can lead to negative user experiences, increased complaints, and reputational damage to establishments. By addressing these issues, institutions can improve operational efficiency, reduce traffic-related stress, and promote a more organized and accessible parking environment.

Objectives:

The primary objective of this project is to develop a modern parking management system that enhances efficiency, transparency, and user satisfaction. The specific objectives include:

1. **Analyzing Current Processes:** Conducting a detailed assessment of the existing parking management practices to identify gaps and inefficiencies.
2. **Automating Parking Operations:** Implementing a digital system for automated entry, exit, and payment processes to minimize human intervention and reduce errors.
3. **Real-Time Space Monitoring:** Introducing sensors or cameras to monitor parking space availability in real time and provide accurate updates to users.
4. **User Notifications:** Developing a mobile or web-based application to notify users about available spaces, parking duration, and payment details.
5. **Streamlining Unauthorized Parking Detection:** Integrating license plate recognition technology to detect and address unauthorized vehicles efficiently.
6. **Improving Space Utilization:** Designing algorithms to optimize the allocation of parking spaces and minimize wasted capacity during peak times.
7. **Ensuring Security:** Implementing robust surveillance and access control systems to enhance safety within the parking facility.
8. **Training Staff:** Providing comprehensive training to staff members for effective use of the new system and handling technical issues.
9. **Feedback Mechanisms:** Establishing a system to collect and act on feedback from users to ensure continuous improvement.
10. **Scalability:** Designing the system with scalability in mind to accommodate future expansion and increasing user demand.

Result:

EX NO:2	WRITE THE SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT
DATE:	

AIM:

To do requirement analysis and develop Software Requirement Specification Sheet(SRS) for any Project.

ALGORITHM:

SRS shall address are the following:

a) **Functionality.** What is the software supposed to do?

b) **External interfaces.** How does the software interact with people, the system‘s hardware, other hardware, and other software?

c) **Performance.** What is the speed, availability, response time, recovery time of various software functions, etc.?

d) **Attributes.** What is the portability, correctness, maintainability, security, etc. considerations?

e) **Design constraints imposed on an implementation.** Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

1. Introduction

1.1 Purpose

This document defines the requirements for a Parking Management System, aimed at automating the management of parking spaces and improving the overall parking experience through real-time updates, efficient space allocation, and secure payment systems.

1.2 Document Conventions

This document uses the following conventions:

- **DB** - Database
- **UI** - User Interface
- **LPR** - License Plate Recognition
- **API** - Application Programming Interface
- **RFID** - Radio Frequency Identification
- **IoT** - Internet of Things

1.3. Intended Audience and Reading Suggestions

This document is intended for:

- **Developers:** To understand the technical requirements for system implementation.
- **Project Managers:** To oversee project timelines and deliverables.
- **Stakeholders:** To ensure the system meets operational needs and user expectations.
- **Quality Assurance:** To validate and verify the functionality and performance of the system.

1.4. Project Scope

The Parking Management System will provide functionalities for real-time monitoring of parking spaces, automated payment processing, and improved user convenience. It will optimize parking operations, reduce congestion, and enhance security.

1.5. References

- Traffic and parking management guidelines.
- Institutional safety and operational protocols.
- Data privacy and IoT device integration policies.

2. Overall Description

2.1 Product Perspective

The system is a web-based and IoT-enabled application that integrates with existing parking infrastructure to streamline space allocation, monitor vehicle movements, and facilitate secure payments.

2.2 Product Features

The main features of the Parking Management System include:

- Real-time monitoring of parking space availability.
- Automated ticket generation using RFID or LPR.
- Online payment and invoicing options.
- Notifications for available spaces, overstay alerts, and payment reminders.

2.3 User Class and Characteristics

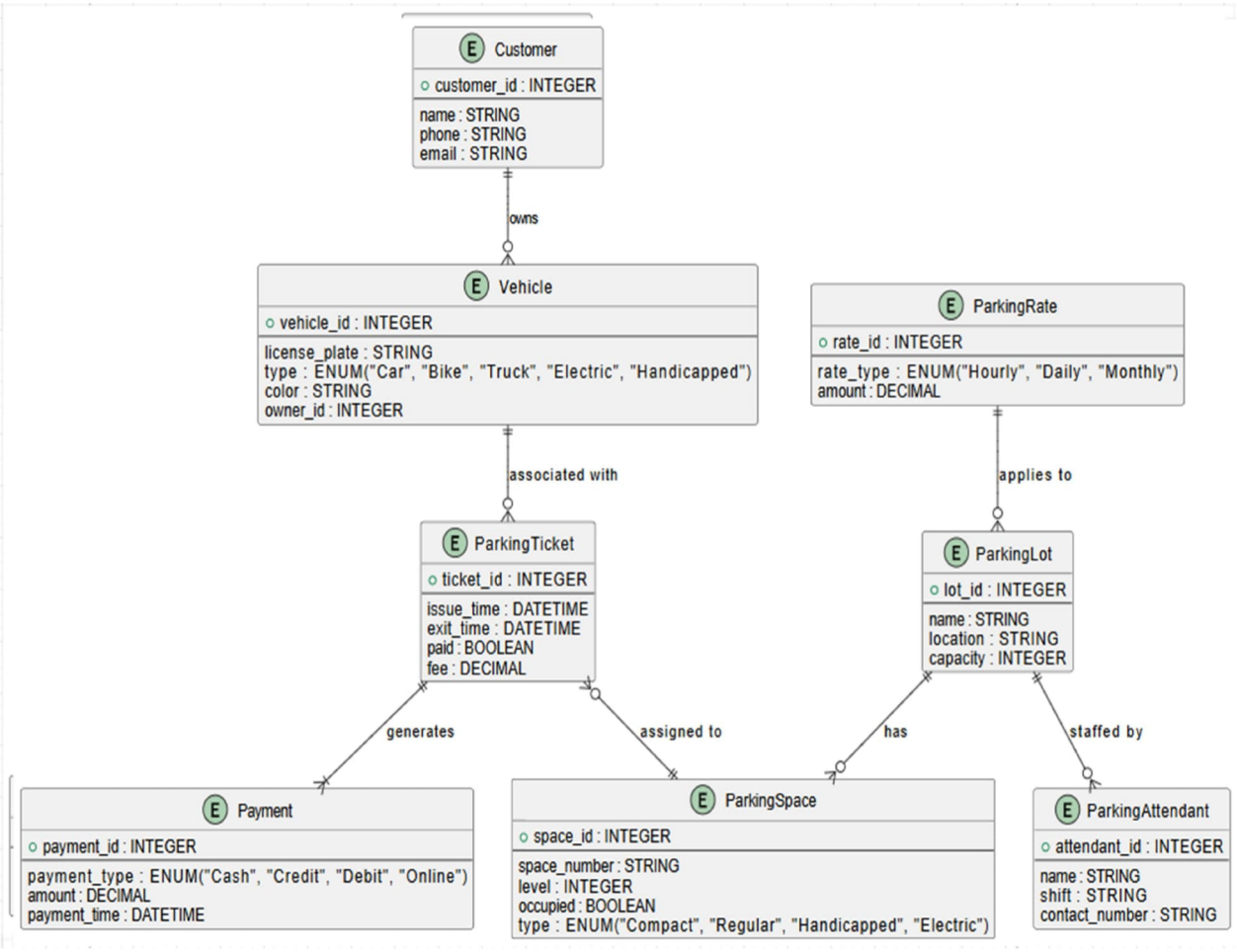
- **Parking Operators:** Manage and oversee parking operations.
- **Drivers/Users:** Locate and book parking spaces and process payments.
- **Administrators:** Monitor system performance and manage records.

2.4 Operating Environment

The system operates in the following environment:

- IoT-enabled sensors and cameras.
- Distributed database system.
- Cloud-based or on-premise server architecture.
- Modern browsers and mobile platforms for user interaction.

Entity Relationship Diagram



2.5 Design and Implementation Constraints

- **IoT Device Integration:** Integration with RFID readers, LPR cameras, and parking sensors.
- **Secure Transactions:** Secure APIs for payment gateways.
- **System Scalability:** The system must support multiple facilities or campuses.

2.6. Assumptions and Dependencies

The following assumptions and dependencies are considered in the design and implementation of the system:

- IoT sensors and devices will be deployed at all parking facilities.
- Internet connectivity is reliable for real-time data updates.
- Users will have access to smartphones for mobile app functionality.

3. Specific Requirements

Description and Priority

The system prioritizes efficient parking space management and enhanced user convenience, reducing operational inefficiencies and improving overall satisfaction.

Stimulus/Response Sequence

- A vehicle enters the parking lot: Space allocation is triggered.
- User requests payment processing: Online payment and invoicing are initiated.
- A vehicle exits: Parking record is updated, and the space is marked available.

Functional Requirements

- **Real-Time Monitoring:** IoT devices provide live updates on space availability.
- **Automated Entry/Exit:** RFID or LPR automatically logs vehicle movements.
- **Payment Processing:** Integrated payment gateway for online and onsite payments.

4. External Interface Requirements

4.1 User Interfaces

- Front-end software: Python
- Back-end software: SQL

4.2 Hardware Interfaces

- Operating System: Windows
- Browser: Any modern browser supporting HTML, CSS, and JavaScript.

4.3 Software Interfaces

- Operating System: We have selected Windows for its robust support and user-friendliness.
- Database: MySQL or MongoDB for storing parking records, user data, and transaction logs.
- Programming Language: Python with Streamlit is chosen for developing the application due to its ease of use, rapid development capabilities, and interactive features.

4.4 Communication Interfaces

- The system supports all modern web browsers. Users interact with the system through the Streamlit web application, which provides forms for entering student results and managing email notifications.

5. Additional Requirements

5.1 Performance Requirements

- **ER Diagram:** Represents entities such as vehicles, spaces, and transactions and their relationships.
- **Normalization:** Ensures efficient data storage, retrieval, and minimal redundancy.

5.2 Safety Requirements

In case of a catastrophic failure, the system should have a robust recovery mechanism. The database will regularly backup to archival storage, allowing the reconstruction of the system to a recent state using backend-ups.

5.3 Security Requirements

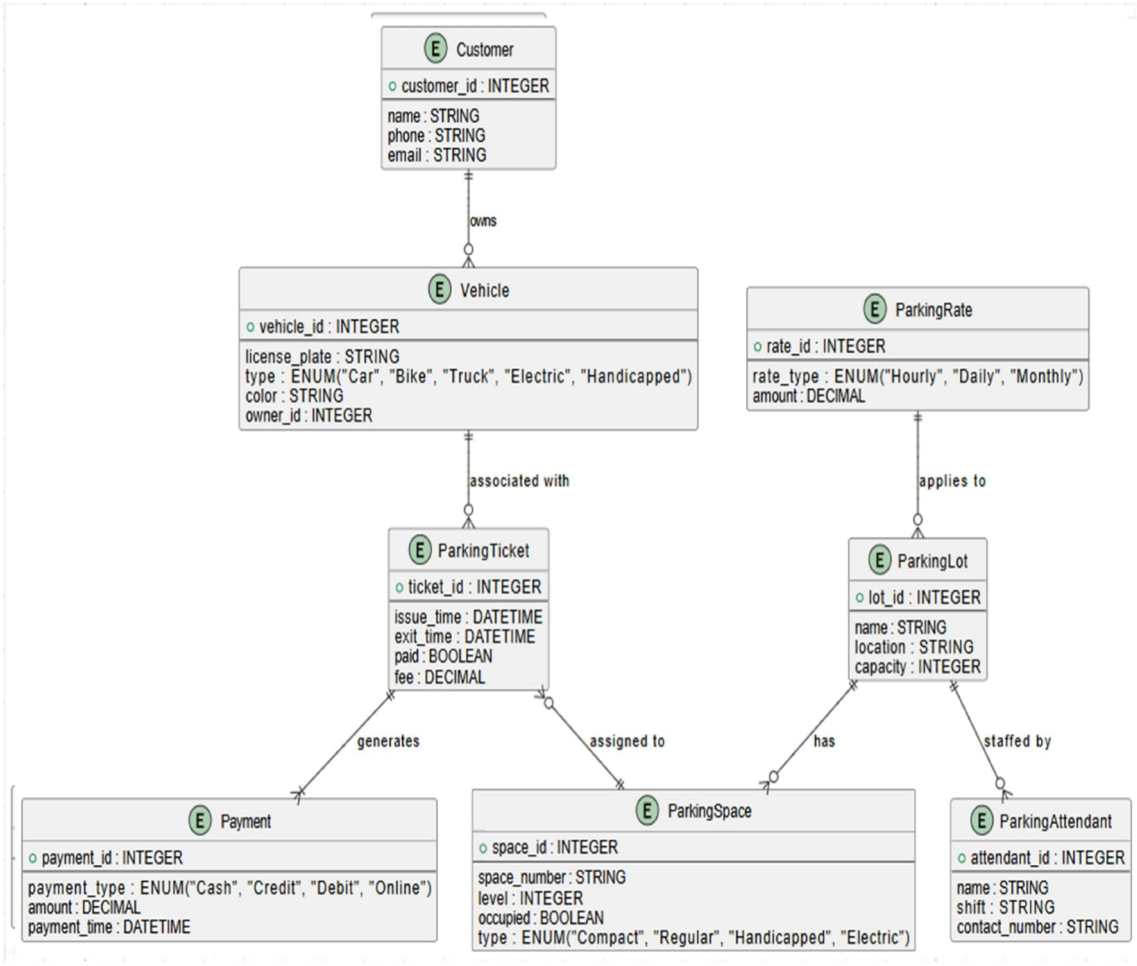
The system must ensure secure storage and transmission of student and parent data. Security protocols must be in place to prevent unauthorized access.

5.4 Software Quality Attributes

- Availability: The system should be available at all times.
- Correctness: All data entries and email notifications should be accurate.
- Maintainability: The system should support regular updates, including bug fixes and security enhancements.
- Usability: The application should be user-friendly.

Result:

Entity-Relationship Diagram



EX NO:3	DRAW THE ENTITY RELATIONSHIP DIAGRAM
DATE:	

AIM:

To Draw the Entity Relationship Diagram for any project.

ALGORITHM:

Step 1: Mapping of Regular Entity Types

Step 2: Mapping of Weak Entity Types

Step 3: Mapping of Binary 1:1 Relation Types

Step 4: Mapping of Binary 1:N Relationship Types.

Step 5: Mapping of Binary M:N Relationship Types.

Step 6: Mapping of Multivalued attributes.

INPUT:

Entities

Entity Relationship Matrix

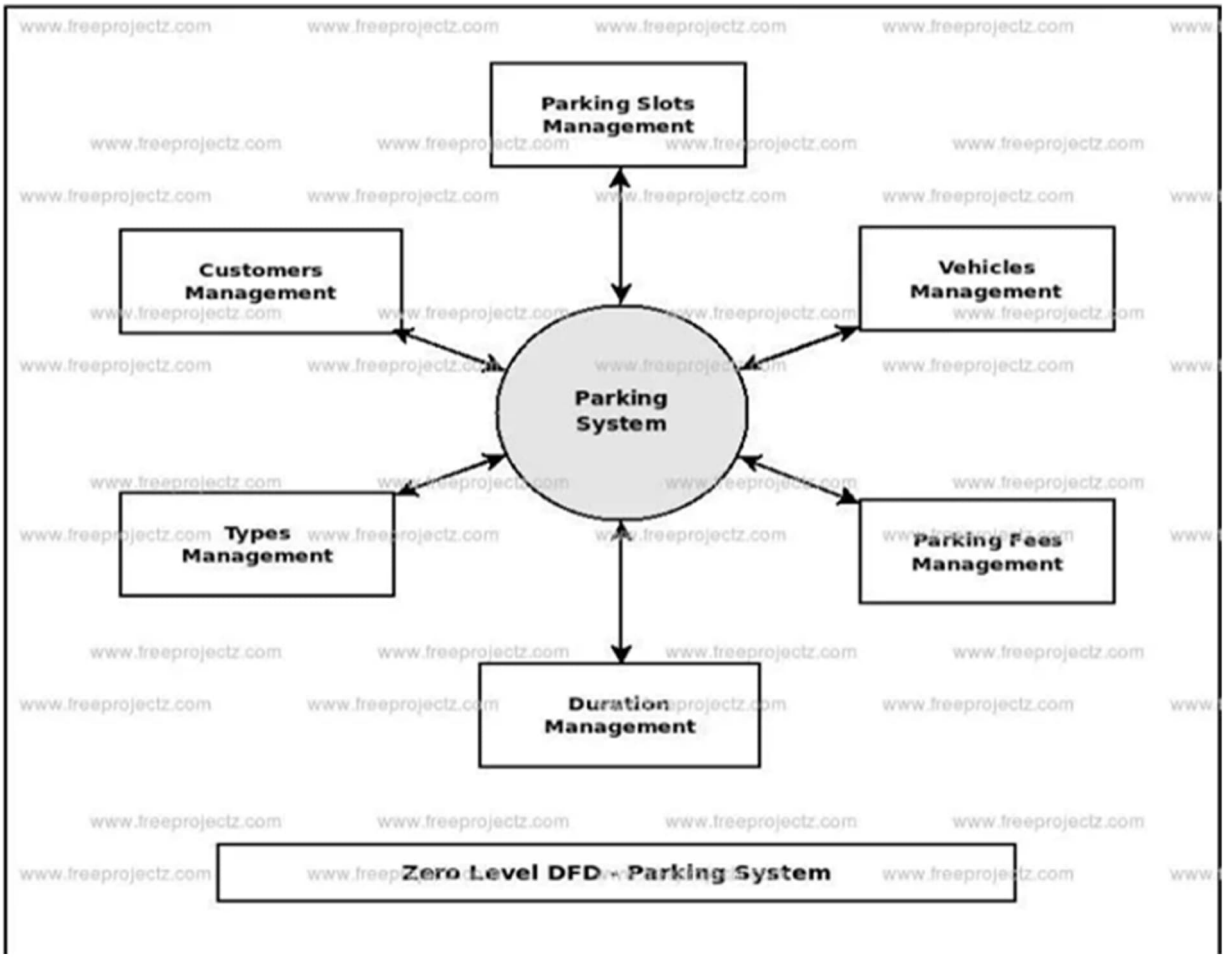
Primary Keys

Attributes

Mapping of Attributes with Entities

Result:

Data Flow Diagram



EX NO:4	DRAW THE DATA FLOW DIAGRAMS AT LEVEL 0 AND LEVEL 1
DATE:	

AIM:

To Draw the Data Flow Diagram for any project and List the Modules in the Application.

ALGORITHM:

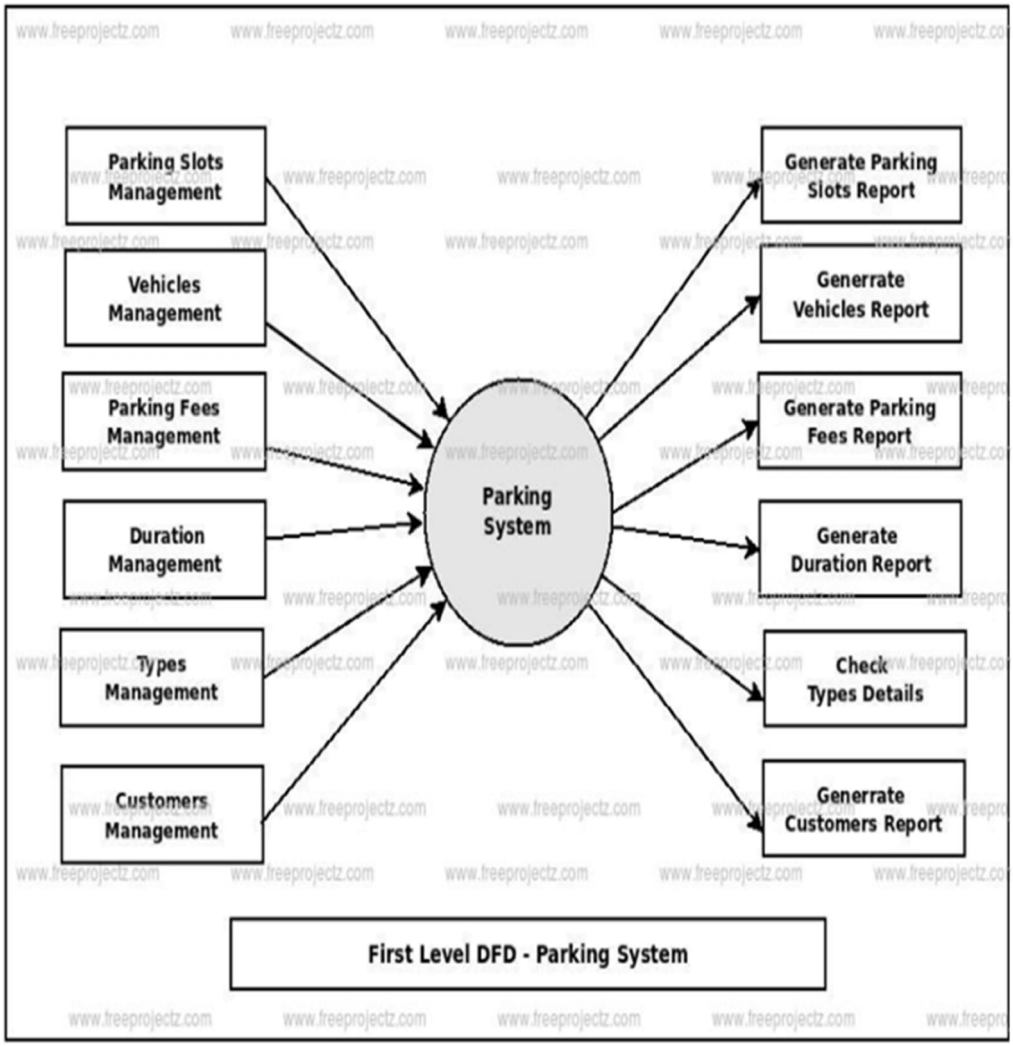
1. Open the Visual Paradigm to draw DFD (Ex.Lucidchart)
2. Select a data flow diagram template
3. Name the data flow diagram
4. Add an external entity that starts the process
5. Add a Process to the DFD
6. Add a data store to the diagram
7. Continue to add items to the DFD
8. Add data flow to the DFD
9. Name the data flow
10. Customize the DFD with colours and fonts
11. Add a title and share your data flow diagram

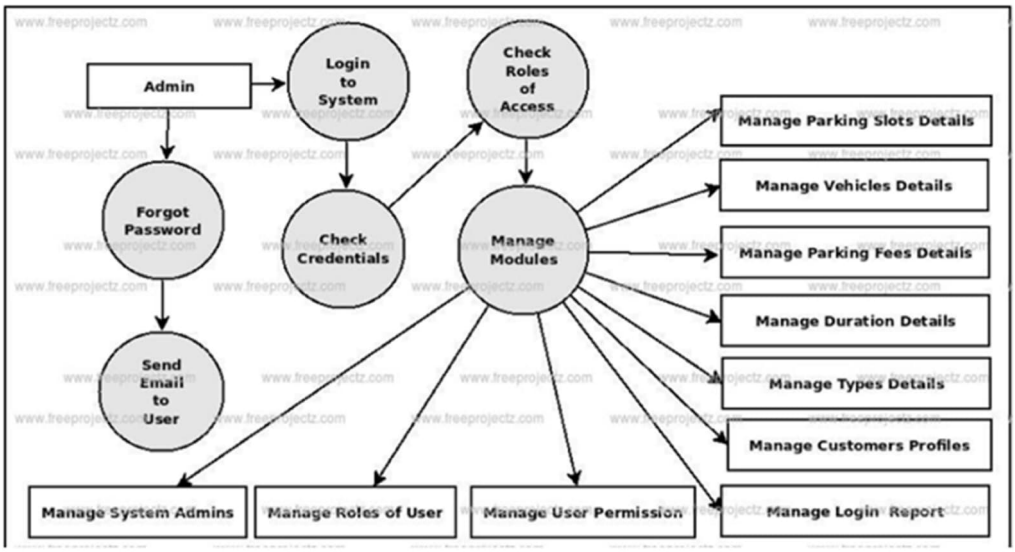
INPUT:

Processes

Datastores

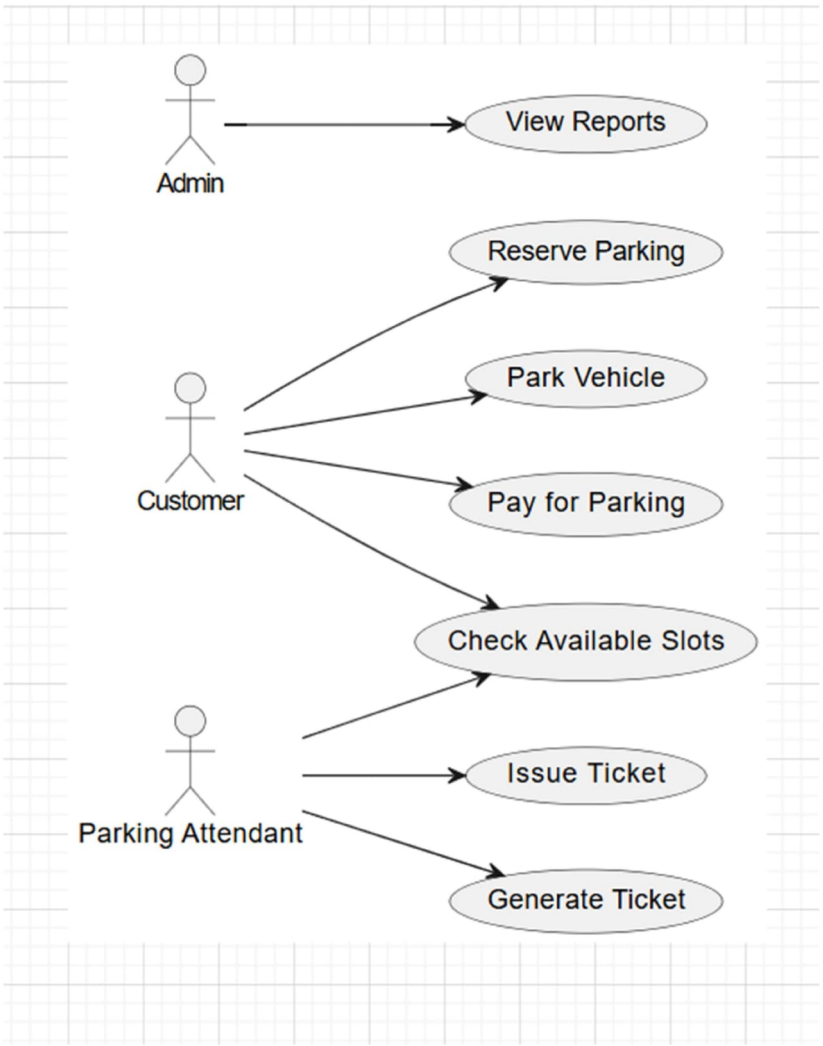
External Entities


Parking Slots Management
Vehicles Management
Parking Fees Management
Duration Management
Types Management
Customers Management
Parking System
Generate Parking Slots Report
Generrate Vehicles Report
Generate Parking Fees Report
Generate Duration Report
Check Types Details
Generrate Customers Report
First Level DFD - Parking System


Admin
Login to System
Check Roles of Access
Forgot Password
Check Credentials
Manage Modules
Send Email to User
Manage Parking Slots Details
Manage Vehicles Details
Manage Parking Fees Details
Manage Duration Details
Manage Types Details
Manage Customers Profiles
Manage Login Report
Manage System Admins
Manage Roles of User
Manage User Permission

Result:

Use case Diagram



EX NO:5	DRAW USE CASE DIAGRAM
DATE:	

AIM:

To Draw the Use Case Diagram for any project

ALGORITHM:

Step 1: Identify Actors

Step 2: Identify Use Cases

Step 3: Connect Actors and Use Cases

Step 4: Add System Boundary

Step 5: Define Relationships

Step 6: Review and Refine

Step 7: Validate

INPUTS:

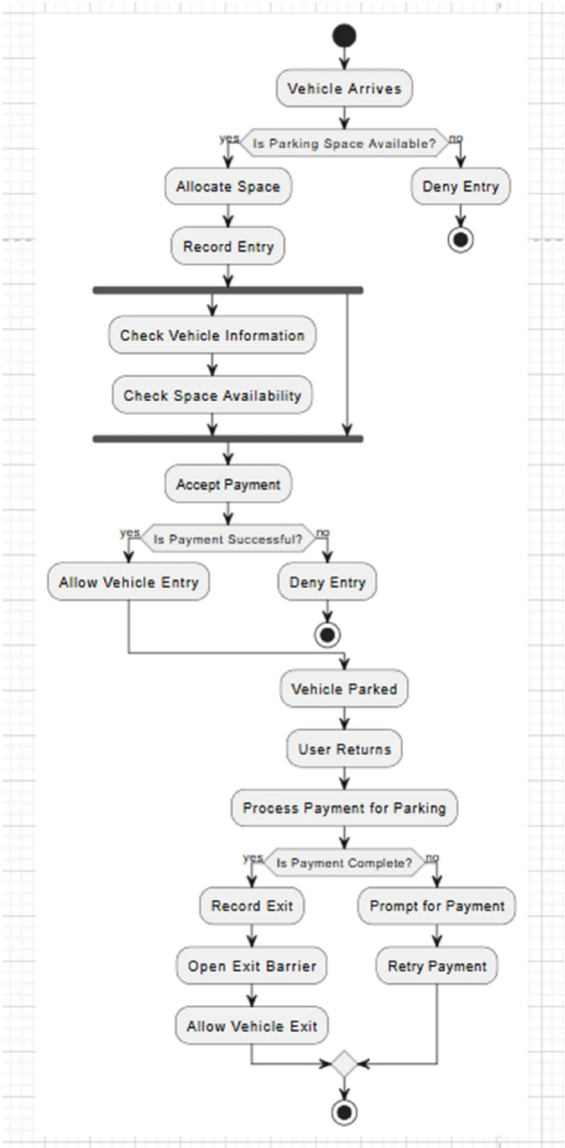
Actors

Use Cases

Relations

Result:

Activity Diagram



EX NO:6	DRAW ACTIVITY DIAGRAM OF ALL USE CASES.
DATE:	

AIM:

To Draw the activity Diagram for any project

ALGORITHM:

Step 1: Identify the Initial State and Final States

Step 2: Identify the Intermediate Activities Needed

Step 3: Identify the Conditions or Constraints

Step 4: Draw the Diagram with Appropriate Notations

INPUTS:

Activities

Decision Points

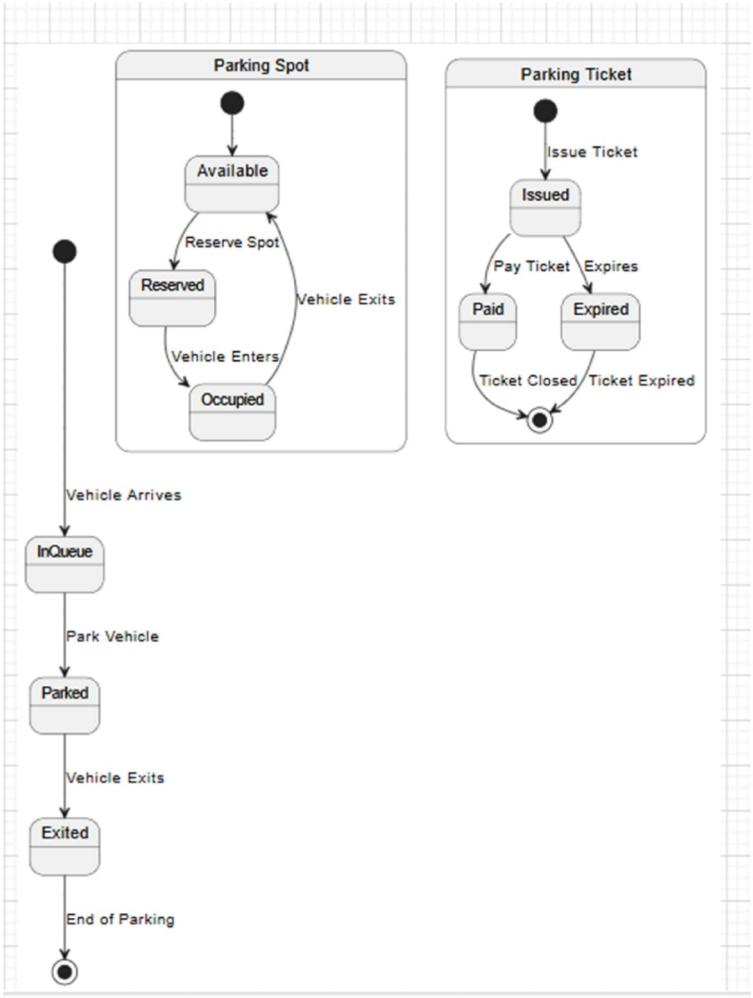
Guards

Parallel Activities

Conditions

Result:

State Chart Diagram



EX NO:7	DRAW STATE CHART DIAGRAM OF ALL USE CASES.
DATE:	

AIM:

To Draw the State Chart Diagram for any project

ALGORITHM:

STEP-1: Identify the important objects to be analysed.

STEP-2: Identify the states.

STEP-3: Identify the events.

INPUTS:

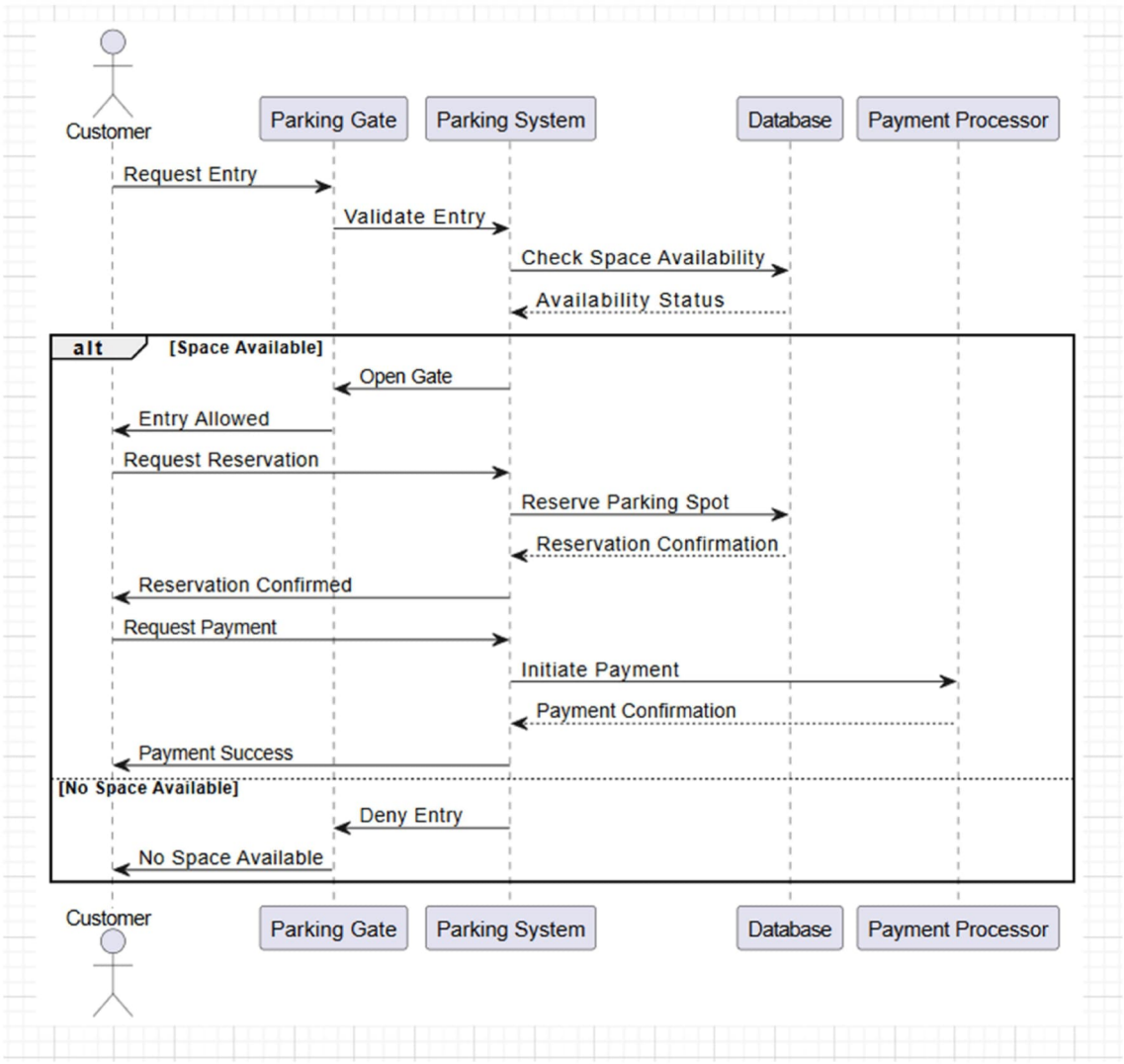
Objects

States

Events

Result:

Sequence Diagram



EX NO:8	DRAW SEQUENCE DIAGRAM OF ALL USE CASES.
DATE:	

AIM: To Draw the Sequence Diagram for any project

ALGORITHM:

1. Identify the Scenario
2. List the Participants
3. Define Lifelines
4. Arrange Lifelines
5. Add Activation Bars
6. Draw Messages
7. Include Return Messages
8. Indicate Timing and Order
9. Include Conditions and Loops
10. Consider Parallel Execution
11. Review and Refine
12. Add Annotations and Comments
13. Document Assumptions and Constraints
14. Use a Tool to create a neat sequence diagram

INPUTS:

Objects taking part in the interaction.

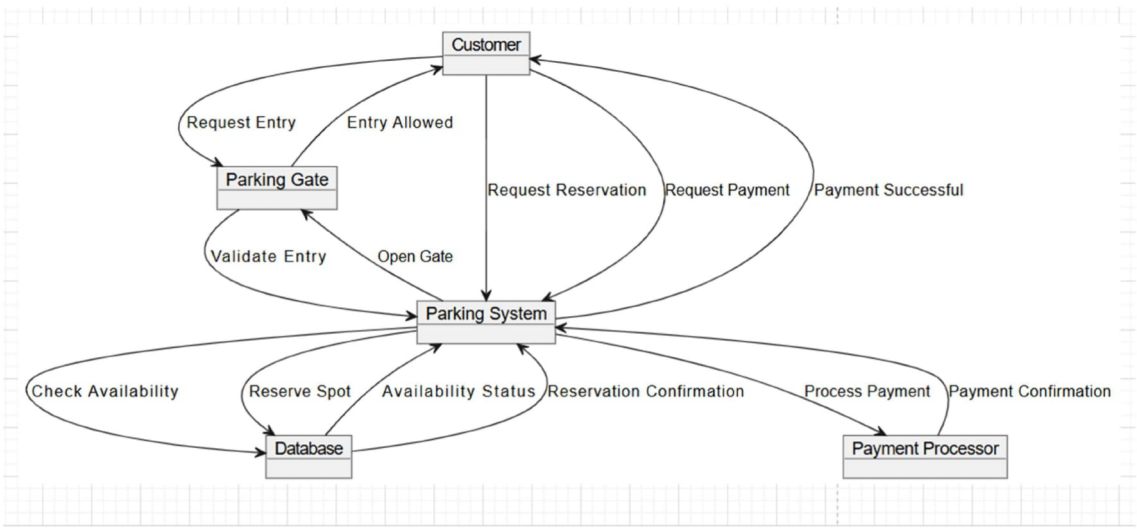
Message flows among the objects.

The sequence in which the messages are flowing.

Object organization.

Result:

Collaboration Diagram



EX NO:9	DRAW COLLABORATION DIAGRAM OF ALL USE CASES
DATE:	

AIM:

To Draw the Collaboration Diagram for any project

ALGORITHM:

Step 1: Identify Objects/Participants

Step 2: Define Interactions

Step 3: Add Messages

Step 4: Consider Relationships

Step 5: Document the collaboration diagram along with any relevant explanations or annotations.

INPUTS:

Objects taking part in the interaction.

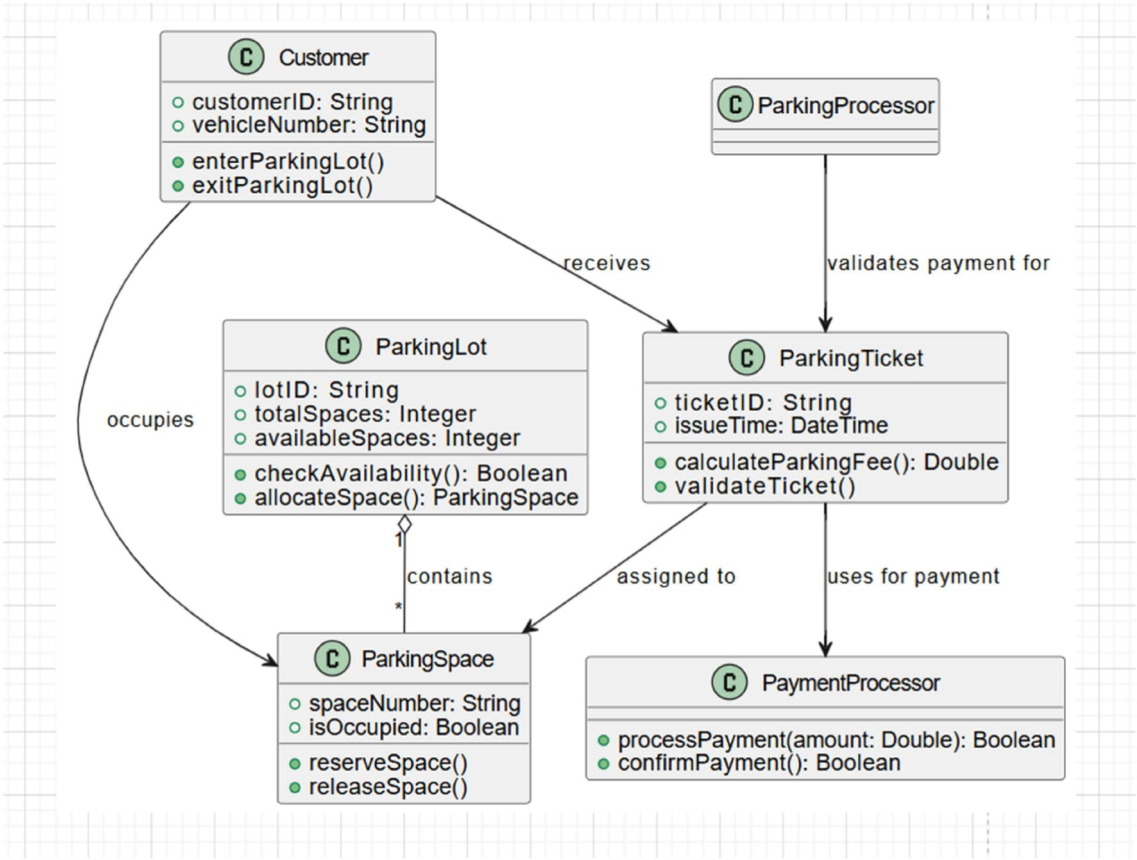
Message flows among the objects.

The sequence in which the messages are flowing.

Object organization.

Result:

Class Diagram



EX NO:10	ASSIGN OBJECTS IN SEQUENCE DIAGRAM TO CLASSES AND MAKE CLASS DIAGRAM.
DATE:	

AIM:

To Draw the Class Diagram for any project

ALGORITHM:

1. Identify Classes
2. List Attributes and Methods
3. Identify Relationships
4. Create Class Boxes
5. Add Attributes and Methods
6. Draw Relationships
7. Label Relationships
8. Review and Refine
9. Use Tools for Digital Drawing

INPUTS:

1. Class Name
2. Attributes
3. Methods
4. Visibility Notation

RESULT:

EX NO:11	MINI PROJECT- PARKING MANAGEMENT SYSTEM
DATE:	

AIM:

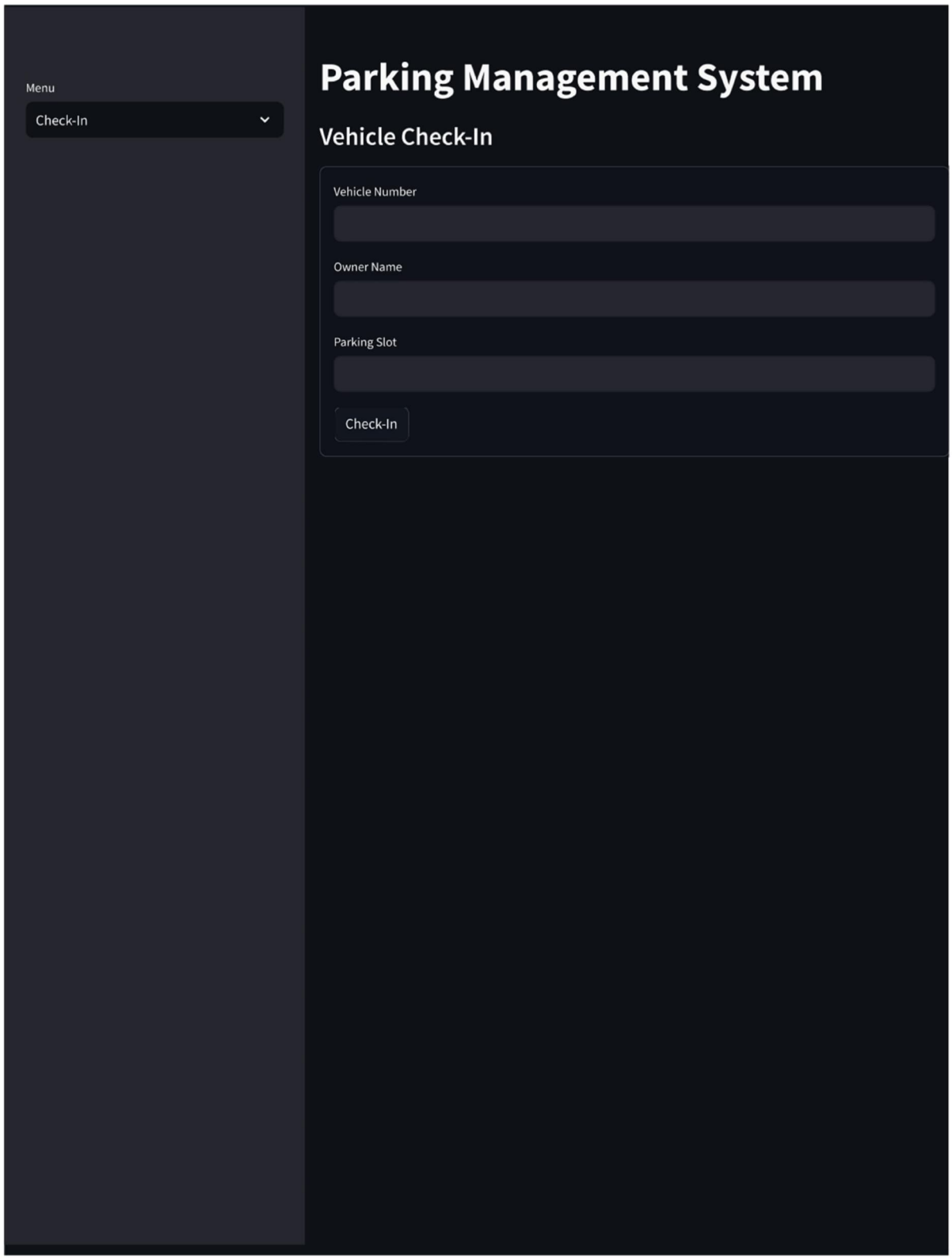
To develop a Parking Management System using Streamlit and MySQL that allows staff members to efficiently manage parking slots, register vehicles, track parking activity, and notify vehicle owners via email. The focus is to streamline parking operations, enhance user convenience, and ensure reliability and simplicity in managing parking facilities.

ALGORITHM:

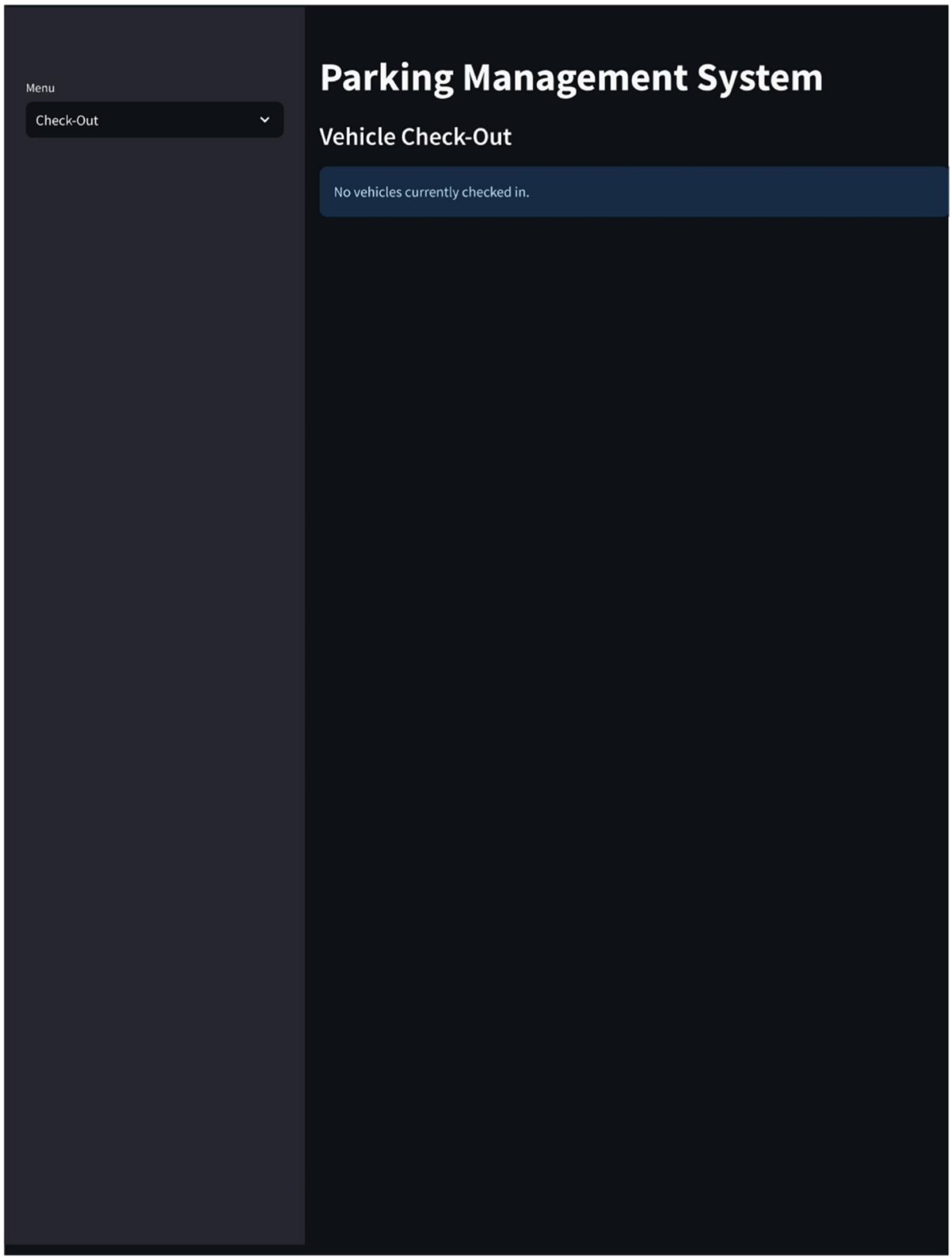
1. Database Connection Initialization
2. Streamlit Interface Setup
3. Operation Selection
    - Update Parking Slot
    - Display Parking Slot
4. Database Query Execution
5. Email Notification (for Update Parking Slots)
6. Feedback Display
7. Application Termination

PROGRAM:

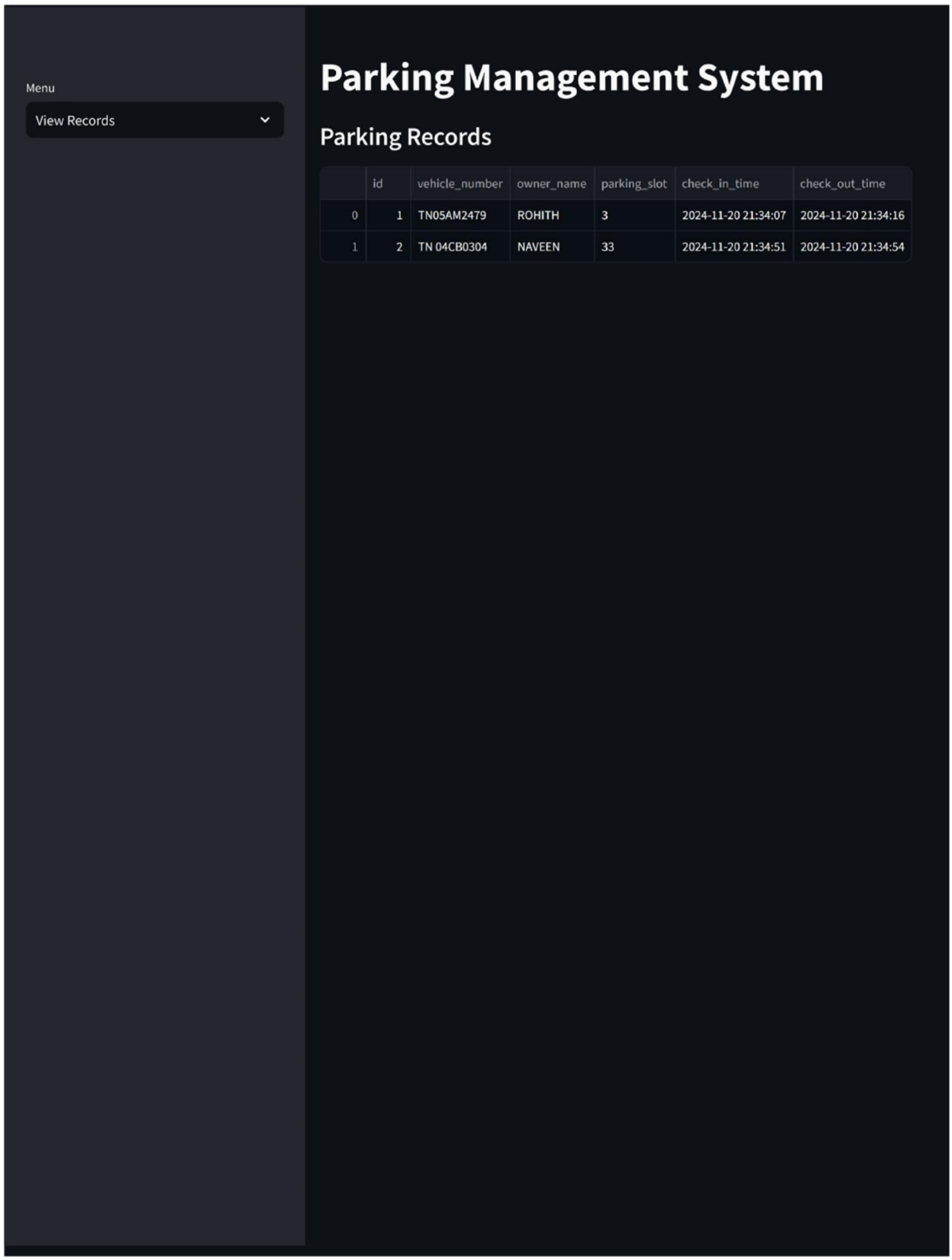
```
import streamlit as st

import sqlite3

import pandas as pd

from datetime import datetime

# --- database connection ---

def init_db():

    conn = sqlite3.connect("parking_system.db")

    cursor = conn.cursor()

    cursor.execute("""

        create table if not exists parking (

            id integer primary key autoincrement,
```


Menu
Check-In
Parking Management System
Vehicle Check-In
Vehicle Number
Owner Name
Parking Slot
Check-In

```
            vehicle_number text not null,
            owner_name text not null,
            parking_slot text not null,
            check_in_time text not null,
            check_out_time text
        )
    """)
    conn.commit()
    return conn

# insert parking record
def add_parking_record(conn, vehicle_number, owner_name, parking_slot):
    cursor = conn.cursor()
    cursor.execute("""
        insert into parking (vehicle_number, owner_name, parking_slot, check_in_time)
        values (?, ?, ?, ?)
    """, (vehicle_number, owner_name, parking_slot, datetime.now().strftime("%y-%m-%d %h:%m:%s")))
    conn.commit()

# fetch parking records
def get_parking_records(conn):
    return pd.read_sql("select * from parking", conn)

# update checkout time
def update_checkout_time(conn, record_id):
    cursor = conn.cursor()
    cursor.execute("""
        update parking
```


Menu
Check-Out
Parking Management System
Vehicle Check-Out
No vehicles currently checked in.

```
        set check_out_time = ?
        where id = ?
    """, (datetime.now().strftime("%y-%m-%d %h:%m:%s"), record_id))
    conn.commit()


# --- streamlit frontend ---
def main():
    st.title("parking management system")


    # sidebar menu
    menu = ["check-in", "check-out", "view records"]
    choice = st.sidebar.selectbox("menu", menu)


    conn = init_db()


    if choice == "check-in":
        st.subheader("vehicle check-in")
        with st.form("check_in_form"):
            vehicle_number = st.text_input("vehicle number")
            owner_name = st.text_input("owner name")
            parking_slot = st.text_input("parking slot")
            submit_button = st.form_submit_button("check-in")


        if submit_button:
            if vehicle_number and owner_name and parking_slot:
                add_parking_record(conn, vehicle_number, owner_name, parking_slot)
                st.success(f"vehicle {vehicle_number} checked in successfully!")
            else:
```


Menu
View Records
Parking Management System
Parking Records
id vehicle_number owner_name parking_slot check_in_time check_out_time
0 1 TN05AM2479 ROHITH 3 2024-11-20 21:34:07 2024-11-20 21:34:16
1 2 TN 04CB0304 NAVEEN 33 2024-11-20 21:34:51 2024-11-20 21:34:54

```
            st.error("please fill in all fields.")
    elif choice == "check-out":
        st.subheader("vehicle check-out")
        parking_data = get_parking_records(conn)
        if not parking_data.empty:
            parking_data = parking_data[parking_data["check_out_time"].isnull()]
            if not parking_data.empty:
                record_id = st.selectbox("select check-in record", parking_data["id"].tolist())
                if st.button("check-out"):
                    update_checkout_time(conn, record_id)
                    st.success("check-out successful!")
            else:
                st.info("no vehicles currently checked in.")
        else:
            st.info("no records found.")
    elif choice == "view records":
        st.subheader("parking records")
        parking_data = get_parking_records(conn)
        st.dataframe(parking_data)
# correct __name__ check
if __name__ == "__main__":
    main()
```

Conclusion

The Parking Management System developed using Streamlit and SQLite offers a user-friendly interface for staff, vehicle owners, and administrators to efficiently manage parking operations. This system centralizes key functionalities such as vehicle check-in and check-out, tracking parking slot usage, and maintaining detailed parking records. By streamlining these processes, the system ensures simplicity, enhances operational efficiency, and provides a reliable solution for managing parking facilities effectively.