CITY COLLEGE OF CALAMBA
DEPARTMENT OF COMPUTING AND INFORMATICS
Bachelor of Science in Computer Science

PAY-PARKING SYSTEM USING FIRST COME, FIRST SERVE (FCFS) ALGORITHM

An Operating System Project

Presented to

Prof. Jacqueline A. Dela Torre

Of Department of Computing and Informatics

Of City College of Calamba

Calamba City, Laguna

Andrade, Raechelle B.

Leyba, Geneen Lucia T.

Paring, Roy Vincent M.

July 17, 2023

CHAPTER 1

INTRODUCTION

This chapter includes the introduction, objectives of the study, general objective, specific objective, scope and limitations and significance of the study.

For drivers who frequently visit busy commercial areas like malls, searching for an available parking spot can sometimes turn into a frustrating experience due to limited availability and poorly managed operations resulting in congestion and delays leading to disgruntled shoppers. To effectively address these challenges head-on during peak visiting hours at these locations this study puts forward a proposal: utilizing the first come first serve algorithm via implementing an adequately priced pay-parking system. The objective being optimized space allocation translating into more contented customers thinking positively about their overall mall visit experience.

The fairness of this approach lies within its usage of allocated resources based on arrival order when implemented for mall location parking solutions which essentially means available spots are assigned per arriving vehicles' sequence with impartiality across-the-board sans any bias or preference; simplifying the process that much more alleviating burdensome complexities with prioritization or reservation requests made by drivers.

Implementing a pay-parking system with the first come, first serve algorithm offers several advantages specifically tailored for mall environments. Firstly, it optimizes parking space utilization by efficiently assigning available spots to vehicles as they arrive. This helps to minimize the time spent searching for parking, reduce congestion within the parking lot, and enhance traffic flow in the surrounding areas. By effectively managing the parking demand, the system aims to improve overall accessibility and convenience for mall visitors.

Secondly, a pay-parking system based on the first come, first serve algorithm brings transparency and fairness to the parking experience. Customers can rely on the system to allocate parking spaces impartially, without any preferences or discriminatory practices. This promotes trust and ensures that all drivers have an equal opportunity to access parking facilities, regardless of their status or affiliations.

This study focuses on the design, implementation, and evaluation of a pay-parking system utilizing the first come, first serve algorithm specifically tailored for a mall environment. We will explore the technical aspects involved in developing such a system, including the integration of ticketing or payment mechanisms, parking occupancy monitoring, and user-friendly interface for parking administrators. Furthermore, we will assess the impact of this system on reducing congestion, enhancing parking efficiency, and improving the overall customer satisfaction within the mall premises.

The findings from this study aim to contribute valuable insights to the field of parking management within commercial spaces, with a focus on malls. By exploring the potential benefits of a pay-parking system utilizing the first come, first serve algorithm, we aspire to provide practical recommendations and guidelines for implementing efficient parking solutions. Ultimately, the goal is to create a seamless and enjoyable parking experience for mall visitors while optimizing the utilization of parking spaces and contributing to sustainable urban development.

OBJECTIVES OF THE STUDY

The general purpose of this research is to develop a Pay-parking system that uses a First come, First serve Algorithm to optimize the utilization of a parking space in mall environments.

General Objectives

The study assesses the effectiveness of a pay-parking system in a mall context using the first come, first served algorithm. It evaluates it influence on parking spot use, congestion reduction, and overall parking experience. The findings offer suggestions for effective parking management solutions in commercial spaces, specifically malls.

Specific Objectives

- To design and develop a pay-parking system that incorporates the first come, first serve algorithm, tailored to the specific needs and requirements of a mall environment.
- To implement the developed system in a real-world mall parking setting, ensuring seamless integration with existing infrastructure, ticketing or payment mechanisms, and monitoring tools.
- To analyze the effectiveness of the system in allocating parking spaces fairly and efficiently, based on the order of vehicle arrival, while considering a factor such as parking lot capacity.

SCOPE AND LIMITATIONS

The scope of this study examines the use of a first come, first serve algorithm in a pay-parking system in shopping malls. It covers design, implementation, and evaluation aspects, including integration with existing infrastructure, assessing its effectiveness in optimizing parking space utilization and improving the overall parking experience. The study also examines the impact of the system on reducing congestion and customer satisfaction, as well as the economic feasibility and financial aspects of implementing such a system in a mall setting.

However, there are limitations to consider, such as scalability issues, technical capability, stakeholder coordination, payment methods, external factors like road traffic patterns, weather conditions, and non-mall events, time limits, and privacy issues. Despite these limitations, the report provides valuable recommendations and insights for implementing a first come, first serve algorithm pay-parking system, contributing to the development of prevailing parking management strategies in commercial spaces.

SIGNIFICANCE OF THE STUDY

This study is conducted to benefit the following:

- The client - The system will quickly determine what type of vehicle you will park, and the procedure will be faster and they will know exactly where to park the vehicle.
- The parking attendant - They can easily locate vacant parking spaces and easily record the vehicle's information as well as the time allotted by the clients.
- The owner of pay parking – This system will help in making their service more efficient and faster, ensuring that vehicles are not messy when they arrive at the parking area.
- The researchers – This study may help the researchers expand their knowledge and abilities and improve the system's knowledge.
- The future researchers - The findings of this study will help to give future researchers an idea of what they will do and provide additional knowledge.

CHAPTER 2

SOFTWARE DEVELOPMENT METHODOLOGY

In this research project, a software development methodology is being introduced to effectively address the challenges and requirements involved in the development of the software component.

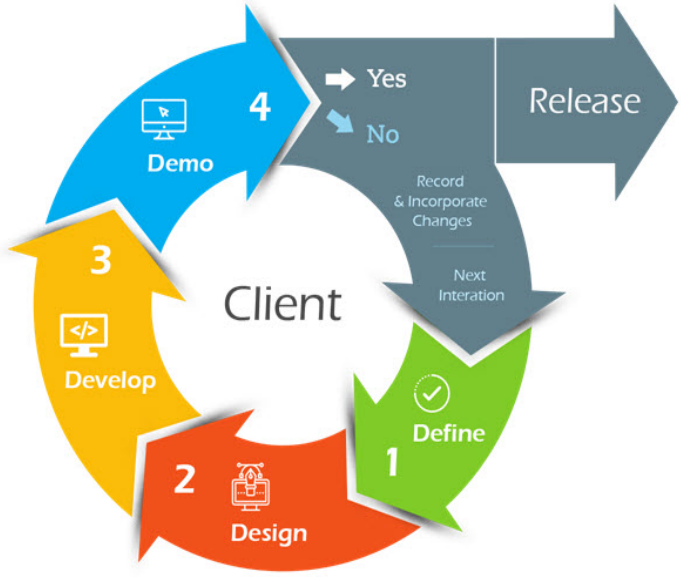

Figure 1.1: Agile Methodology Phases

The agile methodology follows a series of phases to guide the development process. The first phase involves defining the project's vision and prioritizing user requirements. In the design phase, the system's architecture and detailed specifications are established. The development phase focuses on coding and implementing the system in iterative sprints. After each sprint, a demo session is held to showcase the implemented features and gather feedback. Finally, the release phase marks the deployment of the system or its specific features to end-users. These phases promote collaboration, adaptability, and continuous improvement throughout the development journey.

Agile methodology is an iterative and incremental approach to project management and software development. It emphasizes flexibility, collaboration, and continuous improvement. Unlike traditional waterfall methods, where the entire project is planned upfront, agile methodologies encourage adaptive planning and embrace change throughout the development process.

There are several reasons why we chose the agile methodology for the development of our Pay-parking system. Firstly, agile allows us to respond to evolving customer needs and market demands effectively. By breaking down the project into smaller increments called sprints, we can continuously gather feedback and incorporate changes to deliver a high-quality product that aligns with user expectations.

In addition, agile methodology promotes close collaboration among cross-functional teams. This enables better communication, knowledge sharing, and coordination between developers, testers, designers, and other stakeholders. By working together throughout the development cycle, we can ensure a more comprehensive and integrated system.

Finally, the iterative nature of agile allows for early and frequent delivery of working software. By delivering incremental releases, we can showcase the system's functionality and gather user feedback at each stage. This iterative feedback loop enables us to validate assumptions, identify and resolve issues early on, and make adjustments as needed. It also provides stakeholders with a tangible product that they can interact with, increasing transparency and trust.

SYSTEM OVERVIEW

The pay-parking system efficiently manages and monitors parking spaces in a mall facility by implementing a first come, first serve algorithm. This algorithm ensures that parking spaces are allocated in the order of vehicle arrival, minimizing congestion and maximizing space utilization within the parking facility.

The system consists of several main subsystems that work together to achieve its objectives. The entrance and exit gates act as the primary entry and exit points for vehicles, being monitored by the parking attendant. The vehicle type detection includes special codes to strategically place the vehicles throughout the parking facility.

The central control system acts as the core component of the pay-parking system, receiving occupancy information from the vehicle codes. It utilizes the first come, first serve algorithm to assign parking spaces to incoming vehicles, ensuring fairness and minimizing conflicts. Additionally, the central control system tracks parking duration and calculates parking fees based on the time of exit.

When it comes to payments, it is also conveniently within the system placed in the entrance/exit gates operated by the parking attendant. Provided with a user-friendly interface to make the parking fee payments easier and efficient. It displays the calculated parking fees but only accept one payment method which is cash.

Overall, the pay-parking system in mall environments optimizes the utilization of parking spaces, providing a seamless and convenient parking experience for customers. It efficiently allocates parking spots based on arrival times, and contributing to improved traffic flow and customer satisfaction in mall parking facilities.

SOFTWARE DEVELOPMENT PROCESS (FLOWCHART)

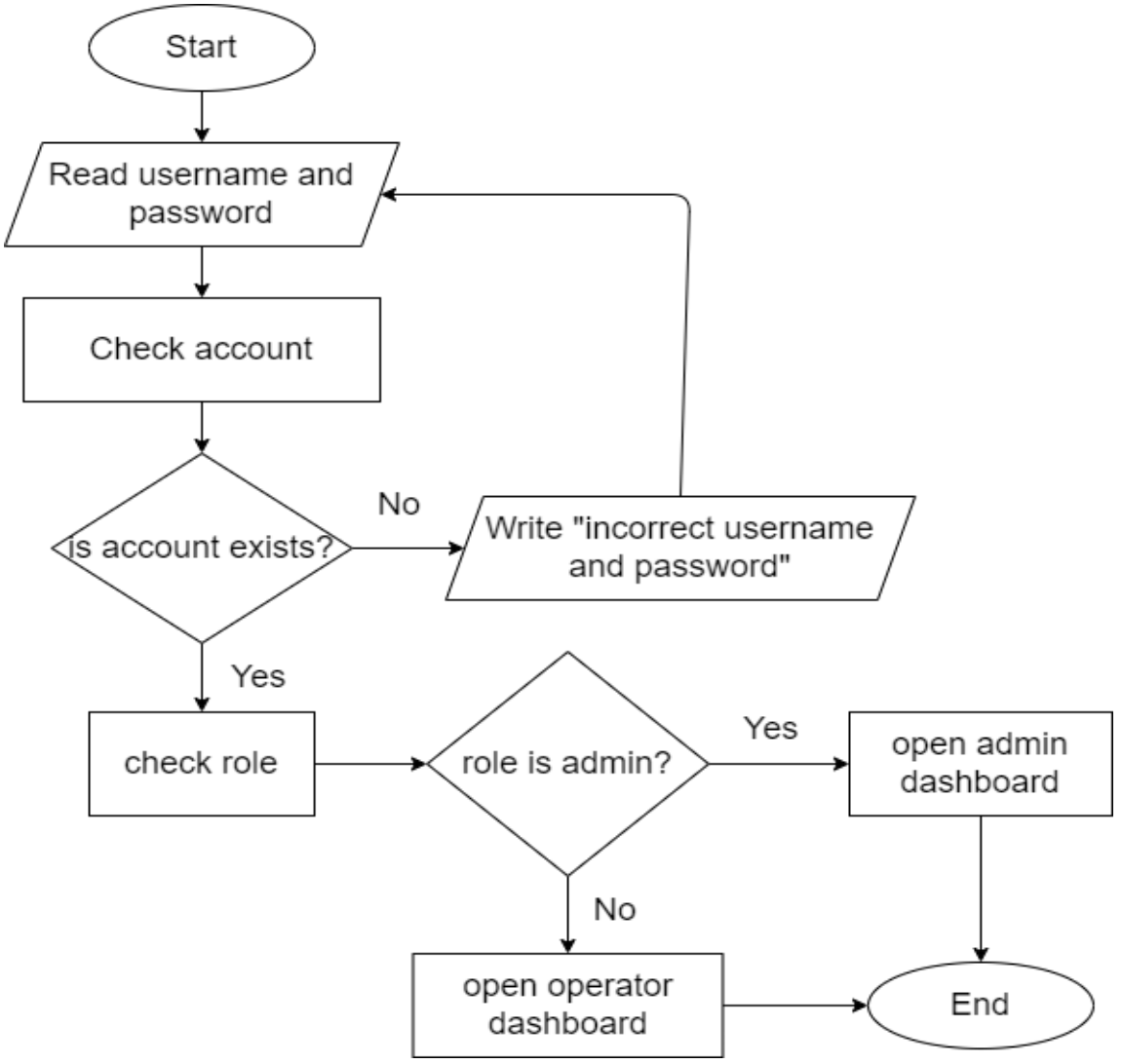

Figure 2.1: Login form flowchart

The figure 2.1 shows the flowchart of the login form of the pay-parking system. Starting the system will require a username and password, only the operator and the admin can access the login form with their respective username and password.

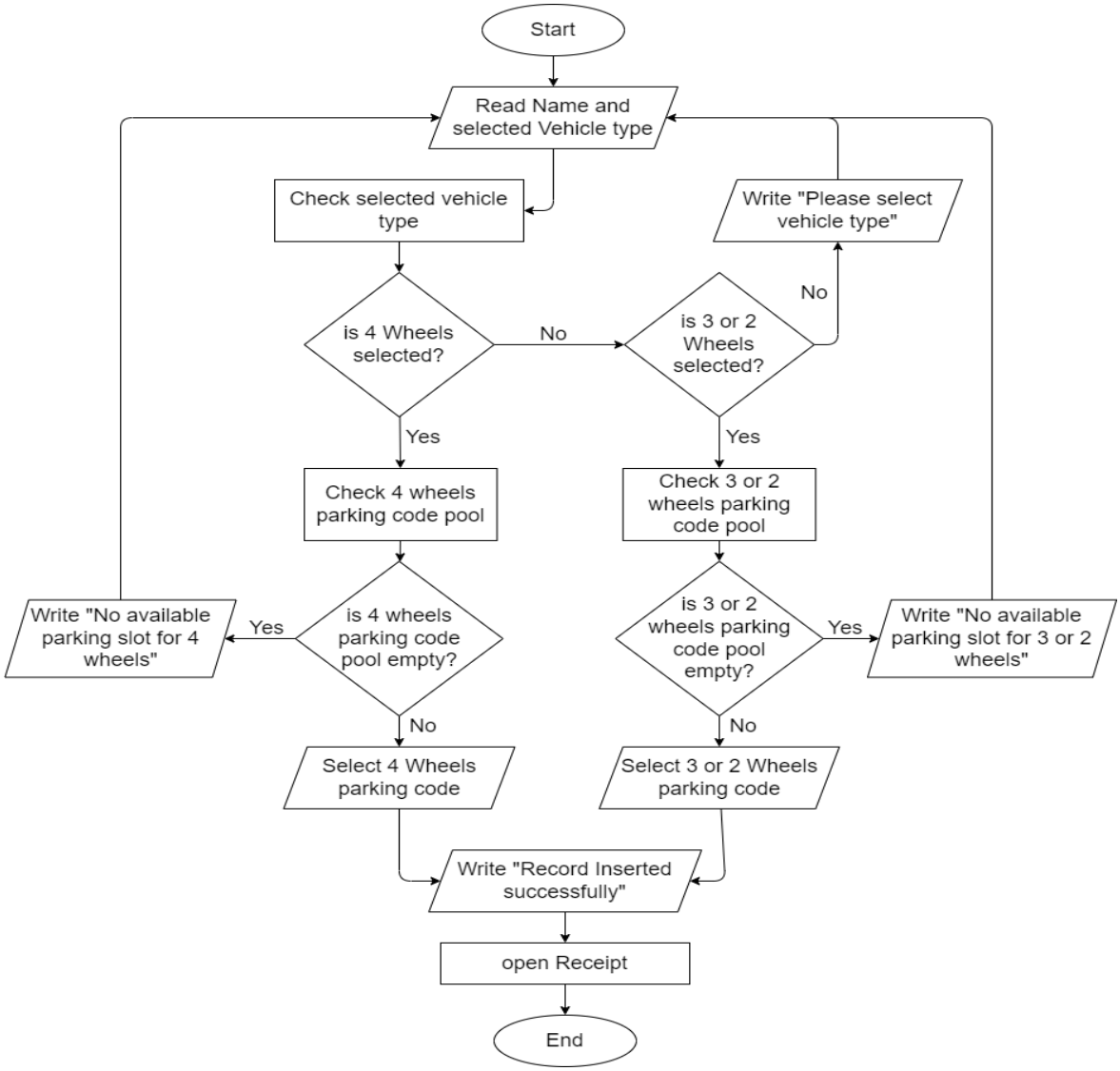

Figure 2.2: Time-in process flowchart

The process of timing-in various vehicles by the operator is shown on Figure 2.2: Operator dashboard of the time-in process flowchart. To start the process, the operator will input the name and the vehicle type of the customer. There are 3 vehicle types (4 wheels, 3 wheels and 2 wheels). After that input, the operator will now check if there are any available spots/parking space left for the inserted vehicle type. If there aren't any, the process will be immediately terminated, therefore the vehicle will be forced to go out of the premises. But if many parking space/slots are available, the operator will now choose where will the vehicle will park. After inputting the name, vehicle type and choosing the code, the operator will have to click submit in

order for the process to be written in the system. After successfully recording the vehicle in the system, a receipt will pop-up in the system and a customer's copy will be printed and the operator will hand it to the customer.

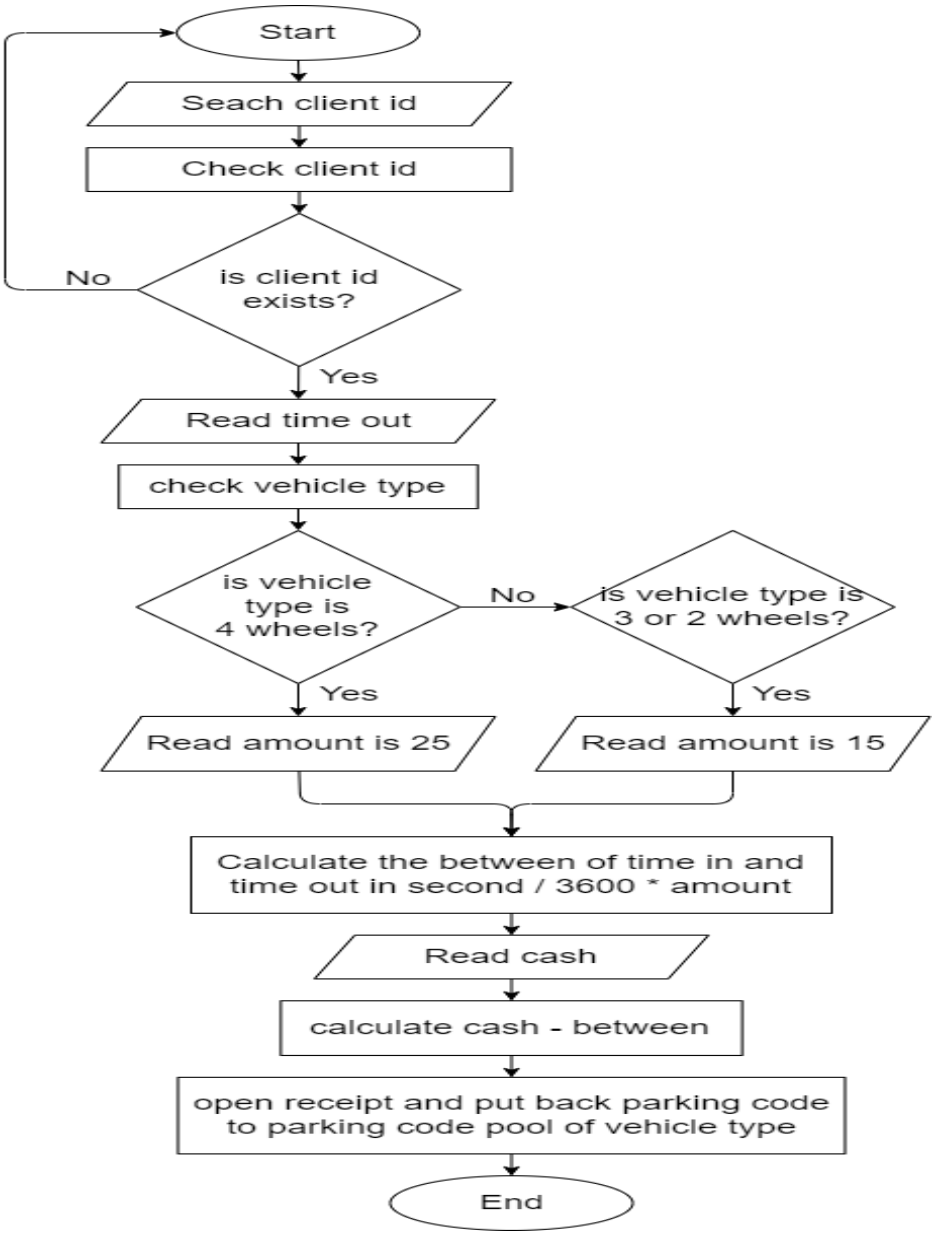

Figure 2.3: Time-out process flowchart

The figure above is the timing-out process flowchart. After following the procedures of the Figure 2.2, the next process is in figure 2.3. Timing-out the client only requires the client I.D. generated by the system. It is a four-digit code, randomly generated by the system. After fetching the client I.D., the system will show the time-in and what type of vehicle that the customer parked. In that data, the system will now decide whether it qualifies for the 4 wheels pricing or the 3- and 2-wheels pricing because it varies. Next, the system will now accumulate the accurate price judging from the time-in and time-out of the vehicle. The operator can now enter the denomination of the client's payment and the change will reflect in the system if there are any. To finish the timing-out process, the operator will print the receipt and hand it to the customer. And now the used parking space will be automatically available after the end of the process.

HARDWARE AND SOFTWARE RESOURCES

In this section, we describe the hardware and software resources used in the development and implementation of our pay-parking system, which utilizes the first-come, first-serve algorithm.

Table 1.1: Hardware Resources

The pay-parking system development was carried out on an HP Pavilion g6 laptop, which served as the primary computing device for programming and testing purposes. The laptop featured the following specifications:

RECOMMENDED HARDWARE SYSTEM SPECIFICATIONS
• Model: HP Pavilion g6 • Processor: Intel Core i5- 2410M (2.90 GHz, 2 cores, 4 threads) • RAM: 8 GB DDR3 • Storage: 256 GB SSD • Operating System: Windows 10 (64-bit)

Despite being a mid-range laptop, the HP Pavilion g6 provided sufficient computing power to support the development environment and run the necessary software components.

Table 1.2: Software Resources

RECOMMENDED SOFTWARE RESOURCES
• Netbeans 18.0 • Xampp

Frontend Development

For frontend development, the NetBeans Integrated Development Environment (IDE) was utilized. NetBeans offers a feature-rich and user-friendly environment for building web-based applications with HTML, CSS, and JavaScript.

IDE: NetBeans 18.0

Backend Development

The backend of the pay-parking system was implemented using JavaScript, specifically leveraging the Node.js runtime environment. Node.js enables server-side JavaScript execution, making it well-suited for building efficient and scalable web applications.

JavaScript**,** is an interpreted language that can calculate, modify, and validate data as well as update HTML and CSS. Because it is integrated with HTML, it is simpler to apply in web applications. It cannot, yet support multi-threading or multiprocessors, making it unsuitable for networking applications. JavaScript can be written without the use of a compiler using a simple text editor such as Notepad.

XAMPP, is one of the widely used cross-platform web servers, which helps developers to create and test their programs on a local webserver.

SQL, is a powerful database manipulation tool that allows users to create, retrieve, and alter existing records. It quickly gets massive amounts of data, executes insertion, deletion, and manipulation operations, and provides quick query processing to save time and ensure correctness. This efficient solution saves time and provides accurate data transmission without the need for hours of waiting.

Algorithm Description

The First Come, First Serve scheduling algorithm is an efficient and self-contained CPU scheduling approach that ensures the task that comes first in the queue is executed first. Since it only executes new processes when the current process is entirely processed by the CPU, this method is the simplest and easiest to implement. The scheduling technique is simple to comprehend and implement since it makes use of a Queue data structure, in which a new process enters through the tail of the queue and the scheduler chooses a process from the head.

On the other hand, it is not possible to process CPU when the previous one is not finished; those who come next must wait in the waiting area until the previous one's process is completed, at which point they can process again. It is not permitted to withdraw resources from a process before it has completed, and resources are switched when a running process completes and transfers to the waiting state.

Application of Algorithm

The first come, first served scheduling algorithm assures that the first vehicles to enter a pay parking system will be processed until finished. And if the parking spaces are full, the parking attendant will not accept another vehicle and process it. Even if a vehicle takes a little time in the parking lot than the previous one, it will not be processed because the prior vehicle must be processed first in the parking lot, and the other vehicles will have to wait in the next queue. This method restricts the number of cars that can park in a given spot, ensuring that the operation is efficient and effective.

CHAPTER 3

CONCLUSION AND RECOMMENDATION

Conclusion

In conclusion, a well-implemented pay-parking system offers three essential benefits to the parking community and surrounding areas. Firstly, by optimizing parking space utilization through efficient allocation using the first-come, first-serve algorithm, the system reduces waiting times and maximizes parking lot capacity. This leads to smoother traffic flow and decreased congestion, positively impacting the overall driving experience.

Secondly, the pay-parking system contributes to improved revenue generation for parking lot owners and operators. By implementing fair payment mechanisms, the system generates income that can be reinvested to enhance facilities, security measures, and overall user convenience. This financial sustainability ensures the continuous improvement of parking services and facilities.

Lastly, the pay-parking system prioritizes user satisfaction by offering a user-friendly experience. With quick and easy entry and exit processes, and clear guidance for finding available parking spaces, the system minimizes driver frustration and promotes a positive parking experience. This, in turn, fosters customer loyalty and favorable word-of-mouth recommendations, strengthening the reputation and success of the parking establishment.

In combination, these three benefits underscore the significance of a well-designed pay-parking system in promoting efficient parking management, contributing to a smoother urban mobility ecosystem, and enhancing the overall satisfaction of both parking lot users and operators.

Recommendation

The study's findings have led the researchers to recommend the following course of action:

1. Incorporating real-time updates on parking space availability through digital signage or a dedicated mobile app can significantly impact user satisfaction. By providing drivers with accurate information on available parking spots, we can reduce traffic congestion and enhance the overall parking experience. This feature enables drivers to make informed decisions and directs them to open spaces promptly, optimizing parking space utilization.
2. As we explore ways to improve the pay-parking system, introducing contactless payment options emerges as a promising solution. With the rising demand for touchless transactions, incorporating mobile payment apps or card-based payments can enhance payment convenience and promote a safer environment, particularly in the context of public health concerns.
3. A parking reservation system offers a valuable opportunity to enhance user convenience and predictability. By allowing users to pre-book parking spaces, especially during busy hours or events, we can minimize uncertainties and ensure a smooth parking experience. This feature has the potential to attract more users and improve overall customer satisfaction.
4. A dedicated mobile app tailored to the pay-parking system can be a game-changer. Integrating various functionalities, such as real-time availability updates, contactless payments, and loyalty programs, into a user-friendly app can significantly improve engagement and user experience. This integration fosters greater convenience and encourages users to make the most of the parking facilities.

CHAPTER 4

USER MANUAL AND CODE

This pay-parking system user manual will guide users on using the research tool, while the Code Introduction provides a brief overview of the algorithm's implementation.

Figure 3.1: Log-in Interface

Figure 3.1 will show first as you run the system. Here is where the administrator and operator will log-n their accounts.

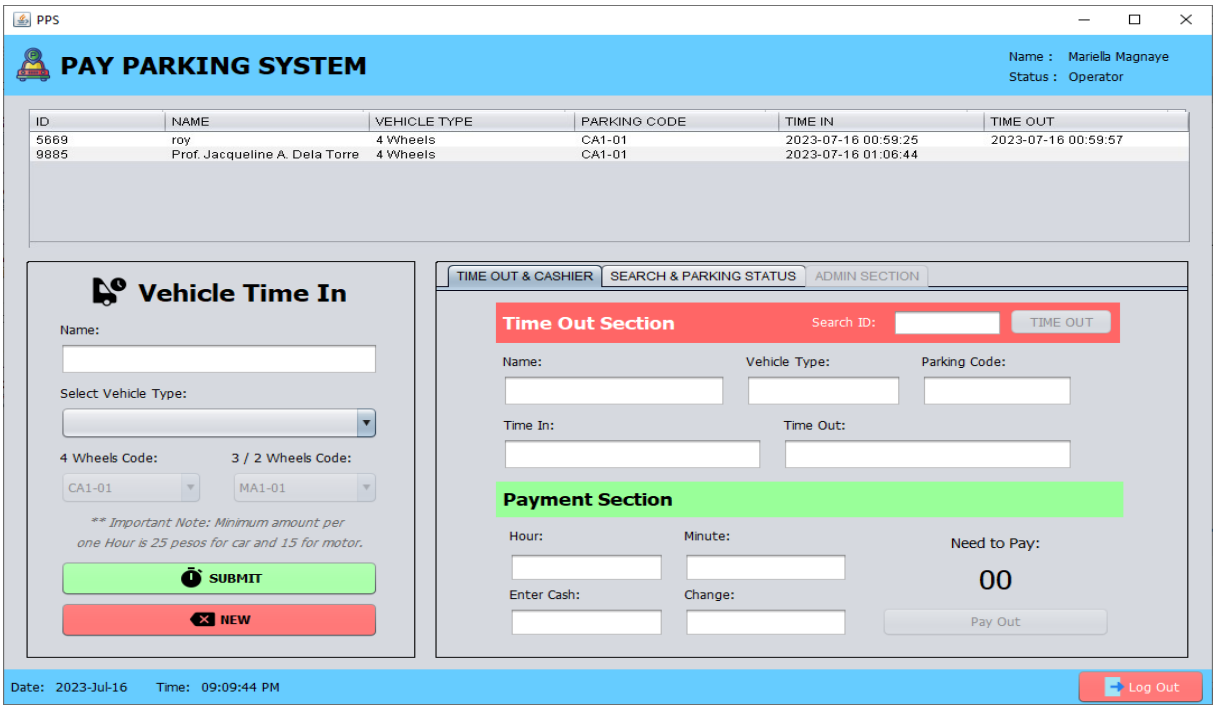

Figure 3.2: Operator Dashboard

Once the Operator login their account, the operator dashboard will then be displayed. Which where it contains the Vehicle Time-in & Time-out section, Payment section, and Search & parking status tab.

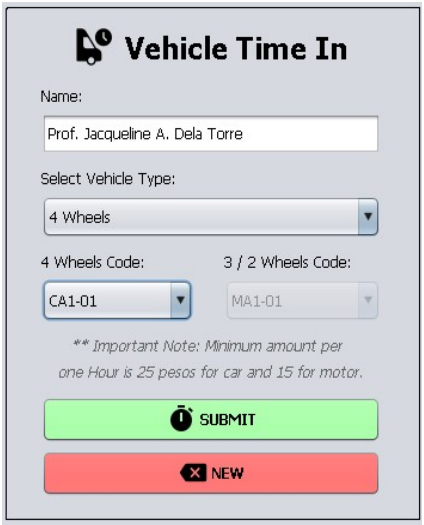

Figure 3.3: Time-in section

In the figure above, is where the operator accumulate the name and the vehicle type of the client to be recorded in the system in order for them to be eligible to park. There are 2 buttons present, the “Submit“ and “New” buttons. Once the “submit” button is clicked, the displayed information will be recorded in the system and will reflect in the table above, while the “new” button will refresh the text field and drop-down boxes.

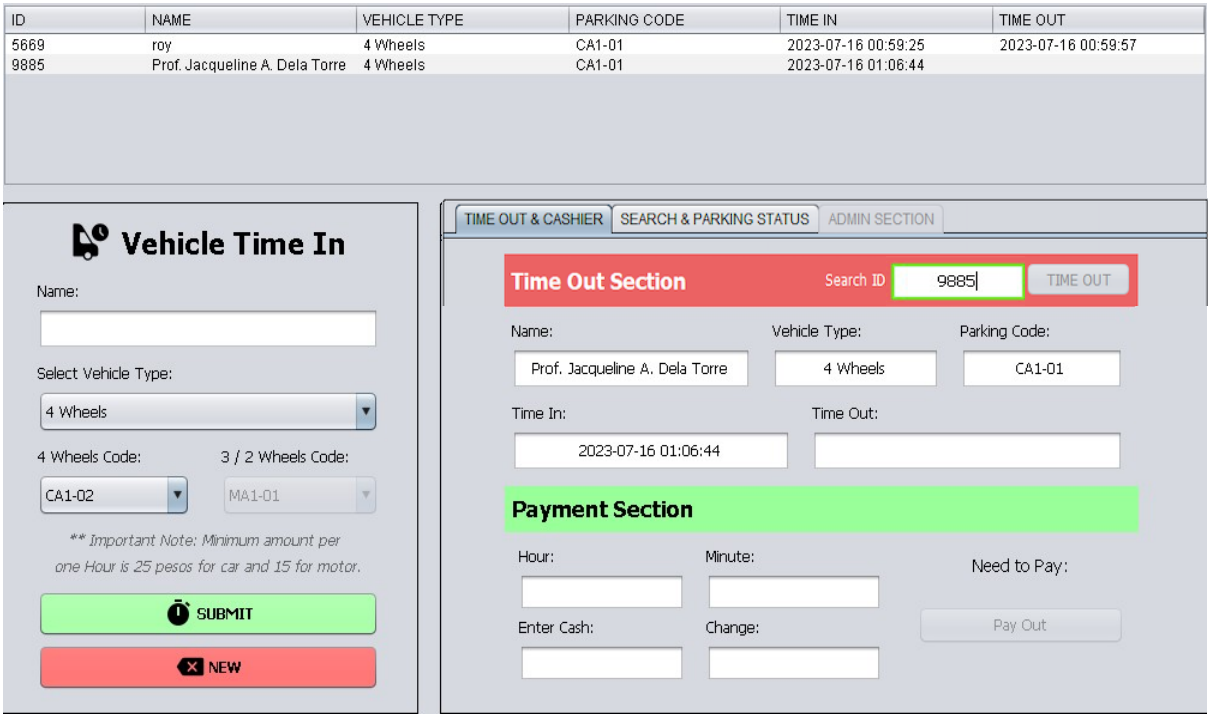

Figure 3.4: Time-out section

The Time-out section in the figure above is where you will easily fetch the data of the client heading out. Only the Search I.D. text box is clickable, you cannot alter the information in the displayed information once it was submitted. In that said text-box, is where you will search the client I.D. to be provided by the client once they are ready to time-out.

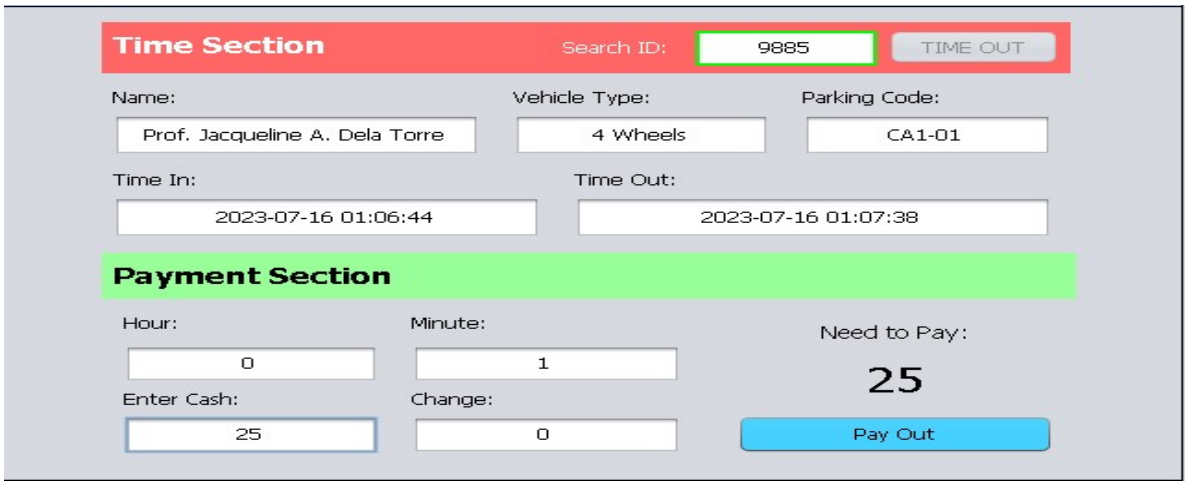

Figure 3.5: Payment section

The figure above will show the hours and minutes used by the client's parked vehicle, and it will automatically generate the price based on the parking venue's criteria. The operator will enter the cash tendered by the client and the system will calculate to give the right change.

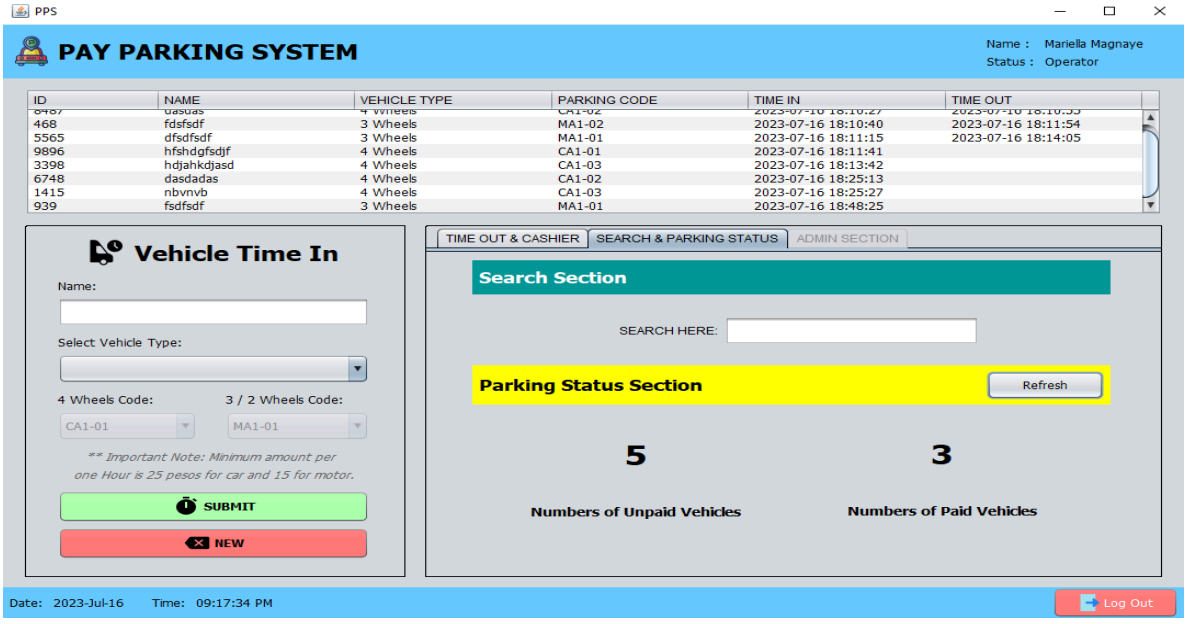

Figure 3.6: Search & Parking status tab

The Search & Parking status tab is like the main search engine of the system. It can search any vehicle code to see it status.

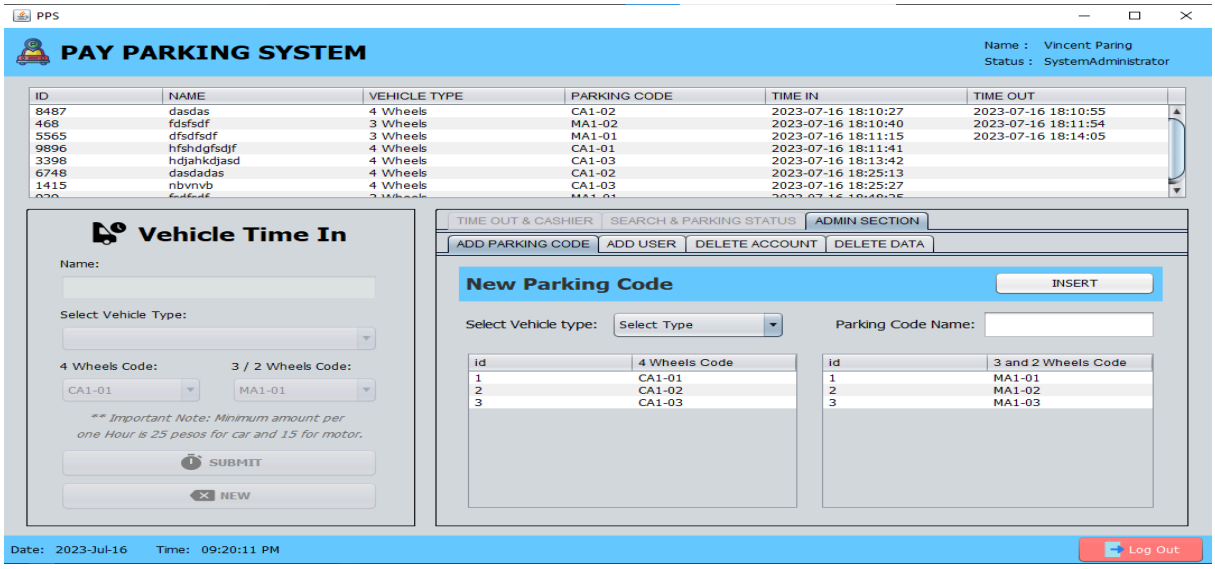

Figure 3.7.1: Admin dashboard

In the Admin dashboard, the admin can add parking code, user, delete account, and data. It cannot use the operator dashboard functions.

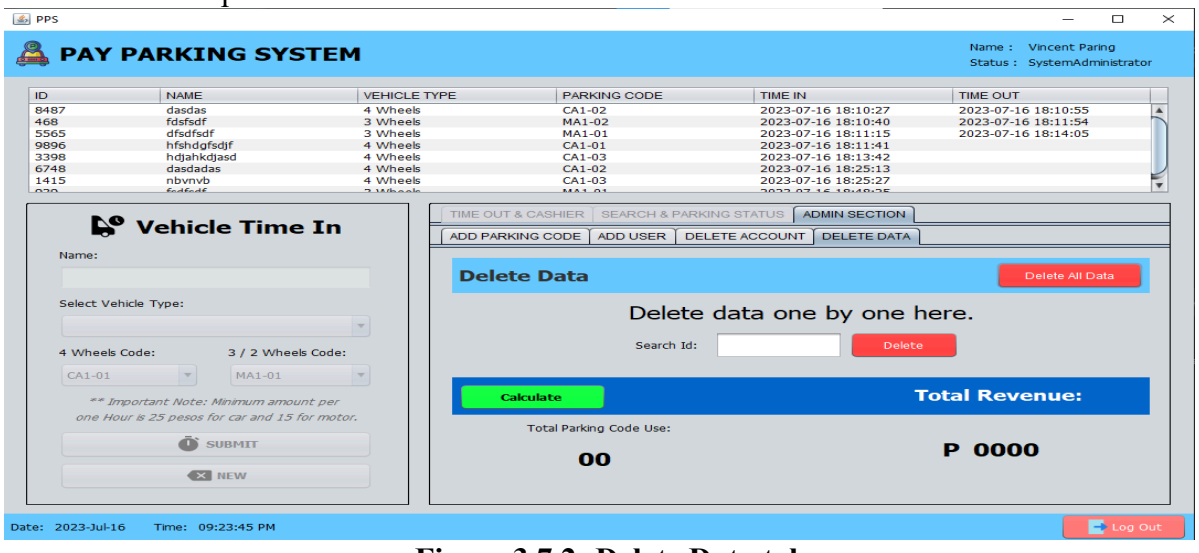

Figure 3.7.2: Delete Data tab

In this tab, the administrator can delete any data that has been recorded in the system at the table above it.

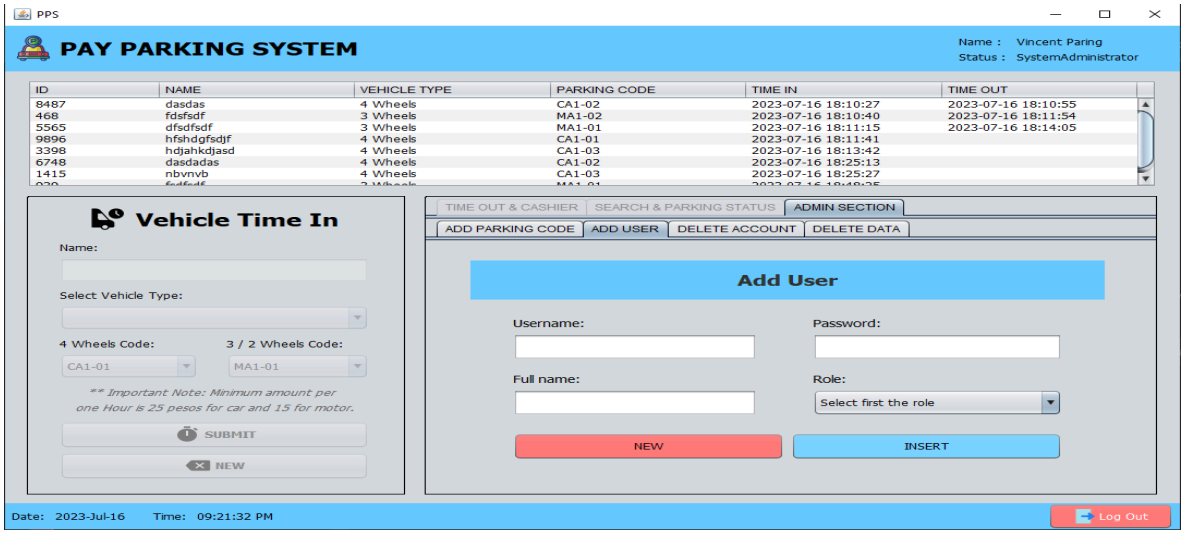

Figure 3.7.3: Add user tab

In the frame above, the administrator can add users of the system and give their roles.

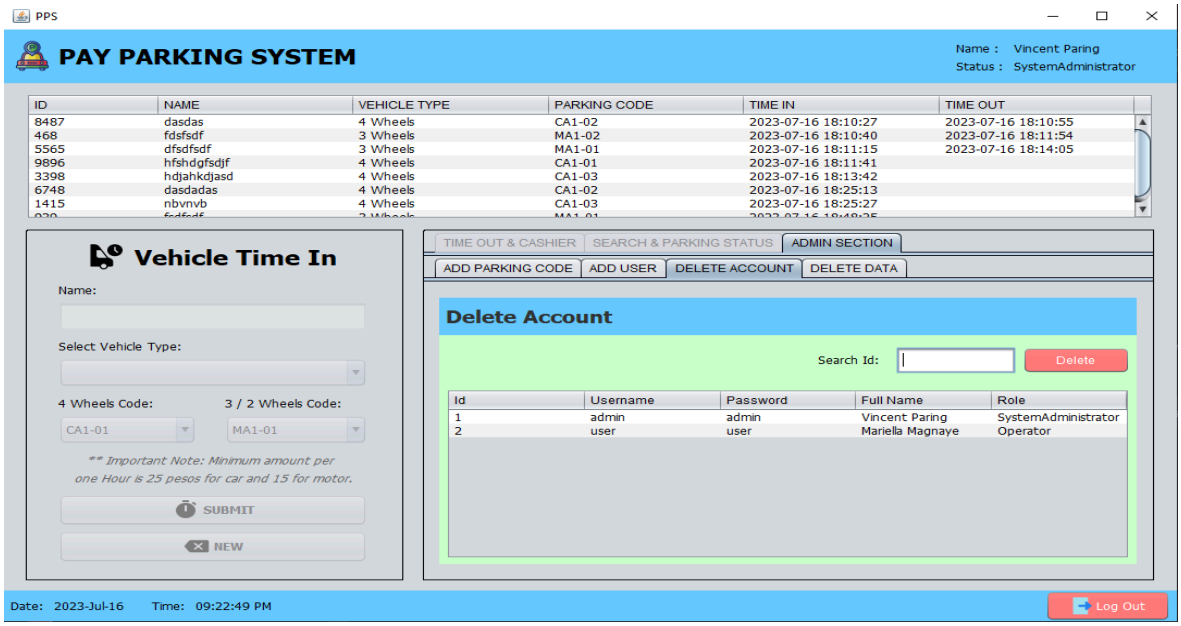

Figure 3.7.4: Delete account tab

This is where the administrator can remove/delete accounts also made by the administrator.

Code

```
public final class Dashboard extends javax.swing.JFrame {

    public static String fName;
    public static String Role;
    DefaultTableModel model;

    public Dashboard() {
        initComponents();
    }

    public Dashboard(String fname, String role) {
        initComponents();
        fName = fname;
        Role = role;
        carItem();
        motorItem();
        carFetch();
        motorFetch();
        AccountFetch();
        dt();
        times();

        userInfo();
        roleChecker();
        SearchAndFetch();
        Fetch();
    }
```

```
    public void dt() {
        Date d = new Date();

        SimpleDateFormat sdf = new SimpleDateFormat(pattern:"yyyy-MMM-dd");

        String dd = sdf.format(date:d);
        l_date.setText(text:dd);
    }

    Timer t;
    SimpleDateFormat st;

    public void times() {
        t = new Timer(delay:0, (ActionEvent e) -> {
            Date dt = new Date();
            st = new SimpleDateFormat(pattern:"hh:mm:ss a");

            String tt = st.format(date:dt);
            l_time.setText(text:tt);
        });
        t.start();
    }
```

```
    public void userInfo(){
        lblFullname.setText(text: fName);
        lblStatus.setText(text: Role);
    }

    public void roleChecker(){
        switch(Role) {
            case "SystemAdministrator" -> {
                tabPane.setEnabledAt(index: 0, enabled:false);
                tabPane.setEnabledAt(index: 1, enabled:false);
                tabPane.setSelectedIndex(index: 2);
                btnSubmit.setEnabled(b: false);
                btnNew.setEnabled(b: false);
                boxVehicle.setEnabled(b: false);
                txtName.setEnabled(enabled:false);
            }
            case "Operator" -> {
                tabPane.setSelectedIndex(index: 0);
                tabPane.setEnabledAt(index: 2, enabled:false);
            }
            default -> {
            }
        }
    }
```

```
public void carItem() {
    try {
        PreparedStatement prl = Connection.conn.prepareStatement(string: "SELECT * From car_code_tbl");
        java.sql.ResultSet rsl = prl.executeQuery();

        while(rsl.next()) {
            boxCar.addItem(item:rsl.getString(string: "car"));
        }
    } catch (SQLException e) {  }
}

public void motorItem() {
    try {
        PreparedStatement prl = Connection.conn.prepareStatement(string: "SELECT * From motor_code_tbl");
        java.sql.ResultSet rsl = prl.executeQuery();
        while(rsl.next()) {
            boxMotor.addItem(item:rsl.getString(string: "motor"));
        }
    } catch (SQLException e) {  }
}
```

```
    private void SearchAndFetch() {
        try {
            Connection.pst = Connection.conn.prepareStatement(string: "SELECT * FROM clients_tbl");
            Connection.rs = Connection.pst.executeQuery();

            while(Connection.rs.next()) {
                String id = Connection.rs.getString(string: "client_id");
                String date = Connection.rs.getString(string: "client_name");
                String prog = Connection.rs.getString(string: "vehicle_type");
                String fname = Connection.rs.getString(string: "parking_code");
                String addr = Connection.rs.getString(string: "time_in");
                String cont = Connection.rs.getString(string: "time_out");

                Object[] obj = {
                    id, date, prog, fname, addr, cont
                };

                model = (DefaultTableModel)tblMonitor.getModel();
                model.addRow(rowData:obj);
            }
        } catch (SQLException ex) {  }
    }

    public void search(String str) {
        model = (DefaultTableModel)tblMonitor.getModel();
        TableRowSorter<DefaultTableModel> trs = new TableRowSorter<>(model);
        tblMonitor.setRowSorter(sorter: trs);
        trs.setRowFilter(filter: RowFilter.regexFilter(regex: str));
    }
```

```
private void Fetch() {
    try {
        int q;

        Connection.pst = Connection.conn.prepareStatement(string: "SELECT * FROM clients_tbl");
        Connection.rs = Connection.pst.executeQuery();
        ResultSetMetaData rss = Connection.rs.getMetaData();
        q = rss.getColumnCount();

        DefaultTableModel df = (DefaultTableModel)tblMonitor.getModel();
        df.setRowCount(rowCount: 0);

        while(Connection.rs.next()) {
            Vector v2 = new Vector();
            for(int a = 1; a <= q; a++) {
                v2.add(e: Connection.rs.getString(string: "client_id"));
                v2.add(e: Connection.rs.getString(string: "client_name"));
                v2.add(e: Connection.rs.getString(string: "vehicle_type"));
                v2.add(e: Connection.rs.getString(string: "parking_code"));
                v2.add(e: Connection.rs.getString(string: "time_in"));
                v2.add(e: Connection.rs.getString(string: "time_out"));
            }
            df.addRow(rowData:v2);
        }
    } catch (SQLException ex) {  }
}
```

```
private void carFetch() {
    try {
        int q;

        Connection.pst = Connection.conn.prepareStatement(string: "SELECT * FROM car_code_tbl");
        Connection.rs = Connection.pst.executeQuery();
        ResultSetMetaData rss = Connection.rs.getMetaData();
        q = rss.getColumnCount();

        DefaultTableModel df = (DefaultTableModel)tbl4W.getModel();
        df.setRowCount(rowCount: 0);

        while(Connection.rs.next()) {
            Vector v2 = new Vector();
            for(int a = 1; a <= q; a++) {
                v2.add(e: Connection.rs.getString(string: "id"));
                v2.add(e: Connection.rs.getString(string: "car"));
            }
            df.addRow(rowData:v2);
        }
    } catch (SQLException ex) {  }
}
```

```
private void motorFetch() {
    try {
        int q;

        Connection.pst = Connection.conn.prepareStatement(string: "SELECT * FROM motor_code_tbl");
        Connection.rs = Connection.pst.executeQuery();
        ResultSetMetaData rss = Connection.rs.getMetaData();
        q = rss.getColumnCount();

        DefaultTableModel df = (DefaultTableModel)tbl32W.getModel();
        df.setRowCount(rowCount: 0);

        while(Connection.rs.next()) {
            Vector v2 = new Vector();
            for(int a = 1; a <= q; a++) {
                v2.add(e: Connection.rs.getString(string: "id"));
                v2.add(e: Connection.rs.getString(string: "motor"));
            }
            df.addRow(rowData:v2);
        }
    } catch (SQLException ex) {  }
}
```

```
private void AccountFetch() {
    try {
        int q;

        Connection.pst = Connection.conn.prepareStatement(string: "SELECT * FROM user_account");
        Connection.rs = Connection.pst.executeQuery();
        ResultSetMetaData rss = Connection.rs.getMetaData();
        q = rss.getColumnCount();

        DefaultTableModel df = (DefaultTableModel)tblAccount.getModel();
        df.setRowCount(rowCount: 0);

        while(Connection.rs.next()) {
            Vector v2 = new Vector();
            for(int a = 1; a <= q; a++) {
                v2.add(e: Connection.rs.getString(string: "id"));
                v2.add(e: Connection.rs.getString(string: "username"));
                v2.add(e: Connection.rs.getString(string: "password"));
                v2.add(e: Connection.rs.getString(string: "fullname"));
                v2.add(e: Connection.rs.getString(string: "role"));
            }
            df.addRow(rowData:v2);
        }
    } catch (SQLException ex) {  }
}
```

```
private void btnLogOffActionPerformed(java.awt.event.ActionEvent evt) {
    int result = JOptionPane.showConfirmDialog(parentComponent:null, message:"Do you want to leave?",
                title: "Log Out?",
              optionType: JOptionPane.YES_NO_OPTION,
            messageType:JOptionPane.QUESTION_MESSAGE);

    if(result == JOptionPane.YES_OPTION) {
        Login in = new Login();
        in.show();
        this.hide();
    }
}
```

```
private void boxVehicleActionPerformed(java.awt.event.ActionEvent evt) {
    int type = boxVehicle.getSelectedIndex();

    switch(type) {
        case 1 -> {
            boxCar.setEnabled(b: true);
            boxMotor.setEnabled(b: false);
            if(boxCar.getSelectedItem() == null) {
                JOptionPane.showMessageDialog(parentComponent:this, message:"No available parking slots for 4 wheels!",
                        title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
                boxVehicle.setSelectedIndex(anIndex:0);
            }
        }
        case 2, 3 -> {
            boxCar.setEnabled(b: false);
            boxMotor.setEnabled(b: true);
            if(boxMotor.getSelectedItem() == null) {
                JOptionPane.showMessageDialog(parentComponent:this, message:"No available parking slots for 3 or 2 wheels!",
                        title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
                boxVehicle.setSelectedIndex(anIndex:0);
            }
        }
        default -> {
            boxMotor.setEnabled(b: false);
            boxCar.setEnabled(b: false);
        }

    }
}
```

```
private void btnSubmitActionPerformed(java.awt.event.ActionEvent evt) {
    Random ran = new Random();
    String num = Integer.toString(i: ran.nextInt(bound: 10000));

    String vehicle = (String) boxVehicle.getSelectedItem();
    int type = boxVehicle.getSelectedIndex();
    String name = txtName.getText();

    if(!name.isEmpty()) {
        switch(type) {
            case 1 -> {
                String code = (String) boxCar.getSelectedItem();
                int index = boxCar.getSelectedIndex();
```

```
if(code == null) {
    JOptionPane.showMessageDialog(parentComponent:this, message:"Parking code for 4 wheels is empty!!",
            title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
}else {
    try {
        Connection.pst = Connection.conn.prepareStatement("INSERT INTO clients_tbl (client_id, client_name, vehicle_type, parking_code, category, time_in,
                "VALUES ('"+num+"', '"+name+"', '"+vehicle+"', '"+code+"', 'Car Area', CURRENT_TIMESTAMP, '1')");

        if(Connection.pst.executeUpdate() == 1) {
            boxCar.removeItemAt(anIndex:index);

            Connection.pst = Connection.conn.prepareStatement("SELECT * FROM clients_tbl WHERE client_id = '"+num+"'");
            Connection.rs = Connection.pst.executeQuery();

            if(Connection.rs.next() == true) {
                JOptionPane.showMessageDialog(parentComponent:this,
                    """
                    Parking Inc.
                    Client ID: \t""" + Connection.rs.getString(string: "client_id") + "\nClient Name: " + Connection.rs.getString(string: "client_name") +
                            "\nVehicle Type: " + Connection.rs.getString(string: "vehicle_type") + "\nParking Code: " + Connection.rs.getString(string: "parkin
                            "\nCategory: " + Connection.rs.getString(string: "category") + "\nTime in: " + Connection.rs.getString(string: "time_in") + "\nPark
                    title: "Official Receipt", messageType:JOptionPane.INFORMATION_MESSAGE);
                }
                txtName.setText(s: null);
                Fetch();
            } else {
                JOptionPane.showMessageDialog(parentComponent:this, message:"Record Failed to Save!!");
            }
        } catch (SQLException ex) {
            Logger.getLogger(name: Dashboard.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
        }
}
```

```
case 2, 3 -> {
    String code = (String) boxMotor.getSelectedItem();
    int index = boxMotor.getSelectedIndex();

    if(code == null) {
        JOptionPane.showMessageDialog(parentComponent:this, message:"Parking code for 3 / 2 wheels is empty!!", title: "Error", messageType:JOptionPane.ERROR_MESSAGE)
    }else {
        try {
            Connection.pst = Connection.conn.prepareStatement("INSERT INTO clients_tbl (client_id, client_name, vehicle_type, parking_code, category, time_
                    "VALUES ('"+num+"', '"+name+"', '"+vehicle+"', '"+code+"', 'Motor Area', CURRENT_TIMESTAMP, '1')");

            if(Connection.pst.executeUpdate() == 1) {
                boxMotor.removeItemAt(anIndex:index);
                Connection.pst = Connection.conn.prepareStatement("SELECT * FROM clients_tbl WHERE client_id = '"+num+"'");
                Connection.rs = Connection.pst.executeQuery();

                if(Connection.rs.next() == true) {
                    JOptionPane.showMessageDialog(parentComponent:this,
                        """
                        Parking Inc.
                        Client ID: \t""" + Connection.rs.getString(string: "client_id") + "\nClient Name: " + Connection.rs.getString(string: "client_name") +
                                "\nVehicle Type: " + Connection.rs.getString(string: "vehicle_type") + "\nParking Code: " + Connection.rs.getString(string: "pa
                                "\nCategory: " + Connection.rs.getString(string: "category") + "\nTime in: " + Connection.rs.getString(string: "time_in") + "\n
                        title: "Official Receipt", messageType:JOptionPane.INFORMATION_MESSAGE);
                    }
                    txtName.setText(s: null);
                    Fetch();
                } else {
                    JOptionPane.showMessageDialog(parentComponent:this, message:"Record Failed to Save!!");
                }
            } catch (SQLException ex) {
                Logger.getLogger(name: Dashboard.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
            }
        }
}
```

```
    private void btnNewActionPerformed(java.awt.event.ActionEvent evt) {
        txtName.setText(t: null);
        boxVehicle.setSelectedIndex(anIndex:0);
    }

    private void txtSearchKeyReleased(java.awt.event.KeyEvent evt) {
        String src = txtSearch.getText();
        search(str:src);
    }

    private void txtCashKeyReleased(java.awt.event.KeyEvent evt) {
        int payment = Integer.parseInt(s: lblPay.getText());
        int cash = Integer.parseInt(s: txtCash.getText());

        int total = cash - payment;
        btnPayment.setEnabled(b: false);

        if(total > 0 || cash == Integer.parseInt(s: lblPay.getText())) {
            txtChange.setText(t: Integer.toString(i: total));
            btnPayment.setEnabled(b: true);
        }else {
            txtChange.setText(t: "");
        }
    }
```

```
private void btnPaymentActionPerformed(java.awt.event.ActionEvent evt) {
    String id = txtClientId.getText(); String tk = txtHour.getText() + " Hr and " + txtMinute.getText() + " Min"; String tm = lblPay.getText();
    try {
        Connection.pst = Connection.conn.prepareStatement("UPDATE clients_tbl SET time_taken = '"+tk+"', total_amount = '"+tm+"' WHERE client_id = '"+id+"'
        if(Connection.pst.executeUpdate() == 1) {
            Connection.pst = Connection.conn.prepareStatement("SELECT * FROM clients_tbl WHERE client_id = '"+id+"'");
            Connection.rs = Connection.pst.executeQuery();

            if(Connection.rs.next() == true) {
                String cate = Connection.rs.getString(string: "category");
                int amt;
                if(cate.equals(anObject:"Car Area")) { boxCar.addItem(item:txtCPC.getText()); amt = 25;
                }else { boxMotor.addItem(item:txtCPC.getText()); amt = 15; }

                JOptionPane.showMessageDialog(parentComponent:this,
                    """
                    Parking Inc.
                    Client ID: \t""" + Connection.rs.getString(string: "client_id") + "\nClient Name: " + Connection.rs.getString(string: "client_name") +
                    "\nVehicle Type: " + Connection.rs.getString(string: "vehicle_type") + "\nParking Code: " + Connection.rs.getString(string: "parking_code")
                    "\nCategory: " + cate + "\nTime in: " + Connection.rs.getString(string: "time_in") + "\nTime out: " + Connection.rs.getString(string: "time
                    "\nParking Price: " + amt + "\nTime Taken: " + Connection.rs.getString(string: "time_taken") + "\nAmount to Pay: " + Connection.rs.getStr
                    "\nCustomer Cash: " + txtCash.getText() + "\nChange: " + txtChange.getText(),
                    title: "Official Receipt", messageType:JOptionPane.INFORMATION_MESSAGE);
            }
            txtClientId.setText(s: null); txtClientId.setEditable(b: true); txtClientId.setBorder(new LineBorder(color: Color.black,thickness: 0));
            btnOut.setEnabled(b: false); txtCName.setText(s: null); txtCVT.setText(s: null); txtCPC.setText(s: null); txtIn.setText(s: null);
            txtOut.setText(s: null); txtHour.setText(s: null); txtMinute.setText(s: null); lblPay.setText(text: null); txtCash.setText(s: null);
            txtCash.setEditable(b: false); txtChange.setText(s: null); btnPayment.setEnabled(b: false);
        }else {
            JOptionPane.showMessageDialog(parentComponent:this, message:"Error occured!!", title: "Record Error", messageType:JOptionPane.ERROR_MESSAGE);
        }
    } catch (SQLException ex) {  }
}
```

```
private void btnOutActionPerformed(java.awt.event.ActionEvent evt) {
    String cId = txtClientId.getText();
    if(cId.isEmpty()) {
        JOptionPane.showMessageDialog(parentComponent:this, message:"Error!!", title: "Record Error", messageType:JOptionPane.ERROR_MESSAGE);
        btnOut.setEnabled(b: false);
    }else {
        try {
            Connection.pst = Connection.conn.prepareStatement("UPDATE clients_tbl SET time_out = CURRENT_TIMESTAMP, status = '0' WHERE client_id = '"+cId+

            if(Connection.pst.executeUpdate() == 1) {
                JOptionPane.showMessageDialog(parentComponent:this, message:"Time Out Successfully!!", title: "Record", messageType:JOptionPane.INFORMATION_MESSAGE);

                Connection.pst = Connection.conn.prepareStatement("SELECT * FROM clients_tbl WHERE client_id = '"+cId+"'");
                Connection.rs = Connection.pst.executeQuery();
                if(Connection.rs.next() == true) {
                    txtOut.setText(s: Connection.rs.getString(i: 8));
                    String category = Connection.rs.getString(i: 6);
                    Fetch(); String amount; PreparedStatement sql;

                    if(category.equals(anObject:"Motor Area")) { amount = "15";
                        sql = Connection.pst = Connection.conn.prepareStatement("SELECT timestampdiff(hour, time_in, time_out) hours, round((timestampdiff
                    }else { amount = "25";
                        sql = Connection.pst = Connection.conn.prepareStatement("SELECT timestampdiff(hour, time_in, time_out) hours, round((timestampdiff
                    }

                     Connection.rs = sql.executeQuery();
                     if(Connection.rs.next() == true) {
                         int hour = Integer.parseInt(s: Connection.rs.getString(i: 1));
                         if(hour < 1) {
                             txtHour.setText(s: Connection.rs.getString(i: 1));
                             txtMinute.setText(s: Connection.rs.getString(i: 2));
                             lblPay.setText(text: amount);
                         }else {
                             txtHour.setText(s: Connection.rs.getString(i: 1));
                             txtMinute.setText(s: Connection.rs.getString(i: 2));
                             lblPay.setText(text: Connection.rs.getString(i: 3));
                         }
                     }
                     txtCash.setEditable(b: true); txtClientId.setEditable(b: false); btnOut.setEnabled(b: false);
                 }
             }
         } catch (SQLException e) { }
    }
}
```

```
private void btnInsActionPerformed(java.awt.event.ActionEvent evt) {
    int sIn = boxNewPc.getSelectedIndex();
    String cnew = txtPCN.getText();

    switch(sIn) {
        case 0 -> {
            JOptionPane.showMessageDialog(parentComponent:this, message:"Select or input new code name first!",
                    title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
        }
        case 1 -> {
            if(!txtPCN.getText().isEmpty()) {
                try {
                    Connection.pst = Connection.conn.prepareStatement("INSERT INTO car_code_tbl (car) VALUES ('"+cnew+"')");
                    if(Connection.pst.executeUpdate() == 1) {
                        JOptionPane.showMessageDialog(parentComponent:this, message:"Inserted Successfully!",
                                title: "New Parking Code", messageType:JOptionPane.INFORMATION_MESSAGE);
                        boxNewPc.setSelectedIndex(anIndex:0);
                        txtPCN.setText(t: null);
                        carFetch();
                    }
                } catch(SQLException ex) {}
            }else {
                JOptionPane.showMessageDialog(parentComponent:this, message:"Select or input new code name first!",
                        title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
            }
        }
```

```
private void btnNewAccActionPerformed(java.awt.event.ActionEvent evt) {
    if((txtNewAccUname.getText().isEmpty()) || (txtNewAccPass.getText().isEmpty()) || (txtNewAccFname.getText().isEmpty())) {
        JOptionPane.showMessageDialog(parentComponent:this, message:"all fields are required!", title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
    }else {
        int newAcc = boxNewAccRole.getSelectedIndex();
        switch(newAcc) {
            case 0 -> {
                JOptionPane.showMessageDialog(parentComponent:this, message:"Reselect role type required!",
                        title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
            }
            case 1, 2 -> {
                try {
                    Connection.pst = Connection.conn.prepareStatement("INSERT INTO user_account (username, password, fullname, role)" +
                            "VALUES ('"+txtNewAccUname.getText()+"', '"+txtNewAccPass.getText()+"', '"+txtNewAccFname.getText()+"', '"+boxNewAccRole.getSel
                    if(Connection.pst.executeUpdate() == 1) {
                        JOptionPane.showMessageDialog(parentComponent:this, message:"New Account Inserted Successfully!",
                                title: "New Account", messageType:JOptionPane.INFORMATION_MESSAGE);
                        txtNewAccUname.setText(s: null);
                        txtNewAccPass.setText(s: null);
                        txtNewAccFname.setText(s: null);
                        boxNewAccRole.setSelectedIndex(anIndex:0);
                        AccountFetch();
                    }
                } catch(SQLException ex) {}
            }
        }
    }
}
```

```
private void txtClientIdKeyReleased(java.awt.event.KeyEvent evt) {
    String cId = txtClientId.getText();
    try {
        Connection.pst = Connection.conn.prepareStatement("SELECT * FROM clients_tbl WHERE client_id = '"+cId+"'");
        Connection.rs = Connection.pst.executeQuery();
        if(Connection.rs.next() == true) {
            txtClientId.setBorder(new LineBorder(color: Color.green,thickness: 2));
            txtClientId.setForeground(fg: Color.black);
            txtCName.setText(s: Connection.rs.getString(i: 3));
            txtCVT.setText(s: Connection.rs.getString(i: 4));
            txtCPC.setText(s: Connection.rs.getString(i: 5));
            txtIn.setText(s: Connection.rs.getString(i: 7));
            txtOut.setText(s: Connection.rs.getString(i: 8));
            btnOut.setEnabled(b: false);

            if(txtOut.getText().isEmpty()) {
                btnOut.setEnabled(b: true);
            }
        }else {
            txtClientId.setBorder(new LineBorder(color: Color.red,thickness: 2));
            txtClientId.setForeground(fg: Color.red);
            txtCName.setText(s: null); txtCVT.setText(s: null);
            txtCPC.setText(s: null); txtIn.setText(s: null);
            txtOut.setText(s: null); txtHour.setText(s: null);
            txtMinute.setText(s: null); lblPay.setText(text: null);
        }
    } catch (SQLException ex) {}
}
```

```
            case 2 -> {
                if(!txtPCN.getText().isEmpty()) {
                    try {
                        Connection.pst = Connection.conn.prepareStatement("INSERT INTO motor_code_tbl (motor) VALUES ('"+cnew+"')");
                        if(Connection.pst.executeUpdate() == 1) {
                            JOptionPane.showMessageDialog(parentComponent:this, message:"Inserted Successfully!",
                                    title: "New Parking Code", messageType:JOptionPane.INFORMATION_MESSAGE);
                            boxNewPc.setSelectedIndex(anIndex:0);
                            txtPCN.setText(t: null);
                            motorFetch();
                        }
                    } catch(SQLException ex) {}
                }else {
                    JOptionPane.showMessageDialog(parentComponent:this, message:"Select or input new code name first!",
                            title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
                }
            }
        }

    }
```

```
private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {
    if(!txtDelAcc.getText().isEmpty()) {
        int result = JOptionPane.showConfirmDialog(parentComponent:null, message:"Do you want to delete this Account?",
                title: "Delete Account",
                optionType: JOptionPane.YES_NO_OPTION,
                messageType:JOptionPane.QUESTION_MESSAGE);

        if(result == JOptionPane.YES_OPTION) {
            try {
                Connection.pst = Connection.conn.prepareStatement("DELETE FROM user_account WHERE id = '"+txtDelAcc.getText()+"'");

                if(Connection.pst.executeUpdate() == 1) {
                    JOptionPane.showMessageDialog(parentComponent:this, message:"Account has been successfully deleted!!",
                            title: "Account Record", messageType:JOptionPane.INFORMATION_MESSAGE);
                    AccountFetch();
                }else {
                    JOptionPane.showMessageDialog(parentComponent:this, message:"Account Failed to Delete!!\nPlease check your Inputed ID.");
                }
            } catch (SQLException ex) {}
        }else {
            txtDelAcc.setText(t: null);
        }
    }else {
        JOptionPane.showMessageDialog(parentComponent:this, message:"Id field is required!", title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
    }
}
```

```
private void btnEraseActionPerformed(java.awt.event.ActionEvent evt) {
    txtNewAccUname.setText(t: null);
    txtNewAccPass.setText(t: null);
    txtNewAccFname.setText(t: null);
    boxNewAccRole.setSelectedIndex(anIndex:0);
}
```

```
private void txtDelAccKeyTyped(java.awt.event.KeyEvent evt) {
    char c = evt.getKeyChar();
    if(!Character.isDigit(ch: c)) {
        evt.consume();
    }
}
```

```
private void btnDItemActionPerformed(java.awt.event.ActionEvent evt) {
    int result = JOptionPane.showConfirmDialog(parentComponent:null, message:"Do you want to delete this Record?",
            title: "Delete Record",
            optionType: JOptionPane.YES_NO_OPTION,
            messageType:JOptionPane.QUESTION_MESSAGE);
    if(result == JOptionPane.YES_OPTION) {
        try {
            Connection.pst = Connection.conn.prepareStatement("DELETE FROM clients_tbl WHERE client_id = '"+txtSItem.getText()+"'");

            if(Connection.pst.executeUpdate() == 1) {
                JOptionPane.showMessageDialog(parentComponent:this, message:"Record has been successfully deleted!!",
                        title: "Delete Record", messageType:JOptionPane.INFORMATION_MESSAGE);
                Fetch();
            } else {
                JOptionPane.showMessageDialog(parentComponent:this, message:"Record Failed to Delete!!\nPlease check your Inputed ID.");
            }
        } catch (SQLException ex) {}
    }else {

    }
}
```

```
private void txtCashKeyTyped(java.awt.event.KeyEvent evt) {
    char c = evt.getKeyChar();
    if(!Character.isDigit(ch: c)) {
        evt.consume();
    }
}

private void txtClientIdKeyTyped(java.awt.event.KeyEvent evt) {
    char c = evt.getKeyChar();
    if(!Character.isDigit(ch: c)) {
        evt.consume();
    }
}

private void txtDelAccKeyReleased(java.awt.event.KeyEvent evt) {
    String src = txtDelAcc.getText();
    searchAcc(str:src);
}

private void txtSItemKeyReleased(java.awt.event.KeyEvent evt) {
    String src = txtSItem.getText();
    search(str:src);
}
```

```
private void btnRevActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        Connection.pst = Connection.conn.prepareStatement(string: "SELECT COUNT(parking_code) FROM clients_tbl");
        Connection.rs = Connection.pst.executeQuery();
        if(Connection.rs.next() == true) {
            lblUseCode.setText(text: Connection.rs.getString(i: 1));
            Connection.pst = Connection.conn.prepareStatement(string: "SELECT SUM(total_amount) as total FROM clients_tbl");
            Connection.rs = Connection.pst.executeQuery();
            if(Connection.rs.next() == true) {
                lblRevenue.setText(text: Connection.rs.getString(i: 1));
            }
        }
    } catch (SQLException ex) {}
}

private void btnRefreshActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        PreparedStatement count1 = Connection.conn.prepareStatement(string: "SELECT COUNT(status) FROM clients_tbl WHERE status = '1'");
        java.sql.ResultSet rs1 = count1.executeQuery();
        if(rs1.next() == true) {
            String up = rs1.getString(i: 1);
            lblUnpaid.setText(text: up);

            PreparedStatement count2 = Connection.conn.prepareStatement(string: "SELECT COUNT(status) FROM clients_tbl WHERE status = '0'");
            java.sql.ResultSet rs2 = count2.executeQuery();
            if(rs2.next() == true) {
                String p = rs2.getString(i: 1);
                lblPaid.setText(text: p);
            }
        }
    } catch (SQLException ex) {}
}
```

```
private void txtSItemKeyTyped(java.awt.event.KeyEvent evt) {
    char c = evt.getKeyChar();
    if(!Character.isDigit(ch: c)) {
        evt.consume();
    }
}

private void btnTruncateActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        Connection.pst = Connection.conn.prepareStatement(string: "TRUNCATE TABLE clients_tbl");

        Connection.pst.executeUpdate();
        Fetch();

    } catch (SQLException ex) {}
    lblRevenue.setText(text: "0");
    lblUseCode.setText(text: "0");
}
```