# System Design Specification

## Team 2

## Parking Lot Management Application

Drew Caldwell, Brady Covyeou,
Sukhkaran Gill, and Dustin Groh

C308 - Systems Analysis and Design

November 2024

# Table of Contents

# 1.0 Introduction

## 1.1 Goals and Objectives

The main goal of this project is to develop a software that will assist with parking space management in parking lots/garages. The software will allow customers to view how many spaces are currently open and reserve a space ahead of time. Parking lot staff will utilize the software to confirm or cancel customer reservations, as well as adjust the number of open versus occupied spaces as cars leave or enter the lot. Admins will be able to manage everything from parking lots to even users.

The end goal is a java-based software that will run on most common operating systems.

## 1.2 Description

Parking lots are a major part of city traffic, containing rows and columns of cars potentially on multiple floors. A lot of parking lots, especially in major destinations or cities, are paid. These lots often have customers being charged a ticket rate for the amount of hours they spend parked in the parking lot, paying whence they leave. This process is often managed by a ticket machine or lot attendant.

## 1.3 Scope

There are 3 major functions of the software:

- Parking Lot Space Management:
    - Individual parking lots will be able to utilize the software in order to keep a running count of how many spaces in the parking lot are vacant versus

occupied. This information can be accessed by all users of the software for any parking lot in the system.

- Parking Lot Space Reservation:
    - Individual parking lots will utilize the software to allow customers to make reservations for parking spaces in the lot without actually physically being present at the location of the lot. If there are vacant spaces for a lot, customers can choose to make a reservation. If customers already have a reservation, they can choose to cancel that reservation.
- Parking Lot and Account Database Management:
    - Admin-level users will be able to utilize the software to add, delete, or modify parking lots in the system. Attributes such as total number of spaces or parking lot name can be changed. Admin-level users will also be able to modify or delete user accounts. The software will give admins the ability to promote or demote accounts between account types.

# 2.0 Data Design

## 2.1 Data Objects Overview

### Data Object: Customer

Attributes:

- userID: Unique identifier for each customer.
- userType: Type of user (Customer, Staff, Admin, Super Admin).
- firstName: Customer's first name.
- lastName: Customer's last name.
- plateNumber: Customer's vehicle license plate number.

Methods:

- setUserType(): Set the type of user.
- getUserType(): Retrieve the type of user.

- getUserID(): Retrieve the unique user ID.

- setFirstName(): Set the customer's first name.
- getFirstName(): Get the customer's first name.

- setLastName(): Set the customer's last name.
- getLastName(): Get the customer's last name.

- setPlateNumber(): Set the customer's vehicle license plate number.
- getPlateNumber(): Get the customer's vehicle license plate number.

Data Object: ParkingLot

Attributes:

- lotID: Unique identifier for each parking lot.
- lotName: The name or identifier of the parking lot.
- totalSpaces: The total number of parking spaces in the lot.
- emptySpaces: The number of available (empty) parking spaces.
- reservations: A list/map of reservations for the lot, including customer details.

Methods:

- getLotID(): Get the parking lot's unique ID.
- setLotName(): Set the parking lot's name.
- getLotName(): Get the parking lot's name.

- isVacant(): Check if there are empty spaces available.

- setTotalSpaces(): Set the total number of spaces in the parking lot.
- getTotalSpaces(): Retrieve the total number of spaces in the parking lot.

- setEmptySpaces(): Set the number of empty spaces available.
- getEmptySpaces(): Retrieve the number of empty spaces available.

- addReservation(): Add a reservation for a customer.
- getReservations(): Get all reservations for the lot.
- checkReservation(): Check if a customer has a reservation.
- confirmReservation(): Confirm a reservation.
- cancelReservation(): Cancel a reservation.

Data Object: Staff (extends Customer)

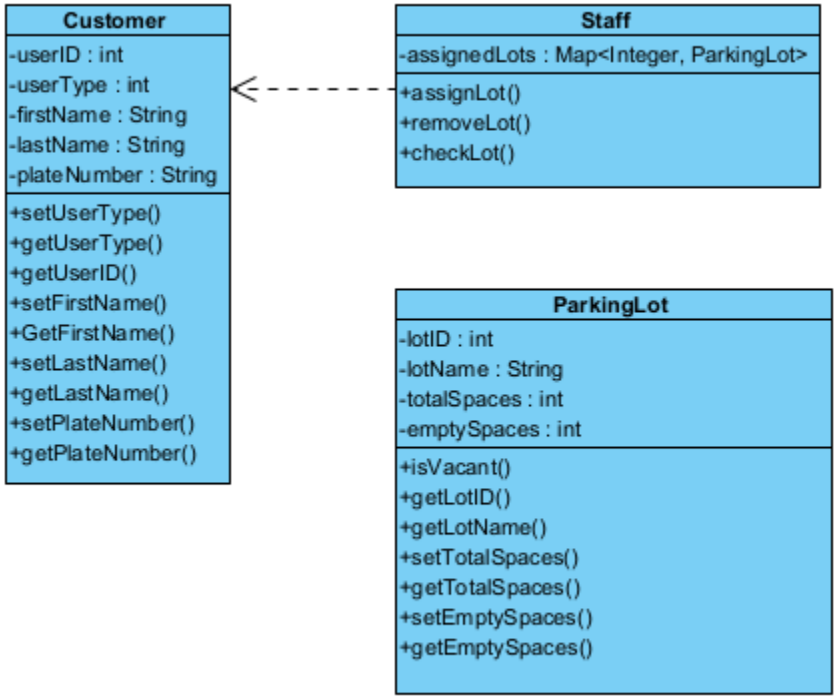
Attributes:

- assignedLots: A map of parking lots assigned to the staff member.

Methods:

- assignLot(): Assign a parking lot to a staff member.
- removeLot(): Remove a parking lot from the assigned list.
- checkLot(): Check if a parking lot is assigned to the staff member.

## 2.2 Class Diagram




# 3.0 Software Design

## 3.1 Overview

Since the program will be an installed application, once opened, the program will display a graphical user interface. All of the application's logic and database will initially be contained within the application itself. In potential future iterations, the database will be moved from within the application to a server.

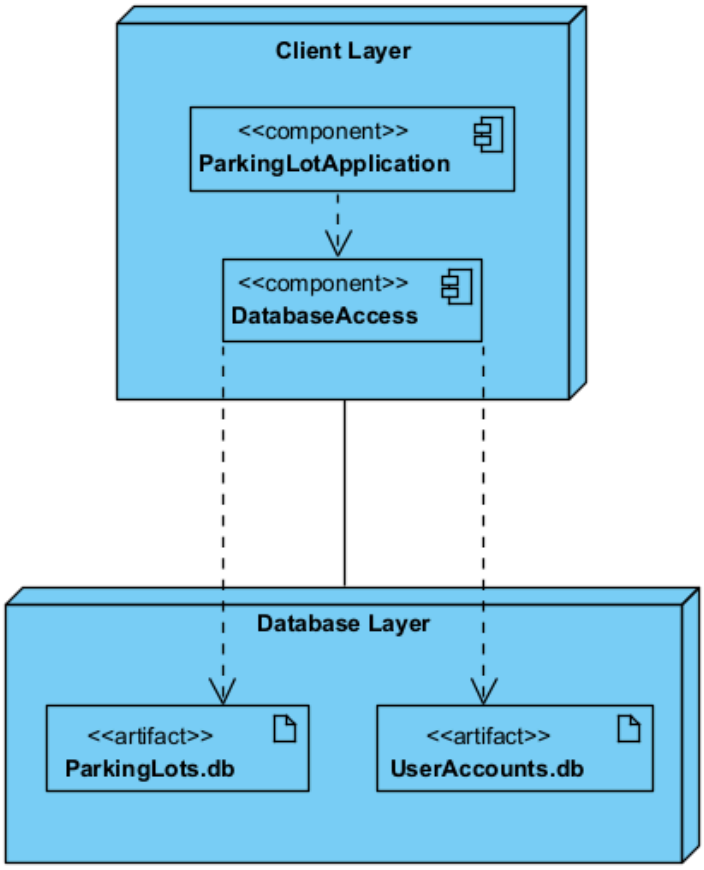
## 3.2 Client Layer

The application, once launched, will provide a user interface to login and to create a customer or staff account. Once logged in, a user can use intuitive GUI functionality pertaining to their access level. The Client Layer interfaces with the Database Layer to retrieve account information, manage parking spaces, and manage parking lots through the GUI.

## 3.3 Database Layer

For simplicity, this iteration of the database is embedded within the application for quick access to the Client Layer. Database objects like Admins or ParkingLots will be saved as formatted generic files. The database can be accessed anywhere but only heavily modified by Admin and SuperAdmin users.

## 3.4 Architecture Diagram




# 4.0 Detailed Design

## 4.1 Class Member Functions and Details

### 4.1.1 Customer

- private userType : int
    - The type of user
    - 0 = Customer(default), 1 = Staff, 2 = Admin, 3 = Super Admin
- private userID : int

    - Unique user identifier
- private firstName : String
    - The users first name
- private lastName : String
    - The users last name
- private plateNumber : String
    - The users license plate number
- public function setUserType( userType : int )
    - Changes the value of userType
- public function getUserType() : int
    - Returns the userType
- public function getUserID() : int
    - Returns the userID
- public function setFirstName( firstName : String )
    - Changes the value of firstName
- public function getFirstName() : String
    - Returns the firstName
- public function setLastName( lastName : String )
    - Changes the value of lastName
- public function getLastName() : String
    - Returns the lastName
- public function setPlateNumber( plateNumber : String )
    - Changes the value of plateNumber
- public function getPlateNumber() : String
    - Returns the plateNumber
- public Customer( firstName : String , lastName : String , plateNumber : String )
    - CONSTRUCTOR: Initializes a Customer object with the given firstName, lastName, and plateNumber. A unique customer ID will be generated and all Customers start with a userType of 0.

### 4.1.2 Staff (extends Customer)

- private assignedLots : Map<Integer, ParkingLot>
    - A map of all ParkingLots assigned, with key = lotID
- public function assignLot( parkingLot : ParkingLot )
    - Adds the given lot to assignedLots
- public function removeLot( parkingLot : ParkingLot )
    - Removes the given lot from assignedLots, if it exists
- public function checkLot( parkingLot : ParkingLot ) : boolean
    - Checks whether the given parkingLot exists in assignedLots

### 4.1.3 Admin (extends Staff)

- public function addLot() : void
    - Adds a parking lot to the database

- public function deleteLot() : void
    - Deletes a parking lot from the database
- public function updateLot() : void
    - Provides functionality to update the details of a lot like the size of the lot
- public function changeLotLocation(): void
    - Changes the location of the lot
- public function totalLotSpace() : int
    - Returns the amount of spaces in a given parking lot
- public function addStaffAccount : void
    - Creates a Staff account in the database
- public function deleteStaffAccount(): void
    - Deletes  a specified Staff account in the database
- public function addCusAccount() : void
    - Creates a Customer account in the database
- public function updateCusAccount() : void
    - Provides functionality to update the details of a Customer account

### 4.1.4 SuperAdmin (extends Admin)

- public function addAdmin() : void
    - Creates an Admin account from the database
- public function deleteAdmin() : void
    - Deletes an Admin account from the database
- public function updateAdmin() : void
    - Provides functionality to update the details of an Admin account

### 4.1.5 ParkingLot

- private lotID : int
    - Unique ID number
- private final lotName : String
    - The name of the ParkingLot
- private totalSpaces : int
    - The number of total spaces in the ParkingLot
- private emptySpaces : int
    - The number of empty spaces in the ParkingLot
- private reservations : Map<Integer, Customer>
    - A map of the Customers that have made a reservation, with key = userID
- public function getLotID() : int
    - Returns the lotID
- public function setLotName( lotName : String)
    - Changes the lotName
- public function getLotName() : String
    - Returns the lotName
- public function isVacant() : boolean

    - Checks whether the ParkingLot has available spaces
- public function setTotalSpaces( spaces : int )
    - Changes the value of totalSpaces
- public function getTotalSpaces() : int
    - Returns totalSpaces
- public function setEmptySpaces( spaces : int )
    - Changes the value of emptySpaces, making sure not to exceed totalSpaces
- public function getEmptySpaces() : int
    - Returns emptySpaces
- public function incrementSpaces() : void
    - Increments emptySpaces by 1
    - Used when a vehicle leaves the ParkingLot
- public function decrementSpaces() : void
    - Decrements emptySpaces by 1
    - Used when a vehicle enters the ParkingLot
- public function addReservation( customer : Customer )
    - Adds the given Customer to reservations and decrements emptySpaces
- public function getReservations() : Map<Integer, Customer>
    - Returns reservations
- public function checkReservation( customer : Customer )
    - Checks whether the given Customer is in reservations
- public function confirmReservation( customer : Customer )
    - Removes the given Customer from reservations
- public function cancelReservation( customer : Customer)
    - Removes the given Customer from reservations and increments emptySpaces
- public function resetLot()
    - Removes all reservations and sets emptySpaces = totalSpaces
- public ParkingLot( lotName : String , totalSpaces : int )
    - CONSTRUCTOR: Initializes a ParkingLot object with the given lotName and totalSpaces. A unique lot ID will be generated. Lots will start with 0 reservations and all spaces empty.

### 4.1.6 Authentication

- private loginID : string
    - Unique ID string that matches a user's ID from of the four access levels
- private password : string
    - String that matches a user's password
- public function authenticateUser() : boolean
    - Returns true when a correct loginID and password are input and compared to an existing user, allowing access to the rest of the application

### 4.1.7 ParkingLotDatabase

- private totalLots : int

    - Holds that total amount of parking lots
- public function initializeLot() : int
    - Reads from saved files into the database and loads them into a hashmap
- public function readLot(String name): ParkingLot
    - Returns a parking lot class for temporary access
- Public function writeLot(ParkingLot lot) : booleans
    - Saves a lot from the passed in reference to the database's filesystem

### 4.1.8 ParkingManagementGUI (extends JFrame)

- private currentUser : Customer
    - Stores the information of the currently logged in user
- private parkingLots : Map<Integer, ParkingLot>
    - Stores the ParkingLots in the system
- private selectedLot : ParkingLot
    - Stores whichever ParkingLot the user selects from the main menu
- private function drawLogin()
    - Draws the login interface
    - Has text boxes for the user to enter a username and password and a button to call login()
    - Has an additional button for creating a new account
- private function login() : boolean
    - Gets the username and password that the user typed in and calls Authentication.autheticateUser()
    - Returns true if login is successful
- private function drawCreateAccount()
    - Draws the create account interface
    - Has text boxes for the user to enter a username, password, first name, last name, and plate number
    - Has a button to call createAccount()
- private function createAccount() : boolean
    - Gets the username and password that the user typed in and calls Admin.addCusAccount()
    - Returns true if account creation is successful
- private function drawMainMenu()
    - Draws the main menu interface
    - Displays a list of parkingLots, each of which can be selected to view individual lot details (calls selectLot())
    - Admin+ level users will see additional options for adding or removing lots
- private function selectLot()
    - Function called when a ParkingLot is selected from the interface
    - Updates selectedLot
    - Calls drawLotDetails()
- private function drawLotDetails()
    - Draws the interface for an individual ParkingLot (selectedLot)

    - Displays the ParkingLot name, total number of spaces, and number of empty spaces
    - Customer level users will have the option to make a reservation if there is open space, or cancel an already existing reservation
    - Staff+ level users will have additional options to view all reservations, confirm a reservation, cancel a reservation, or manage space numbers
- public ParkingManagementGUI()
    - CONSTRUCTOR: Initializes a ParkingManagementGUI object, which sets up the frame for the application and makes it visible. Draws the login interface by default, or the main menu if already logged in.

## 4.2 Member Function Examples

### 4.2.1 Customer

```
public class Customer {
    // Define the constants for different user types
    public static final int CUSTOMER = 0;     // Default user type
    public static final int STAFF = 1;        // Staff user type
    public static final int ADMIN = 2;        // Admin user type
    public static final int SUPER_ADMIN = 3;  // Super Admin user type

    private static int nextUserID = 1;   // To generate unique user IDs for each new Customer

    private final int userID;   // Unique identifier for the user

    // Type of the user (can represent different access levels such as regular Customer, Admin,
etc.)
    private int userType;

    // User's first name, last name, and vehicle plate number
    private String firstName, lastName, plateNumber;

    // Constructor to initialize a Customer object with first name, last name, and plate number
    public Customer(String firstName, String lastName, String plateNumber) {
        this.userID = nextUserID++;  // Generate unique userID
        this.userType = CUSTOMER; // Default userType is 0 (Customer)
        this.firstName = firstName;
        this.lastName = lastName;
        this.plateNumber = plateNumber;
    }

    // Setter method to change the userType
    public void setUserType(int userType) {
```

```
    if (userType >= 0 && userType <= 3) { // Ensure the userType is valid
        this.userType = userType;
    } else {
        System.out.println("Invalid user type");
    }
}

// Getter method to return the userType
public int getUserType() {
    return userType;
}

// Getter method to return the userID
public int getUserID() {
    return userID;
}

// Getter method to return the firstName
public String getFirstName() {
    return firstName;
}

// Setter method to change the firstName
public void setFirstName(String firstName) {
    this.firstName = firstName;
}

// Getter method to return the lastName
public String getLastName() {
    return lastName;
}

// Setter method to change the lastName
public void setLastName(String lastName) {
    this.lastName = lastName;
}

// Getter method to return the plateNumber
public String getPlateNumber() {
    return plateNumber;
}

// Setter method to change the plateNumber
public void setPlateNumber(String plateNumber) {
```

```
        this.plateNumber = plateNumber;
    }
}
```

### 4.2.2 Staff

```
public class Staff extends Customer {
    // A map to store the assigned parking lots, where key = lotID and value = ParkingLot
    private Map<Integer, ParkingLot> assignedLots;

    // Constructor to initialize the assigned lots map
    public Staff(String firstName, String lastName) {
        super(firstName, lastName, null);
        this.assignedLots = new HashMap<>();
    }

    // Method to assign a parking lot
    public void assignLot(ParkingLot parkingLot) {
        if (parkingLot != null) {
            assignedLots.put(parkingLot.getLotID(), parkingLot);
        }
    }

    // Method to remove a parking lot
    public void removeLot(ParkingLot parkingLot) {
        if (parkingLot != null) {
            assignedLots.remove(parkingLot.getLotID());
        }
    }

    // Method to check if a parking lot exists in the assigned lots
    public boolean checkLot(ParkingLot parkingLot) {
        if (parkingLot != null) {
            return assignedLots.containsKey(parkingLot.getLotID());
        }
        return false;
    }
}
```

### 4.2.3 ParkingLot

```
public class ParkingLot {
    private static int nextLotID = 1;  // To generate unique lot IDs for each new ParkingLot
    private int lotID;
    private String lotName;
```

```
private int totalSpaces;
private int emptySpaces;
private Map<Integer, Customer> reservations;  // Map of reservations, with key = userID

// Constructor: Initializes a ParkingLot object with the given lotName and totalSpaces
public ParkingLot(String lotName, int totalSpaces) {
   this.lotID = nextLotID++;  // Generate unique lotID
   this.lotName = lotName;
   this.totalSpaces = totalSpaces;
   this.emptySpaces = totalSpaces;  // Initially all spaces are empty
   this.reservations = new HashMap<>();  // Initialize the reservations map
}

public int getLotID() {
   return lotID;
}

public void setLotName(String lotName) {
   this.lotName = lotName;
}

public String getLotName() {
   return lotName;
}

// Method to check if the parking lot has available spaces
public boolean isVacant() {
   return emptySpaces > 0;
}

public void setTotalSpaces(int spaces) {
   this.totalSpaces = spaces;
   if (emptySpaces > spaces) {
      emptySpaces = spaces;  // Adjust emptySpaces if the totalSpaces is reduced
   }
}

public int getTotalSpaces() {
   return totalSpaces;
}

public void setEmptySpaces(int spaces) {
   if (spaces <= totalSpaces) {
      this.emptySpaces = spaces;
```

```
    }
}

public int getEmptySpaces() {
    return emptySpaces;
}

// Method to increment emptySpaces by 1, used when a vehicle leaves the ParkingLot
public void incrementSpaces(){
    if (emptySpaces < totalSpaces) {
        emptySpaces++;
    }
}

// Method to decrement emptySpaces by 1, used when a vehicle enters the ParkingLot
public void decrementSpaces(){
    if (emptySpaces > 0) {
        emptySpaces--;
    }
}

// Method to add a reservation
public void addReservation(Customer customer) {
    if (customer != null && emptySpaces > 0) {
        reservations.put(customer.getUserID(), customer);
        emptySpaces--;
    }
}

// Method to get all reservations
public Map<Integer, Customer> getReservations() {
    return new HashMap<>(reservations);  // Return a copy of the reservations map
}

// Method to check if a customer has a reservation
public boolean checkReservation(Customer customer) {
    if (customer != null) {
        return reservations.containsKey(customer.getUserID());
    }
    return false;
}

// Method to confirm a reservation, removing the customer from reservations
public void confirmReservation(Customer customer) {
```

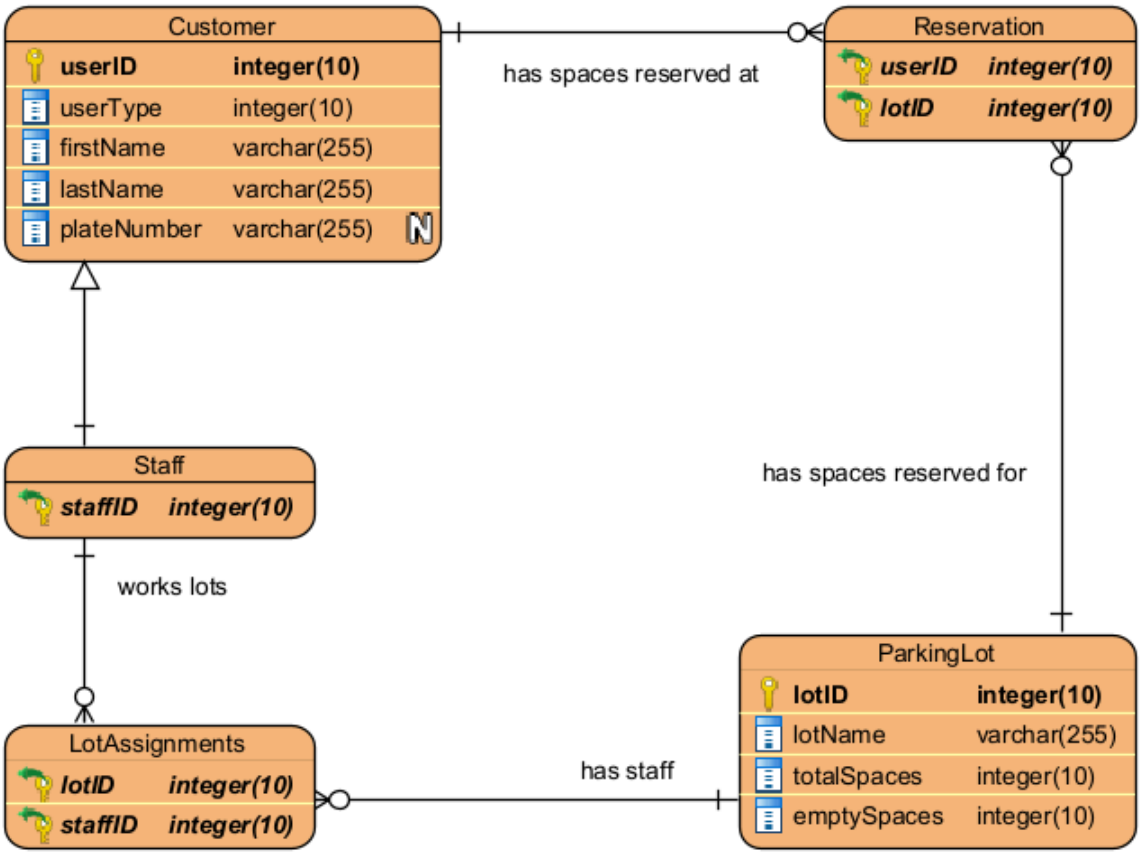
```
        if (customer != null) {
            reservations.remove(customer.getUserID());
        }
    }

    // Method to cancel a reservation, removing the customer from reservations and incrementing
emptySpaces
    public void cancelReservation(Customer customer) {
        if (customer != null && reservations.containsKey(customer.getUserID())) {
            reservations.remove(customer.getUserID());
            emptySpaces++;
        }
    }

    // Method to reset the parking lot (remove all reservations and reset emptySpaces to
totalSpaces)
    public void resetLot() {
        reservations.clear();
        emptySpaces = totalSpaces;
    }
}
```

# 5.0 Database Design

## 5.1 Database Description

Each ParkingLot will be identified by a unique lotID along with a descriptive name and store the number of total and empty spaces in the lot. Each ParkingLot will also have a list of the Staff that work at the lot and a list of Customers that have reserved spaces. Customers will be identified by a unique userID, and also have a userType, firstName, lastName, and plateNumber. Staff will extend the Customer class and have access to additional functionality for the ParkingLots they work at.

## 5.2 Entity Relationship Diagram




# 6.0 Software Interface Description

## 6.1 External Interfaces

### 6.1.1 Parking Lot Gate Interface

This interface performs the following two functions:

a) Upon customer arrival, the interface prints a ticket for the customer's vehicle by the customer pressing a ticket button. Once the ticket is printed, the gate opens for the customer and their vehicle to enter the parking lot.

b) When leaving, the customer scans their ticket into the interface with an integrated barcode scanner and will then insert a credit card to make a payment. The gate opens once the payment is processed and the customer can exit the parking lot.

### 6.1.2 Sensor Detection Interface

a) Once a customer vehicle is parked or exits a parking space, sensors detect the occupancy of the space, whether it is occupied or vacant.
b) The changed occupancy of a space is signaled to the parking lot database, and the database updates the number of empty versus taken spaces.
c) Infrared sensors will be used for this purpose.

### 6.1.3 Network Interface

a) The External interfaces are connected to each other by LAN(Local Area Network). Ethernet/RJ45 is used to wire the interfaces that connect with the Parking Lot Management Application.
b) Firewall is used to protect information within the network from external hacking.

### 6.1.4 Display Interface

a) This interface has an LCD display that shows the number of empty spaces, percentage of lot fullness, or an alert if the parking lot is completely full to new customers. This is a fully automated display at the entrance of the lot.
b) It will be updated in real time as vehicles enter or exit the parking lot.

## 6.2 Human Interface

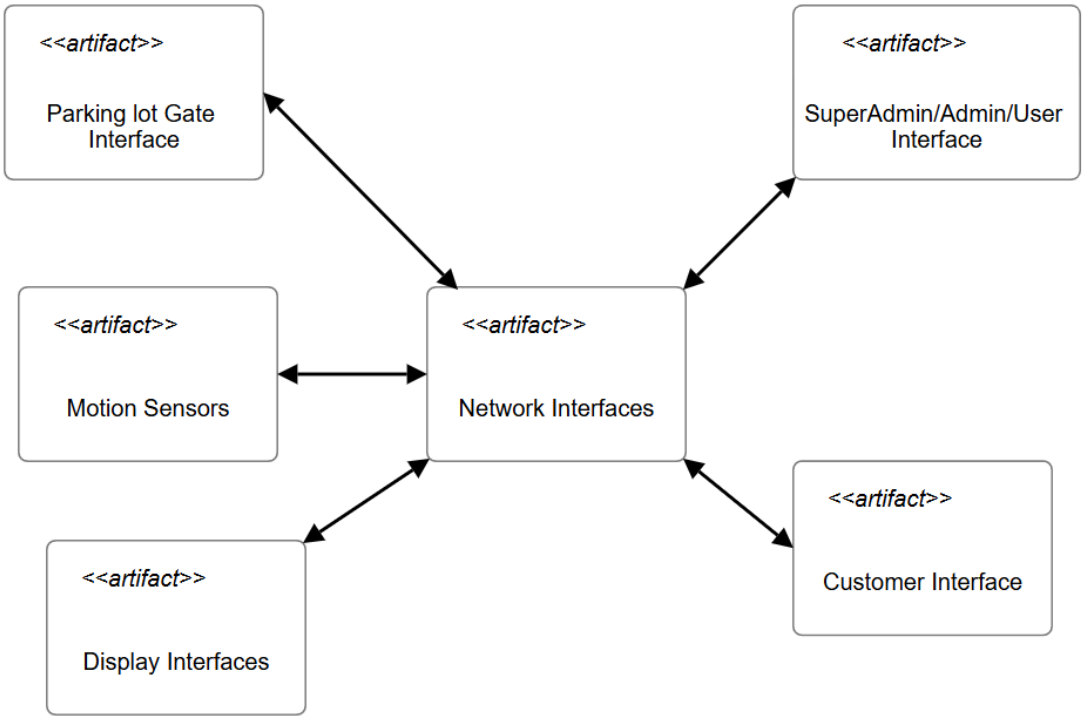
### 6.2.1 Admin / Super Admin / Staff Interface

a) This is the interface for the Admin/Super Admin or the Staff at the parking lot, with a display showing options to manage parking lot spaces and reservations, manage customer and parking lot databases, and generate reports.
b) The display is on a 22-27 inch monitor running on a Windows PC.

### 6.2.2 Customer Interface

a) This interface will display a list of all parking lots in the system. Each lot can be selected to view more details, and customers can make reservations for the parking lot before they arrive at the lot.
b) Customers will use VPN or WIFI to access the parking lot reservation system.

## 6.3 Deployment Diagram



# 7.0 Appendices

## 7.1 Glossary

### 7.1.1 Parking Lot

A parking lot is a location intended for parking vehicles. A parking lot can have a varying amount of rows, columns, and levels. A parking lot has parking spaces.

### 7.1.2 Parking Space

A parking space is a location within a parking lot where a vehicle may park. May be referred to simply as a space.

### 7.1.3 Reservation

A reservation is the act of a parking space in a specific parking lot being reserved for a specific Customer. The space will be held until the Customer arrives at the parking lot. A reservation can be confirmed or canceled by users of access level Staff or higher.

### 7.1.4 Access Level

Access level refers to the function permissions granted to the varying users of the Parking Lot Management Application. A Customer has access to the least amount of functionality within the application while a Super Admin has the most.

### 7.1.5 Customer

A user who uses the Parking Lot Management Application to reserve a parking space within a given parking lot. They have the lowest access level.

### 7.1.6 Staff

A user who uses the Parking Lot Management Application to manage an associated parking lot and its customers.

### 7.1.7 Admin / Super Admin

A user who uses the Parking Lot Management Application to manage customers, staff, and parking lots, with Super Admins having capability to manage admins. They have the highest access level.

### 7.1.8 Parking Lot Management Application

The basic placeholder name used for this software.

## 7.2 Assumptions and Constraints

1. The initial implementation assumes an embedded database, with the possibility of future server-based database migration
2. The system requires Java Runtime Environment (JRE) to operate
3. External interfaces, such as parking lot gates, are assumed to be compatible with the software's integration protocols
4. Infrared motion sensors are the assumed standard for detecting space occupancy

## 7.3 Technology Stack

1. Programming Language: Java (with Swing for GUI development)
2. Database: Embedded databases (Java HashMap)
3. Network Protocol: LAN with Ethernet/RJ45 for connectivity
4. External Devices:
    a. Ticket printers and barcode scanners
    b. Infrared sensors for motion detection
    c. LCD display screens for real-time updates

## 7.4 Future Enhancements

1. Migration of the database to a server for multi-location management
2. Integration with mobile applications for enhanced customer experience
3. Options to add additional details to parking lots, such as open hours
4. Use of AI for real-time parking space allocation optimization
5. Expansion of user roles to include regional managers overseeing multiple lots
6. Enhanced security protocols for network protection against cyber threats