PHASE -3 INNOVATION

SMART PARKING SYSTEM

INTRODUCTION:

Smart parking system using IoT. This is a great system and project for techies, hobbyists, project makers and programming.

What do you need to make this system and how you can make it we will share all the details. We have uploaded a lot of projects before from our smart irrigation system using Arduino.

PROGRAMMING:

```
Import colorama

From termcolor import colored

Options_message = “””

Choose:

1. To park a vehicle

2. To remove a vehicle from parking

3. Show parking layout

4. Exit

“””
```

```
Class Vehicle:
    Def _init_(self, v_type, v_number):
        Self.v_type = v_type
        Self.v_number = v_number
        Self.vehicle_types = {1: ‘c’, 2: ‘b’, 3: ‘t’}
Def _str_(self):
        Return self.vehicle_types[self.v_type]
Class Slot:
  Def _init_(self):
        Self.vehicle = None

    @property
    Def is_empty(self):
        Return self.vehicle is None
Class Parking:

    Def _init_(self, rows, columns):
        Self.rows = rows
        Self.columns = columns
        Self.slots = self._get_slots(rows, columns)

    Def start(self):
        While True:
            Try:
                Print(options_message)

                Option = input(“Enter your choice: “)

                If option == ‘1’:
```

```
            Self._park_vehicle()

        If option == '2':
            Self._remove_vehicle()

        If option == '3':
            Self.show_layout()

        If option == '4':
            Break

    Except ValueError as e:
Print(colored(f"An error occurred: {e}. Try again.", "red"))

    Print(colored("Thanks for using our parking assistance system", "green"))

Def _park_vehicle(self):
    Vehicle_type = self._get_safe_int("Available vehicle types: 1. Car\t2. Bike\t3. Truck.\nEnter your choice: ")

    If vehicle_type not in [1, 2, 3]:
        Raise ValueError("Invalid vehicle type specified")

    Vehicle_number = input("Enter vehicle name plate: ")
    If not vehicle_number:
        Raise ValueError("Vehicle name plate cannot be empty.")
    Vehicle = Vehicle(vehicle_type, vehicle_number)

    Print('\n')
```

```
        Print(colored(f”Slots available: {self._get_slot_count()}\n”, “yellow”))
        Self.show_layout()
        Print(‘\n’)

        Col = self._get_safe_int(“Enter the column where you want to park the vehicle: “)
        If col <= 0 or col > self.columns:
            Raise ValueError(“Invalid row or column number specified”)

        Row = self._get_safe_int(“Enter the row where you want to park the vehicle: “)
        If row <= 0 or row > self.rows:
            Raise ValueError(“Invalid row number specified”)

        Slot = self.slots[row-1][col-1]
        If not slot.is_empty:
            Raise ValueError(“Slot is not empty. Please choose an empty slot.”)
Slot.vehicle = vehicle

    Def _remove_vehicle(self):
        Vehicle_number = input(“Enter the vehicle number that needs to be removed from parking slot: “)
        If not vehicle_number:
            Raise ValueError(“Vehicle number is required.”)

        For row in self.slots:
            For slot in row:
                If slot.vehicle and slot.vehicle.v_number.lower() == vehicle_number.lower():
                    Vehicle: Vehicle = slot.vehicle
                    Slot.vehicle = None
                    Print(colored(f”Vehicle with number ‘{vehicle.v_number}’ removed from parking”, “green”))
                    Return
```

```
        Else:
            Raise ValueError(“Vehicle not found.”)

    Def show_layout(self):
        Col_info = [f’<{col}>’ for col in range(1, self.columns + 1)]
        Print(colored(f”|{‘’.join(col_info)}|columns”, “yellow”))
Self._print_border(text=”rows”)

        For I, row in enumerate(self.slots, 1):
            String_to_printed = “|”
            For j, col in enumerate(row, 1):
                String_to_printed += colored(f”[{col.vehicle if col.vehicle else ‘ ‘}]”,
                                             “red” if col.vehicle else “green”)
            String_to_printed += colored(f”|<{i}>”, “cyan”)
            Print(string_to_printed)

        Self._print_border()

    Def _print_border(self, text=””):
        Print(colored(f”|{‘-‘ * self.columns * 3}|{colored(text, ‘cyan’)}”, “blue”))

    Def _get_slot_count(self):
        Count = 0
        For row in self.slots:
            For slot in row:
                If slot.is_empty:
                    Count += 1
        Return count
```

```
    @staticmethod
    Def _get_slots(rows, columns):
        Slots = []
        For row in range(0, rows):
            Col_slot = []
            For col in range(0, columns):
                Col_slot.append(Slot())
            Slots.append(col_slot)
        Return slots


    @staticmethod
    Def _get_safe_int(message):
        Try:
            Val = int(input(message))
            Return val
        Except ValueError:
            Raise ValueError(“Value should be an integer only”)
Def main():
    Try:
        Print(colored(“Welcome to the parking assistance system.”, “green”))
        Print(colored(“First let’s setup the parking system”, “yellow”))
        Rows = int(input(“Enter the number of rows: “))
        Columns = int(input(“Enter the number of columns: “))


        Print(“Initializing parking”)
        Parking = Parking(rows, columns)
        Parking.start()


    Except ValueError:
```

```
        Print("Rows and columns should be integers only.")

    Except Exception as e:
        Print(colored(f"An error occurred: {e}", "red"))


If _name_ == '_main_':
    Colorama.init()  # To enable color visible in command prompt
    Main()
```

CONCLUSION:

The development of the Internet of Things and cloud technology opens up new opportunities for smart cities. Smart parking has always been the backbone of building smart cities. IoT-based smart parking system offers real-time slots, parking procedures, information and improves users' ability to save time on proper parking. It helps to solve growing traffic congestion concerns. As for future work, users can book parking in a remote location. GPS, reservations, and license plate scanners can be included in the future.