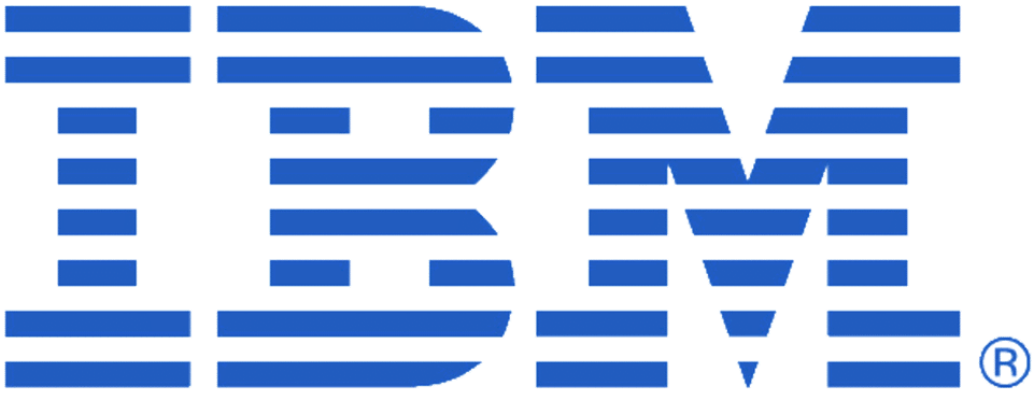


SMART PARKING

Phase 4 submission

Phase4: Development part 2

Topic : Smart parking

Explanation:

The python program should make a car park where it's a 8×9 grid using [square brackets], the parking space should be filled with colours, and letters to signify its availability and which type of car is parked. For example bikes, trucks, cars, sports cars… etc.

- Here the car parking needs to be a code which will make a car parking area.
- There should be red and green light to indicate if it's vacant or occupied.
- [square brackets] should be used to show each parking spot
- Use different letters to show what kind of vehicle is in the spot. For example car,truck,bike, cycle..etc.

Python Coding:

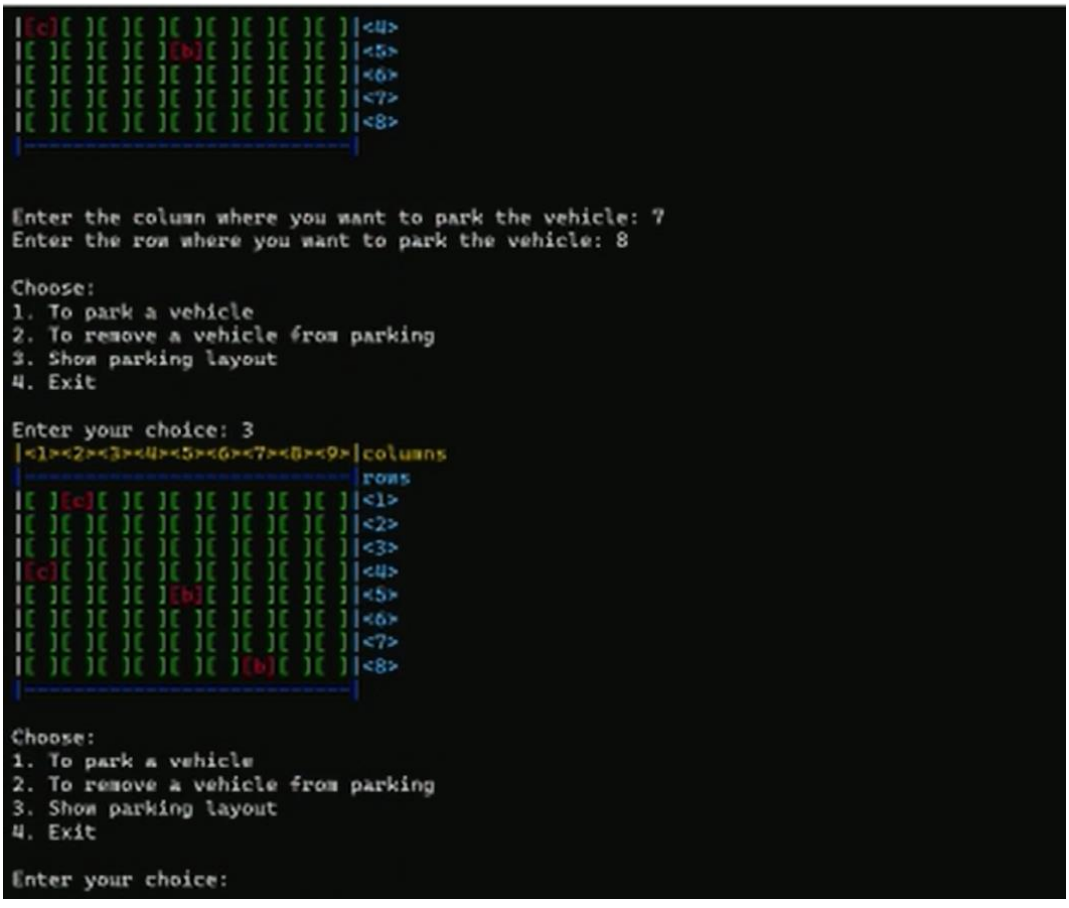
```
import colorama

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):
```

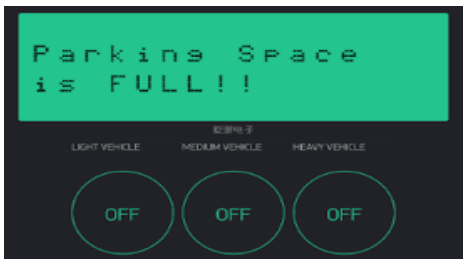
```
        vehicle_number = input("Enter the vehicle number that needs to be removed from parking slot: ")
        if not vehicle_number:
            raise ValueError("Vehicle number is required.")

        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()
```

Output:



Mobile application(Framework:Blynk):



Conclusion:

The proposed IoT-cloud based smart parking model comprises of remote sensor systems. Here i execute a framework that permits vehicle drivers to handily locate the empty parking spots. This framework comprises of remote sensor systems, Blynk cloud, wifi module and mobile application. In this framework, minimal effort remote sensors systems module are sent into each stopping opening furnished with three sensors. The condition of the stopping opening is distinguished by sensor hub and that were accounted for intermittently to the blynk server through the sent remote sensor systems. What's more, this data is sent to blynk server through Wi-Fi organizes progressively. Here the drivers can discover empty parking spot by utilizing their portable application. By this ICSP framework we accept this proposed design can satisfy every one of your prerequisites of cloud based shrewd stopping framework and the foundation of the versatile application, remote sensor systems can be a well utilitarian innovation to comprehend forthcoming smart parking works