INTERNET OF THINGS – GROUP 4

PHASE-4

SMART PARKING

NAME	**: ABIRAMAN.S**
REGISTER NUMBER	**: 822621106002**
NM ID	**: au822621106002**
COLLEGE NAME	**:ARIFA INSTITUTE OF TECHNOLOGY**
COLLEGE CODE	**:8226**
EMAIL ID	**:abiramantech@gmail.com**

DESIGN OF THE SYSTEM:

Network Time protocol :-

The Network Time Protocol is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks. We have used NTP for fetching time from the NTP server so that we can show the start time and end time for the user when he parks or unparks his vehicle making information real-time.

Blynk app:-

Blynk app is a hardware-agnostic IoT platform with white label mobile apps, private cloud ,device management, data analytics and machine learning .On using the blynk app we tried to pop notification to every possible event that is occurring in the parking zone .

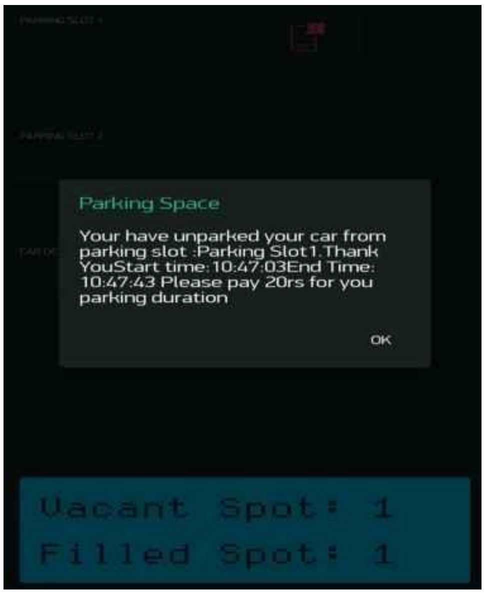
Used a serial algorithm to display the slot number to the user who is going to park his vehicle .For example we display the empty slot number in a serial manner which gets filled , if the slot 1 is filled and when an another vehicle turns up we display slot 2 and further like these for all other vehicles , and if any vehicle leaves the slot number then we display the earlies slot number , not making the user to travel long if an initial spot is vacant .

PARKING A VEHICLE:

Once when the user enters the parking detect sensor he would receive a parking slot number on his mobile application which he is supposed to park his vehicle. Upon parking the vehicle in the respective slot and IR sensor successfully detecting the vehicle it would show a notification on the app the start time of the vehicle and the slot number in which the vehicle is parked and it would be similarly updated on the 16*2 display.

UN PARKING AVEHICLE:

Unparking your vehicle from the parking slot would pop a notification on the application app stating the start time and the end time the user has parked his vehicle in the parking slot and an small amount which the user needs to pay when he leaves the parking zone which is fixed for any duration .



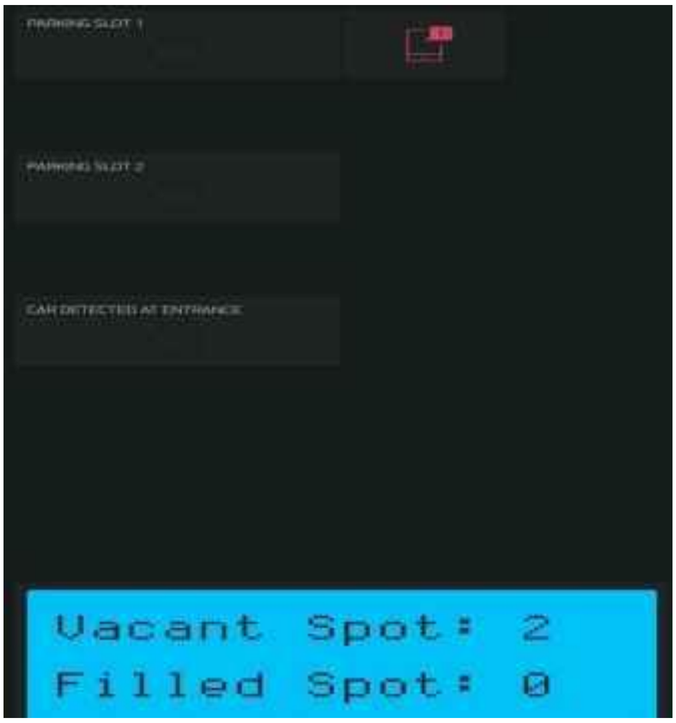
Parking Space

Your have unparked your car from parking slot :Parking Slot1.Thank YouStart time:10:47:03End Time: 10:47:43 Please pay 20rs for you parking duration

Vacant Spot: 1
Filled Spot: 1

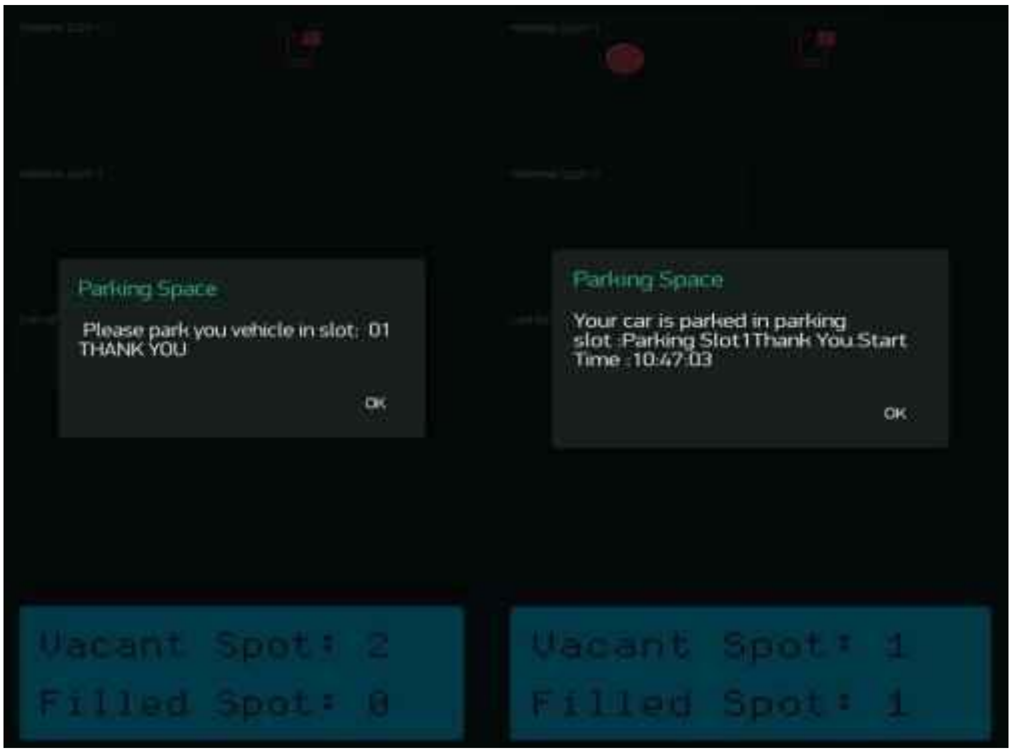
TESTING:

This case shows that all the parking slots are empty and therefore, the system will allow a car to enter into the parking zone . The 16*2 LCD will display the number of vacant spot and filled spot and similarly it would be displayed on the application.



```
Vacant  Spot: 2
Filled  Spot: 0
```

This following case focuses on showing a slot number when the user is near to the parking detect sensor. It shows a parking slot number where the user should park his vehicle and upon parking it shows the start time of his parking.


Parking Space

Please park you vehicle in slot: 01
THANK YOU

Parking Space

Your car is parked in parking slot :Parking Slot1Thank You Start Time :10:47:03

```
Vacant  Spot: 2
Filled  Spot: 0
```

```
Vacant  Spot: 1
Filled  Spot: 1
```

PROGRAM:

Source Code:

```
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()
```