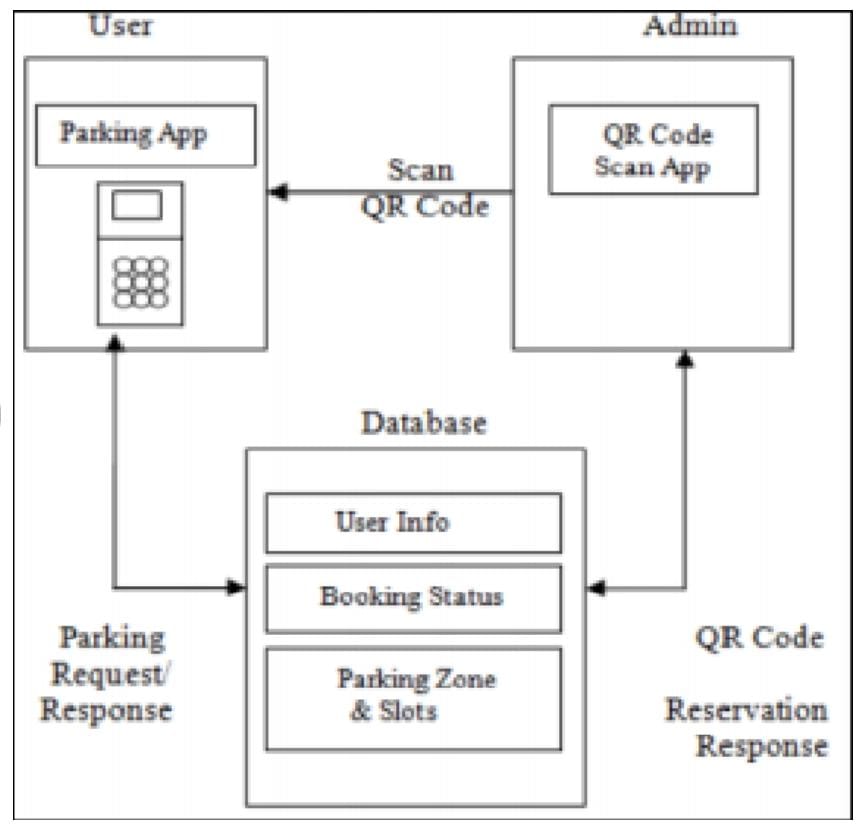
|  |
| --- |
|  |

|  |
| --- |
|  |
| SMART PARKING  Using IoT |
| Submitted by,    KAVIYARASAN A  au812921106022  kaviyarasanarun04@gmail.com |

Introduction

Being in one of the Indian Metro Cities which has a highly active urban life, people face a lot of everyday situations with automobiles. And one of the top few problems is the car parking, for sure. Car Parking in shopping malls is a task that takes up a lot of human time and effort. And it happens in a very large scale that is difficult for us to comprehend. The proposed automatic smart car parking system will overcome all the challenges and difficulties that are there in conventional car parking system. It saves us time and effort that we put into this task, and also the fuel wastage can be reduced. It could help to bring order out of the chaos that exists at present in the car parking issues and make it really simple and easy.

Block diagram



Explanation

In the past, there have been many works done on smart parking system approaching an even smarter system in where researches have been done and still being done to create a system which is not technologically savvy but also at ease. This paper proposes a design of smart parking system where it helps the users to reserve parking slots using Android application. This project is aimed to create a system that helps people with personal vehicles to find for parking easily at selected areas. Both software and hardware platform have been developed in this system.

Program

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")