# Innovative Smart Parking

# Phase-3

***Team Leader:***

S Yogul Kishore

21152104187

***Members:***

Veniyan U

211521104176

Sai Niranjan M S

211521104131

Viknesh K S

211521104179

Sowmiya Narayanan S V

211521104154

# Table Of Contents

1. **Introduction**
2. **Installation and Setup**

* Prerequisites
* Installing Flask
* Database Setup

1. **Python Script for Web Application**

* Code Explanation
* Function for Fetching Parking Slot Data
* Web Application Routes

1. **HTML Template**

* Designing the User Interface
* Displaying Parking Slot Information

1. **Conclusion**

* Summary
* Next Steps and Enhancements

## Introduction

The integration of technology into our daily lives has paved the way for innovative solutions across various domains. One such area of innovation is the management of parking facilities, where the need for efficient utilization of available parking spaces has led to the development of smart parking systems.

This document presents a solution for monitoring and displaying the availability of parking slots in real-time. The system uses computer vision to detect the presence of vehicles in parking slots and stores this information in a SQLite database. Additionally, it provides a web-based interface for users to check the current status of parking slots.

The primary goal of this system is to offer convenience to both parking lot operators and users. Operators can efficiently manage their parking facilities, while users can easily find vacant parking spaces, reducing the time and effort required for parking.

In the following sections, we will explore the components of this system, including the Python code responsible for car detection and database management, as well as the web application that displays parking slot availability. This document also serves as a guide for setting up and running the system, offering a starting point for anyone interested in developing similar smart parking solutions.

Let's embark on a journey to create a smart parking solution that streamlines the parking experience and contributes to more efficient urban mobility.

2. Installation and Setup

Before you can implement the smart parking solution, you need to set up the necessary tools and components. This section provides a step-by-step guide for the installation and initial setup of the system. Ensure you follow these instructions carefully to prepare your environment for the subsequent steps.

#### 2.1 Prerequisites

Before you begin, make sure you have the following prerequisites in place:

**Python**: You will need Python installed on your system. You can download and install Python from the [official Python website](https://www.python.org/).

**OpenCV:** This solution relies on OpenCV for car detection. You can install OpenCV using pip:

bash

pip install opencv-python

**SQLite Database:** You will need SQLite, which is often included with Python installations.

**Flask:** To create the web interface, you need Flask. You can install Flask with pip:

bash

pip install Flask

**Haar Cascade Classifier:** Download a pre-trained Haar Cascade classifier XML file for car detection. You can find these classifiers from sources like the [OpenCV GitHub repository](https://github.com/opencv/opencv/tree/master/data/haarcascades).

#### 2.2 Database Setup

1. \*\*Database Creation:\*\* Create an SQLite database to store parking slot data. You can create a new database or use an existing one. In this document, we assume the database is named `parking\_data.db`. You can create it using the following Python code:

python

import sqlite3

conn = sqlite3.connect('parking\_data.db')

cursor = conn.cursor()

cursor.execute('''

CREATE TABLE IF NOT EXISTS parking\_slots (

slot\_id INTEGER PRIMARY KEY,

is\_occupied INTEGER,

timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP

)

''')

conn.close()

Make sure this database file is in the same directory as your Python script.

#### 2.3 Code and Template Files

Make sure to have the Python script for the smart parking system (as shown in Section 3) and the HTML template file (as shown in Section 4) available.

With these prerequisites and the database in place, you're now ready to proceed with the Python script for the web application and car detection.

### 3. Python Script for Web Application

In this section, we will dive into the Python script responsible for creating the web application that displays parking slot availability. The script uses the Flask framework to set up the web server and routes for handling web requests. Additionally, it connects to the SQLite database to retrieve parking slot data for display.

#### 3.1 Code Explanation

The Python script is divided into several sections to achieve its functionality:

- \*\*Importing Libraries:\*\* The script starts by importing the necessary libraries, including Flask for web development, SQLite for database operations, and OpenCV for car detection.

- \*\*Database Connection:\*\* It establishes a connection to the SQLite database, or creates one if it doesn't exist. A table named `parking\_slots` is defined to store slot information.

- \*\*Car Detection Classifier:\*\* The script loads the pre-trained Haar Cascade classifier XML file for car detection.

#### 3.2 Function for Fetching Parking Slot Data

One of the key functions in the script is responsible for fetching parking slot data from the database. It uses a SQLite SELECT query to retrieve the slot ID, occupancy status, and timestamp of each slot.

#### 3.3 Web Application Routes

The script defines a route for the root URL ('/') to handle web requests. When a user accesses this URL, the script calls the function for fetching parking slot data and passes the retrieved information to an HTML template for rendering.

The data from the database is displayed on the website, indicating which parking slots are occupied and which are vacant.

#### 3.4 Running the Application

To make this web application accessible to users, you'll need to run the script. It sets up a local development server using Flask, allowing you to access the website via a web browser. The script includes a block for running the application only when it is directly executed, not when it's imported as a module.

In the next section, we'll delve into the HTML template used for presenting parking slot availability on the website.

#### Code

from flask import Flask, render\_template

import sqlite3

app = Flask(\_\_name\_\_)

# Function to fetch parking slot data from the database

def get\_parking\_slots():

conn = sqlite3.connect('parking\_data.db')

cursor = conn.cursor()

cursor.execute("SELECT \* FROM parking\_slots")

slots = cursor.fetchall()

conn.close()

return slots

@app.route('/')

def index():

# Get parking slot data

parking\_slots = get\_parking\_slots()

return render\_template('index.html', parking\_slots=parking\_slots)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

4. HTML Template

The web interface for displaying parking slot availability is designed using HTML and can be customized to suit your preferences. This section provides an overview of the HTML template used in the web application, explaining its structure and how it presents parking slot information to users.

### 4.1 Designing the User Interface

The HTML template is designed with a clean and user-friendly layout. It includes the following elements:

* Title: A title at the top of the page indicating the purpose of the website, "Parking Slot Availability."
* Table Display: Slot data is organized in a table format for easy readability. Each row in the table represents a parking slot.
* Table Headers: The table includes headers for each column: Slot ID, Occupied Status, and Timestamp.
* Data Display: Parking slot information is dynamically inserted into the table using Flask's template engine. For each slot, the template displays the slot ID, occupancy status (occupied or vacant), and the timestamp of the last update.

### 4.2 Displaying Parking Slot Information

The script passes parking slot data to the HTML template, which uses the Flask template engine to render the data dynamically. The occupancy status is presented as "Occupied" if the value in the database is 1, and "Vacant" if the value is 0.

The template provides a clear visual representation of parking slot availability to users, making it easy for them to identify vacant slots quickly.

This HTML template forms the user-facing component of the web application and ensures that parking slot information is presented in a user-friendly and easily accessible format.

In the next section, we will discuss database management and how the system collects and stores parking slot data.

### Code:

<!DOCTYPE html>

<html>

<head>

<title>Parking Slot Availability</title>

<style>

table {

width: 80%;

border-collapse: collapse;

margin: 20px auto;

}

th, td {

border: 1px solid #ddd

padding: 8px;

text-align: center;

}

th {

background-color: #f2f2f2;

}

</style>

</head>

### <body>

### <h1>Parking Slot Availability</h1>

### <table>

### <tr>

### <th>Slot ID</th>

### <th>Occupied Status</th>

### <th>Timestamp</th>

### </tr>

### {% for slot in parking\_slots %}

### <tr>

### <td>{{ slot[0] }}</td>

### <td>{{ slot[1] == 1 ? 'Occupied' : 'Vacant' }}</td>

### <td>{{ slot[2] }}</td>

### </tr>

### {% endfor %}

### </table>

### </body>

### </html>

**Conclusion:**

In conclusion, the backend development of the Smart Parking Application is an integral part of creating a seamless and efficient parking management system. This section has provided an overview of the backend architecture, technology stack, database design, and the key features of the application.

We've designed a robust system that effectively manages parking slots, integrates with Fast Tag technology for hassle-free entry and exit, assesses parking prices based on various factors, and allows users to make reservations conveniently. Additionally, the backend features a comprehensive data logging and analysis system, which provides valuable insights into parking usage and user behavior.

The application's pricing and payment processing mechanisms are designed to offer flexibility and convenience to users while ensuring secure and reliable transactions. We've also incorporated a feature that allows users to compare parking prices from various other parking spaces, enabling them to make informed decisions.

As the backend development is a critical component of the Smart Parking Application, it forms the foundation for a user-friendly, efficient, and data-driven parking management system. By focusing on data accuracy, system reliability, and a user-centric approach, we aim to deliver a high-quality application that meets the needs of both parking space owners and users.

The development process is ongoing, and we are committed to refining and enhancing the backend to provide a top-tier user experience. The backend development of the Smart Parking Application is a significant step towards making urban parking more convenient, efficient, and accessible for everyone. We look forward to bringing this innovative solution to the market and continuously improving it to meet the evolving needs of our users.