ДОДАТОК А

Створення віртуальної моделі в Wokwi

Для демонстрації роботи автоматизованої системи обліку паркомісць було використано платформу Wokwi, яка дозволяє імітувати роботу мікроконтролера ESP32, кнопок, сенсорів та світлодіодів.

1. Компоненти:

– ESP32 DevKit;

– 2 кнопки (імітація датчиків в’їзду та виїзду);

– 1 світлодіод (індикація заповнення паркінгу);

– резистор 220 Ом.

2. Підключення:

– кнопка “В’їзд” підключена до GPIO4;

– кнопка “Виїзд” – до GPIO5;

– LED індикатор – до GPIO2 (через резистор на землю).

3. Код для віртуального контроллера

```
#include <Arduino.h>

#define entryPin  4
#define exitPin   5
#define ledPin    2

int capacity = 5;
int current = 0;

void setup() {
  pinMode(entryPin, INPUT_PULLUP);
  pinMode(exitPin, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);
  Serial.begin(115200);
}

void loop() {
  if (digitalRead(entryPin) == LOW && current < capacity) {
    current++;
    Serial.print("Car Entered: ");
    Serial.println(current);
    delay(1000);
```

```
  }
  if (digitalRead(exitPin) == LOW && current > 0) {
    current--;
    Serial.print("Car Exited: ");
    Serial.println(current);
    delay(1000);
  }
  if (current >= capacity) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
}
```

Цей код дозволяє у віртуальному середовищі протестувати логіку в’їзду/виїзду авто та заповнення паркінгу.

ДОДАТОК Б

Підключення ESP32 до веб-застосунку (реальна реалізація)

У разі наявності фізичної плати ESP32, логіку передачі даних можна реалізувати через Wi-Fi з використанням HTTP-запитів до сервера Flask, що розгорнутий на ПК.

1. Підключення ESP32:

– живлення від USB або 5 В джерела;

– кнопки підключаються між GPIO4/5 і GND;

– LED з резистором – до GPIO2 і GND.

2. Код для ESP32 з Wi-Fi та HTTP

```
#include <WiFi.h>
#include <HTTPClient.h>

#define entryPin 4
#define exitPin 5

const char* ssid = "Wokwi-GUEST"; // або назва домашньої Wi-Fi мережі
const char* password = "";

const char* server = "http://192.168.1.10:5000"; // IP ПК з Flask

void sendUpdate(String event) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    String url = String(server) + "/update/" + event;
    http.begin(url);
    int httpCode = http.GET();
    if (httpCode > 0) {
      Serial.println("Відправлено: " + event);
    } else {
      Serial.println("HTTP помилка: " + String(httpCode));
    }
    http.end();
  }
}

void setup() {
  Serial.begin(115200);
  pinMode(entryPin, INPUT_PULLUP);
  pinMode(exitPin, INPUT_PULLUP);

  WiFi.begin(ssid, password);
```

```
  Serial.print("Підключення до Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nПідключено!");
}

void loop() {
  if (digitalRead(entryPin) == LOW) {
    sendUpdate("entry");
    delay(1000);
  }
  if (digitalRead(exitPin) == LOW) {
    sendUpdate("exit");
    delay(1000);
  }
}
```

3. На комп’ютері повинен працювати Flask-сервер з відкритими маршрутами /update/entry та /update/exit. Дані передаються у базу, а HTML-інтерфейс адміністратора оновлює кількість зайнятих місць у режимі реального часу.

Цей варіант дозволяє повністю імітувати інтелектуальну поведінку ESP32 та інтеграцію з веб-системою без використання хмарних сервісів або зовнішнього обладнання.

ДОДАТОК В

Серверна частина: Flask API та база даних

Для обробки запитів від ESP32 або браузера на ПК створено Flask-сервер, який відповідає за оновлення та відображення кількості зайнятих місць на паркінгу.

1. Код Flask-сервера (app.py)

```
from flask import Flask, jsonify
import sqlite3
import os
from flask_cors import CORS

app = Flask(__name__)
CORS(app)
DB_PATH = "parking.db"

def init_db():
    if not os.path.exists(DB_PATH):
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()
        cursor.execute("CREATE TABLE status (name TEXT, value INTEGER)")
        cursor.execute("INSERT INTO status VALUES ('occupied', 0)")
        conn.commit()
        conn.close()
        print("Базу даних створено!")

init_db()

@app.route("/update/<event>")
def update(event):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    if event == "entry":
        cursor.execute("UPDATE status SET value = value + 1 WHERE name =
'occupied'")
    elif event == "exit":
        cursor.execute("UPDATE status SET value = value - 1 WHERE name =
'occupied'")
    conn.commit()
    conn.close()
    return jsonify({"status": "updated"})

@app.route("/status")
def status():
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
```

```
    cursor.execute("SELECT value FROM status WHERE name = 'occupied'")
    value = cursor.fetchone()[0]
    conn.close()
    return jsonify({"occupied": value})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

2. База даних parking.db автоматично створюється, якщо її немає. Таблиця status містить поле occupied, яке змінюється при кожному запиті.

3. Доступ до API здійснюється через маршрути:

– GET /status – повертає кількість зайнятих місць;

– GET /update/entry – збільшує лічильник;

– GET /update/exit – зменшує лічильник.

Сервер підтримує CORS, тому може приймати запити з браузера чи інших клієнтів у локальній мережі.

ДОДАТОК Г

Веб-інтерфейс адміністратора

Для візуалізації стану паркінгу та взаємодії з системою створено простий HTML-інтерфейс, який автоматично отримує дані про кількість зайнятих місць у реальному часі з Flask-сервера.

1. Структура index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Адміністратор Паркінгу</title>
  <style>
    body {
      font-family: sans-serif;
      background: #f4f4f4;
      padding: 20px;
    }
    .status {
      font-size: 2em;
      margin-bottom: 20px;
    }
    .full {
      color: red;
    }
    .ok {
      color: green;
    }
  </style>
</head>
<body>
  <h1>Система паркування</h1>
  <div class="status" id="status">Завантаження...</div>

  <button onclick="sendEvent('entry')">Імітувати в'їзд</button>
  <button onclick="sendEvent('exit')">Імітувати виїзд</button>

  <script>
    async function updateStatus() {
      try {
        const response = await fetch("http://localhost:5000/status");
        const data = await response.json();
        const el = document.getElementById("status");
```

```
        const capacity = 5;
        const occupied = data.occupied;

        el.innerText = `Зайнято ${occupied} із ${capacity}`;
        el.className = occupied >= capacity ? "status full" : "status ok";

      } catch (e) {
        document.getElementById("status").innerText = "Помилка з'єднання!";
      }
    }

    async function sendEvent(type) {
      try {
        await fetch(`http://localhost:5000/update/${type}`);
        updateStatus();
      } catch {
        alert("Не вдалося надіслати подію!");
      }
    }

    setInterval(updateStatus, 1000);
    updateStatus();
  </script>
</body>
</html>
```

2. Принцип роботи:

– функція updateStatus() виконує AJAX-запит до /status кожну секунду;

– кнопки імітують події в’їзду/виїзду, надсилаючи запити /update/entry та /update/exit;

– дані автоматично оновлюються на екрані без перезавантаження сторінки.

Інтерфейс простий, але повністю відображає логіку роботи системи та дозволяє демонструвати роботу навіть без фізичних сенсорів чи ESP32.

ДОДАТОК Д

Демонстрація роботи системи та макет

1. Макет наведено на рис. Д.1:

– кнопки з’єднані з GPIO4 та GPIO5;

– світлодіод до GPIO2 (через резистор).

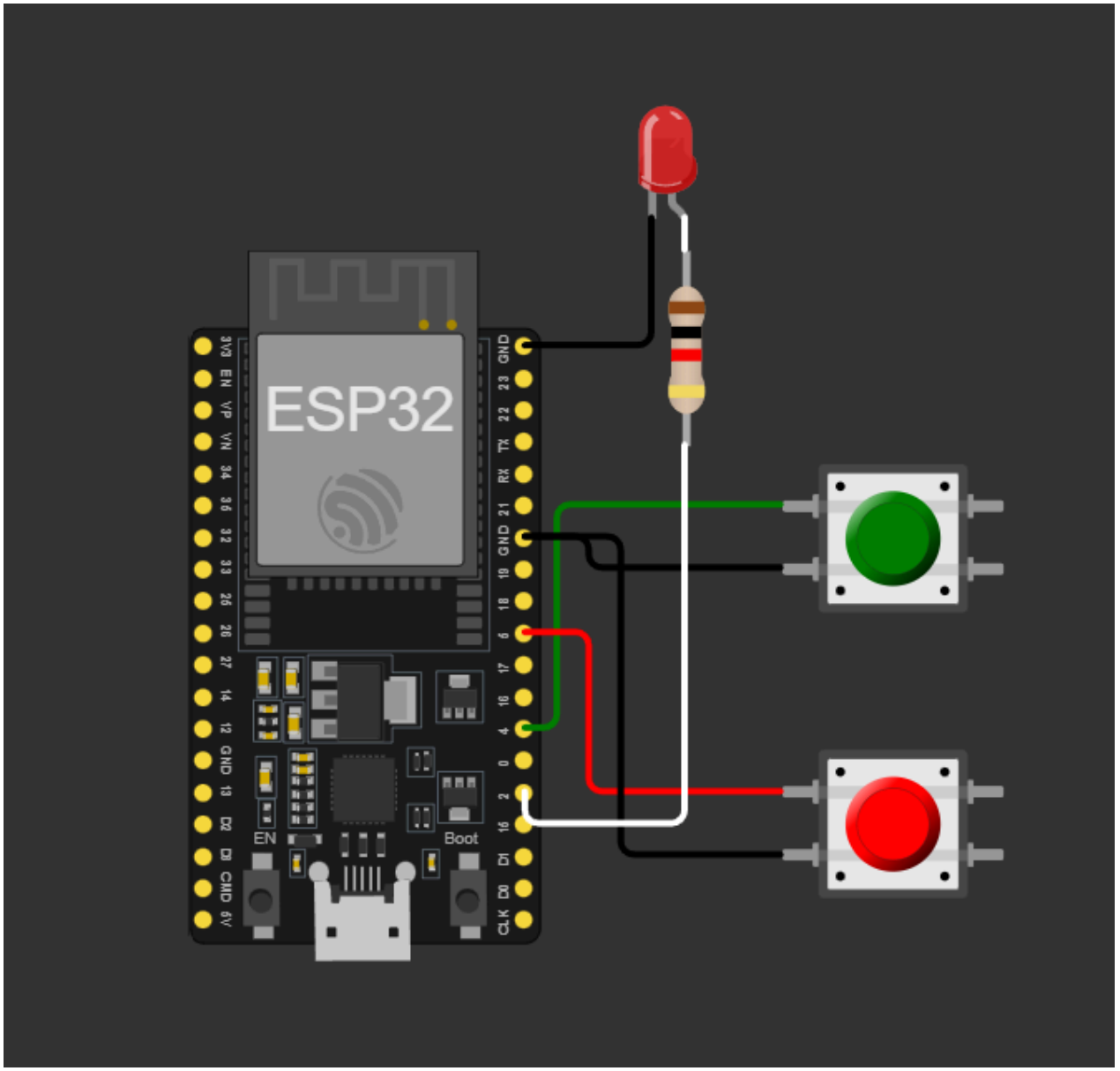
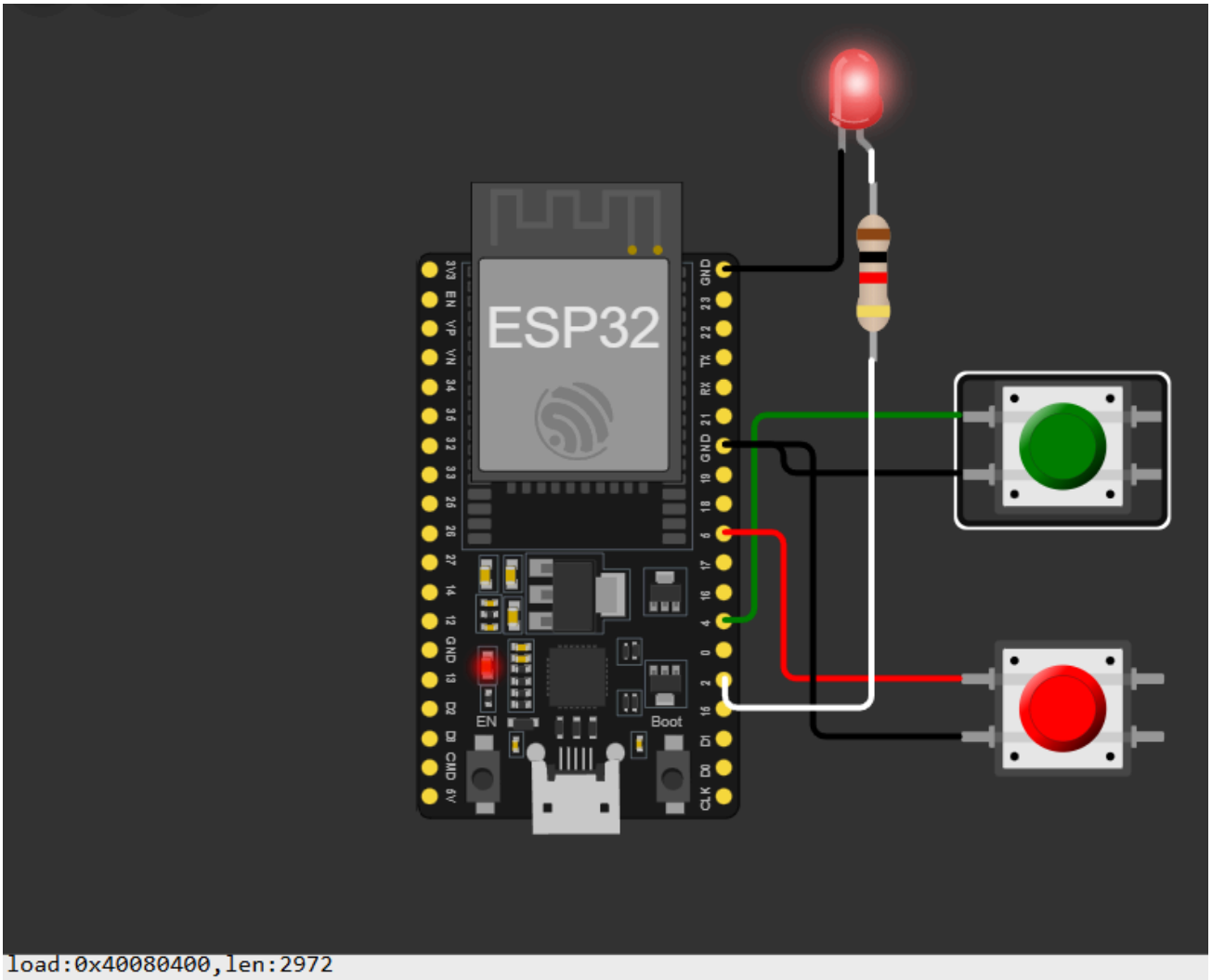


Рисунок Д.1 – Схема з'єднання

Поводження системи у випадку коли на парковці немає вільних місць наведено на рис. Д.2.



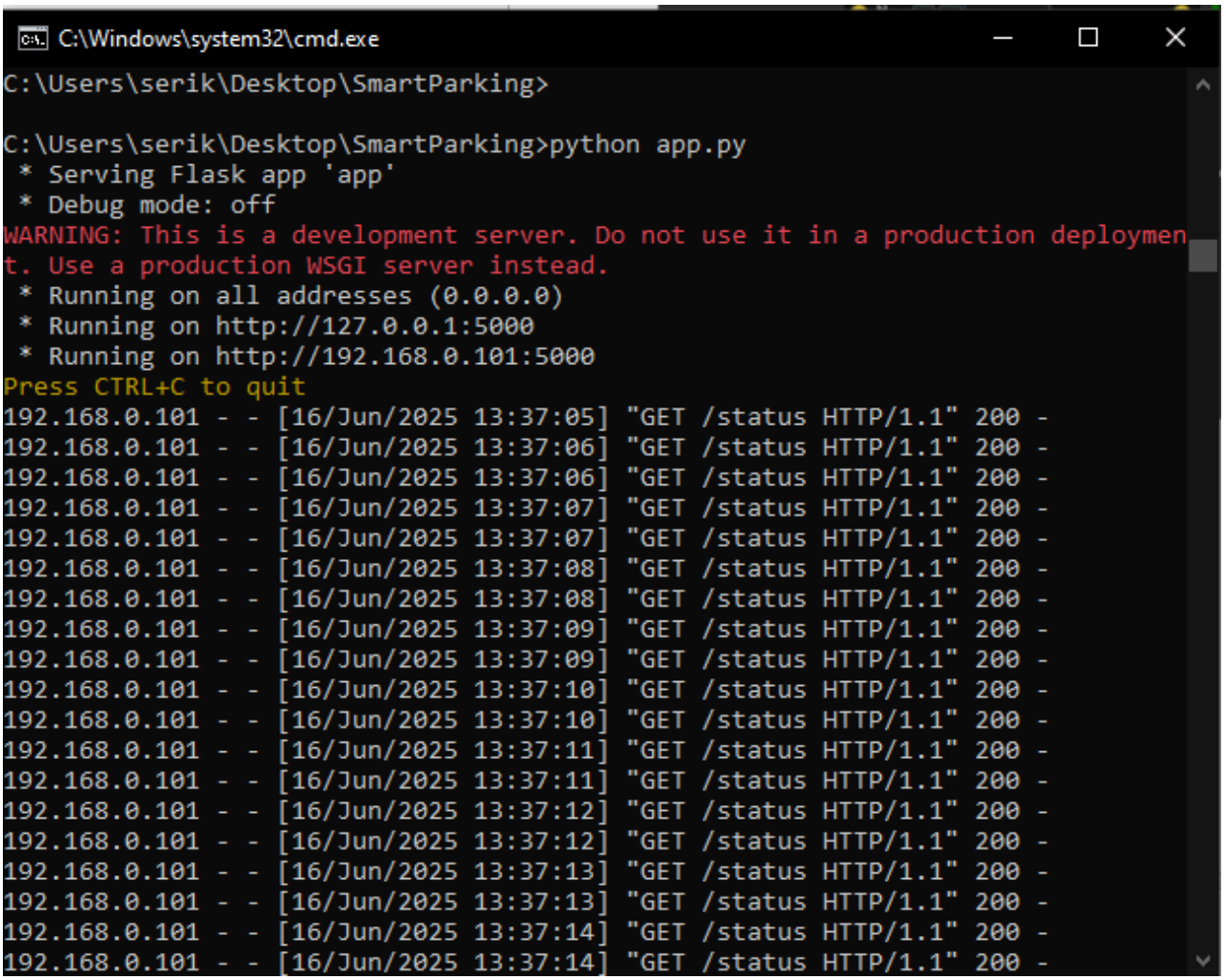

```
load:0x40080400,len:2972
entry 0x400805dc
Car Entered: 1
Car Entered: 2
Car Entered: 3
Car Entered: 4
Car Entered: 5
```

Рисунок Д.2 – Коли на парковці немає місць

2. Роботу Flask-серверу наведно на рис. Д.3 :

– запущений з терміналу;

– приймає запити /status та /update.



Рисуно Д. 3 – Термінал Flask

3. Інтерфейс показує актуальний статус (рис. Д.4):

– «Зайнято 4 з 5»;

– кольорова індикація.

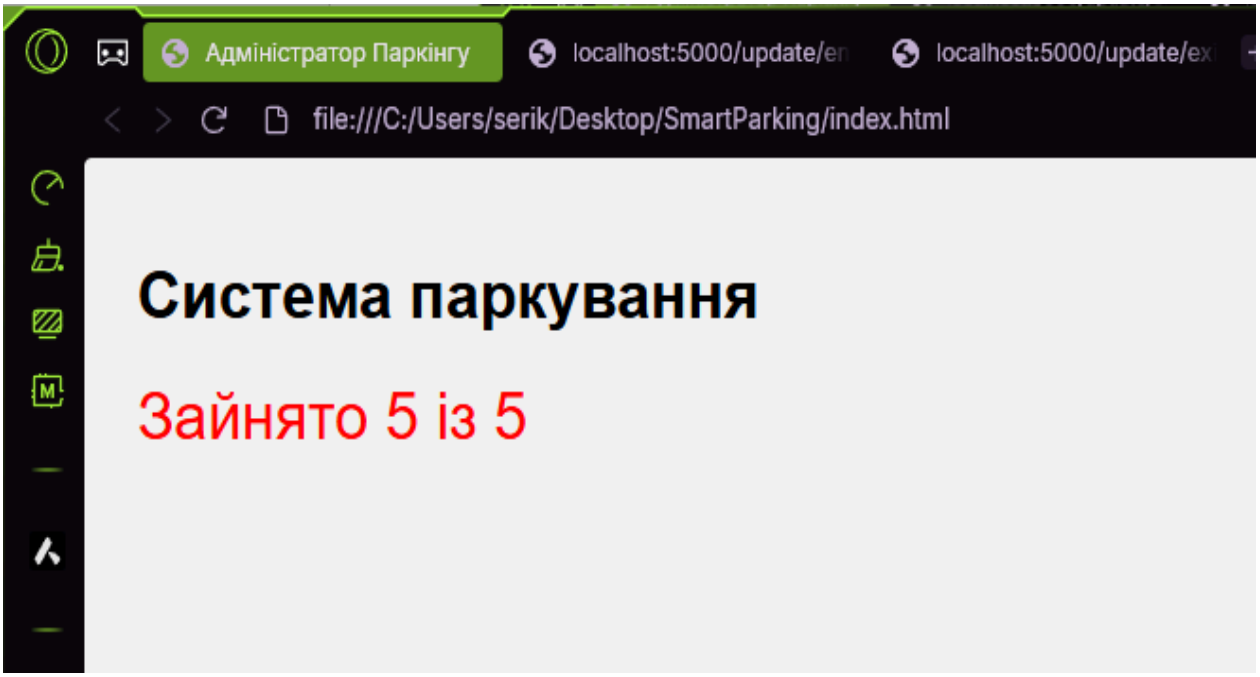


Рисунок Д. 4 – Інтерфейс адміністратора

ДОДАТОК Е

Демонстаційний матеріал

№ документа	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1	ГЮИК.505200.003 ПЗ	Пояснювальна записка	А4, 52с.
2		Створення віртуальної	А4, 2с.
		моделі в Wokwi	
3		Підключення ESP32 до	А4, 2с.
		веб-застосунку	
4		Серверна частина: Flask	А4, 2с.
		API та база даних	
5		Веб-інтерфейс	А4, 2с.
		адміністратора	
6		Демонстрація роботи	А4, 4с.
		системи та макет	
7		Демонстаційний	А4, 1с.
		матеріал	

Змін.	Арк.	Номер докум.	Підпис	Дата	ГЮИК.505200.003 ВД			
Розробка		Запорожченко С. О.						
Перевірив		Стародубцев М. Г.			Розроблення автоматизованої системи розумного паркування	Літера	Аркуш	Аркушів
						Н	1	1
Н. контроль		Стародубцев М. Г.				Кафедра КІТАМ ХНУРЕ		
Затвердив		Невлюдов І. Ш.						