Institut Puig Castellar
Santa Coloma de Gramenet

ParqueoYA

(Projecte de desenvolupament)

CFGS Administració de Sistemes Informàtics i Xarxes

Resum del projecte (màxim 250 paraules):

Aquest projecte tracta sobre fer un sistema que ajudi a saber si una plaça d'aparcament està ocupada o lliure. La idea és fer-ho de manera automàtica, sense que algú hagi d'anar a mirar-ho. La temàtica és ajudar a millorar els aparcaments a les ciutats, fent-los més fàcils d'utilitzar.

L'objectiu principal del projecte és poder detectar si hi ha un cotxe aparcat en una plaça i mostrar aquesta informació en una pàgina web perquè qualsevol persona la pugui consultar des del mòbil o ordinador.

Per aconseguir això, hem utilitzat sensors d'ultrasons connectats a una placa Arduino que detecten si hi ha un vehicle a sobre. També hem fet servir leds que s'encenen de diferent color segons si la plaça està ocupada o lliure. Tot això està connectat amb cables i una protoboard, i ho hem programat perquè enviem les dades a un servidor. Aquest servidor, creat amb Python i Flask, guarda els canvis i mostra la informació en un mapa web fet amb HTML, CSS i JavaScript. També hem utilitzat ngrok perquè la pàgina es pugui veure des de fora de la xarxa.

En resum, hem aconseguit fer un sistema senzill però útil per saber en temps real l'estat d'unes places d'aparcament. Ha sigut un projecte pràctic per aprendre a combinar programació, electrònica i web d'una manera real.

Paraules clau (entre 4 i 8):

ParqueoYA, places, lliures, ocupades, aparcar, estacionar, lloc, trobar

Abstract (in English, 250 words or less):

This project is about making a system that helps to know if a parking space is occupied or free. The idea is to do it automatically, without anyone having to go and look. The theme is to help improve parking lots in cities, making them easier to use.

The main objective of the project is to be able to detect if there is a car parked in a space and display this information on a web page so that anyone can consult it from their mobile phone or computer.

To achieve this, we have used ultrasonic sensors connected to an Arduino board that detect if there is a vehicle on it. We have also used LEDs that light up in different colors depending on whether the space is occupied or free. All of this is connected with cables and a protoboard, and we have programmed it to send the data to a server. This server, created with Python and Flask, saves the changes and displays the information on a web

map made with HTML, CSS and JavaScript. We have also used ngrok so that the page can be viewed from outside the network.

In short, we have managed to make a simple but useful system to know in real time the status of parking spaces. It has been a practical project to learn how to combine programming, electronics and the web in a real way.

Keywords (entre 4 i 8):

ParqueoYA, spaces, free, occupied, parking, place, search

Índex

Llista de figures

1 Introducció

El nostre projecte, anomenat ParqueoYA, consisteix en una pàgina web on qualsevol usuari pot consultar, en temps real, la disponibilitat d'aparcament al carrer. A través d'un mapa interactiu, es mostren punts de diferents colors que indiquen l'estat actual de cada plaça d'aparcament: verd si està lliure i vermell si està ocupada. Aquesta informació es recull a través de sensors instal·lats físicament en els llocs d'aparcament.

A més del mapa, hem desenvolupat un apartat anomenat "Logs", que mostra una taula estructurada amb les següents columnes: Data i hora, Sensor, i Estat. Aquest registre permet als usuaris consultar el comportament històric de les places i fer-se una idea de quines zones solen estar més ocupades en determinades hores o dies. Si es consulta durant el mateix dia, la informació es mostra en temps real, actualitzant-se automàticament quan es detecta un canvi d'estat en algun sensor.

Tota aquesta infraestructura funciona gràcies a una placa Arduino, sensors ultrasònics i llums LED. El funcionament físic és senzill: quan un sensor detecta un objecte a menys de 12 cm, s'encén el LED vermell i s'apaga el verd. Quan no hi ha cap obstacle, s'encén el LED verd, indicant que la plaça està lliure. Els sensors i LEDs estan connectats a l'Arduino mitjançant cables macho-macho, formant un sistema senzill però eficient.

1.1 Context

Actualment, l'aparcament en zones urbanes és una de les problemàtiques més comunes que afecten tant a conductors com a ciutats. Moltes persones passen una quantitat considerable de temps buscant aparcament, fet que incrementa el trànsit, el consum de carburant i la contaminació. Tot i que existeixen aplicacions que informen sobre pàrquings públics o privats, no solen oferir dades precises i en temps real sobre aparcaments al carrer, que és la forma més habitual de deixar el vehicle a moltes ciutats.

Amb l'avanç de la tecnologia, cada cop es fa més necessari desenvolupar sistemes intel·ligents que millorin l'eficiència urbana. El nostre projecte s'emmarca dins d'aquest context, proposant una solució assequible i adaptable que pot ser utilitzada per ajuntaments o comunitats locals per a millorar la gestió de l'espai públic i reduir el temps de cerca d'aparcament.

1.2 Justificació

La realització d'aquest projecte és rellevant perquè respon a una necessitat creixent dins de les ciutats modernes: la millora de la mobilitat urbana i la gestió eficient de l'espai públic. Trobar aparcament en zones urbanes és un repte diari per a molts conductors, i aquest problema no només causa frustració, sinó que també contribueix a l'increment del trànsit, el consum de combustible i la contaminació ambiental. Davant d'aquest escenari, ParqueoYA proposa una solució tecnològica simple però efectiva, basada en l'ús de sensors i una plataforma web que mostra en temps real la disponibilitat de les places d'aparcament.

Aquest projecte és també una oportunitat d'aprenentatge pràctic en el marc del nostre cicle formatiu. Ens ha permès aplicar coneixements de diferents àmbits com l'electrònica, la programació, la gestió de dades i el disseny web, integrant-los en un sistema funcional amb aplicació real. A més, la seva estructura escalable i de baix cost fa que sigui perfectament adaptable a altres entorns o fins i tot a una implementació a gran escala. En

definitiva, es tracta d'un projecte que combina l'interès social i tecnològic amb el desenvolupament personal i professional dels integrants.

1.3 Objectius

1.3.1 Objectiu general

Desenvolupar un sistema intel·ligent de monitorització d'aparcaments en temps real mitjançant sensors ultrasònics connectats a una plataforma web que permeti als usuaris visualitzar, des de qualsevol lloc, la disponibilitat d'espais lliures o ocupats a la via pública.

1.3.2 Objectius específics

- Desenvolupar una pàgina web interactiva que mostri en temps real l'estat d'ocupació dels aparcaments al carrer mitjançant punters sobre un mapa.
- Programar una lògica Arduino funcional que detecti la presència de vehicles mitjançant sensors ultrasònics i indiqui visualment l'estat mitjançant LEDs.
- Establir un sistema de comunicació entre l'Arduino i el servidor mitjançant WiFi, que permeti enviar l'estat dels sensors en temps real.
- Crear una base de dades SQLite per registrar els logs de cada sensor amb data i hora, accessible des de la web.
- Permetre la consulta d'històrics mitjançant una taula de logs a la pàgina web, amb filtratge per data i actualització automàtica.
- Utilitzar la tecnologia ngrok per fer accessible la web des de qualsevol lloc del món, sense limitacions de xarxa local.
- Dissenyar un sistema físic funcional i estable utilitzant protoboard, cables macho-macho, sensors ultrasònics i LEDs, garantint un muntatge clar i ordenat.
- Implementar un sistema de control visual amb LEDs vermells i verds que indiqui l'ocupació o disponibilitat de la plaça.
- Integrar totes les parts del projecte juntes: maquinari, servidor, base de dades i interfície web de manera fluida i sincronitzada.

1.4 Estratègia i planificació del projecte

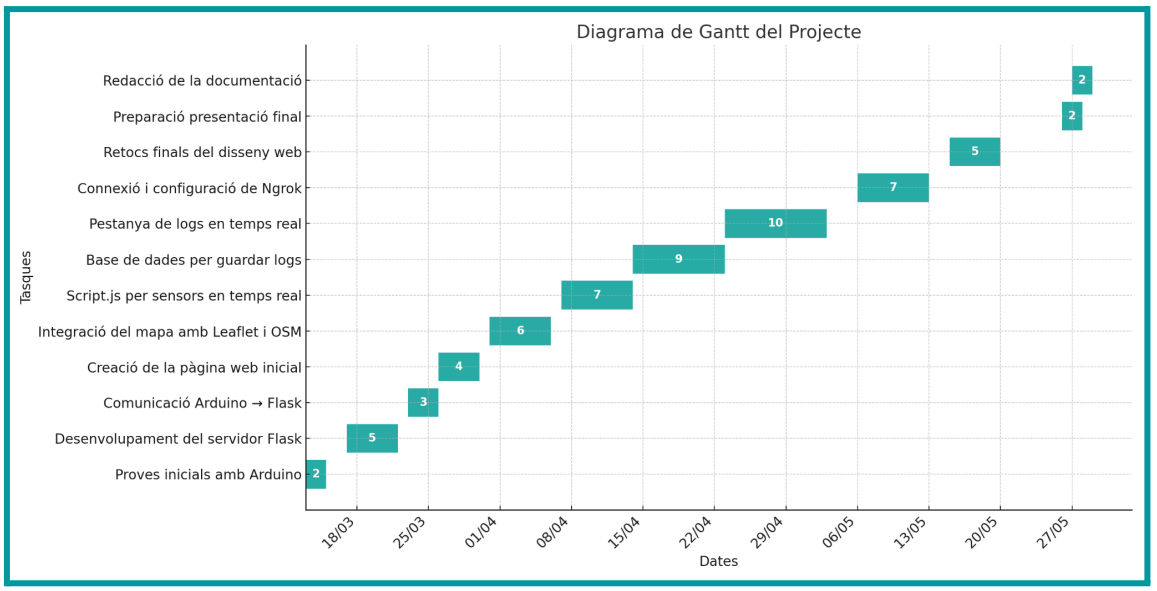
Per dur a terme aquest projecte, es van considerar diverses estratègies possibles: adaptar un sistema ja existent de gestió d'aparcaments intel·ligents, utilitzar plataformes comercials de sensors o desenvolupar des de zero un producte propi que integrés tots els components. Finalment, es va optar per desenvolupar un producte nou. Aquesta estratègia permet tenir un control total sobre el funcionament del sistema, des dels sensors, connexions, LEDs, fins a la comunicació, base de dades, interfície web. Això ha fet possible personalitzar completament el projecte, adaptar-lo als requeriments del nostre entorn i aprendre durant tot el procés de desenvolupament.

Un altre motiu per escollir aquesta estratègia és la viabilitat tècnica i econòmica. Utilitzar Arduino, sensors econòmics d'ultrasons i eines gratuïtes com SQLite, Flask i ngrok ha

permès desenvolupar un sistema funcional amb cost molt reduït. L'estratègia triada també és molt adequada per assolir els objectius específics, ja que:

- Ens permet controlar i modificar la lògica del funcionament dels sensors i les comunicacions.
- Facilita la integració en temps real amb la pàgina web i la base de dades.
- Ens dona llibertat per escalar el sistema amb més sensors o noves funcionalitats.

1.5 Metodologia de treball



1.6 Estudi econòmic i pressupostari

Inventari de components

Component	Quantitat	Preu unitari (€)	Cost total (€)
Arduino UNO R4 Wifi	1	34,90	34,90
Sensor ultrasònic HC-SR04	1 pack 5 sensores	12,99	12,99
LEDs i resistències	1 pack	14,99	14,99
cables	1 pack macho macho	5,90	5,90
Protoboard	1 pack 6 protboard	6,79	6,79
Font d'alimentació	1	9,00	9,00

USB	
Cost total per unitat	**84,57 €**

Òbviament això es escalable y si es magnifiques la escala creixeria els costos.

2 Descripció del projecte

2.1.1 Anàlisi de requisits [projecte d'implementació]

Requisits funcionals

Per a considerar la implementació com a finalitzada, el sistema ha de satisfer els següents requisits funcionals:

- R1. Detecció d'ocupació de places de pàrquing: El sistema ha de detectar en temps real si una plaça està ocupada o lliure mitjançant sensors ultrasònics.
- R2. Comunicació amb el servidor: El dispositiu Arduino ha de connectar-se per WiFi a una xarxa local i enviar periòdicament l'estat dels sensors al servidor.
- R3. Emmagatzematge de logs: El servidor ha de registrar els canvis d'estat dels sensors en una base de dades SQLite, incloent la data i hora.
- R4. Visualització en mapa web: La interfície web ha de mostrar l'estat actual de cada plaça de pàrquing en un mapa mitjançant icones de color.
- R5. Visualització dels logs: L'aplicació web ha de permetre consultar els últims canvis d'estat dels sensors en una taula.
- R6. Navegació per seccions: L'usuari ha de poder accedir a seccions com "Com funciona", "Qui som" i "Logs" des del menú principal.
- R7. Actualització en temps real: Els canvis d'estat s'han d'actualitzar automàticament a la pàgina web cada segon sense necessitat de recarregar la pàgina.

Requisits no funcionals

- RF1. Robustesa: El sistema ha de ser capaç de continuar funcionant davant errors puntuals de connexió o mesures incorrectes dels sensors.
- RF2. Facilitat d'ús: La interfície web ha de ser clara, intuïtiva i accessible des de dispositius mòbils.
- RF3. Eficiència: L'enviament de dades i les actualitzacions a la web han de tenir una latència mínima d'un segon.
- RF4. Escalabilitat: El sistema hauria de poder adaptar-se fàcilment per a gestionar més sensors o places.

- RF5. Seguretat mínima: Les dades es transmeten dins d'una xarxa local controlada, però s'ha de validar el contingut rebut al servidor per evitar injeccions o errors.

Requisits previs mínims del sistema

Maquinari:

- Arduino Uno R4 WiFi
- 2 sensors ultrasònics HC-SR04
- 4 LEDs (2 verds, 2 vermells)
- Cables mascle-mascle i 2 protoboard
- Ordinador amb màquina virtual per allotjar el servidor

Xarxa / serveis de xarxa:

- Connexió WiFi local amb IP estàtica
- Accés al port 5000 per comunicació entre Arduino i servidor
- Ngrok instal·lat al servidor per crear un túnel HTTP segur i poder accedir a la web desde fora de la red local.

Sistema operatiu:

- Ubuntu

Aplicacions i dependències:

- Python 3
- Flask
- SQLite
- Chrome
- Leaflet
- OpenStreetMap

2.1.2 Previsió de tasques d'investigació [projecte d'investigació]

Buscar quines tecnologies o components podem fer servir: Mirarem quins programes, eines o components existeixen que ens podrien anar bé. També mirarem si són fàcils d'utilitzar, si són compatibles entre ells i si es poden adaptar al nostre projecte.

Aprendre com funcionen alguns components concrets: Algunes peces o programes poden ser una mica complicats, així que volem entendre bé com funcionen abans de començar a fer proves. Potser farem petits experiments per veure com responen.

Comparar diferents maneres de fer una mateixa cosa: Per algunes parts del projecte, potser hi ha més d'una manera de fer-les. Per exemple, diferents mètodes per analitzar dades o controlar un motor. Compararem quins són millors segons el que necessitem.

Fer proves senzilles per veure què funciona millor: Un cop tinguem algunes opcions, les posarem a prova per veure quina va millor. Farem proves senzilles per veure quins resultats donen.

Valorar els resultats i decidir com continuar: Analitzarem com han anat les proves i decidirem si seguim pel mateix camí o si hem de canviar alguna cosa, com per exemple fer servir una altra eina o modificar alguna part del projecte.

Mirar si tot es pot ajuntar bé i funciona com un tot: Un cop tenim diverses parts que funcionen, comprovarem si es poden combinar bé i si tot plegat funciona conjuntament sense problemes.

Buscar maneres de millorar o fer-ho més eficient: Si tot va bé, mirarem si es pot fer més ràpid, més barat, o més senzill. Potser hi ha detalls que es poden polir.

Treure conclusions i pensar com es podria seguir treballant: Finalment, resumirem tot el que hem après i pensarem si val la pena seguir millorant el projecte, fer-ne una segona versió o aplicar-ho a una altra idea.

2.2 Tecnologies

2.2.1 Comparativa de les tecnologies valorades

Tipo	Tecnologia	Pros	Contres
Microcontroladors	Arduino UNO R4 WiFi	WiFi integrat, més ràpid (RA4M1 32-bit), més memòria, suport oficial	Nou (menys documentació), pot requerir biblioteques específiques
	Arduino MKR WiFi 1010	WiFi integrat, suport oficial	Preu més alt
Sensors	HC-SR04	Econòmic, fàcil d'usar, bona precisió a curta distància	Pot fallar amb superfícies toves o absorbents
	IR	Molt econòmic	Poc fiable amb llum ambiental variable

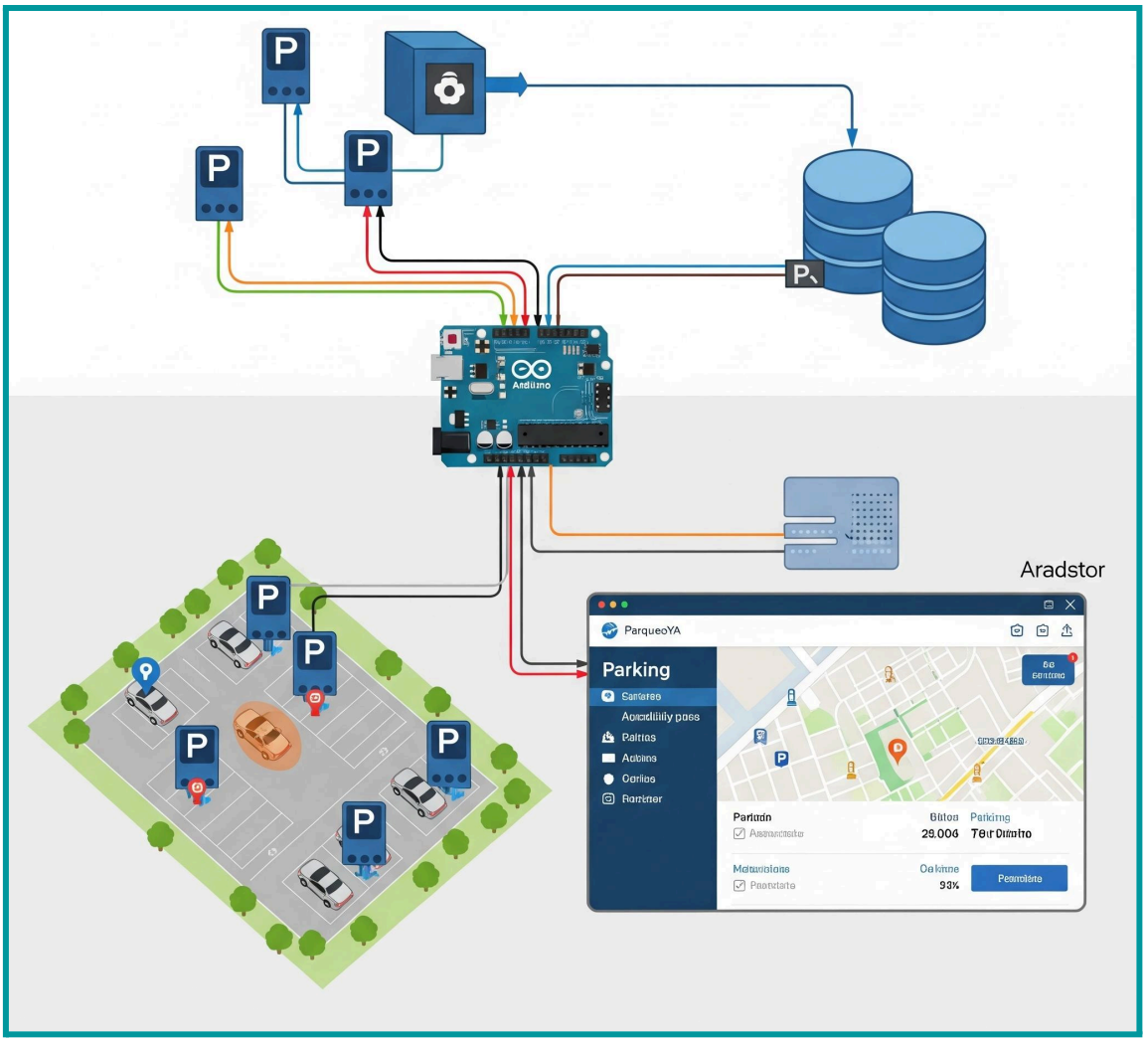
Backend	Flask (Python)	Lleuger, fàcil de desplegar, molt compatible amb sensors i SQLite	No ideal per a projectes grans o molt complexos
	Django (Python)	Més robust i estructurat	Massa pesat per a un projecte petit
BBDD	SQLite	Lleuger, simple, no cal servidor	No recomanat per a ús concurrent massiu
	MySQL/PostgreSQL	Potents, segures, suport multiusuari	Necessiten instal·lació i configuració addicional
Frontend - Mapes	Leaflet + OSM	Open source, fàcil integració, sense cost	Més simple que alternatives com Google Maps
	Google Maps API	Molt complet i detallat	Cost per ús, requisit de clau API
Túnel a l'exterior	Ngrok	Fàcil d'usar, segura, zero configuració de router	L'accés gratuït pot caducar, canvi de URL constant

2.2.2 Tecnologies escollides

Tecnologia	Motiu de la tria
Arduino UNO R4 WiFi	Microcontrolador modern amb WiFi integrat, més potència i memòria que l'UNO R3, ideal per IoT.
Sensor HC-SR04	Econòmic, fàcil d'integrar i prou precís per detectar distàncies a vehicles.
Flask (Python)	Framework lleuger per al backend, fàcil de desplegar i integrar amb bases de dades i serveis web.

SQLite	Base de dades local, sense necessitat de servidor, adequada per a aplicacions petites i portables.
Leaflet + OpenStreetMap	Solució de mapes gratuïta i oberta, compatible amb visualitzacions interactives sense llicència API.
Ngrok	Túnel segur per exposar el servidor local a Internet sense configurar el router.
JavaScript (.js)	Permet la manipulació del mapa i la visualització dinàmica de dades a la interfície web.

2.3 Estructura del projecte



El diagrama mostra com els sensors es connecten a un Arduino, que envia dades a un servidor i una base de dades, i com una interfície web mostra la informació de l'aparcament.

2.4 Descripció dels components

Arduino UNO R4 WiFi amb sensors ultrasònics HC-SR04

- **Funcionalitat:** El microcontrolador Arduino UNO R4 WiFi està connectat a dos sensors ultrasònics HC-SR04 que mesuren la distància per detectar si una plaça d'aparcament està ocupada o lliure.

- **Funcionament intern:** El sensor ultrasònic emet un pols ultrasònic i mesura el temps que triga en rebre l'eco. Aquesta durada es converteix en distància.
  Quan la distància detectada és menor o igual a 12 cm, el sistema considera la plaça com "ocupada", sinó "lliure". L'Arduino envia l'estat de cada sensor via WiFi a un servidor mitjançant una petició HTTP POST en format JSON.

- **Tecnologies:** Arduino UNO R4 WiFi, sensor HC-SR04, biblioteca WiFiS3 per la comunicació WiFi.

Servidor Flask (Python)

- **Funcionalitat:** Rep les dades enviades per l'Arduino a través d'una API REST, emmagatzema els estats a una base de dades SQLite, i serveix la pàgina web per a visualitzar l'estat dels sensors en temps real.
- **Funcionament intern:** El servidor disposa d'una ruta /estado que accepta dades POST amb l'identificador del sensor i el seu estat. Cada canvi d'estat es registra a la base de dades, però només si l'estat ha canviat respecte l'últim registrat, per evitar dades duplicades. També proporciona rutes GET per obtenir l'estat actual dels sensors i els logs històrics.
- **Tecnologies:** Python, Flask, SQLite, Ngrok per exposar el servidor local a Internet.

Base de dades SQLite

- **Funcionalitat:** Emmagatzema els logs dels canvis d'estat dels sensors amb timestamp, estat i identificador.
- **Funcionament intern:**És una base de dades lleugera integrada en un fitxer, que permet consultes ràpides per recuperar l'històric i mostrar-lo a la interfície web.
- **Tecnologies:** SQLite, integrada dins el servidor Flask.

Client Web

- **Funcionalitat:** Mostra un mapa interactiu on es visualitzen les places d'aparcament monitoritzades amb colors que indiquen si estan ocupades (vermell) o lliures (verd). També permet veure informació addicional, com logs en temps real i seccions descriptives del projecte.
- **Funcionament intern:** El client fa peticions periòdiques al servidor per actualitzar l'estat dels sensors i actualitzar el mapa dinàmicament. Utilitza Leaflet i OpenStreetMap per a la renderització del mapa i JavaScript per gestionar la interacció i les peticions.
- **Tecnologies:** HTML, CSS, JavaScript, Leaflet, OpenStreetMap.

Ngrok

. **Funcionalitat:**
Permet exposar el servidor Flask a Internet a través d'un túnel segur, sense necessitat de configurar el router o obrir ports.

. **Funcionament intern:** Ngrok crea una URL pública que redirigeix el tràfic cap al servidor local, facilitant així la comunicació remota amb l'Arduino o altres clients.

2.5 Definició de les tasques [projecte d'investigació]

Funcionalitat principal: Monitoratge en temps real de l'estat de les places d'aparcament.

- Lectura de sensors: Cada sensor mesura la distància i determina si hi ha un vehicle.
- Codi Arduino: Envia l'estat lliure o ocupat de cada sensor via WiFi al servidor.
- Servidor Flask: Processa les dades rebudes i les actualitza per a la web.
- Web: Mostra un mapa del centre amb punts verds (lliure) i vermells (ocupat).
- Actualització en temps real: Mitjançant peticions POST
- Base de dades: Guarda l'historial d'estats per a consultes futures y mostra en temps real nous registres
- Accés remot : Mitjançant ngrok, per accedir des de fora del centre.

2.6 Definició de les funcionalitats [projecte d'implementació]

La solución implementada permite monitorizar en tiempo real el estado de ocupación de las plazas de aparcamiento mediante sensores ultrasónicos conectados a un Arduino. La información se muestra en una página web con un mapa interactivo donde cada plaza aparece marcada como libre u ocupada.

Las funcionalidades principales son:

- **Detección de ocupación**: Cada sensor detecta si hay un vehículo presente y enciende un LED rojo (ocupat) o verde (libre). Esta información se envía automáticamente.

- **Visualización en tiempo real**: Los cambios se actualizan en el mapa al instante gracias al uso de Flask y Flask-SocketIO, sin necesidad de recargar la página.

- **Historial de registros**: Todos los cambios de estado se almacenan en una base de datos SQLite y pueden consultarse filtrando por fecha.

- **Identificación por sensor**: Cada sensor tiene un sensor_id y aparece con su ubicación específica en el mapa.

- **Acceso remoto**: Gracias a Ngrok, el sistema puede visualizarse desde cualquier dispositivo conectado a Internet.

3 Altres capítols

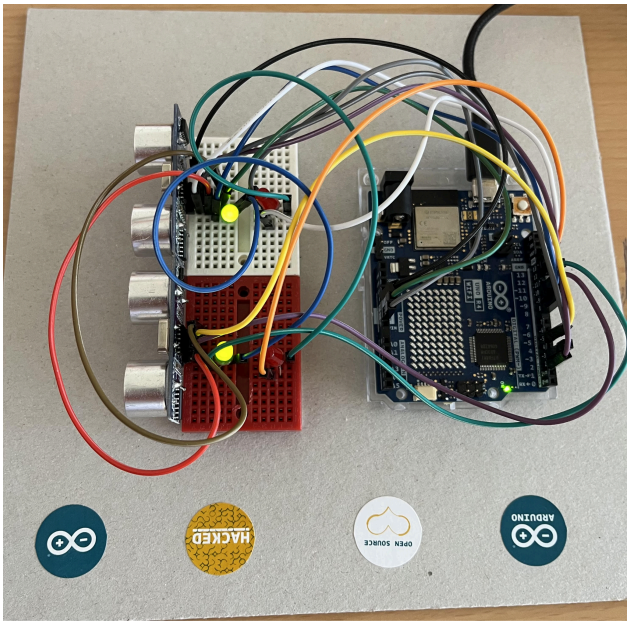
3.1 Arduino

3.1.1 Hardware del Arduino

El cervell del sistema és una placa Arduino UNO R4 WiFi, escollida perquè incorpora la funcionalitat de connexió sense fils, ideal per a projectes IoT com aquest. Aquesta placa controla dos sensors ultrasònics HC-SR04, que s'encarreguen de mesurar la distància entre el sensor i el possible vehicle aparcat. A més, controla quatre LEDs, dos per cada sensor.

Els components electrònics es connecten mitjançant una protoboard, facilitant un muntatge modular i sense soldadures. Els cables macho-macho asseguren connexions directes entre els pins de l'Arduino i els components externs.

Cada sensor es posiciona de manera que pugui detectar l'altura del vehicle estacionat. Quan detecta un objecte a menys de 12 cm, el sistema considera que la plaça està ocupada. Aquest llindar s'ha determinat experimentalment per garantir una detecció fiable sense confondre petits obstacles.




```
  // Sensor 1
  pinMode(trig1, OUTPUT);
  pinMode(echo1, INPUT);
  pinMode(ledR1, OUTPUT);
  pinMode(ledV1, OUTPUT);


  // Sensor 2
  pinMode(trig2, OUTPUT);
  pinMode(echo2, INPUT);
  pinMode(ledR2, OUTPUT);
  pinMode(ledV2, OUTPUT);
}
```

3.1.2 Software del Arduino

El codi controla tot el comportament de l'Arduino. Aquest es divideix en diverses parts clau:

1. **Connexió WiFi:** Mitjançant la llibreria WiFiS3, el dispositiu es connecta a una xarxa definida per l'usuari. Aquesta connexió és imprescindible per enviar dades al servidor.

```
void setup() {
  Serial.begin(9600);
  WiFi.begin(ssid);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.println("Conectando a WiFi...");
    delay(1000);
  }
  Serial.println("WiFi conectado");
```

2. **Mesura de distàncies:** Cada sensor envia ultrasons a través del pin trig i mesura el temps que triguen en tornar al pin echo. Aquest temps es converteix a centímetres:

```
long medirDistancia(int trig, int echo) {
  digitalWrite(trig, LOW);
  delayMicroseconds(2);
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);
  long duracion = pulseIn(echo, HIGH, 30000);
  long distancia = duracion * 0.034 / 2;
  return (distancia > 0 && distancia < 300) ? distancia : 999;
}
```

3. **Detecció d'estat i actualització de LEDs**: Es compara la distància amb el llindar de 12 cm. Si hi ha un canvi respecte a l'estat anterior, es fa una crida a enviarEstado():

```
void loop() {
 // Sensor 1
 long d1 = medirDistancia(trig1, echo1);
 String nuevoEstado1 = d1 <= 12 ? "ocupado" : "libre";
 if (nuevoEstado1 != estado1) {
   estado1 = nuevoEstado1;
   enviarEstado("sensor1", estado1);
 }
 digitalWrite(ledR1, estado1 == "ocupado" ? HIGH : LOW);
 digitalWrite(ledV1, estado1 == "libre" ? HIGH : LOW);


 delay(100); // separación entre sensores


 // Sensor 2
 long d2 = medirDistancia(trig2, echo2);
 String nuevoEstado2 = d2 <= 12 ? "ocupado" : "libre";
 if (nuevoEstado2 != estado2) {
   estado2 = nuevoEstado2;
   enviarEstado("sensor2", estado2);
 }


 //  Invertimos el comportamiento porque los LEDs están cruzados
 digitalWrite(ledR2, estado2 == "libre" ? HIGH : LOW);
 digitalWrite(ledV2, estado2 == "ocupado" ? HIGH : LOW);
```

4. **Enviament d'estat al servidor**: Utilitzant WiFiClient, l'Arduino crea una petició HTTP POST que envia l'estat actual en format JSON.

```
void enviarEstado(String id, String estado) {
  String payload = "{\"sensor_id\":\"" + id + "\",\"estado\":\"" + estado + "\"}";


  if (client.connect(server, port)) {
    client.println("POST /estado HTTP/1.1");
    client.println("Host: " + String(server));
    client.println("Content-Type: application/json");
    client.print("Content-Length: ");
    client.println(payload.length());
    client.println();
    client.println(payload);
    client.stop();
    Serial.println("Enviado: " + payload);
  } else {
    Serial.println("Error al conectar con servidor");
  }
}
```

Aquesta estructura permet una integració directa amb el servidor Flask que processa dades JSON.

3.2 Servidor Flask

El servidor web és un component fonamental del sistema, ja que actua com a intermediari entre l'Arduino i la pàgina web. Aquest ha estat desenvolupat utilitzant Flask, un microframework de Python molt lleuger però potent, ideal per projectes on cal una API REST personalitzada i ràpida de desplegar.

```
# Guardar log solo si ha cambiado (sin esperar tiempo)
def guardar_log_si_corresponde(sensor_id, estado_actual):
    ahora = datetime.now()

    if sensor_id not in ultimo_log:
        ultimo_log[sensor_id] = ahora
        ultimo_estado_guardado[sensor_id] = ""

    ha_cambiado = (estado_actual != ultimo_estado_guardado[sensor_id])

    if ha_cambiado:
        conn = sqlite3.connect('logs.db')
        c = conn.cursor()
        c.execute("INSERT INTO logs (timestamp, estado, sensor_id) VALUES (?, ?, ?)",
                  (ahora.isoformat(), estado_actual, sensor_id))
        conn.commit()
        conn.close()
```

3.2.1 Arquitectura General

El servidor Flask realitza les següents funcions:

1. Rep dades dels sensors Arduino a través de peticions HTTP POST amb format JSON.
2. Emmagatzema aquestes dades de manera persistent en una base de dades SQLite.
3. Proporciona dades actualitzades al navegador web mitjançant rutes GET.
4. Serveix la pàgina web al navegador amb les rutes HTML i gestiona estàtics.

Rutes principals implementades

1. POST /estado Aquesta ruta és cridada per l'Arduino cada cop que detecta un canvi d'estat en un sensor. L'objectiu és actualitzar l'estat i registrar el canvi si cal:

```
@app.route('/estado', methods=['POST'])
def actualizar_estado():
    datos = request.get_json()
    if datos and 'sensor_id' in datos and 'estado' in datos:
        sensor_id = datos['sensor_id']
        estado = datos['estado']
        estado_sensores[sensor_id] = estado
        guardar_log_si_corresponde(sensor_id, estado)
        return 'OK', 200
    return 'Datos incorrectos', 400
```

Aquesta funció fa dues coses importants:

- Actualitza l'estat a memòria amb diccionari estado_sensores.
- Crida guardar_log_si_corresponde() per escriure a la base de dades només si l'estat ha canviat realment.

2. GET /get_estado Retorna un JSON amb l'estat actual de tots els sensors:

```
@app.route('/get_estado')
def get_estado():
    return jsonify(estado_sensores)
```

Aquesta ruta és consultada cada segon pel navegador mitjançant JavaScript per mantenir el mapa actualitzat.

3. GET /logs Recupera els últims 100 registres de la base de dades per mostrar-los en taula HTML.

```
@app.route('/get_estado')
def get_estado():
    return jsonify(estado_sensores)

@app.route('/logs')
def obtener_logs():
    conn = sqlite3.connect('logs.db')
    c = conn.cursor()
    c.execute("SELECT timestamp, estado, sensor_id FROM logs ORDER BY id DESC LIMIT 100")
    data = c.fetchall()
    conn.close()
    return jsonify(data)
```

Aquesta informació permet als usuaris analitzar l'ús de les places amb dades històriques.

Persistència i optimització: guardar_log_si_corresponde()

Aquesta funció assegura que només es registren canvis reals, evitant duplicats innecessaris:

```python
# Guardar log solo si ha cambiado (sin esperar tiempo)
def guardar_log_si_corresponde(sensor_id, estado_actual):
    ahora = datetime.now()

    if sensor_id not in ultimo_log:
        ultimo_log[sensor_id] = ahora
        ultimo_estado_guardado[sensor_id] = ""

    ha_cambiado = (estado_actual != ultimo_estado_guardado[sensor_id])

    if ha_cambiado:
        conn = sqlite3.connect('logs.db')
        c = conn.cursor()
        c.execute("INSERT INTO logs (timestamp, estado, sensor_id) VALUES (?, ?, ?)",
                  (ahora.isoformat(), estado_actual, sensor_id))
        conn.commit()
```

Aquesta optimització evita generar registres redundants, redueix el tamany de la base de dades i fa que el sistema sigui més escalable.

Resum d'avantatges de Flask en aquest projecte

- **Simplicitat**: estructura de codi clara, ideal per a estudiants.
- **Potència**: pot gestionar múltiples sensors i escalar fàcilment.
- Integració nativa amb SQLite i JSON, que simplifica l'enviament i recepció de dades.
- Fàcil desplegament amb ngrok per fer proves externes.

El servidor Flask és, per tant, la peça central que fa que tot el sistema sigui interactiu, funcional i connectat. Sense ell, el projecte només seria un muntatge local sense capacitat de comunicar-se amb l'exterior.

3.3 OpenStreetMap + Leaflet

Per a la representació visual de les dades en un mapa interactiu, s'ha optat per l'ús conjunt de OpenStreetMap i la llibreria Leaflet.js.

Aquest conjunt és una alternativa gratuïta i de codi obert a solucions comercials com Google Maps, amb l'avantatge addicional de no requerir cap clau d'API ni pagar pel seu ús. Aquesta decisió afavoreix tant la viabilitat econòmica com l'autonomia tècnica del projecte.

3.3.1 Funcionament del mapa

El mapa es crea i configura mitjançant JavaScript al fitxer script.js. L'inicialitzador del mapa estableix el centre de la vista i el nivell de zoom:

```javascript
var map = L.map('map').setView([41.455436, 2.202046], 17);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
  attribution: '© OpenStreetMap contributors'
}).addTo(map);

const sensores = {
  sensor1: {
    coords: [41.455436, 2.202046],
    marker: null
  },
  sensor2: {
    coords: [41.456072, 2.200383],
    marker: null
  }
};
```

Això mostra un mapa centrat a la zona de Santa Coloma de Gramenet amb un nivell de detall ideal per visualitzar carrers.

3.3.2 Representació de sensors

Representació de sensors

Cada sensor del sistema es representa amb un cercle de petit radi que canvia de color segons l'estat:

```
// Crear marcadores
Object.keys(sensores).forEach(id => {
  sensores[id].marker = L.circle(sensores[id].coords, {
    radius: 5,
    color: 'green',
    fillColor: 'green',
    fillOpacity: 0.8,
    weight: 2
  }).addTo(map);
});
```

Aquest codi crea un marcador de color verd. Quan el sensor canvia a "ocupat", aquest color es transforma a vermell mitjançant la funció actualizarEstado():

```
        const color = estado === "ocupado" ? "red" : "green";
        if (sensores[id]) {
          sensores[id].marker.setStyle({ color: color, fillColor: color });
        }
```

Aquest canvi visual és immediat, proporcionant una experiència d'usuari clara i intuïtiva.

3.3.3 Sincronització amb el servidor

Cada segon, la funció actualizarEstado() fa una petició al servidor Flask per obtenir l'estat actual dels sensors:

```
function actualizarEstado() {
  fetch('/get_estado')
    .then(res => res.json())
    .then(data => {
      Object.entries(data).forEach(([id, estado]) => {
        const color = estado === "ocupado" ? "red" : "green";
        if (sensores[id]) {
          sensores[id].marker.setStyle({ color: color, fillColor: color });
        }
      });
```

Aquesta actualització constant assegura que l'usuari visualitzi l'estat en temps real sense haver de recarregar la pàgina.

3.4 Script.js

El fitxer script.js és fonamental per al funcionament del client web. Es tracta d'un arxiu JavaScript que dinamitza la pàgina i connecta amb el servidor Flask per mantenir la informació actualitzada en temps real.

3.4.1 Inicialització del mapa

La primera acció que fa el script és crear i configurar el mapa interactiu utilitzant la llibreria Leaflet:

```
  GNU nano 6.2
var map = L.map('map').setView([41.455436, 2.202046], 17);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
  attribution: '© OpenStreetMap contributors'
}).addTo(map);
```

Això centra el mapa a les coordenades posades, en aquest cas a l'Institut Puig Castellar, amb un nivell de zoom 17, ideal per veure amb claredat els carrers i les ubicacions de les places.

3.4.2 Creació de marcadors

Es defineix un objecte sensores amb les coordenades de cada sensor. Es genera un marcador per cada un:

```
const sensores = {
  sensor1: {
    coords: [41.455436, 2.202046],
    marker: null
  },
  sensor2: {
    coords: [41.456072, 2.200383],
    marker: null
  }
};

// Crear marcadores
Object.keys(sensores).forEach(id => {
  sensores[id].marker = L.circle(sensores[id].coords, {
    radius: 5,
    color: 'green',
    fillColor: 'green',
    fillOpacity: 0.8,
    weight: 2
  }).addTo(map);
});
```

Cada marcador representa una plaça d'aparcament. El color canvia depenent de si la plaça està lliure en verd o ocupada en vermell.

3.4.3 Actualització periòdica d'estat

La funció actualizarEstado() fa una crida AJAX al servidor per obtenir l'estat dels sensors. A partir de la resposta JSON, s'actualitzen els colors dels marcadors:

```
// Actualizar punteros del mapa
function actualizarEstado() {
  fetch('/get_estado')
    .then(res => res.json())
    .then(data => {
      Object.entries(data).forEach(([id, estado]) => {
        const color = estado === "ocupado" ? "red" : "green";
        if (sensores[id]) {
          sensores[id].marker.setStyle({ color: color, fillColor: color });
        }
      });
    })
    .catch(err => console.error("Error al obtener estado:", err));
}
```

La funció s'executa automàticament cada 1000 ms:

```
setInterval(actualizarEstado, 1000);
```

3.4.4 Gestió de seccions del menú

La web conté diverses seccions. Quan l'usuari clica una pestanya del menú, la funció mostrarSeccion(id) mostra la secció corresponent i oculta la resta:

```
function mostrarSeccion(id) {
  ['mapa', 'como-funciona', 'quienes-somos', 'logs'].forEach(sec => {
    document.getElementById(sec).style.display = 'none';
  });
  document.getElementById(id).style.display = 'block';

  if (id === 'logs') {
    actualizarLogs();
  }
}
```

Aquesta lògica fa que la web sigui interactiva i dinàmica sense necessitat de recarregar-la.

3.4.5 Logs en temps real

Quan l'usuari accedeix a la secció de logs, es mostra una taula amb els últims canvis d'estat dels sensors:

```
// Actualizar tabla de logs en tiempo real
function actualizarLogs() {
  fetch('/logs')
    .then(res => res.json())
    .then(data => {
      const tbody = document.querySelector("#tabla-logs tbody");
      tbody.innerHTML = '';
      data.forEach(log => {
        const fila = document.createElement('tr');
        fila.innerHTML = `
          <td>${new Date(log[0]).toLocaleString()}</td>
          <td>${log[2]}</td>
          <td>${log[1]}</td>
        `;
        tbody.appendChild(fila);
      });
    })
    .catch(err => console.error("Error al obtener logs:", err));
}
```

També hi ha un setInterval que refresca els logs si la secció està visible:

```
// Refrescar logs automáticamente cada segundo si está visible
setInterval(() => {
  if (document.getElementById('logs').style.display !== 'none') {
    actualizarLogs();
  }
}, 1000);
```

Aquesta funcionalitat permet veure l'històric en temps real i fa que la pàgina sigui molt informativa.

3.5 BD

3.5.1 Estructura

Es defineix una única taula logs:

```
CREATE TABLE IF NOT EXISTS logs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    timestamp TEXT,
    estado TEXT,
    sensor_id TEXT
)
```

Cada registre representa un canvi d'estat, incloent la data/hora, l'identificador del sensor i el nou estat.

3.5.2 Inserció condicional

Només es guarda un registre si l'estat del sensor ha canviat. Això s'implementa al servidor Flask i evita duplicació de dades.

3.6 Logs

Els logs són el registre històric de canvis d'estat de les places d'aparcament, i són essencials per analitzar patrons d'ocupació, hores punta, i comportament general de l'ús de les places.

Quan un sensor canvia d'estat, aquest canvi s'insereix a la base de dades amb la data i hora exacta, el sensor afectat i l'estat nou. Aquesta informació es pot consultar des de la interfície web, a la secció de Logs.

Cada entrada del log té el format: data i hora, sensor, ocupat o lliure. Mostra els resultats ordenats de més recent a més antic. Aquesta funcionalitat és especialment útil per fer un seguiment del comportament de les places i detectar patrons com hores punta o sensors defectuosos.

Els logs també serveixen per a validacions i comprovacions durant les proves del sistema, i poden ajudar a detectar errors o problemes de connexió en el funcionament general.

3.7 Página Web (html + css)

La pàgina web és la interfície gràfica amb què interactua l'usuari. Està composta per un fitxer HTML (index.html) i una fulla d'estils CSS (style.css).

3.7.1 HTML

Aquest fitxer estructura la pàgina i defineix les seccions:

- Mapa amb Leaflet.
- Explicació de com funciona.
- Qui som.
- Taula de logs.

Cada secció es mostra i oculta dinàmicament mitjançant JavaScript.

3.7.2 CSS

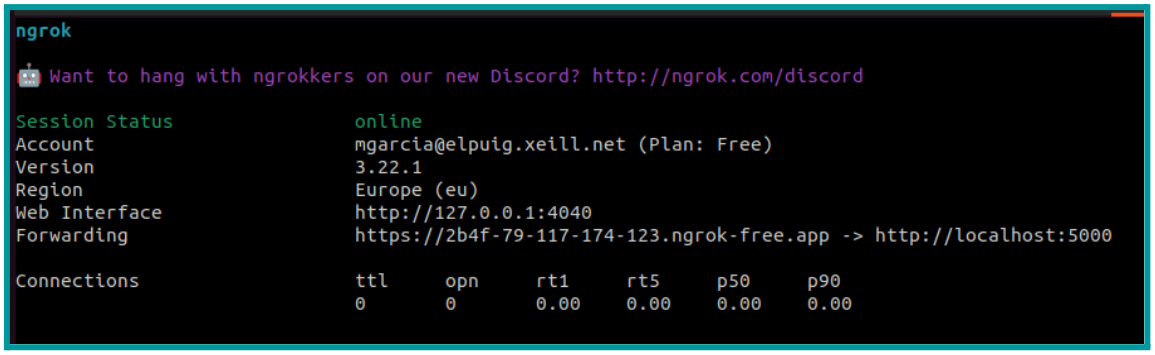
Defineix l'estètica de la pàgina:

- Colors corporatius: verd, blanc i turquesa.
- Layout responsiu mitjançant Flexbox i @media queries.
- Estils moderns: ombres, corbes arrodonides, tipografia clara.

3.8 Ngrok

Permet accedir a una aplicació Flask, que normalment només seria accessible des de la xarxa local, mitjançant una URL pública.

Per visualitzar la página web tindrem que posar la següent url, on posa Forwarding:



Aquesta URL redirigeixi automàticament cap al port local on està corrent l'aplicació, com ara localhost:5000.

3.8.1 Com s'utilitza al projecte

Quan el servidor Flask està actiu, només és accessible des del mateix ordinador o dins de la xarxa local. Amb Ngrok, es pot executar la comanda, ngrok http 5000

Això crea un túnel que permet que qualsevol dispositiu del món amb connexió a Internet pugui accedir a la pàgina web del projecte, incloent dispositius mòbils. Això ha estat fonamental per fer demostracions a professors o companys fora de l'aula, i per fer proves des del telèfon mòbil.

3.8.2 Avantatges de l'ús de Ngrok

- Configuració immediata, no requereix configuració de xarxa o router.
- URL pública segura i temporal.
- Possibilitat de veure trànsit i debugar les peticions entrants.
- Compatible amb qualsevol servei web, com Flask, Node.js, etc.

Ngrok, tot i ser una eina externa, ha estat una part important del procés de desenvolupament i validació del sistema, permetent convertir una aplicació local en una demo realment accessible des de qualsevol lloc.

4 Conclusions

4.1 Conclusions generals del projecte

Aquest projecte ens ha ajudat molt a entendre com funcionen els sensors i com podem connectar-los a Internet per veure la informació en temps real. Hem après a treballar amb Arduino, programar una web senzilla i fer que tot això funcioni junt. A més, ens ha servit per agafar experiència treballant en equip i planificant el projecte.

4.2 Consecució dels objectius

Objectiu 1: Crear una web funcional i intuïtiva per consultar l'ocupació d'un espai

Resultat: Assolit
Justificació: S'ha creat una web accessible des de qualsevol navegador, on es mostra si l'espai està lliure o ocupat. La interfície és clara, minimalista i adaptada a dispositius mòbils. No requereix registre ni inici de sessió, facilitant l'accés a qualsevol usuari.

Objectiu 2: Desenvolupar un sistema físic amb sensors (Arduino) que detecti si l'espai està ocupat

Resultat: Assolit
Justificació: S'ha utilitzat una placa Arduino amb sensors de proximitat, per detectar la presència d'un objecte o persona a la zona monitoritzada. Quan es detecta moviment o ocupació, l'Arduino envia l'estat al servidor mitjançant una connexió serial o via xarxa, segons la configuració.

Objectiu 3: Implementar un servidor amb Flask que rebi les dades i les posi a disposició de la web

Resultat: Assolit
Justificació: S'ha creat un servidor amb Python i Flask que rep les dades provinents de l'Arduino i les emmagatzema temporalment per poder ser consultades des de la web. El servidor proporciona una API senzilla que permet accedir al darrer estat i als registres anteriors.

Objectiu 4: Connectar la web amb el servidor mitjançant JavaScript (script.js) per mostrar l'estat en temps real

Resultat: Assolit
Justificació: Mitjançant un script de JavaScript, la web fa consultes periòdiques al servidor Flask per obtenir l'estat actual de l'espai. Aquest procés es fa de manera automàtica, sense necessitat de refrescar la pàgina. Això permet a l'usuari veure els canvis en temps real.

Objectiu 5: Integrar un mapa interactiu amb Leaflet i OpenStreetMap per mostrar la ubicació de l'espai

Resultat: Assolit
Justificació: S'ha integrat un mapa a la web utilitzant la llibreria Leaflet i dades de OpenStreetMap. El mapa mostra la ubicació exacta de l'espai controlat, i es poden afegir marcadors i informació contextual. Aquesta funció amplia la comprensió de l'usuari sobre on es troba l'espai monitoritzat.

Objectiu 6: Oferir un històric de registres i possibilitat de veure tendències (hores punta, etc.)

Resultat: Parcialment assolit
Justificació: Encara que s'emmagatzema les dades històriques dels sensors les funcionalitats, d'anàlisi y de visualització grafica no estan del tot desenvolupades. Ara es poden consultar els registres bàsics però les estadístiques detallades i les gràfiques es preveuen com una futura millora de la web.

Objectiu 7: Utilitzar Ngrok per exposar el servidor local a internet durant les proves

Resultat: Assolit
Justificació: Durant la fase de proves, s'ha fet ús de Ngrok per obrir el servidor Flask a l'exterior, permetent accedir-hi des de dispositius que no estaven a la mateixa xarxa local. Això ha estat útil per validar el funcionament remot del sistema i per fer demostracions.

4.3 Valoració de la metodologia i planificació

La forma en què vam organitzar el treball ha funcionat força bé, però de vegades hem hagut de canviar coses sobre la marxa perquè alguna cosa no anava o ens ha costat més del que esperàvem. Hem treballat bé junts i hem respectat els terminis, tot i que no sempre ha estat fàcil.

4.4 Problemes sorgits i solucions

Durant el desenvolupament del projecte, ens hem trobat amb diversos problemes que ens han fet aprendre molt.

Primer de tot, no sabíem gairebé res sobre Arduino, ja que mai abans havíem utilitzat cap placa. Això va fer que al principi ens costés molt entendre com funcionava i com programar-la correctament.

També, mentre anàvem modificant els codis, de vegades canviàvem alguna cosa sense voler i això feia que deixés de funcionar. Ens va costar bastant temps descobrir què havia passat i arreglar-ho per tornar a tenir tot bé.

Un altre problema va ser amb Ngrok. No sabíem com connectar-lo ni com funcionava per fer accessible el nostre servidor des de fora de la nostra xarxa local. Vam haver de demanar ajuda als professors i investigant una mica ens vam adonar de com funcionava ngrok.

A més, l'ordinador on treballàvem tenia molts recursos ocupats perquè teníem obert el programa d'Arduino i la màquina virtual al mateix temps. La màquina virtual sovint es penjava o deixava de respondre, així que vam haver de reiniciar-la moltes vegades, cosa que feia que avancéssim més lentament.

Malgrat aquests problemes, vam anar aprenent i buscant solucions pas a pas, i això ens ha servit molt per adquirir més experiència i saber com superar obstacles similars en el futur.

4.5 Visió de futur

Si tinguéssim molts més recursos i diners per desenvolupar el projecte a gran escala, el que faríem seria instal·lar tot el hardware necessari a cada aparcament de totes les places d'estacionament a les carrers de Santa Coloma de Gramenet. Això vol dir que a cada plaça hi hauria un sensor connectat a una placa Arduino amb WiFi, i aquests dispositius estarien instal·lats físicament al carrer, controlant en temps real si les places estan ocupades o no.

Tot aquest hardware enviaria la informació directament a un servidor central, que processaria les dades i les posaria a disposició dels ciutadans a través d'una aplicació web i mòbil. Així, qualsevol persona que vulgui aparcar podria saber des de qualsevol lloc l'estat real de les places i estalviar temps buscant.

Si ho vendéssim a un ajuntament, com el de Santa Coloma de Gramenet, el projecte hauria de ser molt més robust i escalable. Hauríem de garantir que la xarxa WiFi cobreixi totes les zones, o utilitzar altres tecnologies com xarxes LPWAN (LoRa, NB-IoT) per garantir la connexió de tots els sensors a llarg termini. També caldria crear un sistema de manteniment perquè els sensors i les plaques estiguin sempre funcionant i per resoldre avaries ràpidament.

Actualment, el que tenim és una simulació amb només dos sensors connectats a un mateix Arduino, que està a una aula i no al carrer. Estem fent tot el procés de lectura i enviament de dades des del nostre ordinador, no amb dispositius físics instal·lats a la via pública. Això ens ha permès provar la idea i comprovar que funciona, però per portar-ho a la realitat caldria fer aquesta instal·lació massiva i crear una infraestructura de suport que pugui gestionar molts sensors alhora i que sigui fiable.

5. Glossari

Arduino: Placa electrònica que permet controlar sensors i actuadors.

Sensor ultrasònic: Dispositiu que mesura distàncies fent servir ones de so.

LED: Diodo emissor de llum que indica visualment l'estat del sensor.

Flask: Framework lleuger de Python per crear aplicacions web.

Ngrok: Eina que permet accedir a serveis locals des d'Internet mitjançant túnels segurs.

WebSocket: Protocol que permet comunicació en temps real entre client i servidor.

Leaflet: Llibreria JavaScript per mostrar mapes interactius al web.

SQLite: Base de dades lleugera utilitzada per emmagatzemar registres localment.

sensor id: Identificador únic de cada sensor per distingir-ne l'origen.

Port sèrie: Mitjà de comunicació entre l'Arduino i l'ordinador.

JSON: Format de dades lleuger usat per enviar informació entre l'Arduino i el web.

6. Bibliografia

Arduino. Sensor ultrasónico HC-SR04. URL: https://docs.arduino.cc/built-in-examples/sensors/Ping.

Arduino. Documentación oficial de Arduino Uno. URL: https://docs.arduino.cc/hardware/uno-rev3.

Flask. Welcome to Flask. URL: https://flask.palletsprojects.com/en/stable/

Flask-SocketIO. Flask-SocketIO Documentation. URL: https://flask-socketio.readthedocs.io/en/latest/.

Leaflet. Leaflet - an open-source JavaScript library for interactive maps. URL: https://leafletjs.com/.

Ngrok. ngrok - Secure introspectable tunnels to localhost. URL: https://ngrok.com/.

SQLite. SQLite Home Page. URL: https://www.sqlite.org/

Bootstrap. Bootstrap 5 Documentation. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/.

OpenStreetMap. OpenStreetMap Wiki. URL: https://wiki.openstreetmap.org/wiki/Main_Page.

Python. Python Official Documentation. URL: https://docs.python.org/3/.

7 Annexos

7.1 Códigos para el funcionamiento de todo el proyecto

7.1.1 Arduino

```
#include <WiFiS3.h>
#include <WiFiClient.h>


const char ssid[] = "SuperXEiLL";
const char server[] = "192.168.251.204";
const int port = 5000;
WiFiClient client;


// Sensor 1
const int trig1 = 2;
const int echo1 = 3;
const int ledR1 = 8;
const int ledV1 = 9;


// Sensor 2
const int trig2 = 5;
const int echo2 = 10;
const int ledR2 = 12;
const int ledV2 = 13;


String estado1 = "";
String estado2 = "";


void setup() {
Serial.begin(9600);
WiFi.begin(ssid);
while (WiFi.status() != WL_CONNECTED) {
  Serial.println("Conectando a WiFi...");
  delay(1000);
}
Serial.println("WiFi conectado");
```

```
// Sensor 1
pinMode(trig1, OUTPUT);
pinMode(echo1, INPUT);
pinMode(ledR1, OUTPUT);
pinMode(ledV1, OUTPUT);


// Sensor 2
pinMode(trig2, OUTPUT);
pinMode(echo2, INPUT);
pinMode(ledR2, OUTPUT);
pinMode(ledV2, OUTPUT);
}


long medirDistancia(int trig, int echo) {
digitalWrite(trig, LOW);
delayMicroseconds(2);
digitalWrite(trig, HIGH);
delayMicroseconds(10);
digitalWrite(trig, LOW);
long duracion = pulseIn(echo, HIGH, 30000);
long distancia = duracion * 0.034 / 2;
return (distancia > 0 && distancia < 300) ? distancia : 999;
}


void loop() {
// Sensor 1
long d1 = medirDistancia(trig1, echo1);
String nuevoEstado1 = d1 <= 12 ? "ocupado" : "libre";
if (nuevoEstado1 != estado1) {
  estado1 = nuevoEstado1;
  enviarEstado("sensor1", estado1);
}
digitalWrite(ledR1, estado1 == "ocupado" ? HIGH : LOW);
digitalWrite(ledV1, estado1 == "libre" ? HIGH : LOW);


delay(100); // separación entre sensores
```

```
// Sensor 2
long d2 = medirDistancia(trig2, echo2);
String nuevoEstado2 = d2 <= 12 ? "ocupado" : "libre";
if (nuevoEstado2 != estado2) {
  estado2 = nuevoEstado2;
  enviarEstado("sensor2", estado2);
}


//  Invertimos el comportamiento porque los LEDs están cruzados
digitalWrite(ledR2, estado2 == "libre" ? HIGH : LOW);
digitalWrite(ledV2, estado2 == "ocupado" ? HIGH : LOW);


delay(200);
}


void enviarEstado(String id, String estado) {
String payload = "{\"sensor_id\":\"" + id + "\",\"estado\":\"" + estado + "\"}";


if (client.connect(server, port)) {
  client.println("POST /estado HTTP/1.1");
  client.println("Host: " + String(server));
  client.println("Content-Type: application/json");
  client.print("Content-Length: ");
  client.println(payload.length());
  client.println();
  client.println(payload);
  client.stop();
  Serial.println("Enviado: " + payload);
} else {
  Serial.println("Error al conectar con servidor");
}
}
```

7.1.2 Index.html

```
DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<title>ParqueoYA</title>
<link rel="stylesheet" href="/static/style.css">
<link rel="stylesheet" href="https://unpkg.com/leaflet/dist/leaflet.css" />
</head>
<body>
<header>
<h1>ParqueoYA</h1>
<nav>
<ul>
<li><a href="#" onclick="mostrarSeccion('mapa')">Inicio</a></li>
<li><a href="#" onclick="mostrarSeccion('como-funciona')">Cómo funciona</a></li>
<li><a href="#" onclick="mostrarSeccion('quienes-somos')">Quiénes somos</a></li>
<li><a href="#" onclick="mostrarSeccion('logs')">Logs</a></li>
</ul>
</nav>
</header>

<section id="mapa">
<div id="map" style="width: 100%; height: 100%;"></div>
</section>

<section id="como-funciona" style="display:none;">
<div style="border: 2px solid #00796B; border-radius: 12px; padding: 20px;
background-color: #f9f9f9; max-width: 800px; margin: 0 auto; text-align: center;">
<h2 style="color: #00796B; font-size: 2em; margin-bottom: 1em;">¿Cómo
funciona?</h2>

<p style="font-size: 1.1em; line-height: 1.8;">
<strong>ParqueoYa</strong> utiliza sensores ultrasónicos conectados a un sistema
Arduino.
</p>

<p style="font-size: 1.1em; line-height: 1.8;">
Los datos se envían automáticamente a un servidor web que actualiza un mapa a
tiempo real.
</p>

<p style="font-size: 1.1em; line-height: 1.8;">
Además, se almacena un historial de eventos en la base de datos, permitiendo
visualizar el estado de las plazas.
```

```
</p>

<p style="font-size: 1.1em; line-height: 1.8;">
Gracias a esta tecnología, ParqueoYa ayuda a los conductores a encontrar
aparcamiento con mayor facilidad.
</p>
<img src="/static/parqueoya.png" alt="parqueoya" width="200" />
</div>
</section>

<section id="quienes-somos" style="display:none;">
<h2 style="text-align:center; font-size: 2em; margin-top: 1em;">QUIÉNES
SOMOS</h2>
<p style="text-align:center; max-width: 600px; margin: 0 auto 2em; font-size: 1.1em;">
Somos Marc y Dani, dos estudiantes de 20 años del ciclo formativo de grado superior.
Nuestra idea nació con un objetivo claro: aplicar la tecnología para resolver problemas
cotidianos.
</p>

<div class="equipo-container">
<div class="miembro">
<img src="/static/IMG_6826.jpg" alt="Marc García" />
<h3>Marc García</h3>
<p>Soy Marc tengo 20 años, soy de Palau-Solità i Plegamans y trabajo en una
empresa como helpdesk.</p>
</div>
<div class="miembro">
<img src="/static/dani.png" alt="Daniel Jiménez" />
<h3>Daniel Jiménez</h3>
<p>Soy Dani tengo 21 años y soy de Santa Coloma de Gramanet</p>
</div>
</div>
</section>

<section id="logs" style="display:none;">
<h2>Registro de Logs</h2>
<table border="1" id="tabla-logs">
<thead>
<tr><th>Fecha y Hora</th><th>Sensor</th><th>Estado</th></tr>
</thead>
<tbody></tbody>
</table>
</section>

<script src="https://unpkg.com/leaflet/dist/leaflet.js"></script>
<script src="/static/script.js"></script>
</body>
</html>
```

7.1.3 Init_db

```
import sqlite3

conn = sqlite3.connect('logs.db')
c = conn.cursor()
c.execute('''
CREATE TABLE IF NOT EXISTS logs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    timestamp TEXT,
    estado TEXT,
    sensor_id TEXT
)
''')
conn.commit()
conn.close()
print("Base de datos creada correctamente")
```

7.1.4 script.js

```
var map = L.map('map').setView([41.455436, 2.202046], 17);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
  attribution: '© OpenStreetMap contributors'
}).addTo(map);

const sensores = {
  sensor1: {
    coords: [41.455436, 2.202046],
    marker: null
  },
  sensor2: {
    coords: [41.456072, 2.200383],
    marker: null
  }
};

// Crear marcadores
Object.keys(sensores).forEach(id => {
  sensores[id].marker = L.circle(sensores[id].coords, {
    radius: 5,
    color: 'green',
    fillColor: 'green',
    fillOpacity: 0.8,
    weight: 2
```

```
  }).addTo(map);
});

// Actualizar punteros del mapa
function actualizarEstado() {
  fetch('/get_estado')
    .then(res => res.json())
    .then(data => {
      Object.entries(data).forEach(([id, estado]) => {
        const color = estado === "ocupado" ? "red" : "green";
        if (sensores[id]) {
          sensores[id].marker.setStyle({ color: color, fillColor: color });
        }
      });
    })
    .catch(err => console.error("Error al obtener estado:", err));
}

setInterval(actualizarEstado, 1000);
actualizarEstado();

function mostrarSeccion(id) {
  ['mapa', 'como-funciona', 'quienes-somos', 'logs'].forEach(sec => {
    document.getElementById(sec).style.display = 'none';
  });
  document.getElementById(id).style.display = 'block';

  if (id === 'logs') {
    actualizarLogs();
  }
}

// Actualizar tabla de logs en tiempo real
function actualizarLogs() {
  fetch('/logs')
    .then(res => res.json())
    .then(data => {
      const tbody = document.querySelector("#tabla-logs tbody");
      tbody.innerHTML = '';
      data.forEach(log => {
        const fila = document.createElement('tr');
        fila.innerHTML = `
          <td>${new Date(log[0]).toLocaleString()}</td>
          <td>${log[2]}</td>
          <td>${log[1]}</td>
        `;
        tbody.appendChild(fila);
      });
```

```
    })
    .catch(err => console.error("Error al obtener logs:", err));
}

// Refrescar logs automáticamente cada segundo si está visible
setInterval(() => {
  if (document.getElementById('logs').style.display !== 'none') {
    actualizarLogs();
  }
}, 1000);
```

7.1.5 server.py

```
from flask import Flask, request, render_template, jsonify
from flask_cors import CORS
from datetime import datetime
import sqlite3

app = Flask(__name__)
CORS(app)

estado_sensores = {
    "sensor1": "libre",
    "sensor2": "libre"
}

ultimo_log = {}
ultimo_estado_guardado = {}

# Crear base de datos si no existe
def init_db():
    conn = sqlite3.connect('logs.db')
    c = conn.cursor()
    c.execute('''
        CREATE TABLE IF NOT EXISTS logs (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            timestamp TEXT,
            estado TEXT,
            sensor_id TEXT
        )
    ''')
    conn.commit()
    conn.close()

init_db()

# Guardar log solo si ha cambiado (sin esperar tiempo)
def guardar_log_si_corresponde(sensor_id, estado_actual):
```

```
    ahora = datetime.now()

    if sensor_id not in ultimo_log:
        ultimo_log[sensor_id] = ahora
        ultimo_estado_guardado[sensor_id] = ""

    ha_cambiado = (estado_actual != ultimo_estado_guardado[sensor_id])

    if ha_cambiado:
        conn = sqlite3.connect('logs.db')

      c = conn.cursor()
        c.execute("INSERT INTO logs (timestamp, estado, sensor_id) VALUES (?, ?, ?)",
                  (ahora.isoformat(), estado_actual, sensor_id))
        conn.commit()
        conn.close()

        ultimo_log[sensor_id] = ahora
        ultimo_estado_guardado[sensor_id] = estado_actual
        print(f"[LOG] {sensor_id}: {estado_actual} a las {ahora.strftime('%H:%M:%S')}")
    else:
        print(f"[INFO] {sensor_id}: {estado_actual} (sin guardar)")

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/estado', methods=['POST'])
def actualizar_estado():
    datos = request.get_json()
    if datos and 'sensor_id' in datos and 'estado' in datos:
        sensor_id = datos['sensor_id']
        estado = datos['estado']
        estado_sensores[sensor_id] = estado
        guardar_log_si_corresponde(sensor_id, estado)
        return 'OK', 200
    return 'Datos incorrectos', 400

@app.route('/get_estado')
def get_estado():
    return jsonify(estado_sensores)

@app.route('/logs')
def obtener_logs():
    conn = sqlite3.connect('logs.db')
    c = conn.cursor()
    c.execute("SELECT timestamp, estado, sensor_id FROM logs ORDER BY id DESC
LIMIT 100")
```

```
    data = c.fetchall()
    conn.close()
    return jsonify(data)

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

7.1.6 style.css

```
html, body {
    margin: 0;
    padding: 0;
    height: 100%;
    font-family: Arial, sans-serif;
}

header {
    background-color: #00796B;
    color: white;
    padding: 10px 20px;
    height: 60px;
    display: flex;
    align-items: center;
    justify-content: center; /* centrado real */
    gap: 60px; /* separación equilibrada entre título y menú */
}

header h1 {
    margin: 0;
    font-size: 1.5em;
}

nav ul {
    display: flex;
    gap: 30px;
    list-style-type: none;
    padding: 0;
    margin: 0;
    align-items: center;
}

nav li {
    display: inline;
}

nav a {
    color: white;
    text-decoration: none;
```

```
    font-weight: bold;
}

section {
    display: none;
    height: calc(100vh - 60px);
    overflow-y: auto;
    padding: 20px;
    box-sizing: border-box;
}

#mapa {
    display: block;
    height: calc(100vh - 60px);
    padding: 0;
    margin: 0;
}

#map {
    height: 100%;
    width: 100%;
}

table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 10px;
}

th, td {
    padding: 8px;
    text-align: left;
}

/* Estilos para la sección "Quiénes somos" */
.equipo-container {
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    gap: 2rem;
    padding: 2rem 1rem;
    box-sizing: border-box;
}

.miembro {
    background-color: #f1f1f1;
    border-radius: 16px;
    padding: 1.5rem;
```

```
    text-align: center;
    max-width: 280px;
    box-shadow: 0 4px 10px rgba(0,0,0,0.1);
    transition: transform 0.3s ease;
    flex: 1 1 220px;
}

.miembro:hover {
    transform: translateY(-5px);
}

.miembro img {
    width: 140px;
    height: 140px;
    object-fit: cover;
    border-radius: 50%;
    margin-bottom: 1rem;
    border: 3px solid #00796B;
}

.miembro h3 {
    margin: 0.5rem 0 0.3rem;
    font-size: 1.2rem;
    color: #00796B;
}

.miembro p {
    font-size: 0.95rem;
    color: #333;
}

/* Responsive para pantallas pequeñas */
@media (max-width: 768px) {
    header {
        flex-direction: column;
        height: auto;
        gap: 10px;
    }

    .equipo-container {
        flex-direction: column;
        align-items: center;
    }
}
```