Université de Sfax

École nationale de l'électronique et des télécommunications de Sfax

Département des Systèmes électroniques et Communication

Nom du Projet

Smart Parking

Élaboré par :

Maghrebi Mohamed Ali

Abdallah Maryem

Classe & Groupe :

3 GEC SE – Groupe 2

Superviseur :

M. Karim Jaber

Année universitaire : 2023 - 2024

Sommaire

Sommaire

Table des figures

Introduction générale

La recherche incessante de places de stationnement, source de frustration, de gaspillage de temps et d'embouteillages, pose un défi quotidien omniprésent. C'est dans ce contexte que le smart parking émerge comme une solution avant-gardiste, révolutionnant la gestion des espaces dédiés au stationnement par une intégration technologique judicieuse. Cette approche astucieuse permet au smart parking de fournir en temps réel des informations cruciales sur la disponibilité des places, offrant une guidance précise vers les espaces libres et optimisant l'utilisation globale des zones de stationnement. Cette transformation technologique engendre une panoplie d'avantages, de l'amélioration significative de la circulation urbaine à la réduction notable des émissions de gaz, offrant ainsi une expérience de stationnement plus pratique pour les utilisateurs.

Ce rapport se structure en trois chapitres, chacun explorant des facettes spécifiques du smart parking. Le premier chapitre plonge en détail dans les composants matériels, examinant les systèmes de capteurs et l'utilisation avancée d'Arduino. Cette section offre une compréhension approfondie des bases matérielles essentielles à une mise en œuvre efficace du smart parking. Le deuxième chapitre se focalise sur les logiciels, mettant en évidence leur rôle crucial dans la gestion et l'optimisation des espaces de stationnement. Cette exploration souligne l'importance des solutions logicielles pour maximiser l'efficacité du smart parking. Enfin, le troisième chapitre aborde le développement du code, mettant en avant la programmation des capteurs, l'intégration avec Arduino, Blynk et ThingSpeak, et illustrant des exemples concrets pour une mise en œuvre réussie du smart parking.

Chaque section contribue de manière significative à une compréhension globale de cette innovation, unissant les composants matériels, logiciels et pratiques pour façonner l'avenir de la gestion des espaces de stationnement urbains.

Chapitre I :
Anatomie des Dispositifs Matériels

I.1. Introduction

Dans ce premier chapitre, nous plongeons dans l'anatomie du smart parking en explorant les composants matériels qui façonnent son infrastructure. De la carte NodeMCU ESP32 aux capteurs spécialisés, chaque élément est examiné en détail. Ce chapitre jettera les bases de notre compréhension du système, en mettant l'accent sur l'assemblage concret de ces composants pour créer le socle du smart parking.

I.2. Équipements Essentiels : Capteurs et Actionneurs.

Dans cette section, nous présentons les divers capteurs et actionneurs qui peuvent être utilisés dans le système de Smart Parking afin d'assurer des fonctionnalités étendues.

I.2.1. WiFi ESP32 ESP-WROOM-32

La carte de développement WiFi ESP32 ESP-WROOM-32 est une plateforme électronique avancée, conçue pour répondre aux exigences complexes du projet Smart Parking. Intégrant le module ESP32, cette carte offre une connectivité sans fil robuste, avec des fonctionnalités WiFi et Bluetooth, permettant une communication efficace au sein du réseau IoT dédié au stationnement intelligent. Grâce à ses capacités de traitement puissantes, cette carte constitue le socle technologique idéal pour la création d'une infrastructure intelligente de gestion des places de stationnement. Elle offre une flexibilité inégalée pour le développement d'applications embarquées hautement personnalisées, garantissant ainsi des performances optimales dans un environnement exigeant tel que celui du Smart Parking.

La figure suivante représente la carte dédiée.

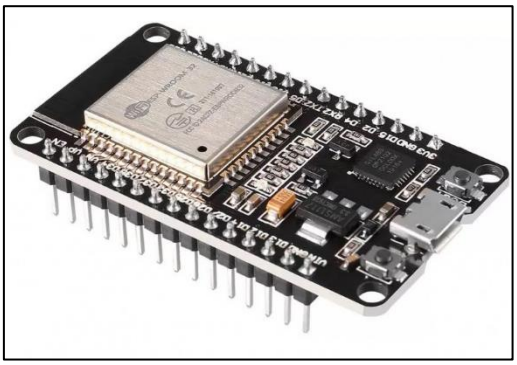

Figure 1 : Carte WiFi ESP32 ESP-WROOM-32

I.2.2. Module Relais HLS8L-DC3V-S-C

Le module relais HLS8L-DC3V-S-C, illustré dans la figure ci-dessus, agit comme un interrupteur normalement ouvert. Compatible avec Grove, il permet de commuter des charges

plus élevées que les cartes Esp32 standard, offrant ainsi une flexibilité accrue dans le contrôle des dispositifs électriques.

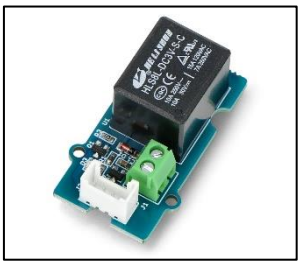

Figure 2 : Module Relais HLS8L-DC3V-S-C

I.2.3. Module Capteur Ultrason HC-SR-04

Le module capteur ultrason HC-SR-04, intégré à la figure ci-dessus, est dédié à la détection et à la mesure précise des distances entre divers objets, indépendamment de leur forme (liquide, solide, granuleux, etc.).

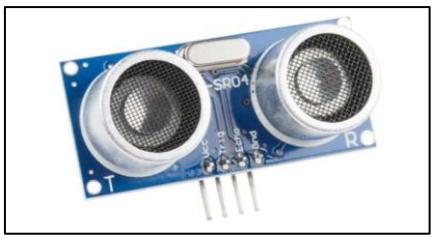

Figure 3 : Module Capteur Ultrason HC-SR-04

I.2.4. Module Capteur de Lumière

Le module capteur de lumière, clairement identifié dans la figure, intègre une photorésistance pour détecter l'intensité lumineuse, permettant ainsi une gestion intelligente de l'éclairage dans le parking.

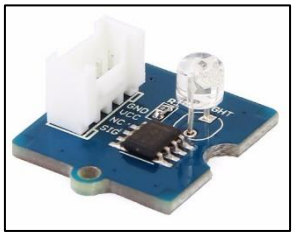

Figure 4 : Module Capteur de Lumière

I.2.5. Module Capteur Infrarouge

Le capteur infrarouge, discret dans la figure, détecte les signaux infrarouges pour localiser les véhicules, améliorant ainsi la précision de la détection des emplacements de stationnement dans le système Smart Parking.

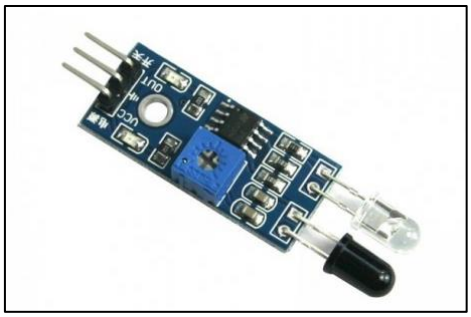

Figure 5 : Module Capteur Infrarouge

I.2.6. Module Capteur de Gaz MQ5

Le capteur de gaz MQ5, tel qu'intégré dans la configuration, joue un rôle essentiel dans le Smart Parking en détectant divers gaz, notamment le gaz de pétrole liquide (LPG), le propane, le méthane, le butane, et d'autres gaz naturels. Son utilité réside dans la surveillance proactive des niveaux de gaz potentiellement dangereux, contribuant ainsi à la sécurité globale des espaces de stationnement.

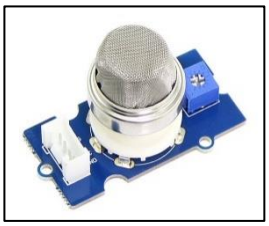

Figure 6 : Module Capteur de Gaz MQ5

I.2.7. Servomoteur

Le servomoteur, essentiel au système, assure le contrôle mécanique précis de la barrière de stationnement, illustré dans la figure ci-dessous. Équipé d'un codeur interne, il permet une gestion efficace des accès aux places de stationnement.

Figure 7 : Servomoteur

I.2.8. Récepteur infrarouge

Le récepteur infrarouge, intégré au système, facilite la commande à distance de la barrière de parking, comme indiqué dans la figure ci-dessous. En recevant les signaux infrarouges, il offre

une fonctionnalité de commande à distance, optimisant ainsi l'expérience des utilisateurs du système Smart Parking.

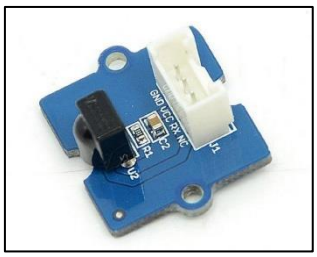

Figure 8 : Récepteur infrarouge

I.2.9. Afficheur OLED

Afficheur OLED 128x64 pixels, basé sur le circuit SH1106. Se connecte au microcontrôleur via le bus I2C, idéal pour une communication efficace avec des dispositifs Arduino ou compatibles, comme illustré dans la figure ci-dessous.

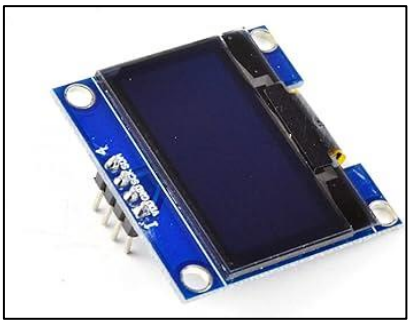

Figure 9 : Afficheur OLED

I.2.10. Accessoires Essentiels

En plus des composants susmentionnés, d'autres éléments essentiels tels que la plaque d'essai, les résistances 220 Ohm, les LEDs, la télécommande et le module d'alimentation sont également inclus dans la configuration, comme démontré dans la figure ci-dessus. Ces composants contribuent à la fonctionnalité et à l'esthétique globales du système Smart Parking.

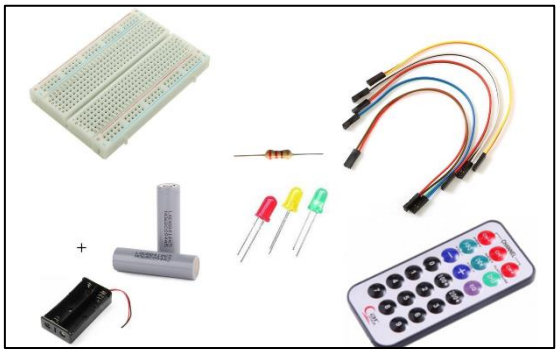

Figure 10 : Accessoires Essentiels

Dans la conception de ce prototype pour le système de parking intelligent, notre objectif principal est d'intégrer divers composants pour optimiser la gestion des espaces de stationnement. Nous prévoyons d'utiliser un capteur ultrason pour calculer la distance, permettant ainsi un accès facile une fois que la voiture atteint une distance préétablie. De plus, un capteur infrarouge sera déployé pour détecter la présence de véhicules. Pour offrir une visualisation claire des détections de voitures et fournir des informations sur le nombre de véhicules stationnés, nous incorporerons un afficheur OLED. En complément, les LEDs seront utilisées comme actionneurs, fournissant une indication visuelle de l'état de chaque capteur. Cette approche, alliant précision, efficacité et convivialité, vise à offrir une expérience de stationnement optimale.

I.3. Schéma Global du Câblage

Dans cette section, nous présentons le schéma complet du câblage, une représentation visuelle cruciale illustrant l'interconnexion harmonieuse des composants clés du smart parking. Le schéma offre une vue d'ensemble claire, démontrant la manière dont la carte NodeMCU ESP32, les capteurs spécialisés tels que le capteur ultrason et infrarouge, les actionneurs représentés par les LEDs, ainsi que d'autres équipements essentiels comme l'afficheur pour la visualisation et la batterie pour l'alimentation, s'entrelacent pour former un réseau fonctionnel. Chaque liaison et connexion sont méticuleusement documentées, fournissant une référence visuelle essentielle pour la phase de développement et d'implémentation pratique du projet. La figure ci-dessous illustre le schéma hardware complet du système.

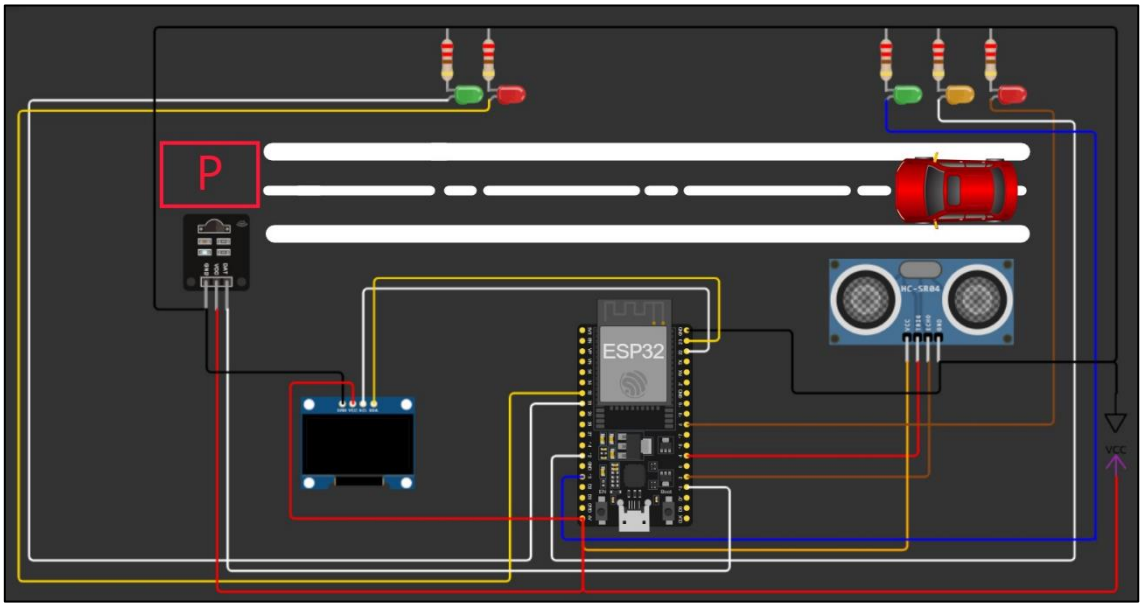

Figure 11 : Schéma du câblage

I.4. Conclusion

En conclusion de ce chapitre, nous avons exploré les composants matériels essentiels du smart parking, du NodeMCU ESP32 aux capteurs spécialisés. Le schéma global du câblage a souligné l'importance de leur interconnexion. Cette compréhension solide jette les bases pour le développement pratique à venir. Nous passerons du concept à la réalité dans le prochain chapitre dédié au développement logiciel.

Chapitre II :
Programmation et Intégration Logicielle

II.1. Introduction

Dans ce chapitre, nous explorons l'essence même du smart parking à travers sa programmation et son intégration logicielle. De la genèse du code avec Arduino à l'interface conviviale de Blynk, en passant par la consolidation des données sur ThingSpeak, découvrez comment chaque ligne de code contribue à rendre notre système de gestion de stationnement intelligent et fonctionnel.

II.2. Mise en Œuvre Logicielle

Cette section plonge dans la mise en œuvre logicielle, explorant la programmation avec Arduino et les interfaces de visualisation pour créer une solution parfaitement intégrée.

II.2.1. Arduino

Dans cette section cruciale, nous définissons tout d'abord Arduino : le logiciel, symbolisé par son logo illustré dans la figure ci-dessous, constitue la base programmable du Smart Parking. Arduino, programmé en langage C++, orchestre la carte NodeMCU ESP32, jouant un rôle central dans la gestion des capteurs et des actions du parking intelligent.

Figure 12 : logo ARDUINO

II.2.2. Blynk

Dans cette section, nous présentons Blynk, une plateforme logicielle polyvalente accessible à la fois sur mobile et sur le web, identifiable par son logo ci-dessous. Blynk offre une interface intuitive permettant de gérer les données générées par le Smart Parking de manière pratique et efficace.

Figure 13 : Logo Blynk

L'interaction Arduino et Blynk est une intégration cohérente des données se concrétise par

l'utilisation de l'API Blynk. Arduino communique avec Blynk en envoyant des données à l'aide de cette interface de programmation d'application. Pour établir cette connexion, Arduino doit être configuré avec une clé d'authentification fournie par Blynk lors de la création du projet sur la plateforme. Cette clé d'authentification garantit une liaison sécurisée entre Arduino et Blynk, permettant ainsi une intégration sans heurts des données du Smart Parking dans l'interface Blynk.

Les pins virtuels sur Arduino jouent un rôle clé dans cette interaction. Blynk propose des widgets spécifiques, tels que les boutons et les indicateurs, qui sont assignés à des pin virtuels. Arduino, de son côté, est programmé pour lire ou écrire sur ces pins virtuels, assurant ainsi une communication bidirectionnelle fluide entre le matériel du Smart Parking et l'interface Blynk.

Cette approche offre une flexibilité exceptionnelle pour personnaliser l'interface utilisateur sur Blynk, permettant aux utilisateurs de surveiller et de contrôler le Smart Parking de manière précise et efficace, que ce soit sur un appareil mobile ou via le web.

II.2.3. ThingSpeak

ThingSpeak, symbolisé par son logo ci-dessous, se distingue en tant que plateforme cloud IoT dédiée à la collecte, à la visualisation et à l'analyse en temps réel des données. Il propose une solution complète pour stocker et interpréter les informations générées par le Smart Parking, organisées selon un modèle de canal où chaque canal est attribué à un ensemble spécifique de données.

Figure 14 : Logo ThingSpeak

La structure de ThingSpeak repose sur des "fields", des emplacements spécifiques dans le canal associé à des types de données particuliers. Chaque field peut représenter une mesure spécifique, telle que la distance mesurée par le capteur ultrason ou la détection de présence par le capteur infrarouge. Cette approche facilite la gestion systématique des données, renforçant l'analyse et la visualisation sur la plateforme ThingSpeak.

Dans le cadre de l'interaction entre Arduino et ThingSpeak, Arduino initie une requête HTTP POST pour transmettre les données à un canal désigné. Des éléments cruciaux dans cette

communication incluent l'URL de l'API ThingSpeak, l'apiKey unique au canal, et le numéro du canal (channelID). Cette identification précise du canal garantit le cheminement correct des données, tandis que l'apiKey assure une connexion sécurisée entre Arduino et ThingSpeak. Grâce à sa souplesse et à son potentiel analytique, ThingSpeak se positionne comme une solution robuste pour la gestion des données du Smart Parking.

II.3. Programmation Avancée

Nous allons subdiviser l'exploration du script en deux volets distincts, chacun étroitement lié à une plateforme spécifique.

Avant de nous plonger dans les spécificités de chaque script, examinons les éléments partagés entre Blynk et ThingSpeak. Ces éléments comprennent la configuration des broches, la gestion du capteur ultrason, la détection de voiture, l'affichage sur l'écran OLED, et le contrôle des LEDs. Ces aspects communs assurent une base solide pour la communication bidirectionnelle entre le Smart Parking et les plateformes Blynk et ThingSpeak.

II.3.1. Script Blynk

Passons maintenant à la partie dédiée à Blynk. Ce script est minutieusement élaboré pour garantir une intégration harmonieuse avec la plateforme Blynk. Les spécifications du code sur Blynk comprennent la configuration des paramètres tels que l'authentification, le réseau WiFi, et la configuration des widgets virtuels pour l'affichage des données sur l'application Blynk. Le code Blynk utilise un timer pour gérer les mises à jour régulières des widgets et assure une synchronisation fluide entre le matériel et l'interface utilisateur. Pour découvrir en détail le script Blynk, veuillez-vous référer à l'Annexe 1.

II.3.2. Script ThingSpeak

Ensuite, explorons le script dédié à ThingSpeak. Conçu pour l'envoi régulier des données du Smart Parking vers ThingSpeak, le script utilise la bibliothèque HTTPClient.h pour effectuer des requêtes HTTP POST vers la plateforme ThingSpeak. Les paramètres essentiels incluent l'apiKey et le channelID, nécessaires pour identifier le canal spécifique sur ThingSpeak où les données doivent être envoyées. Pour une vue complète du script ThingSpeak, veuillez consulter l'Annexe 2.

II.4. Conclusion

Ce chapitre a tracé le chemin de la genèse à la réalité du Smart Parking, mettant en lumière l'importance cruciale de la programmation et de l'intégration logicielle. L'interfaçage permettra une série de tests visant à valider pleinement notre projet dans la prochaine étape.

Chapitre III :
Validation et Performance des Interfaces

III.1. Introduction

Ce chapitre représente la phase d'évaluation et de validation opérationnelle de notre système de gestion de stationnement intelligent. De l'interface série d'Arduino à la visualisation en temps réel sur Blynk et à la confirmation des données sur ThingSpeak, nous explorons les tests approfondis effectués pour assurer le fonctionnement optimal de notre solution innovante.

III.2. Interface série

Dans le cadre de l'évaluation instantanée de notre système, l'Interface Série (Serial Monitor) émerge comme un outil essentiel. En se connectant à la carte ESP32 via Arduino, le Serial Monitor offre une fenêtre transparente sur le fonctionnement interne du Smart Parking. En temps réel, cette interface permet de vérifier la connexion Wi-Fi de la carte ESP32, confirmant ainsi son intégration harmonieuse au réseau. De plus, elle offre une visibilité immédiate sur la connexion avec les plateformes Blynk et ThingSpeak, assurant une communication fluide.

En scrutant le Serial Monitor, les informations cruciales générées par le capteur ultrason deviennent rapidement accessibles. La distance mesurée, la détection de la présence d'une voiture, ainsi que le nombre actuel de voitures stationnées sont affichés en temps réel. Cette fonctionnalité permet une évaluation instantanée des performances du système, fournissant des données en direct qui sont essentielles pour identifier tout dysfonctionnement potentiel et assurer une réactivité optimale du Smart Parking.

III.3. Contrôle Visuel via Blynk

Blynk offre une interface polyvalente et réactive, accessible à la fois sur les versions mobile et web. L'intégration avec Blynk s'étend sur plusieurs fonctionnalités cruciales. Trois LED, représentant les états de la voiture (rouge, jaune et vert), agissent comme des actionneurs. Lorsque la distance mesurée par le capteur ultrason est inférieure à 10, signalant la présence d'une voiture, la LED passe du rouge au vert avec une transition clignotante en jaune.

L'interface affiche également la distance en temps réel, fournissant une surveillance constante des conditions du parking. Une icône distincte indique la présence de la voiture, avec une valeur binaire (1 ou 0) signalant si une voiture est détectée par le capteur infrarouge.

Pour illustrer ces fonctionnalités, des simulations des interfaces web et mobile sont présentées dans les figures ci-dessous. Ces représentations visuelles offrent une vision claire de

l'interaction fluide entre le Smart Parking et l'interface Blynk, garantissant une surveillance précise et une gestion intuitive depuis divers appareils.

Figure 15 : Simulation de l'Interface Blynk - Version Web

Figure 16 : Simulation de l'Interface Blynk - Version Mobile

III.4. Supervision via ThingSpeak

La plateforme ThingSpeak offre une expérience de supervision riche en fonctionnalités pour le Smart Parking. Sur son interface, la visualisation de données se présente sous la forme de courbes de variation, illustrant en temps réel la distance mesurée par le capteur ultrason. Les trois LEDs, en rouge, jaune et verte, agissent en tant qu'indicateurs visuels reflétant le rôle de la barrière dans le parking, en fonction de la distance du capteur ultrason comme actionneur.

En outre, une icône de valeur numérique accompagne la courbe de distance, fournissant une représentation numérique précise de la distance mesurée. Cela offre une compréhension instantanée des changements dans l'environnement du parking. Deux LEDs distinctes, rouge et verte, signalent la présence ou l'absence d'une voiture devant le capteur infrarouge, offrant une vérification visuelle rapide. Un compteur intégré, représenté par une icône dédiée, effectue un suivi en temps réel du nombre de voitures stationnées. Cette fonctionnalité offre une métrique quantitative pour évaluer l'occupation du parking. La Figure ci-dessous illustre cette interface complète, offrant une supervision détaillée et intuitive des activités du Smart Parking.

Figure 17 : Interface de Supervision ThingSpeak

III.5. Conclusion

L'évaluation approfondie des interfaces, de l'interface série à Blynk et ThingSpeak, confirme la robustesse du Smart Parking. Chaque plateforme offre des fonctionnalités spécifiques, intégrées de manière harmonieuse pour une gestion précise. Ces tests posent les bases solides pour les épreuves opérationnelles à venir, démontrant la fiabilité et l'efficacité de notre système de stationnement intelligent.

Conclusion générale

La recherche incessante de places de stationnement, source de frustration, de gaspillage de temps et d'embouteillages, pose un défi quotidien omniprésent. C'est dans ce contexte que le smart parking émerge comme une solution avant-gardiste, révolutionnant la gestion des espaces dédiés au stationnement par une intégration technologique judicieuse. Cette approche astucieuse permet au smart parking de fournir en temps réel des informations cruciales sur la disponibilité des places, offrant une guidance précise vers les espaces libres et optimisant l'utilisation globale des zones de stationnement. Cette transformation technologique engendre une panoplie d'avantages, de l'amélioration significative de la circulation urbaine à la réduction notable des émissions de gaz, offrant ainsi une expérience de stationnement plus pratique pour les utilisateurs.

Ce rapport, structuré en trois chapitres, a exploré les composants matériels, logiciels et pratiques du smart parking. Le premier chapitre a plongé dans l'anatomie des dispositifs matériels, détaillant chaque composant, du NodeMCU ESP32 aux capteurs spécialisés, jetant ainsi les bases de notre compréhension du système. Le deuxième chapitre s'est concentré sur la programmation et l'intégration logicielle, illustrant comment chaque ligne de code contribue à rendre notre système de gestion de stationnement intelligent et fonctionnel. Enfin, le troisième chapitre a représenté la phase d'évaluation et de validation opérationnelle, confirmant la robustesse du Smart Parking à travers l'interface série, Blynk et ThingSpeak.

Alors que notre projet de smart parking représente une étape significative dans l'amélioration de la gestion des espaces de stationnement, les perspectives d'amélioration restent vastes. L'intégration de technologies émergentes telles que l'intelligence artificielle, l'apprentissage automatique, et l'exploitation de capteurs plus avancés pourraient affiner la prédiction des disponibilités de stationnement.

En conclusion, notre projet de smart parking ne marque pas une fin, mais plutôt le commencement d'un voyage continu vers des solutions plus intelligentes et plus adaptatives pour résoudre les défis complexes de la gestion du stationnement urbain.

Annexe

Annexe 1

```
#define BLYNK_PRINT Serial
#define BLYNK_TEMPLATE_ID "TMPL2W0WTjDxQ"
#define BLYNK_TEMPLATE_NAME "PARKING"
#include <WiFi.h>
#include <Wire.h>
#include <BlynkSimpleEsp32.h>
#include <Ultrasonic.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

#define echoPin 2       // Broche pour le signal de retour du capteur ultrason
#define trigPin 4       // Broche pour le signal de déclenchement du capteur ultrason
#define greenLED 13     // Broche pour la LED verte
#define yellowLED 12    // Broche pour la LED jaune
#define redLED 5        // Broche pour la LED rouge

#define SENSOR_PIN 15   // ESP32 pin GPIO18 connected to OUT pin of IR car detection sensor
#define GREEN_LED 33    // ESP32 pin for green LED
#define RED_LED 32      // ESP32 pin for red LED

long duration, distance_ultrason; // Variables pour stocker la durée du signal retour et la distance mesurée
int carCounter = 0;      // Compteur de voitures
int presence_voiture = 0;

char auth[] = "-hDdGbvXDUHl7QqFGewaarUbVyax42To";
char ssid[] = "Med_Ali";   // votre ssid
char pass[] = "19991999"; // votre mot de passe
BlynkTimer timer;

// Declaration for SSD1306 display connected using I2C
#define OLED_RESET     -1 // Reset pin
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

int isRedLEDOn = 0;
```

```
int isgreenLEDOn = 0;
int isyellowLEDOn = 0;

void setup() {
  Serial.begin(9600);
  Blynk.begin(auth, ssid, pass);
  pinMode(trigPin, OUTPUT);  // Configuration de la broche trigPin en mode sortie
  pinMode(echoPin, INPUT);   // Configuration de la broche echoPin en mode entrée
  pinMode(greenLED, OUTPUT); // Configuration de la broche greenLED en mode sortie
  pinMode(yellowLED, OUTPUT);// Configuration de la broche yellowLED en mode sortie
  pinMode(redLED, OUTPUT);   // Configuration de la broche redLED en mode sortie

  pinMode(SENSOR_PIN, INPUT); // Configuration de la broche SENSOR_PIN en mode entrée
  pinMode(GREEN_LED, OUTPUT); // Configuration de la broche GREEN_LED en mode sortie
  pinMode(RED_LED, OUTPUT);   // Configuration de la broche RED_LED en mode sortie

  // initialize the OLED object
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;)
      ; // Don't proceed, loop forever
  }

  // Clear the buffer.
  display.clearDisplay();

  // Display Text
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0, 28);
  display.println("Smart Parking");
  display.display();
  delay(2000);
  display.clearDisplay();

}



void loop() {
```

```
  // Ultrasonic sensor logic
  digitalWrite(trigPin, LOW);       // Met le signal de déclenchement à bas niveau
  delayMicroseconds(2);             // Attend pendant 2 microsecondes
  digitalWrite(trigPin, HIGH);      // Passe le signal de déclenchement à haut niveau
  delayMicroseconds(10);            // Maintient le signal de déclenchement à haut niveau pendant 10
microsecondes
  digitalWrite(trigPin, LOW);       // Met le signal de déclenchement à bas niveau

  duration = pulseIn(echoPin, HIGH); // Mesure la durée du signal de retour (echo)
  distance_ultrason = duration / 58.2;         // Calcule la distance en fonction de la durée mesurée

  Serial.print("distance_ultrason: ");         // Affiche le texte "Distance: "
  Serial.print(distance_ultrason);             // Affiche la distance
  Serial.println(" cm");             // Affiche " cm" suivi d'un saut de ligne
  delay(3000);                       // Attend 3 secondes avant de répéter la mesure

  if (distance_ultrason < 10) {
    digitalWrite(redLED, HIGH);     // Allume la LED rouge
    isRedLEDOn = 1;
    delay(5000);                    // Attend 3 seconde avec la LED rouge allumée
    digitalWrite(redLED, LOW);      // Éteint la LED rouge
    isRedLEDOn = 0;
    digitalWrite(greenLED, LOW);    // Éteint la LED verte
    isgreenLEDOn = 0;
    digitalWrite(yellowLED, HIGH);  // Allume la LED jaune
    isyellowLEDOn = 1;
    Blynk.virtualWrite(V2, isyellowLEDOn);
    delay(5000);                    // Attend 3 seconde avec la LED jaune allumée
    digitalWrite(yellowLED, LOW);   // Éteint la LED jaune
    isyellowLEDOn = 0;

    digitalWrite(greenLED, HIGH);   // Allume la LED verte
    isgreenLEDOn = 1;
    delay(5000);                    // Attend 3 secondes avec la LED verte allumée
  } else {
    // Allume la LED rouge si la distance est supérieure ou égale à 10 cm
    digitalWrite(redLED, HIGH);
    isRedLEDOn = 1;
    digitalWrite(greenLED, LOW);    // Éteint la LED verte
    isgreenLEDOn = 0;
```

```
  digitalWrite(yellowLED, LOW);    // Éteint la LED jaune
  isyellowLEDOn = 0;
}


// Car detection sensor logic
int state = digitalRead(SENSOR_PIN);

// Display car status on OLED
display.clearDisplay();
display.setTextSize(1);
display.setTextColor(WHITE);
display.setCursor(0, 0);

if (state == LOW) {
  display.println("Car detected!");
  carCounter++;                  // Increment car counter
  presence_voiture = 1;          // Set presence_voiture to 1 when car is detected
  digitalWrite(GREEN_LED, HIGH);   // Turn on the green LED
  digitalWrite(RED_LED, LOW);      // Turn off the red LED
} else {
  display.println("No car detected");
  presence_voiture = 0;          // Set presence_voiture to 0 when no car is detected
  digitalWrite(GREEN_LED, LOW);    // Turn off the green LED
  digitalWrite(RED_LED, HIGH);     // Turn on the red LED
}

// Affiche le nombre de voitures détectées
display.setCursor(0, 20);
display.print("Car count: ");
display.println(carCounter);

// Display presence_voiture and nombre_de_voiture in Serial Monitor
Serial.print("presence_voiture: ");
Serial.println(presence_voiture);
Serial.print("nombre_de_voiture: ");
Serial.println(carCounter);


Blynk.run();
```

```
  timer.run();

  Blynk.virtualWrite(V0, distance_ultrason);
  Blynk.virtualWrite(V1, isRedLEDOn);
  Blynk.virtualWrite(V2, isyellowLEDOn);
  Blynk.virtualWrite(V3, isgreenLEDOn);


  Blynk.virtualWrite(V4, presence_voiture);


  display.display();

}
```

Annexe 2

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <WiFi.h>
#include <HTTPClient.h>


const char *ssid = "Med_Ali";
const char *password = "19991999";
const char *host = "api.thingspeak.com";
const String apiKey = "NGBLSAR692KOH3U4";
const String channelID = "2373745";

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

#define echoPin 2       // Broche pour le signal de retour du capteur ultrason
#define trigPin 4       // Broche pour le signal de déclenchement du capteur ultrason
#define greenLED 13     // Broche pour la LED verte
#define yellowLED 12    // Broche pour la LED jaune
#define redLED 5        // Broche pour la LED rouge

#define SENSOR_PIN 15   // ESP32 pin GPIO18 connected to OUT pin of IR car detection sensor
#define GREEN_LED 33    // ESP32 pin for green LED
#define RED_LED 32      // ESP32 pin for red LED

long duration, distance; // Variables pour stocker la durée du signal retour et la distance mesurée
int carCounter = 0;      // Compteur de voitures

// Declaration for SSD1306 display connected using I2C
#define OLED_RESET     -1 // Reset pin
#define SCREEN_ADDRESS 0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);  // Configuration de la broche trigPin en mode sortie
  pinMode(echoPin, INPUT);   // Configuration de la broche echoPin en mode entrée
  pinMode(greenLED, OUTPUT); // Configuration de la broche greenLED en mode sortie
```

```
  pinMode(yellowLED, OUTPUT);// Configuration de la broche yellowLED en mode sortie
  pinMode(redLED, OUTPUT);   // Configuration de la broche redLED en mode sortie

  pinMode(SENSOR_PIN, INPUT); // Configuration de la broche SENSOR_PIN en mode entrée
  pinMode(GREEN_LED, OUTPUT); // Configuration de la broche GREEN_LED en mode sortie
  pinMode(RED_LED, OUTPUT);   // Configuration de la broche RED_LED en mode sortie

  // initialize the OLED object
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;)
      ; // Don't proceed, loop forever
  }

  // Clear the buffer.
  display.clearDisplay();

  // Display Text
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0, 28);
  display.println("Smart Parking");
  display.display();
  delay(2000);
  display.clearDisplay();

  // Connexion au réseau WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connexion WiFi en cours...");
  }
  Serial.println("Connecté au WiFi");
}

void loop() {
  // Ultrasonic sensor logic
  digitalWrite(trigPin, LOW);       // Met le signal de déclenchement à bas niveau
  delayMicroseconds(2);             // Attend pendant 2 microsecondes
  digitalWrite(trigPin, HIGH);      // Passe le signal de déclenchement à haut niveau
```

```
  delayMicroseconds(10);            // Maintient le signal de déclenchement à haut niveau pendant 10
microsecondes
  digitalWrite(trigPin, LOW);       // Met le signal de déclenchement à bas niveau

  duration = pulseIn(echoPin, HIGH); // Mesure la durée du signal de retour (echo)
  distance = duration / 58.2;         // Calcule la distance en fonction de la durée mesurée

  Serial.print("Distance: ");         // Affiche le texte "Distance: "
  Serial.print(distance);             // Affiche la distance

  if (distance < 10) {
    digitalWrite(yellowLED, LOW);       // Éteint la LED rouge
    digitalWrite(greenLED, LOW);     // Éteint la LED verte
    delay(1000);
    digitalWrite(redLED, HIGH);     // Allume la LED rouge
    sendToThingSpeak(distance,1,0,0,carCounter,0,0);
    delay(5000);                    // Attend 1 seconde avec la LED rouge allumée
    digitalWrite(redLED, LOW);    // Éteint la LED jaune
    digitalWrite(greenLED, LOW);     // Éteint la LED verte
    delay(1000);
    digitalWrite(yellowLED, HIGH);   // Allume la LED jaune
    sendToThingSpeak(distance,0,1,0,carCounter,0,0);
    delay(5000);                    // Attend 1 seconde avec la LED jaune allumée
    digitalWrite(redLED, LOW);
    digitalWrite(yellowLED, LOW);
    delay(1000);
    digitalWrite(greenLED, HIGH);    // Allume la LED verte
    sendToThingSpeak(distance,0,0,1,carCounter,0,0);
    delay(5000);                    // Attend 2 secondes avec la LED verte allumée

  } else {
    // Allume la LED rouge si la distance est supérieure ou égale à 10 cm

    digitalWrite(redLED, HIGH);
    sendToThingSpeak(distance,1,0,0,carCounter,0,0);
    digitalWrite(greenLED, LOW);     // Éteint la LED verte
    digitalWrite(yellowLED, LOW);    // Éteint la LED jaune
  }

  Serial.println(" cm");              // Affiche " cm" suivi d'un saut de ligne
```

```
  delay(1000);                    // Attend 3 secondes avant de répéter la mesure

  // Car detection sensor logic
  int state = digitalRead(SENSOR_PIN);

  // Envoi de la distance et l'état de la LED rouge à ThingSpeak
  sendToThingSpeak(distance, digitalRead(redLED), digitalRead(yellowLED), digitalRead(greenLED),
carCounter, digitalRead(GREEN_LED), digitalRead(RED_LED) );

  // Display car status on OLED
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0, 0);

  if (state == LOW) {
    display.println("Car detected!");
    delay(1000);
    carCounter++;                 // Incrémente le compteur de voitures
    digitalWrite(RED_LED, LOW);      // Éteint la LED rouge
    digitalWrite(GREEN_LED, HIGH);   // Allume la LED verte
    sendToThingSpeak(distance,0,0,0,carCounter,1,0);
    delay(5000);
  } else {
    display.println("No car detected");
    delay(1000);
    digitalWrite(GREEN_LED, LOW);    // Éteint la LED verte
    digitalWrite(RED_LED, HIGH);     // Allume la LED rouge
    sendToThingSpeak(distance,0,0,0,carCounter,0,1);
    delay(5000);
  }

  // Affiche le nombre de voitures détectées
  display.setCursor(0, 20);
  display.print("Car count: ");
  display.println(carCounter);

  display.display();
  delay(3000);
}
```

```
void sendToThingSpeak(float distance, int redLEDState, int yellowLEDState, int greenLEDState, int
carCounter, int GREENLEDState, int REDLEDState) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;

    // Construit l'URL pour l'API ThingSpeak
    String url = "http://api.thingspeak.com/update?api_key=" + apiKey +
                 "&field1=" + String(distance) +
                 "&field2=" + String(redLEDState)+
                 "&field3=" + String(yellowLEDState)+
                 "&field4=" + String(greenLEDState)+
                 "&field5=" + String(carCounter)+
                 "&field6=" + String(GREENLEDState)+
                 "&field7=" + String(REDLEDState);

    http.begin(url);

    int httpCode = http.GET();
    if (httpCode > 0) {
      Serial.println("Envoi à ThingSpeak réussi");
    } else {
      Serial.println("Erreur d'envoi à ThingSpeak");
    }

    http.end();
  } else {
    Serial.println("Pas de connexion WiFi");
  }
}
```