RÉPUBLIQUE DU SÉNÉGAL

UNIVERSITÉ CHEIKH ANTA DIOP DE DAKAR

ÉCOLE SUPÉRIEURE POLYTECHNIQUE

DÉPARTEMENT GÉNIE INFORMATIQUE

Rapport du Projet de Système Embarqué

SUJET :

Système d'Information et de Navigation pour les Passagers des Autobus (Abribus connecté)

Présenté par

Rawane DIOUF

Mouhamed Lamine FAYE

Adolphe Amadou GALLAND

Cheikh Ahmadou Bamba SALL

Cheikh Ahmed Tidiane THIANDOUM

Sous la supervision de

Dr Mangoné FALL

Année universitaire : 2023-2024

Table des matières

I Introduction

La conception d'un système d'information et de navigation pour les passagers des autobus a pour objectif de fournir des informations en temps réel sur le trafic et les horaires des bus. À chaque arrêt de bus, un écran affichera ces informations, améliorant ainsi l'expérience des usagers et leur permettant de planifier leurs trajets de manière plus efficace.

II Définition des objectifs du projet

— **Fournir des informations en temps réel :** Afficher les horaires des bus, les retards et les informations sur le trafic.

— **Améliorer l'expérience des passagers :** Réduire l'incertitude et le temps d'attente perçu.

— **Optimiser la gestion des flux de passagers :** Aider les passagers à prendre des décisions éclairées.

III Conception de l'architecture

1 Sélection des composants clés

— **GPS des bus :** Pour suivre la position en temps réel des autobus.

— **Écrans aux arrêts de bus :** Pour afficher les informations aux passagers.

2 Choix des technologies et des protocoles

— **Protocole de communication :** Utilisation de MQTT pour la transmission des données en temps réel.

— **Technologies de réseau :** Wi-Fi et 4G pour la connectivité des dispositifs.

3 Conception des couches de l'architecture

Pour garantir une architecture IoT robuste et efficace, il est essentiel de structurer le système en plusieurs couches distinctes, chacune ayant des rôles spécifiques : la couche de perception pour la collecte des données, la couche de réseau pour la transmission des données, la couche de traitement pour l'analyse des informations, et la couche d'application pour l'interaction utilisateur.

— **Couche de perception :** Capteurs GPS intégrés aux bus.

— **Couche de réseau :** Réseau Wi-Fi et 4G pour la transmission des données.

— **Couche de traitement :** Serveurs cloud pour le traitement et l'analyse des données en temps réel.

— **Couche d'application :** Écrans aux arrêts de bus pour les passagers.

IV Développement du prototype

1 Couche de perception

N'ayant pas de bus sous la main, nous avons décidé de simuler le déplacement des bus par un programme.

En effet, nous avons mis en place un programme nous permettant de faire "rouler" un bus sur un itinéraire en envoyant ça position sur un intervalle de temps régulier.

```
Position Message{type=POSITION_SENSOR, value=PositionSensorData{id='1', position=Position{longitude='-17.40437', latitude='14.75654'}}}
Position Message{type=POSITION_SENSOR, value=PositionSensorData{id='1', position=Position{longitude='-17.40488', latitude='14.75587'}}}
Position Message{type=POSITION_SENSOR, value=PositionSensorData{id='1', position=Position{longitude='-17.40491', latitude='14.75581'}}}
Position Message{type=POSITION_SENSOR, value=PositionSensorData{id='1', position=Position{longitude='-17.40575', latitude='14.7547'}}}
Position Message{type=POSITION_SENSOR, value=PositionSensorData{id='1', position=Position{longitude='-17.40601', latitude='14.75429'}}}
Position Message{type=POSITION_SENSOR, value=PositionSensorData{id='1', position=Position{longitude='-17.40651', latitude='14.75354'}}}
Position Message{type=POSITION_SENSOR, value=PositionSensorData{id='1', position=Position{longitude='-17.40689', latitude='14.75298'}}}
Position Message{type=POSITION_SENSOR, value=PositionSensorData{id='1', position=Position{longitude='-17.40695', latitude='14.75289'}}}
Position Message{type=POSITION_SENSOR, value=PositionSensorData{id='1', position=Position{longitude='-17.40699', latitude='14.75283'}}}
```

FIGURE 1 – Simulation du déplacement du bus : messages reçus par le contrôleur

2 Couche de communication

La communication se fait à l'aide du protocole MQTT.

Il est approprié pour les applications IoT en raison de sa légèreté et de son efficacité dans la transmission de données à faible bande passante.

Il utilise un modèle de publication/abonnement qui minimise la consommation d'énergie des appareils connectés tout en permettant une communication asynchrone fiable entre de nombreux clients et un serveur centralisé.

Nous l'utilisons comme suit :

- **Publisher** :
  - **Module de localisation du bus** : Il envoie des coordonnées GPS au niveau du contrôleur.
  - **Contrôleur (Own server)** : Il envoie des informations sur le trafic affichés aux arrêts de bus.

- **Subscriber** :
  - **Module d'affichage aux arrêts de bus** : Il reçoit des informations sur le trafic depuis le contrôleur.
  - **Contrôleur (Own server)** : Il reçoit les coordonnées GPS des différents bus.

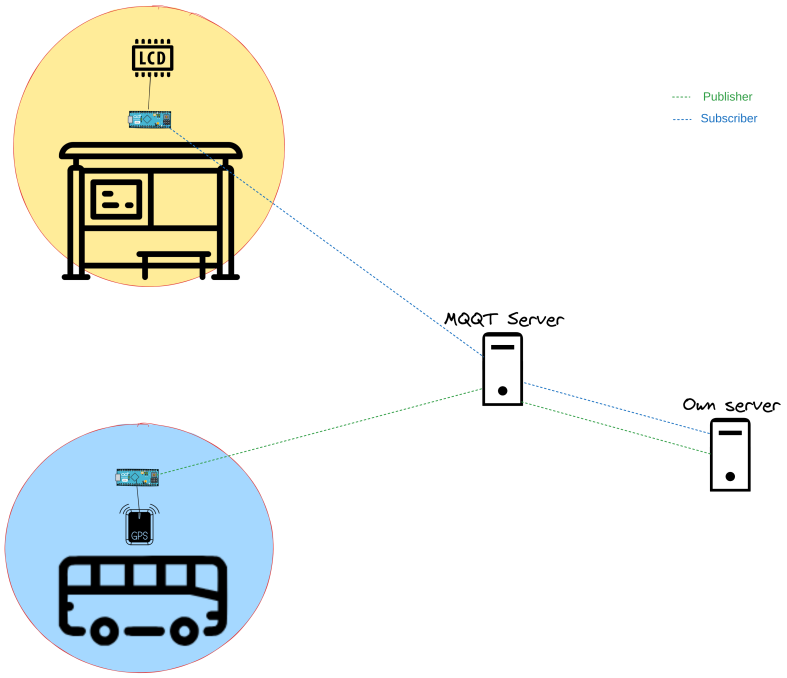

FIGURE 2 – Schéma de représentation de la communication entre les différentes composants

3 Couche de traitement

3.1 Modélisation

Diagramme de classe

Le diagramme de classe est un outil de modélisation utilisé pour représenter de manière statique un système logiciel. Il met en relation différentes classes ainsi que des types de données et des types énumérés.

Une esquisse d'un diagramme de classe de notre projet est représenté dans la figure ci-dessous.

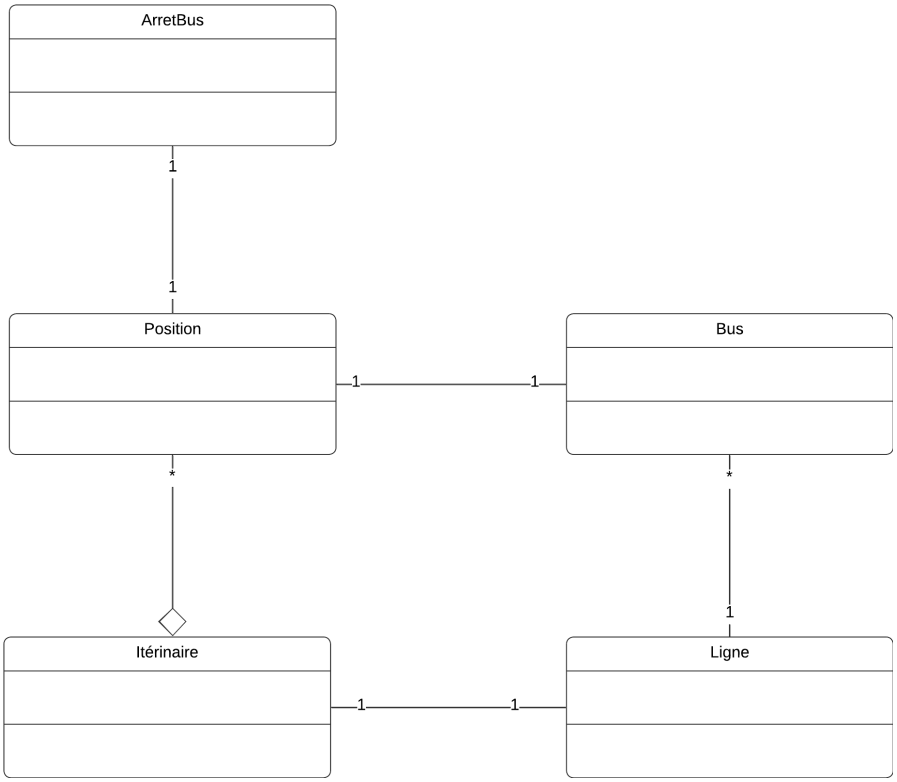

FIGURE 3 – Diagramme de classe d'analyse

3.2 Choix de technologie

Java

Java est largement utilisé en raison de sa portabilité, ce qui signifie que le code écrit

FIGURE 4 – Logo de Java

en Java peut fonctionner sur différentes plateformes sans nécessiter de modifications

majeures. De plus, Java est apprécié pour sa sécurité intégrée grâce à son système de gestion de la mémoire et à son modèle de gestion des exceptions, qui contribuent à la fiabilité et à la robustesse des applications.

4 Couche d'application

Un écran LCD nous renseignant sur les informations du trafic en temps réel.

5 Développement logiciel

5.1 Programmation des microcontrôleurs

Le code commence par inclure les bibliothèques nécessaires : `ArduinoJson` pour manipuler les objets JSON, `Wire` pour la communication I2C, `hd44780` et `hd44780_I2Cexp` pour contrôler l'écran LCD, `WiFi` pour la connexion réseau, et `PubSubClient` pour la communication MQTT.

Un objet LCD est déclaré avec les spécifications de l'écran (16 colonnes et 2 lignes). Les informations de connexion Wi-Fi, telles que le SSID et le mot de passe, ainsi que les détails du broker MQTT (adresse, topic, port) sont définis.

Ensuite, deux fonctions sont créées pour désérialiser les chaînes JSON : `byteJsonParse` pour les données au format `byte` et `charJsonParse` pour les données au format `char`. Ces fonctions vérifient également les erreurs de désérialisation.

Dans la fonction `setup`, la vitesse de transmission série est définie à 115200 bauds, l'écran LCD est initialisé et un message est affiché. Le module ESP32 se connecte ensuite

au réseau Wi-Fi. Une fois connecté, il se connecte au broker MQTT et s'abonne au topic spécifié.

La fonction `callback` est appelée lorsqu'un message est reçu sur le topic abonné. Elle affiche le message reçu sur le moniteur série et désérialise le contenu JSON pour extraire les valeurs nécessaires. Ces valeurs sont ensuite affichées sur l'écran LCD.

Enfin, la boucle principale (`loop`) assure que le client MQTT reste connecté et traite les messages entrants en appelant `client.loop()`.

```cpp
#include <ArduinoJson.h>
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h> // include i/o class header
#include <WiFi.h>
#include <PubSubClient.h>

  // LCD Screen
hd44780_I2Cexp lcd; // declare lcd object: auto locate & config display for hd44780 chip
const int LCD_COLS = 16;   // Nombre de colonnes de l'écran LCD
const int LCD_ROWS = 2;    // Nombre de lignes de l'écran LCD

 // WiFi
const char * ssid = "Antidote"; // Entrez votre SSID WiFi
const char * password = "304error"; // Entrez votre mot de passe WiFi

// MQTT Broker
const char * mqtt_broker = "192.168.146.228";
const char * topic = "test/busStop1DataTransfer";
const char * mqtt_username = "";
const char * mqtt_password = "";
const int mqtt_port = 1883;

WiFiClient espClient;
PubSubClient client(espClient);

DynamicJsonDocument byteJsonParse(byte* str) {
  // Créer un objet JSON pour stocker les données
  DynamicJsonDocument doc(512); // Taille du document JSON, ajustez selon vos besoins

  // Désérialiser le JSON
  DeserializationError error = deserializeJson(doc, str);

  // Vérifier les erreurs de désérialisation
  if (error) {
    Serial.print("Erreur lors de la désérialisation : ");
    Serial.println(error.c_str());
  }

  // Extraire les valeurs nécessaires
  return doc;

}
```

```
DynamicJsonDocument charJsonParse(const char* str) {
  // Créer un objet JSON pour stocker les données
  DynamicJsonDocument doc(512); // Taille du document JSON, ajustez selon vos besoins

  // Désérialiser le JSON
  DeserializationError error = deserializeJson(doc, str);

  // Vérifier les erreurs de désérialisation
  if (error) {
    Serial.print("Erreur lors de la désérialisation : ");
    Serial.println(error.c_str());
  }

  // Extraire les valeurs nécessaires
  return doc;

}
```

```
void setup() {
    //Mise de la vitesse de transmission à 115200;
    Serial.begin(115200);

    // initialize LCD with number of columns and rows:
    lcd.begin(LCD_COLS, LCD_ROWS);
    // Print a message to the LCD
    lcd.print("Arret de bus 1 I1");

    // Connecting to a Wi-Fi network
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.println("Connecting to WiFi..");
    }
    Serial.println("Connected to the Wi-Fi network");
    //connexion au broker MQTT
    client.setServer(mqtt_broker, mqtt_port);
    client.setCallback(callback);
    while (!client.connected()) {
      String client_id = "esp32-client-";
      client_id += String(WiFi.macAddress());
      Serial.printf("The client %s connects to the public MQTT brokern \n", client_id.c_str());
      if (client.connect(client_id.c_str(), mqtt_username, mqtt_password)) {
        Serial.println("Public EMQX MQTT broker connected");
      } else {
        Serial.print("failed with state ");
        Serial.print(client.state());
        delay(2000);
      }
    }
    // Publish et subscribe
    //client.publish(topic, "Hi, I'm ESP32");
    client.subscribe(topic);

}
```

```
// Reception du message MQTT
void callback(char * topic, byte * payload, unsigned int length) {
    Serial.print("Message arrived in topic: ");Serial.println(topic);
    Serial.print("=>");
    for (int i = 0; i < length; i++) {
      Serial.print((char) payload[i]);
    }
    Serial.println("");

    // Extraire les valeurs nécessaires
    DynamicJsonDocument payloadDoc = byteJsonParse(payload);
    const char* value = payloadDoc["value"];
    DynamicJsonDocument valueDoc = charJsonParse(value);

    // Extraire les valeurs nécessaires
    const char* message = valueDoc["message"];
    const char* busId = valueDoc["id"];

    // Afficher les valeurs extraites
    Serial.print("Message : ");Serial.println(message);
    Serial.println("-----------------------");

    lcd.clear();
    //lcd.autoscroll();

    lcd.setCursor(0, 0);
    lcd.print("BUS "); lcd.print(busId);
    lcd.print(" A "); lcd.print(message); lcd.print("KM");
    lcd.setCursor(0, 1);
    lcd.print("Arrive dans "); lcd.print(message); lcd.print("min");


}

void loop() {
  client.loop();

}
```

5.2 Développement du contrôleur

Le contrôleur est responsable de la collecte, du traitement et de la distribution des données en temps réel. Cela inclut les horaires des autobus, les positions GPS, les conditions de trafic, et toute autre information pertinente. Les fonctions spécifiques comprennent :

— **Collecte des Données** : Récupération des informations des capteurs GPS installés dans les autobus et des autres sources de données.

— **Traitement des Données** : Filtrage, agrégation et analyse des données pour générer

des informations utiles.

— **Distribution des Données** : Diffusion des informations traitées aux écrans d'affichage dans les arrêts de bus.

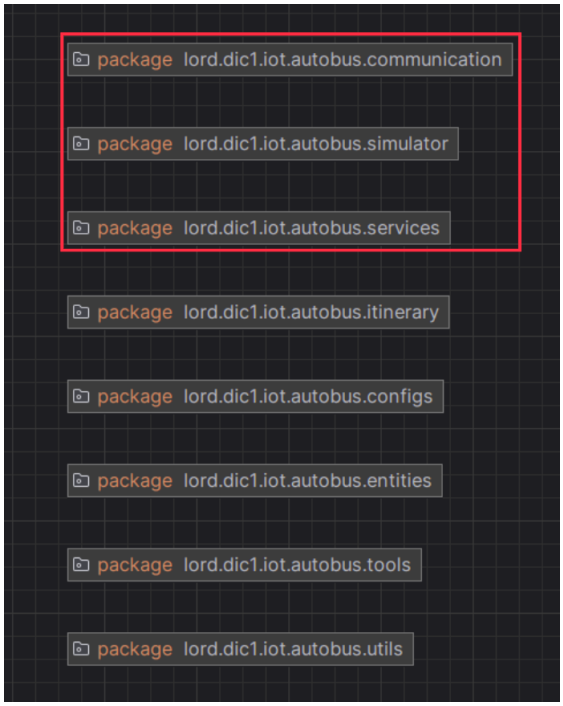

FIGURE 5 – architecture global

6 Intégration des systèmes

Assurer la compatibilité et l'interopérabilité entre les différents composants matériels et logiciels.

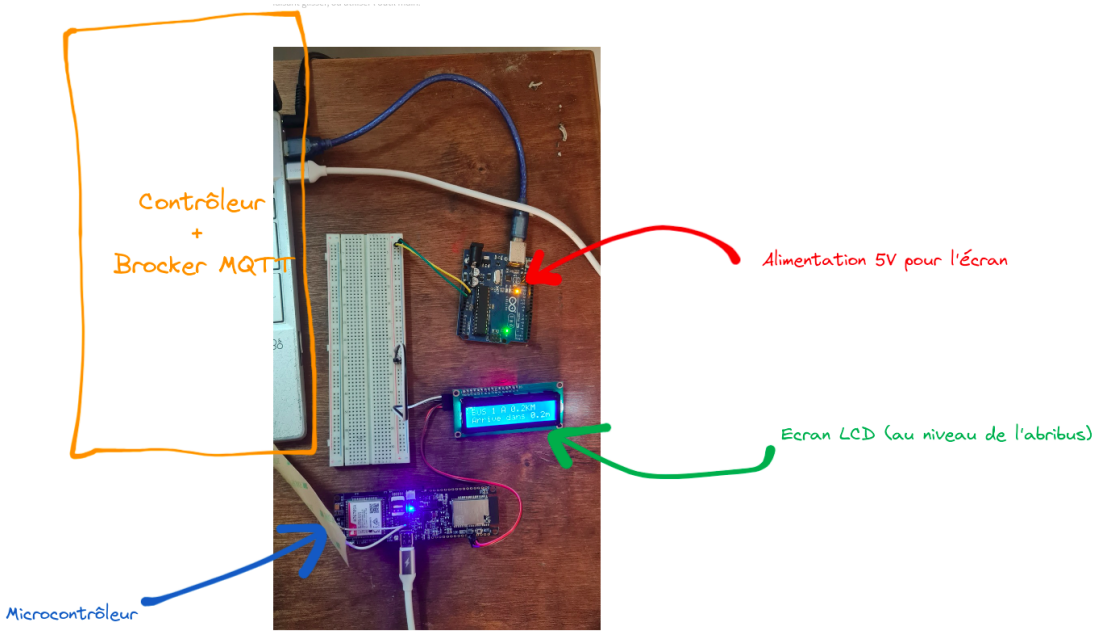

FIGURE 6 – Enter Caption

Conclusion

Une méthodologie bien structurée pour la conception nous a permis de garantir la réussite du projet. En suivant ces étapes, nous pouvons développer cette solution.