

**Intelligent and Communicating Systems, ICS**  
2<sup>nd</sup> Year Specialty SIQ G02

# Project: Intelligent SMS Gateway Implementation Guide

Studied by:

HADDAD Amira  
REBHI Assala Nour Elimane  
DERROUECHE Samia  
DEBBIH Ikram Zineb  
KOUADRI Nada  
GHODBANE Zineb  
ALLOUCHE Imene

# Contents

<b>A. Project Context</b>	<b>3</b>
<b>B. Option 1</b>	<b>4</b>
1 Hardware	4
1.1 GSM Module Specifications:	4
1.2 Components:	4
1.3 Setup of the circuit:	4
2 Software	5
2.1 Configuration of Google Calender:	5
2.2 Configuration of RaspiSMS:	9
2.3 Configuration of Gammu:	13
2.4 Development of a Web Application	16
<b>C. Option 2</b>	<b>17</b>
1 Hardware	17
1.1 DHT22 Temperature and Humidity Sensor:	17
1.2 Components:	17
1.3 Setup of the circuit:	17
2 Software	18
2.1 What is Microsoft Azure	18
2.2 Connecting to Azure IoT Hub:	18
2.3 Storing Data on Azure Blob Storage	20
2.4 Creating an Azure Stream Analytics	20
2.5 Define a Query	21
2.6 Start the Job	21
<b>D. Final result: Demonstration of the application</b>	<b>22</b>
1 User interface: Remindini	22
2 Admin interface: RaspiSMS	23
<b>E. Conclusion</b>	<b>26</b>
<b>F. Bibliography</b>	<b>27</b>

# A. Project Context

With the increasing need for real-time notifications, SMS alerts provide an effective way to stay updated on important events. This project focuses on integrating SMS notifications for Google Calendar events and sensor-based alerts, ensuring that users receive timely reminders and critical warnings. The project explores two implementation options:

- A direct SMS gateway linked to Google Calendar
- An advanced IoT-based solution leveraging an edge computing platform

## Objectives:

The main goal is to develop an SMS gateway capable of sending notifications based on calendar events and sensor triggers. This will be achieved through the following two approaches:

- **Option 1: SMS Gateway using Raspberry Pi**
  - A basic implementation where a Raspberry Pi, a GSM module, and Python scripts interact with Google Calendar to send event-based SMS notifications.
  - Uses Raspbian, RaspiSMS, and Google Calendar API for communication.
- **Option 2: Intelligent SMS Gateway with Edge Computing**
  - A more advanced version that integrates an edge platform to process data locally before sending notifications.
  - Enables additional real-time processing and scalability for IoT applications.

# B. Option 1

## 1 Hardware

### 1.1 GSM Module Specifications:

The SIM800L v2 GSM module requires a stable 5V power supply with a current of 3A, which is provided by an external power supply. To function correctly, the SIM card must be inserted into the module before turning it on; otherwise, it will not be detected. If there are power interruptions (voltage drops or outages), the GSM module will be affected, causing it to shut down and restart once power is restored.

### 1.2 Components:

- Raspberry Pi 4 Model B
- GSM Module SIM800L v2
- MicroSD Card
- Power Supply
- Cables and Connectors

### 1.3 Setup of the circuit:

#### Connections:

- Connect the Raspberry Pi GPIO TX pin to the GSM Module RX pin (for sending data).
- Connect the Raspberry Pi GPIO RX pin to the GSM Module TX pin (for receiving data).
- Connect the Raspberry Pi GND pin to the GSM Module GND pin.
- Connect the GSM Module VCC pin to a 5V power source (external power supply).
- Connect the GSM Module GND pin to the external power supply GND.

The figure below shows the connection between the Raspberry Pi and the GSM module:

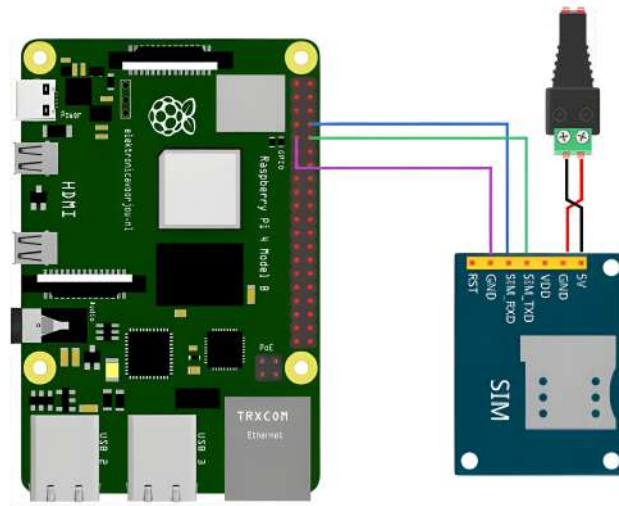


Figure 1: Schematic Representation of the circuit for option 1

## 2 Software

### 2.1 Configuration of Google Calender:

These steps are necessary to integrate Google Calendar with a Raspberry Pi-based SMS gateway.

#### Creating a Google Cloud Account

This step sets up a workspace in Google Cloud where APIs and credentials can be managed.

- Go to Google Cloud Console and sign in with your Google account.
- Create a new project by clicking on **Select a project > New Project**.
- Give the project a name (e.g: *SMS Gateway*) then click on **Create**.

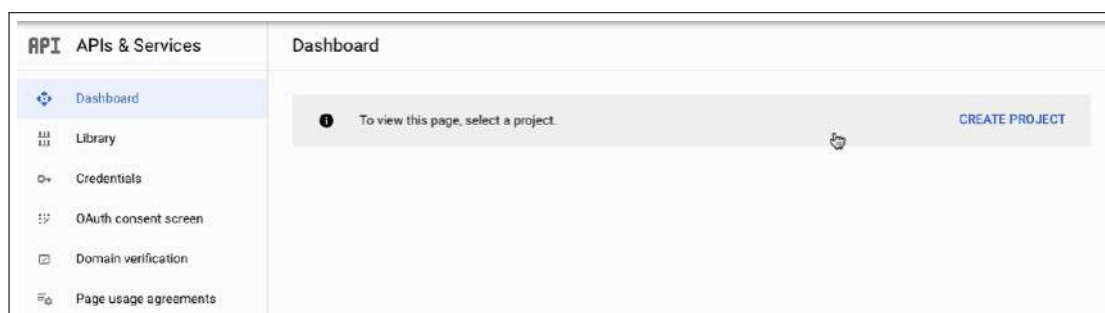
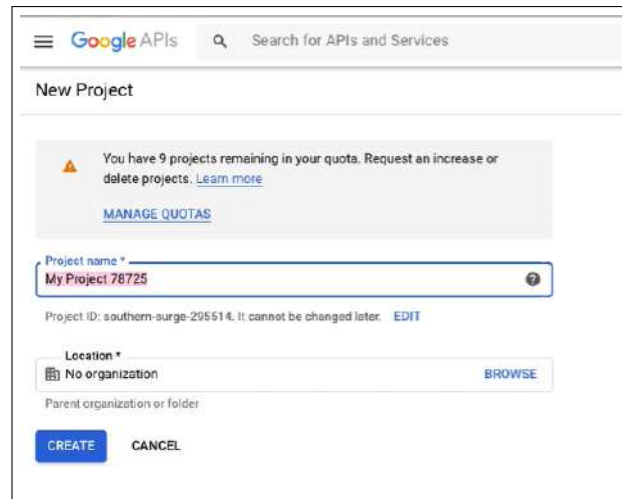


Figure 2: API & Services section



The screenshot shows the 'New Project' form in the Google APIs console. At the top, there's a search bar and a 'New Project' title. Below the title, a warning message states: 'You have 9 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)'. A link 'MANAGE QUOTAS' is also present. The 'Project name' field is filled with 'My Project 78725'. Below it, the 'Project ID' is 'southern-surge-295514', with a note that it cannot be changed later and an 'EDIT' link. The 'Location' is set to 'No organization' with a 'BROWSE' button. At the bottom, there are 'CREATE' and 'CANCEL' buttons.

Figure 3: Creating a new project

## Enabling the Google Calendar API

The Google Calendar API allows external applications (Python script on Raspberry Pi) to access calendar events programmatically.

- In Google Cloud Console, go to the **Library**.

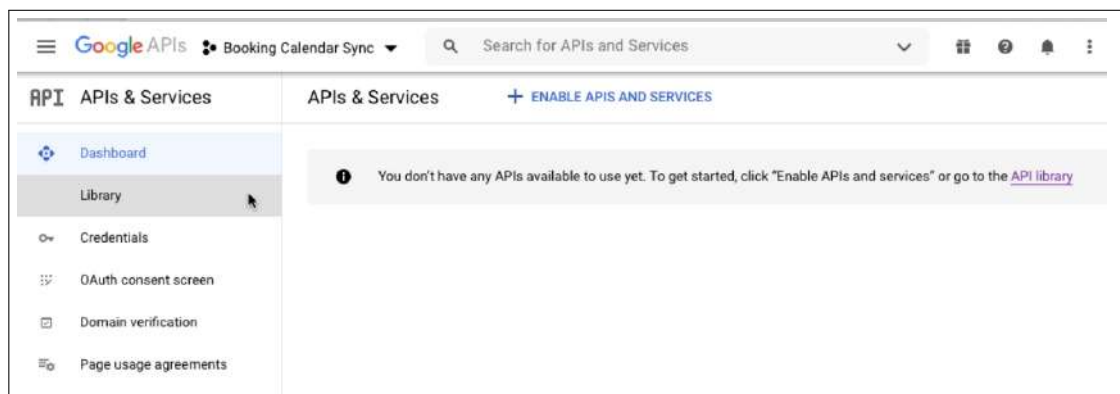


Figure 4: Library's section

- Search for **Google Calendar API** then click on **Enable**.

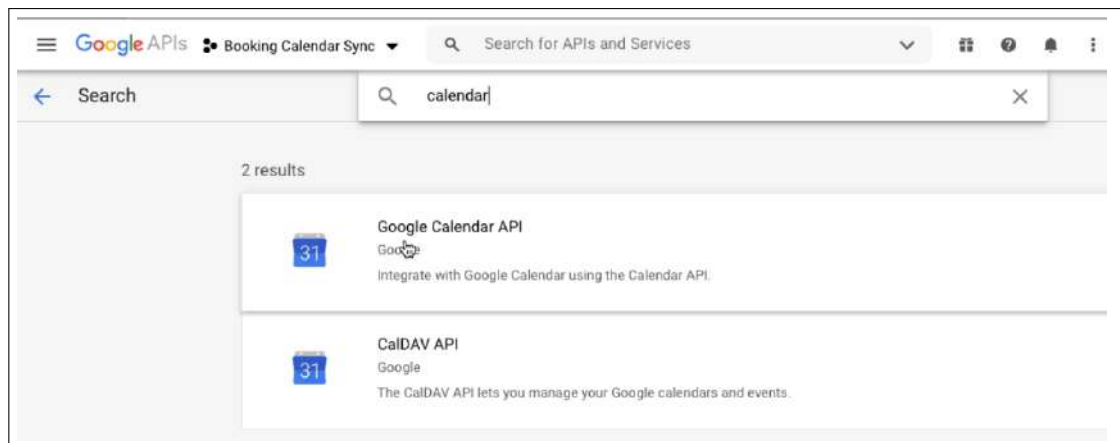


Figure 5: Enable Google Calendar API -step01-

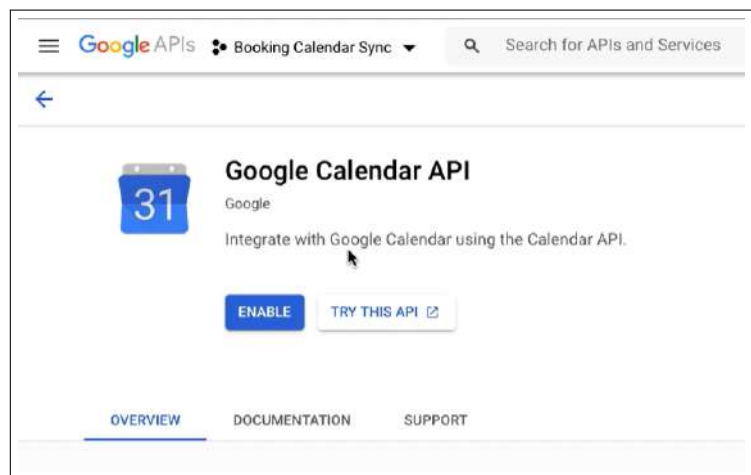
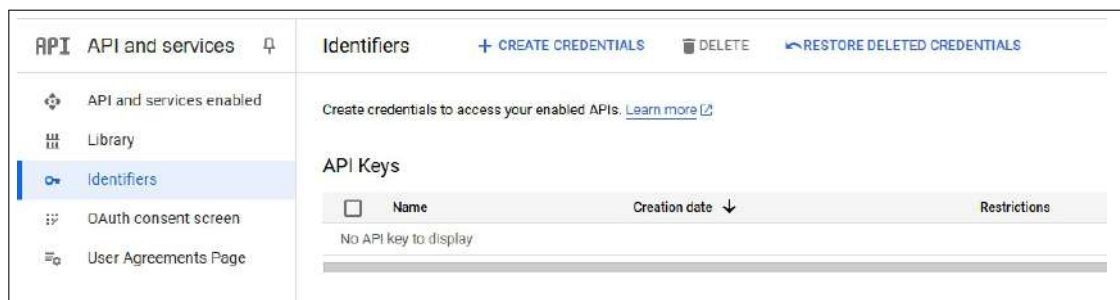


Figure 6: Enable Google Calendar API -step02-

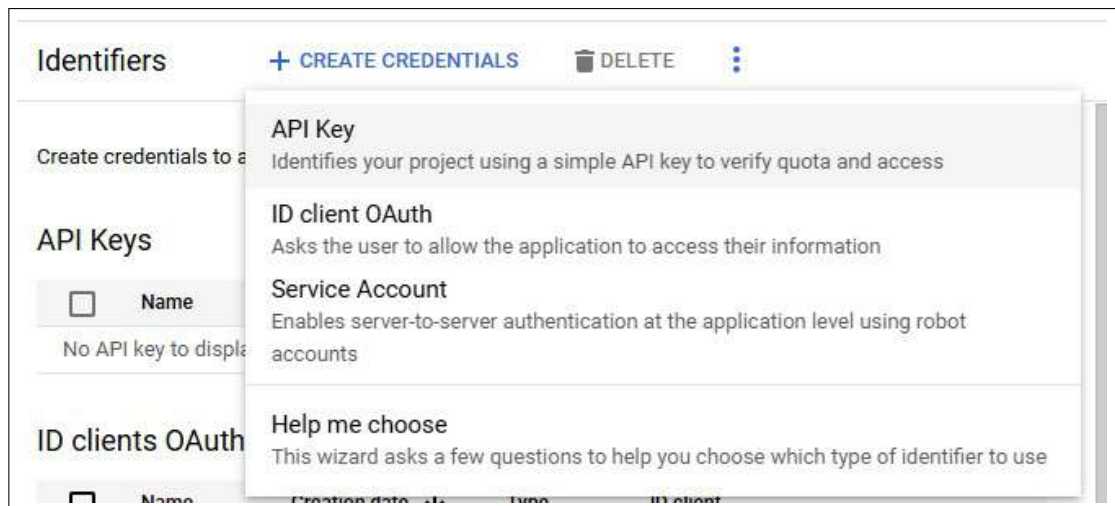
### Creating OAuth Credentials:

OAuth credentials (a service account with a key file) provide authentication for the Python script to securely access Google Calendar.

- Go to **API & Services > Credentials**.



- Click on **Create Credentials > ID client OAuth**.



- If prompted, configure the **OAuth consent screen**:
  - Choose **External** if you need broader access, otherwise select **Internal** (for organization use).
  - Enter basic details such as **app name** and **support email**.
  - Under **Scopes**, add <https://www.googleapis.com/auth/calendar> for calendar access.
  - Click **Save and Continue** until the setup is complete.

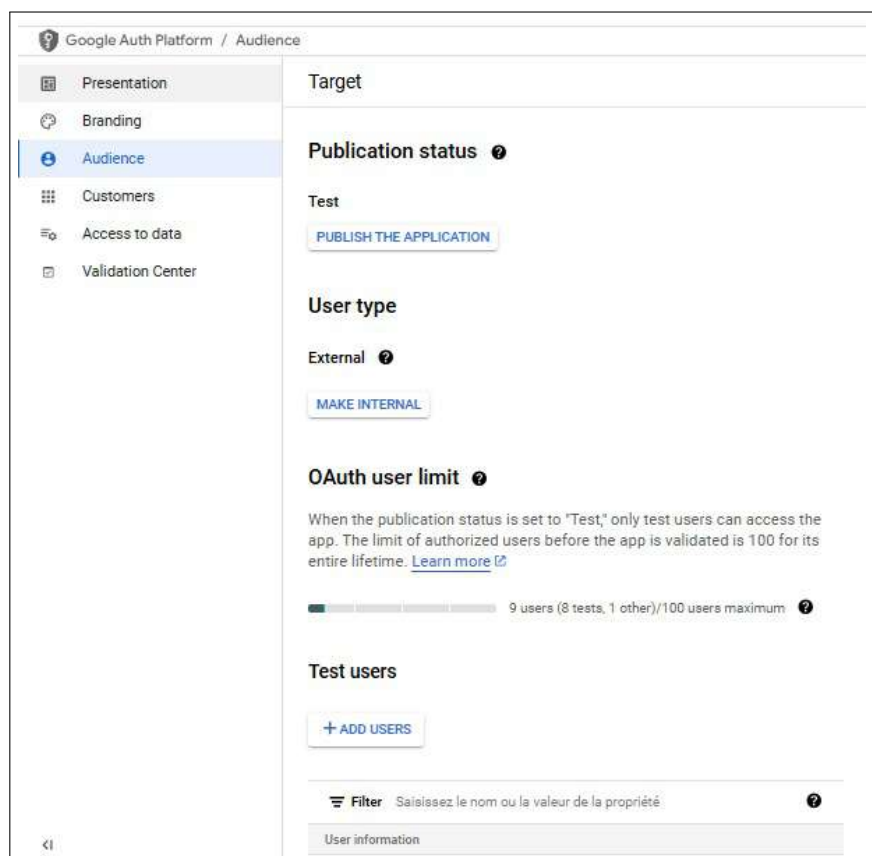


Figure 7: Audience's section



- In the **Application Type** section, choose:
  - **Desktop App** for local execution.
  - **Web Application** if running on a server.
- Give a name to the client (e.g., *Google Calendar App*).

← Create an OAuth Client ID

A client ID is used to uniquely identify an application to Google's OAuth servers. If your application runs on multiple platforms, each platform will need its own client ID. For more information, see [Setting up OAuth 2.0](#) or [Learn more](#) on OAuth client types.

Application type \*  
Desktop application

Name \*  
Google calendar app

Name of your OAuth 2.0 client. This name is only used to identify the client in the console. It is not visible to end users.

Note: It may take anywhere from five minutes to a few hours for the settings to apply.

CREATE CANCEL

Figure 8: Creating an OAuth Client ID

- Click **Create**, then download the `credentials.json` file.

OAuth 2.0 Client IDs				
<input type="checkbox"/>	Name	Creation date ↓	Kind	Customer ID
<input type="checkbox"/>	SMS	February 2, 2025	Desktop computer	796417999936-bb7u...

Figure 9: OAuth Client ID's section

## Installing the Required Libraries on Raspberry Pi

The necessary Python libraries enable the script to communicate with Google Calendar and process event data.

```
pip3 install google-auth google-auth-oauthlib google-auth-httpplib2
google-api-python-client
```

## 2.2 Configuration of RaspiSMS:

### What is RaspiSMS

RaspiSMS is an open-source SMS gateway designed for Raspberry Pi. It allows users to send and receive SMS messages via a GSM module connected to the Raspberry Pi. The system provides a web interface to manage messages, schedule SMS, and configure sending rules. It has many features such as the following:

- Web-based dashboard to manage and monitor SMS activity from a browser.
- Message scheduling: Automate SMS sending at specific times.
- Easily integrate with Python or Bash scripts for automation.
- Multi-user support securely.

### Installation of RaspiSMS:

#### Add the apt.raspisms.fr Repository

The commands below will install all the packages needed to use APT with SSL, as well as the repository's public key.

- Update the package list:

```
sudo apt update -y
```

- Install necessary packages to enable APT to use secure HTTPS connections:

```
sudo apt install -y apt-transport-https gnupg2 curl
```

- Add the official RaspiSMS repository to the APT sources list:

```
echo "deb https://apt.raspisms.fr/ buster contrib" | sudo tee  
-a /etc/apt/sources.list.d/raspisms.list
```

This command adds the repository address to a separate file (`raspisms.list`) instead of modifying `/etc/apt/sources.list` directly.

- Import the repository's GPG key to verify package authenticity:

```
curl https://apt.raspisms.fr/conf/pub.gpg.key | sudo apt-key  
add -
```

- Update the package list again after adding the repository:

```
sudo apt update -y
```

#### Install the RaspiSMS Package

Run the following command and follow the installer instructions:

```
sudo apt install raspisms
```

During installation, a setup wizard will appear to help configure the service. Follow the on-screen instructions to complete the installation.

## Find Your RaspiSMS Credentials

After installation, RaspiSMS automatically generates login credentials (username and password). These are stored in a specific file. To display the credentials, run:

```
sudo cat /usr/share/raspisms/.credentials
```

## Log In

Open a browser and go to the server's IP address (Raspberry Pi's IP address) followed by `/raspisms`. For example, if your server has the IP `127.0.0.1`, go to:

```
http://127.0.0.1/raspisms
```

## Using the RaspiSMS Database

Those steps explain how to access the RaspiSMS database to examine its structure and use this information to develop our application that interacts with the message tables.

## Accessing the RaspiSMS Installation Directory

To navigate to the main installation directory of RaspiSMS on the Raspberry Pi, use:

```
cd /usr/share/raspisms
```

## Exploring Configuration Files

To understand how to connect to the database, we need to examine the configuration files:

```
sudo cat env.prod.php
```

This file contains database connection parameters, including:

- Hostname: `localhost`
- Database name: `raspisms`
- Username: `raspisms`
- Password: `GjN0ZCgUI0F8POUzud1eRuFoov1QYTtE`

## Connecting to the Database

Use the following command to connect to the MySQL/MariaDB server with the `raspisms` user:

```
mysql -u raspisms -p
```

The system will then prompt you to enter the password found in the configuration file.

## Exploring the Database Structure

Once connected, you can explore the database structure using the following SQL commands:

```
SHOW DATABASES;  
USE raspisms;  
SHOW TABLES;
```

These commands allow you to:

- View all available databases.
- Select the **raspisms** database.
- Display all tables in this database.

## Analyzing the Sent Messages Table

To examine the first 10 records in the **sended** table, which contains sent SMS messages, use:

```
SELECT * FROM sended LIMIT 10;
```

### Structure of the **sended** Table:

The **sended** table contains the following columns:

- **id**: Unique message identifier
- **at**: Date and time of sending
- **text**: Message content
- **destination**: Recipient phone number
- **flash**: Flash message indicator (0/1)
- **status**: Message status (unknown, failed, etc.)
- **uid**: Unique message identifier in the system
- **adapter**: Module used for sending (e.g., `adapters\GammuAdapter`)
- **id\_user**: ID of the user who sent the message
- **id\_phone**: ID of the phone used for sending
- **mms**: MMS indicator (0/1)
- **originating\_scheduled**: Reference to a scheduled message

- `created_at`: Timestamp of record creation
- `updated_at`: Timestamp of the last update

## Using This Information in Your Code

To insert new messages into the `sended` table, use the following SQL query:

```
INSERT INTO sended (at, text, destination, flash, status, uid,
adapter, id_user, id_phone, mms, created_at)
VALUES (NOW(), 'Message content', '+33612345678', 0, 'unknown',
'unique_uid', 'adapters
GammuAdapter', 1, 1, 0, NOW());
```

## 2.3 Configuration of Gammu:

### What is Gammu

Gammu is an open-source command-line tool and library for managing mobile phones and GSM modems. It allows sending and receiving SMS messages, handling calls, and managing phonebook entries. Gammu is widely used in IoT and automation projects where SMS notifications are required.

### Installation of Gammu

#### System Update

Before installing Gammu, update your system with the following command:

```
sudo apt update  sudo apt upgrade -y
```

#### Installing Gammu and Gammu-SMSD

Install Gammu and its SMS service manager:

```
sudo apt install gammu gammu-smsd -y
```

#### Checking the GSM Modem

Connect the GSM module and check its serial port:

```
ls /dev/ttyUSB*
ls /dev/ttyS*
```

The modem is usually detected as `/dev/ttyUSB0` or `/dev/ttyS0`.

#### Configuring Gammu

Edit the Gammu configuration file:

```
sudo nano /etc/gammu-smsdrc
```

Add the following configuration:

```
[gammu]
port = /dev/ttyUSB0
connection = at
logfile = /var/log/gammu.log
logformat = textall

[smsd]
service = files
logfile = /var/log/gammu-smsd.log
logformat = textall
runonreceive = /home/pi/sms-handler.sh
```

Save and exit using CTRL+X, then press Y and Enter.

## Testing the Modem Connection

Check if Gammu detects the GSM module:

```
gammu --identify
```

If the modem is correctly configured, you should see output similar to:

```
Device : /dev/ttyUSB0
Manufacturer : SIMCOM
Model : SIM800L
```

## Sending a Test SMS

Try sending an SMS:

```
gammu sendsms TEXT +1234567890 -text "Hello from Raspberry Pi!"
```

Replace +1234567890 with your phone number.

## Enabling the Gammu-SMSD Service (Optional)

If you want Gammu to run in the background for automatic SMS management:

```
sudo systemctl enable gammu-smsd
sudo systemctl start gammu-smsd
```

## Integration of Gammu with RaspiSMS

### Accessing the RaspiSMS Web Interface

To configure RaspiSMS, access its web interface through a browser by entering the following address:

`http://<Raspberry_Pi_IP>/raspisms`

Replace <Raspberry\_Pi\_IP> with the actual IP address of your Raspberry Pi.

### Configuring a Gammu Phone in RaspiSMS

To integrate Gammu with RaspiSMS, follow these steps:

- Go to the settings in the RaspiSMS web interface.
- Add a new phone device.

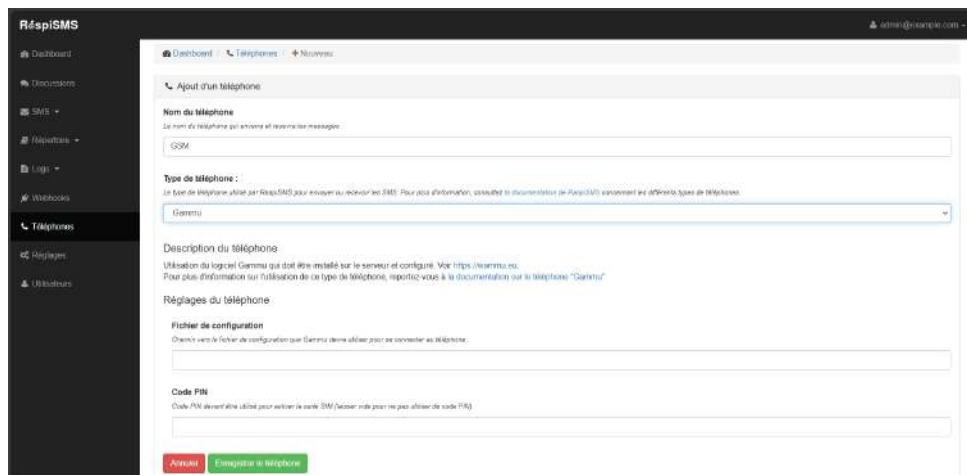


Figure 10: RaspiSMS add a new phone number

- Specify the path to the Gammu configuration file:

`/etc/gammu-smsdrc`

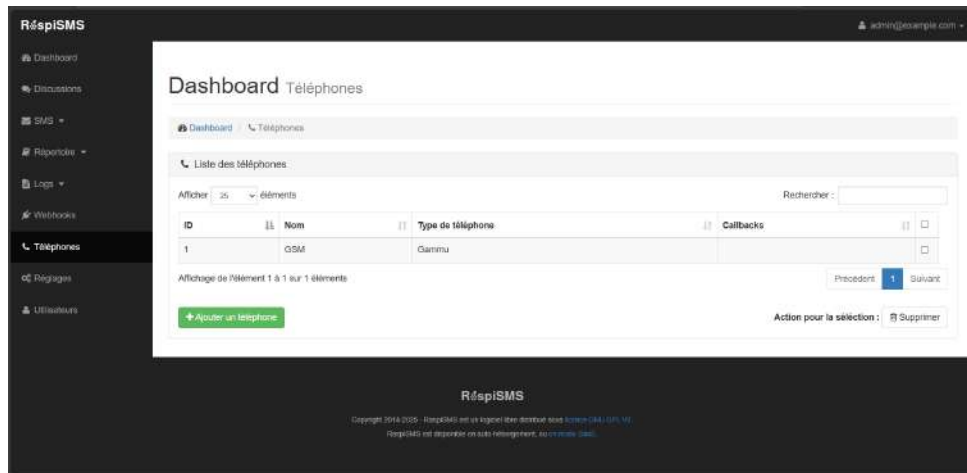


Figure 11: Phone number added successfully

- Save the configuration.

Once configured, RaspSMS will use Gammu to manage SMS sending and receiving.

## 2.4 Development of a Web Application

To facilitate interaction with the system, we developed a web application using Python "Flask" for the backend and HTML/CSS/JS for the frontend. This application allows users to manage SMS communications through an intuitive interface. The application is connected to a database to store message logs and other relevant data.

The source code of the application is available on GitHub:

**Remindini-iot-project**



# C. Option 2

## 1 Hardware

### 1.1 DHT22 Temperature and Humidity Sensor:

The DHT22 is a digital sensor capable of measuring both temperature and humidity with high accuracy. It operates on a 3.3V to 5V power supply and communicates using a single-wire digital protocol. The sensor has a built-in capacitor for signal stability and requires an external 10k $\Omega$  pull-up resistor for proper data communication.

### 1.2 Components:

- Raspberry Pi 4 Model B
- DHT22 Sensor
- Power Supply
- 10k $\Omega$  Resistor

### 1.3 Setup of the circuit:

#### Connections:

- Connect Pin 1 of the DHT22 to Physical Pin 1 (3.3V) on the Raspberry Pi.(Ensure the power supply is stable and sufficient for the entire circuit.)
- Connect Pin 2 of the DHT22 to Physical Pin 7 (GPIO4) on the Raspberry Pi.
- Place a 10k $\Omega$  resistor between Pin 1 and Pin 2 to act as a pull-up resistor for proper data transmission.
- Connect Pin 4 of the DHT22 to Physical Pin 6 (GND) on the Raspberry Pi.

The following image illustrates the complete circuit wiring, showing the connections between the Raspberry Pi, GSM module, and DHT22 sensor:

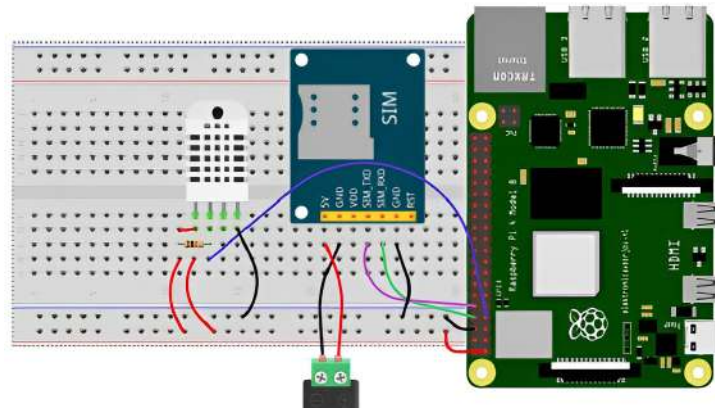


Figure 12: Schematic Representation of the circuit for option 2

## 2 Software

### 2.1 What is Microsoft Azure

Microsoft Azure is a cloud computing platform that provides various services, including IoT solutions. Azure IoT Edge extends cloud intelligence to edge devices, allowing them to process data locally, reducing latency and bandwidth usage while ensuring faster responses and offline capabilities. It enables devices to run AI, analytics, and business logic closer to the data source before sending relevant information to the cloud.

### 2.2 Connecting to Azure IoT Hub:

#### Access Azure Portal

- Log in to the Azure Portal.
- Navigate to IoT Hub and click on Create.

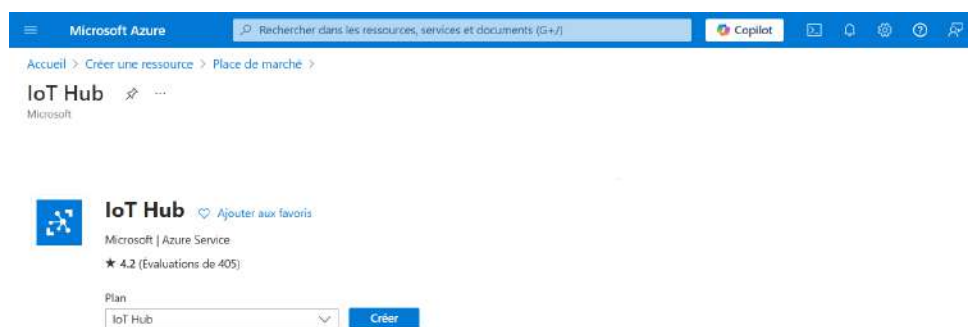


Figure 13: Azure iot create iot hub

- Configure the IoT Hub settings (Subscription, Resource Group, Region, and Pricing Tier).
- Click Review + Create and then Create to deploy the IoT Hub.

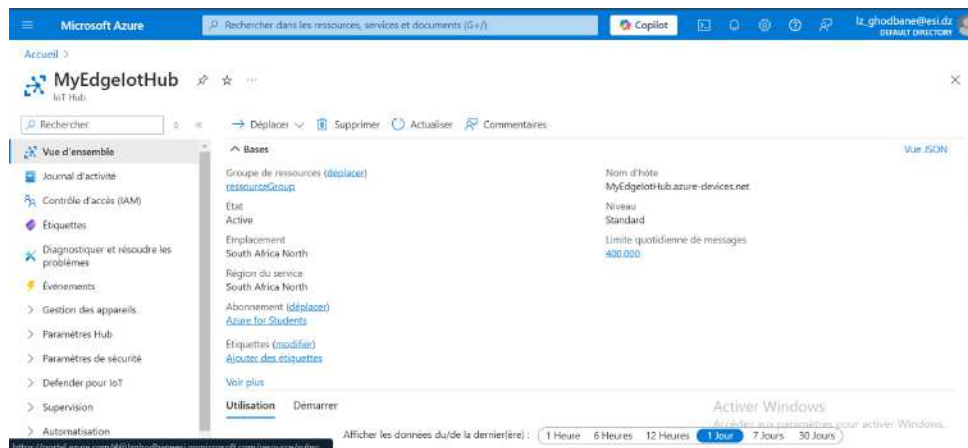


Figure 14: Iot hub created successfully

## Configuring the IoT Device:

- In the IoT Hub, go to IoT Edge.



Figure 15: Azure iot navigate iot edge in iot hub

- Click + New to add a new device.
- Provide a Device ID and select authentication type (e.g., symmetric key).
- Click Save and retrieve the Connection String from the device details.

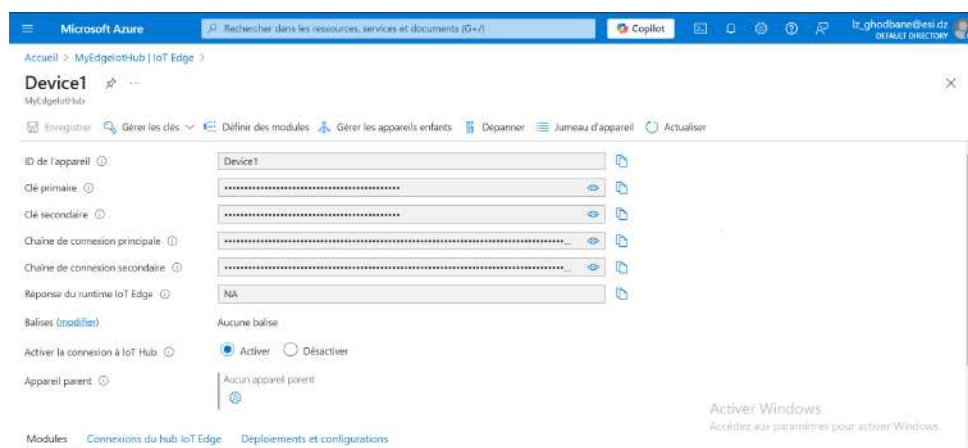


Figure 16: Iot edge created successfully

## 2.3 Storing Data on Azure Blob Storage

### Configuring a Storage Account:

- Navigate to Storage Accounts in the Azure Portal.
- Click + Create, select the subscription, resource group, and configure the storage settings.

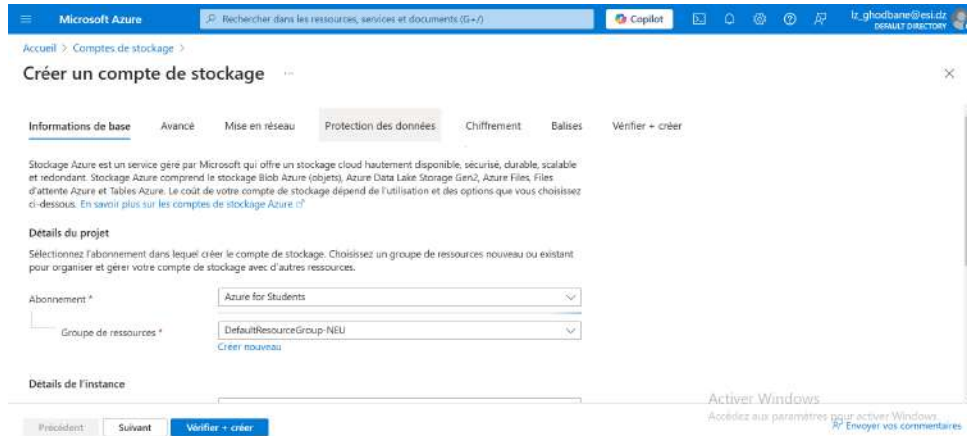


Figure 17: Azure iot create storage account

- Once created, go to Containers and create a Blob container to store IoT data.

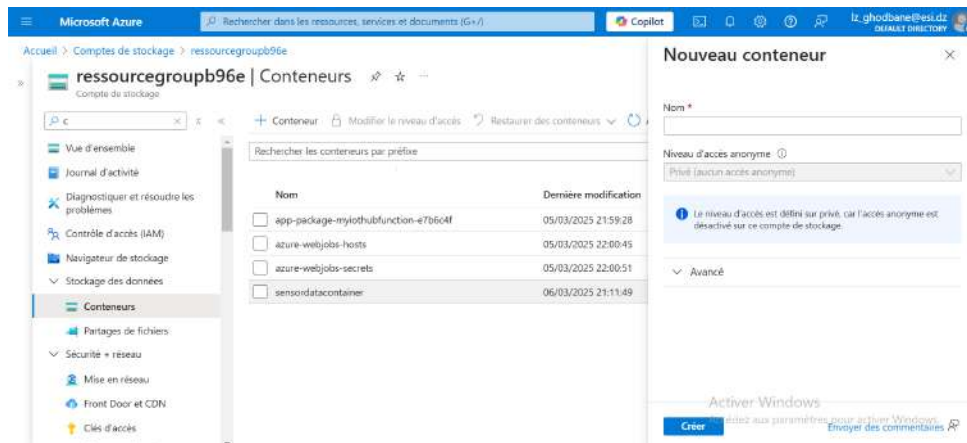


Figure 18: Azure iot create blob container in the storage account

## 2.4 Creating an Azure Stream Analytics

### Create a Stream Analytics Job

- Click + Create and configure Subscription and Resource Group
- Click Review + Create and then Create.

Figure 19: Azure iot create stream analytics

## Configure Inputs and Outputs

- Open the created Stream Analytics job.
- Navigate to Inputs → + Add Stream Input → IoT Hub
- Navigate to Outputs → + Add Output → Blob Storage.

## 2.5 Define a Query

- Go to Query and write a Stream Analytics query.

```
SELECT * INTO [BlobStorage] FROM [IoTHubStream]
```

- Save and test the query.

## 2.6 Start the Job

- Click Start and choose Now or a specific timestamp.
- Monitor the job under the Job Diagram and Monitoring tabs.

Once the Stream Analytics is set up, we integrate it into our IoT system by sending device telemetry data to Azure IoT Hub using the connection string. The Stream Analytics job processes this data in real time, applying filtering or transformations as needed, and stores it in Azure Blob Storage. The stored data can then be retrieved using the Azure Storage SDK for further analysis or visualization. You can find the complete implementation on GitHub.

## D. Final result

### 1 User interface: Remindini

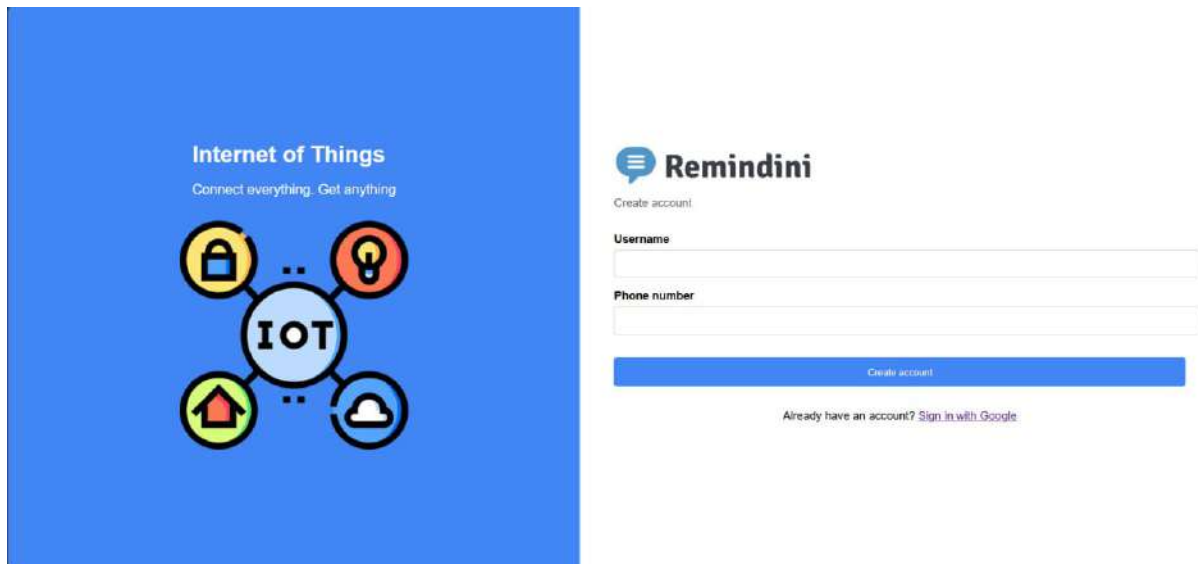


Figure 20: Remindini Sign up page

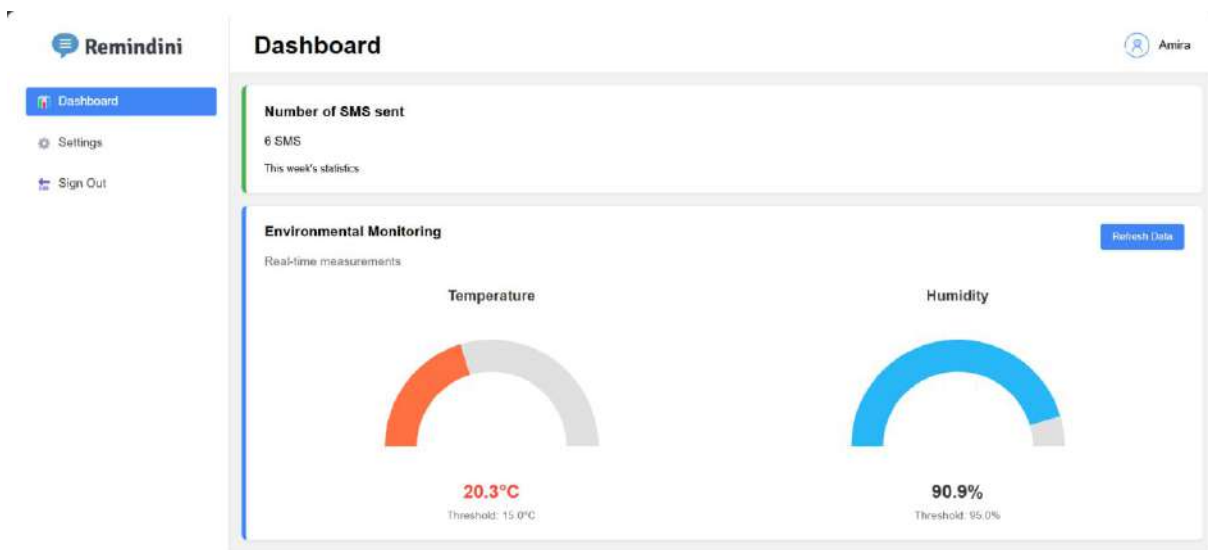


Figure 21: Remindini Dashboard page

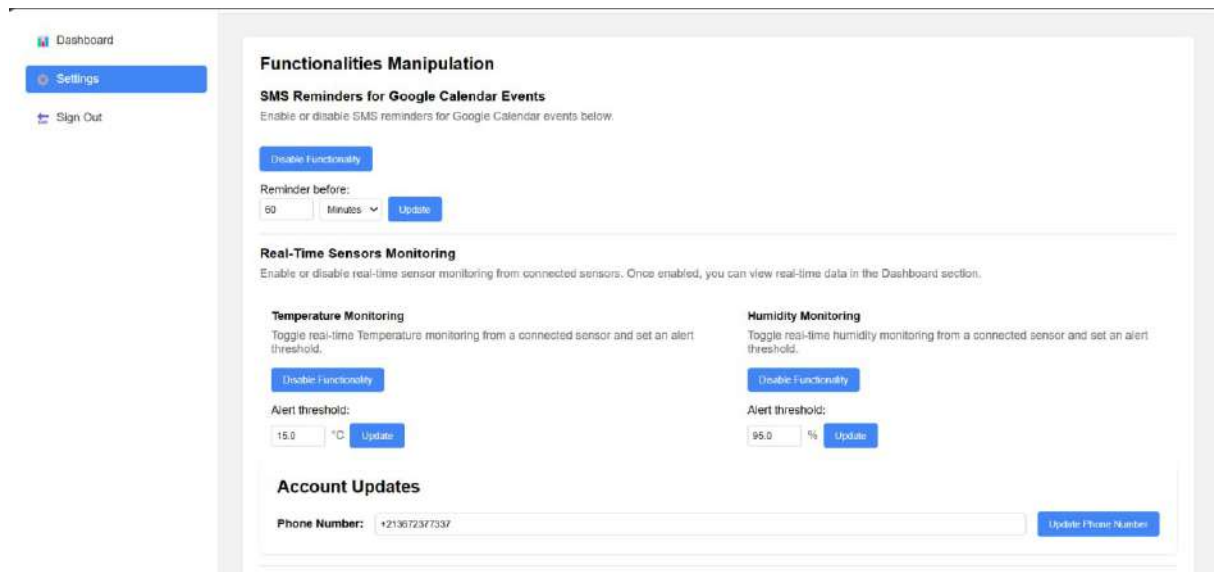


Figure 22: Remindini Settings page

## 2 Admin interface: RaspiSMS

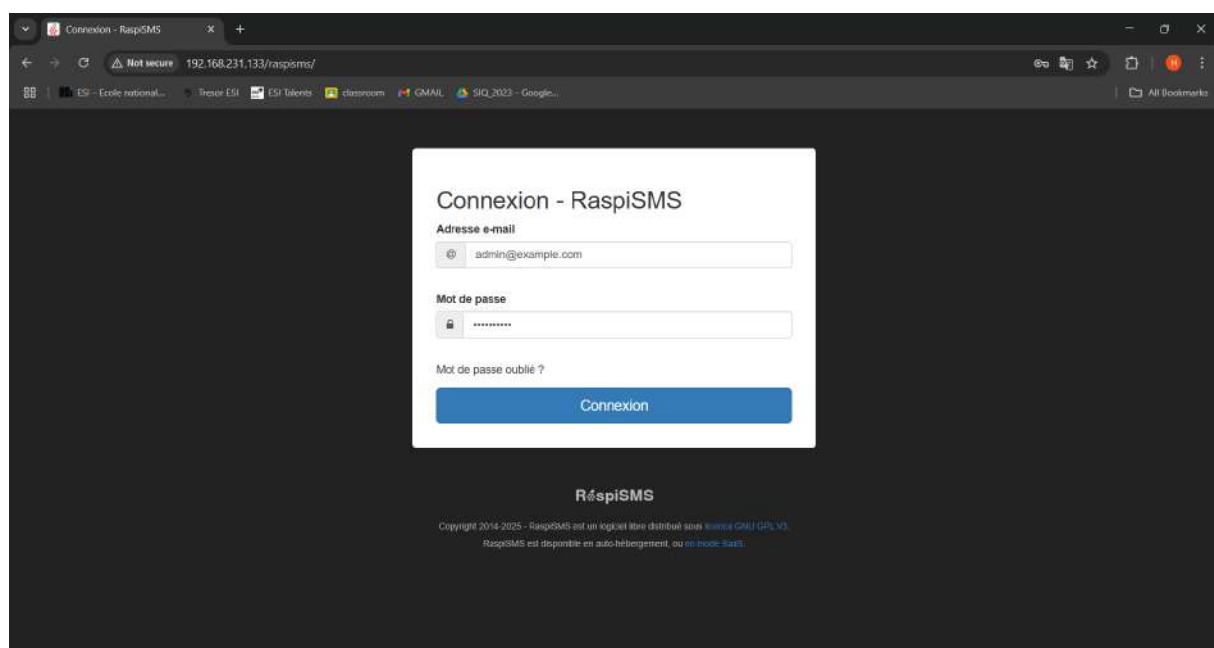


Figure 23: RaspiSMS Sign in Page

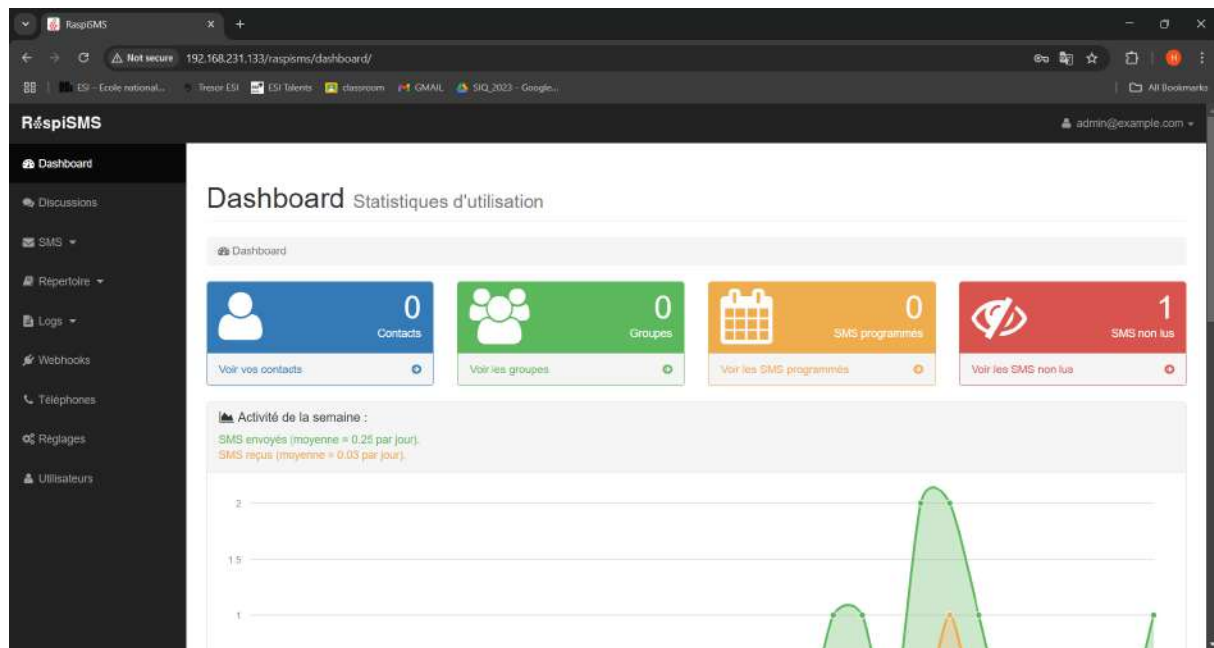


Figure 24: RaspiSMS Dashboard Page(users statistics)

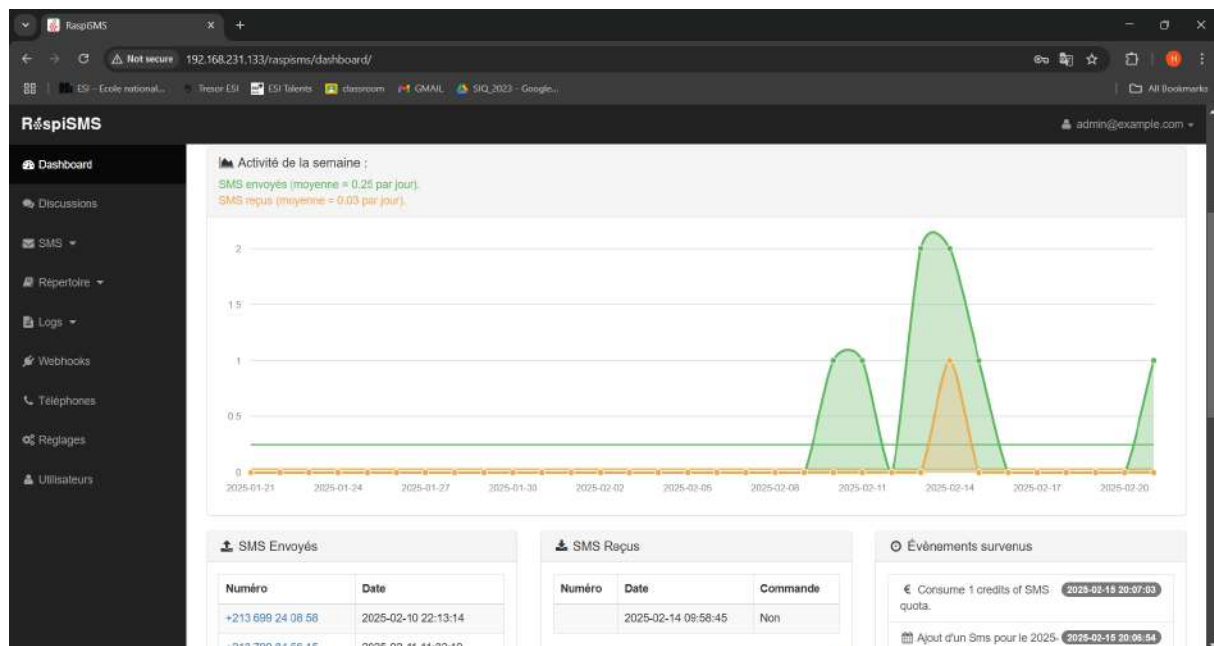
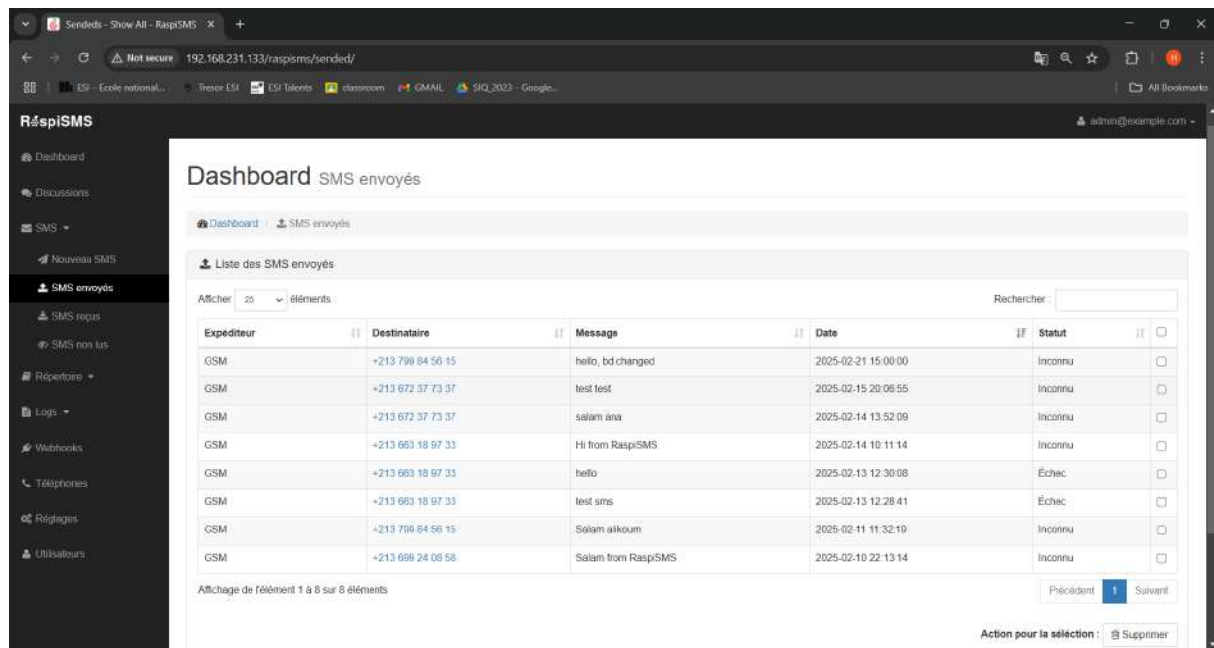


Figure 25: RaspiSMS Dashboard Page(messages statistics)





**RaspiSMS**

Dashboard SMS envoyés

Dashboard SMS envoyés

Liste des SMS envoyés

Montrer 25 éléments

Rechercher :

Expéditeur	Destinataire	Message	Date	IF	Statut	
GSM	+213 798 84 50 15	hello, bd changed	2025-02-21 15:00:00		Inconnu	<input type="checkbox"/>
GSM	+213 672 37 73 37	test test	2025-02-15 20:06:55		Inconnu	<input type="checkbox"/>
GSM	+213 672 37 73 37	salam ana	2025-02-14 13:52:09		Inconnu	<input type="checkbox"/>
GSM	+213 663 18 97 33	Hi from RaspiSMS	2025-02-14 10:11:14		Inconnu	<input type="checkbox"/>
GSM	+213 663 18 97 33	hello	2025-02-13 12:30:08		Echec	<input type="checkbox"/>
GSM	+213 663 18 97 33	test sms	2025-02-13 12:28:41		Echec	<input type="checkbox"/>
GSM	+213 798 84 50 15	Salam aïkhoum	2025-02-11 11:32:19		Inconnu	<input type="checkbox"/>
GSM	+213 698 24 08 58	Salam from RaspiSMS	2025-02-10 22:13:14		Inconnu	<input type="checkbox"/>

Affichage de l'élément 1 à 8 sur 8 éléments

Précédent Suivant

Action pour la sélection : Supprimer

Figure 26: RaspiSMS Sent messages Page

## E. Conclusion

This project developed an intelligent SMS gateway that delivers real-time notifications based on Google Calendar events or sensor data. The system was implemented in two versions:

The first version is a solution that uses Gammu, RaspiSMS and Python to send SMS alerts for Google Calendar events. Whenever a scheduled event is approaching, the system automatically sends a notification, helping users stay on top of their appointments.

The second version is a more advanced implementation that integrates edge computing to process data locally. This approach enables real-time alerts based on sensor readings while reducing dependency on cloud services.

Both versions aim to provide timely and efficient notifications. Future improvements could include support for additional types of sensors, integration with more communication protocols, and the development of predictive alert systems to anticipate important events before they occur.

# F. Bibliography

- Google Calendar API overview
- RaspiSMS
- [sim800l-gsm-gps-raspberry/](#)
- Send Receive SMS Call with SIM800L GSM Module Arduino
- SIM800L restart automatically
- Azure iot
- Raspberry PI GPIO Pinout
- Gammu et Wammu
- RaspiSMS documentation
- RaspiSMS github project
- Sim800L v2
- Binarytech Electronique Algérie
- DHT22 sensor pinout
- gauges in frontend