

SCI Project Report  
2nd Year Higher Cycle (2CS)  
Option : Computer Systems (SQ)

NotifyTrack - Smart IoT Notification  
Gateway for ESI

*Directed by :*

- BOUKHETALA Zaineb
- KHELLAS Yacine
- REMIL MahaFatimaZohraa
- KHADIR Amina
- HENNANE DouaaElIkhlas
- BENYAHIA Yahia

*Supervised by :*

- Mr. SEHAD Abdenmour

# Table des matières

Table des matières	I
Table des figures	III
List of abbreviations	IV
General introduction	1
<b>1 Project Overview</b>	<b>2</b>
1.1 Introduction . . . . .	2
1.2 Problem Statement . . . . .	2
1.3 Proposed Solution . . . . .	3
1.4 Conclusion . . . . .	4
<b>2 Conception</b>	<b>5</b>
2.1 Introduction . . . . .	5
2.2 System Architecture . . . . .	5
2.3 System Functionalities . . . . .	6
2.3.1 User Management and Event Scheduling . . . . .	6
2.3.2 IoT Based Enviromenntal Monitoring . . . . .	7
2.4 Technologies Used . . . . .	7
2.5 Conclusion . . . . .	9
<b>3 Team Organization</b>	<b>10</b>
3.1 Introduction . . . . .	10
3.2 Management & Communication Tools . . . . .	10
3.3 Task Breakdown . . . . .	11
3.4 Conclusion . . . . .	12

<b>4</b>	<b>Implementation Details</b>	<b>13</b>
4.1	Introduction . . . . .	13
4.2	Hardware Setup and GSM Module Integration . . . . .	13
4.2.1	Hardware and devices . . . . .	13
4.2.2	Setting Up the Raspberry Pi . . . . .	14
4.2.3	GSM Module Wiring and Testing . . . . .	16
4.3	RaspiSMS Setup and Testing . . . . .	18
4.4	Web Interface Development . . . . .	19
4.5	Integration of Web Application with RaspiSMS . . . . .	19
4.6	Python Script Development and Web App Integration . . . . .	19
4.6.1	OAuth 2.0 Authentication (Google) . . . . .	20
4.6.2	Accessing Calendars . . . . .	20
4.6.3	Event Filtering . . . . .	20
4.6.4	Data Structuring and Saving . . . . .	21
4.6.5	Web Access via Flask Interface . . . . .	21
4.6.6	Hosting on PythonAnywhere . . . . .	21
4.7	Sensor Integration and Node-RED Workflow . . . . .	21
4.7.1	Configuring DHT22 Sensor in Node-RED : . . . . .	22
4.7.2	Real-Time Dashboard Visualization : . . . . .	23
4.7.3	Sending Data to Azure IoT Hub : . . . . .	24
4.8	Azure Cloud Platform Exploration . . . . .	24
4.8.1	Azure IoT Hub . . . . .	24
4.8.2	Azure Stream Analytics . . . . .	26
4.8.3	Azure Functions . . . . .	27
4.9	Conclusion . . . . .	27
	<b>General conclusion</b>	<b>29</b>
	<b>Références</b>	<b>30</b>

# Table des figures

2.1	NotifyTrack System Architecture . . . . .	6
3.1	A Glimpse of the Tasks Organization . . . . .	12
4.1	Raspberry Pi Board . . . . .	13
4.2	GSM Module SIM800L . . . . .	14
4.3	DHT22 Sensor . . . . .	14
4.4	Circuit Wiring Between GSM Module and Raspberry Pi . . . . .	17
4.5	Raspisms Interface . . . . .	18
4.6	Circuit Wiring Between DHT22 sensor and Raspberry Pi . . . . .	22
4.7	Node-RED flow for Processing Sensor Data . . . . .	23
4.8	Node-RED Dashboard : Sensor data outputs . . . . .	23
4.9	IoT Hub Creation . . . . .	25
4.10	Incoming Sensor Data from the Raspberry . . . . .	26
4.11	ASA Job Created . . . . .	27

# List of abbreviations

**ESI** Ecole Nationale Supérieure de l'Informatique

**IoT** Internet of Things

**LAN** Local Area Network

**MQTT** Message Queuing Telemetry Transport

Ecole Nationale Supérieure de l'Informatique (ESI), Internet of Things (IoT),  
Message Queuing Telemetry Transport (MQTT), Local Area Network (LAN)

# General introduction

In today's fast-paced educational environment, timely communication is essential. University and school settings require that **important events** such as exam dates, deadlines, meetings, and emergency notifications are **communicated immediately** to students and staff. However, **bad internet network** access can severely hinder the effectiveness of traditional communication systems. A smart SMS notification system ensures that even when online connectivity is unreliable, important information is still delivered directly to users' mobile devices, thereby maintaining the continuity and effectiveness of academic and administrative operations.

The primary objective of this project is to provide real-time notifications via SMS for critical events for our school ESI. We aim also to integrate **an intuitive admin dashboard** that enables administrators to monitor notification statuses, manage user settings, and configure system parameters seamlessly, and for users to customize the SMS messages they receive. By addressing these needs, the project seeks to enhance **communication efficiency and reliability** within the educational institution.

This report covers the development of a comprehensive SMS notification system that leverages IoT devices and containerized software components. It includes the design and implementation of every step along the way.

# Chapitre 1

## Project Overview

### 1.1 Introduction

In this chapter, we will outline the challenges that highlight the necessity of implementing such a system in our school environment. We will begin by identifying the key problems that motivated this project, including inefficiencies in existing processes, gaps in communication, or the lack of an automated solution.

Then we will talk briefly about the nature of the solution, before detailing the system architecture and the technical aspects in the next chapters.

### 1.2 Problem Statement

In our environment (oued semar, university residences and in a lot of other places), unstable internet connectivity is a **frequent challenge** that can delay or even prevent the timely communication of critical information, which can have serious consequences, particularly in situations that require **immediate action**.

In an academic setting, various operations rely on **instant notifications**. Some of the key areas where the lack of an effective communication system presents challenges include :

- **Exam Schedules and Assignment Deadlines** : Students and faculty need timely reminders about upcoming exams, assignment due dates, and schedule changes. An unreliable notification system can lead to missed deadlines and confusion.
- **Emergency Alerts** : Critical events, such as security threats, fire alarms, or health-related emergencies, demand **instant notifications** to ensure appropriate responses from staff and students.

- **Environmental Monitoring** : Certain facilities within the school, such as the **data center**, require continuous monitoring. For instance, a malfunction in the air conditioning system could cause the temperature to rise beyond safe limits, potentially damaging hardware. Immediate alerts must be sent to responsible personnel to prevent costly failures.
- **Food and Safety Compliance** : Environmental sensors in the **school restaurant** need to track parameters like **temperature, humidity, and gas leaks**. Any anomaly must trigger an instant alert to prevent food spoilage and ensure safety compliance.
- **Classroom and Laboratory Conditions** : Monitoring air quality, classroom occupancy, and laboratory safety conditions can improve comfort and security for students and staff.
- **Transport Updates** : Many students rely on university-provided transportation. Real-time notifications regarding **bus (kous) delays or cancellations** would greatly enhance convenience and reduce uncertainty.

The absence of an integrated and **reliable notification system** creates inefficiencies, safety risks, and communication gaps that negatively impact both academic performance and campus operations.

### 1.3 Proposed Solution

To mitigate these challenges, we propose the development of a **Smart Notification Gateway System** that provides **real-time SMS alerts** for both **personal and institutional** events.

The system is designed to serve students, faculty, and administrative staff by integrating multiple data sources to ensure **timely and actionable notifications**. The key features of the solution include :

- **Google Calendar Integration** : Users can link their accounts to receive SMS reminders for personal and professional appointments.
- **Automated Facility Monitoring** : IoT sensors will be deployed in **critical areas**, such as the data center, school restaurant, and classrooms, to monitor environmental conditions. If abnormal readings are detected, immediate alerts will be sent.
- **Emergency Alert System** : The system will provide an automated mechanism to send **urgent notifications** in case of security threats, power failures, or medical emergencies.



- **Transportation Notification Service** : Students will receive real-time **bus (kous) updates**, ensuring that they are informed about any schedule changes or delays.

By implementing this system, we aim to **enhance safety, efficiency, and communication** within the institution. The use of SMS ensures that notifications are delivered even in cases of **poor internet connectivity**, making the system highly **reliable and accessible**.

## 1.4 Conclusion

In this chapter, we have outlined the motivation behind our project by identifying key challenges in communication and operational efficiency within our academic environment.

To address these challenges, we proposed a **Smart Notification Gateway System** that integrates SMS-based alerts, Google Calendar synchronization, and IoT-enabled facility monitoring.

This solution is designed to provide **real-time, reliable, and actionable notifications** to students, faculty, and administrative staff, ensuring improved **safety, efficiency, and responsiveness** in various scenarios.

The following chapters will dive deeper into the **technical architecture, implementation details, and system functionalities**, demonstrating how our solution effectively addresses the identified challenges and enhances communication within the institution.

## Chapitre 2

# Conception

### 2.1 Introduction

After having stating the problem in the last chapter, we will present our detailed proposed solution, how it addresses the stated issues through innovative features and technologies. This discussion will include an overview of the system’s architecture, its core functionalities, and the expected impact on improving efficiency, accessibility, and user experience.

Furthermore, we will explore the technical aspects, including the tools, frameworks, and methodologies adopted to develop the solution. Finally, we will provide insights into the project’s scope, objectives, and anticipated challenges, setting the stage for a deeper dive into its implementation in the subsequent chapters.

### 2.2 System Architecture

NotifyTrack is an IoT-driven smart notification system that leverages **edge computing** and **cloud services** to automate SMS alerts for both **event scheduling** and **environmental monitoring**. At its core, a Raspberry Pi 4 acts as the **central controller**, orchestrating data processing, SMS transmission, and IoT communications.

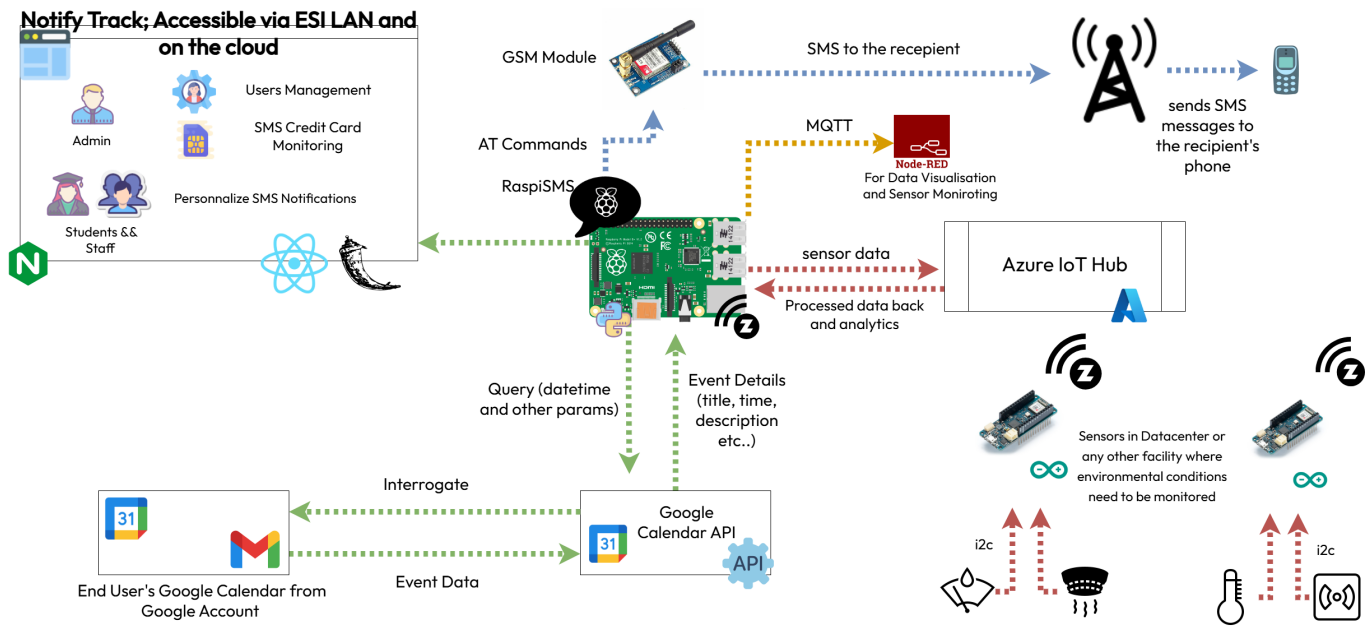


FIGURE 2.1 – NotifyTrack System Architecture

The system is modular and consists of 2 primary functions :

**User Management & Event Scheduling** – It ensures students, professors and staff receive timely reminders and urgent notifications.

**IoT-Based Environmental Monitoring** - Provide real-time safety monitoring through sensor-based alerts.

By integrating a bunch of technologies that will be stated later, NotifyTrack ensures a seamless, automated communication pipeline that is both **scalable** and **reliable**.

## 2.3 System Functionalities

### 2.3.1 User Management and Event Scheduling

The event scheduling module ensures that students and faculty are **always informed about important events**.

Each **user authorizes** NotifyTrack via OAuth, to grant limited access to their Google Calendar, our system then **queries event details**, such as lecture times, exams, and meetings.

In parallel, a relational database (both local and cloud-based) stores user contact details, by the time the queries responses reach, the system performs an efficient join operation between event data and user contact details **before**

### sending notifications.

The event data are then processed and queued in **RaspiSMS database** (on scheduled). Then, RaspiSMS, built on Gammu, executes **AT commands** to control the SIM800L GSM module.

The message is then dispatched to the recipient at the scheduled time (this being entirely handled by the GSM module)

Through a user-friendly web interface, individuals can **customize their notification settings** for example : Early reminders (e.g., an hour before an event) or daily summaries (morning digest of scheduled events), or urgent alerts (last-minute reminders or emergency notices).

We have also an admin panel for administration purposes and **monitoring the SMS card on the GSM module's state** .

### 2.3.2 IoT Based Enviromenntal Monitoring

The environmental monitoring module ensures that **safety conditions** in key areas (e.g., data centers, labs, and classrooms) are **actively monitored**.

A network of low-cost **Arduino-based sensors** is deployed in certain places to measure temperature, humidity, air quality, and other environmental factors.

These arduinos communicate sensors readings to the Raspberry pi via Zigbee, an energy-efficient wireless protocol ideal for low-power IoT networks.

Node-RED, a flow-based programming tool, processes sensor readings which are organized in MQTT topics. A cron job runs periodic checks on sensor values, if a threshold is exceeded (e.g., overheating in the server room), an alert SMS is sent via RaspiSMS to the appropriate staff (e.g., IT personnel for data center overheating).

Sensor data is stored in **Azure IoT Hub**, for **advanced analytics**, historical analysis. It provides scalability, and helps the system handle large-scale deployments without performance issues. (if we implemented this part i'll add more details)

## 2.4 Technologies Used



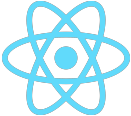






Icon	Technology
	<b>Zigbee</b> : For low-cost, low-power sensing capabilities, transmitting environmental data over a reliable wireless protocol.
	<b>MQTT</b> : Ensures low-bandwidth, real-time data streaming for sensor readings, ideal for IoT environments.
	<b>React.js (Frontend UI)</b> : A modern component-based framework, to build a dynamic and responsive user experience.
	<b>Flask (Backend API)</b> : Chosen for its lightweight and scalable nature, ensuring efficient API handling.
	<b>Google Calendar API</b> : Provides seamless event synchronization, by integrating with OAuth.
	<b>Nginx</b> : A high-performance web server and reverse proxy used for load balancing and secure API gateway management.
	<b>Node-RED</b> : A visual programming tool that simplifies IoT workflow automation, allowing quick integration of sensors and event-driven alerts.
	<b>Azure IoT Hub</b> : Provides scalable cloud storage and analytics, enables predictive insights for environmental monitoring.
	<b>RaspiSMS</b> : An open-source SMS gateway built in on top of gammu for sending and receiving messages via a Raspberry Pi, useful for automated notifications.

TABLE 2.1 – System Technologies and Justifications

## 2.5 Conclusion

In this chapter, we explored the key technologies that form the backbone of our system, each selected based on its efficiency, reliability, and compatibility with the project's requirements.

By deciding a coherent architecture and leveraging such technologies, our system achieves a robust, responsive, and efficient infrastructure, and help us provide for reliable event scheduling and environmental monitoring, and sufficient management, thus satisfying all the stakeholders.






## Chapitre 3

# Team Organization

### 3.1 Introduction

In any collaborative project, **effective communication and task management** are essential to ensure good coordination between team members and successful achievement of objectives. For ours, we've selected a range of tools to meet the specific needs of planning, collaboration and real-time tracking.

### 3.2 Management & Communication Tools

Icon	Tool
	<b>Discord</b> : Used for real-time communication and collaboration among team members, supporting text, voice, and video chat.
	<b>Google Drive</b> : Provides cloud storage for project files, allowing easy access and collaboration.
	<b>Google Sheets</b> : Enables data organization, collaboration, and real-time updates for project tracking and analysis.
	<b>Tailscale</b> : A secure VPN that simplifies remote access to internal services without exposing them to the internet.
	<b>SSH</b> : Secure Shell protocol for remote server administration, ensuring encrypted access.



**Real VNC** : A remote desktop protocol that allows graphical access to remote machines for system administration.

TABLE 3.1: Management & communication tools

### 3.3 Task Breakdown

The project tasks were divided among the six team members, covering hardware setup, software development, and system integration.

— **System Architecture Design and brainstorming functionalities**

— **Google Calendar Integration :**

1. Develop a Python script on the Raspberry Pi for OAuth2 authentication so the users authorize us to access their calendar.
2. Implement queries to retrieve event data from Google Calendar API.

— **GSM Module Setup and Testing**

— **RaspiSMS Configuration :**

1. Parse incoming event details and preprocess data for RaspiSMS.
2. RaspiSMS converts parsed data into AT commands for SMS transmission.

— **SMS Notification System** : Integrate GSM module with RaspiSMS to send event-based notifications.

— **Web Dashboard Development :**

1. Provides controls for updating user settings and managing notification schedules.
2. Users can modify their SMS preferences

— **Sensor Integration** : Connect temperature sensors to the Raspberry Pi and implement data processing scripts.

— **Admin Dashboard Sensor Visualization** : Utilize Node-RED for graphical representation of sensor readings.

— **Temperature Monitoring and Alert System :**

1. Monitor temperature in the data center.



2. Send SMS alerts to personnel when temperature exceeds predefined thresholds.

— **Azure IoT Hub Integration :**

1. Configure Azure IoT Hub.
2. Install and deploy IoT Edge runtime on the Raspberry Pi.
3. Explore IoT Hub and Analytics for efficient sensor data processing

Tr	Task	State	Phase	Deadline	Tr	Notes & Remarks	Tr	Task Responsible
	Raspberry Pi configuration steps	Done	Organizing	13/02/2025		1-install raspbian OS on the rpi 2-configure the rpi's network access so we don't have to connect to a monitor each time we use it (just by a hotspot) 3-install python and necessary libraries like google agenda (google-api-python-client, requests); RaspSMS, nodered etc..		KHELLAS YACINE
	Draw a coherent and well structured architecture to demonstrate all system components	Done	Organizing	dd/mm/yyyy		- temperature sensor in datacenter side		BOUKHETALA ZAINEB
	prepare python script to interact with google calendar and retrieve data for the SMS	Done	Programming	13/02/2025		A Python script on the Raspberry Pi for OAuth2 authentication and queries the Google Calendar API. + Unit Tests (see how google user would allow a third party check its google calendar		REMIL MAHAFAATIMAZOHR
	Assembling hardware and testing the GSM module	Done	Hardware	11/02/2025		Notes & Remarks		KHELLAS YACINE
	setting up raspisms , preprocess data to RaspSMS	Done	Configuration	11/02/2025		Incoming event details are parsed and sent to RaspSMS. RaspSMS converts this into AT commands for the GSM module, which then sends the SMS to the user.		BOUKHETALA ZAINEB
	send the SMS (AT Commands from RaspSMS )(Integrating GSM module RaspSMS)	Done	Integration	15/02/2025		Notes & Remarks		BOUKHETALA ZAINEB

FIGURE 3.1 – A Glimpse of the Tasks Organization

This structured breakdown ensured efficient collaboration and progress toward project completion.

### 3.4 Conclusion

The selection of appropriate communication and management tools and organizing tasks in a coherent manner depending on the timeline of the project, is an essential step for ensuring smooth collaboration, efficient task tracking, and streamlined project execution. By integrating real-time communication, structured planning, and organized documentation, our team remains coordinated and productive.

# Chapitre 4

## Implementation Details

### 4.1 Introduction

The implementation phase followed a structured approach, starting with hardware setup and gradually integrating software components. The process involved configuring the GSM module, setting up RaspiSMS for SMS notifications, developing the web interface, and integrating sensor data.

### 4.2 Hardware Setup and GSM Module Integration

#### 4.2.1 Hardware and devices

##### Raspberry Pi 4 model B :

The Raspberry Pi 4 Model B is the central processing unit of the project. It features a quad-core ARM Cortex-A72 CPU running at 1.5 GHz, providing sufficient processing power for data collection and communication tasks. The board includes a 40-pin GPIO header, which supports protocols like I2C, SPI, and UART, enabling seamless integration with sensors and peripherals.



FIGURE 4.1 – Raspberry Pi Board

##### GSM Module SIM800L :

The SIM800L is a compact GSM module used for SMS and voice communication over the 2G network. It supports quad-band frequencies (850/900/1800/1900 MHz), ensuring compatibility with most GSM networks worldwide. The module operates on 3.4V to 4.4V, making it energy-efficient, and can be controlled using AT commands.



FIGURE 4.2 – GSM Module SIM800L

### DHT22 Sensor :

The DHT22 is a digital sensor for measuring temperature and humidity. It has a temperature range of  $-40^{\circ}\text{C}$  to  $80^{\circ}\text{C}$  with an accuracy of  $\pm 0.5^{\circ}\text{C}$  and a humidity range of 0% to 100% with an accuracy of  $\pm 2\%$ . The sensor provides data via a single-wire serial interface, making it easy to integrate with the Raspberry Pi.



FIGURE 4.3 – DHT22 Sensor

## 4.2.2 Setting Up the Raspberry Pi

### Operating System Installation

To initialize the Raspberry Pi, the following steps were performed :

1. Write the operating system image to an SD card using the Raspberry Pi OS Imager tool.
2. Insert the SD card into the Raspberry Pi and connect peripherals such as a keyboard, mouse, and monitor via USB and HDMI ports.
3. Power on the device and use the initial configuration wizard to set up Wi-Fi and other basic settings.
4. Install required libraries and software packages, such as `gpiozero` for GPIO control and `RPi.GPIO` for PWM control.

### Advanced Features Configuration

To enable remote access and communication capabilities, the following configurations were applied :

1. Enable the SSH interface to allow remote command-line access to the Raspberry Pi.
2. Enable the VNC interface to provide graphical user interface (GUI) access via RealVNC.
3. Enable the serial port to facilitate communication with external devices such as the GSM module and the DHT22 sensor.
4. Reboot the Raspberry Pi to save changes.

### Node-RED Configuration

Node-RED was set up to visualize sensor data and automate workflows. The setup process included the following steps :

1. Install Node.js and npm packages using the command :  
`sudo apt install nodejs npm.`
2. Install Node-RED globally using npm :  
`sudo npm install -g node-red.`
3. Install nodes for DHT22 and Azure IoT Hub :  
`cd /.node-red`  
`sudo npm install -g node-dht-sensor`  
`sudo npm install -g node-red-contrib-azure-iot-hub`  
`node-red-restart`
4. Start Node-RED using the command : `node-red start.`
5. Access the Node-RED interface through a web browser at `http ://localhost :1880.`
6. Install additional libraries, such as **node-red-dashboard**, from the Node-RED palette manager to enable visualization features.

### Node-RED Startup Automation

To ensure Node-RED starts automatically upon system boot, the following steps were implemented :

1. Install the PM2 process manager using the command :  
`sudo npm install -g pm2.`
2. Start Node-RED using PM2 :  
`pm2 start /usr/local/bin/node-red - -v.`

3. Generate a startup configuration file using :  
`pm2 save && pm2 startup.`

### Tailscale Configuration

Tailscale is a secure, zero-configuration virtual private network (VPN) platform that enables seamless and encrypted communication between devices over the internet. It uses the WireGuard protocol to create a private network, allowing devices to connect as if they were on the same local network, even when they are geographically dispersed.

In this project, Tailscale was configured on the Raspberry Pi to provide secure remote access for team members. The setup process included the following steps :

1. Install Tailscale on the Raspberry Pi and register the device to a Tailscale account.
2. Assign a public IP address to the Raspberry Pi, allowing secure access from external networks.
3. Share the device with other team members' Tailscale accounts, enabling them to access the Raspberry Pi remotely.

### 4.2.3 GSM Module Wiring and Testing

#### Circuit Wiring Between GSM Module and Raspberry Pi

The GSM module SIM800L was connected to the Raspberry Pi using a combination of GPIO pins and external power. Since the GSM module requires a higher current than the Raspberry Pi can provide, an external power source was used to ensure stable operation. The wiring was configured as follows :

1. Connect the **TX pin** of the GSM module to the **RX pin** (GPIO 15) of the Raspberry Pi for receiving data.
2. Connect the **RX pin** of the GSM module to the **TX pin** (GPIO 14) of the Raspberry Pi for transmitting data.
3. Connect the **GND pin** of the GSM module to the **GND pin** of the Raspberry Pi and the external power supply to establish a common ground.
4. Connect the **VCC pin** of the GSM module to an external **5V power supply** capable of delivering at least 2A to meet the power requirements of the GSM module.

A prototype circuit was designed using Fritzing to visualize the physical connections.

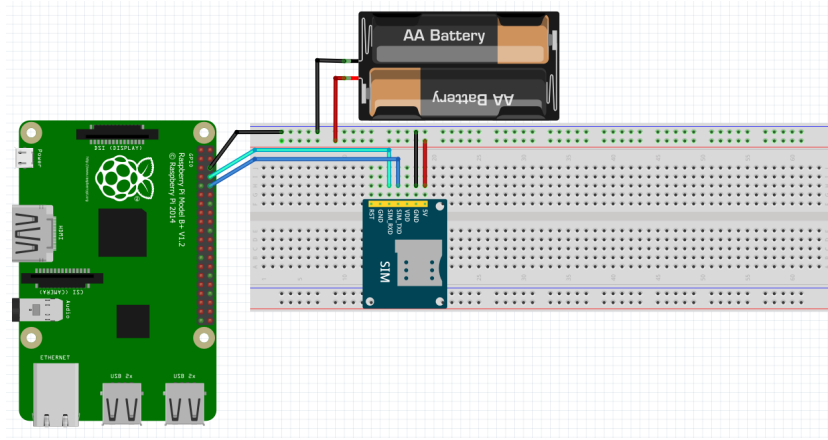


FIGURE 4.4 – Circuit Wiring Between GSM Module and Raspberry Pi

### GSM Environment Preparation

To enable communication with the GSM module, the Raspberry Pi's environment was prepared by installing and configuring the necessary software tools. The following steps were performed :

1. Install Minicom and gammu using the command :  
`sudo apt install minicom gammu.`
2. Configure Minicom and set the serial port parameters (e.g., baud rate, data bits, parity) to match the requirements of the GSM module `sudo minicom -s.`
3. Configure Gammu by creating a configuration file (`.gammurc`) with the GSM module's settings :

```
[gammu]
port = /dev/ttyserial0
connection = at
```

### Testing the GSM Module

To verify the functionality of the GSM module, AT commands were sent via the Raspberry Pi using the Minicom terminal. The following steps were performed :

1. Open the Minicom terminal with the command :  
`sudo minicom -D /dev/ttyserial0 -b 9600.`
2. Send the **AT** command to check if the GSM module responds with **OK**, indicating a successful connection.

3. Send the **AT+CSQ** command to check the signal strength, which should return a value between 0 and 31 (higher values indicate better signal quality).
4. Send the **AT+CMGF=1** command to set the SMS mode to text format.
5. Send the **AT+CMGS="<phone number>"** command to compose an SMS, followed by the message text and the **Ctrl+Z** key combination to send the message.

Additionally, Gammu commands were used to test the GSM module :

```
gammu sendsms TEXT <phone number> -text "Test Message".
```

### 4.3 RaspiSMS Setup and Testing

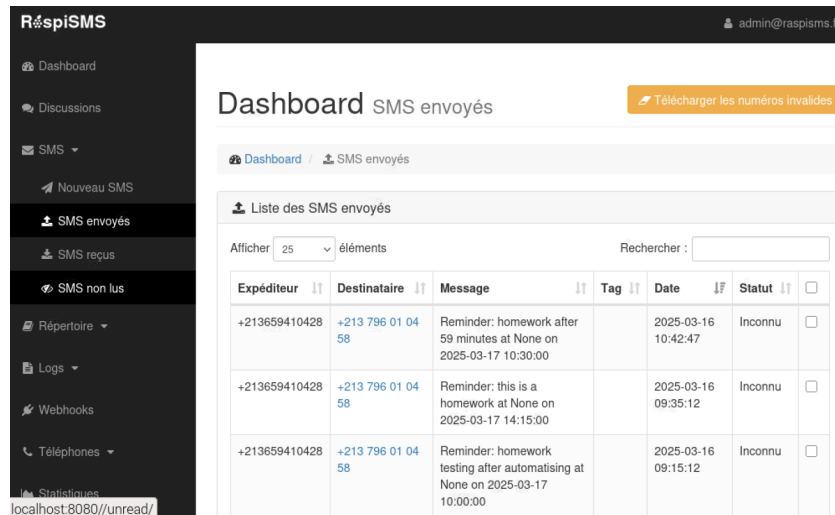


FIGURE 4.5 – Raspisms Interface

To manage SMS notifications efficiently, RaspiSMS was deployed inside a Docker container after having trouble installing from documentation due to conflicting packages versions in the OS, the github repo can be found here. The setup included :

1. Installing Docker and setting up the RaspiSMS container with docker-compose.
2. Testing SMS transmission via Gammu to ensure proper functionality, inside the container (by mapping the physical serial port `ttyS0` to the port "modem" in the container).

```
sudo docker exec -it raspisms gammu sendsms TEXT phonenummer  
-text "Test SMS"
```

3. The raspisms interface can be accessed at **http://localhost:8000/raspisms**

## 4.4 Web Interface Development

While the web application interface was being developed to give users control over their notification preferences, efforts were also made to ensure seamless deployment both on ESI's local network and in the cloud. The frontend, built with React.js, provides an intuitive user experience, while the backend, powered by Flask, handles efficiently API requests for scheduling events and managing notifications.

Currently, the system is accessible through a virtual LAN, allowing for controlled testing and internal use. However, once fully deployed within ESI, it will run on our school's LAN, to ensure reliable, low-latency access for users on campus. To facilitate this, Nginx serves as a reverse proxy, optimizing performance, load balancing, and securing API endpoints. This setup ensures smooth communication between the frontend and backend while making the system scalable and resilient to high traffic loads.

## 4.5 Integration of Web Application with RaspiSMS

To automate SMS notifications, the web application was integrated with **RaspiSMS**, to ensure timely reminders for scheduled events. The integration process involved : developing a **scheduler service** running as a background job under **systemd** to fetch events from Google Calendar, that will query for events scheduled, to ensure only upcoming events are processed, and will send a **POST request** to the **RaspiSMS API** to insert the sms details into the **scheduled** table in MariaDB : RaspiSMS database, which will allow RaspiSMS to automatically send the message at the designated time.

## 4.6 Python Script Development and Web App Integration

To automate the retrieval of important events (exams, meetings, presentations, etc.) from users' Google Calendars, a Python script was developed. It



uses the **Google Calendar API** and is built on **Flask**. The script also incorporates **Google OAuth 2.0 authentication** to securely access calendar data.

#### 4.6.1 OAuth 2.0 Authentication (Google)

Accessing calendars requires explicit user authorization. The script uses Google's OAuth 2.0 protocol :

1. On first use, the user is redirected to a Google login and consent page ;
2. Once access is granted, a secure access token is generated and stored locally (in a `token.json` file) ;
3. This token is reused for future requests to the Google Calendar API without requiring the user to log in again.

This mechanism ensures both user privacy and compliance with Google's security standards.

#### 4.6.2 Accessing Calendars

After authentication, the script :

1. Retrieves the list of calendars linked to the user's Google account ;
2. Iterates through each calendar to extract events scheduled for a specific date (by default, the day after the script is run).

#### 4.6.3 Event Filtering

The events are filtered using a **predefined list of keywords** (such as *exam*, *meeting*, *presentation*, *test*, *control*, etc.) :

1. Filtering is applied to both the **event title** and **description** ;
2. Only events deemed relevant are kept ;
3. Duplicate events are removed, especially if they appear across multiple calendars.

#### 4.6.4 Data Structuring and Saving

Filtered events are cleaned and converted into a structured format (a Python dictionary) containing only key information :

1. Event title ;
2. Start and end date/time ;
3. Location (if available) ;
4. Description.

The data is then saved to a JSON file, organized hierarchically by user and date.

#### 4.6.5 Web Access via Flask Interface

The script includes a Flask-based HTTP interface :

1. A route `/events` allows triggering the script and retrieving the next day's events ;
2. It is primarily accessed by the Raspberry Pi to fetch the data for display or processing.

#### 4.6.6 Hosting on PythonAnywhere

The script is hosted on **PythonAnywhere**, a cloud platform for running Python scripts accessible via the web.

This hosting choice was made for the following reasons :

1. **Permanent accessibility** : the Raspberry Pi can access the script remotely at any time ;
2. **Simplified deployment** : PythonAnywhere is optimized for Flask applications ;
3. **Free tier available** : the free plan is sufficient for the needs of the project ;
4. **Reliability** : no need to manage or maintain a physical or local server.

### 4.7 Sensor Integration and Node-RED Workflow

Finally, temperature monitoring sensors (DHT22) were integrated to enable automated SMS alerts and cloud-based monitoring via Azure IoT Hub. The workflow included :

1. Connect the DHT22 sensor to the Raspberry Pi.
2. Use Node-RED for real-time data visualization.
3. Set up automatic SMS alerts when temperature thresholds were exceeded.
4. Send temperature and humidity data to Azure IoT Hub for remote monitoring.

A prototype circuit was designed using Fritzing to visualize the physical connections.

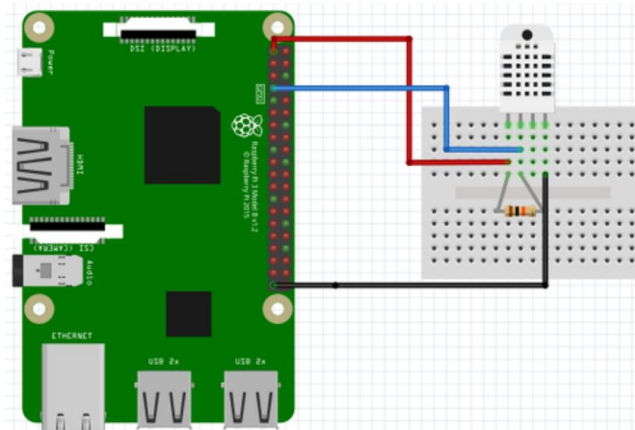


FIGURE 4.6 – Circuit Wiring Between DHT22 sensor and Raspberry Pi

#### 4.7.1 Configuring DHT22 Sensor in Node-RED :

The MQTT data is received by a Node-RED flow, which processes and visualizes the readings. The flow subscribes to the MQTT topics, processes the data, and displays it on a dashboard using charts and gauges. The following figure illustrates the Node-RED flow :

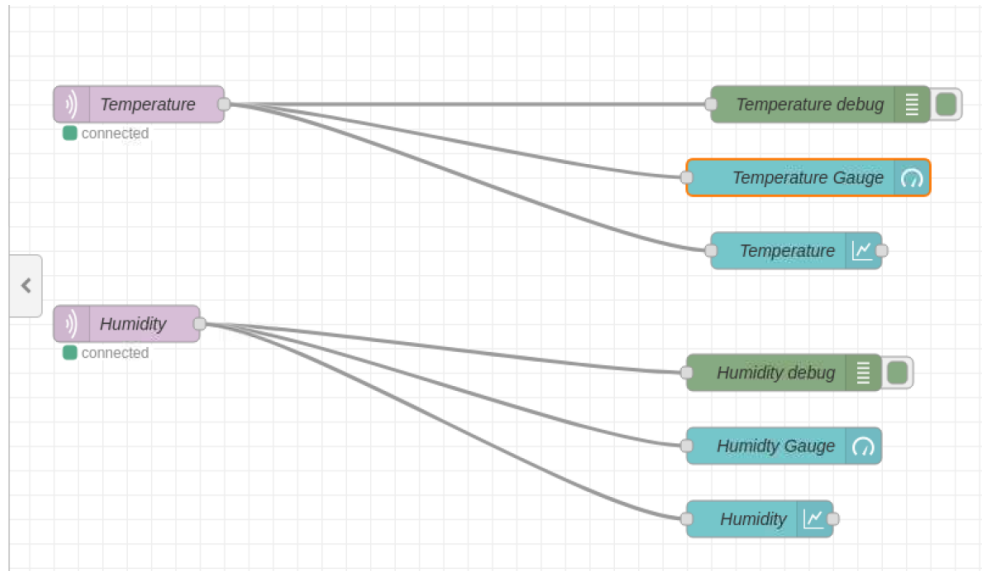


FIGURE 4.7 – Node-RED flow for Processing Sensor Data

#### 4.7.2 Real-Time Dashboard Visualization :

The Python script initializes the DHT22 sensor and reads temperature and humidity values every 10 seconds. It connects to an MQTT broker and publishes the data to two topics : `home/temperature` for temperature readings and `home/humidity` for humidity readings. The script handles errors gracefully, such as sensor read failures, and ensures continuous data transmission. The following figure shows the temperature and humidity gauges displayed on the Node-RED dashboard :

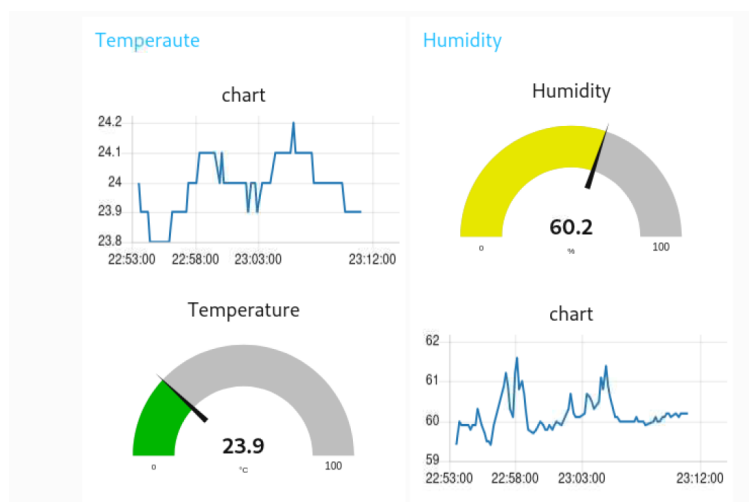


FIGURE 4.8 – Node-RED Dashboard : Sensor data outputs

### 4.7.3 Sending Data to Azure IoT Hub :

1. Drag the `rpi-dht22` node (input).
2. Drag a function node and add the following code :

```
msg.payload = {  
    temperature: msg.payload,  
    humidity: msg.humidity  
};  
return msg;
```

3. Drag the `azure-iot-hub` node (output).
4. Configure it with the Azure IoT Hub connection string.
5. Connect the nodes and deploy the flow.

This setup allows real-time monitoring of temperature and humidity, with data being visualized via Node-RED Dashboard and transmitted to Azure IoT Hub for remote access. Automated SMS alerts are triggered when temperature thresholds are exceeded.

## 4.8 Azure Cloud Platform Exploration

The aim of this activity is to explore Azure cloud services and see how they leverage IoT technologies by cloud integration.

As we mentioned before, the implementation of Azure IoT Hub in our system ensures real-time data collection, processing, and alerting based on sensor readings from multiple ESI facilities. The process follows a structured pipeline : ingesting **sensor data** into *Azure IoT Hub*, **processing** it through *Azure Stream Analytics (ASA)*, and triggering *Azure Functions* to send **SMS alerts** when specific thresholds are exceeded.

### 4.8.1 Azure IoT Hub

The first step involved setting up Azure IoT Hub as the core communication gateway.

#### *Set Up Azure IoT Hub*

1. Go to Azure Portal → Create a Resource → Search IoT Hub.

2. Create a new IoT Hub : with the name you choose
3. Choose a Subscription and Resource Group.
4. Select a Region near you.
5. For us, we used the free tier.
6. Wait for deployment to finish, then open the IoT Hub.

**IoT hub** ...  
Microsoft

**Validating**

Basics   Networking   Management   Add-ons   Tags   **Review + create**

**Pricing**

IoT hub	<b>\$25 USD</b> per month <a href="#">Change basics</a>
Add-ons total	<b>\$0.001 USD</b> per device per month <a href="#">Change add-ons</a>

**Basics**

Subscription	Azure for Students
Resource group	sci
IoT hub name	Notifytrack-RaspberryPi
Region	East US
Disaster recovery enabled	Yes
Tier	Standard
Daily message limit	400,000 (\$25/month)

**Networking**

Connectivity configuration	Public access
Private endpoint connections	None
Allow public network access	Enabled
Minimum TLS Version	1.0

**Management**

Tier	S1
Number of S1 IoT hub units	1
Device-to-cloud partitions	4
Enable Defender for IoT	Disabled

FIGURE 4.9 – IoT Hub Creation

Once deployed, the IoT Hub was configured to register our Raspberry Pi 4 device as an IoT endpoint. The RPi, acting as a sensor node, is added as a device in the IoT Hub.

1. Register the Raspberry Pi as an IoT Device
2. In the IoT Hub, go to IoT Devices.
3. Click + Add Device → Give it a name (e.g., RaspberryPi-01).
4. Keep authentication Symmetric Key.
5. Click Create and copy the Primary Connection String (because we'll need it later).

On the Raspberry Pi side, the Azure IoT SDK was installed to facilitate data transmission, and a python script was developed to collect data from the DHT sensor and send it to the cloud. The script continuously transmits real-time telemetry data to the Azure IoT Hub, where the data is stored and queued for processing. To verify the integrity of the data flow, the Azure IoT Hub monitoring interface was used to inspect incoming messages.

### *Install the sdk*

1. Install Azure IoT SDK on Raspberry Pi
2. On your Raspberry Pi, run : `pip install azure-iot-device`

Then, create a python script (`send_data.py`) to send sensor data to Azure :  
`python send_data.py`

Now your Raspberry Pi is sending data to Azure IoT Hub!

To check if your raspberry is sending data :

1. Go to IoT Hub → Click on your device.
2. Go to "Message to Device" and monitor the incoming data.

Look closely at the console below in Figure 4.10

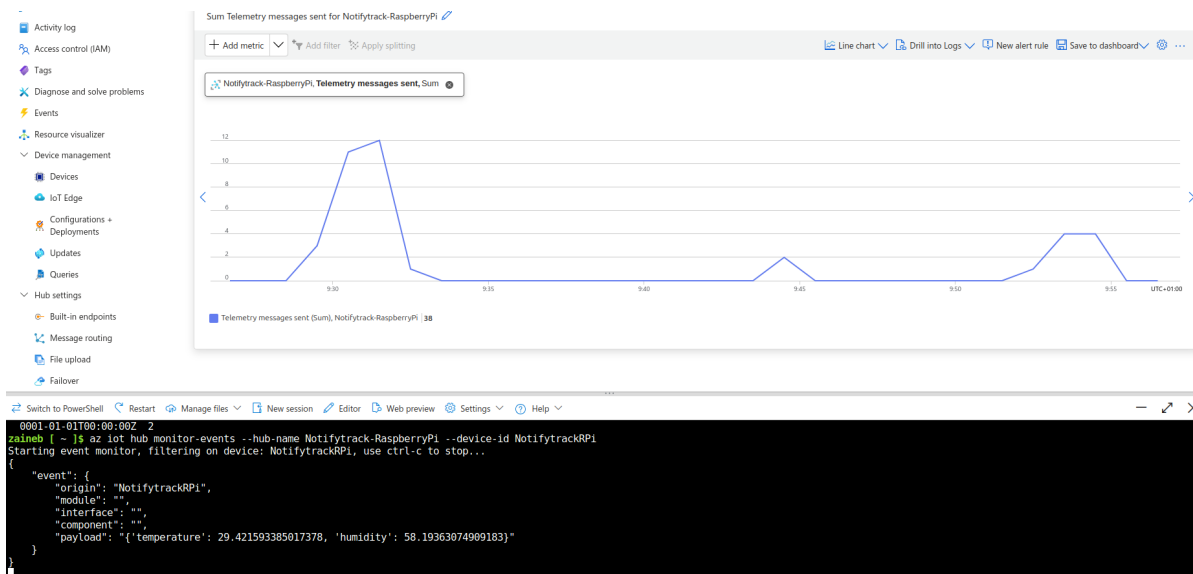


FIGURE 4.10 – Incoming Sensor Data from the Raspberry

## 4.8.2 Azure Stream Analytics


Once the data was successfully ingested, an ASA job was created to process and analyze the telemetry data in real-time. The **IoT Hub** was configured as the

**data input source**, to ensure continuous streaming of sensor readings. The job was set up with an event serialization format of JSON to ensure compatibility with downstream processing services.

That is what we went through during creation :

1. Go to Azure Stream Analytics → NotifytrackASA.
2. Click Inputs → Add stream input → IoT Hub.
3. Select Notifytrack-RaspberryPi as the IoT Hub.
4. Set Event serialization format to JSON.
5. Click Save.


Home > StreamAnalyticsJob | Overview > ASA-sensor-processing

 **ASA-sensor-processing | Inputs** ☆ ...  
Stream Analytics job

◦ « + Add input ↻ Refresh

 Overview

 Activity log

 Access control (IAM)

 Tags

 Diagnose and solve problems

 Resource visualizer

▼ Job topology

Alias ↑	Source type	Type
Notifytrack-RaspberryPiASA	Stream	IoT Hub

FIGURE 4.11 – ASA Job Created

### 4.8.3 Azure Functions

1. In Azure Stream Analytics → Click Outputs → Add → Select Azure Function.
2. Create a new function (or use an existing one).
3. Set Function App Name → Function Name (SendAlertSMS).
4. Click Save.

so here is how it will happen : ASA detects threshold variations (e.g., temperature too high), it triggers the Azure Function (configured as an output). The Azure Function processes the alert and sends an SMS.

## 4.9 Conclusion

This chapter detailed the implementation of the project, focusing on three core components : the integration of the GSM module for offline communica-



tion, the development of a RaspiSMS-compatible platform, the deployment of a Node-RED dashboard, and the cloud integration using Azure IoT Hub for storing and analyzing DHT sensor data. The **GSM module integration** enabled reliable SMS notifications in areas with limited internet connectivity, while the **dashboard platform** provided a user-friendly interface for managing users, events, and preferences. The **Node-RED dashboard** facilitated real-time data visualization and alert notifications, and the **Azure IoT Hub integration** ensured secure storage and advanced analytics for scalability. Together, these components form a robust and scalable system for environmental monitoring and notification.

# General conclusion

In today's fast-paced educational environment, effective communication is critical for ensuring smooth academic and administrative operations. This project addressed the challenge of unreliable internet connectivity by developing a **Smart Notification Gateway System** that leverages SMS for real-time notifications. By integrating IoT devices, a user-friendly dashboard, and cloud-based analytics, the system ensures timely delivery of critical information, even in areas with poor internet access.

The system combines key components : the **GSM module** for offline SMS notifications, the **RaspiSMS-compatible platform** for managing users and events, the **Node-RED dashboard** for real-time data visualization, and **Azure IoT Hub** for secure storage and advanced analytics. These components provide a robust solution for environmental monitoring, emergency alerts, exam reminders, and transportation updates, enhancing safety, efficiency, and communication within the institution.

While the current implementation connects only one sensor to the Raspberry Pi, the system is designed for scalability. By introducing **Arduinos with sensors** as peripheral devices and using **Zigbee** for communication, the Raspberry Pi can process data from multiple sources. This approach allows scaling the number of charts in **Node-RED** while maintaining **MQTT** for efficient data transmission. Additionally, the scalability of **Azure IoT Hub** enables the implementation of **machine learning (ML)** models to predict temperature and humidity trends, further enhancing the system's capabilities.

In conclusion, this project demonstrates the potential of IoT and cloud technologies to address real-world challenges in educational settings. The system improves communication reliability and lays the foundation for future enhancements, contributing to a safer, more efficient, and connected academic environment.

# Références

- [1] <https://nodered.org/docs/>
- [2] <https://tailscale.com/kb>
- [3] <http://documentation.raspisms.fr/>
- [4] <https://www.instructables.com/Raspberry-Pi-Tutorial-How-to-Use-the-DHT->
- [5] [https://help.sia-connect.com/en\\_US/azure-iot-hub-device-provisioning-ser](https://help.sia-connect.com/en_US/azure-iot-hub-device-provisioning-ser)  
[monitoring-iot-hub-events-messages-&-metrics](https://help.sia-connect.com/en_US/azure-iot-hub-device-provisioning-ser)
- [6] <https://learn.microsoft.com/en-us/azure/iot-hub/>
- [7] <https://developers.google.com/calendar/api/guides/overview>