WatchTower

Titolo del progetto: WatchTower
Alunno/a: Tom Schillerwein
Classe: Info 4BC
Anno scolastico: 2024/2025
Docente responsabile: Pascal Poncini

1 Introduzione ... 3
1.1 Informazioni sul progetto ... 3
1.2 Abstract ... 3
1.3 Scopo ... 3
2 Analisi ... 4
2.1 Analisi del dominio ... 4
2.2 Analisi e specifica dei requisiti ... 4
2.3 Use case ... 6
2.4 Pianificazione ... 7
2.5 Analisi dei mezzi ... 7
2.5.1 Software ... 7
2.5.2 Hardware ... 8
3 Progettazione ... 9
3.1 Design dell'architettura del sistema ... 9
3.2 Design dei dati e database ... 10
3.3 Design delle interfacce ... 10
3.4 Design procedurale ... 11
3.4.1 Connessione LDAP ... 11
3.5 Limiti ... 11
4 Implementazione ... 13
4.1 Applicativo Web ... 13
4.1.1 Installazione frontend ... 13
4.1.2 Installazione backend ... 13
4.1.3 Frontend ... 14
4.1.4 Backend ... 16
4.1.5 Container ... 19
4.2 Microcontrollore ... 20
4.2.1 CoreS3 ... 21
4.2.2 NanoC6 ... 22
4.2.3 Architettura implementata ... 22
4.2.4 Registrazione badge ... 23
4.2.5 Accesso ... 24
4.2.6 Comunicazione wireless ... 24
4.2.7 Sensori ... 25
4.2.8 LED ... 25
4.2.9 Montaggio sistema ... 26
5 Test ... 27
5.1 Protocollo di test ... 27
5.2 Risultati test ... 31
5.3 Mancanze/limitazioni conosciute ... 38
6 Consuntivo ... 39
7 Conclusioni ... 40
7.1 Sviluppi futuri ... 40
7.2 Considerazioni personali ... 40
8 Glossario ... 41
9 Bibliografia ... 43
9.1 Sitografia ... 43
10 Indice delle Figure ... 44
11 Allegati ... 45

1 Introduzione

1.1 Informazioni sul progetto

- Titolo Progetto: WatchTower
- Allievi: Tom Schillerwein I4BC
- Docente responsabile: Pascal Poncini
- Scuola: Arti e Mestieri Trevano, sezione Informatica
- Data di inizio e fine: 03.02.2025 – 04.04.2025
- Data di presentazione: 14 – 17.04.2025

1.2 Abstract

The aim of this project is to implement a physical monitoring system for the server room of the black network at the CPT in Trevano. This monitoring includes temperature, humidity, access control and more and is carried out by means of various sensors specially positioned in the server room. It is also necessary to develop a web application that allows the data collected by the sensors to be visualised in graphs and tables, and a system of notifications and alerts for the network admins and teachers, which can also be managed from the application.

1.3 Scopo

Lo scopo di questo progetto è implementare un sistema di monitoraggio fisico della sala server della rete nera della CPT di Trevano. Questo monitoraggio include la temperatura, l'umidità, il controllo degli accessi e altro. Questo monitoraggio viene eseguito tramite diversi sensori posizionati appositamente nella server room. È inoltre necessario sviluppare un'applicazione web che permette la visualizzazione dei dati raccolti dai sensori in grafici e tabelle e un sistema di notifiche e allerte per i sistemisti della rete e i docenti. Anche queste notifiche sono gestibili dall'applicativo web.

2 Analisi

2.1 Analisi del dominio

Questo progetto permette di semplificare il monitoraggio fisico della sala server tramite sensori che trasmettono le informazioni raccolte ad una dashboard a cui i sistemisti hanno accesso. Se i valori rilevati superano un certo livello definito, i sistemisti ricevono un avviso. Al momento i server nella sala vengono monitorati, ma non la sala fisica. Viene unicamente richiesto aprire la porta con un badge con i permessi necessari. Esistono già soluzioni simili, come Monnit, che però sono costose e meno specifiche per la nostra situazione.

2.2 Analisi e specifica dei requisiti

ID: REQ-001	
Nome	Installazione sensori
Priorità	1
Versione	1.0
Note	Montare i sensori, cavi e controller nella sala server.
Sotto requisiti	
001	Avere tutti i componenti hardware necessari per il progetto.

ID: REQ-002	
Nome	Raccolta dati dai sensori
Priorità	1
Versione	1.0
Note	Necessita che i sensori siano correttamente configurati e comunicano con il controller.
Sotto requisiti	
001	Il sistema deve raccogliere i dati di temperatura e umidità dal sensore ENV III.
002	Deve raccogliere i dati del sensore di CO2 e azionare il LED.
003	Deve registrare gli accessi tramite RFID.
004	Deve rilevare movimento attraverso il sensore PIR.
005	Deve acquisire input dalla mini tastiera per capire il tipo di accesso.

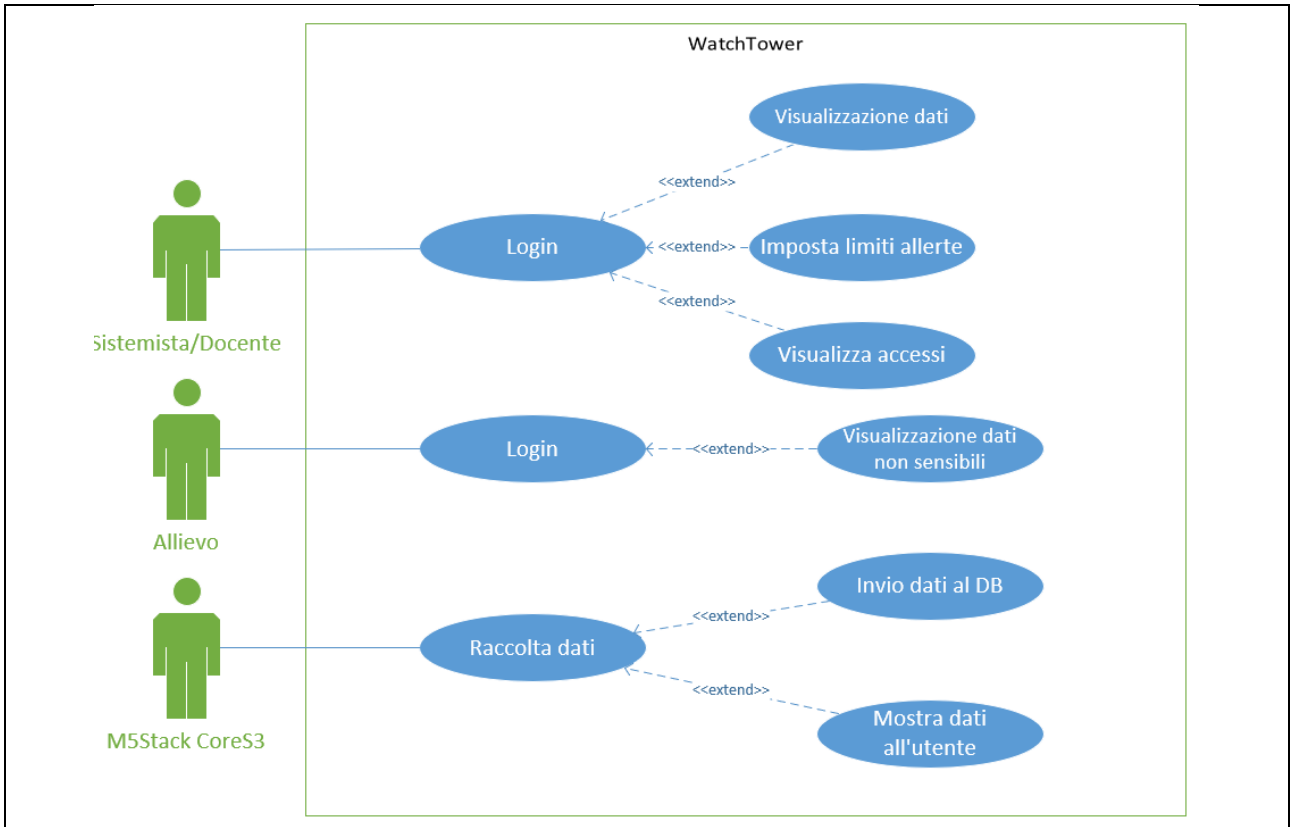
ID: REQ-003	
Nome	Creazione database
Priorità	1
Versione	1.0
Note	Si necessitano i permessi di root
Sotto requisiti	
001	Il database deve supportare la memorizzazione dei dati rilevati dai sensori.

ID: REQ-004	
Nome	Creazione dashboard
Priorità	1
Versione	1.0
Note	Sviluppare un'interfaccia utente chiara e responsive.
Sotto requisiti	
001	Si necessita una maschera di login
002	Bisogna visualizzare i dati dei sensori in tempo reale.
003	Grafici con storico dei dati.
004	Possibilità di configurare notifiche e allerte per valori specifici.

ID: REQ-005	
Nome	Notifiche e allerte
Priorità	1
Versione	1.0
Note	Si necessitano i permessi di root / Dipende dal requisito REQ-001 (Creazione DB)
Sotto requisiti	
001	Invio email in base a valori di default o custom.

2.3 Use case

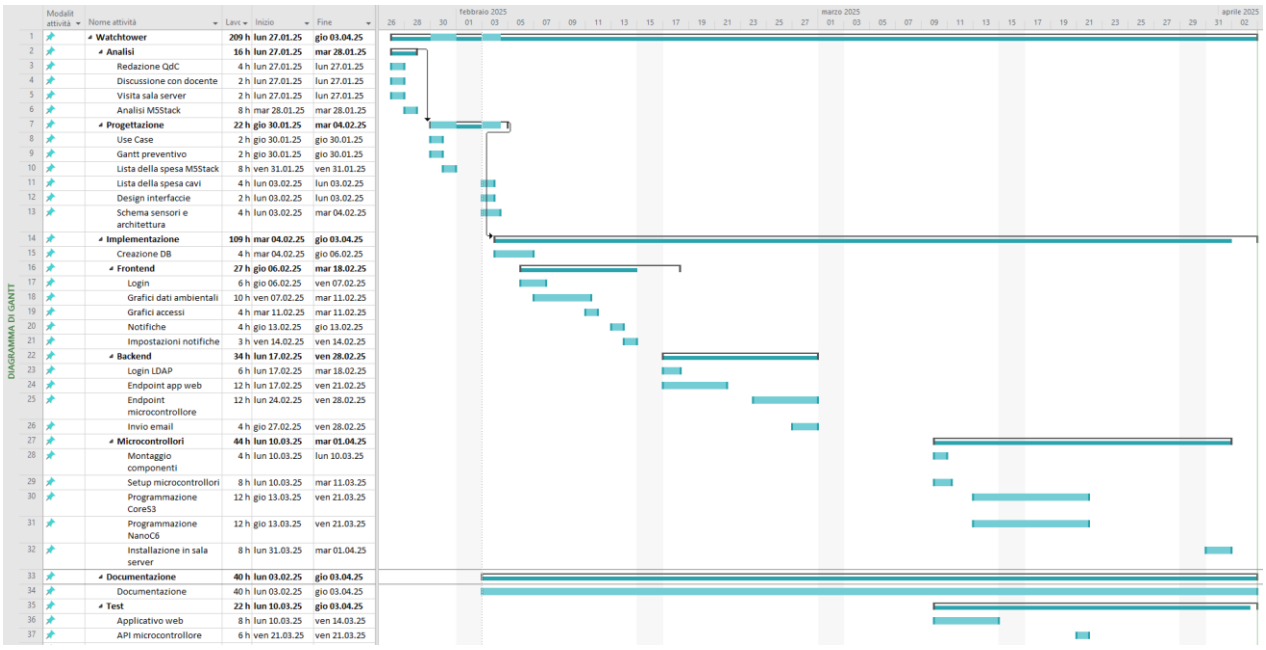
Nel mio progetto ci sono diversi attori, tra cui allievi, sistemisti e docenti. Essi sono gli utilizzatori umani che interagiscono con i microcontrollori per gli accessi e utilizzano la dashboard web per visualizzare i dati registrati.
L'ultimo utente è invece il microcontrollore principale, CoreS3, che raccoglie i dati dagli altri microcontrollori e li invia tramite API REST al backend.



1 Use Case

2.4 Pianificazione

La mia pianificazione del progetto in modalità kanban adattato ad una singola persona.



	Modalità attività	Nome attività	Lavo	Inizio	Fine
1		**Watchtower**	**209 h**	**lun 27.01.25**	**gio 03.04.25**
2		**Analisi**	**16 h**	**lun 27.01.25**	**mar 28.01.25**
3		Redazione QdC	4 h	lun 27.01.25	lun 27.01.25
4		Discussione con docente	2 h	lun 27.01.25	lun 27.01.25
5		Visita sala server	2 h	lun 27.01.25	lun 27.01.25
6		Analisi M5Stack	8 h	mar 28.01.25	mar 28.01.25
7		**Progettazione**	**22 h**	**gio 30.01.25**	**mar 04.02.25**
8		Use Case	2 h	gio 30.01.25	gio 30.01.25
9		Gantt preventivo	2 h	gio 30.01.25	gio 30.01.25
10		Lista della spesa M5Stack	8 h	ven 31.01.25	ven 31.01.25
11		Lista della spesa cavi	4 h	lun 03.02.25	lun 03.02.25
12		Design interfaccie	2 h	lun 03.02.25	lun 03.02.25
13		Schema sensori e architettura	4 h	lun 03.02.25	mar 04.02.25
14		**Implementazione**	**109 h**	**mar 04.02.25**	**gio 03.04.25**
15		Creazione DB	4 h	mar 04.02.25	gio 06.02.25
16		**Frontend**	**27 h**	**gio 06.02.25**	**mar 18.02.25**
17		Login	6 h	gio 06.02.25	ven 07.02.25
18		Grafici dati ambientali	10 h	ven 07.02.25	mar 11.02.25
19		Grafici accessi	4 h	mar 11.02.25	mar 11.02.25
20		Notifiche	4 h	gio 13.02.25	gio 13.02.25
21		Impostazioni notifiche	3 h	ven 14.02.25	ven 14.02.25
22		**Backend**	**34 h**	**lun 17.02.25**	**ven 28.02.25**
23		Login LDAP	6 h	lun 17.02.25	mar 18.02.25
24		Endpoint app web	12 h	lun 17.02.25	ven 21.02.25
25		Endpoint microcontrollore	12 h	lun 24.02.25	ven 28.02.25
26		Invio email	4 h	gio 27.02.25	ven 28.02.25
27		**Microcontrollori**	**44 h**	**lun 10.03.25**	**mar 01.04.25**
28		Montaggio componenti	4 h	lun 10.03.25	lun 10.03.25
29		Setup microcontrollori	8 h	lun 10.03.25	mar 11.03.25
30		Programmazione CoreS3	12 h	gio 13.03.25	ven 21.03.25
31		Programmazione NanoC6	12 h	gio 13.03.25	ven 21.03.25
32		Installazione in sala server	8 h	lun 31.03.25	mar 01.04.25
33		**Documentazione**	**40 h**	**lun 03.02.25**	**gio 03.04.25**
34		Documentazione	40 h	lun 03.02.25	gio 03.04.25
35		**Test**	**22 h**	**lun 10.03.25**	**gio 03.04.25**
36		Applicativo web	8 h	lun 10.03.25	ven 14.03.25
37		API microcontrollore	6 h	ven 21.03.25	ven 21.03.25

2 Diagramma di Gantt preventivo

2.5 Analisi dei mezzi

Per questo progetto utilizzo un PC scolastico per lo sviluppo, un controller M5Stack CoreS3 e alcuni sensori.

2.5.1 Software

- Visual Studio Code 1.93, come editor di testo
- Google Chrome 131.0, come browser per visualizzare il sito durante lo sviluppo
- Mozilla Firefox 126.0, per testare la visualizzazione del sito
- Onlinegantt.com, per il Gantt
- Microsoft Visio, per fare lo Use Case
- Microsoft Word e Obsidian, per redigere la documentazione, il QdC e il diario
- GitLab, come Repository locale della scuola
- Excalidraw, per creare schemi e diagrammi
- ChatGPT, per riscrivere testi, cercare informazioni, correggere errori nel codice del sito web, risolvere problemi nella creazione del sito web, sia backend che frontend, soprattutto per le funzionalità di CRUD
- V0, per creare una grafica del sito web composta da componenti Shadcn

2.5.2 Hardware

Computer utilizzato per le ricerche e lo sviluppo:

- Processore: Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz
- RAM: 32GB
- Scheda Video: NVIDIA GeForce RTX 2060
- SSD: 512GB
- Monitor 1920x1080 60Hz
- Monitor 1600x900 60Hz

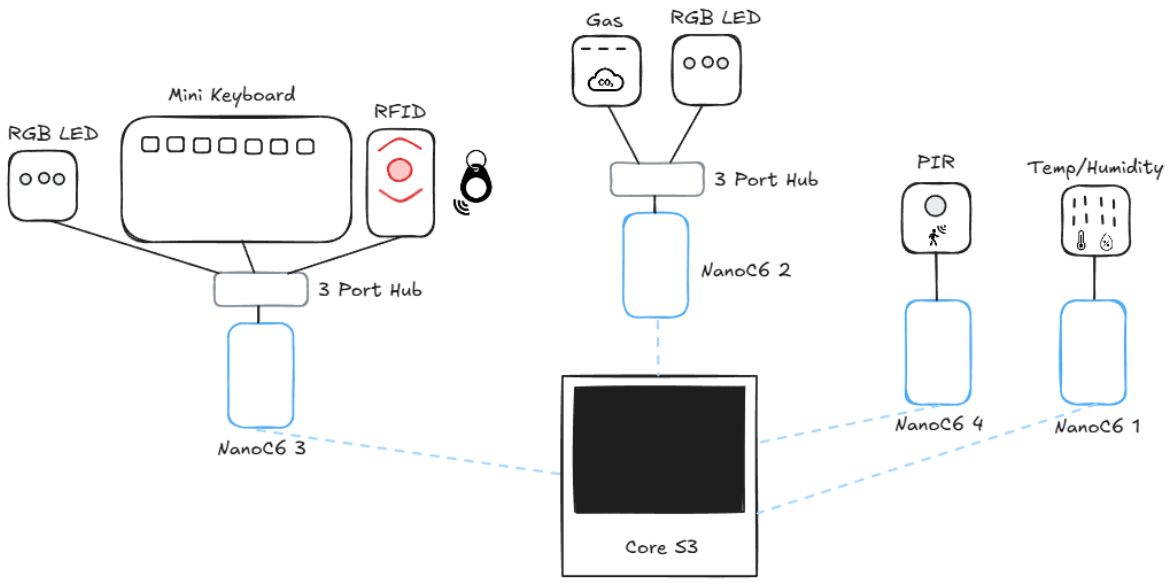
Componenti M5Stack:

- Cores3
- NanoC6
- Mini keyboard
- RFID sensor
- PIR sensor
- ENV III sensor
- 2x LED RGB
- Tvoc/eCO2

3 Progettazione

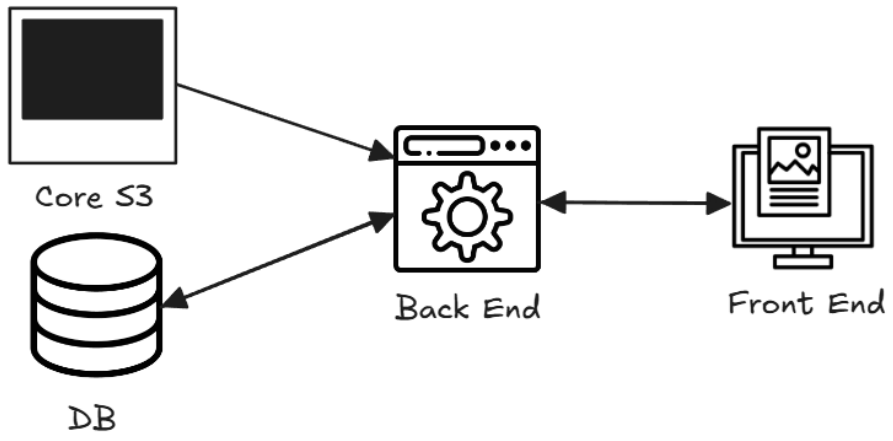
3.1 Design dell'architettura del sistema

Per quanto riguardo la parte IoT del progetto, esso viene create con dei microcontrollori e sensori M5Stack. I microcontrollori Nano C6 comunicano con il microcontrollore principale CoreS3 tramite ESPnow, invece i Nano C6 comunicano con Led, sensori e input tramite cavo Grove. Per collegare più sensori ad un singolo Nano C6, perché ha senso metterli insieme, uso dei hub per convertire 1 port in 3.



3 Schema Microcontrollore e Sensori

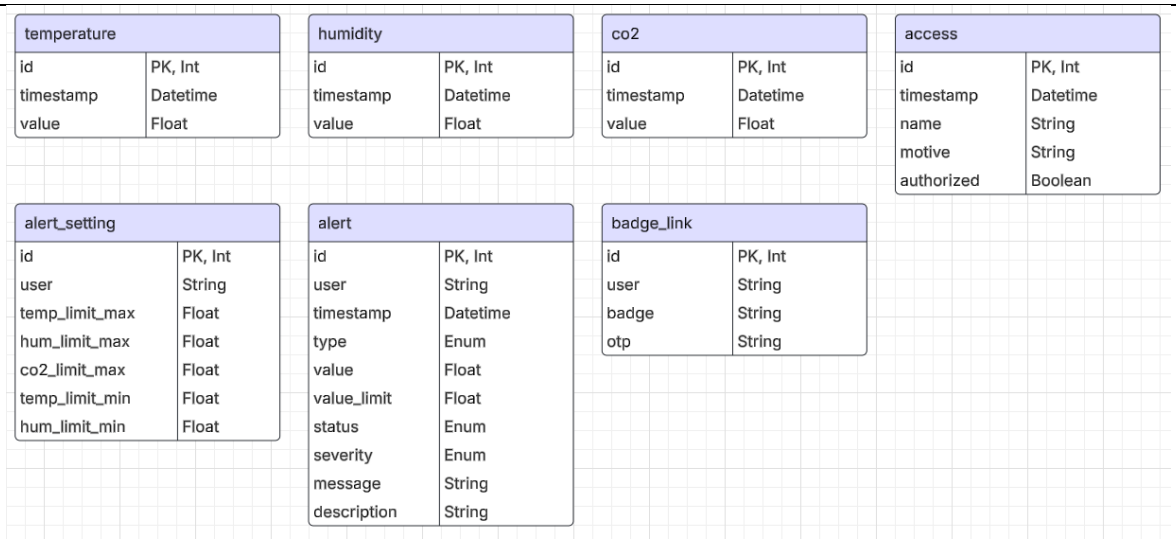
Questo schema rappresenta l'architettura principale del sito web con il CoreS3 che comunica i dati al backend, e questo li salva successivamente nel DB MySQL. Se il frontend fa una richiesta questa viene gestita dal Back End e se necessario prende i dati dal DB.



4 Architettura sistema

3.2 Design dei dati e database

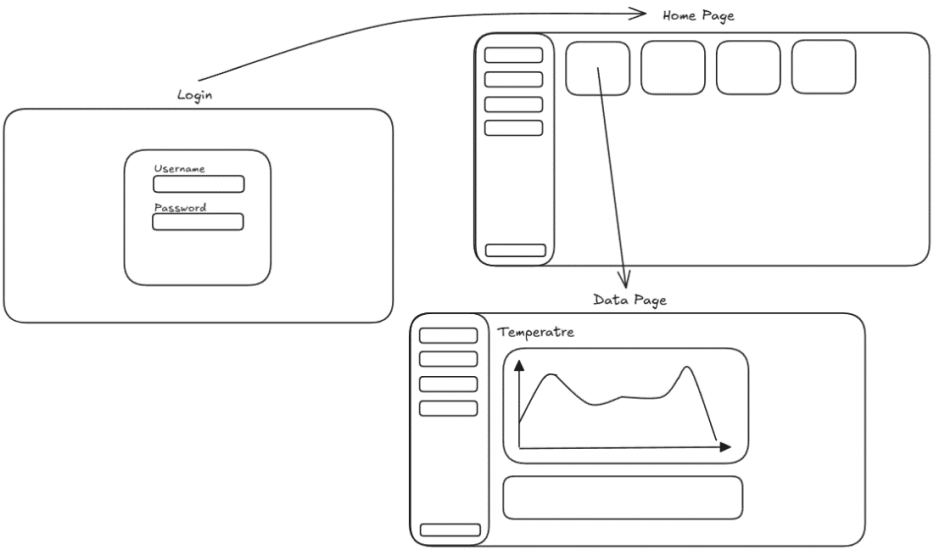
Il database serve soprattutto a salvare i dati provenienti dai sensori, ma anche per salvare le notifiche e le impostazioni di esse degli utenti e il collegamento tra utenti e badge. Infatti, utilizzando LDAP non mi è necessario gestire gli utenti nel DB e devo soltanto salvare il nome univoco degli utenti per le notifiche e le impostazioni. Ho optato per l'uso di enum in alcune tabelle per semplificare il database e perché i possibili valori di questi campi sono moto pochi.



5 Diagramma ER

3.3 Design delle interfacce

Ho creato questa struttura di interfacce, per capire quali dovrò creare, inoltre ho messo in esse un design di base che poi verrà finalizzato quando le implementerò. Per le pagine che visualizzano i dati (temperatura, umidità, livello di CO2, ecc.) userò lo stesso layout di pagina, cambiando semplicemente descrizione, dati e colori.

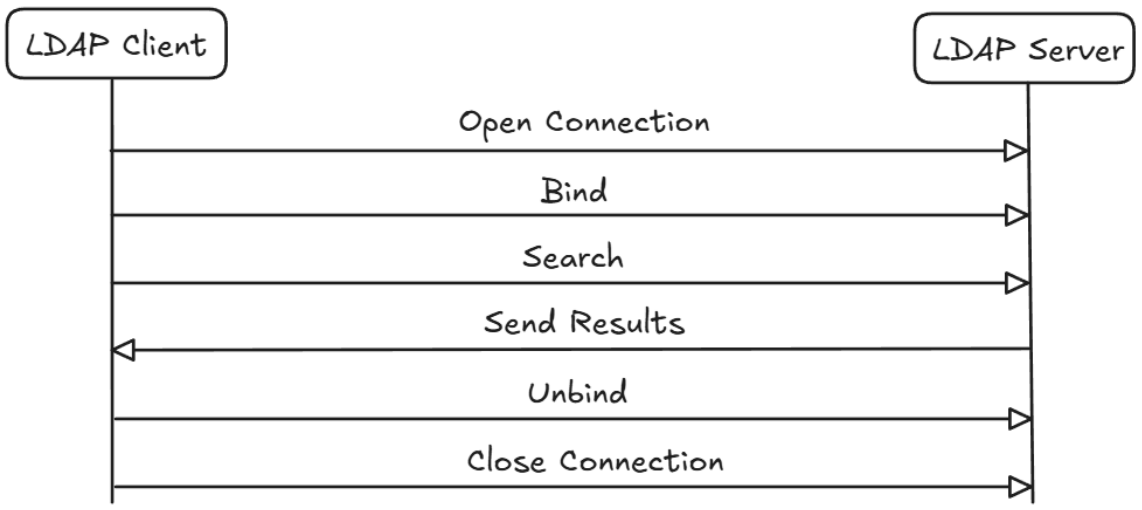


6 Struttura interfacce

3.4 Design procedurale

3.4.1 Connessione LDAP

Il mio progetto richiede l'integrazione con un servizio LDAP per l'autenticazione degli utenti in quanto solo persone della scuola devono avere la possibilità di accedere al mio applicativo. Inoltre usando questa tecnologia, non devo gestire i permessi dei singoli utenti, ma posso semplicemente capire di quale gruppo fa parte un utente e in base a questo decidere cosa può fare/vedere. Per eseguire il bind (autenticazione) iniziale con il server LDAP, utilizzo un utente di servizio creato apposta per il mio applicativo (watch.tower). In seguito faccio la ricerca dell'utente che ha immesso le proprie credenziali, tramite username, che nel LDAP della rete nera corrisponde a sAMAccountName.Se l'utente viene trovato, provo a fare un bind anche di questo utente con la sua password, e se questa riesce vuol dire che l'utente ha immesso username e password corrette. Se questo è il caso viene generato un JWT per l'utente che viene poi reindirizzato alla home page.



7 Protocollo LDAP

3.5 Limiti

Questa implementazione, con M5Stack e i sensori scelti, presenta ovviamente delle limitazioni di cosa è possibile rilevare, di seguito viene elencato ciò che è possibile rilevare con i vari sensori. Inoltre è importante notare che questo sistema può non registrare dati o registrarne di errati, generando potenzialmente falsi allarmi o non generandone degli altri.

ENV III: rileva periodicamente la temperatura e l'umidità dell'ambiente, ma non ne traggono conclusioni, sta perciò all'utente che legge i dati capire i problemi a partire dai dati forniti.

Tvoc/eCO2: rileva periodicamente la concentrazione di CO2 nell'aria, con discetta precisione. Come per il sensore di temperatura e umidità, non è in grado di trarre conclusioni dai dati rilevati e li riporta semplicemente. Per questo motivo non è un rilevatore di incendi, ma semplicemente un indicatore che qualcosa potrebbe essere al di fuori dal normale se i valori rilevati sono troppo elevati.

PIR: rileva il movimento di una persona, tuttavia, tramite dei test con il sensore, ho notato che questo non è accurato al 100%, infatti ogni tanto un movimento non viene rilevato da esso, questo potrebbe portare ad un accesso non autorizzato che non viene colto dal sistema.

Keyboard: tramite questa mini tastiera vengono immessi gli input di un utente, sia per motivare la propria presenza, sia per creare il link tra badge e utente che per segnalare l'uscita dalla sala server. È necessario premere invio dopo l'inserimento dell'input desiderato, per garantire che ciò che è stato scritto venga trasmesso.

RFID: legge lo UID del badge scolastico e lo inva al microcontrollore. Nessuna limitazione conosciuta in questo ambito.

LED: mostrano tramite delle luci lo stato di alcuni valori, per esempio quello di concentrazione di CO2. Nessuna limitazione conosciuta in questo ambito.

4 Implementazione

4.1 Applicativo Web

4.1.1 Installazione frontend

Il frontend dell'applicazione è sviluppato con Vue.js e utilizza Tailwind CSS e Shadcn Vue per la gestione dei componenti UI.
Prima di iniziare l'installazione verificare che node.js e npm sono installati tramite il comando:

```
node -v
npm -v
```

Installazione di Vue.js:

```
npm install -g @vue/cli
npm create vue@latest
cd watchtower
npm install
```

Installazione di tailwind:

```
npm install -D tailwindcss@3 postcss autoprefixer
npx tailwindcss init -p
```

Installazione di Shadcn Vue:

```
npm install shadcn-vue
npx shadcn-vue@latest init
```

Per aggiungere i componenti di Shadcn Vue:

```
npx shadcn-vue@latest add nome_componente (es: button)
```

Per fare partire il server:

```
npm run dev
```

4.1.2 Installazione backend

Il backend è basato su Node.js e utilizza Express.js, Prisma, LDAP.js, JWT e un database MySQL.

Prima di iniziare l'installazione verificare che node.js e MySQL siano installati.

Creazione progetto Node.js con i pacchetti necessari al progetto:

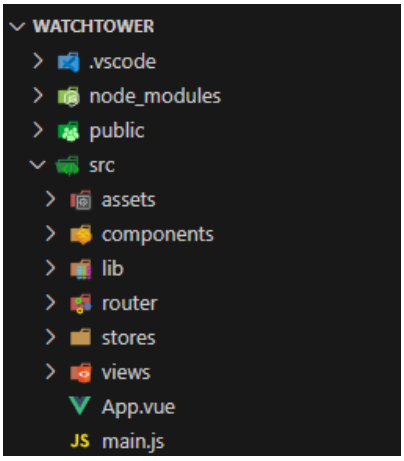
```
npm init
npm install express cors dotenv mysql2 @prisma/client ldapjs jsonwebtoken nodemailer
winston
npx prisma init
```

Modificare i file .env (connessione e credenziali DV) e schema.prisma (modello dei dati) con le informazioni giuste dopodiché utilizzare il seguente comando prisma per creare una migrazione:

```
npx prisma migrate dev --name nome_migrazione
```

4.1.3 Frontend

Il frontend è sviluppato con Vue.js usando la sintassi di script setup e composition api. Per la parte grafica ho usato Tailwind CSS in combinazione con Shadcn Vue. Ho strutturato le view utilizzando soprattutto componenti di Shadcn aggiustandoli in modo da soddisfare le mie necessità.



8 Struttura cartelle frontend

4.1.3.1 Componenti

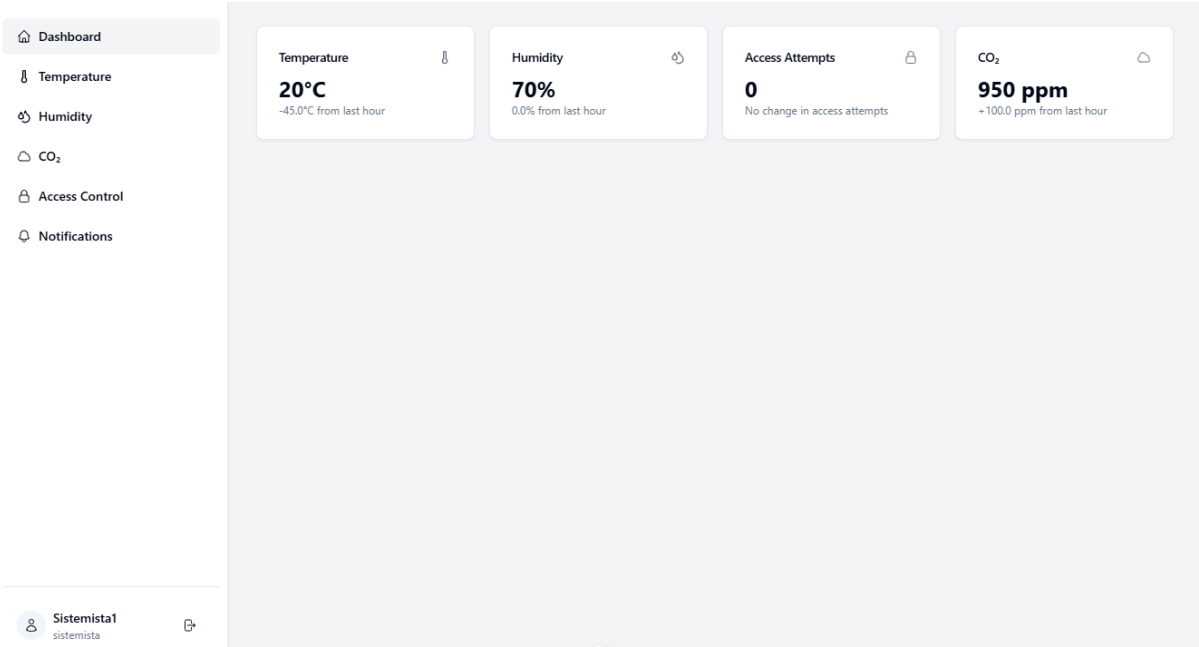
Per rendere il codice più leggibile e riutilizzabile, oltre a usare i componenti Shadcn, ho creato dei componenti custom, che una volta definiti posso riutilizzare nell'intero applicativo. Questo è molto comodo soprattutto perché questi componenti sono parametrizzabili, perciò dalla view principale è possibile modificare i dati visualizzati dai componenti.

Esempio di componente custom parametrizzabile, in questo caso la card dell'home page:

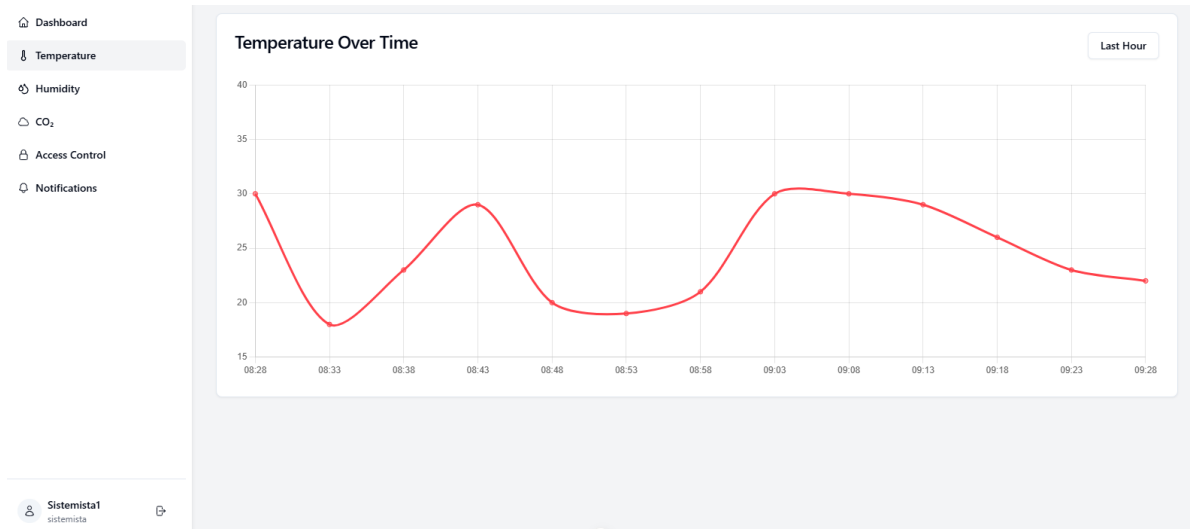
```
<MetricCard v-if="co2Data" :title="'CO₂'" :value="co2Data.value + ' ppm'"
      :description="co2Data.description" link="/co2">
       <template #icon>
            <CloudIcon class="size-4 text-gray-500" />
      </template>
</MetricCard>
```

4.1.3.2 Interfacce

Per poter confrontare il design iniziale e il risultato finale delle interfacce grafiche di seguito viene mostrato il risultato finale della pagina home e della pagina della temperatura. Come si può notare la disposizione degli elementi è rimasta uguale, e ho semplicemente popolato di scritte, valori e grafici le pagine.



9 Pagina Home

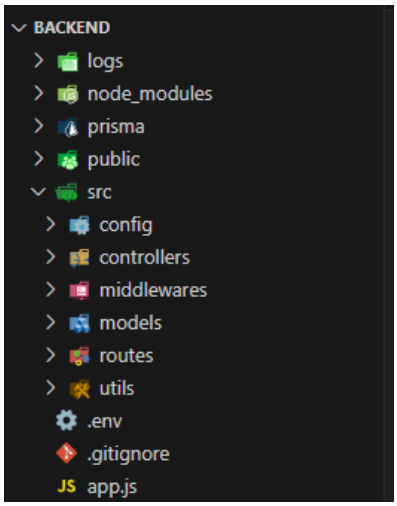


10 Pagina Temperatura

Per ogni pagina dei dati, è possibile cambiare il grafico da “ultima ora”, a “ultimo giorno” e “ultima settimana” dove in base al grafico cambia la scala temporale e di dati visualizzati.

4.1.4 Backend

Il backend è basato su Node.js con Express per gestire le API.



11 Struttura cartelle backend

Ho strutturato il progetto secondo le best practice di Node.js, per permettere una buona leggibilità del codice e una facile modifica del progetto.

Ho inoltre utilizzato le seguenti tecnologie per implementare le funzionalità richieste:

- Express → Framework web per gestire API REST.
- Prisma ORM → Gestione del database MySQL in modo efficiente.
- LDAP.js → Connessione e autenticazione con il server LDAP.
- JWT → Gestione della sessione degli utenti.
- Nodemailer → Invio delle email di notifica.
- Winston → Gestione dei log.

4.1.4.1 Autenticazione

L'utente si autentica tramite LDAP, una volta che le credenziali inserite sono validate dal server LDAP, il mio applicativo crea un JWT, il quale scade dopo 6 ore dalla creazione. Questo token deve essere incluso nelle richieste successive fatte dal client al server, altrimenti risulteranno con una risposta di errore. Il server può verificare la validità del token JWT e autorizzare l'accesso alle risorse in base alle informazioni contenute nel token (tramite middleware), per esempio in base al ruolo (allievo o sistemista/docente).

Nel codice seguente c'è la funzione per verificare un JWT passato nella richiesta del client, dove per prima cosa viene estratto il token dall'header della richiesta, dopodiché viene verificato se il token è presente e infine verificato, usando anche la secret key con cui era stato creato. Se il token è valido il middleware non interviene e fa continuare l'esecuzione altrimenti ritorna un errore:

```
function authenticateToken(req, res, next) {
  const token = req.headers["authorization"];
  if (!token) {
    return res.status(403).json({ message: "Access denied" });
  }
  jwt.verify(token, process.env.SECRET_KEY, (err, user) => {
    if (err) {
      return res.status(403).json({ message: "Invalid token" });
    }
    req.user = user;
    next();
  });
}
```

4.1.4.2 Log

In questo progetto ho dato importanza anche ai log, utilizzando la libreria Winston per la loro creazione, in questo modo posso gestire la gravità degli eventi e categorizzarli in log di info, warning e error. Ho inoltre diviso i log in tre file diversi: nel primo si trovano i log del server, per esempio di quando si accende; nel secondo si trovano i log che riguardano l'autenticazione di un utente e l'ultimo contiene tutti i log degli endpoint del server, siano essi chiamati dal frontend o dal microcontrollore.
Di seguito un esempio di creazione di log e il suo risultato in forma scritta nel file di log, sia per una richiesta da parte dell'applicativo web, sia che del microcontrollore.

Richiesta dati grafico ultima ora temperatura:

```
logger.info(`Successfully fetched ${type} records for the last hour.`);
```

Risultato:

```
03-04-2025, 11:19:17 [INFO]: Successfully fetched temperature records for the last hour.
```

Richiesta salvataggio dati temperatura registrati dal microcontrollore:

```
logger.info(`API Call: Temperature data saved successfully - Value: ${value}`);
```

Risultato:

```
03-04-2025, 12:10:14 [INFO]: API Call: Temperature data saved successfully - Value: 23.10
```

4.1.4.3 Salvataggio dati

Gli endpoint per salvare dati nel DB sono preceduti da /api, e vengono chiamati dal microcontrollore per salvare i dati rilevati dai sensori. C'è un endpoint per ogni categoria di dati rilevata (temperatura, umidità, accessi e CO2), qui i dati vengono validati e successivamente salvati nella corrispondente tabella.

4.1.4.4 Fetch dei dati

Gli endpoint che permettono all'applicativo web di prendere i dati per i grafici e le tabelle per ogni tipo di dato sono situati in /. Qui ho creato un endpoint comune per ogni tipo di grafico in cui dinamicamente viene utilizzato il tipo giusto in base al parametro della richiesta:

```
router.get("/:type/lastHour", authenticateToken, async (req, res) => {
  await getLastHourData(req.params.type, res);
});
```

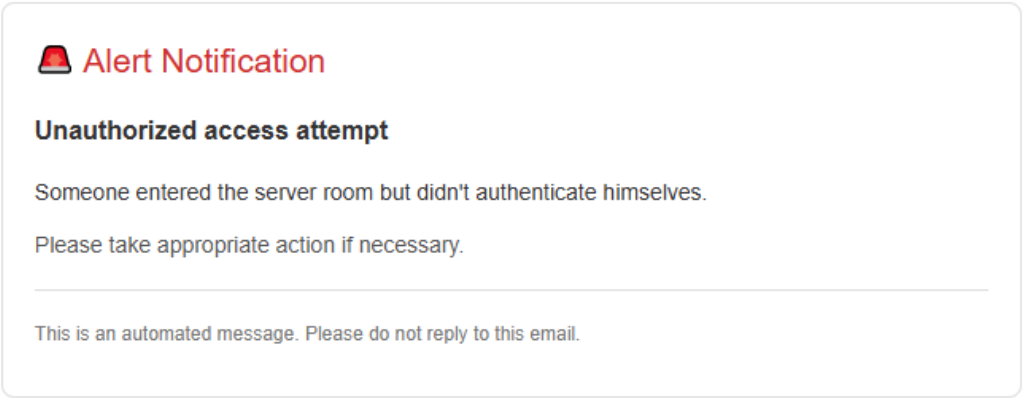
4.1.4.5 Salvataggio accessi

L'endpoint che salva gli accessi alla sala server, /api/access, ma prima di salvare direttamente i dati che ha ricevuto, controlla i dati ricevuti e verifica se l'utente che eseguito l'accesso si trova ancora nel LDAP. Se questo non è il caso viene eliminato il recordi del badge di questo utente, e l'accesso viene marcato come non autorizzato. Altrimenti esso viene salvato con utente, data, motivo e se è autorizzato.

```
api.post("/access", authenticateToken, async (req, res) => {
  try {
    const { name, motive, authorized, timestamp } = req.body;
    const ldapUsers = await fetchLdapUsers(["CN=Docenti", "CN=Sistemisti"]);
    if (timestamp && !isNaN(Date.parse(timestamp))) {
      if (!ldapUsers.some(user => user.username[0] === name)) {
        logger.info("User no more in LDAP in /api/access");
        logger.info("Removing user from link badge in /api/access");
        await prisma.badge_link.deleteMany({
          where: { user: name },
        });
        await prisma.access.create({
          data: {
            timestamp: new Date(timestamp),
            name: "",
            motive: "",
            authorized: false
          },
        });
        createNotifications("ACCESS", 0, timestamp);
        return res.status(403).json({ message: "User not in LDAP" });
      }
      await prisma.access.create({
        data: {
          timestamp: new Date(timestamp),
          name: name || "",
          motive: motive || "",
          authorized,
        },
      });
      logger.info("Access data saved successfully in /api/access");
      if (!authorized) {
        createNotifications("ACCESS", 0, timestamp);
      }
      return res.status(201).json({ message: "Access data saved successfully" });
    } else {
      logger.error("Wrong data sended to /api/access");
    }
    return res.status(500).json({ message: "Invalid access data." });
  } catch (error) {
    …
  }
});
```

4.1.4.6 Notifiche e email

Per avvisare gli utenti sistemisti e docenti di accessi non autorizzati o valori critici rilevati dal sistema, esso genera delle notifiche per ogni utente e per eventi gravi invia anche delle email di avviso ad essi. I valori limiti per cui viene generata un'allerta sono personalizzabili per garantire che ogni utente venga solo avvisato per i suoi casi di necessità. Le notifiche sono divise in tre categorie di gravità, high, medium e low, con i rispettivi colori rosso, giallo e verde. Nella prima categoria si situano gli accessi non autorizzati e i valori ambientali rilevati che superano del 10% la soglia impostata, per questi viene generata un'email. Le notifiche di gravità media vengono generate quando i valori ambientali superano i limiti impostati, ma rimangono sotto al 10% di margine. Le notifiche di gravità bassa servono solo per informare l'utente che i valori ambientali sono tornati a dei valori normali. Per esempio la seguente è un'email di notifica quando qualcuno entra nella sala server senza autenticarsi.



12 Email di notifica

Oltre a ricevere un'email, la notifica sarà anche visibile sul sito web tramite una card, se si preme su essa è possibile marcarla come risolta per non visualizzarla più.

4.1.5 Container

Dopo aver sviluppato l'intero applicativo web e averlo testato, ho deciso di fare il deploy tramite container, perciò ho creato un container per il DB, uno per il frontend e uno per il backend. Non ho dovuto cambiare niente nel codice dato che le tre parti erano già separate tra di loro durante lo sviluppo. Per il container database è possibile utilizzare una distribuzione Docker di MySQL di base, invece per il frontend e il backend devo utilizzare dei Dockerfile per creare dei container personalizzati.
Il Dockerfile del backend è piuttosto semplice, in quanto prende la distribuzione Docker di node 22.14 e copia dentro i miei file di progetto, dopodiché installa le dipendenze necessarie e istanzia l'ORM Prisma e in seguito dichiara la porta su cui viene esposto il servizio (3000). Come ultimo esegue la migrazione nel database e poi fa partire il server node:

```
FROM node:22.14
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm install
COPY . .
RUN npx prisma generate
EXPOSE 3000
CMD npx prisma migrate deploy && node app.js
```

Invece per il frontend ci sono un paio di passaggi in più, anche qui si parte dalla stessa distribuzione di base e si copiano i file del progetto e si installano le dipendenze. Dopodiché forzo una variabile di ambiente direttamente nel Dockerfile, cosa che normalmente non si fa, perché si potrebbero assegnare dal docker-compose, ma in questo caso è necessario perché pe far funzionar il frontend devo generare dei file e la variabile d'ambiente deve essere impostata prima che vengano compilati. Dopodiché installo apache web server e rimpiazzo i file di configurazioni di default con quelli creati da me. Prima di avviare apache espongo la porta 80 sui cui è accessibile la dashboard:

```
FROM node:22.14 AS build
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm install
COPY . .
ENV VITE_BASE_URL_API="http://watchtower-api.caprover.samt.local:3333"
RUN npm run build
FROM httpd:latest
COPY --from=build /app/dist/ /usr/local/apache2/htdocs/
COPY public/new-apache.conf /usr/local/apache2/conf/httpd.conf
COPY public/.htaccess /usr/local/apache2/htdocs/
EXPOSE 80
CMD ["httpd", "-D", "FOREGROUND"]
```

Per combinare il tutto viene usato il file docker-compose, nel quale vengono indicate tutte le variabili d'ambiente, le dipendenze dei container tra di loro, la rete su cui inserire i Docker e molto altro.

4.2 Microcontrollore

Per la parte di IoT (Internet of Things), ho optato per una soluzione basata su M5Stack, più precisamente un microcontrollore CoreS3 combinato con dei NanoC6 per tramettere i dati dai sensori al backend che poi salva i dati nel database.
Ho programmato i microcontrollori in Python, con le necessarie librerie per interagire con l'hardware. Questo viene reso molto più facile dalla piattaforma uiflow.m5stack.com, un IDE online sviluppata appositamente per M5Stack, con cui è possibile eseguire codice e scaricare i file sui microcontrollori.

Per poter utilizzare i vari microcontrollori è necessario installare il firmware adatto, io l'ho fatto con M5burner, un applicativo che permette di scegliere il firmware giusto per ogni prodotto m5Stack e "bruciarlo" sul dispositivo. Questo viene reso necessario perché quando questi dispositivi vengono venduti, potrebbero avere una versione di firmware obsoleta oppure in alcuni è installato un firmware di dimostrazione, il quale contiene un programmino per mostrare tutte le funzionalità disponibili allo sviluppatore.

13 M5burner esempio firmware per CoreS3

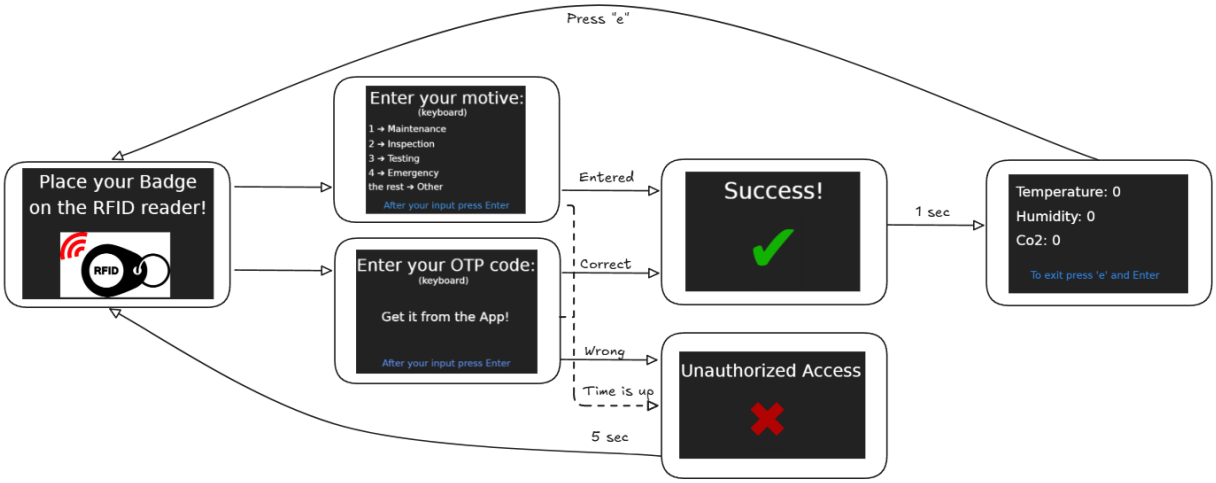
4.2.1 CoreS3

I microcontrollore M5Stack CoreS3 è il controller principale della mia implementazione, infatti è colui che manda i dati al backend e colui che riceve i dati dai sensori e gestisce i LED. Ad esso sono collegati fisicamente un LED e il sensore del CO2, i quali vengono perciò controllati completamente da esso. Tutti gli altri sensori/input sono collegati a dei NanoC6, che mandano a loro volta i dati al CoreS3.
Grazie ad ESPnow, un protocollo wireless che consente una trasmissione veloce ed efficiente dei dati tra dispositivi senza la necessità di una rete Wi-Fi,i NanoC6 possono inviare i dati al CoreS3 con poca latenza e un basso consumo energetico, cosa che lo rende una buona tecnologia per il mio caso d'uso.

14 M5Stack CoreS3

4.2.1.1 Schermate utente

Ho implementato una serie di schermate da visualizzare sul CoreS3, in modo da dare un feedback visivo all'utente di cosa sta facendo oppure di cosa deve fare per continuare il processo di accesso. Una volta eseguita la procedura d'accesso, viene visualizzata una schermata con informazioni utili, come temperatura, umidità e co2.
Di seguito si trovano tutte queste e la logica di come si passa da una all'altra. Per diminuire il consumo energetico e per non bruciare i led dello schermo ho implementato che dopo 5 minuti di inattività da quando viene accesso il CoreS3, o dopo che qualcuno esce dalla sala server, questo schermo va in risparmio energetico e si riaccende solo quando qualcuno rientra nella stanza. Questo processo di accesso, speigato dalle schermate, è anche descritto nella guida d'uso allegata.



15 Schermate CoreS3

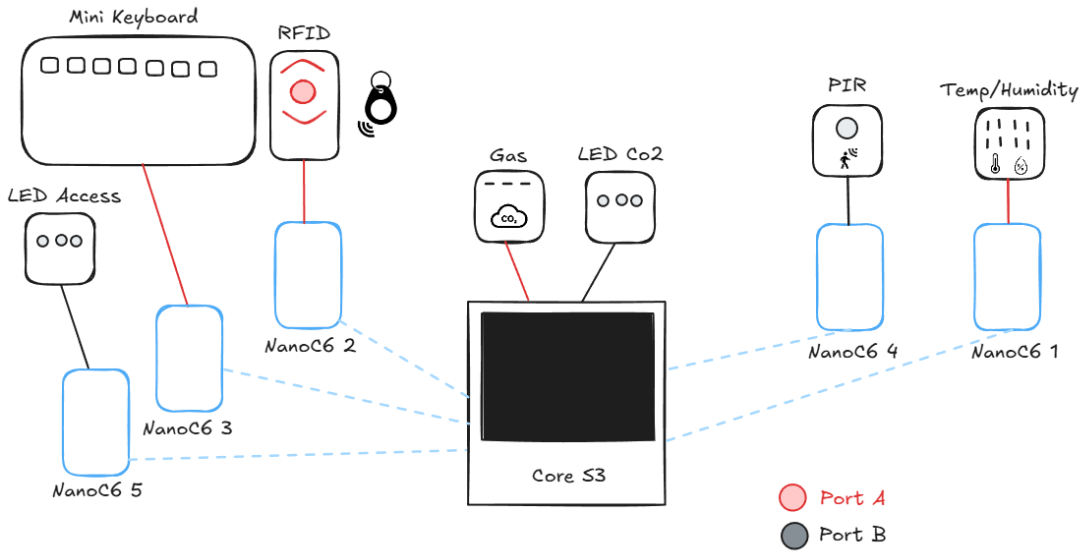
4.2.2 NanoC6

I microcontrollori M5Stack NanoC6 sono versioni compatte del CoreS3, con funzionalità ridotte. Il loro ruolo principale è raccogliere i dati dai sensori o dagli input e trasmetterli al CoreS3. Inoltre, possono ricevere dati da quest'ultimo e, in base a questi, decidere se accendere un LED o attendere un input dalla tastiera. Questi microcontrollori sono utilissimi quando un fattore importante è il peso o la grandezza, dato che questi sono veramente molto compatti e leggerissimi.

16 M5Stack NanoC6

4.2.3 Architettura implementata

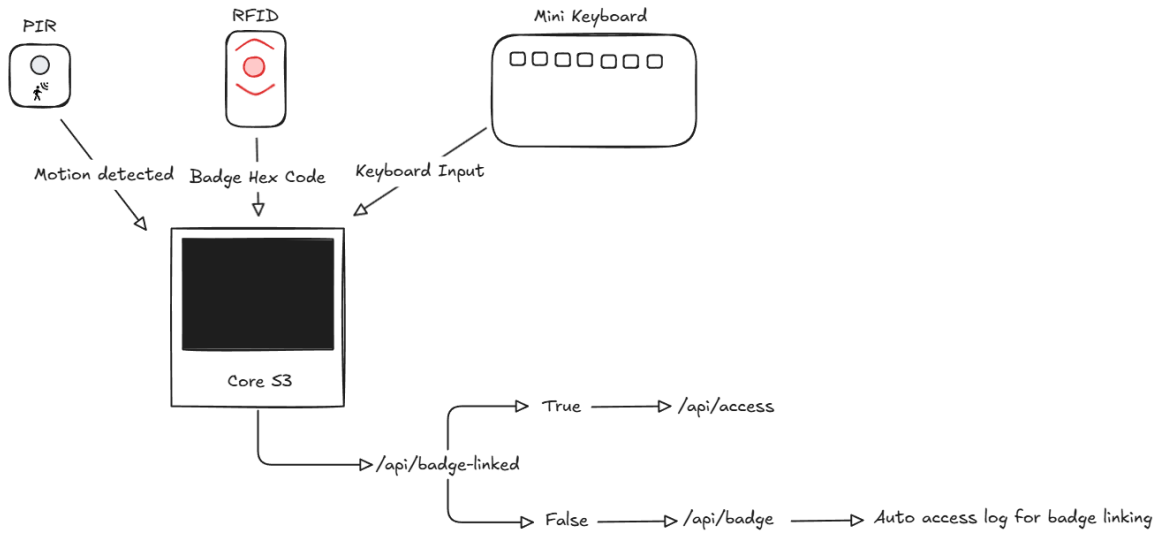
Mentre ho imparato ad usare le componenti M5Stack, mi sono accorto che gli hub che ho comandato non sono adatti al mio caso d'uso, in quanto ripetono semplicemente il segnale in entrata e non è possibile gestire le porte. Per questo motivo ho dovuto modificare la mia architettura dei sensori e microcontrollori. Dopo una ricerca approfondita sugli hub disponibili (in realtà degli switch, ma la nomenclatura di M5Stack li definisce così) ho scoperto che non esistono di tipo misto, perciò che combina I2C (porta A, rossa) e generic I/O (porta B, nera), in modo da collegare sia un LED che un sensore alla stessa porta di un NanoC6. Per questo motivo, e per non fare un'altra comanda, ho deciso di modificare l'architettura in modo da non utilizzare più hub ma collegare direttamente un LED e un sensore al CoreS3 e collegare gli altri componenti direttamente ad un NanoC6.
Questo cambiamento non influenza le funzionalità del progetto ma ne altera solo l'architettura fisica. Da questo cambiamento risulta il seguente schema.



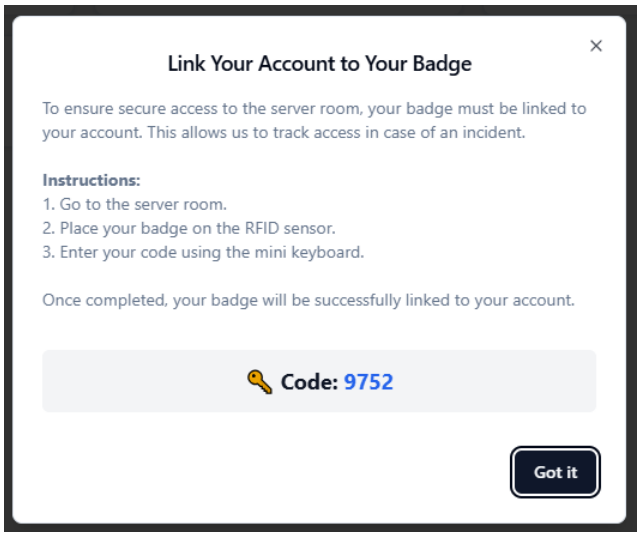
17 Architettura M5Stack implementata

4.2.4 Registrazione badge

Per registrare meglio gli accessi nella sala server, è necessario sapere chi è entrato e quando. Questa informazione viene recuperata tramite il badge della scuola, sul quale è salvata una stringa esadecimale univoca. Essa deve però essere associata ad un nome utente per generare del valore aggiuntivo al mio progetto, perciò, risulta necessario salvare questo collegamento tra badge e utente nel database. Questo processo viene spiegato nello schema sottostante, infatti un utente docente o sistemista, quando accede alla dashboard, riceve un codice di registrazione, con il quale si deve recare nella sala server e immetterlo nella tastiera dopo aver posizionare il badge sul sensore RFID. Quando un utente immette dei valori dalla tastiera e mette il badge sul sensore, il CoreS3 manda una richiesta al backend per capire se questo utente ha già effettuato il link tra badge e username. Se questo è il caso viene registrato come accesso, altrimenti viene effettuato il link nel database e viene registrato automaticamente un accesso con motivo di registrazione badge.



18 Schema funzionamento badge



19 Interfaccia registrazione badge, con istruzioni

4.2.5 Accesso

Una delle principali funzionalità è quella di monitorare gli accessi alla sala server. Per fare ciò viene utilizzata una combinazione di sensori (PIR, RFID e MiniKB). Il sensore di movimento a infrarossi (PIR) percepisce un movimento di una persona che entra dalla porta, e da questo momento essa ha 60 secondi di tempo per appoggiare il proprio badge sul lettore RFID e inserire il motivo della loro presenza tramite la mini tastiera (1: Maintenance, 2: Inspection, 3: Testing, 4: Emergency, tutti gli altri tasti: Other) e premere invio per confermare. Con questo l'accesso è stato registrato come autorizzato, con il motivo, il nome dell'utente e il timestamp nel database. Se dopo 60 secondi queste azioni non sono avvenute, il sistema registra un accesso non autorizzato. Quando l'utente intende uscire, deve premere il tasto "e", per dire al microcontrollore che l'utente ora sta uscendo. Dopodiché potrà essere registrato un nuovo accesso.

4.2.6 Comunicazione wireless

Per fare comunicare i microcontrollori NanoC6 con il microcontrollore CoreS3, ho optato per il protocollo ESP-NOW, creato da Espressif, che permette di fare delle comunicazioni point to point (unicast) o broadcast tramite i mac address dei dispositivi ESP. È molto interessante per IoT a causa del suo basso consumo di energia e di una bassa latenza. Inoltre, ha il vantaggio che non dipende da una connessione Wi-Fi, che potrebbe non essere disponibile a causa di problemi tecnici, e perciò anche la comunicazione dei microcontrollori andrebbe a cadere.
Per rendere sicura la comunicazione wireless esiste un'opzione di ESP-NOW, che permette di criptare la propria comunicazione con una chiave di 16 byte che deve essere condivisa tra i due o più dispositivi che devono comunicare. Quando inizia una comunicazione sicura essa viene cifrata da AES-128, uno degli algoritmi di cifratura più utilizzati globalmente.

```
espnow_0 = M5ESPNow(1)
espnow_0.set_pmk_encrypt(key)
espnow_0.set_add_peer(mac_address, 1, 0, True, key)
```

Per inviare i dati a un peer già registrato con ESP-NOW, basta specificare il numero di connessione (che viene definito durante la registrazione della connessione) e inviare i dati.

```
data = data.hex()
espnow_0.send_data(1, data)
```

Invece la comunicazione del CoreS3 con il backend dell'applicativo, tramite API REST, viene eseguita tramite il Wi-Fi della rete nera, più preciso BLACKNET-DEVICES, al quale il microcontrollore si connette automaticamente all'avvio, e se dovesse fallire, ci riprova ogni secondo:

```
def connect_wifi():
  global ssid, wifi_password, wlan
  wlan = network.WLAN(network.STA_IF)
  wlan.active(True)
  wlan.connect(ssid, wifi_password)
  while not wlan.isconnected():
    time.sleep(1)
    print("Connecting to Wi-Fi...")
  print("Connected to Wi-Fi: ", ssid)
```

Un esempio di richiesta che viene fatta tramite Wi-Fi sono le richieste REST all'API del backend, sia per salvare dati che per richiedere informazioni sui badge degli utenti. In questo esempio viene inviato il dato di CO2 registrato:

```
http_req = requests2.post(server_url + "co2",
json={'value':co2,'timestamp':get_timestamp()}, headers={'Content-Type':
'application/json','Authorization':token})
```

4.2.7 Sensori

Per poter implementare le funzionalità richieste, sono necessari alcuni sensori e input per rilevare i dati necessari nella sala server. Di seguito sono elencati i sensori M5Stack utilizzati, con i dati interessanti per il mio progetto che rilevano:

- ENV III → dati di temperatura e umidità
- PIR → rileva raggi infrarossi riflessi dagli umani, sensore di movimento
- RGB LED → led configurabili con tutti i colori rgb
- RFID → rileva le informazioni del badge
- CardKeyBoard → mini tastiera per inserire codice di registrazione badge e inserimento motivo acceso
- Tvoc/eCo2 → valore del co2 nell'aria

4.2.8 LED

Nell'architettura implementata si trovano due LED, Led Access e Led Co2. Come si può capire anche dal nome, il primo indica se qualcuno si trova nella sala server, infatti, esso è situato all'esterno di essa e tramite il colore verde indica che la stanza è libera, con il rosso che qualcuno si trova al suo interno.

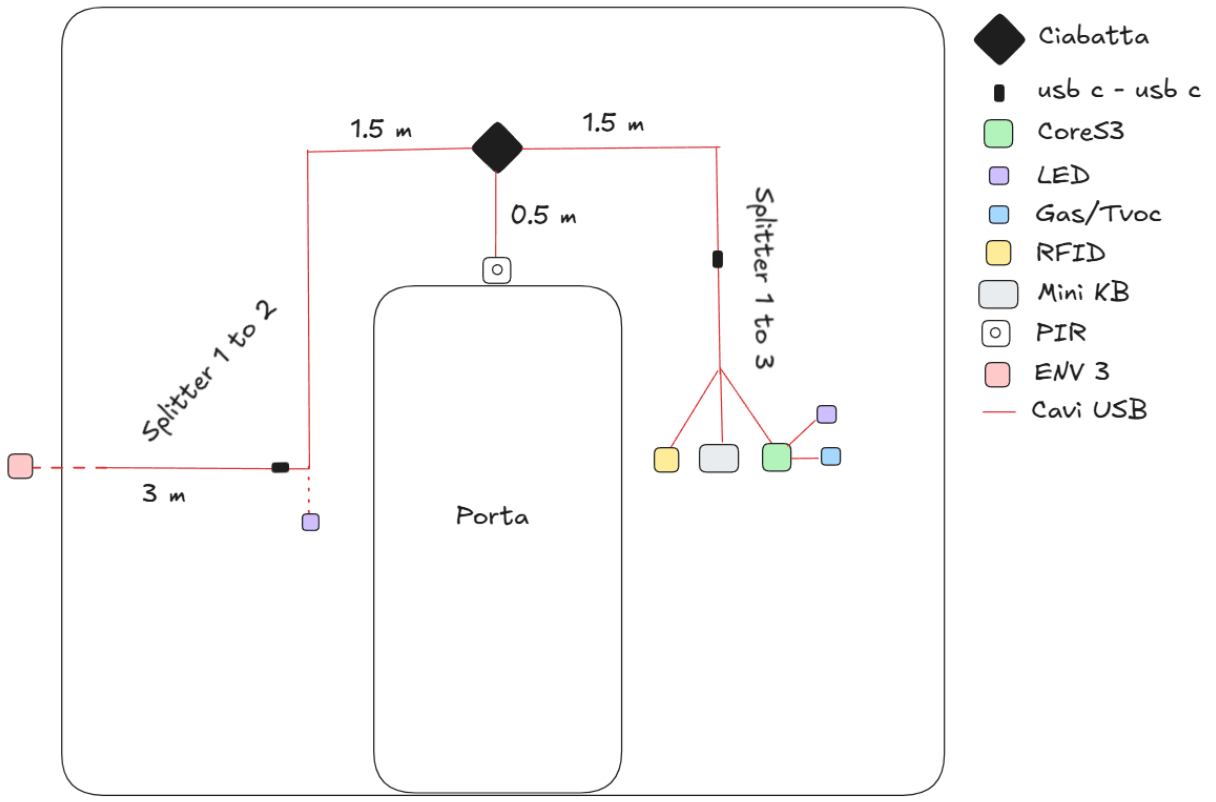
```
if access:
  color = 0xff0000
else:
  color = 0x00ff00
```

Il secondo LED è basato sui dati rilevati dal sensore di Co2, e cambia colore in base alle quantità rilevate nell'aria. Esso è situato vicino al CoreS3, e serve a avvertire un utente che si trova nella sala server quando sarebbe ora di arieggiare il locale. Il colore è verde se il valore rilevato è minore di 800, arancione se maggiore di 800 ma minore di 1200 e rosso quando supera questo valore.

```
if co2 < 800:
  color = 0x00ff00
elif 800 <= co2 <= 1200:
  color = 0xffff00
else:
  color = 0xff0000
```

4.2.9 Montaggio sistema

Di seguito viene riportato come sono stati montati i componenti hardware nella sala server.
Tutti i componenti sono collegati ad una ciabatta centrale, così da permettere l'aggiunta di un sistema di alimentazione come un powerbank per evitare downtime e falle di sicurezza nel sistema.
Inoltre il tutto viene alimentato da cavi usb di diverse lunghezze, come indicato nello schema seguente.



20 Schema di installazione

5 Test

5.1 Protocollo di test

Test Case: **Riferimento**:	TC-001 REQ-001	**Nome:**	Verifica installazione sensori e componenti
Descrizione:	Verificare che tutti i sensori, cavi e controller siano montati correttamente nella sala server.		
Prerequisiti:	Recarsi nella sala server.		
Procedura:	1. Controllare la presenza di tutti i componenti hardware richiesti. 2. Verificare il corretto montaggio dei sensori, cavi e controller nella sala server.		
Risultati attesi:	Tutti i componenti sono installati correttamente e funzionanti.		

Test Case: **Riferimento**:	TC-002 REQ-004	**Nome:**	Accesso Dashboard docente/sistemista
Descrizione:	Accedere alla dashboard di watchtower come docente o sistemista, perciò un utente con permessi elevati.		
Prerequisiti:	-		
Procedura:	1. Aprire https://watchtower.labosamt.ch/login 2. Effettuare il login con username: sistemista1 e password: Password&1		
Risultati attesi:	Login effettuato, nella navbar ci sono le tab per accessi e notifiche e in basso a sinistra sotto all'utente si vede la scritta "docente" rispettivamente "sistemista".		

Test Case: **Riferimento**:	TC-003 REQ-004	**Nome:**	Accesso Dashboard allievo
Descrizione:	Accedere alla dashboard di watchtower come allievo, perciò un utente con permessi ridotti.		
Prerequisiti:	-		
Procedura:	1. Aprire https://watchtower.labosamt.ch/login 2. Effettuare il login con username: allievo2 e password: Password&1		
Risultati attesi:	Login effettuato, nella navbar ci sono solo i tab per navigare ai grafici di temperatura, co2 e umidità e in basso a sinistra sotto all'utente si vede la scritta "allievo".		

Test Case: **Riferimento**:	TC-004 REQ-004	**Nome:**	Dashboard grafici temperatura ultima ora
Descrizione:	Visualizzare i dati dell'ultima ora in un grafico riguardo alla temperatura (o co2 o umidità).		
Prerequisiti:	Avere effettuato l'accesso.		
Procedura:	1. Navigare alla pagina di temperatura (/temperature)		
Risultati attesi:	Si vede un grafico con i dati di temperatura dell'ultima ora, sempre con 5 minuti di distacco.		

Test Case: **Riferimento**:	TC-005 REQ-004	**Nome:**	Dashboard grafici temperatura ultimo giorno
Descrizione:	Visualizzare i dati dell'ultima ora in un grafico riguardo alla temperatura (o co2 o umidità).		
Prerequisiti:	Avere effettuato l'accesso.		
Procedura:	1. Navigare alla pagina di temperatura (/temperature) 2. Cambiare grafico in last day		
Risultati attesi:	Si vede un grafico con i dati di temperatura delle ultime 24 ore.		

Test Case: **Riferimento**:	TC-006 REQ-004	**Nome:**	Dashboard grafici temperatura ultima settimana
Descrizione:	Visualizzare i dati dell'ultima ora in un grafico riguardo alla temperatura (o co2 o umidità).		
Prerequisiti:	Avere effettuato l'accesso.		
Procedura:	1. Navigare alla pagina di temperatura (/temperature) 2. Cambiare grafico in last week		
Risultati attesi:	Si vede un grafico con i dati di temperatura massima di ogni giorno negli scorsi sette giorni.		

Test Case: **Riferimento**:	TC-007 REQ-004	**Nome:**	Dashboard grafico accessi
Descrizione:	Visualizzare il numero di accessi alla sala server.		
Prerequisiti:	Avere effettuato l'accesso come docente/sistemista e avere effettuato un accesso fisico alla sala server da quando il sistema di monitoraggio è in uso.		
Procedura:	1. Navigare alla pagina degli accessi (/access)		
Risultati attesi:	Si vede un grafico con i dati degli accessi nell'ultima settimana, divisi in autorizzati e non. Sotto a questo grafico si vedono inoltre gli ultimi 10 accessi effettuati, con nome, data e motivazione.		

Test Case: **Riferimento**:	TC-008 REQ-004	**Nome:**	Dashboard notifiche
Descrizione:	Visualizzare le notifiche generate dall'applicativo in base ai dati ricevuti dal microcontrollore.		
Prerequisiti:	Avere effettuato l'accesso come docente/sistemista e generare un'anomalia nei dati, per esempio effettuare un accesso non autorizzato.		
Procedura:	1. Navigare alla pagina delle notifiche (/notifications)		
Risultati attesi:	Si vede una lista di notifiche, divise per categoria e per gravità.		

Test Case: **Riferimento**:	TC-009 REQ-004	**Nome:**	Dashboard impostare limite notifiche
Descrizione:	Modificare i valori di soglia per le notifiche.		
Prerequisiti:	Avere effettuato l'accesso come docente/sistemista e recarsi nella pagina delle notifiche.		
Procedura:	1. Premere sul pulsante delle impostazioni. 2. Modificare un qualche valore. 3. Salvare.		
Risultati attesi:	Se si preme nuovamente sul pulsante, il nuovo valore è stato salvato e ora le notifiche verranno generato in base a questo nuovo dato.		

Test Case: **Riferimento**:	TC-010 REQ-004	**Nome:**	Registrazione badge
Descrizione:	Associare il proprio badge scolastico al proprio utente.		
Prerequisiti:	Avere effettuato l'accesso come docente/sistemista e avere il badge scolastico.		
Procedura:	1. Premere sulla card nella home in cui viene richiesto di eseguire un'azione con il badge. 2. Memorizzare il numero scritto. 3. Recarsi in sala server e mettere il badge sul lettore RFID. 4. Inserire il codice memorizzato tramite la tastiera 5. Uscire		
Risultati attesi:	Se si torna sull'applicativo e si fa un refresh, la card dovrebbe essere scomparsa e l'utente è collegato al proprio badge. Inoltre sulla schermata del microcontrollore appare un messaggio di successo di associazione tra badge e utente.		

Test Case: **Riferimento**:	TC-011 REQ-004	**Nome:**	Dashboard risolvi notifiche
Descrizione:	Marcare una notifica come risolta, così che non si più visualizzata.		
Prerequisiti:	Avere effettuato l'accesso come docente/sistemista e avere una notifica nella pagina.		
Procedura:	1. Navigare alla pagina delle notifiche (/notifications) 2. Premere su una notifica 3. Premere Resolved 4. Confermare		
Risultati attesi:	La notifica non è più visibile.		

Test Case: **Riferimento**:	TC-012 REQ-004	**Nome:**	Dashboard risolvi tutte notifiche
Descrizione:	Marcare tette le proprie notifiche come risolte, così che non vengono più visualizzate.		
Prerequisiti:	Avere effettuato l'accesso come docente/sistemista e avere almeno una notifica nella pagina.		
Procedura:	1. Navigare alla pagina delle notifiche (/notifications) 2. Premere su Resolve All 3. Confermare		
Risultati attesi:	Non ci sono più notifiche nella pagina.		

Test Case: **Riferimento**:	TC-013 REQ-005	**Nome:**	Email di notifica
Descrizione:	Quando viene generato una notifica di livello alto, viene inviata una notifica email a docenti e sistemisti.		
Prerequisiti:	Avere un utente con email nel LDAP.		
Procedura:	1. Generare un'allerta di livello alto (accesso non autorizzato) 2. Andare nella propria casella postale.		
Risultati attesi:	C'è un'email inviata da WatchTower di allerta		

Test Case: **Riferimento**:	TC-014 REQ-002	**Nome:**	Raccolta temperatura e umidità
Descrizione:	Il sensore ENV III raccoglie i dati ambientali ogni 5 minuti e li trasmette al microcontrollore, che li mostra e li manda al backend.		
Prerequisiti:	Avere i sensori e microcontrollori montati e accesi, aver effettuato l'acceso tramite badge e tastiera.		
Procedura:	1. Apettare finchè il microcontrollore mostri I dati di temperature e umidità.		
Risultati attesi:	Temperatura e umidità mostrati sullo schermo.		

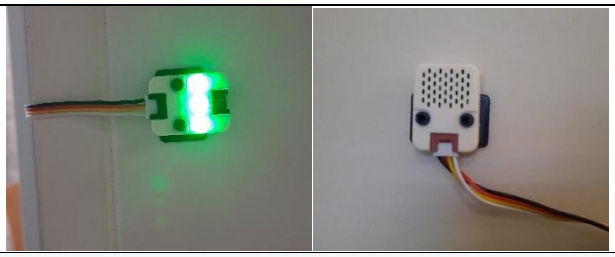
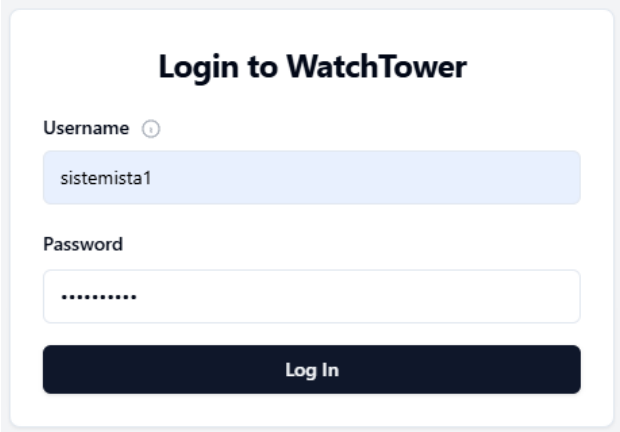
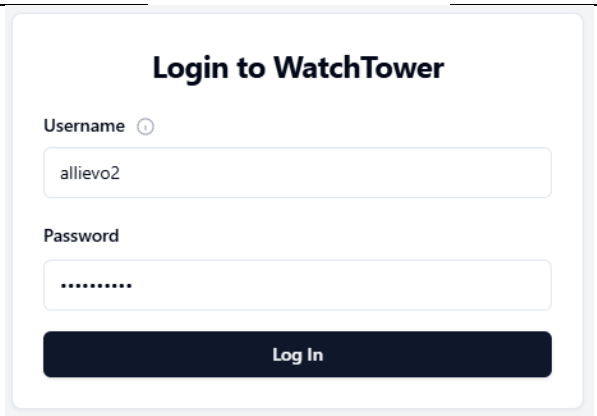
Test Case: **Riferimento**:	TC-015 REQ-002	**Nome:**	Raccolta valore Co2 e accensione LED
Descrizione:	Il sensore di Co2 raccoglie i dati rilevanti e ogni 5 minuti e li trasmette al microcontrollore, che li mostra e li manda al backend e accende un LED in base al valore.		
Prerequisiti:	Avere i sensori e microcontrollori montati e accesi, aver effettuato l'acceso tramite badge e tastiera.		
Procedura:	1. Apettare finchè il microcontrollore mostri il dato di co2 e accenda il LED.		
Risultati attesi:	LED acceso (verde, arancione o roso) e valore Co2 scritto sullo schermo.		

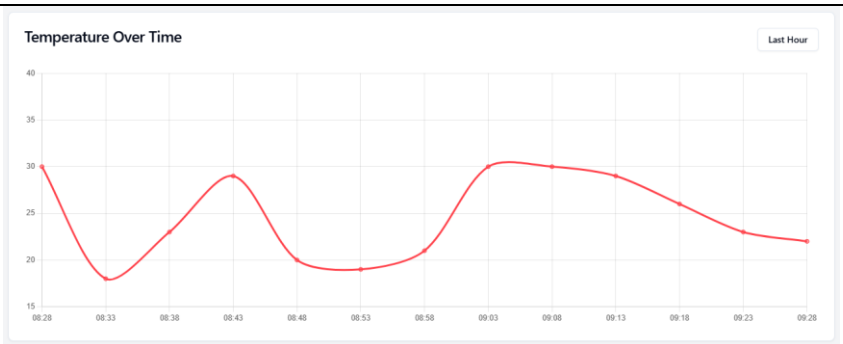
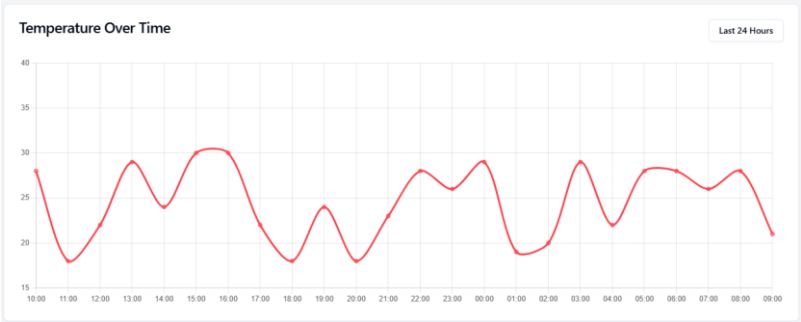
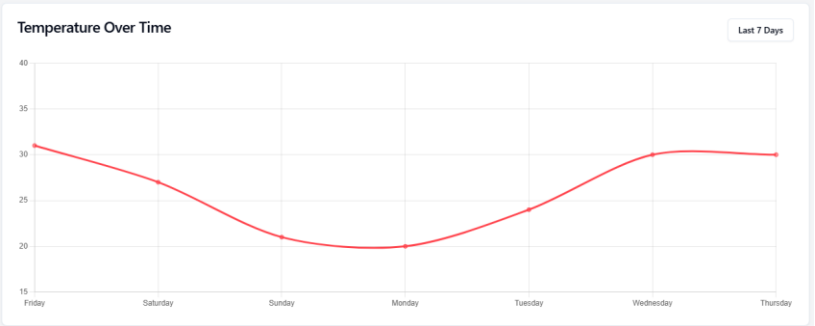
Test Case: **Riferimento**:	TC-016 REQ-002	**Nome:**	Rilevamento accesso consentito
Descrizione:	Il sistema registra gli accessi consentiti alla sala server, con nome, motivo e data.		
Prerequisiti:	Avere i sensori e microcontrollori montati e accesi.		
Procedura:	1. Entrare in sala server 2. Posizionare il badge sul sensore RFID 3. Immettere il motivo di accesso tramite la tastiera (istruzioni sullo schermetto)		
Risultati attesi:	Schermata di successo e registrazione accesso.		

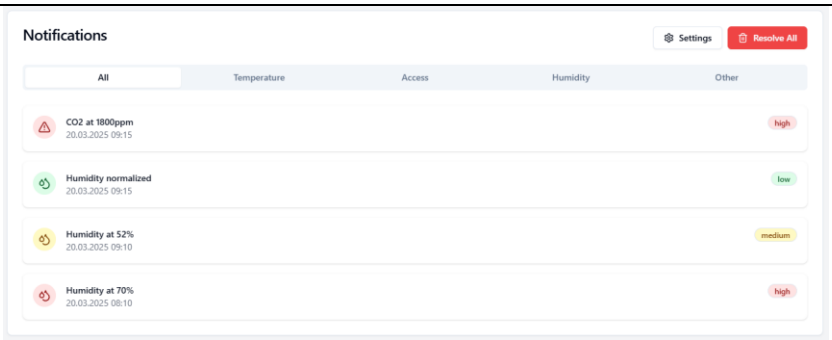
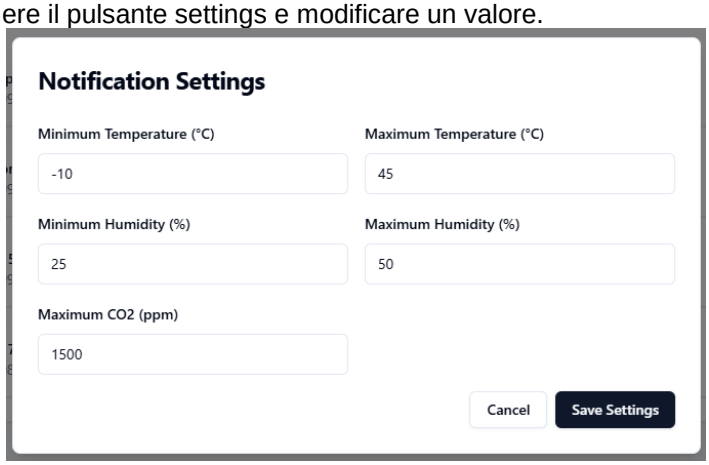
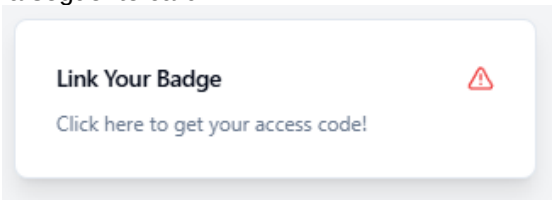
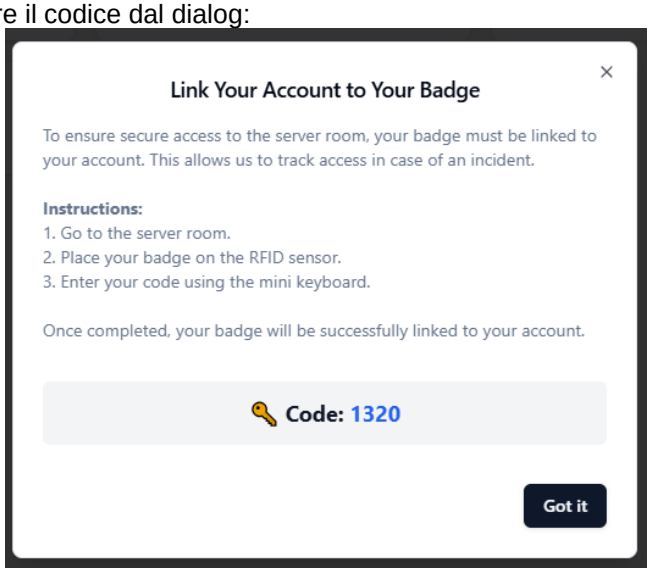
Test Case: **Riferimento**:	TC-017 REQ-002	**Nome:**	Rilevamento accesso non consentito
Descrizione:	Il sistema registra gli accessi non consentiti alla sala server con la data dell'avvenimento.		
Prerequisiti:	Avere i sensori e microcontrollori montati e accesi.		
Procedura:	1. Entrare in sala server 2. Aspettare un minute finché lo schermo mostra il fallimento di autenticazione		
Risultati attesi:	Schermata di fallimento e registrazione accesso.		

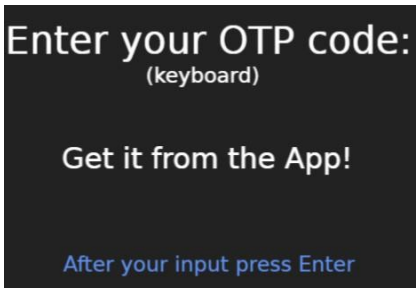
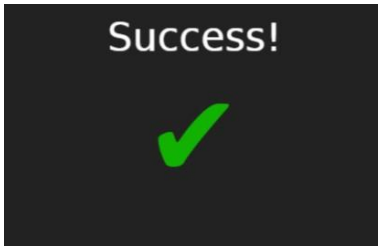
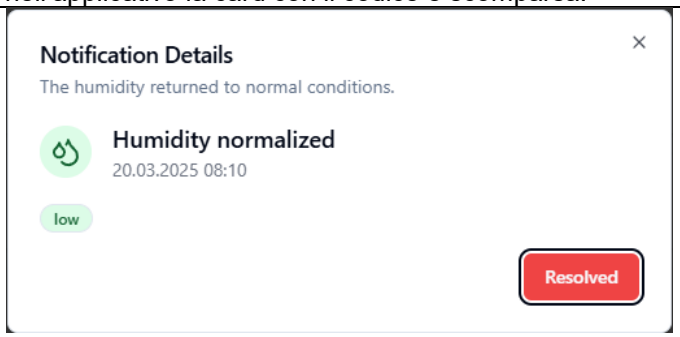
5.2 Risultati test

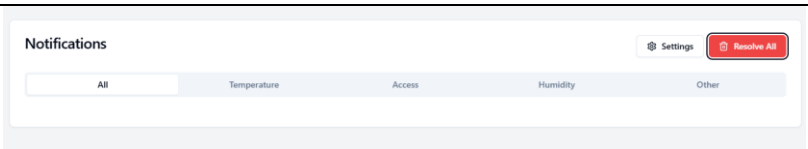
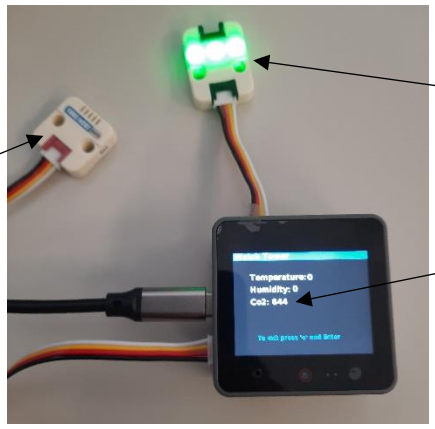
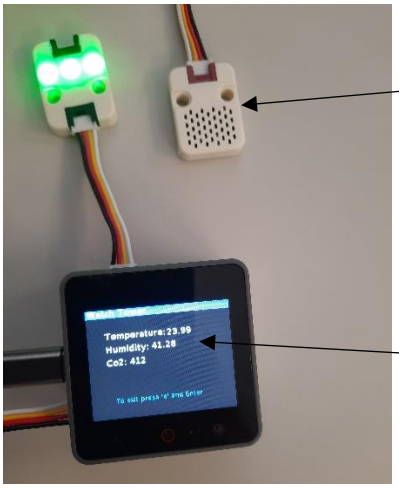
Test Case	Risultato	Data	Stato
TC-001	CoreS3, Mini KB, LED Co2, sensore Co2, RFID Led accessi e sensore temperatura/umidità:	01.04.2025	Passato

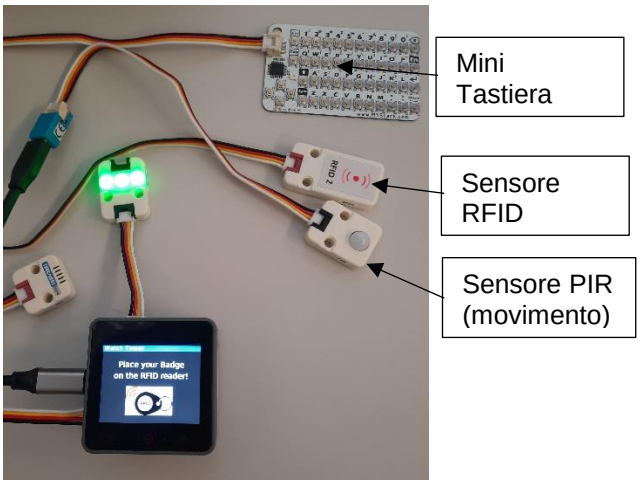
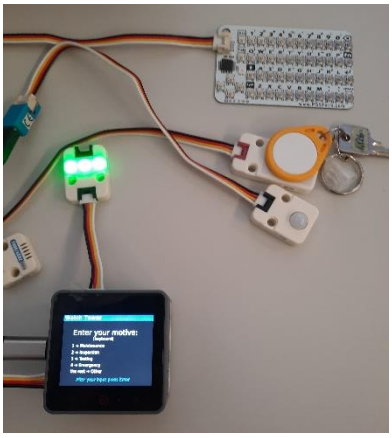
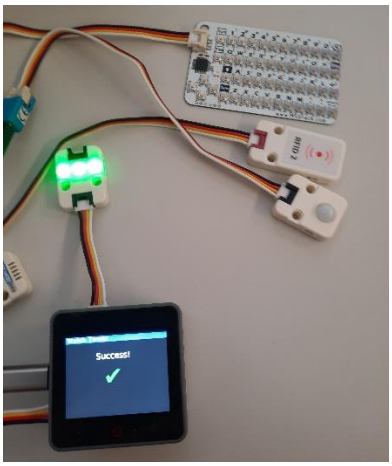
			
TC-002	 Premere su Log In, per passare alla schermata di home. Utente del gruppo sistemista: 	20.03.2025	Passato
TC-003	 Premere su Log In, per passare alla schermata di home. Utente del gruppo allievi: 	20.03.2025	Passato

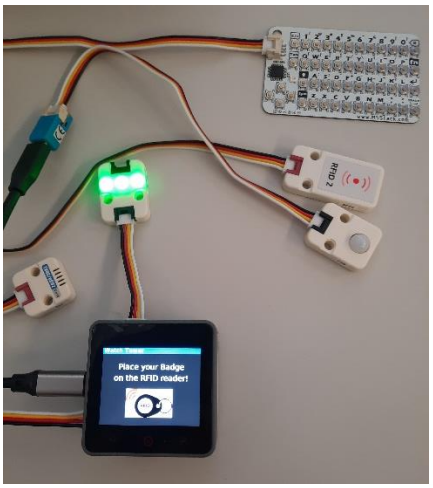
TC-004		20.03.2025	Passato
TC-005		20.03.2025	Passato
TC-006		20.03.2025	Passato
TC-007	Grafico: Tabella: Recent Access Name / Time / Motive / Status sistemista2 / 20.03.2025 10:30 / Maintenance / Authorized sistemista1 / 14.03.2025 10:30 / Maintenance / Authorized - / 14.03.2025 10:30 / - / Unauthorized - / 14.03.2025 10:30 / - / Unauthorized	20.03.2025	Passato

TC-008		20.03.2025	Passato
TC-009	Premere il pulsante settings e modificare un valore.  Salvare.	20.03.2025	Passato
TC-010	Premere sulla seguente card:  Prendere il codice dal dialog:  Appoggiare il badge sul sensore:	20.03.2025	Passato

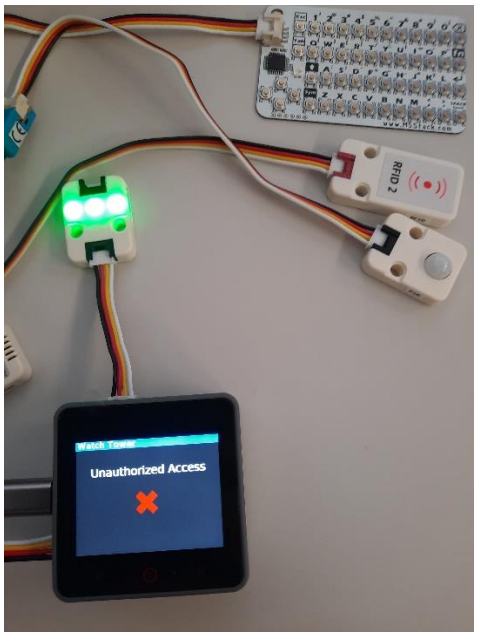
	 Viene mostrata la seguente schermata sul microcontrollore:  Inserire il codice e premere invio, ora vien mostrata la seguente schermata:  Anche nell'applicativo la card con il codice è scomparsa.		
TC-011	 Premere Resolved. La notifica è rimossa dalla lista.	20.03.2025	Passato
TC-012	 Premere Resolve All e confermare. Tutte le notifiche sono rimosse.	20.03.2025	Passato

			
TC-013	Dopo aver generato un accesso non autorizzato, ho ricevuto la seguente email: Alert Notification **Unauthorized access attempt** Someone entered the server room but didn't authenticate himselves. Please take appropriate action if necessary. This is an automated message. Please do not reply to this email.	20.03.2025	Passato
TC-014		20.03.2025	Passato
TC-015		20.03.2025	Passato
TC-016	Entrare nella sala server.	20.03.2025	Passato

	 Posizionare il badge sul sensore.  Inserire il motivo.  Accesso eseguito con successo.		
TC-017	Entrare in sala server.	20.03.2025	Passato



Dopo 1 minuto:



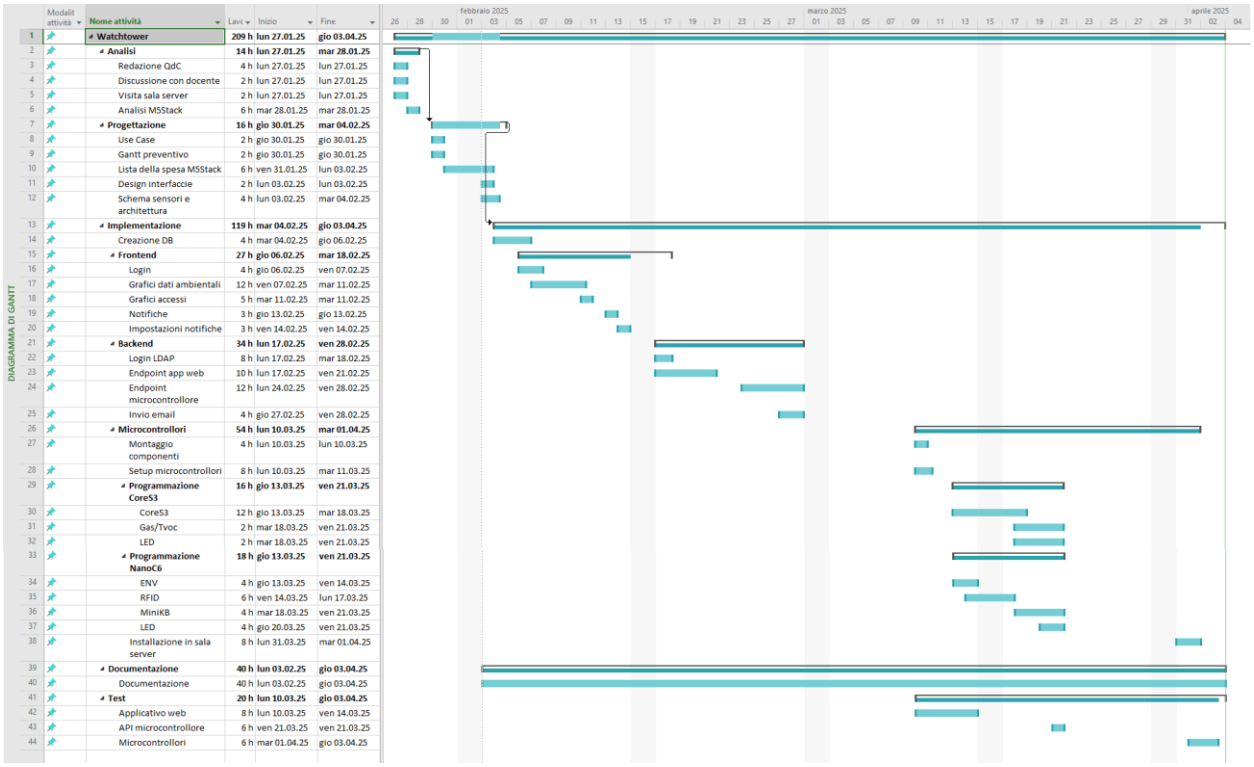
Accesso non autorizzato rilevato.

5.3 Mancanze/limitazioni conosciute

A causa della mancanza di tempo, non sono riuscito a implementare una funzionalità che permetta al sistema di rilevare quando un sensore o un componente smette di rispondere o non funziona più, generando un'allerta.
Purtroppo mi sono accorto solo alla fine che sarebbe stato più opportuno gestire la comunicazione in modo da inviare e attendersi una risposta dal destinatario nell'architettura dei microcontrollori, per evitare eventuali perdite di dati, ma mentre testavo alla postazione non ho mai avuto problemi del genere, e perciò non ho pensato a questo problema. Una volta installati nella sala server ho velocemente realizzato il problema, ma il tempo era ormai troppo poco per rimediare.

6 Consuntivo

La mia progettazione iniziale rispetta abbastanza bene il lavoro effettivo. Ovviamente certe task hanno richiesto e altre meno, e nel Gantt consuntivo ho potuto specificare più task eseguite, ma nel complesso la mia pianificazione è stata abbastanza accurata.



	Modalità attività	Nome attività	Lavo	Inizio	Fine
1		**Watchtower**	**209 h**	**lun 27.01.25**	**gio 03.04.25**
2		**Analisi**	**14 h**	**lun 27.01.25**	**mar 28.01.25**
3		Redazione QdC	4 h	lun 27.01.25	lun 27.01.25
4		Discussione con docente	2 h	lun 27.01.25	lun 27.01.25
5		Visita sala server	2 h	lun 27.01.25	lun 27.01.25
6		Analisi M5Stack	6 h	mar 28.01.25	mar 28.01.25
7		**Progettazione**	**16 h**	**gio 30.01.25**	**mar 04.02.25**
8		Use Case	2 h	gio 30.01.25	gio 30.01.25
9		Gantt preventivo	2 h	gio 30.01.25	gio 30.01.25
10		Lista della spesa M5Stack	6 h	ven 31.01.25	lun 03.02.25
11		Design interfaccie	2 h	lun 03.02.25	lun 03.02.25
12		Schema sensori e architettura	4 h	lun 03.02.25	mar 04.02.25
13		**Implementazione**	**119 h**	**mar 04.02.25**	**gio 03.04.25**
14		Creazione DB	4 h	mar 04.02.25	gio 06.02.25
15		**Frontend**	**27 h**	**gio 06.02.25**	**mar 18.02.25**
16		Login	4 h	gio 06.02.25	ven 07.02.25
17		Grafici dati ambientali	12 h	ven 07.02.25	mar 11.02.25
18		Grafici accessi	5 h	mar 11.02.25	mar 11.02.25
19		Notifiche	3 h	gio 13.02.25	gio 13.02.25
20		Impostazioni notifiche	3 h	ven 14.02.25	ven 14.02.25
21		**Backend**	**34 h**	**lun 17.02.25**	**ven 28.02.25**
22		Login LDAP	8 h	lun 17.02.25	mar 18.02.25
23		Endpoint app web	10 h	lun 17.02.25	ven 21.02.25
24		Endpoint microcontrollore	12 h	lun 24.02.25	ven 28.02.25
25		Invio email	4 h	gio 27.02.25	ven 28.02.25
26		**Microcontrollori**	**54 h**	**lun 10.03.25**	**mar 01.04.25**
27		Montaggio componenti	4 h	lun 10.03.25	lun 10.03.25
28		Setup microcontrollori	8 h	lun 10.03.25	mar 11.03.25
29		**Programmazione CoreS3**	**16 h**	**gio 13.03.25**	**ven 21.03.25**
30		CoreS3	12 h	gio 13.03.25	mar 18.03.25
31		Gas/Tvoc	2 h	mar 18.03.25	ven 21.03.25
32		LED	2 h	mar 18.03.25	ven 21.03.25
33		**Programmazione NanoC6**	**18 h**	**gio 13.03.25**	**ven 21.03.25**
34		ENV	4 h	gio 13.03.25	ven 14.03.25
35		RFID	6 h	ven 14.03.25	lun 17.03.25
36		MiniKB	4 h	mar 18.03.25	ven 21.03.25
37		LED	4 h	gio 20.03.25	ven 21.03.25
38		Installazione in sala server	8 h	lun 31.03.25	mar 01.04.25
39		**Documentazione**	**40 h**	**lun 03.02.25**	**gio 03.04.25**
40		Documentazione	40 h	lun 03.02.25	gio 03.04.25
41		**Test**	**20 h**	**lun 10.03.25**	**gio 03.04.25**
42		Applicativo web	8 h	lun 10.03.25	ven 14.03.25
43		API microcontrollore	6 h	ven 21.03.25	ven 21.03.25
44		Microcontrollori	6 h	mar 01.04.25	gio 03.04.25

21 Diagramma di Gantt consuntivo

7 Conclusioni

Questo progetto è abbastanza unico, dato che integra una parte di sviluppo web con dello sviluppo IoT, cosa che lo rende molto interessante. Il prodotto che ne risulta è molto specifico per le necessità della sala server del CPT di Trevano, in quanto utilizza il suo LDAP per funzionare e anche il montaggio è su misura per la sala server. Se utilizzata correttamente, è sicuramente un'aggiunta utile e interessante per la sicurezza fisica della scuola e potrebbe essere utilizzato per sensibilizzare i futuri allievi.

7.1 Sviluppi futuri

Aggiungere sorveglianza video, per migliorare la sicurezza degli accessi.
Collegare l'intero sistema ad un'alimentazione ridondante, in modo da segnalare che c'è stata una interruzione di corrente tramite una notifica e per garantire la continuità del monitoraggio.
Integrare un sistema di sorveglianza video per migliorare la sicurezza degli accessi.
Collegare l'intero sistema a un'alimentazione ridondante, in modo da rilevare eventuali interruzioni di corrente, inviare una notifica di avviso ai responsabili e garantire la continuità del monitoraggio.
Si potrebbe anche aggiungere una gestione più dettagliata degli utenti che hanno eseguito una registrazione dei badge, in modo da poterli eliminare/modificare.

7.2 Considerazioni personali

Questo progetto si è rivelato estremamente interessante, poiché mi ha permesso di provare nuove tecnologie, come la programmazione di microcontrollori tramite Python e l'utilizzo dei prodotti M5Stack per la realizzazione della componente fisica. Allo stesso tempo, ho avuto modo di applicare competenze già acquisite, come lo sviluppo con Node.js e Vue.js. Nonostante alcune difficoltà iniziali nella creazione delle liste della spesa e nella comprensione della programmazione dei microcontrollori, questo progetto di semestre mi ha permesso di apprendere molto e sono soddisfatto del risultato ottenuto.

8 Glossario

Termine	Descrizione
AES-128	**AES:** è uno degli algoritmi di crittografia simmetrica più sicuri e utilizzati al mondo. La versione AES-128 utilizza una chiave di 128 bit per cifrare e decifrare i dati, garantendo la protezione delle informazioni contro intercettazioni e manipolazioni.
API	**Application Programming Interface**: insieme di strumenti e procedure che consentono a diversi software di comunicare tra loro.
CRUD	**Create Read Update Delete:** Sono le quattro operazioni fondamentali per la gestione dei dati in un sistema informatico. La creazione consente di inserire nuovi dati in un database, mentre la lettura permette di recuperarli e visualizzarli. L'aggiornamento serve a modificare le informazioni già presenti, e l'eliminazione consente di rimuoverle definitivamente.
Endpoint	**Endpoint:** Punto di accesso a un'API o a un servizio, che consente la comunicazione tra client e server.
ESP-NOW	**ESP-NOW:** è un protocollo di comunicazione wireless sviluppato da Espressif Systems per dispositivi basati su ESP32. Permette una trasmissione dati a bassa latenza tra dispositivi senza la necessità di una rete Wi-Fi tradizionale.
Express	**Express**: Framework web minimalista per Node.js che semplifica la creazione di API e applicazioni web, fornendo un sistema di routing e middleware flessibile.
Firmware	**Firmware:** Software incorporato in un dispositivo hardware che fornisce le istruzioni fondamentali per il funzionamento del dispositivo. È memorizzato in una memoria non volatile e gestisce operazioni a basso livello, come l'inizializzazione dell'hardware e la gestione delle risorse.
I²C	**Inter Integrated Circuit**: sistema di comunicazione seriale bifilare utilizzato tra circuiti integrati. Il classico bus I²C è composto da almeno un master ed un slave.
IoT	**Internet of Things:** Insieme di dispositivi connessi a Internet che trasmettono informazioni per migliorare la propria usabilità o fornire informazioni ad altri sistemi. Normalmente si tratta di sensori, microcontrollori, veicoli, ed elettrodomestici.
JWT	**JSON Web Token**: standard web per lo scambio di dati definito dalla RFC 7519 proposto nel 2015. Viene spesso impiegato nei sistemi di autenticazione per garantire accessi sicuri senza la necessità di memorizzare sessioni sul server. Grazie alla sua natura stateless, il JWT è molto utilizzato per l'autenticazione e autorizzazione nelle API REST.

LDAP	**Lightweight Directory Access Protocol** : protocollo di rete per l'accesso e la gestione di directory distribuite su una rete. Viene utilizzato principalmente per autenticare utenti e gestire informazioni in organizzazioni, come liste di utenti, gruppi e permessi.
M5Stack	**M5Stack:** piattaforma di sviluppo modulare basata su ESP32, progettata per applicazioni IoT e automazione. Offre una serie di dispositivi e moduli espandibili con un design compatto e integrato, permettendo agli sviluppatori di creare progetti hardware in modo efficiente.
Node.js	**Node.js**: Ambiente di runtime JavaScript, progettato per eseguire codice JavaScript lato server in modo asincrono e ad alte prestazioni.
Shadcn Vue	**Shadcn Vue**: libreria di componenti grafici che utilizza vue e Tailwind per creare componenti grafici accattivanti.
Tailwind CSS	**Tailwind CSS:** framework CSS open source usato per applicare velocemnte uno stile alle pagine HTML.
Vue JS	**Vue.js:** un framework JavaScript open-source per la costruzione di interfacce utente e applicazioni web

9 Bibliografia

9.1 Sitografia

- https://uiflow-micropython.readthedocs.io/ *Documentazione di uiflow (diverse pagine), -*
- https://www.espressif.com/en/solutions/low-power-solutions/esp-now *Spiegazione di ESP now, 18-02-2025*
- https://vuejs.org/guide/introduction.html *Documentazione officiale di VueJS (diverse pagine), -*
- https://www.shadcn-vue.com/ *Sito di riferimento per Shadcn per VueJS (diverse pagine), -*
- https://lucide.dev/guide/packages/lucide-vue-next *Icone per siti web con VueJS, 10-02-2025*
- https://chatgpt.com/ *LLM (diverse chat, utilizzata per riscrivere testi, cercare informazioni, correggere errori nel codice del sito web, risolvere problemi nella creazione del sito web, sia backend che frontend, soprattutto per le funzionalità di CRUD), -*
- https://v0.dev/ *LLM (diverse chat, utilizzate per creare una grafica per il sito composta da componenti Shadcn), -*
- https://www.reichelt.com/ *Rivenditore di elettronica (diverse pagine), 21-03-2025*
- https://www.distrelec.ch/de/ *Rivenditore di elettronica (diverse pagine), 28-01-2025*
- https://m5.8266.de/index.php *Sito di appassionati di M5Stack, informazioni utili, 31-01-2025*
- https://de.wikipedia.org/wiki/Advanced_Encryption_Standard *Articolo Wikipedia su AES, 03-04-2025*

10 Indice delle Figure

1 Use Case 6
2 Diagramma di Gantt preventivo 7
3 Schema Microcontrollore e Sensori 9
4 Architettura sistema 9
5 Diagramma ER 10
6 Struttura interfacce 10
7 Protocollo LDAP 11
8 Struttura cartelle frontend 14
9 Pagina Home 15
10 Pagina Temperatura 15
11 Struttura cartelle backend 16
12 Email di notifica 19
13 M5burner esempio firmware per CoreS3 20
14 M5Stack CoreS3 21
15 Schermate CoreS3 21
16 M5Stack NanoC6 22
17 Architettura M5Stack implementata 22
18 Schema funzionamento badge 23
19 Interfaccia registrazione badge, con istruzioni 23
20 Schema di installazione 26
21 Diagramma di Gantt consuntivo 39

11 Allegati

- Diari di lavoro
- Applicativo
- Quaderno dei compiti
- Abstract
- Gantt (preventivo e consuntivo)
- Use Case
- Lista della spesa
- Schemi sensori, ecc.
- Guida d’uso
- Poster