WatchTower

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 6

2.5 Analisi dei mezzi 7

2.5.1 Software 7

2.5.2 Hardware 7

3 Progettazione 8

3.1 Design dell’architettura del sistema 8

3.2 Design dei dati e database 9

3.3 Design delle interfacce 9

3.4 Design procedurale 10

3.4.1 Connessione LDAP 10

4 Implementazione 11

4.1 Applicativo Web 11

4.1.1 Installazione frontend 11

4.1.2 Installazione backend 11

4.1.3 Frontend 12

4.1.4 Backend 14

4.1.5 Container 15

4.2 Microcontrollore 15

4.2.1 CoreS3 16

4.2.2 NanoC6 16

5 Test 17

5.1 Protocollo di test 17

5.2 Risultati test 18

5.3 Mancanze/limitazioni conosciute 18

6 Consuntivo 18

7 Conclusioni 18

7.1 Sviluppi futuri 18

7.2 Considerazioni personali 18

8 Glossario 18

9 Bibliografia 19

9.1 Bibliografia per articoli di riviste: 19

9.2 Bibliografia per libri 19

9.3 Sitografia 19

10 Allegati 19

11 Indice delle Figure 19

# Introduzione

## 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 – 03.04.2025
* Data di presentazione: 07 – 11.04.2025

## 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.

## 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.

# Analisi

## 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.

## 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 registrare gli accessi tramite RFID. |
| **003** | Deve rilevare movimento attraverso il sensore PIR. |
| **004** | 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. |
| **002** | Definire un tempo di mantenimento dei dati. |

|  |  |
| --- | --- |
| **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. |
| **005** | Deve includere un log degli eventi principali. |

|  |  |
| --- | --- |
| **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. |

## Use case

|  |
| --- |
| 1 Use Case |

## Pianificazione

|  |
| --- |
| 2 Diagramma di Gantt preventivo |

## Analisi dei mezzi

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

### 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

### 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
* Led
* Hub

# Progettazione

## 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 “PROTOCOLLO”, 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 |

## 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 |

## 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 |

## Design procedurale

### 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 |

# Implementazione

## Applicativo Web

### 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

### 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*

### 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 |

#### 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>

#### 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.

### 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.

#### 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();

  });

}

#### Log

In questo progetto ho dato importanza anche ai log, utilizzando Winston per la loro creazione, in questo modo posso gestire la gravità degli eventi e categorizzarli in 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:

Codice:

logger.info(`API call to /${type}/lastHour`);

Risultato:

14-02-2025, 11:19:17 [INFO]: API call to /temperature/lastHour

#### Salvataggio dati

Gli endpoint per salvare dati nel DB sono preceduti da /api, e verranno chiamata 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.

#### 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);

});

### 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.

## 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.

|  |
| --- |
| 12 M5burner esempio firmware per CoreS3 |

### CoreS3

Il 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 a questo protocollo di comunicazione wireless, che consente una trasmissione veloce ed efficiente dei dati tra dispositivi senza la necessità di una rete Wi-Fi. Perciò 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.

|  |
| --- |
| M5Stack CoreS3 ESP32S3 Kit di Sviluppo IoT  13 M5Stack CoreS3 |

### NanoC6

I microcontrollori M5Stack NanoC6 sono delle versioni in miniatura del CoreS3, con meno funzionalità, perciò questi hanno il compito di raccogliere i dati dai sensori o dagli input e trasmetterli al CoreS3, oppure di ricevere dati da esso e decidere in base a questi se accendere un LED o se aspettarsi un input dalla tastiera.

|  |
| --- |
| M5Stack M5STACK M5NanoC6 Scheda di sviluppo IoT ESP32-C6 WiFi-6 Matte Three  Zigbee RISC-V 160 MHz : Amazon.it: Videogiochi  14 M5Stack NanoC6 |

### 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 chiama 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 stesa 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. Da questo cambiamento risulta il seguente schema.

|  |
| --- |
| 15 Architettura M5Stack implementata |

### Registrazione badge

Per registrare meglio gli accessi nella sala server, è necessario sapere chi è entrato e quando. L’informazione di chi è entrato 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, perciò risulta necessario salvare questo link 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 e 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.

|  |
| --- |
| 16 Schema funzionamento badge |

|  |
| --- |
| 17 Interfaccia registrazione badge, con istruzioni |

### Comunicazione wireless

Per fare comunicare i microcontrollori NanoC6 con il microcontrollore CoreS3, ho optato per il protocollo ESPnow, creato da Espressif, che permette di fare delle comunicazioni point to point (unicast) tramite i mac address dei dispositivi. Inoltre 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.

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.

### 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 🡪 3 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

# Test

## Protocollo di test

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-012 | **Nome:** | Import a card with KIC, KID and KIK keys, but not shown with the GUI |
| **Descrizione:** | Import a card with KIC, KID and KIK keys with no obfuscation, but not shown with the GUI | | |
| **Prerequisiti:** | Store on local PC: Profile\_1.2.001.xml (appendix n\_n) and Cards\_1.2.001.txt (appendix n\_n).  PIN (OTA\_VIEW\_PIN\_PUK\_KEY) and ADM (OTA\_VIEW\_ADM\_KEY) user right not set. | | |
| **Procedura:** | 1. Go to “Cards manager” menu,  in main page click “Import Profiles” link, Select the “1.2.001.xml” file, Import the Profile 2. Go to “Cards manager” menu,  in main page click “Import Cards” link, Select the “1.2.001.txt” file, Delete the cards,  Select the “1.2.001.txt” file, Import the cards 3. Research the “41795924770” Card, Click the imsi card link Check the card details 4. Execute the SQL: SELECT imsi, dir, keyset, cntr, rawtohex(kickey), rawtohex(kidkey), rawtohex(kikkey), rawtohex(chv), rawtohex(dap)FROM otacardkey a where imsi='340041795924770' ORDER BY keyset; | | |
| **Risultati attesi:** | Keys visible in the DB (OtaCardKey) but not visible in the GUI (Card details) | | |

## Risultati test

Tabella riassuntiva in cui si inseriscono i test riusciti e non del prodotto finale. Se un test non riesce e viene corretto l’errore, questo dovrà risultare nel documento finale come riuscito (la procedura della correzione apparirà nel diario), altrimenti dovrà essere descritto l’errore con eventuali ipotesi di correzione.

## Mancanze/limitazioni conosciute

Descrizione con motivazione di eventuali elementi mancanti o non completamente implementati, al di fuori dei test case. Non devono essere riportati gli errori e i problemi riscontrati e poi risolti durante il progetto.

# Consuntivo

Consuntivo del tempo di lavoro effettivo e considerazioni riguardo le differenze rispetto alla pianificazione (cap. 1.7) (ad esempio Gantt consuntivo).

# Conclusioni

Quali sono le implicazioni della mia soluzione? Che impatto avrà? Cambierà il mondo? È un successo importante? È solo un’aggiunta marginale o è semplicemente servita per scoprire che questo percorso è stato una perdita di tempo? I risultati ottenuti sono generali, facilmente generalizzabili o sono specifici di un caso particolare? ecc.

## Sviluppi futuri

Migliorie o estensioni che possono essere sviluppate sul prodotto.

## Considerazioni personali

Cosa ho imparato in questo progetto? ecc.

# Glossario

|  |  |
| --- | --- |
| **Termine** | **Descrizione** |
| 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. |
| JWT | **JSON Web Token**: standard web per lo scambio di dati definito dalla RFC 7519 proposto nel 2015. |
| Vue JS | **Vue.js:** un framework JavaScript open-source per la costruzione di interfacce utente e applicazioni web |
| Tailwind CSS | **Tailwind CSS:** framework CSS open source usato per applicare velocemnte uno stile alle pagine HTML. |
| Shadcn Vue | **Shadcn Vue**: libreria di componenti grafici che utilizza vue e Tailwind per creare componenti grafici accattivanti. |
| Node.js | **Node.js**: Ambiente di runtime JavaScript, progettato per eseguire codice JavaScript lato server in modo asincrono e ad alte prestazioni. |
| 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. |
| API | **Application Programming Interface**: insieme di strumenti e procedure che consentono a diversi software di comunicare tra loro. |
| I²C | **Inter Integrated Circuit**: sistema di comunicazione seriale bifilare utilizzato tra [circuiti integrati](https://it.wikipedia.org/wiki/Circuito_integrato). Il classico bus I²C è composto da almeno un master ed uno slave. |

# Bibliografia

## Bibliografia per articoli di riviste:

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo dell’articolo (tra virgolette),
3. Titolo della rivista (in italico),
4. Anno e numero
5. Pagina iniziale dell’articolo.

## Bibliografia per libri

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo del libro (in italico),
3. ev. Numero di edizione,
4. Nome dell’editore,
5. Anno di pubblicazione,
6. ISBN.

## Sitografia

1. URL del sito (se troppo lungo solo dominio, evt completo nel diario),
2. Eventuale titolo della pagina (in italico),
3. Data di consultazione (GG-MM-AAAA).

**Esempio:**

* http://standards.ieee.org/guides/style/section7.html, *IEEE Standards Style Manual*, 07-06-2008.

# Allegati

* Diari di lavoro
* Applicativo
* Quaderno dei compiti
* Abstract
* Gantt
* Use Case
* …

# Indice delle Figure

[1 Use Case 6](#_Toc191021194)

[2 Diagramma di Gantt preventivo 6](#_Toc191021195)

[3 Schema Microcontrollore e Sensori 8](#_Toc191021196)

[4 Architettura sistema 8](#_Toc191021197)

[5 Diagramma ER 9](#_Toc191021198)

[6 Struttura interfacce 9](#_Toc191021199)

[7 Protocollo LDAP 10](#_Toc191021200)

[8 Struttura cartelle frontend 12](#_Toc191021201)

[9 Pagina Home 13](#_Toc191021202)

[10 Pagina Temperatura 13](#_Toc191021203)

[11 Struttura cartelle backend 14](#_Toc191021204)

[12 M5burner esempio firmware per CoreS3 16](#_Toc191021205)

[13 M5Stack CoreS3 16](#_Toc191021206)

[14 M5Stack NanoC6 17](#_Toc191021207)

[15 Architettura M5Stack implementata 17](#_Toc191021208)