

Università degli Studi di Bologna
Corso di Laurea Magistrale in Informatica
Laboratorio di Making

Prototipo IoT per il controllo degli stalli della Velostazione di Bologna

Relazione di progetto

Youssef Awni Barty Fahmi Hanna
Matricola **0001132285**

Indice

1	Introduzione	3
1.1	La Velostazione e lo spazio fisico	3
1.2	Problema affrontato	5
1.3	Obiettivi del progetto	5
1.3.1	Panoramica del flusso end-to-end	5
1.4	Modello logico RFID: tag come “gettone”	6
2	Analisi del problema e requisiti	7
2.1	Requisiti funzionali	7
2.2	Requisiti non funzionali	8
3	Architettura del sistema	9
3.1	Panoramica generale	9
3.2	Scelta della tecnologia LoRa	9
3.3	Gestione degli errori e robustezza del sistema	10
4	Progettazione hardware	12
4.1	Nodo di stallo: Heltec + RC522 + buzzer	12
4.2	Postazione di desk su Raspberry Pi	15
4.2.1	Desk RFID per l’assegnazione dei tag	15
4.2.2	Gateway LoRa per la ricezione degli eventi	18
5	Logica dei nodi IoT	19
5.1	Nodo di stallo: gestione RFID e anti-spam	19
6	Servizi sul Raspberry Pi	20
6.1	Gateway LoRa - HTTP	20
6.2	API Desk RFID	20
7	Backend Django e integrazione IoT	22
7.1	Modello dati	22
7.2	Logica degli eventi IoT	22

7.3 Rilascio del parcheggio e liberazione del tag	23
8 Frontend Angular e flusso utente	24
8.1 Interazione con il desk RFID	24
8.2 Creazione e gestione dei parcheggi	25
9 Discussione finale, analisi dei consumi e sviluppi futuri	26
9.1 Analisi dei consumi energetici	26
9.2 Sviluppi futuri e passaggio alla produzione	27
10 Appendice tecnica	28
10.1 Codice dei nodi IoT	28
10.1.1 Invio di un evento <code>rfid_scan</code>	28
10.1.2 Logica di anti-spam per la lettura RFID	28
10.1.3 Codice del gateway Heltec RX	29
10.2 Servizi sul Raspberry Pi	30
10.2.1 Gateway LoRa → HTTP	30
10.2.2 API Desk RFID	31
10.3 Snippet di codice aggiuntivi	31
10.3.1 Pattern di beep sul nodo Heltec	31
10.4 Comandi essenziali	32
10.5 Repository e figure	33

Capitolo 1

Introduzione

1.1 La Velostazione e lo spazio fisico

Questo progetto nasce all'interno di un lavoro più ampio dedicato alla nuova Velostazione di Bologna¹, per la quale è stato sviluppato un gestionale web per la gestione di biciclette, clienti, tariffe e parcheggi.

La Velostazione occupa gli spazi sotto la scalinata monumentale del Pincio, affacciata su via Indipendenza. L'ambiente è composto da sequenza di locali con destinazioni d'uso differenti: aree tecniche, spazi per il pubblico, zone di servizio e, soprattutto, una vasta area destinata al deposito delle biciclette.

La planimetria di progetto, riportata in Figura 1.1, mostra la suddivisione degli spazi. Nella parte inferiore si sviluppa il grande *Deposit biciclette*, organizzato in blocchi di portabici paralleli e numerati, all'interno dei quali vengono collocati gli stalli gestiti dal sistema.

Al centro della porzione superiore è collocato il locale **Box Office**, che costituisce il principale punto di contatto con il pubblico. Qui l'operatore registra ingressi e uscite, gestisce abbonamenti e tariffe e interagisce con la piattaforma software. Nell'ambito di questo progetto il Box Office è anche il luogo in cui viene allestita la postazione IoT del desk: un banco con Raspberry Pi², lettore RFID³, e buzzer⁴, collegato alla rete interna e al backend Django⁵.

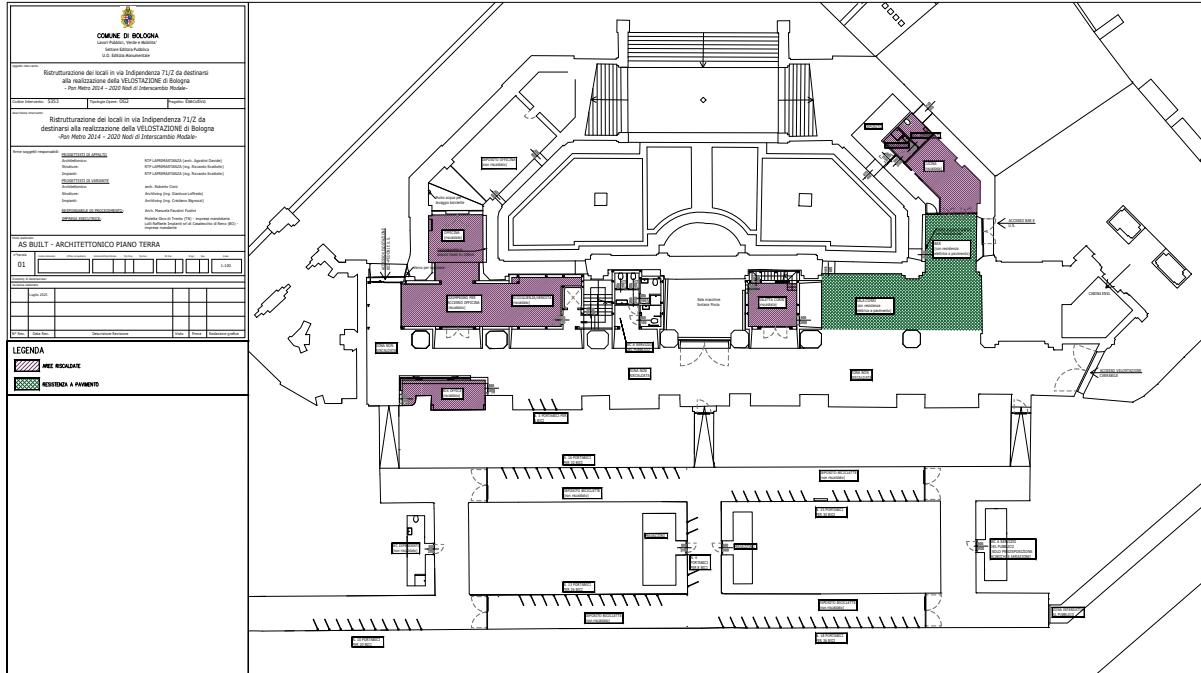
¹Velostazione di Bologna, collegamento Google Maps: <https://maps.google.com/?q=Velostazione+di+Bologna>

²Raspberry Pi 4 Model B con 2 GB di RAM, single-board computer utilizzato come postazione di desk per la lettura dei tag RFID e l'esecuzione dei servizi locali. Sito ufficiale: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/>

³Modulo RFID RC522, lettore a 13.56 MHz compatibile con tag MIFARE, utilizzato per l'acquisizione degli UID dei tag. Pagina prodotto: <https://www.az-delivery.de/it/products/rfid-set>

⁴Buzzer attivo a 3.3 V utilizzato per fornire un feedback acustico immediato all'operatore durante le operazioni di lettura RFID.

⁵Framework web Python utilizzato per l'implementazione del backend applicativo, responsabile della logica di dominio, della persistenza dei dati e dell'integrazione con i dispositivi IoT.



1.2 Problema affrontato

Nel gestionale ogni parcheggio è rappresentato come una tripla (*bici, stallo, intervallo di tempo*). Al momento della creazione di un parcheggio attivo, il sistema stabilisce che una data bicicletta X deve occupare, ad esempio, lo stallo denominato P001 a partire da un certo istante e fino al rilascio.

Nulla impedisce però all'utente di collocare la bici in un portabici diverso da quello previsto, o di occupare uno stallo con una bici non registrata. In uno spazio come il deposito biciclette della Velostazione, dove sono presenti decine di portabici nello stesso ambiente, la discrepanza tra stato logico e stato fisico rischia di diventare un problema operativo rilevante, soprattutto nelle ore di punta.

Il problema centrale affrontato dal progetto è l'allineamento affidabile tra lo stato logico dei parcheggi, gestito dal software al Box Office, e lo stato fisico degli stalli nel deposito. In particolare, è necessario disporre di un meccanismo che verifichi automaticamente che la bicicletta effettivamente presente in uno specifico stallo fisico coincida con la bicicletta e lo stallo che il gestionale considera assegnati in quel momento.

Per raggiungere questo obiettivo, il prototipo combina tag RFID e nodi LoRa installati sugli stalli, integrati con il backend Django già esistente, che si occupa di ricevere e interpretare gli eventi generati nel deposito.

1.3 Obiettivi del progetto

Gli obiettivi specifici del prototipo sono:

- progettare un nodo IoT di stallo, basato su scheda Heltec WiFi LoRa 32 v3, in grado di leggere tag RFID e inviare eventi al Box Office;
- allestire una postazione di desk RFID al Box Office (Raspberry Pi + RC522 + buzzer) per l'assegnazione controllata dei tag alle biciclette;
- realizzare un gateway LoRa su Raspberry che trasformi i messaggi radio dei nodi in richieste HTTP verso il backend Django;
- estendere il modello dati del gestionale con entità dedicate all'IoT (`LoRaDevice`, `IoTEvent`, ecc.) e con un campo `rfid_uid` sulle biciclette;
- integrare il frontend Angular con il flusso di assegnazione del tag e con la visualizzazione dello stato IoT dei parcheggi.

1.3.1 Panoramica del flusso end-to-end

Il flusso complessivo tra Box Office e Deposito biciclette si articola nelle seguenti fasi:

1. **Creazione del parcheggio al desk:** dal Box Office l'operatore crea un parcheggio attivo indicando bici, stallo (ad esempio P001) e tipologia di tariffa, e assegnando

un tag RFID tramite un lettore collegato al Raspberry Pi. Il backend memorizza l'UID del tag per quel parcheggio.

2. **Parcheggio nel deposito:** l'utente si sposta nella zona di Deposito biciclette e parcheggia la bici nello stallone fisico corrispondente. Il nodo IoT montato sul portabici legge il tag tramite RC522 e invia via LoRa un evento `rfid_scan`, indicando `node_id`, numero di sequenza e `rfid_uid`.
3. **Gateway e backend:** il gateway su Raspberry riceve il pacchetto LoRa, lo traduce in una richiesta HTTP e lo inoltra al backend Django. Il backend risale al dispositivo (`LoRaDevice`) a partire dal `node_id`, verifica quale stallone gli è associato e confronta questa informazione con i parcheggi attivi per la bici identificata dall'UID.
4. **Esito IoT:** in base al risultato del confronto, il backend registra un `IoTEvent` (ad esempio `ok`, `mismatch`, `unknown_rfid`) e aggiorna lo stato del parcheggio. Il frontend mostra il risultato all'operatore tramite badge colorati nella lista dei parcheggi.
5. **Rilascio:** al momento del ritiro, il parcheggio viene chiuso al Box Office; il backend marca lo stallone come libero e azzerà il campo `rfid_uid` della bici, rendendo il tag nuovamente disponibile.

1.4 Modello logico RFID: tag come “gettone”

Alla base dell'integrazione IoT c'è una scelta di modellazione: il tag RFID non viene usato come identità permanente della bicicletta, ma come *gettone* temporaneo di accesso al deposito. In pratica al Box Office l'operatore associa un tag a una bici solo quando sta per creare (o ha appena creato) un parcheggio attivo, poi durante la vita del parcheggio il campo `rfid_uid` della bici è valorizzato e consente al backend di riconoscere gli eventi provenienti dagli stalloni. Infine, quando il parcheggio viene chiuso, lo stesso metodo di rilascio si occupa di azzerare il campo `rfid_uid`, liberando automaticamente il tag. Questa impostazione permette ai tag di poter essere riutilizzati facilmente.

Nei capitoli successivi vengono spiegati nodi di stallone, servizi su Raspberry Pi, estensioni del backend Django e schermate di frontend per mostrare il flusso tra Box Office e Deposito biciclette.

Capitolo 2

Analisi del problema e requisiti

2.1 Requisiti funzionali

A partire dall’analisi del contesto operativo e del flusso di lavoro descritto nel capitolo introduttivo, emergono una serie di requisiti funzionali che il sistema IoT deve soddisfare per garantire la coerenza tra stato fisico degli stalli e stato logico dei parcheggi gestiti dal software.

In primo luogo, ogni stallo fisico del deposito deve essere associato in modo univoco a un nodo IoT, installato in prossimità del portabici, in grado di leggere un tag RFID e di generare eventi verso il sistema centrale. Tale associazione consente di ricondurre ogni evento osservato nello spazio fisico a uno specifico stallo logico del gestionale.

Ogni evento generato da un nodo IoT deve includere un insieme minimo di informazioni strutturate:

- un identificativo del nodo (`node_id`), che rappresenta l’identità logica del dispositivo IoT e permette al backend di risalire allo stallo associato;
- un numero di sequenza (`seq`), incrementato localmente dal nodo, utilizzato per individuare e scartare eventuali duplicazioni di eventi;
- un tipo di evento (`event_type`), che descrive la natura dell’azione osservata dal nodo. Nel prototipo il tipo di evento principale è `rfid_scan`, utilizzato per segnalare la lettura di un tag RFID presso uno stallo;
- un identificativo RFID (`rfid_uid`)

Per evitare la registrazione multipla dello stesso evento, il sistema deve implementare un meccanismo di deduplicazione basato sulla coppia (`device`, `seq`), imponendo un vincolo di unicità che garantisca che ciascun evento venga processato una sola volta.

Nel caso specifico degli eventi di tipo `rfid_scan`, il backend deve infine essere in grado di classificare l’esito della lettura confrontando lo stato fisico osservato con lo stato logico del gestionale. In particolare, il sistema deve distinguere i seguenti casi:

- `unknown_rfid`: il tag letto non è associato a nessuna bicicletta conosciuta dal sistema;
- `no_active_parking`: il tag è valido, ma la bicicletta associata non ha un parcheggio attivo al momento della lettura;
- `mismatch`: la bicicletta è correttamente registrata e ha un parcheggio attivo, ma risulta collocata in uno stallo diverso da quello assegnato;
- `ok`: la bicicletta è parcheggiata nello stallo fisico corrispondente a quello assegnato dal gestionale.

2.2 Requisiti non funzionali

Oltre ai requisiti funzionali, il progetto è guidato da una serie di requisiti non funzionali legati alle caratteristiche fisiche della Velostazione, ai vincoli di installazione e alla sostenibilità del sistema nel tempo.

In particolare, il sistema deve ridurre al minimo il cablaggio necessario negli spazi del deposito, al fine di semplificare l'installazione e limitare l'impatto sull'infrastruttura esistente. La comunicazione tra nodi di stallo e sistema centrale deve quindi avvenire prevalentemente via radio.

È inoltre necessario garantire una copertura affidabile del segnale anche in presenza di strutture murarie, ostacoli e affollamento, tipici di un ambiente sotterraneo come quello della Velostazione.

Dal punto di vista energetico, i nodi IoT devono mantenere un consumo sufficientemente contenuto da rendere possibile, in prospettiva, un'alimentazione a batteria o soluzioni ibride, riducendo la necessità di interventi di manutenzione.

Infine, la complessità della logica dei nodi di stallo deve essere mantenuta il più possibile limitata, delegando al backend la logica di interpretazione degli eventi. Questa scelta semplifica lo sviluppo.

Capitolo 3

Architettura del sistema

3.1 Panoramica generale

Il prototipo che viene proposto in questo progetto comprende i seguenti componenti:

1. **Desk operatore:** Raspberry Pi con lettore RC522 e buzzer, esposto tramite un microservizio Flask (`desk_rfid_api.py`);
2. **Nodi di stallo:** Heltec WiFi LoRa 32 v3 con RC522 e buzzer, uno per ciascuno stallo fisico (nel prototipo: uno stallo);
3. **Gateway LoRa:** seconda Heltec configurata in ricezione e collegata al Raspberry via USB;
4. **Servizi Raspberry:** script Python `gateway_lora_to_http.py` per il forwarding degli eventi verso Django;
5. **Stack web:** backend Django con le estensioni IoT e frontend Angular generato da OpenAPI.

Il flusso informativo principale è il seguente: al desk l'operatore assegna un tag RFID a una bici tramite lettura RC522 e chiamata al backend; l'operatore crea quindi un parcheggio attivo (bici + stallo logico). Quando la bici viene parcheggiata nello stallo con nodo IoT, quest'ultimo legge il tag e genera un evento `rfid_scan` via LoRa. Il gateway riceve il messaggio e lo inoltra al backend Django via HTTP; il backend interpreta l'evento, aggiorna la tabella `IoTEvent` e restituisce lo stato, che viene mostrato dal frontend nella vista dei parcheggi. La Figura 3.1 mostra l'architettura del prototipo, organizzata in quattro livelli principali: deposito biciclette, gateway/desk su Raspberry Pi, backend web e frontend utilizzato dall'operatore al Box Office.

3.2 Scelta della tecnologia LoRa

Per il collegamento tra nodi di stallo e gateway è stata scelta la tecnologia LoRa a 868 MHz, è risultata sin da subito un'ottima soluzione perché presenta maggiore robustezza alle



Figura 3.1: Architettura logica del prototipo IoT per il controllo degli stalli della Velostazione.

interferenze rispetto al WiFi. Poi LoRa è progettato appositamente per dispositivi IoT a bassissimo consumo energetico, permettendo loro di funzionare per anni con una singola batteria, mentre il Wi-Fi è concepito per alte velocità di trasferimento dati, il che comporta un maggiore assorbimento energetico.

3.3 Gestione degli errori e robustezza del sistema

Poiché il sistema integra dispositivi IoT distribuiti, comunicazioni radio e servizi web, sono stati introdotti diversi meccanismi di gestione degli errori con l'obiettivo di rendere il flusso complessivo robusto a duplicazioni, messaggi malformati e tentativi di accesso non autorizzati.

A livello di **nodo di stallo**, è implementata una logica di anti-spam per la lettura RFID. Il nodo mantiene uno stato locale relativo all'ultimo tag rilevato e invia un evento `rfid_scan` solo quando viene rilevato un nuovo UID oppure quando un tag, precedentemente rimosso, viene nuovamente appoggiato sul lettore. In questo modo si evita la generazione continua di eventi nel caso in cui una bici rimanga ferma nello stallo per un periodo prolungato.

Ogni evento generato dal nodo include inoltre un **numero di sequenza incrementale (seq)**, che viene aumentato a ogni nuovo evento valido. Questo campo consente di distinguere in modo univoco gli eventi provenienti dallo stesso dispositivo e costituisce la base per la gestione delle duplicazioni a valle.

Il **gateway LoRa** su Raspberry Pi introduce un ulteriore livello di filtraggio. Lo script di forwarding analizza le stringhe ricevute dalla porta seriale e considera validi solo i messaggi che rispettano il formato atteso e contengono i campi obbligatori (`node`, `seq`, `ev`). Le linee non conformi o incomplete vengono scartate e registrate nei log, senza essere inoltrate al backend.

Per ogni messaggio valido, il gateway costruisce una richiesta HTTP verso il backend Django includendo un header `X-Device-Key`. Tale header contiene una chiave segreta associata al dispositivo sorgente e permette al backend di autenticare il nodo che ha generato l'evento. In questo modo, anche se l'endpoint HTTP fosse raggiungibile dall'esterno, solo i dispositivi registrati e autorizzati possono produrre eventi IoT accettati dal sistema.

Sul **backend**, la robustezza è garantita da ulteriori controlli. Ogni evento viene associato a un dispositivo logico e viene verificata l'unicità della coppia (`device`, `seq`). Se un evento con lo stesso numero di sequenza risulta già presente nel database, esso viene classificato come duplicato e ignorato, evitando aggiornamenti ripetuti o incoerenti dello stato degli stalli.

Infine, per gli eventi RFID, il backend valuta la coerenza tra lo stato fisico osservato e lo stato logico del gestionale, classificando l'esito dell'evento (ad esempio tag sconosciuto, assenza di un parcheggio attivo o corrispondenza corretta). Questo approccio consente di segnalare eventuali anomalie operative senza compromettere la consistenza dei dati principali e mantenendo separata la logica di validazione dal livello hardware.

Capitolo 4

Progettazione hardware

4.1 Nodo di stallo: Heltec + RC522 + buzzer

Ogni nodo di stallo è realizzato con una scheda Heltec WiFi LoRa 32 v3, un lettore RFID RC522 e un buzzer attivo. La Figura 4.1 mostra la scheda Heltec utilizzata nel prototipo.

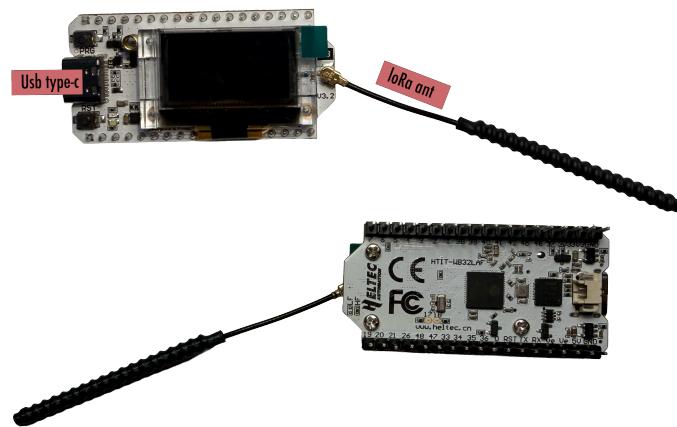


Figura 4.1: Scheda Heltec WiFi LoRa 32 v3 utilizzata come nodo di stallo.

Il lettore RC522 è collegato direttamente alla scheda Heltec tramite cavetti jumper. Questa configurazione, adottata nel prototipo, consente una rapida installazione e facilita le operazioni di test e riconfigurazione del nodo. Nella Figura 4.2 è mostrato il modulo RC522 utilizzato.

Il buzzer è posizionato in modo da essere udibile dall'utente nel momento in cui la bicicletta viene correttamente inserita nello stallo, così da fornire un feedback acustico immediato sull'esito della lettura RFID e sull'invio dell'evento. La Figura 4.3 mostra il componente utilizzato.

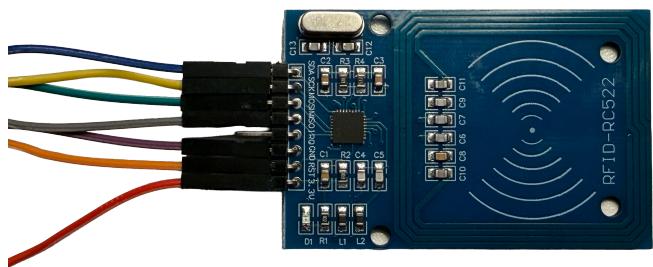


Figura 4.2: Modulo RC522 utilizzato per la lettura dei tag RFID sul nodo di stallo.



Figura 4.3: Buzzer attivo 3.3V collegato al nodo Heltec.

Nel prototipo il nodo di stallo è alimentato tramite un power bank USB collegato alla porta Type-C della scheda Heltec. La mappatura dei pin per la radio LoRa (SX1262) segue le specifiche della scheda e della libreria RadioLib:

Segnale	GPIO Heltec
NSS / CS	GPIO 8
DIO1 / IRQ	GPIO 14
RESET	GPIO 12
BUSY	GPIO 13
SCK	GPIO 9
MISO	GPIO 11
MOSI	GPIO 10

Tabella 4.1: Mappatura pin LoRa SX1262 sulla Heltec WiFi LoRa 32 v3

La Tabella 4.2 mostra come sono collegati lettore RC522 e la scheda Heltec.

Segnale RC522	Segnale logico	GPIO Heltec
3.3V	Alimentazione	3V3
GND	Ground	GND
SCK	SPI SCK	GPIO 36
MOSI	SPI MOSI	GPIO 35
MISO	SPI MISO	GPIO 34
SDA / SS	Chip Select	GPIO 33
RST	Reset	GPIO 21

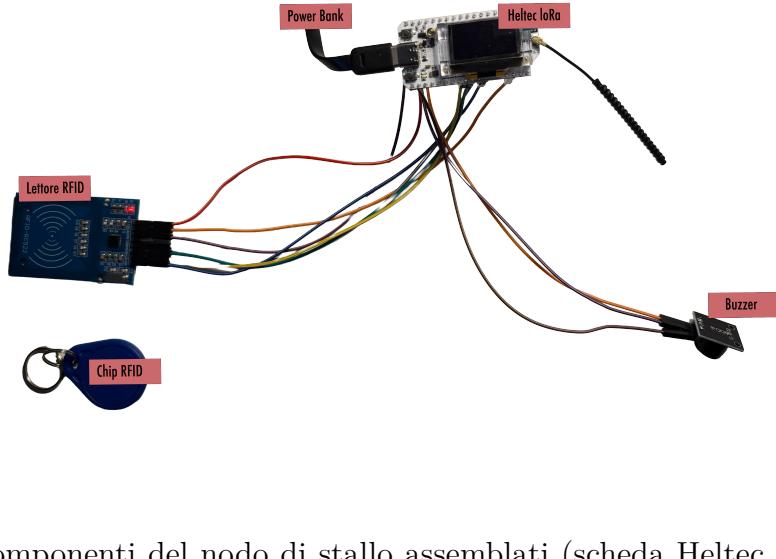
Tabella 4.2: Mappatura RC522 sul nodo Heltec (GPIO ESP32-S3)

Invece la Tabella 4.3 mostra come il buzzer è collegato ai pin della scheda Heltec:

Segnale buzzer	Descrizione	Collegamento
S (signal)	Ingresso logico	GPIO 45
+ (VCC)	Alimentazione	3V3
- (GND)	Ground	GND

Tabella 4.3: Mappatura buzzer sul nodo Heltec

La Figura 4.4 mostra il nodo di stallo assemblato in configurazione da banco, con tutti i componenti collegati (scheda Heltec, lettore RC522 e buzzer), mentre la Figura 4.5 ne illustra la disposizione reale nello spazio del deposito, installato in prossimità di un portabici e alimentato tramite power bank durante la fase di prototipazione.



S

Figura 4.4: Componenti del nodo di stallo assemblati (scheda Heltec, lettore RC522 e buzzer) in configurazione da banco.

4.2 Postazione di desk su Raspberry Pi

Al Box Office è presente un'unica postazione fisica basata su Raspberry Pi, che svolge due funzioni distinte all'interno del sistema: da un lato l'assegnazione dei tag RFID alle biciclette, dall'altro la ricezione degli eventi LoRa provenienti dai nodi di stallo nel deposito.

La Figura 4.6 mostra la postazione completa, comprensiva del Raspberry Pi, del lettore RFID RC522, del buzzer e del gateway LoRa collegato via USB. Nella figura è possibile distinguere visivamente (dai colori) le due parti funzionali della postazione, che verranno descritte separatamente nelle sezioni seguenti.

4.2.1 Desk RFID per l'assegnazione dei tag

La prima funzione della postazione di desk è l'assegnazione controllata dei tag RFID alle biciclette. A questo scopo il Raspberry Pi è collegato a un lettore RC522 e a un buzzer, che fornisce un feedback acustico all'operatore durante le operazioni di lettura del tag.

La mappatura dei pin del lettore RC522, utilizzata nel prototipo in modalità BOARD, è riportata nella Tabella 4.4.

Il buzzer è collegato al Raspberry Pi e viene utilizzato per segnalare l'inizio della lettura, l'avvenuta acquisizione del tag o eventuali errori:

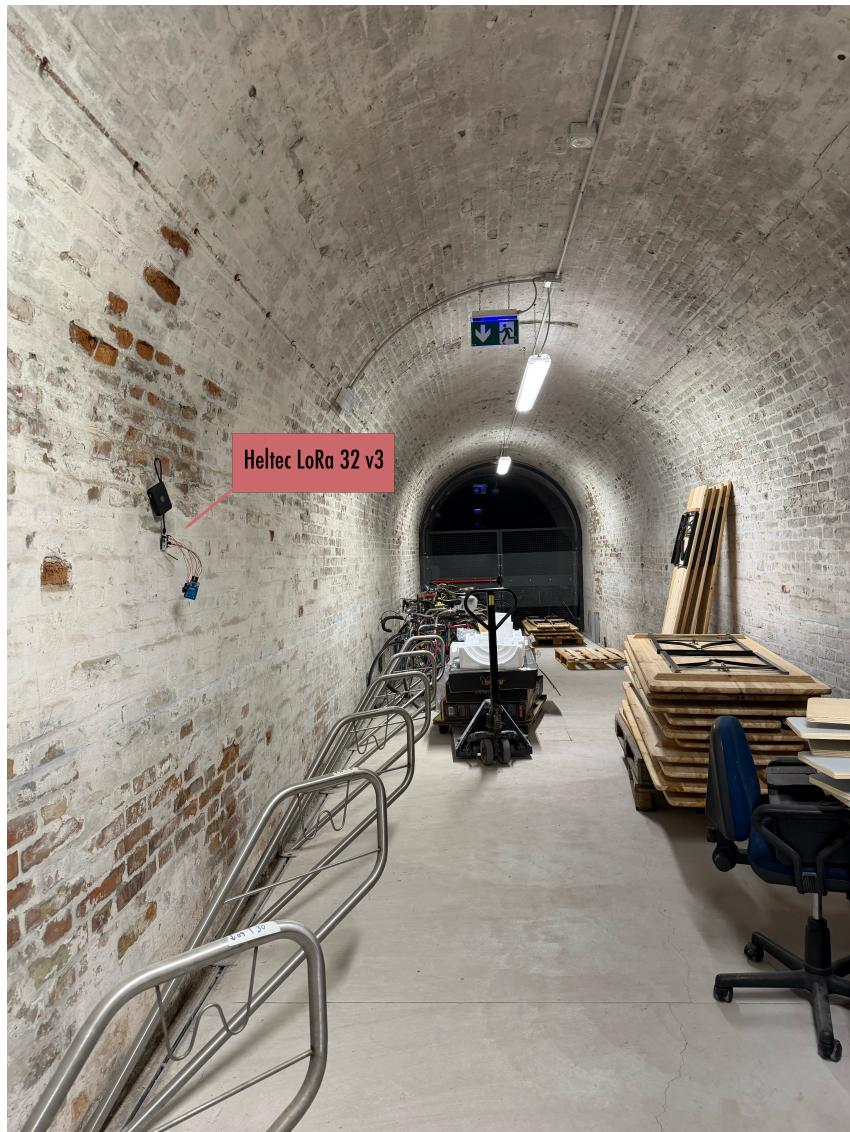


Figura 4.5: Nodo di stallo assemblato e installato sul portabici (Heltec + RC522 + buzzer), alimentato tramite power bank.

Segnale RC522	Raspberry (pin fisico)	Note
3.3V	Pin 1	Alimentazione 3.3V
GND	Pin 6	Ground
SCK	Pin 23	GPIO11, SPI0 SCLK
MOSI	Pin 19	GPIO10, SPI0 MOSI
MISO	Pin 21	GPIO9, SPI0 MISO
SDA / SS	Pin 24	GPIO8, SPI0 CE0
RST	Pin 22	GPIO25

Tabella 4.4: Mappatura RC522 su Raspberry Pi

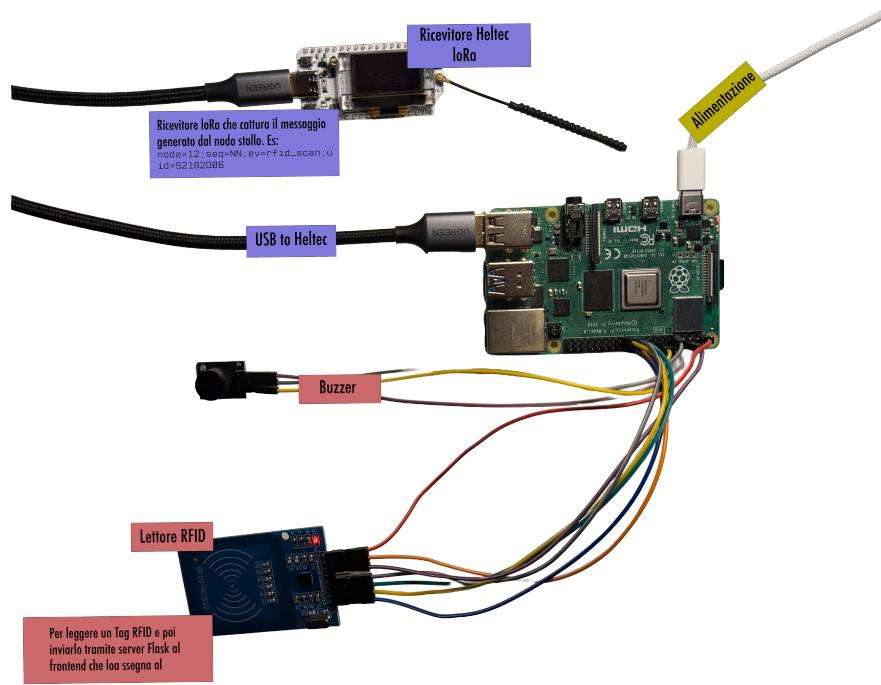


Figura 4.6: Postazione di desk al Box Office basata su Raspberry Pi, utilizzata sia per l'assegnazione dei tag RFID sia come gateway LoRa per la ricezione degli eventi dagli stalli.

Segnale buzzer	Raspberry (pin fisico)	Note
S (signal)	Pin 12	GPIO18 in BCM
+ (VCC)	3.3V	Alimentazione
- (GND)	GND	Ground

Tabella 4.5: Mappatura buzzer su Raspberry Pi

4.2.2 Gateway LoRa per la ricezione degli eventi

La seconda funzione della postazione di desk è quella di gateway LoRa. A questo scopo una scheda Heltec, configurata in modalità di ricezione, è collegata al Raspberry Pi tramite USB.

Dal punto di vista hardware il cablaggio è minimale: la scheda Heltec è alimentata direttamente dalla porta USB del Raspberry Pi ed espone una porta seriale virtuale, utilizzata dal software per ricevere i messaggi LoRa provenienti dai nodi di stallo installati nel deposito.

Capitolo 5

Logica dei nodi IoT

5.1 Nodo di stallo: gestione RFID e anti-spam

Il codice del nodo di stallo, sviluppato in Arduino C++, è responsabile dell'inizializzazione dei moduli LoRa e RFID (RC522), della lettura continua dei tag e dell'invio degli eventi verso il sistema centrale.

Ogni nodo è identificato da una costante `ID_STALL0`, coerente con il campo `node_id` del modello `LoRaDevice` nel backend. Questo identificativo consente di associare in modo stabile ciascun evento generato a uno specifico stallo logico del gestionale, indipendentemente da parametri di rete come indirizzi IP o MAC.

Gli eventi vengono trasmessi come messaggi strutturati, contenenti l'identità del nodo, un numero di sequenza incrementale, il tipo di evento e, quando presente, l'UID del tag RFID letto. Un esempio di messaggio generato dal nodo di stallo è il seguente:

```
node=12;seq>NN;ev=rfid_scan;uid=52182D06
```

L'invio degli eventi è accompagnato da un feedback tramite buzzer, che segnala all'utente il corretto invio del messaggio.

Per evitare la generazione ripetuta di eventi nel caso in cui un tag rimanga fermo sul lettore, è implementata una logica di anti-spam basata su tre elementi: l'ultimo UID rilevato, lo stato di presenza del tag e un intervallo di cooldown temporale. Questa strategia consente di distinguere tra una permanenza del tag sul lettore e una nuova azione intenzionale di parcheggio.

Il codice completo del nodo di stallo, comprensivo delle funzioni di invio degli eventi e della logica di anti-spam, è riportato in Appendice [10.1](#).

Capitolo 6

Servizi sul Raspberry Pi

6.1 Gateway LoRa - HTTP

Il gateway LoRa rappresenta il punto di collegamento tra la rete radio dei nodi di stallo e il backend applicativo. Dal punto di vista software, questo ruolo è svolto da uno script Python eseguito su Raspberry Pi.

Il servizio apre una connessione seriale verso la scheda Heltec configurata in ricezione e legge in modo continuo le righe di testo corrispondenti ai messaggi LoRa ricevuti. Da ciascuna linea viene estratto il payload dell'evento, ignorando eventuali informazioni aggiuntive come il valore di RSSI.

Il payload viene quindi convertito in una struttura dati JSON conforme all'endpoint `/api/core/iot/events` del backend Django. Per gli eventi di tipo `rfid_scan`, il servizio include anche il campo `rfid_uid`.

Ogni richiesta HTTP inviata al backend include un header di autenticazione (`X-Device-Key`), che consente di associare in modo sicuro ciascun evento al dispositivo che lo ha generato. Il servizio gestisce inoltre i casi di errore e registra le risposte del backend per finalità di debugging e monitoraggio.

Il codice completo del gateway LoRa–HTTP è riportato in Appendice [10.2](#).

6.2 API Desk RFID

Al Box Office è presente un servizio dedicato alla lettura dei tag RFID, implementato tramite Flask¹ e ospitato sul Raspberry Pi. Questo servizio espone un'endpoint HTTP utilizzata dal frontend per acquisire in modo controllato l'UID di un tag RFID.

L'endpoint principale `/api/rfid/read-once` effettua una lettura bloccante: una volta ricevuta la richiesta, il servizio attende che l'operatore appoggi un tag sul lettore RC522,

¹Flask, framework web scritto in Python, utilizzato per esporre un servizio REST leggero sul Raspberry Pi per la lettura dei tag RFID. Documentazione ufficiale: <https://flask.palletsprojects.com/>.

fornendo un feedback acustico tramite buzzer sia all'inizio dell'operazione sia al termine della lettura.

L'UID letto viene codificato in formato esadecimale e restituito al frontend come risposta JSON. Questo approccio consente di separare la gestione dell'hardware RFID dalla logica applicativa del backend, mantenendo il sistema modulare e facilmente estendibile.

Il codice completo del microservizio di desk RFID è riportato in Appendice [10.2](#).

Capitolo 7

Backend Django e integrazione IoT

7.1 Modello dati

Il backend Django riutilizza i modelli già presenti per la gestione di biciclette, stalli e parcheggi, estendendoli per supportare l'integrazione con i dispositivi IoT. In particolare, il modello `Bicicletta` include un campo opzionale `rfid_uid`, utilizzato come identificativo temporaneo del tag RFID assegnato alla bici durante il periodo di parcheggio.

Per rappresentare i dispositivi fisici, è stato introdotto il modello `LoRaDevice`, che consente di mappare ciascun `node_id` a uno stallone logico del sistema e a una chiave API dedicata. Questa associazione permette al backend di riconoscere e autenticare in modo univoco i nodi che inviano eventi.

Gli eventi provenienti dal livello IoT vengono infine registrati nel modello `IoTEvent`, che memorizza il tipo di evento, il dispositivo, il numero di sequenza e l'esito dell'elaborazione. Su questo modello è imposto un vincolo di unicità sulla coppia `(device, seq)`, che consente di identificare ed eliminare eventuali duplicazioni dovute a ritrasmissioni.

7.2 Logica degli eventi IoT

La ricezione e l'elaborazione degli eventi IoT avvengono tramite l'endpoint `/api/core/iot/events`, che rappresenta il punto di ingresso unico per i messaggi provenienti dai nodi di stallone. All'arrivo di una richiesta, il backend autentica innanzitutto il dispositivo verificando la coerenza tra il `node_id` dichiarato nel payload e il valore dell'header `X-Device-Key`. Solo i dispositivi registrati e autorizzati possono generare eventi accettati dal sistema.

Superata la fase di autenticazione, l'endpoint verifica se esiste già un evento associato allo stesso dispositivo e allo stesso numero di sequenza. In presenza di un duplicato, l'evento viene ignorato, garantendo che ogni evento venga elaborato una sola volta.

Per gli eventi di tipo `rfid_scan`, il backend applica una logica di validazione che mette in relazione lo stato fisico osservato con lo stato logico del gestionale. A partire

dall'UID RFID ricevuto, il sistema ricerca la bicicletta associata; se il tag non risulta registrato, l'evento viene classificato come `unknown_rfid`. Se la bicicletta esiste ma non ha un parcheggio attivo, l'esito è `no_active_parking`. Nel caso in cui esista un parcheggio attivo ma lo stallo associato non coincida con quello del dispositivo che ha generato l'evento, l'evento viene marcato come `mismatch`. Solo quando lo stallo fisico e quello logico coincidono l'evento viene considerato valido, con esito `ok`.

Indipendentemente dall'esito, ogni evento viene persistito nella tabella `IoTEvent` insieme a un messaggio descrittivo e, quando applicabile, ai riferimenti allo stallo e al parcheggio coinvolti. Questo consente sia il tracciamento storico degli eventi sia l'analisi di eventuali anomalie operative.

7.3 Rilascio del parcheggio e liberazione del tag

Il ciclo di vita del tag RFID è strettamente legato a quello del parcheggio. Il modello `Parcheggio` espone il metodo `rilascia_bici()`, che viene invocato al momento dell'uscita della bicicletta dallo stallo. Tale metodo si occupa di chiudere il parcheggio aggiornandone lo stato e la data di fine, rendere nuovamente disponibile lo stallo e, se presente, azzerare il campo `rfid_uid` associato alla bicicletta.

In questo modo il tag RFID viene automaticamente liberato e reso riutilizzabile senza richiedere operazioni aggiuntive da parte dell'operatore.

Capitolo 8

Frontend Angular e flusso utente

8.1 Interazione con il desk RFID

Nel frontend Angular sono stati aggiunti servizi per:

- interrogare il Raspberry al desk tramite `/api/rfid/read-once`;
- inviare l'UID letto al backend con `/api/core/rfid/assign`.

La schermata di assegnazione del tag mostra la bicicletta selezionata e uno stato di attesa finché il servizio da banco non restituisce l'UID. Un esempio di schermata è previsto in Figura 8.1.

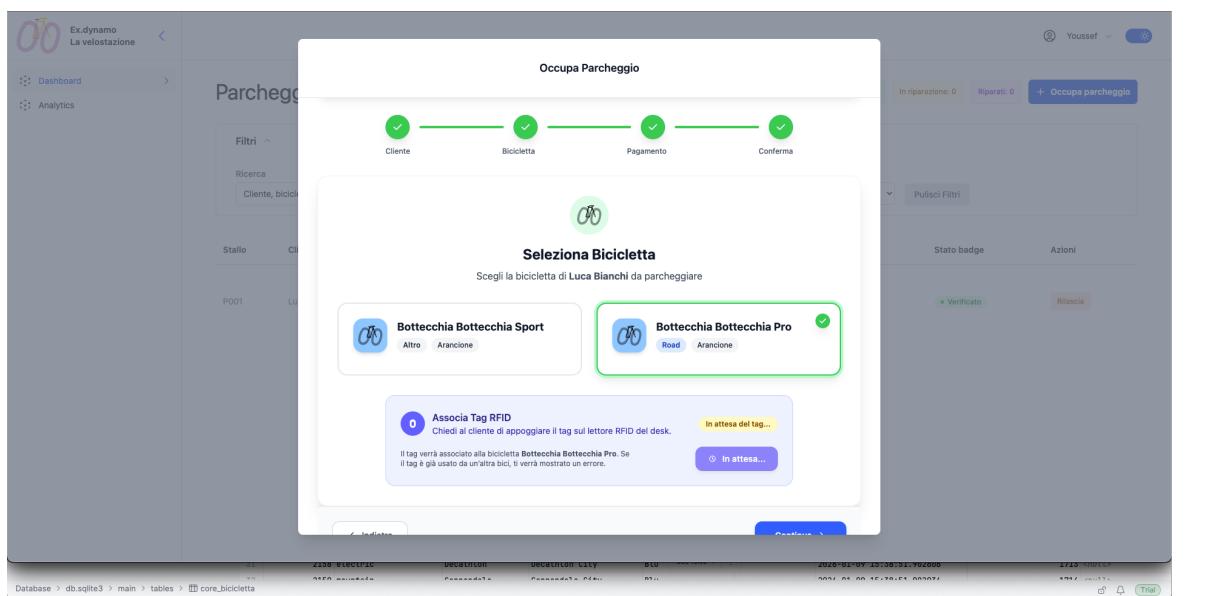


Figura 8.1: Schermata di assegnazione di un tag RFID a una bicicletta al desk.

8.2 Creazione e gestione dei parcheggi

Per la creazione di un parcheggio attivo, l'operatore seleziona utente, bici, tag RFID, stallo e tariffa e invia i dati al backend. Il frontend mostra un riepilogo e consente di chiudere il parcheggio al ritiro della bici.

La Figura 8.2 mostra la lista dei parcheggi attivi con badge di stato IoT.

The screenshot shows a user interface for managing bike parking slots. At the top, there's a header with the logo 'Ex.dynamo La velostazione', a user profile for 'Youssef', and a settings icon. Below the header, there are two navigation tabs: 'Dashboard' (selected) and 'Analytics'. The main area is titled 'Parcheggio / Stalli' (Parking / Slots). It features a search bar with filters for 'Ricerca' (Search), 'Tipo' (Type), and 'Stato' (Status). A button 'Pulisci Filtri' (Clear Filters) is also present. Below the filters, a table lists two active parking slots:

Stallo	Cliente	Bicicletta	Data Inizio	Tariffa	Stato badge	Azioni
P002	Luca Bianchi	Bottecchia Bottecchia Pro	07/02/2026, 16:22	★ GIORNALIERA Durata: 39min € 2,00	In attesa	Rilascia
P001	Luca Barbieri	Giant Giant Classic	07/02/2026, 15:10	★ GIORNALIERA Durata: 1h 51min € 2,00	Verificato	Rilascia

Figura 8.2: Elenco dei parcheggi attivi con badge di stato IoT.

Capitolo 9

Discussione finale, analisi dei consumi e sviluppi futuri

Il progetto ha avuto come obiettivo la realizzazione di un prototipo IoT capace di integrare il livello gestionale della Velostazione con lo stato fisico degli stalli di parcheggio. In questa sezione vengono discusse le principali limitazioni del prototipo, un'analisi dei consumi energetici e le possibili evoluzioni verso una soluzione pronta per l'uso in produzione.

9.1 Analisi dei consumi energetici

Nel prototipo i nodi di stallo sono alimentati tramite power bank USB commerciali, scelta funzionale a una rapida sperimentazione e alla riduzione della complessità di cablaggio. In un contesto reale, tuttavia, l'autonomia energetica rappresenta un fattore determinante.

Il nodo di stallo è basato sulla scheda Heltec WiFi LoRa 32 V3.2, che integra un microcontrollore ESP32-S3 e un ricetrasmettitore LoRa SX1262. Dai datasheet del produttore emerge come il consumo vari a seconda dello stato: durante la trasmissione LoRa a 868 MHz la corrente può raggiungere valori dell'ordine dei 200–230 mA¹, mentre in ricezione si attesta intorno ai 90 mA. In modalità di sleep profondo, quando alimentata a batteria, la stessa scheda può scendere molto arrivando ai microampere (circa 15 μ A)² [1].

Al nodo è collegato un lettore RFID RC522, mantenuto costantemente alimentato nel prototipo per semplificare la logica. Secondo il datasheet del modulo, l'assorbimento tipico è compreso tra circa 13 e 26 mA in funzionamento, con una corrente di standby dell'ordine di 10–13 mA [2].

Nel prototipo il microcontrollore rimane costantemente attivo per garantire reattività (circa 30–60 mA), mentre il lettore RFID è mantenuto alimentato in modo continuo. Al

¹mA: milliampere, unità di misura della corrente elettrica. 1 mA = 0.001 A (un millesimo di ampere).

² μ A: microampere, unità di misura della corrente elettrica. 1 μ A = 0.000001 A (un milionesimo di ampere). 1 mA = 1000 μ A.

contrario, le trasmissioni LoRa avvengono raramente e per intervalli di tempo molto brevi, risultando trascurabili nel calcolo del consumo medio. La Tabella 9.1 riassume il ragionamento seguito e i contributi considerati per giungere alla stima del consumo medio di circa 55 mA.

Componente	Stato operativo	Corrente stimata
ESP32 (Heltec WiFi LoRa 32)	attivo	~40–50 mA
Modulo RFID RC522	standby continuo	~10–13 mA
Trasmissione LoRa	eventi rari	trascurabile in media
Buzzer e LED		trascurabile
Totale stimato		~55 mA

Tabella 9.1: Stima dei principali contributi al consumo medio del nodo di stallo

Nel prototipo l'alimentazione è fornita da un power bank commerciale da 10 000 mAh. Applicando la relazione standard tra capacità e corrente media assorbita, si ottiene un'autonomia teorica di circa:

$$\frac{10\,000 \text{ mAh}}{55 \text{ mA}} \approx 182 \text{ h} \approx 7.5 \text{ giorni.}$$

La durata è adeguata per un prototipo sperimentale, ma non sufficiente per un'installazione permanente.

9.2 Sviluppi futuri e passaggio alla produzione

Per facilitare l'evoluzione del prototipo verso una soluzione pronta per la produzione, il primo intervento riguarderebbe la gestione energetica: l'uso sistematico delle modalità di sleep profondo del microcontrollore e l'alimentazione del lettore RFID solo quando necessario permetterebbero di ridurre il consumo medio. A parità di capacità di batteria, ciò consentirebbe di estendere l'autonomia.

Ulteriori miglioramenti includono l'integrazione dei nodi in involucri protettivi adeguati all'ambiente di installazione, l'adozione di soluzioni di alimentazione dedicate e la persistenza su memoria non volatile del numero di sequenza degli eventi, al fine di garantire continuità operativa anche in caso di riavvio del dispositivo.

Capitolo 10

Appendice tecnica

10.1 Codice dei nodi IoT

10.1.1 Invio di un evento rfid_scan

Listing 10.1: Invio di un evento rfid_scan dal nodo di stallo

```
1 void sendRfidScanEvent(const String &uidHex) {
2     String msg = "node=" + String(ID_STALLO) +
3                 ";seq=" + String(seq) +
4                 ";ev=rfid_scan;uid=" + uidHex;
5
6     bool ok = sendLoRaMessage(msg);
7     if (ok) {
8         beepShort();
9     } else {
10        beepLong();
11    }
12 }
```

10.1.2 Logica di anti-spam per la lettura RFID

Listing 10.2: Anti-spam per la lettura RFID sul nodo di stallo

```
1 bool checkRfidAndSendIfNeeded() {
2     if (!rfid.PICC_IsNewCardPresent()) {
3         if (lastTagPresent) {
4             lastTagPresent = false;
5             lastUidHex = "";
6         }
7         return false;
8     }
9 }
```

```

10  if (!rfid.PICC_ReadCardSerial()) {
11      return false;
12  }
13
14  String uidHex = "";
15  for (byte i = 0; i < rfid.uid.size; i++) {
16      if (rfid.uid.uidByte[i] < 0x10) {
17          uidHex += '0';
18      }
19      uidHex += String(rfid.uid.uidByte[i], HEX);
20  }
21  uidHex.toUpperCase();
22
23  unsigned long now = millis();
24  bool isNewUid = (uidHex != lastUidHex);
25  bool shouldSend = false;
26
27  if (isNewUid) {
28      shouldSend = true;
29  } else if (!lastTagPresent) {
30      shouldSend = true;
31  }
32
33  lastTagPresent = true;
34  lastUidHex = uidHex;
35  lastTagMillis = now;
36
37  rfid.PICC_HaltA();
38  rfid.PCD_StopCrypto1();
39
40  if (shouldSend) {
41      sendRfidScanEvent(uidHex);
42  }
43
44  return shouldSend;
45 }

```

10.1.3 Codice del gateway Heltec RX

Listing 10.3: Loop di ricezione sul gateway Heltec RX

```

1 void loop() {
2     String msg;
3     int state = radio.receive(msg);
4
5     if (state == RADIOLIB_ERR_NONE) {

```

```

6   float rssi = radio.getRSSI();
7   Serial.print(msg);
8   Serial.print(" RSSI=");
9   Serial.println(rssi);
10 }
11 }
```

10.2 Servizi sul Raspberry Pi

10.2.1 Gateway LoRa → HTTP

Listing 10.4: Parsing della linea seriale nel gateway LoRa

```

1 def parse_line(line: str):
2     line = line.strip()
3     if not line:
4         return None
5
6     if " " in line:
7         first_token = line.split(" ", 1)[0]
8     else:
9         first_token = line
10
11    if not first_token.startswith("node="):
12        return None
13
14    parts = first_token.split(" ;")
15    data = {}
16    for p in parts:
17        if "=" not in p:
18            continue
19        key, value = p.split("=", 1)
20        key = key.strip().lower()
21        value = value.strip()
22        data[key] = value
23
24    if "node" not in data or "seq" not in data or "ev" not in data:
25        print(f"Unvalid line: {line}")
26        return None
27    try:
28        data["node"] = int(data["node"])
29        data["seq"] = int(data["seq"])
30        print(f"Parsed data: {data}")
31    except ValueError:
32        return None
33 }
```

```
34     return data
```

Listing 10.5: Costruzione del payload JSON per Django

```
1 def build_payload(parsed: dict) -> dict:
2     node_id = parsed["node"]
3     seq = parsed["seq"]
4     event_type = parsed["ev"].strip().lower()
5
6     payload = {
7         "node_id": node_id,
8         "seq": seq,
9         "event_type": event_type,
10    }
11    if event_type == "rfid_scan" and "uid" in parsed:
12        payload["rfid_uid"] = parsed["uid"]
13
14    return payload
```

10.2.2 API Desk RFID

Listing 10.6: Endpoint di lettura singola del tag RFID al desk

```
1 @app.route("/api/rfid/read-once", methods=["GET"])
2 def read_once():
3     init_gpio()
4     try:
5         beep("short")
6         uid_int, text = reader.read()
7         uid_hex_full = format(uid_int, "X").upper()
8         uid_hex = uid_hex_full[:8]
9         beep("double")
10        return jsonify({"uid_hex": uid_hex}), 200
11    except Exception as e:
12        beep("long")
13        return jsonify({"detail": str(e)}), 500
```

10.3 Snippet di codice aggiuntivi

10.3.1 Pattern di beep sul nodo Heltec

Listing 10.7: Funzioni di controllo del buzzer sul nodo di stallo

```
1 void beep(unsigned int durationMs = 80) {
2     digitalWrite(PIN_BUZZER, HIGH);
```

```

3   delay(durationMs);
4   digitalWrite(PIN_BUZZER, LOW);
5 }
6
7 void beepShort() {
8   beep(80);
9 }
10
11 void beepDouble() {
12   for (int i = 0; i < 2; i++) {
13     beep(60);
14     delay(80);
15   }
16 }
17
18 void beepLong() {
19   beep(300);
20 }
```

10.4 Comandi essenziali

Di seguito alcuni comandi di riferimento per l'utilizzo in locale (da adattare alle directory effettive):

- attivazione virtualenv sul Raspberry:

```
source ~/velostazione-venv/bin/activate
```

- esecuzione del gateway LoRa:

```
python gateway_lora_to_http.py
```

- esecuzione dell'API desk RFID:

```
python desk_rfid_api.py
```

- avvio del backend Django:

```
python manage.py runserver 0.0.0.0:8000
```

- avvio del frontend Angular:

```
npm start
```

10.5 Repository e figure

I link alle repository Git del progetto verranno inseriti qui:

- componente IoT *Velostazione IoT* (Codice dei nodi di stallo e servizi Python su Raspberry Pi): <https://github.com/mussida/velostazione-iot>
- sito web del progetto: <https://velostazione-iot-doc.onrender.com/>

Bibliografia

- [1] Heltec Automation. *WiFi LoRa 32 V3.2 Datasheet*. Rev 1.3. Heltec Automation Technology Co., Ltd. Ott. 2024. URL: https://resource.heltec.cn/download/WiFi_LoRa_32_V3/HTIT-WB32LA_V3.2.pdf (visitato il giorno 08/02/2026).
- [2] Handson Technology. *RC522 RFID Development Kit Data Specs*. SKU: MDU1040. Handson Technology. 2024. URL: <https://www.handsontec.com/dataspecs/RC522.pdf> (visitato il giorno 08/02/2026).