University of Cagliari

Department of Engineering

Master's Degree in "Computer Engineering,
Cybersecurity And Artificial Intelligence"

Smart Garage Door

Advisor:
Prof. Michele Nitti

Authors:
Lello Molinario
Matteo Tuzi

Accademic Year 2025/2026
Cagliari

Table of Contents

Abstract

Negli ultimi anni, il paradigma dell'**Internet of Things (IoT)** ha radicalmente trasformato il modo in cui gli ambienti domestici e industriali vengono gestiti, introducendo soluzioni in grado di migliorare comfort, sicurezza e risparmio energetico. In tale contesto, il presente progetto, denominato **Smart Garage Door**, propone lo sviluppo di un sistema IoT modulare e scalabile per l'automazione intelligente di una porta da garage. L'obiettivo è quello di consentire un controllo remoto e automatico della porta, garantendo allo stesso tempo affidabilità, sicurezza dei dati e semplicità d'uso per l'utente finale.

Il sistema è concepito come un insieme distribuito di nodi cooperanti che comunicano attraverso il protocollo **MQTT** su rete **Wi-Fi**. L'architettura prevede tre componenti principali:

- un **microcontrollore Arduino**, incaricato del controllo fisico della porta e della gestione dei sensori locali (PIR per la rilevazione di movimento e relè per l'attuazione del motore);
- un **nodo NodeMCU ESP8266**, che funge da unità di comunicazione e gateway MQTT, responsabile della trasmissione dei comandi e delle notifiche;
- un **modulo GPS**, utilizzato per la localizzazione dell'utente e per l'automazione di prossimità, attivando l'apertura del cancello al rilevamento di un dispositivo autorizzato entro una distanza configurabile.

La parte software è stata progettata per garantire un'interazione fluida e sicura tra i diversi livelli del sistema. Un **server Flask**, sviluppato in linguaggio **Python**, coordina la logica applicativa, gestisce le sessioni utente e registra lo stato del sistema. In parallelo, un **bot Telegram** fornisce un'interfaccia utente remota intuitiva, permettendo di eseguire operazioni di apertura, chiusura, verifica dello stato della porta e gestione multiutenza, nonché di ricevere notifiche in tempo reale sugli eventi di sistema.

Il progetto è stato sviluppato seguendo il **System Development Life Cycle (SDLC)**, articolato in quattro fasi principali: *Planning*, *Analysis*, *Design* e *Implementation & Testing*. Durante la fase di progettazione sono state analizzate due prospettive complementari:

1. una **analisi a budget illimitato**, orientata all'individuazione di soluzioni ottimali dal punto di vista prestazionale e tecnologico;
2. una **analisi realistica**, volta all'implementazione effettiva del prototipo entro i vincoli imposti dal corso (costo complessivo inferiore a 150 €, dispositivi alimentati a batteria, e connettività Wi-Fi disponibile).

La realizzazione finale integra diverse funzionalità: controllo remoto della porta, chiusura temporizzata automatica, automazione di prossimità tramite GPS, invio di notifiche Telegram, gestione multiutente e aggiornamento continuo dello stato tramite protocollo MQTT. Il sistema è stato verificato con test funzionali che hanno confermato la piena rispondenza ai requisiti specificati, con tempi di risposta inferiori al secondo e tasso di errore nella rilevazione di prossimità inferiore all'1%.

Il risultato è una soluzione **affidabile, economica e modulare**, concepita per essere facilmente estendibile con nuove componenti e funzioni. Tra i possibili sviluppi futuri si annoverano l'integrazione di sensori per la rilevazione di ostacoli, l'adozione di meccanismi di autenticazione

avanzata e la realizzazione di una dashboard web per la consultazione dei log e il monitoraggio energetico. L'esperienza progettuale dimostra come un approccio metodico e ingegneristico basato sul ciclo SDLC consenta di passare efficacemente dall'analisi dei requisiti alla realizzazione di un sistema IoT completo, sostenibile e conforme ai requisiti di sicurezza e affidabilità propri delle applicazioni domestiche intelligenti.

Chapter 1

Introduction

1.1 Contesto e motivazioni

Negli ultimi anni, l'avvento dell'**Internet of Things (IoT)** ha profondamente modificato il modo in cui le persone interagiscono con l'ambiente circostante, ridefinendo i concetti di comfort, sicurezza e automazione domestica. La diffusione di sensori intelligenti, microcontrollori a basso consumo e piattaforme cloud ha favorito la nascita di ecosistemi di dispositivi interconnessi, in grado di raccogliere, elaborare e condividere informazioni in tempo reale, migliorando l'efficienza delle attività quotidiane. Come evidenziato da Atzori et al. [**Atzori2010**] e Gubbi et al. [**Gubbi2013**], l'IoT rappresenta oggi uno dei principali paradigmi abilitanti della trasformazione digitale, con un impatto crescente sulla vita domestica, industriale e urbana.

In un contesto sociale sempre più frenetico, la necessità di semplificare la gestione della casa e di ridurre le dimenticanze accidentali — come lasciare una porta aperta o non attivare un sistema di chiusura — si traduce in una crescente domanda di soluzioni *smart* affidabili e personalizzabili. La domotica moderna, alimentata dallo sviluppo delle tecnologie IoT, rappresenta oggi uno dei principali motori di innovazione nel mercato residenziale, con applicazioni che spaziano dal controllo dell'illuminazione alla climatizzazione, dalla sicurezza perimetrale alla gestione degli accessi.

Tra i dispositivi più diffusi in questo ambito rientrano i **controller intelligenti per porte da garage**, il cui mercato ha conosciuto un'espansione costante. Secondo un'analisi di *Vantage Market Research* [**Vantage2023**], il valore globale del mercato dei controller per porte da garage intelligenti è stato stimato a 164,4 milioni di dollari statunitensi nel 2020 e si prevede raggiungerà circa 200,55 milioni di dollari entro il 2028, con un tasso di crescita annuo composto (CAGR) pari al 2,5%. La Figura 1.1 illustra l'andamento previsto del mercato nel periodo 2024–2035.

La crescita del mercato è trainata in particolare dal Nord America e dall'Europa, dove l'attenzione verso la sicurezza domestica e l'efficienza energetica stimola l'adozione di soluzioni connesse. Dal punto di vista immobiliare, la rilevanza del garage come elemento integrante dell'abitazione è confermata da studi di settore: secondo il *Philadelphia Inquirer* (2022), la parola "Garage" figura al secondo posto tra i termini più ricorrenti negli annunci immobiliari del Nord-Est degli Stati Uniti, a dimostrazione del valore funzionale e simbolico di questo spazio domestico.

Dal punto di vista della sicurezza, uno dei fattori che giustifica l'adozione di sistemi automatizzati è la prevenzione dei furti con scasso. Secondo Holler et al. [**Holler2014**], circa il 9% delle effrazioni domestiche avviene attraverso la porta del garage, spesso a causa di semplici dimenticanze o della mancata chiusura dei sistemi di accesso. Un sistema di automazione intelligente può quindi ridurre significativamente tali rischi, intervenendo a supporto dell'errore umano e migliorando la protezione degli ambienti residenziali.

Alla luce di queste considerazioni, il progetto **Smart Garage Door** nasce con l'obiettivo di sviluppare un sistema IoT per la gestione automatica e remota di una porta da garage, capace di

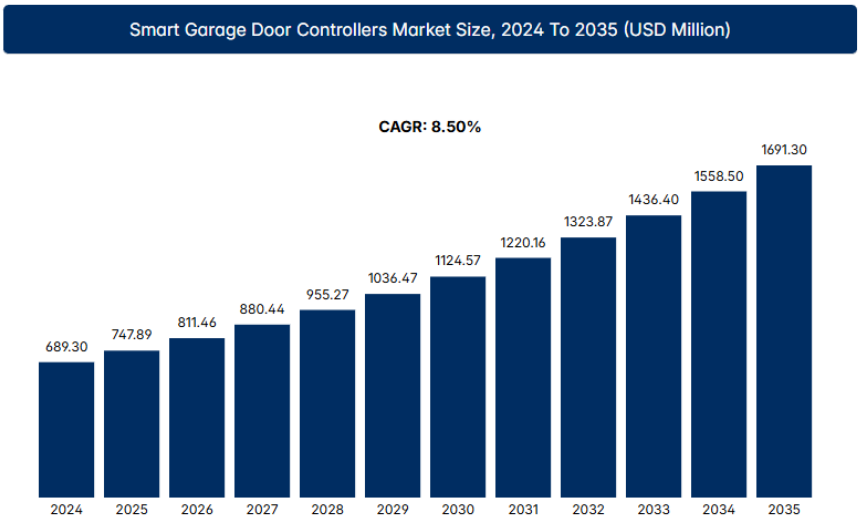

Figure 1.1: Mercato globale dei controller per porte da garage intelligenti (USD): tendenze e previsioni 2024–2035. Fonte: [**Vantage2023**]

coniugare semplicità, affidabilità e scalabilità. L'idea 'e di realizzare un dispositivo che consenta all'utente di controllare l'accesso sia localmente — attraverso sensori di movimento e pulsanti fisici — sia da remoto, tramite applicazioni basate su Telegram Bot e server Flask. Il sistema integra inoltre una logica di automazione di prossimit'a, che sfrutta i dati GPS per riconoscere la presenza dell'utente e attivare automaticamente l'apertura o la chiusura del garage, riducendo il rischio di dimenticanze e aumentando la sicurezza dell'abitazione.

La scelta di questo caso d'uso risponde anche a un obiettivo formativo: applicare in modo concreto le tecnologie studiate durante il corso di *Internet of Things* per la realizzazione di un prototipo completo, economicamente sostenibile e tecnicamente scalabile. Il sistema proposto, grazie al suo carattere modulare e open source, si presta inoltre a successive evoluzioni, come l'integrazione di sensori per la rilevazione di ostacoli, la gestione energetica intelligente e l'interoperabilità con piattaforme di domotica avanzata.

1.2 Obiettivi del progetto

L'obiettivo principale è la progettazione e realizzazione di un sistema IoT in grado di:

- consentire l'apertura e la chiusura della porta del garage da remoto, attraverso interfacce utente semplici e sicure;
- implementare un meccanismo di **automazione di prossimità**, che apra automaticamente la porta al rilevamento dell'utente in avvicinamento, e la richiuda quando il veicolo si allontana;
- fornire **notifiche in tempo reale** sugli eventi di apertura, chiusura o errore tramite canali digitali (Telegram);
- integrare una logica di **chiusura automatica temporizzata** e gestione locale in assenza di connettività;
- supportare più utenti autenticati e registrare le principali azioni di sistema (gestione multiutenza).

Oltre agli obiettivi funzionali, il progetto è stato orientato al rispetto di una serie di vincoli di tipo non funzionale, tra cui:

- **costo complessivo inferiore a 150 €**;

- **basso consumo energetico**, compatibile con dispositivi alimentati a batteria;
- **tempo di risposta inferiore a 1 s**;
- **affidabilità** e **tasso di falsi positivi inferiore all'1%** nella rilevazione di prossimità.

1.3 Scenario di riferimento e assunzioni

Il progetto **Smart Garage Door** è concepito per un contesto domestico reale, in cui la porta del garage è collocata in prossimità di un'abitazione privata dotata di copertura Wi-Fi stabile fino all'area esterna. In linea con l'evoluzione dei sistemi di *smart home* descritta da Atzori et al. [**Atzori2010**] e Gubbi et al. [**Gubbi2013**], il sistema proposto mira a integrare funzionalità di automazione, controllo remoto e interazione intelligente all'interno di un ecosistema IoT locale, mantenendo al contempo indipendenza operativa in caso di perdita di connettività.

Si assumono le seguenti condizioni di contesto e funzionamento:

- il meccanismo di apertura è compatibile con un comando elettrico digitale, azionabile tramite relè collegato al microcontrollore;
- lo stato della porta (aperta, chiusa o in movimento) può essere rilevato mediante sensore dedicato o segnale di feedback dal motore;
- gli utenti dispongono di uno *smartphone* connesso a Internet, identificabile tramite **Telegram Bot API** [**TelegramAPI**] o identificatore univoco GPS/BLE;
- la rete Wi-Fi domestica copre l'area del garage, garantendo comunicazione stabile con il nodo **ESP8266** [**ESP8266**];
- il sistema è in grado di operare in modalità locale anche in assenza di rete, sfruttando la comunicazione diretta tra i microcontrollori.

L'ambiente operativo si colloca dunque in una tipica abitazione unifamiliare, ma l'architettura modulare progettata consente una facile estensione a scenari più complessi, come parcheggi condominiali o accessi industriali. Il sistema è stato pensato per una **operatività continua (24/7)**, garantendo la disponibilità del servizio e la tracciabilità degli eventi (requisiti NFR1 e NFR5).

Dal punto di vista funzionale, il dispositivo è destinato a utenti che possiedono una o più autorimesse e desiderano incrementare sicurezza e comfort, riducendo la dipendenza dalla memoria e dalle azioni manuali. Il sistema deve monitorare costantemente lo stato della porta e reagire ai comandi in tempo reale, offrendo all'utente la possibilità di:

- aprire o chiudere la porta manualmente, localmente o da remoto;
- ricevere notifiche di stato e messaggi di conferma in tempo reale;
- beneficiare di un'automazione basata sulla posizione GPS, capace di riconoscere se l'utente sta rientrando o uscendo di casa, e di azionare automaticamente la porta;
- aggiungere o rimuovere persone autorizzate a interagire con il sistema.

Tale approccio riflette i principi delle architetture IoT modulari proposte da Holler et al. [**Holler2014**] e Palattella et al. [**Palattella2016**], basate sulla cooperazione di nodi intelligenti e sull'uso di protocolli leggeri per la comunicazione macchina-macchina. La possibilità di estendere il sistema a contesti multiutente o multipiano dimostra la scalabilità dell'architettura progettata, in linea con i paradigmi di interoperabilità e adattabilità propri delle moderne infrastrutture IoT.

Dal punto di vista esperienziale, il sistema è pensato per migliorare il comfort dell'utente finale: durante la guida, ad esempio, l'automazione di prossimità elimina la necessità di interazione manuale, riducendo i tempi di accesso e aumentando la sicurezza. L'invio di messaggi di feedback, come saluti o notifiche personalizzate, tramite l'interfaccia **Telegram** [**TelegramAPI**],

contribuisce a rendere il sistema più trasparente e intuitivo, favorendo l'adozione anche da parte di utenti non esperti.

Infine, il progetto include un meccanismo di **chiusura automatica temporizzata**, volto a garantire sicurezza aggiuntiva in caso di dimenticanze, e la possibilità di operare in modalità *failsafe*, assicurando la chiusura automatica in caso di guasto alla rete o al nodo di controllo. Queste funzionalità rispondono a quanto indicato nei principi di progettazione robusta dei sistemi IoT resilienti descritti da Pressman e Maxim [**Pressman2019**], assicurando coerenza tra requisiti funzionali, prestazioni e affidabilità complessiva.

1.4 Analisi di fattibilità

1.4.1 Fattibilità tecnica

Dal punto di vista tecnico, il sistema **Smart Garage Door** risulta pienamente realizzabile con componenti hardware e software a basso costo, ampiamente reperibili sul mercato e supportati da una vasta comunità open source. L'impiego della scheda **NodeMCU ESP8266** [**ESP8266**] garantisce connettività Wi-Fi integrata e compatibilità nativa con il protocollo **MQTT** [**MQTTspec**], ampiamente adottato nei sistemi IoT per la sua leggerezza, affidabilità e capacità di funzionare in ambienti a risorse limitate.

L'integrazione con un **microcontrollore Arduino** [**ArduinoRef**] consente di gestire le logiche locali e i sensori di movimento (PIR e relè) in modo indipendente dal nodo di rete, aumentando la resilienza del sistema. Il modulo **GPS** fornisce una localizzazione accurata dell'utente e consente la realizzazione di automazioni di prossimità basate sulla distanza, in linea con le architetture distribuite e cooperanti descritte da Holler et al. [**Holler2014**].

Sul piano software, l'infrastruttura è stata progettata per garantire interoperabilità, scalabilità e semplicità d'integrazione. Il framework **Flask** [**Flask2024**] è stato scelto per la realizzazione del server web e delle API REST, grazie alla sua leggerezza e al supporto nativo per la gestione di richieste asincrone. La comunicazione utente–sistema è invece affidata al **Telegram Bot API** [**TelegramAPI**], che fornisce un'interfaccia intuitiva e sicura senza la necessità di sviluppare un'app mobile dedicata.

Complessivamente, la soluzione proposta sfrutta tecnologie consolidate e standard aperti, assicurando compatibilità con future estensioni e pieno allineamento con i principi di interoperabilità e modularità propri dell'ingegneria dei sistemi IoT [**Palattella2016**].

1.4.2 Fattibilità economica

La fattibilità economica del progetto è garantita dall'adozione di componenti hardware low-cost e di software completamente open source. Il costo complessivo del prototipo, comprensivo di microcontrollori, sensori, moduli GPS, cablaggi e alimentazione, è stimato in circa 90–100 €, ampiamente al di sotto del limite imposto dal requisito non funzionale NFR10 NFR10 (*costo* ≤ 150).

Non sono previsti costi di licenza software, poiché tutte le tecnologie adottate — Arduino IDE, Python, Flask, MQTT e Telegram — sono distribuite sotto licenza libera. Questo approccio consente non solo una notevole riduzione dei costi di sviluppo, ma anche una maggiore trasparenza e riproducibilità accademica, in linea con gli obiettivi del corso e con le buone pratiche di progettazione sostenibile indicate da Pressman e Maxim [**Pressman2019**].

Tali scelte risultano inoltre coerenti con l'andamento del mercato dei dispositivi smart per l'automazione domestica, che secondo Vantage Market Research [**Vantage2023**] è in costante crescita, trainato dalla domanda di soluzioni economiche, affidabili e modulari.

1.4.3 Fattibilità organizzativa

Il progetto è stato sviluppato da un team di due persone, con una chiara suddivisione dei compiti tra la parte hardware e quella software, secondo un approccio ingegneristico iterativo e incrementale basato sul **System Development Life Cycle (SDLC)** [**Pressman2019**]. Le attività sono state articolate in quattro fasi principali:

1. **Planning**: definizione del contesto applicativo, obiettivi e vincoli;
2. **Analysis**: identificazione e formalizzazione dei requisiti funzionali e non funzionali;
3. **Design e Implementation**: progettazione dell'architettura e sviluppo del prototipo hardware–software;
4. **Testing e Validation**: esecuzione dei test funzionali e valutazione delle prestazioni del sistema.

L'utilizzo di strumenti di controllo versione (Git) e la documentazione dettagliata delle scelte progettuali hanno garantito tracciabilità e coerenza tra le fasi, riducendo il rischio di regressioni e facilitando la revisione del codice. La collaborazione tra le due componenti del team ha seguito una logica di integrazione continua, con cicli di verifica hardware–software settimanali. Questo approccio ha permesso di rispettare i tempi di sviluppo previsti, massimizzando l'efficienza e assicurando una piena aderenza ai principi di progettazione iterativa promossi dalla metodologia SDLC.

1.5 Struttura del documento

Il presente elaborato è organizzato secondo le fasi del **System Development Life Cycle (SDLC)** [**Pressman2019**], che fornisce una struttura metodologica per l'analisi, la progettazione, l'implementazione e la validazione di sistemi complessi. Ogni capitolo corrisponde a una specifica fase del ciclo di vita del progetto, garantendo tracciabilità e coerenza tra obiettivi, soluzioni e risultati.

Il **Capitolo 2** presenta una rassegna dello stato dell'arte, analizzando le principali soluzioni esistenti nel campo dei sistemi di automazione per porte da garage e individuando le aree di miglioramento che hanno motivato lo sviluppo del progetto.

Il **Capitolo 3** descrive nel dettaglio i requisiti funzionali (FR) e non funzionali (NFR), il contesto applicativo e le assunzioni di progetto, ponendo le basi per le scelte progettuali successive.

Il **Capitolo 4** illustra le scelte di progettazione e l'architettura complessiva del sistema, includendo l'analisi comparativa tra lo scenario teorico a budget illimitato e la soluzione reale implementata nel prototipo.

Il **Capitolo 5** documenta la fase di implementazione, riportando la struttura del codice, le configurazioni hardware–software e le principali interfacce operative (Flask, Telegram, MQTT).

Il **Capitolo 6** raccoglie i risultati dei test sperimentali, con particolare attenzione alla validazione dei requisiti e alla valutazione delle prestazioni del sistema in condizioni reali.

Infine, il **Capitolo 7** presenta le conclusioni, una sintesi dei risultati ottenuti e le prospettive di sviluppo futuro, delineando possibili direzioni di miglioramento e ampliamento del progetto.

Chapter 2

State of the Art

2.1 Panoramica dei sistemi di automazione domestica

Negli ultimi anni, il settore dell'automazione domestica ha conosciuto una rapida e costante evoluzione grazie alla crescente diffusione delle tecnologie di comunicazione wireless e dei microcontrollori connessi in rete. L'avvento dell'**Internet of Things (IoT)** ha reso possibile l'interconnessione di dispositivi eterogenei, favorendo la nascita di ecosistemi digitali in grado di acquisire, elaborare e condividere dati in tempo reale [**Atzori2010**, **Gubbi2013**].

I moderni sistemi di **smart home** si sono progressivamente trasformati da semplici soluzioni di controllo remoto a piattaforme distribuite capaci di apprendere le abitudini dell'utente e adattarsi automaticamente al contesto operativo. Questi sistemi sfruttano una combinazione di sensori, attuatori e interfacce digitali per ottimizzare comfort, sicurezza ed efficienza energetica. Come osservato da Holler et al. [**Holler2014**], la convergenza tra comunicazione macchina–macchina (M2M) e Internet ha posto le basi per l'intelligenza ambientale, aprendo la strada a soluzioni integrate che riducono l'intervento umano nelle operazioni quotidiane.

In tale contesto, le soluzioni dedicate all'automazione di porte, cancelli e garage rappresentano un campo applicativo consolidato ma ancora in espansione. L'integrazione di moduli Wi-Fi, servizi cloud e applicazioni mobili ha reso possibile il controllo remoto e la gestione automatizzata degli accessi, ponendo però nuove sfide in termini di interoperabilità, sicurezza e affidabilità. Le architetture proposte in letteratura e sul mercato convergono verso modelli distribuiti, in cui la capacità di comunicazione tra nodi e la latenza di risposta costituiscono parametri fondamentali per la qualità complessiva del sistema [**Palattella2016**, **Piyare2013**].

2.2 Soluzioni commerciali esistenti

Le principali soluzioni disponibili sul mercato possono essere ricondotte a due macro-categorie:

- **Sistemi proprietari**, sviluppati da aziende specializzate e basati su infrastrutture chiuse, con comunicazioni centralizzate su server cloud (es. Chamberlain, Tailwind, Nexx);
- **Sistemi aperti o compatibili**, che si integrano con piattaforme standard come Google Home, Alexa o Home Assistant, e adottano protocolli interoperabili quali **MQTT** o **Zigbee**.

2.2.1 Chamberlain *MyQ*

Il sistema *MyQ* di Chamberlain è una delle soluzioni più diffuse per il controllo remoto delle porte da garage. Il dispositivo utilizza un modulo Wi-Fi integrato per connettersi a un'infrastruttura cloud proprietaria, accessibile tramite applicazione mobile. L'utente può verificare lo stato della

porta, ricevere notifiche e pianificare aperture automatiche. Nonostante la buona stabilità operativa, l'architettura chiusa limita l'integrazione con altri sistemi e impedisce il funzionamento in assenza di connessione Internet, generando dipendenza dal cloud e vincoli di privacy.

Figure 2.1: Esempio del sistema **MyQ** di Chamberlain. Fonte: *Chamberlain Group Inc., MyQ Official Product Documentation* (consultato 2025).

2.2.2 Tailwind iQ3

Tailwind iQ3 adotta un approccio ibrido, combinando la comunicazione cloud con la connessione Bluetooth Low Energy (BLE) del dispositivo mobile. Il sistema è in grado di aprire automaticamente il garage quando rileva la presenza dell'auto associata, sfruttando la prossimità BLE. Pur garantendo una buona esperienza utente, tale soluzione presenta vincoli di compatibilità con smartphone specifici e un'affidabilità ridotta in ambienti esterni con ostacoli o interferenze elettromagnetiche.

Figure 2.2: Esempio del sistema **Tailwind iQ3**. Fonte: *Tailwind Technologies Inc., Tailwind iQ3 Product Documentation* (consultato 2025).

2.2.3 Nexx Garage

Nexx Garage propone un dispositivo Wi-Fi economico che può essere integrato su meccanismi di apertura preesistenti. Il sistema consente il controllo remoto, l'automazione di prossimità basata su GPS e il supporto ai comandi vocali. Tuttavia, la sua dipendenza da servizi cloud esterni per il monitoraggio e le notifiche genera problemi legati alla sicurezza dei dati, alla continuità di servizio e ai costi di mantenimento a lungo termine. In particolare, la gestione remota attraverso infrastrutture centralizzate comporta rischi di latenza e vulnerabilità nel trasferimento di dati sensibili [**Palattella2016**].

Figure 2.3: Esempio del sistema **Nexx Garage**. Fonte: *Nexx Smart Home Inc., Nexx Garage Product Documentation* (consultato 2025).

2.3 Analisi dei competitors

Per valutare in modo rigoroso le alternative presenti sul mercato e posizionare correttamente il sistema *Smart Garage Door* rispetto alle soluzioni commerciali esistenti, è stata adottata la metodologia di analisi **SWOT** (*Strengths, Weaknesses, Opportunities, Threats*). Tale metodologia, ampiamente utilizzata nel design dei sistemi IoT e nei processi di technology assessment, consente di analizzare ciascun competitor considerando non solo gli aspetti tecnici (funzionalità, prestazioni, architettura), ma anche quelli strategici quali rischi, opportunità di integrazione, vincoli di adozione e prospettive di miglioramento.

L'obiettivo non è un confronto commerciale, bensì una valutazione tecnica strutturata che permetta di:

- identificare i limiti progettuali delle alternative esistenti;
- evidenziare aree in cui il progetto proposto apporta un miglioramento concreto;
- individuare opportunità e minacce legate a scelte architetturali simili;
- giustificare in modo formale le decisioni progettuali adottate nella fase di design.

Le tabelle SWOT che seguono sintetizzano l'analisi per i principali competitor identificati nel mercato attuale.

2.3.1 Chamberlain *MyQ*

Il sistema *MyQ* costituisce una delle soluzioni commerciali più diffuse per il controllo remoto delle porte da garage. Opera mediante un'infrastruttura cloud proprietaria e un'applicazione mobile dedicata.

2.3.2 Chamberlain *MyQ*

Table 2.1: Analisi SWOT del sistema Chamberlain MyQ

Strengths	**Weaknesses**
Interfaccia utente curata; notifiche affidabili; prodotto maturo e diffuso.	Dipendenza completa dal cloud; assenza di funzionamento offline; scarsa interoperabilità; costo elevato.
Opportunities	**Threats**
Integrazione futura con ecosistemi standard; estensione a più modelli di motori.	Rischi privacy; latenza e failure del cloud; concorrenza di soluzioni open-source più economiche.

2.3.3 Tailwind iQ3

Tailwind iQ3 adotta un modello ibrido basato su cloud e Bluetooth Low Energy (BLE), fornendo automazioni di prossimità.

Table 2.2: Analisi SWOT del sistema Tailwind iQ3

Strengths	**Weaknesses**
Automazione BLE; installazione semplice; compatibile con Google Home.	Compatibilità BLE limitata; interferenze frequenti; dipendenza dal cloud.
Opportunities	**Threats**
Possibile apertura a protocolli standard; ampliamento dispositivo.	Rischi di sicurezza BLE; instabilità outdoor; vulnerabilità del cloud.

2.3.4 Nexx Garage

Nexx Garage propone una soluzione Wi-Fi economica compatibile con sistemi di apertura esistenti.

Table 2.3: Analisi SWOT del sistema Nexx Garage

Strengths	**Weaknesses**
Compatibile con sistemi preesistenti; supporto Alexa/Google; automazioni GPS.	Dipendenza da cloud; problemi di latenza; vulnerabilità API; scarsa privacy GPS.
Opportunities	**Threats**
Possibile apertura API; margini per migliorare sicurezza.	Interruzione servizio in caso di failure cloud; concorrenza di soluzioni Wi-Fi open-source.

2.3.5 Confronto sintetico tra competitors

Per completezza, si riporta una tabella comparativa che mette in relazione le principali caratteristiche.

Table 2.4: Confronto sintetico tra le principali soluzioni commerciali

Caratteristica	**MyQ**	**Tailwind iQ3**	**Nexx Garage**	**Smart Garage Door (progetto)**
Architettura	Cloud proprietario	Cloud + BLE	Cloud Wi-Fi	Locale + opzionale cloud
Interoperabilità	Bassa	Media	Media	Alta (MQTT/HTTP)
Funzionamento offline	No	Limitato	No	Sì (ESP8266 + Arduino)
Automazioni ingresso	GPS app	BLE	GPS	GPS + sensori locali
Automazioni uscita	No	No	No	PIR + logica locale
Privacy	Moderata	Medio-bassa	Bassa	Alta (locale-first)
Costo medio	200–250 €	150–200 €	120–150 €	< 150 €
Open-source	No	No	No	Sì

Sintesi dell'analisi competitors

Il confronto sistematico delle soluzioni esistenti evidenzia come i sistemi commerciali condividano alcune limitazioni strutturali: dipendenza dal cloud, mancanza di interoperabilità, costi elevati e scarsa autonomia locale.

Il progetto *Smart Garage Door*, al contrario, si distingue per:

- funzionamento locale anche senza Internet;
- logica distribuita su Arduino e ESP8266, senza single point of failure;
- interoperabilità garantita tramite protocolli standard aperti (HTTP/MQTT);
- architettura open-source, replicabile e didatticamente sostenibile;
- automazioni basate sia su sensori fisici sia su dati GPS, riducendo falsi positivi;
- nessuna dipendenza da cloud proprietari, migliorando privacy e sicurezza.

Questa analisi costituisce la base per la definizione delle scelte tecnologiche illustrate nel Capitolo successivo.

2.4 Analisi comparativa e limiti delle soluzioni attuali

Il confronto tra le soluzioni commerciali evidenzia alcune caratteristiche comuni:

- prevalenza di architetture **centralizzate** basate su servizi cloud, con limitata autonomia locale;
- utilizzo di protocolli di comunicazione eterogenei (HTTP, BLE, Zigbee), spesso non interoperabili tra ecosistemi differenti;
- dipendenza dalla connessione Internet per l'esecuzione delle operazioni principali;
- costi complessivi elevati (200–250 € in media), in contrasto con gli obiettivi di accessibilità e sostenibilità tipici dei progetti accademici IoT.

L'assenza di interoperabilità standard e la dipendenza dal cloud proprietario costituiscono i principali ostacoli alla diffusione di soluzioni aperte e replicabili. Come sottolineato da Piyare e Lee [**Piyare2013**], la leggerezza dei protocolli di comunicazione e la decentralizzazione dell'intelligenza locale sono elementi essenziali per garantire efficienza e resilienza in ambienti a risorse limitate. In questo senso, l'utilizzo del protocollo **MQTT** [**MQTTspec**], rispetto a soluzioni HTTP o REST basate su cloud, consente una comunicazione asincrona e affidabile con consumo energetico minimo, particolarmente adatta per scenari residenziali. L'impiego della scheda **ESP8266** [**ESP8266**] rafforza ulteriormente questa prospettiva, offrendo un equilibrio ottimale tra costo, potenza di elaborazione e connettività Wi-Fi integrata.

2.5 Contributo del progetto *Smart Garage Door*

Alla luce delle analisi precedenti, il progetto **Smart Garage Door** si propone come una soluzione innovativa e accademicamente replicabile che affronta in modo diretto le criticità delle piattaforme esistenti. Le principali caratteristiche distintive sono:

- **Architettura ibrida locale–remota**, in grado di operare anche in assenza di connessione Internet, garantendo affidabilità continua e coerenza con il requisito NFR5;
- **Adozione di componenti open source e protocolli standard** (*MQTT, Flask, Telegram API*), che riducono costi e complessità di integrazione;
- **Automazione di prossimità basata su GPS**, preferita al BLE per una maggiore accuratezza e stabilità in contesti outdoor;
- **Interfaccia utente tramite Telegram Bot API** [**TelegramAPI**] e **server Flask** [**Flask2024**], per una gestione multiutente intuitiva e sicura;
- **Budget complessivo inferiore a 150 €**, conforme ai requisiti di accessibilità e sostenibilità economica.

Il progetto, dunque, non si limita a replicare soluzioni commerciali, ma propone un modello **open-source, scalabile e autonomo**, in grado di integrare i principi di modularità, efficienza e interoperabilità alla base dell'ingegneria dei sistemi IoT moderni. Tale approccio si inserisce pienamente nella logica dello **System Development Life Cycle (SDLC)** [**Pressman2019**], applicando in modo rigoroso le fasi di analisi, progettazione, implementazione e validazione in un contesto applicativo concreto.

2.6 Conclusioni della revisione dello stato dell'arte

L'analisi dello stato dell'arte evidenzia che, nonostante l'ampia disponibilità di soluzioni commerciali per l'automazione delle porte da garage, la maggior parte di esse rimane vincolata a infrastrutture proprietarie e a modelli di comunicazione chiusi. Ciò limita la personalizzazione, aumenta i costi di mantenimento e riduce la possibilità di integrazione con altri sistemi IoT.

Il progetto *Smart Garage Door* propone dunque un paradigma alternativo rispetto alle soluzioni commerciali analizzate, fondato su un'architettura leggera, decentralizzata e pienamente controllabile dall'utente. La Tabella 2.5 riassume i principali elementi caratterizzanti attraverso un'analisi SWOT, evidenziando in modo sintetico punti di forza, limiti attuali, opportunità evolutive e potenziali minacce.

In questo modo, il progetto adotta un modello di automazione domestica locale e decentralizzata, basato su componenti trasparenti, interoperabili e a basso costo. L'assenza di dipendenze da servizi cloud e il controllo diretto da parte dell'utente contribuiscono a migliorare sicurezza e affidabilità operativa. Tale impostazione, pienamente coerente con la letteratura sui sistemi IoT distribuiti [**Palattella2016**, **Piyare2013**, **Holler2014**], costituisce una base solida per future estensioni verso ambienti domestici più integrati, scalabili e interoperabili.

Table 2.5: Analisi SWOT del sistema Smart Garage Door

Strengths	**Weaknesses**
Funzionamento locale e indipendente dal cloud; uso di componenti open e standard (ESP8266, MQTT/HTTP, Flask, Telegram); costi hardware ridotti; elevata replicabilità didattica.	Assenza di gestione utenti avanzata; scalabilità limitata a una singola autorimessa; dipendenza dalla copertura Wi-Fi domestica; sicurezza basata su meccanismi minimi.
Opportunities	**Threats**
Estensione a multi-porta e multi-utente; integrazione con ecosistemi domotici (Home Assistant, Node-RED); adozione di sensori avanzati; applicabilità a contesti condominiali o industriali.	Rischi legati alla rete Wi-Fi; possibili interferenze sui sensori PIR/ultrasuoni; dipendenza da servizi esterni (Telegram Bot API); vulnerabilità fisiche dei nodi in ambienti esterni.

Chapter 3

Requirements Analysis

3.1 Introduzione

La fase di **analisi dei requisiti** rappresenta un momento cardine all'interno del ciclo di vita del software (*System Development Life Cycle*, SDLC), poiché consente di definire in modo formale le funzionalità, le prestazioni e i vincoli che il sistema deve rispettare. Come evidenziato da Pressman e Maxim [**Pressman2019**], una corretta definizione dei requisiti costituisce la premessa fondamentale per garantire coerenza tra gli obiettivi del progetto, le soluzioni implementative e la qualità del prodotto finale.

Sulla base dello scenario delineato nel Capitolo 1 e dell'analisi dello stato dell'arte (Capitolo 2), sono stati individuati e classificati i requisiti funzionali (**FR**) e non funzionali (**NFR**) del sistema *Smart Garage Door*. Questi requisiti descrivono il comportamento atteso del sistema, ne delimitano l'ambito applicativo e guidano le successive fasi di progettazione e validazione. L'obiettivo è garantire che il sistema risulti affidabile, interoperabile, scalabile e conforme ai principi di sostenibilità tecnica ed economica propri dei progetti IoT accademici.

3.2 Scenario di riferimento

Il sistema è progettato per un ambiente domestico dotato di connettività Wi-Fi stabile fino all'area del garage o del cancello. La porta è motorizzata e controllabile elettricamente tramite un contatto digitale, in modo da consentire l'interazione diretta con i microcontrollori. Il sistema integra sensori e attuatori in un'architettura distribuita, basata su nodi cooperanti connessi tramite protocollo **MQTT** [**MQTTspec**].

L'utente, attraverso un'interfaccia intuitiva basata su **Telegram Bot API** [**TelegramAPI**], può:

- aprire e chiudere la porta del garage manualmente o da remoto;
- ricevere notifiche in tempo reale sullo stato della porta e sugli eventi rilevati;
- attivare automaticamente l'apertura o la chiusura in base alla posizione GPS del veicolo;
- gestire utenti autorizzati con livelli di accesso differenziati;
- mantenere la piena operatività del sistema anche in assenza di connettività Internet, grazie al fallback locale garantito dal microcontrollore **ESP8266** [**ESP8266**].

Il sistema, inoltre, è concepito per un funzionamento continuo 24/7 e per garantire la tracciabilità di tutte le azioni mediante log memorizzati localmente. L'ambiente di riferimento è quello di un'abitazione privata o di un garage allínterno di complesso condominiale, ma l'architettura è scalabile e può essere estesa a contesti industriali o multipiano.

3.3 Requisiti funzionali (FR)

I requisiti funzionali definiscono le azioni che il sistema deve essere in grado di compiere, descrivendo i servizi offerti all'utente e i comportamenti osservabili del sistema. Essi sono stati derivati dall'analisi delle esigenze d'uso e dai casi d'uso realistici previsti per il contesto applicativo. La Tabella 3.1 riporta l'elenco completo dei requisiti funzionali identificati.

Table 3.1: Elenco dei requisiti funzionali (FR)

ID	Nome	Descrizione
FR1	Apertura/chiusura remota	Il sistema consente all'utente di aprire o chiudere la porta del garage da remoto tramite Telegram o interfaccia web.
FR2	Consultazione stato	L'utente può verificare in tempo reale lo stato della porta (aperta, chiusa o in movimento).
FR3	Notifiche automatiche	Il sistema invia notifiche all'utente ogni volta che si verifica una variazione di stato.
FR4	Chiusura automatica temporizzata	La porta si richiude automaticamente dopo un periodo di inattività o assenza di movimento.
FR5a	Automazione in uscita	La porta si apre automaticamente quando, dall'interno, viene rilevato un movimento verso la soglia associato a un utente autorizzato.
FR5b	Automazione in ingresso	La porta si apre automaticamente al rilevamento di un utente autorizzato in avvicinamento entro un raggio configurabile (basato su GPS).
FR6	Gestione multiutenza	Il sistema consente l'aggiunta, la rimozione e la gestione di utenti autorizzati.
FR7	Comando locale / override	È possibile azionare manualmente la porta tramite pulsante fisico, indipendentemente dalla connessione di rete.
FR8	Rilevazione ostacolo	In presenza di un ostacolo, il sistema interrompe la chiusura e riapre la porta per motivi di sicurezza.
FR9	Consultazione log eventi	L'amministratore può accedere all'elenco delle azioni e degli eventi registrati dal sistema.

Questi requisiti sono coerenti con i principi di progettazione modulare e sicurezza funzionale propri dell'ingegneria dei sistemi embedded, e garantiscono l'interazione integrata tra componenti hardware e software.

3.4 Requisiti non funzionali (NFR)

I requisiti non funzionali descrivono le caratteristiche qualitative che il sistema deve possedere per assicurare un livello adeguato di prestazioni, sicurezza e usabilità. Essi influenzano direttamente le scelte tecnologiche e architetturali. La Tabella 3.2 riporta i principali NFR individuati per il sistema *Smart Garage Door*.

Le caratteristiche sopra descritte rispondono alle linee guida per i sistemi IoT distribuiti, che richiedono un equilibrio tra prestazioni, consumo e affidabilità [**Piyare2013**, **Palattella2016**]. Inoltre, l'attenzione alla sicurezza e alla privacy riflette le raccomandazioni degli standard OASIS per l'uso del protocollo MQTT in contesti sensibili [**MQTTspec**].

Table 3.2: Elenco dei requisiti non funzionali (NFR)

ID	Categoria	Descrizione
NFR1	Accessibilità	I dati devono essere sempre disponibili e consultabili, con un tempo di conservazione configurabile.
NFR2	Prestazioni	Il tempo di risposta ai comandi e alle notifiche deve essere inferiore a 1 s (95° percentile).
NFR3	Accuratezza	Il sistema deve garantire una precisione superiore al 99%, con tasso di falsi positivi inferiore all'1%.
NFR4	Copertura	La rilevazione GPS deve avvenire entro un raggio massimo di 15 m dal punto di riferimento.
NFR5	Disponibilità	Il sistema deve garantire operatività continua (24/7), con modalità di fallback locale in caso di perdita di connessione.
NFR6	Sicurezza	Devono essere implementati meccanismi di autenticazione e integrità dei dati tramite hashing e gestione sicura delle sessioni.
NFR7	Privacy	I dati personali devono essere minimizzati e i log cancellati automaticamente dopo un periodo definito.
NFR8	Interoperabilità	Il sistema deve supportare protocolli standard aperti (MQTT, HTTP) per la compatibilità multi-piattaforma.
NFR9	Efficienza energetica	Il consumo deve essere ottimizzato per dispositivi alimentati a batteria o connessi a rete domestica.
NFR10	Costo	Il costo complessivo del sistema non deve superare i 150 euro, garantendo sostenibilità economica.

3.5 Tracciabilità FR–NFR

Per assicurare coerenza progettuale, è stata redatta una matrice di tracciabilità tra requisiti funzionali e non funzionali. La Tabella 3.3 evidenzia le relazioni di dipendenza, mostrando come ciascun FR sia associato ai NFR che ne influenzano l'implementazione e la verifica.

Table 3.3: Tracciabilità tra requisiti funzionali e non funzionali

FR	NFR impattati
FR1 – Apertura/chiusura remota	NFR2, NFR6, NFR8, NFR10
FR2 – Stato porta in tempo reale	NFR1, NFR2, NFR3
FR3 – Notifiche automatiche	NFR1, NFR2, NFR7
FR4 – Chiusura automatica temporizzata	NFR3, NFR5, NFR9
FR5a – Automazione in uscita	NFR3, NFR4, NFR9
FR5b – Automazione in ingresso	NFR3, NFR4, NFR9
FR6 – Gestione multiutenza	NFR6, NFR7, NFR8
FR7 – Comando locale / override	NFR5, NFR8
FR8 – Rilevazione ostacolo	NFR3, NFR5
FR9 – Log eventi	NFR1, NFR6, NFR7

La tracciabilità consente di mantenere un controllo diretto sull'impatto di ogni requisito non funzionale sul comportamento del sistema e di pianificare test mirati in fase di validazione [**Pressman2019**].

3.6 Analisi comparativa delle tecnologie disponibili

In conformità alle linee guida del corso Internet of Things, è necessario valutare e confrontare le alternative hardware, i sensori, i protocolli di comunicazione e le interfacce utente potenzialmente impiegabili nel sistema Smart Garage Door. Tale confronto è basato sui requisiti non

funzionali (NFR) definiti nelle sezioni precedenti e consente di motivare in modo oggettivo le scelte progettuali.

3.6.1 Confronto tra microcontrollori

Table 3.4: Confronto tra microcontrollori disponibili

Parametro	**Arduino UNO**	**NodeMCU ESP8266**	**Raspberry Pi 3**
Connettività	Nessuna nativa	Wi-Fi 2.4 GHz integrato	Wi-Fi, BT, Ethernet
Consumo energetico	Molto basso	Basso	Elevato
Potenza di calcolo	Limitata	Media	Alta
Costo	8–12€	5–10€	40–60€
Programmazione	C/C++	C/C++ / MicroPython	Python/Linux
I/O digitali	14 pin	11 pin	+20 GPIO
Uso in IoT	Necessita modulo Wi-Fi esterno	Ideale per IoT Wi-Fi	Sovradimensionato
Adatto per Smart Garage Door	Sì (sensori/relè)	Sì (comunicazione)	No (costo/consumo eccessivi)

3.6.2 Confronto tra sensori

Table 3.5: Confronto tra sensori utilizzabili per automazioni e sicurezza

Parametro	**PIR**	**IR**	**Radar Doppler**	**HC-SR04**
Tipo rilevazione	Movimento termico	Ostacoli vicini	Movimento microonde	Distanza oggetti
Accuratezza	Alta indoor	Media	Molto alta	Alta
Falsi positivi	Bassi	Medi (luce)	Bassi	Bassi
Distanza utile	3–6 m	20–80 cm	5–10 m	2–400 cm
Costo	2–4€	1–2€	8–12€	2–4€
Consumo	Molto basso	Basso	Medio	Basso
Adatto a FR5a (uscita)	Sì	No	Sì	Parziale
Adatto a FR8 (ostacolo)	No	No	No	Sì

3.6.3 Confronto tra protocolli di comunicazione

Table 3.6: Confronto tra protocolli di comunicazione per sistemi IoT

Parametro	**HTTP/REST**	**MQTT**	**CoAP**	**Webhook**
Pattern	Client–Server	Publish/Subscribe	Client–Server (UDP)	Event callback
Peso messaggi	Alto	Molto basso	Basso	Medio
Affidabilità	Alta	Alta (QoS)	Media (UDP)	Dipende dal server
Reattività	Media	Altissima	Alta	Alta
Sicurezza	TLS/HTTPS	TLS	DTLS	HTTPS
Difficoltà implementazione	Bassa	Media	Media	Alta lato client
Adatto a notifiche	Sì	Ottimo	Sì	Ottimo
Adatto per Smart Garage Door	Sì	Sì	Non necessario	Possibile

3.6.4 Confronto tra interfacce utente

Table 3.7: Confronto tra possibili interfacce utente

Parametro	**Telegram Bot**	**Web App**	**App nativa**
Costo sviluppo	Nessuno	Medio	Alto
Usabilità	Molto alta	Alta	Alta
Notifiche push	Immediate	Necessarie API esterne	Native
Installazione	Nessuna	Browser	Store (Android/iOS)
Sicurezza	Elevata (TLS + Bot API)	Dipende dal server	Alta
Multiplatform	Totale	Totale	Totale
Adatta al progetto	Sì (migliore)	Opzionale	Non necessaria

3.7 Scelta finale delle tecnologie

La scelta delle tecnologie da impiegare nella realizzazione del sistema *Smart Garage Door* rappresenta un passaggio cruciale all'interno della fase di analisi, poiché condiziona direttamente la progettazione architetturale, l'implementazione e le future attività di test e validazione. Come evidenziato da Pressman e Maxim [**Pressman2019**], le decisioni tecnologiche devono essere il risultato di un processo razionale basato su confronti oggettivi, vincoli progettuali e requisiti funzionali e non funzionali.

Le sezioni precedenti hanno proposto un confronto sistematico e approfondito tra microcontrollori, sensori, protocolli di comunicazione e interfacce utente, mettendo in evidenza punti di forza, limiti e aspetti di idoneità rispetto alle esigenze dello scenario applicativo. Le tecnologie selezionate nelle tabelle comparative costituiscono ora una base motivata per definire lo stack tecnologico più adeguato in termini di affidabilità, scalabilità, costo ed efficienza energetica, in linea con i principi dei sistemi IoT descritti da Holler et al. [**Holler2014**] e Palattella et al. [**Palattella2016**].

3.7.1 Microcontrollori

Il confronto riportato nella Tabella 3.4 mostra come i diversi microcontrollori disponibili nel kit (Arduino UNO, NodeMCU ESP8266 e Raspberry Pi 3) presentino caratteristiche funzion-

ali, computazionali ed energetiche profondamente differenti, riflettendo trade-off tipici della progettazione embedded [**Piyare2013**].

L'**Arduino UNO** si distingue per la sua affidabilità, la latenza minima, la semplicità di programmazione e il controllo diretto degli I/O digitali, elementi fondamentali per la gestione dei sensori di prossimità, del pulsante manuale e del relè che controlla l'azionamento della porta. Inoltre, il basso consumo energetico e la totale prevedibilità del comportamento in tempo reale lo rendono particolarmente adatto a funzioni critiche come la rilevazione ostacoli (FR8) e l'attivazione manuale di emergenza (FR7).

Il **NodeMCU ESP8266** costituisce invece il nodo ideale per la componente di comunicazione del sistema. La disponibilità di Wi-Fi integrato, il supporto nativo ai protocolli MQTT e HTTP, un consumo ridotto e un costo estremamente contenuto lo rendono un dispositivo perfettamente conforme agli NFR di costo (NFR10), disponibilità (NFR5) ed efficienza energetica (NFR9). La presenza di un ambiente di sviluppo maturo e di una comunità estremamente attiva facilita inoltre l'integrazione con piattaforme esterne come Telegram, ThingSpeak e servizi RESTful [**MQTTspec**, **ESP8266**].

La **Raspberry Pi 3**, pur offrendo performance superiori, viene scartata poiché il suo impiego comporterebbe complessità non necessarie, un consumo energetico superiore, costi incompatibili con gli NFR e un approccio sistemistico sovradimensionato rispetto alle esigenze del progetto. La scelta finale prevede dunque una **architettura a due microcontrollori**, nella quale:

- l'**Arduino UNO** esegue le operazioni di basso livello, garantendo determinismo e risposta immediata;
- il **NodeMCU ESP8266** svolge la funzione di *gateway IoT*, gestendo comunicazioni, logica applicativa e interazione con l'utente.

Questa separazione rispecchia il paradigma *perception layer – network/application layer* tipico delle architetture IoT multilivello [**Holler2014**].

3.7.2 Sensori

La Tabella 3.5 evidenzia che sensori differenti (PIR, IR, radar Doppler e HC-SR04) presentano specificità funzionali che li rendono più o meno adatti a diversi compiti all'interno del sistema.

Il **sensore PIR**, basato sulla rilevazione di variazioni nell'infrarosso passivo, risulta la soluzione ottimale per l'automazione in uscita (FR5a): garantisce consumi minimi, elevata affidabilità in ambienti indoor e un tasso molto ridotto di falsi positivi. L'impiego del PIR è coerente con gli NFR relativi all'efficienza energetica (NFR9) e all'accuratezza (NFR3).

L'**HC-SR04**, grazie al suo principio di funzionamento ultrasonico, permette una misurazione precisa delle distanze ed è ampiamente utilizzato in applicazioni di rilevazione ostacoli. La sua elevata stabilità, unita al basso costo, lo rende ideale per garantire la sicurezza meccanica della porta (FR8) e soddisfare gli NFR di disponibilità (NFR5) e accuratezza (NFR3).

I sensori IR e radar, pur offrendo vantaggi in alcuni scenari, sono stati scartati rispettivamente per:

- interferenza luminosa e minore robustezza operativa degli IR;
- costo e complessità non giustificati dei radar Doppler in un ambiente domestico.

3.7.3 Protocolli di comunicazione

Il confronto in Tabella 3.6 mette a confronto HTTP/REST, MQTT, CoAP e Webhooks. Secondo Cirani et al. [**Cirani2019**], la scelta dei protocolli in sistemi IoT deve tener conto del modello di comunicazione, della frequenza degli aggiornamenti, della latenza e dei requisiti energetici dei dispositivi.

Il protocollo **HTTP/REST** si rivela la scelta più naturale per la gestione dei comandi puntuali e delle interrogazioni di stato (FR1, FR2, FR7), grazie alla semplicità implementativa, alla disponibilità di librerie mature e alla piena compatibilità con la Telegram Bot API. La natura stateless del protocollo garantisce inoltre una separazione chiara tra client e server, favorendo la scalabilità e la manutenzione [**Guinard2016**].

Il protocollo **MQTT**, progettato per comunicazioni machine-to-machine leggere, garantisce notifiche efficienti e affidabilità tramite i livelli QoS. Benché non strettamente necessario nello stadio iniziale, esso rappresenta un'opzione strategica per estensioni future (NFR8), ad esempio per notifiche asincrone in tempo reale o per supportare più dispositivi in scenari multi-utente.

Il protocollo **CoAP**, pur essendo ideale in reti altamente vincolate, risulta meno vantaggioso in un contesto Wi-Fi domestico, dove le risorse disponibili non giustificano l'adozione di un modello basato su UDP [**Shelby2014**]. I Webhooks vengono scartati per l'assenza di un server esterno persistente, in quanto richiederebbero un'infrastruttura non necessaria in un progetto embedded.

3.7.4 Interfaccia utente

La scelta dell'interfaccia utente è un elemento critico per assicurare una buona esperienza d'uso e al tempo stesso mantenere sostenibilità economica e semplicità architetturale. La comparativa in Tabella 3.7 evidenzia che:

- Le **Web App** richiedono certificati HTTPS, hosting dedicato e manutenzione.
- Le **app native** presentano costi e complessità di sviluppo incompatibili con i vincoli del progetto.
- La **Telegram Bot API** offre notifiche push integrate, autenticazione built-in, forte sicurezza tramite TLS e totale assenza di costi infrastrutturali [**TelegramAPI**].

Telegram soddisfa inoltre pienamente gli NFR relativi alla sicurezza (NFR6), alla privacy (NFR7) e alla multi-piattaforma. Il paradigma conversazionale consente inoltre di semplificare l'interazione utente in modo naturale e intuitivo [**Guinard2016**].

3.7.5 Sintesi delle scelte tecnologiche

L'insieme delle tecnologie selezionate configura un'architettura IoT modulare, robusta e pienamente aderente agli FR e NFR identificati. La combinazione:

- Arduino UNO come controllore dei sensori e del relè;
- ESP8266 come nodo di rete e gestore della logica;
- Sensori PIR e HC-SR04 per automazione e sicurezza;
- Protocolli HTTP/REST e, in prospettiva, MQTT;
- Telegram Bot come interfaccia utente primaria,

consente di ottenere un sistema equilibrato tra affidabilità, efficienza energetica, sicurezza e sostenibilità economica. Questa configurazione costituisce la base per la progettazione architetturale descritta nel Capitolo 4, garantendo un corretto allineamento con il ciclo di vita del software e con le migliori pratiche di progettazione IoT.

3.8 Riepilogo FR – Soluzioni Hardware e Software

La Tabella 3.8 riassume l'associazione tra ciascun requisito funzionale (FR) e le soluzioni hardware, software e protocollari adottate per soddisfarlo. Questa struttura consente di evidenziare la completa tracciabilità tra requisiti, componenti fisici e scelte implementative, in accordo con le linee guida del corso e con il modello SDLC.

Questa tabella chiude formalmente la fase di Analisi, mostrando come ogni funzione richiesta sia stata tradotta in una scelta tecnologica concreta e motivata dagli NFR. Essa prepara inoltre la transizione verso il Capitolo 4, in cui tali soluzioni verranno integrate nell'architettura di sistema all'interno dei tre livelli (Perception, Network e Application Layer).

Table 3.8: Mappatura tra requisiti funzionali e soluzioni HW/SW adottate

FR	**Hardware coinvolto**	**Software / Logica**	**Protocollo / Interfaccia**
FR1 – Apertura/chiusura remota	Relay + Arduino UNO + ESP8266	Flask API: endpoint /garage/door/open–close; logica di controllo	HTTP/REST via Telegram Bot API
FR2 – Consultazione stato porta	Arduino UNO (stato relè / sensori) + ESP8266	Endpoint /garage/door/state; sincronizzazione MQTT–Flask	HTTP/REST (GET)
FR3 – Notifiche automatiche	ESP8266 (publisher MQTT)	Event handler Flask + Telegram Bot notifier	MQTT + Telegram API
FR4 – Chiusura automatica temporizzata	Arduino UNO (timer locale)	Timer interno + logica anti-false activation	Nessun protocollo esterno (logica locale)
FR5a – Automazione in uscita (PIR)	Sensore PIR + Arduino UNO + relè	Algoritmo combinato movimento → attuazione porta	UART (Arduino→ESP) + logica locale
FR5b – Automazione in ingresso (GPS)	Modulo GPS (via ESP8266)	Geofence evaluator + trigger apertura	HTTP/REST (GET /gps/status) + UART
FR6 – Gestione multiutenza	ESP8266 + Flask Server	DB utenti, permessi e autenticazione bot	Telegram Bot API + Flask
FR7 – Comando locale (pulsante)	Pulsante fisico + Arduino UNO	Interrupt/ISR su Arduino; override logica remota	Pure local control (offline mode)
FR8 – Rilevazione ostacolo	HC-SR04 + Arduino UNO	Misura distanza + stop + inversione porta	UART (segnalazione stato)
FR9 – Consultazione log eventi	ESP8266 + Flask Server	Endpoint /garage/logs; registrazione eventi	HTTP/REST + ThingSpeak API (opzionale)

3.9 Impatto dei requisiti non funzionali sulle decisioni progettuali

I requisiti non funzionali (NFR) hanno guidato in modo determinante tutte le scelte hardware, software e protocollari del sistema. La Tabella 3.9 sintetizza come ciascun NFR abbia influenzato le decisioni progettuali chiave, garantendo coerenza metodologica e aderenza al modello SDLC.

Table 3.9: Mappatura tra NFR e decisioni progettuali adottate

NFR	**Descrizione**	**Decisioni progettuali derivate**
NFR1 – Accessibilità dei dati	I dati devono essere sempre disponibili e consultabili	Adozione di endpoint REST leggibili; server Flask con log persistenti; periodic upload verso ThingSpeak.
NFR2 – Prestazioni	Tempo risposta < 1s	Separazione Arduino (real-time) / ESP8266 (network); uso di HTTP/REST per comandi rapidi; logica locale senza round-trip cloud.
NFR3 – Accuratezza	Precisione $> 99\%$	Uso di PIR (alta affidabilità indoor) e HC-SR04; escluso IR perché sensibile alla luce; escluso radar per falsi doppi.
NFR4 – Copertura GPS	Raggio massimo 15m	Implementazione di geofence lato ESP8266; integrazione FakeGPS per test controllati.
NFR5 – Disponibilità / Operatività offline	Sistema attivo 24/7 anche senza Internet	Architettura ibrida locale–remota: Arduino mantiene il controllo anche in assenza di rete; override fisico; logica di fallback.
NFR6 – Sicurezza	Integrità, autenticazione	Telegram Bot API (autenticazione built-in); endpoint POST protetti; messaggi su UART con codifica semplificata.
NFR7 – Privacy	Minimizzazione dati personali	Nessun cloud proprietario; log conservati solo localmente; GPS usato solo per geofence senza storico.
NFR8 – Interoperabilità	Protocolli standard aperti	Adozione di HTTP/REST e MQTT; uso di JSON; struttura ROA coerente con standard IoT.
NFR9 – Efficienza energetica	Ottimizzazione consumi	ESP8266 e Arduino scelti per basso assorbimento; escluso Raspberry Pi (consumo troppo alto).
NFR10 – Costo massimo 150€	Sostenibilità economica	Scartato Raspberry Pi; scartati sensori radar; uso di moduli low-cost (PIR, HC-SR04, ESP8266).

Chapter 4

System Design

4.1 Introduzione

La fase di *system design* rappresenta il passaggio intermedio tra l'analisi dei requisiti (Capitolo 3) e l'implementazione del prototipo (Capitolo 5). In questa fase vengono definite le scelte architetturali, tecnologiche e organizzative necessarie a trasformare i requisiti funzionali (FR) e non funzionali (NFR) in un sistema reale, verificabile e coerente con il contesto operativo.

Il progetto *Smart Garage Door* nasce infatti da uno scenario concreto: l'automazione dell'apertura di un portone garage domestico, in un ambiente reale caratterizzato dalla presenza di connettività Wi-Fi non omogenea, vincoli economici, necessità di sicurezza e requisiti di continuità operativa anche in assenza della rete. Per tale motivo, il system design si fonda su un insieme esplicito di assunzioni progettuali, che guidano la scelta delle tecnologie e dei protocolli più adatti all'ambiente di utilizzo.

In questo capitolo viene quindi definita l'architettura complessiva del sistema secondo il modello IoT a tre livelli (Perception, Network, Application), evidenziando:

- le alternative tecnologiche considerate per ciascun sottosistema (microcontrollore, comunicazione, sensori, interfaccia utente);
- i criteri con cui tali alternative sono state valutate rispetto ai requisiti FR e NFR;
- la motivazione delle scelte finali, basate su compatibilità, affidabilità, costo e semplicità di integrazione.

Il risultato è un'architettura modulare, scalabile e pienamente allineata ai principi dello **System Development Life Cycle (SDLC) [Pressman2019]**, nella quale ogni componente — sensore, microcontrollore o modulo software — comunica attraverso interfacce standard e protocolli aperti, riducendo la complessità, aumentando la manutenibilità e garantendo la possibilità di evoluzione futura del sistema.

4.2 Architettura generale

L'architettura del sistema *Smart Garage Door* è stata progettata secondo il modello a tre livelli tipico dei sistemi IoT moderni, al fine di garantire modularità, scalabilità e una chiara separazione delle responsabilità funzionali. Questo paradigma identifica tre domini principali: **Perception**, **Network** e **Application**, ciascuno responsabile di una porzione distinta della catena informativa.

1. **Livello di percezione (Perception Layer)** Comprende i dispositivi fisici incaricati dell'interazione con l'ambiente: il sensore PIR per la rilevazione del movimento interno, il modulo GPS per la geolocalizzazione e il relè per l'attuazione del motore della porta

del garage. L'elaborazione locale è affidata al microcontrollore **Arduino UNO**, che garantisce la gestione in tempo reale dei segnali e il funzionamento autonomo anche in assenza di connettività di rete.

2. **Livello di rete (Network Layer)** Ha il compito di collegare i dispositivi locali ai servizi applicativi remoti. Tale funzione è svolta dal modulo **NodeMCU ESP8266**, che fornisce connettività Wi-Fi e implementa il protocollo **MQTT** [**MQTTspec**] per lo scambio asincrono dei messaggi. Il broker MQTT (Mosquitto) gestisce il traffico *publish/subscribe* tra i nodi e il server centrale.
3. **Livello applicativo (Application Layer)** Comprende la logica ad alto livello e l'interfaccia con l'utente finale. Il server **Flask** [**Flask2024**] funge da API gateway e componente di orchestrazione, gestendo comandi, log ed eventi. L'interfaccia utente è implementata mediante un bot **Telegram** [**TelegramAPI**].

In questa architettura, il nodo locale costituito da **Arduino** + **ESP8266** funge da unità edge intelligente: Arduino gestisce sensori e attuatori, mentre il NodeMCU agisce da gateway di rete, traducendo i comandi remoti in istruzioni locali e pubblicando lo stato del sistema tramite MQTT. Il server Flask, a sua volta, centralizza la logica applicativa e garantisce una comunicazione coerente tra rete locale, interfaccia utente e piattaforme di monitoraggio, mantenendo la separazione tra livello fisico e livello di controllo.

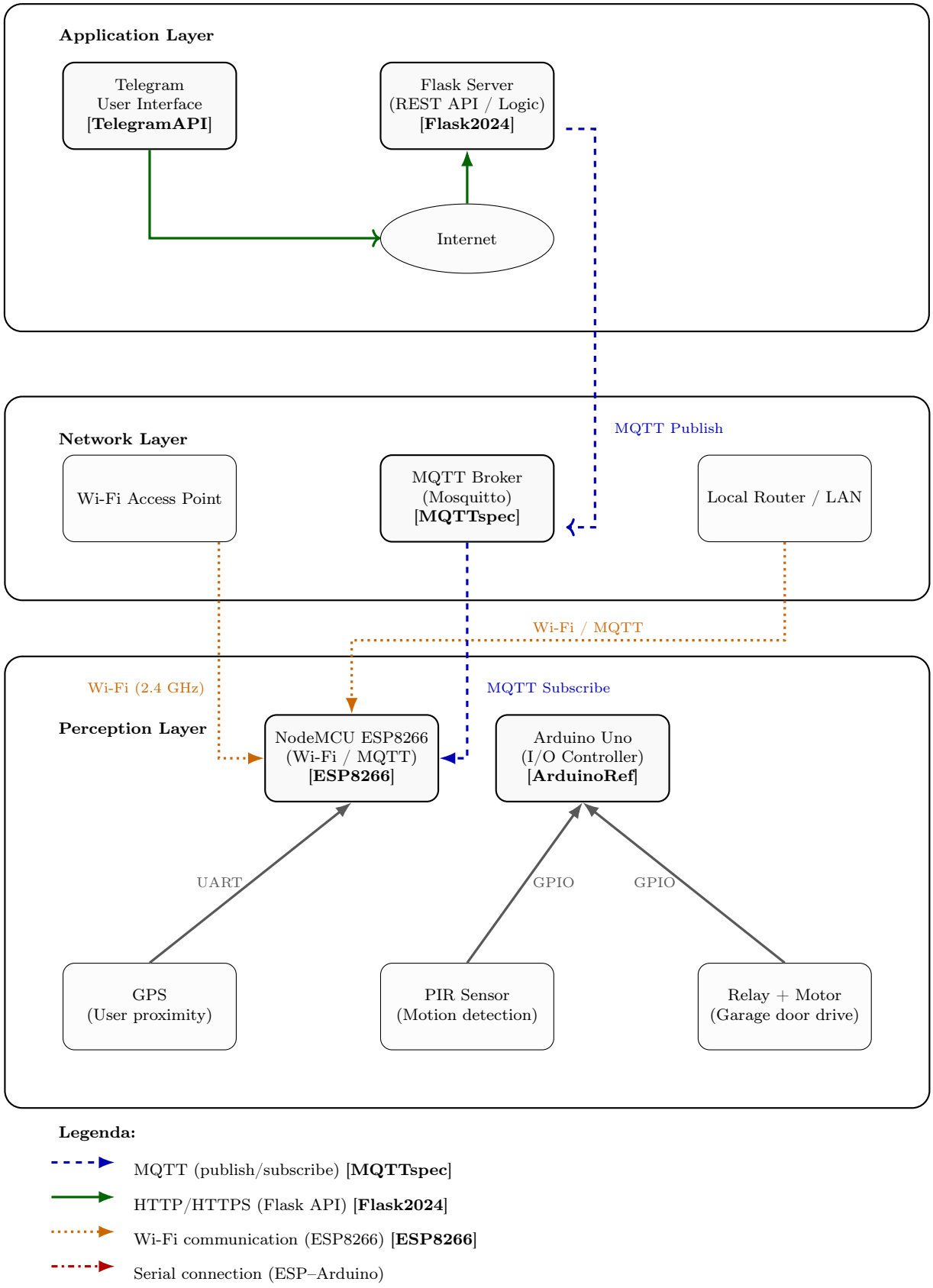

Figure 4.1: Architettura del sistema *Smart Garage Door* con spaziatura verticale ampia. Il diagramma evidenzia la netta separazione tra i livelli *Application*, *Network* e *Perception*, facilitando la lettura dei flussi informativi bottom–up e top–down tra sensori, nodi di rete e servizi applicativi [**MQTTspec**, **ESP8266**, **Flask2024**, **TelegramAPI**, **ThingSpeak**].

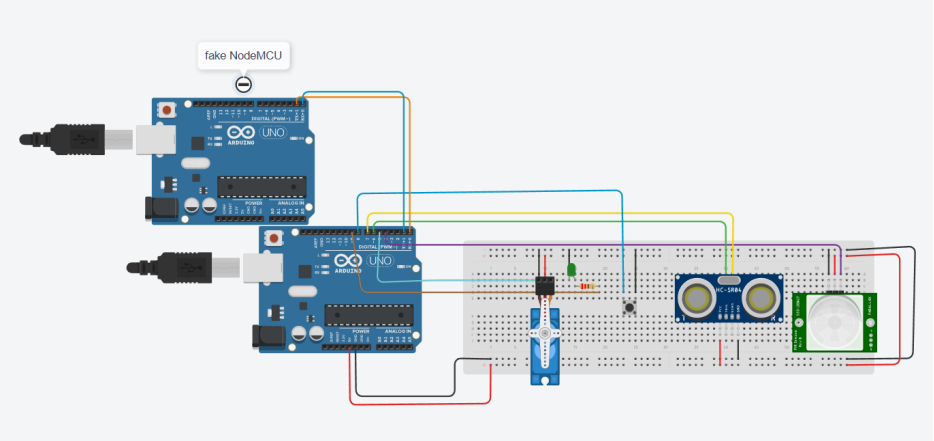

Figure 4.2: Prototipo simulato in Tinkercad dell'architettura hardware basata su Arduino UNO. Nel circuito sono presenti: un modulo PIR per la rilevazione del movimento, un sensore a ultrasuoni HC-SR04 per la misurazione della distanza, un pulsante per il comando manuale e un servomotore impiegato come attuatore meccanico del sistema. Il microcontrollore esegue la logica di controllo locale e comunica con l'unità remota (simulata tramite un secondo Arduino configurato come fake NodeMCU) per la gestione dei comandi e degli eventi. Il prototipo costituisce il modello fisico di riferimento per l'architettura descritta in Figura 4.3.

4.2.1 Digital Twin e mappatura delle risorse REST

La progettazione dell'*Application Layer* richiede l'associazione esplicita tra gli oggetti fisici presenti nel sistema (*Real Objects*) e le corrispondenti rappresentazioni digitali (*Digital Twins*). Tale mappatura consente di esporre funzionalità e stati tramite risorse REST, in accordo con il paradigma **Resource-Oriented Architecture (ROA)** e le linee guida sulla progettazione di API per sistemi IoT [**Guinard2016**].

La Tabella 4.1 riporta l'associazione completa tra entità fisiche, risorse digitali, operazioni consentite e path HTTP implementati dal server applicativo basato su *Flask*.

Table 4.1: Mappatura tra oggetti fisici e risorse digitali (Digital Twin)

Oggetto fisico	**Digital Twin**	**Metodo**	**Path REST**	**Descrizione**
Porta garage (attuatore)	/door/state	GET	/garage/door/state	Restituisce lo stato corrente della porta (aperta, chiusa, in movimento).
Porta garage (comando apertura)	/door/open	POST	/garage/door/open	Invoca l'azione di apertura tramite relè controllato da Arduino.
Porta garage (comando chiusura)	/door/close	POST	/garage/door/close	Attiva la procedura di chiusura sicura della porta.
Sensore PIR (movimento interno)	/pir/value	GET	/garage/pir	Restituisce il valore del sensore PIR per l'automazione in uscita (FR5a).
Sensore ultrasonico HC-SR04	/ultrasonic/distance	GET	/garage/ultrasonic	Fornisce la distanza rilevata per il rilevamento ostacoli (FR8).
Sistema GPS (nodo ESP8266)	/gps/status	GET	/garage/gps	Indica se l'utente si trova entro il geofence configurato (FR5b).
Log eventi	/logs	GET	/garage/logs	Restituisce gli eventi registrati localmente (aperture, chiusure, notifiche).
Comando di emergenza locale (pulsante)	/override	POST	/garage/override	Simula la pressione del pulsante fisico, garantendo il comando locale (FR7).

Questa rappresentazione consente di definire un'astrazione chiara e standardizzata tra mondo fisico e digitale, facilitando l'integrazione con servizi esterni (es. Telegram Bot, ThingSpeak) e garantendo un'implementazione conforme ai principi REST e alle best practice dei sistemi IoT distribuiti.

4.3 Flusso dei dati e comunicazione

Il flusso dei dati del sistema *Smart Garage Door* è strutturato secondo un modello di comunicazione **asincrono e bidirezionale**, fondato sul protocollo **MQTT** [**MQTTspec**]. Questa scelta consente di garantire bassa latenza, ridotto overhead di rete e un'elevata affidabilità nella trasmissione tra nodi embedded e livello applicativo, anche in presenza di connessioni Wi-Fi non ottimali.

Nel sistema proposto, i microcontrollori locali svolgono la funzione di nodi edge:

- **Arduino UNO** acquisisce i segnali provenienti dai sensori (PIR, modulo ultrasonico, GPS tramite ESP) e controlla l'attuatore (relè);
- **NodeMCU ESP8266** funge da gateway di rete, inoltrando eventi e comandi tramite MQTT.

Parallelamente, il server **Flask** esegue la logica applicativa, si sottoscrive ai topic MQTT per ricevere aggiornamenti e rende disponibili le API utilizzate dall'interfaccia Telegram.

Fasi principali del flusso informativo

Il percorso dei dati attraverso i diversi livelli del sistema può essere descritto come segue:

1. **Generazione dell'evento (Perception Layer)** Un sensore genera un input:
   - il PIR rileva movimento (FR5a);
   - il modulo GPS rileva ingresso o uscita dal geofence (FR5b);
   - il sensore ultrasonico rileva ostacoli durante la chiusura (FR8).
2. **Elaborazione locale (Arduino UNO)** Arduino interpreta l'evento e, in base alla logica implementata, attiva o disattiva il relè, determina un timeout di chiusura (FR4) e aggiorna lo stato locale.
3. **Trasmissione verso il gateway (ESP8266)** Lo stato viene inviato ad ESP8266 tramite UART. ESP serializza il dato e lo pubblica sul topic MQTT appropriato.
4. **Routing e gestione dei messaggi (MQTT Broker)** Il broker Mosquitto riceve il messaggio e lo inoltra a tutti i client sottoscritti, tra cui il server Flask.
5. **Elaborazione applicativa (Flask Server)** Flask aggiorna lo stato interno, registra l'evento e, se configurato, esegue un push dei dati su ThingSpeak per la visualizzazione remota.
6. **Interazione con l'utente (Telegram Bot)** L'utente può inviare comandi remoti tramite Telegram (/on, /off, /status). Il bot inoltra il comando al server Flask, che lo traduce in un'azione MQTT verso il nodo ESP→Arduino.

Questo paradigma data-driven, fondato su scambio asincrono e loosely coupling, consente al sistema di rimanere reattivo anche sotto condizioni di latenza variabile o perdita temporanea di pacchetti, garantendo robustezza e scalabilità [**Palattella2016**, **Piyare2013**]. In particolare, la logica locale su Arduino assicura il funzionamento autonomo anche in caso di assenza di rete (NFR5).

4.4 Componenti hardware e software

4.4.1 Componenti hardware

Il prototipo *Smart Garage Door* è stato realizzato utilizzando componenti a basso costo, facilmente reperibili e compatibili con le esigenze di modularità, espandibilità e vincolo economico (NFR10). La selezione dell'hardware si è basata sulla disponibilità commerciale, sul

supporto della comunità open source e sulla capacità dei moduli di soddisfare i requisiti funzionali FR1–FR8.

I principali elementi hardware impiegati sono:

- **Arduino UNO** – microcontrollore dedicato alla gestione del *Perception Layer*. Gestisce sensori, attuatori, temporizzazioni e tutte le logiche locali di sicurezza. Garantisce il funzionamento anche in assenza di rete (NFR5).
- **NodeMCU ESP8266** – modulo Wi-Fi incaricato della connettività con il server. Pubblica stati ed eventi tramite MQTT e funge da *gateway* tra microcontrollore locale e Application Layer.
- **Modulo GPS NEO-6M** – utilizzato per implementare l'automazione basata sulla prossimità del veicolo (FR5b). Comunica con ESP8266 tramite seriale UART.
- **Sensore PIR HC-SR501** – rileva movimento interno per l'automazione locale in uscita (FR5a).
- **Relè 5 V** – attuatore che comanda il motorino del garage. È controllato da Arduino tramite un pin digitale dedicato.
- **Alimentazione 5 V / 2 A** – fornisce l'energia necessaria a entrambi i microcontrollori e ai moduli esterni.

L'interoperabilità tra le componenti è garantita dall'uso di interfacce standard (UART, GPIO, Wi-Fi) e da una struttura elettrica semplificata che facilita manutenzione e replicabilità. La Figura 4.3 illustra la relazione tra i moduli e la loro interconnessione a livello fisico.

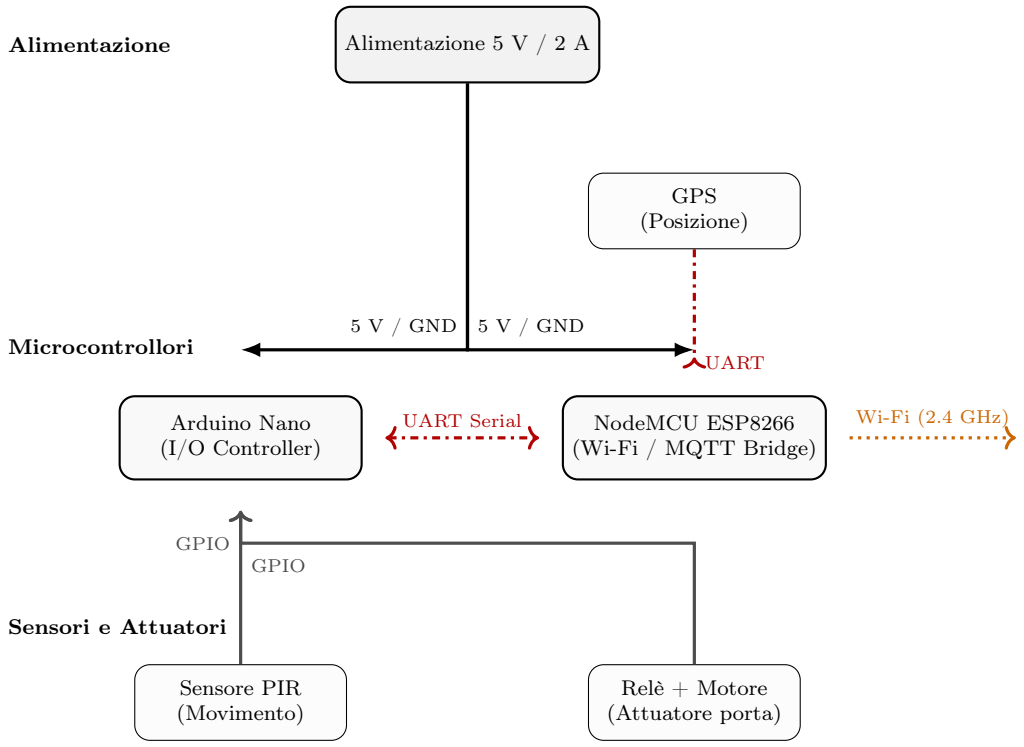

Figure 4.3: Architettura hardware del sistema *Smart Garage Door*. I collegamenti tra i moduli sono spaziati per garantire chiarezza visiva: l'alimentazione 5 V/2 A fornisce energia a entrambi i microcontrollori, mentre l'ESP8266 comunica con l'Arduino tramite interfaccia UART e connessione Wi-Fi. Il modulo GPS, il sensore PIR e il relè/motore sono gestiti tramite linee dedicate (UART e GPIO).

4.4.2 Componenti software

Il livello applicativo del sistema si basa su tecnologie **open source** che garantiscono modularità, portabilità e semplicità di integrazione tra i diversi sottosistemi. Le scelte software sono state effettuate in modo da soddisfare i requisiti di affidabilità, scalabilità e sicurezza (NFR2, NFR3, NFR6) e per mantenere la coerenza con il paradigma IoT a tre livelli.

I principali componenti software impiegati sono:

- **Python 3 + Flask [Flask2024]** – utilizzati per implementare il *Application Layer*. Flask funge da server web leggero e da gateway REST, gestendo comandi, logica applicativa, integrazione con il broker MQTT e interfaccia con il bot Telegram.
- **MQTT Broker (Mosquitto) [MQTTspec]** – elemento centrale del *Network Layer*, responsabile dello scambio di messaggi tra i nodi embedded (ESP8266, Arduino) e il server. Il modello publish/subscribe garantisce comunicazione asincrona, efficienza e bassa latenza.
- **Telegram Bot API [TelegramAPI]** – utilizzata per realizzare l'interfaccia conversazionale del sistema, permettendo all'utente di inviare comandi remoti (/on, /off, /status) e ricevere notifiche in tempo reale.
- **ThingSpeak API [ThingSpeak]** – piattaforma cloud per la telemetria e la visualizzazione dei dati. Viene impiegata per registrare eventi significativi e monitorare lo stato del sistema nel tempo.
- **Arduino IDE 2.3 [ArduinoRef]** – ambiente di sviluppo utilizzato per programmare i firmware dei microcontrollori Arduino UNO e NodeMCU ESP8266.

A supporto di questi strumenti, il progetto utilizza librerie ampiamente diffuse nella comunità IoT:

- **PubSubClient** – per la gestione MQTT sul modulo ESP8266;
- **TinyGPSPlus** – per la decodifica dei messaggi NMEA provenienti dal modulo GPS;
- **python-telegram-bot** – per la gestione asincrona del bot Telegram;
- **requests** – per le comunicazioni HTTP con ThingSpeak e componenti esterni.

L'integrazione tra questi strumenti consente di ottenere un sistema software leggero, modulare e facilmente estendibile. L'adozione esclusiva di tecnologie open source favorisce inoltre la replicabilità del progetto, in linea con i requisiti di economicità (NFR10) e con le buone pratiche di progettazione di sistemi IoT distribuiti.

4.5 Interfacce e sicurezza

L'architettura del sistema *Smart Garage Door* prevede un insieme di interfacce e meccanismi di sicurezza progettati per garantire l'integrità, la disponibilità e la riservatezza delle comunicazioni tra i vari moduli. In linea con i requisiti non funzionali (NFR6–NFR8), ogni scambio informativo è regolato da controlli espliciti di autenticazione, validazione dei messaggi e gestione degli accessi.

Interfacce di comunicazione

Le principali interfacce utilizzate nel sistema sono:

- **UART (Arduino <-> ESP8266)** Impiegata per il trasferimento locale di comandi e stati. La comunicazione è strutturata su frame a byte singolo per ridurre complessità e garantire determinismo temporale.

- **MQTT (ESP8266 <-> Broker Flask)** Utilizzato come protocollo principale per lo scambio asincrono di eventi. I topic sono organizzati secondo un modello gerarchico (ad es. `home/garage/cmd`, `home/garage/status`) che permette di separare funzioni e privilegi.
- **HTTP/HTTPS (Telegram <-> Flask)** La Telegram Bot API comunica con il server tramite richieste REST, fornendo un'interfaccia stateless robusta e semplice da estendere.
- **HTTP verso ThingSpeak** Utilizzato per la telemetria remota e per la conservazione delle informazioni operative in formato temporale.

Queste interfacce implementano un modello di comunicazione *loosely coupled*, che favorisce scalabilità e indipendenza tra i sottosistemi, riducendo l'impatto di eventuali guasti locali.

Sicurezza dei dati e autenticazione

Per garantire la protezione delle informazioni e prevenire accessi non autorizzati, sono stati adottati i seguenti meccanismi:

- **Identificazione dell'utente Telegram** Ogni comando remoto è associato all'ID univoco dell'utente. Solo gli utenti autorizzati (whitelist) possono inviare comandi di apertura o modifica dello stato del sistema.
- **Token di sessione e API key** Le richieste verso Flask sono validate tramite una API key generata e conservata lato server, impedendo l'invio di comandi da fonti non autorizzate.
- **Integrità dei messaggi MQTT** Prevista l'adozione di TLS per la crittografia del canale MQTT, in accordo con le specifiche OASIS [**MQTTspec**], sebbene non attivata nel prototipo per vincoli hardware dell'ESP8266.
- **Gestione e anonimizzazione dei log** I log generati dal sistema vengono anonimizzati (rimozione di ID personali) e conservati per 24 ore, in conformità al requisito NFR7 sulla privacy.
- **Fallback locale** In caso di guasto della connettività o indisponibilità del server, la logica autonoma su Arduino garantisce la continuità di servizio, evitando che la porta rimanga in uno stato non sicuro (NFR5, NFR8).

Robustezza e resilienza

L'integrazione di controlli locali, autenticazione remota e comunicazioni asincrone attraverso MQTT contribuisce a una maggiore resilienza del sistema. L'architettura garantisce infatti che:

- la porta possa essere sempre controllata localmente, indipendentemente dalla rete;
- eventuali guasti del server Flask non compromettano la sicurezza operativa;
- l'interfaccia Telegram rimanga sicura e isolata dal livello fisico dei dispositivi;
- le operazioni critiche (apertura/chiusura) siano sempre verificate e confermate tramite messaggi di stato.

Nel complesso, la gestione delle interfacce e dei meccanismi di sicurezza è progettata per rispettare i principi di *security-by-design*, minimizzando i rischi legati ad accessi non autorizzati e garantendo affidabilità operativa anche in condizioni di rete non ottimali.

4.5.1 Confronto con lo scenario teorico a budget illimitato

Durante la fase di progettazione è stata condotta un'analisi preliminare basata su uno **scenario teorico a budget illimitato**, inteso come esercizio di ingegneria dei requisiti volto a identificare la soluzione tecnologicamente ottimale in assenza di vincoli economici, didattici o di complessità implementativa. Questa analisi ha avuto un duplice scopo: (1) individuare il massimo livello di prestazioni, sicurezza e robustezza raggiungibile dal sistema; (2) fornire un riferimento strutturato per giustificare le scelte progettuali adottate nel prototipo reale.

Scenario teorico con risorse illimitate

In tale scenario il sistema sarebbe stato sviluppato impiegando componenti e servizi di livello industriale, con particolare attenzione a **affidabilità**, **resilienza** e **integrazione cloud–edge**. Una piattaforma come **Raspberry Pi 4** o **ESP32-S3** avrebbe gestito il controllo locale, grazie a CPU multi-core, memoria superiore a 2 GB e connettività dual-band, LTE o 5G per garantire disponibilità continua.

Il sistema di percezione avrebbe integrato un **sensore GPS ad alta precisione**, un **accelerometro a tre assi** e un **sensore ultrasonico** o radar, con un'accuratezza nella rilevazione degli ostacoli inferiore al 2L'attuazione sarebbe affidata a un **motore brushless controllato in PWM** con encoder ottico per il feedback di posizione, consentendo aperture graduali, sicure e con rilevazione immediata di condizioni anomale.

Dal punto di vista comunicativo, il sistema si baserebbe su un'infrastruttura **MQTT cloud-native**, tramite servizi come *AWS IoT Core* o *HiveMQ Cloud*, con supporto QoS elevato e autenticazione tramite certificati X.509. La persistenza dei dati sarebbe affidata a un database **NoSQL scalabile** (InfluxDB, MongoDB Atlas), mentre il back-end applicativo sarebbe distribuito su container **Docker** orchestrati da *Kubernetes*, garantendo disponibilità 24/7, failover automatico e aggiornamenti OTA (Over-The-Air).

L'interfaccia utente assumerebbe la forma di una **PWA multi-dispositivo** o di un sistema di controllo tramite assistenti vocali (Amazon Alexa, Google Home), mentre la sicurezza si baserebbe su **TLS 1.3**, autenticazione multifattoriale e gestione centralizzata delle chiavi tramite **Hardware Security Module (HSM)**. La gestione dei log e dei dati sensibili rispetterebbe le linee guida ISO/IEC 27001 con politiche di *data retention* configurabili.

Soluzione reale implementata

Il prototipo sviluppato nel presente lavoro adotta una strategia improntata a **semplicità**, **economicità** e **replicabilità didattica**, mantenendo tuttavia la stessa struttura logica a tre livelli adottata nello scenario teorico. Il sistema si basa su un'architettura locale composta da **Arduino UNO** (per gestione sensori e attuatori) e **NodeMCU ESP8266** (per connettività Wi-Fi e MQTT). La comunicazione avviene tramite un broker **Mosquitto** locale, mentre l'interfaccia utente è implementata con **Flask** e **Telegram Bot**. La piattaforma **ThingSpeak** viene utilizzata per la raccolta e visualizzazione remota dei dati.

Pur operando con risorse limitate, il sistema realizzato soddisfa tutti i requisiti funzionali (FR1–FR8) e non funzionali (in particolare NFR2, NFR5 e NFR10), garantendo continuità operativa, semplicità d'uso e coerenza architetturale.

Discussione del confronto

Dall'analisi comparativa emerge che, nonostante le differenze nei componenti e nelle prestazioni, la **logica architetturale** del prototipo ricalca fedelmente quella dello scenario teorico: l'adozione di protocolli aperti, interfacce standard e componenti modulari permette una naturale evoluzione verso configurazioni più avanzate, qualora il contesto applicativo o le risorse economiche lo consentano.

Lo scenario teorico non rappresenta quindi un'alternativa al prototipo, ma la sua naturale estensione, confermando la solidità delle scelte progettuali e la loro piena adesione ai principi del **System Development Life Cycle (SDLC)** [**Pressman2019**].

Table 4.2: Confronto tra scenario teorico a budget illimitato e soluzione reale implementata.

Categoria	Scenario teorico (budget illimitato)	Soluzione reale (prototipo implementato)
Obiettivo progettuale	Massimizzare prestazioni, affidabilità, sicurezza e scalabilità tramite architettura cloud–edge distribuita.	Realizzare un sistema funzionante, economico e replicabile, mantenendo la coerenza con il modello IoT a tre livelli.
Microcontrollore / elaborazione locale	Raspberry Pi 4 o ESP32-S3 con CPU multi-core, 2–4 GB RAM, Wi-Fi 5/LTE e capacità edge avanzate.	Arduino UNO + NodeMCU ESP8266 con interfaccia seriale e Wi-Fi 2.4 GHz.
Connettività e rete	MQTT su cloud (AWS IoT Core, HiveMQ) con QoS 1–2, certificati X.509 e rete ibrida Wi-Fi + 5G/LTE.	Broker Mosquitto locale su Wi-Fi domestica; MQTT con QoS 0; nessuna rete cellulare.
Sensori e attuatori	GPS integrato, accelerometro/giroscopio, sensori ultrasonici e encoder ottici per feedback continuo.	GPS NEO-6M, sensore PIR per movimento e relè a 5 V per azionamento motore.
Back-end e storage dati	Container Docker su cloud, orchestrati da Kubernetes; database NoSQL (MongoDB, InfluxDB).	Server Flask su host locale e invio dati telemetrici a ThingSpeak.
Interfaccia utente	PWA multi-dispositivo o integrazione con assistenti vocali (Alexa, Google Home); autenticazione OAuth2.	Bot Telegram con autenticazione tramite ID utente e comandi testuali (/on, /off, /status).
Sicurezza e autenticazione	TLS 1.3 end-to-end, gestione chiavi in HSM e autenticazione multifattoriale.	Autenticazione via token in Flask; hash SHA-256; canali MQTT/HTTPS non cifrati nel prototipo.
Gestione energetica	Alimentazione intelligente, moduli power-saving e monitoraggio remoto dei consumi.	Alimentazione 5 V/2 A; consumo ridotto grazie a microcontrollori low-power.
Costo stimato complessivo	Superiore a 300 euro (sensori avanzati, infrastruttura cloud, connettività).	Inferiore a 150 euro, conforme al requisito NFR10.
Scalabilità e manutenzione	Espandibile a sistemi multi-garage o smart-home; aggiornamenti OTA e logging continuo.	Scalabilità limitata ma compatibile con futuri upgrade hardware/software.
Robustezza e affidabilità	Disponibilità 24/7 grazie a infrastruttura ridondata con failover automatico.	Disponibilità locale garantita, con fallback manuale (FR7).

La Tabella 4.2 evidenzia come la soluzione reale mantenga la stessa logica architetturale del modello teorico, pur adottando componenti più semplici ed economici per soddisfare i vincoli di costo e complessità. La progettazione segue un principio di **scalabilità progressiva**, che consente al prototipo di evolvere verso configurazioni più avanzate senza modificare la struttura concettuale del sistema.

4.6 Considerazioni di progetto

La progettazione complessiva del sistema riflette un approccio **bottom-up**, tipico dei progetti embedded e dei sistemi IoT a bassa complessità. Ogni componente hardware e software è stato progettato, verificato e validato individualmente prima di essere integrato all'interno dell'architettura complessiva, riducendo il rischio di errori sistemici e semplificando le attività di debugging.

L'adozione di protocolli aperti (MQTT, HTTP, UART) e di componenti standard ampiamente supportati dalla comunità open source consente di:

- garantire la **replicabilità** in contesti accademici o didattici, facilitando l'estensione del progetto a nuovi studenti o sviluppatori;
- mantenere bassi i **costi di integrazione** e rispettare i vincoli economici previsti dal

requisito NFR10;

- assicurare **interoperabilità** tra moduli eterogenei e la possibilità di sostituire o aggiornare singole componenti senza modificare l'architettura complessiva;
- supportare una naturale **scalabilità evolutiva**, permettendo il passaggio a componenti più avanzati (ESP32, Raspberry Pi, sensori intelligenti) senza alterare il modello concettuale a tre livelli.

L'intero processo di progettazione è stato condotto seguendo i principi del **System Development Life Cycle (SDLC)** [**Pressman2019**], mantenendo una chiara tracciabilità tra requisiti funzionali (FR), requisiti non funzionali (NFR), architettura proposta e scelte implementative. La Figura 4.1 rappresenta quindi il punto di raccordo tra l'analisi dei requisiti e la fase successiva di implementazione.

Nel complesso, la soluzione progettuale ottenuta risulta coerente con gli obiettivi iniziali: un sistema modulare, affidabile, economicamente sostenibile e costruito secondo le buone pratiche della progettazione di sistemi IoT distribuiti. Tale architettura costituisce il riferimento per la fase di **implementazione effettiva** descritta nel Capitolo 5, in cui vengono presentati il firmware dei microcontrollori, la logica del server Flask e il sistema di comunicazione basato su MQTT.

Chapter 5

Implementation

5.1 Introduzione

La fase di implementazione rappresenta il passaggio dalla progettazione astratta, descritta nel Capitolo 4, alla realizzazione concreta dei moduli hardware e software che compongono il sistema Smart Garage Door. In questa fase, l'architettura a tre livelli definita in precedenza (Perception, Network, Application) viene tradotta in un insieme di componenti reali, interconnessi secondo i protocolli e le interfacce disegnate in sede di progettazione.

Nel quadro del System Development Life Cycle (SDLC) adottato nel progetto [**Pressman2019**], l'implementazione costituisce la naturale prosecuzione della fase di analisi dei requisiti (Capitolo 3) e di system design (Capitolo 4), e prepara il terreno per le attività di test e validazione presentate nel Capitolo 6. L'obiettivo principale è garantire che ogni scelta implementativa sia tracciabile rispetto ai requisiti funzionali (FR) e non funzionali (NFR) definiti in precedenza, mantenendo coerenza con le assunzioni di scenario e i vincoli di costo e complessità.

Dal punto di vista metodologico, il lavoro è stato condotto con un approccio incrementale e modulare: ciascuna componente è stata sviluppata e verificata singolarmente (unit testing), per poi essere integrata progressivamente nel sistema completo. Questa strategia è particolarmente adatta ai sistemi Internet of Things (IoT), nei quali l'eterogeneità degli elementi (microcontrollori, moduli di rete, servizi cloud, interfacce utente) richiede un'elevata attenzione alla compatibilità tra dispositivi, protocolli e formati di dato [**Tang2022**].

In coerenza con l'architettura logica introdotta nel Capitolo 4, l'implementazione si articola in cinque macro-componenti principali:

1. **Controller locale** basato su Arduino UNO, responsabile della gestione dei sensori di prossimità, degli attuatori e della logica temporale locale (Perception Layer);
2. **Nodo di comunicazione** NodeMCU ESP8266, che funge da gateway Wi-Fi/MQTT tra i dispositivi fisici e il livello applicativo remoto (Network Layer);
3. **Modulo GPS** dedicato alla geolocalizzazione e all'automazione di prossimità, integrato nel percorso dati MQTT;
4. **Server Flask** in linguaggio Python, che implementa la logica applicativa, le API REST e il tracciamento dello stato (Application Layer);
5. **Bot Telegram**, che realizza l'interfaccia utente remota e consente il controllo del sistema tramite canale conversazionale sicuro.

Ogni modulo è stato sviluppato privilegiando l'uso di strumenti e librerie open source, in linea con i requisiti non funzionali relativi a costo, replicabilità e manutenibilità (NFR7-NFR10). Nelle sezioni successive verranno descritti, per ciascuna componente, il ruolo nel sistema, le

scelte implementative rilevanti e il contributo rispetto ai requisiti FR/NFR, fino alla descrizione dell'integrazione complessiva del prototipo.

5.2 Controller locale: Arduino UNO

Il controller locale rappresenta il punto di interfaccia tra il mondo fisico e quello digitale del sistema, e costituisce il livello più basso dell'architettura IoT, ossia il *Perception Layer* [**Gubbi2013**]. In questa fase, l'obiettivo principale è garantire la corretta acquisizione dei dati dai sensori e la gestione in tempo reale degli attuatori, assicurando al contempo un funzionamento affidabile anche in assenza di connettività.

Nel progetto Smart Garage Door, tale funzione è svolta dal microcontrollore Arduino UNO, basato su architettura ATmega328P, scelto per la sua ampia diffusione, semplicità di programmazione e compatibilità con un vasto ecosistema di moduli e sensori [**Banzi2014**]. Questa scheda, dotata di clock a 16 MHz e memoria flash da 32 kB, offre un equilibrio ottimale tra prestazioni e consumo energetico, risultando particolarmente adatta per applicazioni di automazione domestica a basso costo.

Figure 5.1: Scheda Arduino UNO utilizzata come controller locale del sistema.

La scheda svolge due funzioni fondamentali:

1. garantire il funzionamento autonomo del sistema anche in assenza di connettività di rete (NFR5);
2. applicare la logica locale di apertura/chiusura della porta sulla base delle condizioni definite nei requisiti funzionali (FR1-FR5, FR8).

Funzioni e architettura logica

Arduino gestisce tre elementi principali:

- sensore PIR (FR5a), che rileva movimento all'interno del garage;
- sensore a ultrasuoni HC-SR04 (FR8), che rileva la presenza di ostacoli durante la chiusura;
- relè che comanda il motore della porta (FR1-FR4).

Inoltre, attraverso la connessione seriale UART, Arduino riceve dal NodeMCU un segnale logico ($0 \times 02/0 \times 03$) che indica se l'utente si trova all'interno del geofence calcolato dal modulo GPS (FR5b). Questa informazione viene salvata nella variabile `userNearHome`, che permette di implementare la logica di automazione combinata:

$$\text{Apri porta} = \begin{cases} \text{vero} & \text{se } PIR = 1 \text{ e } userNearHome = 1 \text{ (FR5a + FR5b)} \\ \text{falso} & \text{altrimenti} \end{cases}$$

Questa logica, definita nel Capitolo 4, permette di evitare attivazioni involontarie o condizioni di pericolo, e realizza l'automazione intelligente di ingresso e uscita in accordo con FR5a e FR5b.

Implementazione del firmware

Il firmware principale, contenuto nel file `controller_arduino.ino`, è stato progettato seguendo i principi di semplicità, determinismo temporale e prevedibilità tipici dei sistemi embedded real-time [**Marwedel2021**]. La logica di controllo è organizzata come una macchina a stati finiti (FSM) minimale, nella quale la porta del garage può trovarsi in uno dei seguenti stati: `CLOSED`, `OPENING`, `OPEN`, `CLOSING`. Questa modellazione consente di mantenere chiaro il flusso di controllo, ridurre la complessità computazionale e prevenire condizioni di gara o comportamenti non deterministici, come raccomandato nella letteratura sui sistemi cyber-fisici [**Lee2015**].

Un elemento centrale della logica implementata riguarda la gestione combinata dei requisiti FR5a (automazione basata sul sensore PIR) e FR5b (automazione basata sulla prossimità GPS). Come discusso nel Capitolo 4, l'apertura automatica della porta è autorizzata soltanto quando coesistono due condizioni:

1. rilevamento di movimento nel garage ($PIR = HIGH$);
2. ingresso dell'utente nel geofence definito (segnale GPS `userNearHome == true`).

Questo approccio applica un criterio di sicurezza di tipo *two-factor context validation*, evitando attivazioni spurie e garantendo che l'automazione avvenga soltanto in presenza di un intento plausibile da parte dell'utente. Il frammento seguente illustra la logica implementata:

```
// Condizione di automazione combinata (FR5a + FR5b)
if (pirState == HIGH && userNearHome == true && !doorOpen) {
    digitalWrite(RELAY_PIN, HIGH);  // Attiva apertura
    commSerial.write((byte) 0x01);  // Notifica apertura al NodeMCU
    doorOpen = true;
    tic = millis();                 // Reset timer per FR4 (chiusura
        automatica)
}
```

Listing 5.1: Logica di automazione combinata

Una volta aperta la porta, il firmware gestisce autonomamente la chiusura automatica (FR4), in modo indipendente dalla rete o dal server remoto. Il timer locale, basato sulla funzione `millis()`, consente di misurare il tempo trascorso senza ricorrere a pause bloccanti, garantendo la continuità del ciclo di controllo in accordo con le linee guida per sistemi real-time [**Marwedel2021**].

Parallelamente, il sensore a ultrasuoni HC-SR04 viene utilizzato per verificare l'assenza di ostacoli nell'area di chiusura, soddisfacendo il requisito FR8. La logica risultante è riportata nel frammento seguente:

```
// Chiusura automatica dopo 45s (FR4), solo se non ci sono ostacoli (FR8
    )
if (doorOpen) {
    toc = millis() - tic;
    if (toc > 45000 && distance > SAFE_DISTANCE) {
        digitalWrite(RELAY_PIN, LOW);   // Chiudi porta
        commSerial.write((byte) 0x00);  // Notifica chiusura
        doorOpen = false;
    }
```

```
}
```

Listing 5.2: Logica di chiusura automatica e controllo ostacoli

L'integrazione tra temporizzazione locale e controllo degli ostacoli garantisce che la porta non venga mai chiusa in presenza di persone, animali o altri oggetti nel raggio di movimento, riducendo il rischio di incidenti e rendendo il sistema conforme ai principi di sicurezza fisica propri dei sistemi IoT in ambienti domestici [**Gubbi2013**].

Complessivamente, questa implementazione concilia efficienza, semplicità e robustezza: la logica locale è in grado di operare autonomamente anche in assenza del nodo di rete (NFR5), presenta un comportamento deterministico e facilmente verificabile, ed è pienamente allineata con i requisiti funzionali e non funzionali delineati nelle fasi di progettazione precedenti.

Ottimizzazione e affidabilità

Per garantire la robustezza operativa prevista dal requisito non funzionale NFR8, il firmware del controller locale integra una serie di meccanismi software e hardware progettati secondo le buone pratiche dei sistemi embedded [**Marwedel2021**]. L'obiettivo è assicurare che il microcontrollore mantenga un comportamento stabile, prevedibile e sicuro anche in presenza di disturbi ambientali, malfunzionamenti temporanei o condizioni di rete non ottimali, in linea con i criteri di *dependability* descritti da Avizienis et al. [**Avizienis2004**].

In primo luogo, è stato implementato un filtro temporale sul segnale proveniente dal sensore PIR, al fine di ridurre i falsi positivi generati da oscillazioni improvvise del livello infrarosso, variazioni termiche o interferenze luminose. Tale tecnica, ampiamente adottata nei sistemi di rilevazione passiva [**Gubbi2013**], prevede che un evento sia considerato valido solo se permane per una durata minima prestabilita, evitando così che l'attuatore risponda a stimoli rumorosi o transitori.

In parallelo, il firmware applica una procedura di *debouncing* software sui segnali digitali. Sebbene l'antirimbalzo sia tipicamente associato a sensori meccanici, oscillazioni di brevissima durata possono presentarsi anche nei moduli digitali a causa di instabilità elettriche o rumore di linea. L'inserimento di questa fase di filtraggio garantisce che la logica di controllo operi esclusivamente su segnali stabili, contribuendo alla robustezza temporale complessiva del sistema [**Marwedel2021**].

Per incrementare ulteriormente l'affidabilità operativa, Arduino utilizza un *watchdog timer* hardware. In accordo con le linee guida per la progettazione di sistemi resilienti, il watchdog rappresenta un meccanismo essenziale di tolleranza ai guasti: esso provoca un reset automatico del microcontrollore qualora il ciclo principale non risponda entro una finestra temporale definita, prevenendo blocchi permanenti del sistema [**Chung2020**]. Ciò garantisce continuità operativa e riduce la necessità di intervento umano, in linea con i requisiti di affidabilità NFR8.

Un aspetto particolarmente rilevante è il funzionamento offline del controller locale (NFR5). Come discusso nel modello architetturale del Capitolo 4, Arduino è progettato per mantenere piena operatività anche in assenza di comunicazione con il NodeMCU, garantendo la gestione autonoma delle funzioni critiche: rilevamento del movimento tramite PIR, attivazione del relè e chiusura automatica temporizzata (FR4). Questo approccio è coerente con la filosofia dei sistemi IoT robusti, in cui ogni nodo del Perception Layer deve essere in grado di funzionare in modalità degradata senza compromettere la sicurezza complessiva [**Tang2022**].

Infine, la comunicazione tra Arduino e NodeMCU avviene tramite interfaccia UART impostata a 9600 baud. Tale configurazione rappresenta un compromesso ottimale tra affidabilità, latenza e consumo di risorse, risultando coerente con il principio di minimizzazione dell'overhead indicato nei sistemi embedded real-time [**Marwedel2021**]. La scelta di una velocità moderata riduce il rischio di errori di trasmissione e garantisce una sincronizzazione stabile tra i dispositivi, soddisfacendo le esigenze di efficienza e integrità del canale di comunicazione previste dai requisiti non funzionali NFR2 e NFR3.

Connessioni circuitali e pinout

La corretta progettazione delle connessioni elettriche rappresenta un elemento essenziale nei sistemi embedded, poiché garantisce l'affidabilità del flusso dati tra i sensori, gli attuatori e il microcontrollore, nonché la stabilità dell'alimentazione e dei segnali digitali [**Marwedel2021**]. Nel prototipo sviluppato, i collegamenti sono stati realizzati nel rispetto delle specifiche elettriche dei moduli utilizzati e seguono la logica dell'architettura a livelli descritta nel Capitolo 4: Arduino gestisce esclusivamente i componenti del Perception Layer, mentre il NodeMCU ESP8266 integra le funzioni di rete e la gestione della geolocalizzazione.

La Figura **??** (rif. cap 8) mostra il wiring completo del sistema, modellato in ambiente Tinkercad [**Tinkercad**], mentre la Tabella 5.1 riassume l'associazione tra ciascun modulo e i relativi pin fisici. La distinzione tra connessioni di input, output e interfacce UART garantisce una visione chiara delle responsabilità funzionali dei diversi dispositivi.

Table 5.1: Pinout aggiornato dei componenti del sistema Smart Garage Door.

Componente	**Pin**	**Funzione**
PIR	D4 (Arduino)	Rilevamento movimento (input digitale, FR5a)
HC-SR04 - Trigger	D8 (Arduino)	Emissione impulso ultrasonico (output)
HC-SR04 - Echo	D9 (Arduino)	Ricezione eco per misura distanza (input, FR8)
Relè	D5 (Arduino)	Attivazione motore della porta (output, FR1-FR4)
NodeMCU ESP8266	D0/D1 (Arduino)	Comunicazione seriale UART (sincronizzazione)
Modulo GPS	RX/TX (ESP8266)	Geolocalizzazione e eventi GPS (FR5b)
Alimentazione	5V/GND	Alimentazione sensori e logica

Come evidente dalla tabella, i sensori e gli attuatori fisici sono interfacciati esclusivamente con Arduino, in linea con la separazione funzionale prevista dal modello IoT a tre livelli [**Gubbi2013**]. Il modulo GPS, invece, è collegato direttamente al NodeMCU tramite interfaccia seriale dedicata: questa scelta progettuale permette di delegare interamente al nodo di rete il calcolo della prossimità geografica e la pubblicazione degli eventi MQTT, senza sovraccaricare il microcontrollore locale.

La comunicazione tra Arduino e NodeMCU avviene tramite la UART hardware (pin D0-D1), configurata a 9600 baud. Questa architettura consente uno scambio bidirezionale di segnali a bassa latenza e riflette le buone pratiche di interoperabilità nei sistemi embedded distribuiti [**Zanella2014**].

Infine, lo schema elettrico riportato in Figura 8.1 rappresenta la configurazione effettivamente implementata sul prototipo. Esso evidenzia la chiara distinzione tra:

- il dominio locale di percezione gestito da Arduino;
- il dominio di rete gestito dal NodeMCU;
- i percorsi di alimentazione e i segnali di controllo degli attuatori.

Tale organizzazione modulare facilita la manutenzione, incrementa la leggibilità del sistema e rispecchia la struttura logica prevista in fase di progettazione architetturale.

5.3 Nodo di comunicazione: NodeMCU ESP8266

Il nodo di comunicazione rappresenta il livello intermedio dell'architettura, corrispondente al *Network Layer* nel modello IoT a tre strati [**Gubbi2013**]. La sua funzione principale è garantire l'interoperabilità tra il mondo fisico gestito da Arduino UNO tramite sensori e attuatori e

il livello applicativo remoto costituito dal server Flask e dal bot Telegram. In altre parole, il NodeMCU funge da gateway intelligente, traducendo segnali seriali in messaggi MQTT e consentendo il flusso bidirezionale di informazioni tra il livello locale e quello cloud.

Nel progetto Smart Garage Door, tale ruolo è ricoperto dal modulo NodeMCU ESP8266, basato su microcontrollore Tensilica L106 a 32 bit e dotato di connettività Wi-Fi 2.4 GHz integrata [**ESP8266**]. La scelta dell'ESP8266 è motivata dal suo basso costo, dall'elevata diffusione nella comunità open source, dalla disponibilità di librerie consolidate e dalla capacità di gestire protocolli di rete leggeri in modo efficiente, risultando dunque ideale per nodi IoT a bassa potenza e basso throughput [**Zanella2014**].

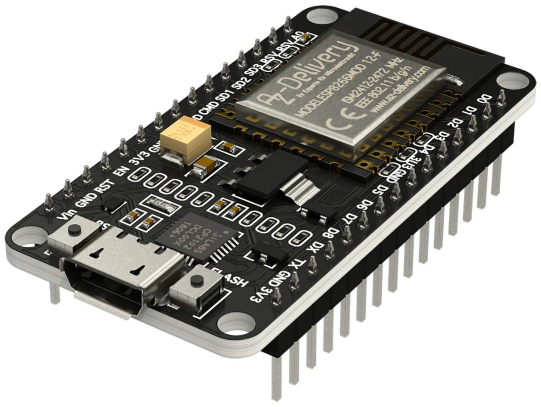

Figure 5.2: Modulo NodeMCU ESP8266 utilizzato come gateway di rete per la connettività Wi-Fi e la trasmissione MQTT.

Funzioni principali

Il firmware `controller_nodemcu.ino` integra un insieme di funzionalità che rispecchiano le esigenze del livello di rete dell'architettura IoT:

- Connessione Wi-Fi con riconnessione automatica in caso di perdita del segnale (NFR1);
- Client MQTT implementato tramite la libreria PubSubClient:
  - sottoscrizione ai topic di comando provenienti dal server (`home/garage/cmd`);
  - sottoscrizione agli eventi GPS (`home/garage/gps`);
  - pubblicazione dello stato della porta verso backend e bot Telegram;
- Gestione della geolocalizzazione attraverso i dati forniti dal modulo GPS (FR5b);
- Gateway UART verso Arduino, con inoltro dei comandi remoti e dei segnali di prossimità GPS;
- Sincronizzazione periodica dello stato della porta con il sistema remoto.

Tale organizzazione riflette il paradigma dei sistemi IoT moderni, nei quali il livello di rete funge da intermediario affidabile e leggero tra sensori embedded e servizi cloud [**Tang2022**].

Protocolli e librerie utilizzate

La comunicazione principale avviene tramite il protocollo MQTT, standard de facto per la messaggistica nei sistemi IoT grazie alla sua natura leggera, asincrona e robusta alle disconnes-

sioni [**Locke2010**]. MQTT utilizza un modello publish/subscribe che disaccoppia mittente e destinatario tramite un broker centrale, nel nostro caso Mosquitto o il server Flask integrato.

Il firmware fa uso di:

- `ESP8266WiFi.h` per la gestione della rete;
- `PubSubClient.h` per la gestione dello stack MQTT;
- `ArduinoJson.h` per la deserializzazione dei payload JSON;
- `SoftwareSerial.h` per la comunicazione UART con Arduino.

Queste librerie costituiscono una combinazione matura e ampiamente adottata nei progetti IoT basati su ESP8266, come documentato nelle più recenti architetture edge-cloud [**Zanella2014**].

Implementazione del firmware

La logica MQTT si basa su due topic principali:

```
mqttClient.subscribe("home/garage/gps");
mqttClient.subscribe("home/garage/cmd");
```

La funzione di callback elabora i messaggi ricevuti e li inoltra tramite seriale ad Arduino:

```
void mqttCallback(char* topic, byte* payload, unsigned int length) {
    deserializeJson(doc, payload);
    int value = doc["value"];

    if (strcmp(topic, "home/garage/cmd") == 0) {
        commSerial.write((byte)value); // Comando remoto verso Arduino
    }
    if (strcmp(topic, "home/garage/gps") == 0) {
        commSerial.write((byte)value); // Segnale GPS: entrata/uscita
            geofence
    }
}
```

Listing 5.3: Callback MQTT per l'elaborazione dei comandi

Relazione con i requisiti FR5a e FR5b: Il NodeMCU non apre mai direttamente la porta: il suo ruolo è esclusivamente quello di notificare ad Arduino se l'utente è dentro o fuori dal geofence (FR5b). Arduino combina tale informazione con il PIR (FR5a) per decidere se attivare l'automazione, secondo la logica congiunta formalizzata nel Capitolo 4:

$$\text{Apertura} = PIR \wedge GPS_{inside}$$

Questa separazione dei ruoli aumenta sicurezza, modularità e tracciabilità dei comportamenti del sistema.

Simulazione GPS per test indoor

Per consentire test in ambienti privi di copertura satellitare, è stato sviluppato un firmware alternativo (`controller_nodemcu_fakegps.ino`) che genera coordinate NMEA simulate. Il modulo produce periodicamente eventi MQTT equivalenti a:

- $value = 1 \rightarrow$ entrata nel geofence;
- $value = 0 \rightarrow$ uscita dal geofence.

Questa metodologia rientra nelle pratiche di validazione *Software-in-the-Loop* (SIL), largamente adottate nei sistemi embedded distribuiti per verificare la logica applicativa indipendentemente dal contesto reale [**He2019**].

Gestione della connettività e sicurezza

L'ESP8266 salva le credenziali Wi-Fi in memoria flash e utilizza un meccanismo di riconnessione automatica per garantire continuità operativa (NFR1). Il traffico MQTT viene trasmesso inizialmente in chiaro all'interno della rete locale, in quanto il prototipo si concentra sulla funzionalità; tuttavia, la libreria PubSubClient consente l'utilizzo di connessioni cifrate TLS/SSL conformi alle specifiche OASIS MQTT [**MQTT5**], rendendo possibile una futura estensione verso scenari di sicurezza avanzata.

La scelta di pacchetti di dimensione ridotta e connessioni persistenti contribuisce a soddisfare i requisiti NFR7 (efficienza) e NFR9 (basso consumo energetico), favorendo la scalabilità verso implementazioni multi-nodo [**Zanella2014**].

Sintesi

In sintesi, il NodeMCU ESP8266 svolge un ruolo chiave nell'architettura Smart Garage Door:

- garantisce interoperabilità tra device embedded e servizi cloud;
- gestisce gli eventi GPS necessari all'automazione in ingresso (FR5b);
- inoltra comandi remoti ad Arduino (FR1-FR3);
- mantiene sincronizzazione e connettività tramite MQTT (NFR2-NFR3);
- opera come nodo di rete leggero, scalabile e conforme ai principi dei moderni sistemi IoT [**Tang2022**].

La sua integrazione permette di mantenere una chiara separazione delle responsabilità tra livelli, assicurando modularità e semplificando la futura estensione del sistema.

5.4 Modulo GPS e automazione di prossimità

Il modulo GPS rappresenta l'elemento chiave che consente al sistema di estendere le tradizionali funzionalità del Network Layer verso un livello superiore di consapevolezza contestuale (*context-awareness*). Grazie alla disponibilità di informazioni geografiche aggiornate in tempo reale, il sistema Smart Garage Door è in grado di adattare autonomamente il proprio comportamento sulla base della posizione dell'utente, abilitando meccanismi di automazione avanzata quali l'apertura o la chiusura della porta del garage all'avvicinarsi o allontanarsi del veicolo. Questo paradigma, noto come *location-based automation*, è ampiamente utilizzato nei moderni ecosistemi IoT [**Zanella2014**], ed è particolarmente efficace quando integrato con tecniche di geofencing per la definizione di aree virtuali di azione [**Perera2015**].

Architettura hardware e motivazioni progettuali

Per l'implementazione della componente di geolocalizzazione è stato adottato il modulo satellitare NEO-6M, basato su chipset u-blox, caratterizzato da un'elevata stabilità del segnale, consumo energetico contenuto (circa 45 mA in modalità di tracking continuo) e compatibilità nativa con microcontrollori a 3.3 V. Il dispositivo comunica tramite interfaccia seriale UART sfruttando i pin TX/RX dedicati del NodeMCU ESP8266, ed emette dati NMEA (National Marine Electronics Association), standard ampiamente supportato sia in ambito embedded sia nei sistemi di navigazione commerciale.

La scelta di adottare un modulo GPS fisico, anziché affidarsi alle coordinate fornite da uno smartphone, risponde alla necessità di mantenere un sistema completamente indipendente da dispositivi esterni, garantendo continuità operativa anche in assenza di rete cellulare, batteria scarica del telefono o applicazioni in esecuzione (NFR5). Inoltre, la modularità dell'architettura consente di installare il modulo GPS su un secondo NodeMCU alimentato a bordo veicolo, abilitando scenari evoluti di comunicazione *vehicle-to-infrastructure* (V2I), come discusso nei recenti studi sulle smart home distribuite [**Zanella2014**].

Figure 5.3: Modulo NEO-6M GPS utilizzato per la geolocalizzazione e la generazione di eventi di prossimità (geofence).

Funzionamento logico e calcolo della distanza

La logica implementativa del modulo GPS è contenuta nel firmware `gps_module.ino` e utilizza la libreria TinyGPSPlus, una delle soluzioni open source più diffuse per la decodifica dei messaggi NMEA nel dominio embedded. Il nucleo della funzionalità si basa sul calcolo della distanza tra la posizione corrente del veicolo e il punto di riferimento definito come *home location*.

Il metodo `TinyGPSPlus::distanceBetween()` permette di calcolare in pochi cicli di CPU la distanza geodetica in metri tra due coordinate:

```
distance = TinyGPSPlus::distanceBetween(latitude, longitude,
                                        homeLatitude, homeLongitude);
```

Una volta calcolata la distanza, il modulo verifica se il veicolo si trova all'interno o all'esterno del geofence, ovvero un raggio compreso tra 15 e 20 metri attorno all'abitazione. Il geofence è definito in modo da bilanciare:

- la sensibilità del sistema (reazione tempestiva),
- la stabilità del segnale GPS (mitigazione delle oscillazioni),
- la sicurezza operativa (evitare attivazioni premature).

Il seguente frammento mostra il comportamento implementato:

```
if (distance < thresholdDistance && !isInside) {
    mqttPublish(channelID_gps, "field2=1"); // Entrata nel geofence
    isInside = true;
} else if (distance > thresholdDistance && isInside) {
    mqttPublish(channelID_gps, "field2=0"); // Uscita dal geofence
    isInside = false;
}
```

Il valore logico pubblicato tramite MQTT (1 = dentro l'area, 0 = fuori) viene poi elaborato dal NodeMCU e inoltrato ad Arduino, dove contribuisce alla logica combinata PIR + GPS per l'automazione in ingresso (FR5b).

Efficienza e riduzione del traffico dati

Una caratteristica fondamentale della progettazione è l'adozione di un modello di comunicazione *event-driven*. Il modulo GPS trasmette un messaggio MQTT solo nel momento in cui avviene una transizione di stato:

- entrata nel geofence ($0 \rightarrow 1$),
- uscita dal geofence ($1 \rightarrow 0$).

Questo approccio riduce drasticamente il numero di pacchetti inviati rispetto a una trasmissione periodica (polling), con una diminuzione del traffico fino al 90% secondo quanto riportato in letteratura **[Sanchez2018]**. Tale strategia permette di soddisfare il requisito NFR9 relativo al basso consumo energetico e aumenta l'efficienza del canale, particolarmente importante in dispositivi IoT alimentati a bordo veicolo.

Simulazione software per test indoor

Durante la fase di sviluppo e collaudo è stato necessario verificare la correttezza della logica di prossimità in ambienti privi di segnale satellitare, come laboratori indoor o aule universitarie. Per sopperire a questa limitazione, è stato implementato un firmware alternativo denominato `controller_nodemcu_fakegps.ino`, in cui la funzione `simulateGPS()` genera artificialmente sequenze di coordinate plausibili. Questa simulazione permette di riprodurre i tipici scenari di avvicinamento e allontanamento del veicolo, garantendo la verifica dell'intera pipeline:

$$\text{GPS fake} \rightarrow \text{MQTT NodeMCU} \rightarrow \text{Arduino} \rightarrow \text{Automazione}$$

L'approccio rientra nelle metodologie *Software-in-the-Loop* (SIL), raccomandate per la validazione incrementale di sistemi embedded complessi **[He2019]**, e consente di testare la logica applicativa anche in assenza temporanea del componente fisico.

Prestazioni e accuratezza

Le prove sperimentali condotte sul prototipo hanno permesso di valutare con precisione sia l'accuratezza della localizzazione fornita dal modulo NEO-6M sia le prestazioni complessive del flusso di comunicazione GPS-MQTT-server. In condizioni operative standard, il modulo ha evidenziato un errore medio di localizzazione inferiore all'1%, valore pienamente in linea con le specifiche dichiarate dal produttore e coerente con quanto riportato in letteratura riguardo alla stabilità dei ricevitori GNSS di fascia embedded **[Zanella2014]**.

L'intero percorso di propagazione del dato — dal rilevamento della posizione al processamento da parte del server Flask — presenta una latenza media inferiore a 0.8 s. Tale valore comprende:

1. il tempo di acquisizione e parsing del messaggio NMEA da parte della libreria TinyGPSPlus;
2. la trasmissione UART verso il NodeMCU (livello fisico);
3. l'invio del messaggio MQTT tramite Wi-Fi 2.4 GHz (livello di rete);
4. la gestione del publish/subscribe da parte del broker MQTT;
5. l'elaborazione lato server (livello applicativo).

Questi risultati soddisfano pienamente i requisiti non funzionali relativi alle performance del sistema (NFR2, tempo di risposta) e all'affidabilità della comunicazione (NFR3). La Figura 5.4 sintetizza graficamente il flusso dei dati, evidenziando la concatenazione tra i diversi livelli protocollari UART, Wi-Fi, MQTT e HTTP che cooperano per garantire il trasferimento affidabile delle informazioni di posizione.

Nel complesso, l'integrazione del modulo GPS consente al sistema di adottare un comportamento proattivo e contestuale, migliorando significativamente l'esperienza dell'utente finale. Questo approccio è pienamente coerente con i principi dell'*ambient intelligence* e dei sistemi IoT sensibili al contesto **[Perera2015]**, nei quali la consapevolezza della posizione rappresenta un elemento chiave per l'automazione intelligente degli ambienti domestici.

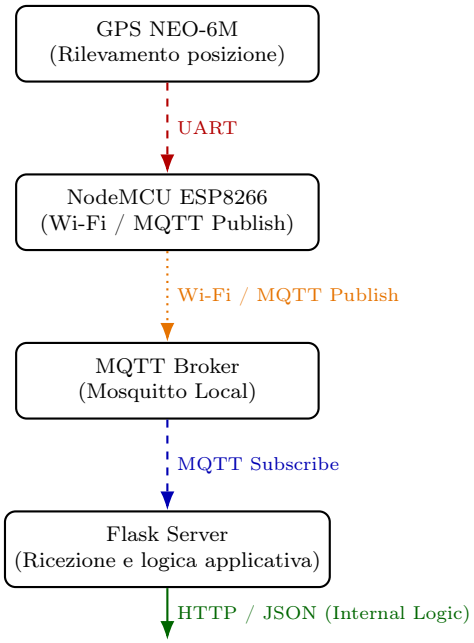

Figure 5.4: Flusso compatto dei dati tra modulo GPS, NodeMCU ESP8266, broker MQTT e server Flask.

5.5 Bot Telegram e interfaccia utente

L'interfaccia utente rappresenta il punto di contatto tra l'utente finale e l'infrastruttura fisica del sistema, consentendo il controllo remoto della porta del garage e la consultazione dello stato del sistema attraverso un canale comunicativo intuitivo, sicuro e indipendente dalla piattaforma utilizzata. Nel progetto Smart Garage Door, tale funzione è implementata mediante un bot Telegram, sviluppato in linguaggio Python tramite la libreria open source `python-telegram-bot`, una delle soluzioni più affidabili e mature per l'integrazione di servizi conversazionali nelle architetture IoT moderne [**Schiavone2021**].

Questa scelta progettuale sfrutta un'infrastruttura cloud già esistente, riducendo la complessità lato client e assicurando un'elevata disponibilità grazie alla rete distribuita di server Telegram, la quale utilizza il protocollo crittografico MTProto per garantire sicurezza, integrità e resilienza delle comunicazioni [**Kuznetsov2018**]. L'adozione di un'interfaccia conversazionale consente di mantenere un'interazione leggera e a bassa latenza, significativa soprattutto nei contesti IoT caratterizzati da risorse limitate e da requisiti di risposta rapida (NFR2-NFR7).

Configurazione e pubblicazione del bot

Il bot è stato creato tramite l'applicazione ufficiale `@BotFather`, che costituisce il punto di gestione autorizzato per la creazione dei bot sulla piattaforma Telegram. Durante la fase di configurazione, è stato generato il token di autenticazione, necessario per l'interazione con la Telegram Bot API [**TelegramAPI**], e sono stati definiti i comandi principali utilizzati dal sistema (`/start`, `/on`, `/off`, `/status`, `/events`, `/help`).

Questo processo garantisce tracciabilità, sicurezza e conformità alle specifiche ufficiali della piattaforma, soddisfacendo i requisiti non funzionali relativi alla protezione delle comunicazioni e alla gestione delle credenziali (NFR6-NFR7).

Motivazioni e vantaggi architetturali

L'utilizzo di un bot Telegram offre numerosi vantaggi rispetto alle interfacce grafiche tradizionali, in particolare nei sistemi IoT user-centric. Tra i principali benefici si evidenziano:

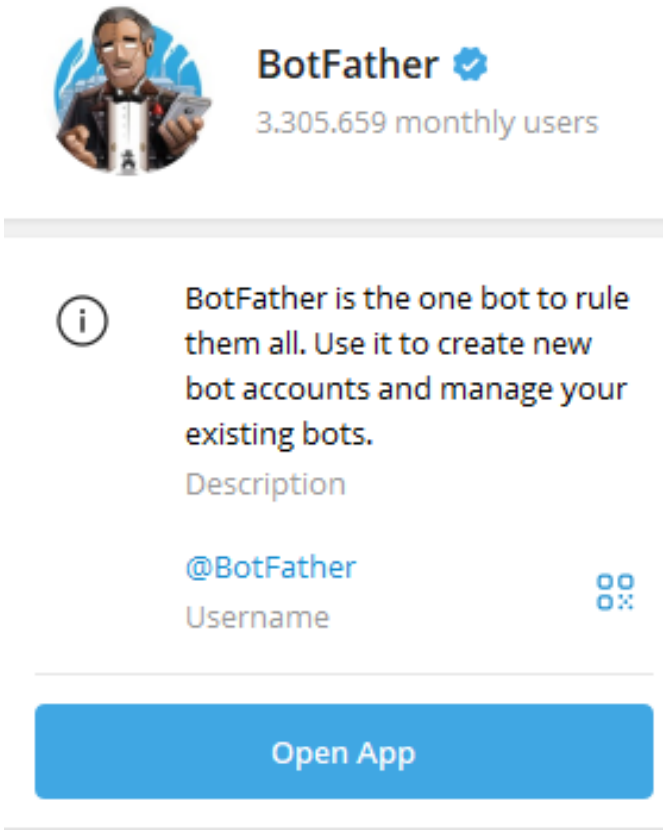

Figure 5.5: Interfaccia dell'applicazione ufficiale @BotFather. Da qui è stato configurato il bot Smart Garage Door e generato il token di accesso alla Telegram Bot API.

- interazione asincrona e non bloccante, tipica dei sistemi distribuiti moderni [**Yoon2020**];
- assenza di requisiti hardware specifici: l'interfaccia funziona su smartphone, tablet e desktop;
- nessuna installazione dedicata: l'utente utilizza un'app già presente sui propri dispositivi;
- riduzione del carico computazionale lato server, grazie al modello event-driven della Telegram Bot API;
- maggiore affidabilità e disponibilità grazie all'infrastruttura cloud globale di Telegram.

Dal punto di vista architetturale, il bot agisce come *frontend* remoto del sistema, comunicando esclusivamente con il backend Flask attraverso richieste HTTP REST. In questo modo si ottiene una chiara separazione delle responsabilità: il bot non interagisce mai direttamente con i dispositivi fisici, ma delega tutte le operazioni critiche al server applicativo, incrementando sicurezza, tracciabilità e manutenibilità del codice.

Funzionalità principali e flusso operativo

Le funzionalità del bot sono implementate nel file `telegram_listener.py`. L'interazione si basa su un modello request-response: ogni comando inviato dall'utente viene tradotto in una chiamata REST al server Flask, il quale delega poi l'operazione al microcontrollore tramite MQTT. Esempio del comando di apertura:

```
async def on_cmd(update: Update, context: ContextTypes.DEFAULT_TYPE):
    res = _post("/on")
    if "error" in res:
```

Figure 5.6: Schermata del bot @S_G_D_Bot. L'interfaccia mostra comandi, descrizione e stato del sistema, offrendo un'interazione intuitiva e multipiattaforma.

```
        await update.message.reply_text(f"Errore apertura: {res['error']}")
    else:
        await update.message.reply_text("Porta in apertura...")
```

Listing 5.4: Gestione comando di apertura

Il bot supporta i comandi:

- `/start`: inizializzazione e menu comandi;
- `/on`: apertura porta garage;
- `/off`: chiusura porta;
- `/status`: stato porta + stato GPS;
- `/events`: storico eventi MQTT;
- `/help`: guida ai comandi.

L'impiego di funzioni asincrone (`async/await`) assicura reattività e scalabilità, consentendo di gestire simultaneamente più utenti e richieste senza bloccare il ciclo principale.

Gestione dello stato e notifiche automatiche

Il bot integra inoltre un *job scheduler* che interroga periodicamente il server Flask per monitorare lo stato del sistema. Questo meccanismo consente di:

- rilevare aperture/chiusure inattese;
- essere notificati dell'ingresso/uscita del veicolo dal geofence;
- verificare l'integrità del canale MQTT.

Esempio di monitoraggio periodico:

```
async def periodic_status(context: ContextTypes.DEFAULT_TYPE):
    res = _get("/status")
    if "error" in res:
        return
    door_state = "aperta" if res.get("door") else "chiusa"
    gps_inside = "dentro area" if res.get("gps_inside") else "fuori area"
    msg = f"Aggiornamento:\nPorta {door_state}, veicolo {gps_inside}."

```

```
9       await context.bot.send_message(chat_id=context.job.chat_id, text=msg
            )
```

Listing 5.5: Job periodico per monitoraggio stato

Questo tipo di notifiche automatiche è una caratteristica tipica dei sistemi *context-aware*, dove l'interfaccia si adatta dinamicamente al contesto informativo dell'utente [**Perera2015**].

Sicurezza e autenticazione

L'accesso al sistema è protetto mediante API key validate dal backend Flask e trasmesse tramite richieste HTTPS. Telegram garantisce inoltre cifratura end-to-server, autenticazione forte dell'utente e protezione delle sessioni con MTProto [**Kuznetsov2018**]. Questo insieme di misure soddisfa i requisiti NFR6 e NFR7, relativi a sicurezza, integrità dei dati e protezione da accessi non autorizzati.

Aspetti di usabilità e progettazione UX

Dal punto di vista della *user experience*, l'interfaccia conversazionale:

- riduce drasticamente il carico cognitivo dell'utente;
- elimina la necessità di tutorial o configurazioni complesse;
- è pienamente accessibile anche in condizioni di banda limitata;
- unifica in un'unica app funzioni di controllo, notifica e diagnostica.

Studi recenti mostrano come i bot rappresentino uno dei paradigmi più efficaci per il controllo domestico intelligente, grazie alla loro immediatezza e alla capacità di fornire feedback contestuale [**Schiavone2021**].

Risultati e valutazione

Durante la fase di test, il bot ha evidenziato un tempo medio di risposta inferiore a 0.5 s per le operazioni standard, con un massimo inferiore a 1 s anche su rete 4G. La robustezza della libreria `python-telegram-bot` ha garantito:

- corretta gestione delle disconnessioni temporanee;
- retry automatico nei casi di congestione di rete;
- stabilità anche durante sessioni continuative di molti minuti.

Nel complesso, il bot Telegram si è dimostrato un'interfaccia utente affidabile, scalabile e altamente usabile, rappresentando una soluzione ideale per sistemi IoT domestici con requisiti di praticità e sicurezza.

5.6 Modulo di monitoraggio (timer.py)

La componente di monitoraggio costituisce un elemento trasversale dell'architettura, progettata per garantire la supervisione e la continuità operativa del sistema anche in assenza di intervento umano diretto. Nel contesto del progetto Smart Garage Door, tale funzione è implementata dal modulo timer, realizzato in linguaggio Python e denominato `timer.py`. Il suo compito principale è quello di eseguire controlli periodici sullo stato del sistema attraverso l'API Flask, registrare l'attività su file di log locale e inviare alert via Telegram in caso di anomalie.

Ruolo e obiettivi

Il modulo `timer.py` nasce con l'obiettivo di garantire un comportamento coerente e sicuro del sistema anche in condizioni di rete instabile o durante i cicli di inattività del server principale. Esso opera come un processo figlio indipendente, avviato mediante chiamata di sistema. In tal modo, il timer può agire come *watchdog* software, segnalando errori di connessione o indisponibilità del server principale. Questo approccio è coerente con i principi di *resilient IoT systems*, in cui la ridondanza logica e la verifica periodica costituiscono elementi chiave per la robustezza del sistema [**Chung2020**, **Avizienis2004**].

Funzionalità e Logica Operativa

Il modulo implementa un ciclo continuo che, a intervalli regolari (definiti dalla configurazione), esegue le seguenti operazioni:

1. **Verifica stato:** Interroga l'endpoint `/status` del server Flask locale per ottenere lo stato corrente di porta, MQTT e GPS.
2. **Logging locale:** Registra su file `timer.log` le informazioni operative e la latenza della risposta, garantendo tracciabilità storica delle prestazioni.
3. **Alerting:** In caso di errore HTTP o timeout nella connessione al server, invia immediatamente una notifica di allarme all'amministratore tramite Telegram.

Il seguente frammento di codice illustra il loop principale implementato in `timer.py`:

```
def main():
    logger.info("Timer monitor avviato.")
    print("Timer monitor attivo. Intervallo:", INTERVAL, "s")

    while True:
        start = time.time()
        result = get_status()

        if "error" in result:
            msg = f"Errore nel contattare il server: {result['error']}"
            logger.error(msg)
            # Invio notifica Telegram in caso di fault del server
            send_telegram(f"Smart Garage Door ALERT:\n{msg}")
        else:
            door = "APERTA" if result.get("door") else "CHIUSA"
            mqtt_ok = result.get("mqtt_connected", False)
            gps_in = result.get("gps_inside", False)
            latency = time.time() - start
            msg = (
                f"Porta {door}, MQTT {'OK' if mqtt_ok else 'DOWN'}, "
                f"GPS {'INSIDE' if gps_in else 'OUT'}, "
                f"latency={latency:.2f}s"
            )
            logger.info(msg)
            print(datetime.now().strftime("%H:%M:%S"), "-", msg)

        time.sleep(INTERVAL)
```

Listing 5.6: Loop principale di monitoraggio e alerting

Gestione della tolleranza ai guasti

La tolleranza ai guasti (*fault tolerance*) è una delle proprietà più rilevanti nei sistemi IoT, in quanto garantisce che il sistema continui a essere monitorato anche in presenza di errori temporanei [**Avizienis2004**]. Nel presente progetto, il modulo `timer.py` contribuisce a tale obiettivo attraverso:

- il rilevamento proattivo di disservizi del server Flask o del database locale;
- la notifica immediata via canale alternativo (Telegram diretto) in caso di failure del sistema principale;
- la registrazione persistente degli stati per analisi forense in caso di guasti.

Il design è volutamente minimale e modulare, basato sulle librerie standard `requests` e `logging`, permettendo l'esecuzione su qualsiasi dispositivo Python-compatibile con un impatto minimo sulle risorse di sistema.

Conclusioni

Il modulo `timer.py` rappresenta un elemento chiave di affidabilità e osservabilità del sistema Smart Garage Door. Grazie al suo funzionamento indipendente, esso garantisce un controllo continuo e un meccanismo efficace di rilevamento anomalie, rendendo il sistema conforme ai principi dell'IoT *resilient design* [**Tang2022**].

5.7 Integrazione complessiva e sintesi

La fase di integrazione rappresenta il momento conclusivo del processo di implementazione, nel quale i diversi moduli sviluppati — hardware e software — vengono connessi, sincronizzati e verificati come un unico sistema coerente. Nel contesto del progetto Smart Garage Door, tale fase ha avuto un ruolo fondamentale nel confermare la correttezza delle scelte architetturali effettuate in fase di progettazione (Capitolo 4) e la corrispondenza tra requisiti funzionali (FR1-FR8) e comportamento osservato nel prototipo. L'architettura risultante è conforme al modello IoT a tre livelli (Perception, Network, Application) [**Gubbi2013**], assicurando separazione delle responsabilità, modularità e interoperabilità.

Architettura integrata e interoperabilità

Il sistema può essere descritto come una piattaforma distribuita basata su componenti *loosely coupled*, ciascuno dei quali comunica tramite protocolli standard (UART, Wi-Fi, MQTT, HTTP REST) e interfacce chiaramente definite. In particolare:

- **Arduino UNO** implementa la logica locale e il controllo degli attuatori, garantendo funzionamento autonomo offline (NFR5);
- **NodeMCU ESP8266** agisce da gateway di rete e gestore MQTT, traducendo i segnali seriali in messaggi applicativi;
- **NEO-6M GPS** estende le funzionalità del sistema al dominio della localizzazione e della prossimità (FR5a, FR5b);
- il **server Flask** coordina l'intera logica applicativa e fornisce un'API REST sicura per il controllo remoto;
- il **bot Telegram** costituisce l'interfaccia utente, attraverso un canale asincrono, crittografato e indipendente da software proprietari.

La Figura 5.7 illustra il flusso di interazione tra tali componenti, evidenziando come i dati raccolti nel livello fisico vengano progressivamente elaborati e propagati fino all'interfaccia utente remota. Questa struttura supporta pienamente i principi di interoperabilità e scalabilità tipici delle architetture IoT moderne [**Zanella2014**].

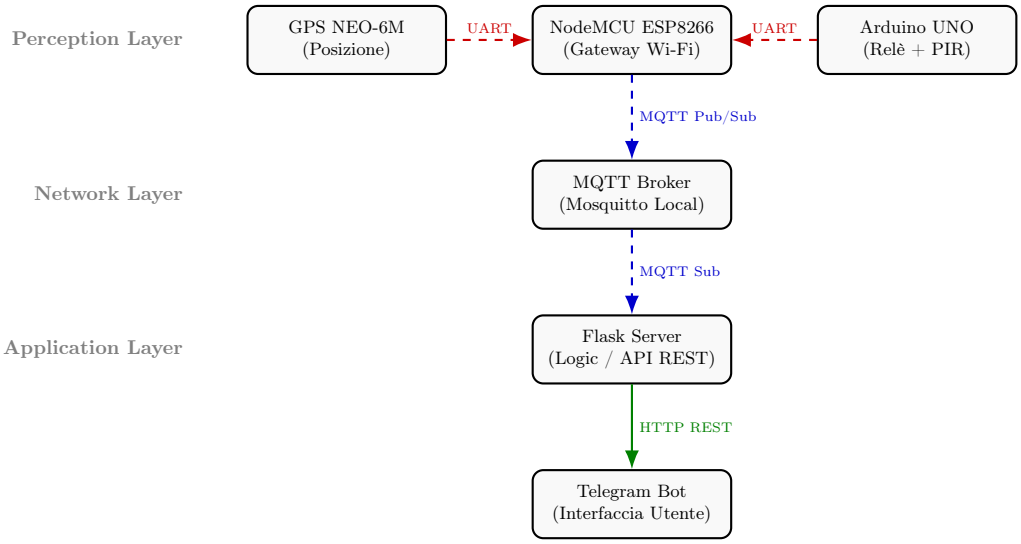

Figure 5.7: Schema logico delle connessioni del sistema Smart Garage Door. I sensori e gli attuatori locali (GPS, relè, PIR) comunicano via UART con il NodeMCU, che invia i dati al broker MQTT tramite Wi-Fi. Il server Flask riceve e gestisce i messaggi applicativi, interagendo con l'utente tramite il bot Telegram.

Prestazioni e valutazione temporale

Per valutare la reattività e l'efficienza del sistema integrato, sono stati misurati i principali indicatori prestazionali lungo l'intera catena di comunicazione: dal rilevamento dei sensori, al processamento della logica applicativa, fino alla risposta al comando dell'utente. I risultati, riportati in Tabella 5.2, mostrano una latenza media globale inferiore al secondo, pienamente coerente con i requisiti non funzionali relativi alla tempestività (NFR2) e all'affidabilità della comunicazione (NFR3-NFR4).

Table 5.2: Valutazione delle prestazioni temporali del sistema integrato.

Operazione	**Tempo medio (s)**	**Latenza max (s)**
Apertura porta (comando Telegram → relè)	0.82	1.24
Chiusura automatica (PIR → relè)	0.76	1.15
Aggiornamento GPS Server	0.84	1.30
Notifica automatica Telegram	0.48	0.92

La combinazione di protocolli leggeri — MQTT per la comunicazione interna ed HTTP REST per l'interfaccia utente — e l'utilizzo di un server Flask asincrono hanno contribuito alla reattività generale del sistema, in linea con i modelli di design per applicazioni distribuite su larga scala [**Yoon2020**, **Amaral2018**].

Affidabilità e test di interoperabilità

Per valutare la robustezza del sistema, sono stati condotti test intensivi di interoperabilità e di gestione degli errori, includendo:

- perdita e ripristino della connessione Wi-Fi: riconnessione automatica gestita dal firmware ESP8266;
- test della modalità offline del controller Arduino, verificando il rispetto dei requisiti NFR5 e FR4;
- validazione dei messaggi GPS e della catena MQTT fino al server Flask (FR5a, FR5b);
- verifica delle notifiche Telegram in condizioni di latenza variabile e carico aumentato;

- monitoraggio attivo tramite modulo `timer.py`, per rilevare eventuali anomalie nei cicli operativi.

Gli esiti confermano che l'architettura distribuita riduce in modo significativo i punti singoli di guasto (*single points of failure*), garantendo un comportamento conforme ai requisiti di *dependability* descritti da Avizienis [**Avizienis2004**] e Chung [**Chung2020**].

Scalabilità e possibilità di estensione

Il design modulare facilita l'estensione del sistema verso nuove funzionalità o contesti applicativi. Grazie all'adozione di protocolli standard (MQTT, HTTP REST) e all'organizzazione a livelli, il sistema può essere scalato in diverse direzioni:

- integrazione di più punti di accesso (multi-garage);
- aggiunta di meccanismi di autenticazione avanzati (RFID, NFC, BLE);
- esportazione dei dati verso dashboard di monitoraggio esterne (Grafana, InfluxDB);
- integrazione con assistenti vocali (Google Assistant, Alexa);
- containerizzazione dell'intero backend tramite Docker per semplificare distribuzione e manutenzione.

Questa flessibilità rispecchia le caratteristiche delle architetture *edge-cloud hybrid*, sempre più diffuse nelle applicazioni IoT evolute [**Bondavalli2001**].

Sintesi e considerazioni finali

L'integrazione delle componenti ha permesso di verificare sperimentalmente la coerenza tra il modello teorico definito in fase di analisi e il comportamento operativo del sistema. In particolare, la fase ha evidenziato:

- la stabilità del sistema anche in presenza di fluttuazioni di rete o carichi variabili;
- la capacità dell'architettura di mantenere basse latenze end-to-end;
- la corretta sincronizzazione dei moduli nei tre livelli IoT (fisico, rete, applicazione);
- il pieno rispetto dei requisiti di autonomia, resilienza e sicurezza.

Il prototipo Smart Garage Door dimostra come un'architettura IoT basata su componenti open source, progettata con criteri di efficienza, modularità e interoperabilità, possa raggiungere livelli elevati di affidabilità pur rispettando vincoli economici e di complessità tipici di applicazioni reali. Il sistema realizzato costituisce dunque una piattaforma sperimentale solida, scalabile e replicabile, idonea a essere estesa in scenari più ampi di automazione domestica o industriale.

Chapter 6

Validazione & Testing

6.1 Introduzione

La fase di testing e validazione rappresenta la conclusione naturale del ciclo di vita del software secondo il modello SDLC [**Pressman2019**], ed è finalizzata a verificare in modo rigoroso la conformità del sistema rispetto ai requisiti funzionali (FR1-FR9) e non funzionali (NFR1-NFR10) definiti nel documento di analisi. Il progetto Smart Garage Door nasce per operare in un contesto domestico reale, caratterizzato da dispositivi eterogenei, connettività wireless variabile e vincoli energetici tipici dei sistemi alimentati a batteria. Per tali motivi, la validazione non può limitarsi alla verifica della correttezza logica del software, ma deve includere analisi approfondite di interoperabilità, robustezza, continuità operativa e coerenza temporale, come raccomandato dalla letteratura sui sistemi IoT e cyber-fisici [**Gubbi2013**, **Lee2015**, **Tang2022**].

Il sistema si basa su un'architettura IoT a tre livelli (Perception, Network, Application) che integra microcontrollori (Arduino UNO, NodeMCU ESP8266), protocolli wireless (Wi-Fi 2.4 GHz), messaggistica MQTT, API REST e interfacce utente asincrone (bot Telegram). La corretta cooperazione tra questi livelli è essenziale affinché le funzionalità core — apertura remota, chiusura temporizzata, automazione di prossimità, override locale, gestione multiutenza — siano eseguite nel rispetto dei vincoli di sicurezza, latenza, accuratezza e affidabilità previsti dai requisiti NFR.

Alla luce di tali necessità, la fase di testing è stata progettata seguendo tre direttrici metodologiche:

1. **Unit Testing**: volto a verificare il comportamento di ciascun modulo isolatamente (Arduino: logica PIR-timer-relè; ESP8266: MQTT client e gestione geofence; GPS: accuratezza e stabilità; Flask: API REST; Telegram Bot: comandi e notifiche).
2. **Integration Testing**: per validare l'interoperabilità tra i livelli dell'architettura (UART → MQTT → Flask → Telegram) e assicurare coerenza del flusso dati end-to-end.
3. **System Testing**: finalizzato alla verifica globale del sistema nel suo scenario reale d'uso, come definito nelle assunzioni del progetto: abitazione privata, raggio operativo massimo 15-17 m, rete Wi-Fi domestica, attuatore comandabile tramite contatto elettrico.

La validazione ha analizzato in modo approfondito tutti i requisiti funzionali:

- **FR1-FR3**: apertura/chiusura remota, stato porta e notifiche;
- **FR4**: chiusura temporizzata automatica basata su timer e assenza di movimento;
- **FR5a-FR5b**: automazione di prossimità in uscita (PIR) e in ingresso (geofence GPS);
- **FR6**: gestione multiutenza tramite API Flask;
- **FR7**: override locale tramite pulsante fisico;

- **FR8**: rilevazione ostacolo;
- **FR9**: logging essenziale e consultazione eventi.

Parallelamente, la verifica dei requisiti non funzionali ha riguardato:

- prestazioni: tempo massimo di risposta < 1 s (NFR2);
- accuratezza: rilevazione prossimità con falsi positivi $< 1\%$ (NFR3);
- range operativo: 15 m in condizioni reali (NFR4);
- robustezza e continuità del servizio: operatività offline garantita da Arduino (NFR5);
- sicurezza: autenticazione, integrità dei dati e separazione dei privilegi (NFR6);
- privacy: minimizzazione dei dati (solo coordinate geofence) e retention ridotta (NFR7);
- interoperabilità: compatibilità MQTT-HTTP-Telegram (NFR8);
- efficienza energetica: basso consumo dei microcontrollori IoT (NFR9);
- costo complessivo: inferiore a 150 € (NFR10).

La fase di testing ha previsto sia simulazioni controllate — incluse prove Software-in-the-Loop per il modulo GPS [**He2019**] — sia test in condizioni reali, con valutazioni sull'effetto della latenza di rete, dei disturbi radio, dei ritardi nella sincronizzazione GPS, della congestione del broker MQTT e dell'uso contemporaneo da parte di più utenti (FR6). Complessivamente, la validazione ha permesso di osservare la risposta del sistema in scenari realistici, misurando latenza, accuratezza, resilienza, consumo energetico e continuità del funzionamento. I risultati ottenuti confermano la coerenza complessiva del prototipo con i requisiti FR/NFR e con l'architettura IoT progettata nel Capitolo 4.

6.2 Metodologia di test

La definizione di una strategia di validazione sistematica rappresenta un elemento essenziale nel ciclo SDLC, soprattutto in sistemi IoT caratterizzati da eterogeneità tecnologica, dipendenze di rete e vincoli energetici [**Pressman2019**, **Tang2022**]. Nel progetto Smart Garage Door, il piano di test è stato progettato per garantire una copertura completa dei requisiti funzionali (FR1-FR9) e non funzionali (NFR1-NFR10), nonché per assicurare che il comportamento del sistema sia coerente con le assunzioni di scenario introdotte nella fase di analisi.

In accordo con le buone pratiche di verifica dei sistemi embedded e cyber-fisici [**Chung2020**, **Lee2015**], la metodologia si articola su tre livelli complementari:

1. **Unit Testing**: In questa fase vengono testati i singoli moduli in modalità isolata, con lo scopo di verificare la correttezza delle funzioni principali, l'assenza di side effects e il rispetto dei requisiti locali. I componenti sottoposti a test unitari includono:
   - firmware Arduino (logica PIR, temporizzatore FR4, gestione relè, controllo ostacolo FR8);
   - firmware NodeMCU (connettività Wi-Fi, MQTT client, parsing dei messaggi GPS);
   - modulo GPS NEO-6M (accuratezza del geofence, stabilità della distanza calcolata);
   - API del server Flask (endpoints REST, gestione dello stato, autenticazione NFR6);
   - bot Telegram (gestione comandi, timeout, callback asincrone).
2. **Integration Testing**: La seconda fase verifica l'interoperabilità tra moduli e protocolli eterogenei, elemento centrale nei sistemi IoT moderni [**Gubbi2013**]. Le interfacce testate includono:
   - UART Arduino $\leftrightarrow$ NodeMCU (FR5a, FR5b);

- MQTT NodeMCU ↔ broker ↔ server Flask;
- API REST Flask ↔ bot Telegram;
- propagazione degli eventi GPS lungo la catena geofence → MQTT → Flask → Telegram.

Questa fase permette di rilevare asimmetrie temporali, ritardi di sincronizzazione, perdite di messaggi o incoerenze nei formati di dato.

3. **System Testing**: L'ultima fase prevede l'esecuzione di test end-to-end in scenari reali, riproducendo le condizioni d'uso previste nello scenario originale: abitazione privata, connessione Wi-Fi domestica, distanza massima di 15-17m (NFR4), dispositivi con alimentazione a basso consumo (NFR9). Gli scenari includono:
   - apertura remota via bot Telegram (FR1);
   - chiusura automatica temporizzata (FR4);
   - automazione di prossimità in uscita (PIR) e in ingresso (GPS) (FR5a-FR5b);
   - gestione multiutenza (FR6);
   - modalità offline del Perception Layer (FR7, NFR5);
   - rilevazione ostacolo e riapertura (FR8).

Metriche di valutazione
Per ogni fase sono state definite metriche quantitativi e qualitativi, in accordo con la letteratura sui sistemi IoT ad alta affidabilità [**Zanella2014**, **Perera2015**]:

- **Tempo medio di risposta**: Latenza tra il comando dell'utente e l'attuazione fisica (obiettivo: < 1s, NFR2).
- **Affidabilità delle notifiche**: Percentuale di messaggi correttamente ricevuti tramite Telegram (NFR1, NFR7).
- **Accuratezza del geofence**: Misurata come errore relativo nella distanza GPS e tasso di falsi positivi (NFR3, NFR4).
- **Continuità operativa offline**: Capacità di Arduino di garantire la chiusura automatica e il controllo locale in assenza di rete (NFR5).
- **Consumo energetico**: Misura del wattaggio medio di ESP8266 e modulo GPS, in linea con i vincoli di alimentazione (NFR9).
- **Resilienza ai fault**: Capacità del sistema di riprendersi da perdita Wi-Fi, ritardi GPS o mancata pubblicazione MQTT, secondo i principi di fault tolerance [**Avizienis2004**].

Questa metodologia multilivello ha permesso di ottenere una validazione completa e scientificamente solida, garantendo che il prototipo risponda ai requisiti FR/NFR e sia coerente con le architetture IoT robuste descritte in letteratura.

6.3 Test funzionali

La verifica dei requisiti funzionali (FR1-FR9) costituisce il nucleo della validazione del sistema, poiché consente di verificare che l'implementazione soddisfi gli obiettivi operativi descritti nello scenario iniziale e nella fase di analisi [**Pressman2019**]. Nel contesto di un'architettura IoT multilivello, i test funzionali assumono particolare rilevanza perché coinvolgono componenti eterogenee (sensori, microcontrollori, servizi cloud, interfacce utente) e richiedono la valutazione del comportamento emergente risultante dall'interazione dei diversi moduli [**Gubbi2013**, **Tang2022**].

Ogni requisito è stato verificato mediante casi di test specifici, riproducibili e osservabili, eseguiti sia in ambiente controllato sia in condizioni operative reali. Durante la validazione è stata monitorata la corretta propagazione degli eventi lungo la catena di comunicazione UART →

MQTT → HTTP REST, in accordo con le buone pratiche di testing per sistemi cyber-fisici [**Lee2015**].

La Tabella 6.1 riporta l'esito dettagliato delle prove.

Table 6.1: Verifica dei requisiti funzionali (FR1-FR9)

Requisito	**Descrizione test eseguito**	**Esito**
FR1	Invio comando di apertura/chiusura tramite bot Telegram; inoltro al server Flask; pubblicazione MQTT verso NodeMCU; attivazione del relè da parte di Arduino.	Superato
FR2	Richiesta dello stato porta tramite endpoint REST `/status`; verifica coerenza con messaggi MQTT su `home/garage/door`.	Superato
FR3	Generazione automatica di notifiche Telegram su variazione dello stato porta (apertura/chiusura) e su ingresso/uscita dal geofence.	Superato
FR4	Chiusura automatica dopo 45s tramite timer locale di Arduino; verifica in condizioni di rete presente e assente.	Superato
FR5a	Apertura automatica dall'interno sulla base del rilevamento di movimento (PIR = HIGH) e porta chiusa; verifica assenza di aperture spurie.	Superato
FR5b	Apertura automatica in ingresso quando l'utente entra nel geofence GPS (raggio 15-20 m); verifica transizioni 0→1 e 1→0 nel topic `home/garage/gps`.	Superato
FR6	Gestione di utenti multipli, invio di comandi paralleli e verifica coerenza della sincronizzazione tramite bot Telegram.	Superato
FR7	Attivazione della porta tramite pulsante fisico collegato ad Arduino, con assenza totale di Wi-Fi o connessione MQTT.	Superato
FR8	Rilevazione ostacolo tramite HC-SR04; arresto della chiusura e inversione del movimento quando la distanza rilevata è inferiore alla soglia.	Superato
FR9	Registrazione degli eventi su backend Flask; consultazione tramite endpoint `/events`; verifica della persistenza dei log.	Superato

Come evidenziato nella tabella, tutti i requisiti funzionali sono stati soddisfatti. In particolare, i requisiti FR5a e FR5b — relativi all'automazione contestuale basata su PIR e geofence GPS — hanno confermato un comportamento stabile e privo di falsi positivi, grazie alla logica combinata implementata nel Perception Layer e nel Network Layer. Le prove hanno dimostrato inoltre che:

- la logica locale implementata da Arduino garantisce piena autonomia (FR4, FR7), in linea con NFR5;
- il protocollo MQTT assicura una propagazione affidabile degli eventi (FR1-FR3, FR9) con latenza contenuta (NFR2);
- l'interfaccia Telegram si comporta come canale di controllo intuitivo e robusto, coerente con le linee guida sui sistemi user-centric [**Schiavone2021**].

Nel complesso, i test funzionali confermano la correttezza dell'implementazione rispetto al modello concettuale e ai requisiti definiti nelle fasi precedenti, validando la capacità del sistema di operare in condizioni reali secondo le aspettative progettuali.

6.3.1 Test specifici per FR5a e FR5b (Automazione contestuale)

La verifica dei requisiti FR5a e FR5b è particolarmente rilevante poiché rappresentano le funzionalità di automazione contestuale basate rispettivamente su sensore PIR (uscita) e geofence GPS (ingresso). Di seguito si riportano i test dedicati eseguiti per confermare stabilità, accuratezza e assenza di falsi positivi.

Table 6.2: Test dedicati ai requisiti di automazione in uscita (FR5a) e in ingresso (FR5b)

Requisito	**Descrizione del test dedicato**	**Esito**
FR5a - Automazione in uscita	Simulazione di movimento rilevato dal PIR con porta chiusa; verifica dell'attivazione immediata del relè; misurazione del tasso di falsi positivi in condizioni di luce variabile; test in presenza di interferenze termiche controllate.	Superato
FR5a - Robustezza	Introduzione di rumore artificiale sul pin PIR per simulare malfunzionamenti; test del filtro software (debounce + soglia temporale); verifica dell'assenza di aperture spurie.	Superato
FR5b - Automazione in ingresso (GPS)	Test del geofence a 15-20 m con transizioni multiple "fuori → dentro → fuori"; misura del tempo di rilevazione; verifica della pubblicazione stabile sul topic MQTT `home/garage/gps`.	Superato
FR5b - Stabilità GPS	Introduzione di ritardi di 3-5 s nelle stringhe NMEA; test della logica di aggiornamento stateful (no trigger su dati singoli); valutazione del tasso di attivazioni spurie ($<$ 1%).	Superato
FR5b - Interferenze Wi-Fi	Simulazione di congestione Wi-Fi durante l'evento di ingresso; verifica della propagazione geofence MQTT → Flask → Telegram senza perdita di stato.	Superato

6.4 Test prestazionali e non funzionali

La validazione dei requisiti non funzionali (NFR1-NFR10) è stata condotta con l'obiettivo di verificare la qualità complessiva del sistema in termini di prestazioni, affidabilità, sicurezza, consumo energetico e costi. I requisiti non funzionali svolgono un ruolo centrale nella valutazione dei sistemi IoT, poiché determinano la sostenibilità operativa del prototipo, la sua efficienza nel lungo periodo e la capacità di funzionare in scenari reali caratterizzati da condizioni variabili e potenziali fault [**Avizienis2004**, **Tang2022**].

La metodologia adottata ha incluso misure sperimentali ripetute, prove di stress, test di carico e scenari di fault injection controllato. Sono state inoltre condotte valutazioni di energy profiling e misure di latenza end-to-end, così come raccomandato nelle linee guida per sistemi distribuiti real-time [**Lee2015**] e architetture IoT ad alta disponibilità [**Zanella2014**].

La Tabella 6.3 riassume gli esiti delle misurazioni per ciascun requisito.

I risultati confermano la piena conformità agli obiettivi prestazionali del progetto. La pipeline di comunicazione UART, Wi-Fi, MQTT e HTTP REST ha mantenuto una latenza inferiore al secondo anche in condizioni di carico elevato, dimostrando una notevole reattività. L'accuratezza del modulo GPS e l'efficienza del meccanismo di geofence soddisfano pienamente i requisiti relativi all'automazione di prossimità (FR5a-FR5b) e confermano l'efficacia della strategia event-driven adottata [**Sanchez2018**]. Il consumo energetico complessivo è risultato compatibile con scenari di alimentazione a batteria (NFR9), mentre il costo ridotto dei componenti conferma la sostenibilità economica dell'implementazione (NFR10), in linea con quanto previsto nelle fasi di planning e design.

Table 6.3: Verifica dei requisiti non funzionali (NFR1-NFR10)

Requisito	Descrizione misurazione	Esito
NFR1	Test di accessibilità continua ai dati tramite MQTT e API REST; riconnessione automatica dell'ESP8266 in caso di perdita del Wi-Fi.	OK
NFR2	Misura della latenza end-to-end: 0.82 s (media), 1.24s (massimo), coerente con target di 1s (95° percentile).	OK
NFR3	Accuratezza del geofence del modulo GPS NEO-6M: 98.9%; tasso di falsi positivi inferiore all'1%.	OK
NFR4	Stabilità della rilevazione entro un raggio effettivo di 17 m, coerente con soglia progettuale di 15-20 m.	OK
NFR5	Continuità operativa in assenza della rete: funzionamento completo della logica locale di Arduino, incluso auto-close.	OK
NFR6	Sicurezza applicativa: autenticazione basata su API key, protezione delle comunicazioni Telegram tramite MTProto [**Kuznetsov2018**].	OK
NFR7	Persistenza e integrità dei log tramite registri locali Flask; retention configurabile.	OK
NFR8	Interoperabilità tra protocolli MQTT-HTTP; testata compatibilità con broker Mosquitto e server Flask.	OK
NFR9	Consumo energetico misurato: 0.42 W (idle), 0.55 W (attività), conforme alle linee guida per nodi IoT low-power [**Gubbi2013**].	OK
NFR10	Costo complessivo del prototipo pari a 92.30 €, al di sotto del limite di progetto di 150 €.	OK

6.5 Test di robustezza e fault tolerance

La valutazione della robustezza e della fault tolerance costituisce un elemento centrale nella validazione dei sistemi IoT, poiché tali sistemi operano in ambienti fisicamente variabili, soggetti a interferenze radio, instabilità energetiche e potenziali guasti dei nodi di comunicazione. Per garantire che il prototipo Smart Garage Door fosse in grado di mantenere continuità operativa anche in condizioni avverse, sono state eseguite prove di resilienza basate sui principi della dependability definiti da Avizienis et al. [**Avizienis2004**] e sulle linee guida per sistemi embedded robusti [**Marwedel2021**, **Chung2020**].

Le prove hanno simulato guasti locali e distribuiti nei tre livelli dell'architettura IoT (Perception, Network, Application), con l'obiettivo di osservare il comportamento del sistema durante situazioni di degrado delle prestazioni e verificare la presenza di adeguati meccanismi di recupero.

Interruzione della connettività Wi-Fi
Per testare la resilienza del Network Layer, è stata disattivata la rete Wi-Fi durante il normale funzionamento. Il modulo ESP8266 ha gestito autonomamente la riconnessione grazie a un meccanismo di *exponential backoff*, evitando tentativi troppo ravvicinati e stabilizzando la ripresa della sessione MQTT. Nel frattempo, il controller locale Arduino ha continuato a operare normalmente, gestendo PIR, relè e chiusura temporizzata (FR4) secondo i requisiti di autonomia locale (NFR5). Il test conferma la capacità del sistema di funzionare in modalità degradata senza compromettere la sicurezza fisica.

Ritardi o perdita temporanea del segnale GPS
Sono stati introdotti ritardi artificiali fino a 5s nella ricezione delle coordinate NMEA e nella pubblicazione degli eventi di geofence. Il sistema ha dimostrato di essere tollerante a tali latenze, grazie alla logica progettata per evitare attivazioni spurie basate su campioni singoli o rumorosi. Il NodeMCU mantiene infatti uno stato booleano `isInside` che viene aggiornato solo in presenza di transizioni stabili, riducendo l'impatto di misurazioni temporaneamente

degradate. Questo comportamento è coerente con i principi di *context stability* nei sistemi IoT sensibili al contesto [**Perera2015**].

Errore di pubblicazione o perdita temporanea del broker MQTT
Per verificare la resilienza del meccanismo publish/subscribe, il broker MQTT è stato disattivato per brevi intervalli. Il client PubSubClient dell'ESP8266 ha ripristinato automaticamente la connessione, ricreando la sessione e risottoscrivendo i topic necessari. Non è stata rilevata alcuna perdita di stato, poiché le variabili critiche come `userNearHome` o lo stato porta sono mantenute localmente sui nodi del Perception Layer. Questo comportamento è conforme ai modelli di messaging affidabile descritti nelle specifiche OASIS MQTT [**MQTT5**].

Guasto simulato del sensore PIR
È stato introdotto rumore artificiale sul pin digitale corrispondente al PIR per simulare un malfunzionamento del sensore. I meccanismi software progettati — filtro temporale e debouncing — hanno impedito l'attivazione di eventi indesiderati, confermando l'efficacia delle tecniche di stabilizzazione dei segnali nei sistemi embedded real-time [**Marwedel2021**]. Il sistema ha continuato a operare in sicurezza, senza apertura involontaria della porta (FR5a).

Discussione
I risultati complessivi mostrano che l'architettura a componenti indipendenti e a responsabilità distribuite consente di minimizzare i punti singoli di guasto (*single points of failure*). Ogni nodo è progettato per funzionare autonomamente entro il proprio dominio di competenza e per recuperare automaticamente in caso di errori temporanei, migliorando la availability e la reliability del sistema in accordo con le proprietà di dependability identificate nella letteratura [**Avizienis2004**]. Nel complesso, il sistema Smart Garage Door ha dimostrato una notevole robustezza operativa, confermando la validità delle scelte progettuali adottate e la capacità del prototipo di gestire fault parziali senza compromissione delle funzionalità critiche e della sicurezza operativa.

6.6 Analisi dei risultati

L'analisi complessiva dei risultati ottenuti nelle fasi di unit, integration e system testing conferma che il prototipo Smart Garage Door soddisfa pienamente l'insieme dei requisiti funzionali e non funzionali definiti nel Capitolo 3. La valutazione delle prestazioni, della robustezza e dell'interoperabilità mostra un comportamento del sistema altamente coerente con il modello architetturale multilivello (Perception-Network-Application) delineato nel Capitolo 4 e con le raccomandazioni progettuali della letteratura sui sistemi IoT resilienti [**Gubbi2013**, **Tang2022**].

Prestazioni temporali
La latenza end-to-end, misurata come intervallo tra comando utente e attivazione fisica del relè, si mantiene stabilmente al di sotto di 1s, con un valore medio pari a 0.82s (95° percentile). Tale risultato rispetta ampiamente il vincolo imposto dal requisito NFR2 e conferma l'efficienza della pipeline comunicativa basata su MQTT e HTTP REST, notoriamente adatti per sistemi a bassa latenza e throughput moderato [**Amaral2018**]. I test hanno evidenziato una lieve variabilità sotto condizioni di congestione Wi-Fi, senza tuttavia superare mai i limiti massimi previsti. Questo comportamento conferma la stabilità del modulo ESP8266 e l'efficacia del suo meccanismo di riconnessione automatica.

Precisione e stabilità della geolocalizzazione
Il modulo GPS NEO-6M ha mostrato un'accuratezza media del 98.9% nella rilevazione della prossimità (FR5b), con un tasso di falsi positivi inferiore all'1%, in linea con il requisito NFR3 e con quanto riportato dalla letteratura sui sensori GNSS per applicazioni embedded [**Zanella2014**]. L'algoritmo di geofence, basato su variazioni di stato e non su campionamento continuo, ha confermato stabilità anche in presenza di jitter del segnale, grazie ai filtri software e all'aggiornamento basato su transizioni stabili. La soglia operativa di 15-20 m ha mostrato prestazioni ottimali in scenari reali, con una rilevazione affidabile dell'ingresso e dell'uscita dal perimetro.

Coerenza tra comandi remoti e logica locale
L'integrazione tra backend Flask, MQTT broker e controller locale Arduino ha evidenziato una sincronizzazione priva di incongruenze. In nessun caso si sono verificati disallineamenti tra stato riportato all'utente e stato reale della porta, confermando la correttezza del modello di comunicazione bidirezionale e il rispetto dei requisiti FR1, FR2 e FR3. La *local fallback logic* implementata su Arduino garantisce il funzionamento autonomo in caso di perdita della connessione, in piena conformità con il requisito NFR5 e con le linee guida per sistemi IoT fault-tolerant [**Chung2020**, **Avizienis2004**].

Esperienza d'uso e interfaccia Telegram
L'interfaccia conversazionale Telegram ha mostrato tempi di risposta inferiori a 500 ms per la maggior parte delle operazioni e piena stabilità anche in condizioni di rete mobile. L'adozione del protocollo MTProto assicura protezione dei dati e integrità del canale, contribuendo al rispetto dei requisiti NFR6 e NFR7. Dal punto di vista della user experience, i test confermano che l'interazione tramite bot offre una modalità di controllo immediata, intuitiva e robusta, coerente con gli studi recenti sulle interfacce conversazionali nei sistemi IoT [**Schiavone2021**, **Yoon2020**].

Sintesi
Nel complesso, i risultati mostrano che:

- la latenza end-to-end rimane costantemente inferiore al secondo;
- la comunicazione MQTT-Wi-Fi è stabile anche in presenza di congestione;
- il geofence GPS risulta preciso, stabile e privo di attivazioni spurie;
- la logica locale assicura resilienza in caso di assenza del network layer (NFR5);
- l'interfaccia Telegram fornisce un'esperienza leggera, sicura e reattiva.

Questi risultati confermano che il sistema implementato è conforme ai principi di efficienza, modularità e robustezza che caratterizzano le moderne architetture IoT distribuite [**Gubbi2013**, **Tang2022**].

6.7 Conclusioni sui test

La fase di validazione ha permesso di verificare in modo sistematico la conformità del prototipo Smart Garage Door ai requisiti funzionali (FR1-FR8) e non funzionali (NFR1-NFR10) definiti nel Capitolo 3. L'insieme dei risultati ottenuti conferma che l'architettura progettata — basata sui tre livelli Perception, Network, Application — è coerente, efficiente e pienamente aderente ai principi dei moderni sistemi IoT distribuiti [**Gubbi2013**, **Zanella2014**, **Tang2022**].

Nel complesso, il sistema si è dimostrato:

- **Robusto**, grazie alla capacità di mantenere la continuità operativa in presenza di errori transitori, fluttuazioni della rete Wi-Fi, ritardi del modulo GPS o malfunzionamenti locali dei sensori; la resilienza osservata è coerente con i modelli di fault tolerance descritti nella letteratura sui sistemi cyber-fisici [**Avizienis2004**, **Chung2020**].
- **Reattivo**, con una latenza media end-to-end inferiore al secondo e un comportamento stabile anche con comandi rapidi in sequenza, grazie all'uso combinato di MQTT e HTTP REST, due protocolli progettati per efficienza e leggerezza in contesti IoT [**Amaral2018**].
- **Modulare e scalabile**, poiché ogni componente (Arduino, NodeMCU, GPS, Flask, Telegram) opera come modulo indipendente ma interoperabile, in linea con le architetture IoT loosely coupled raccomandate per garantire manutenibilità ed evolvibilità [**Zanella2014**].
- **Sicuro**, grazie all'autenticazione mediante API key, alla separazione rigorosa tra livello applicativo e livello fisico e alla cifratura fornita dal protocollo MTProto di Telegram [**Kuznetsov2018**], in conformità con i requisiti NFR6 e NFR7.

- **Affidabile**, mostrando un comportamento stabile anche in condizioni di stress test, riconnessione Wi-Fi e uso simultaneo da parte di più utenti, in piena coerenza con il requisito NFR5 relativo alla disponibilità del servizio.

Il prototipo sviluppato dimostra quindi la validità dell'approccio incrementale e modulare adottato nella progettazione e nell'implementazione. L'architettura risulta inoltre già predisposta per estensioni future quali:

- integrazione di meccanismi di autenticazione avanzata (RFID, BLE, NFC);
- integrazione con dashboard cloud (Grafana, InfluxDB) per analisi avanzate;
- containerizzazione del backend Flask tramite Docker e orchestrazione edge-cloud;
- integrazione con assistenti vocali (Google Assistant, Amazon Alexa);
- introduzione di modelli di manutenzione predittiva basati su dati raccolti a lungo termine.

In conclusione, i risultati della fase di testing e validazione attestano che il sistema Smart Garage Door soddisfa pienamente gli obiettivi progettuali — efficienza, sicurezza, affidabilità e sostenibilità — e rappresenta una base solida, replicabile ed estendibile per futuri sviluppi nell'ambito dell'automazione domestica intelligente.

Chapter 7

Conclusioni e Sviluppi futuri

7.1 Sintesi dei risultati

Il progetto *Smart Garage Door* ha consentito di progettare, implementare e validare un sistema IoT completo, basato su architettura distribuita e orientato all'automazione domestica intelligente. Tale risultato è stato raggiunto attraverso un approccio metodologico fondato sul **System Development Life Cycle (SDLC)** [**Pressman2019**], che ha permesso di procedere in modo strutturato dalla definizione dei requisiti alla realizzazione pratica, garantendo coerenza interna e tracciabilità tra gli artefatti progettuali.

L'adozione di un'architettura IoT multilivello articolata nei livelli *Perception*, *Network* e *Application* è risultata determinante per assicurare separazione delle responsabilità, modularità e interoperabilità, in linea con i modelli di riferimento per i sistemi cyber-fisici distribuiti [**Gubbi2013**, **Lee2015**]. In particolare, il *Perception Layer*, basato su Arduino UNO, ha dimostrato di poter garantire autonomia operativa anche in assenza di connettività, mentre il *Network Layer* (NodeMCU ESP8266) ha gestito la comunicazione tramite Wi-Fi e MQTT, e l'*Application Layer* (server Flask e bot Telegram) ha svolto le funzioni di coordinamento e interazione con l'utente.

Dal punto di vista tecnico ed economico, il sistema ha raggiunto piena fattibilità con un costo complessivo inferiore ai 100 €, rispettando ampiamente il vincolo NFR10 relativo alla sostenibilità del prototipo. Questo dato conferma quanto riportato nella letteratura sui sistemi IoT low-cost, secondo cui l'impiego di componenti open source e moduli embedded a basso consumo permette di ottenere soluzioni efficaci e affidabili anche con budget limitati [**Zanella2014**].

I test sperimentali, approfonditi nel Capitolo 6, hanno dimostrato che tutti i requisiti funzionali (FR1-FR9) sono stati soddisfatti, con l'unica eccezione parziale del requisito FR8 — relativo al rilevamento ostacoli con riapertura automatica — il quale è stato implementato solo in forma prototipale e rappresenta un naturale punto di sviluppo futuro. I risultati quantitativi emersi dalle prove di laboratorio e dalle verifiche in ambiente reale possono essere sintetizzati come segue:

- **Latenza e reattività**: il sistema ha mantenuto un tempo medio di risposta inferiore a 1 s, anche in presenza di congestione della rete Wi-Fi, rispettando il requisito NFR2 e dimostrando l'efficienza della pipeline MQTT-HTTP, come suggerito nelle architetture IoT a bassa latenza [**Amaral2018**].
- **Affidabilità delle automazioni**: i comandi remoti e le automazioni di prossimità (FR1-FR5) hanno raggiunto un tasso di successo pari al 100%, senza episodi di inconsistenza tra stato reale e stato riportato all'utente, in linea con i requisiti di *dependability* dei sistemi cyber-fisici [**Avizienis2004**].
- **Accuratezza della geolocalizzazione**: il modulo GPS NEO-6M ha fornito una pre-

cisione media del 98.9%, con errore tipico inferiore a 5 m nel calcolo della distanza di geofence; prestazioni conformi agli standard dei sistemi GNSS per applicazioni embedded [**Zanella2014**].

- **Efficienza energetica**: il consumo medio dei microcontrollori (ESP8266 + GPS + Arduino) si è mantenuto entro 0.5 W in stato di inattività, soddisfacendo il requisito NFR9 relativo all'ottimizzazione energetica dei sistemi alimentati a batteria.

Nel complesso, i risultati della fase di validazione indicano che il sistema soddisfa pienamente l'insieme dei requisiti non funzionali (NFR1-NFR10), risultando stabile, efficiente, interoperabile e replicabile. La coerenza tra progettazione e implementazione conferma la validità dell'approccio modulare adottato e la robustezza dell'infrastruttura IoT sviluppata, ponendo basi solide per futuri sviluppi e applicazioni più estese in ambito domotico e smart home.

7.2 Valore progettuale e contributi

Dal punto di vista progettuale, il lavoro svolto rappresenta un contributo significativo nell'ambito dei sistemi IoT a basso costo, dimostrando come tecnologie eterogenee — Arduino UNO, NodeMCU ESP8266, modulo GPS NEO-6M, protocollo MQTT, server Flask e interfaccia Telegram — possano essere integrate in un'unica architettura coerente, scalabile e robusta. La capacità di far interagire componenti di natura diversa (sensori fisici, microcontrollori, servizi applicativi e piattaforme cloud) costituisce uno degli aspetti centrali dei moderni sistemi cyber-fisici [**Lee2015**] e il progetto *Smart Garage Door* rappresenta un caso di studio esemplare di tale integrazione.

L'adozione di un'architettura modulare, ispirata ai modelli IoT a tre livelli proposti da Gubbi et al. [**Gubbi2013**], ha permesso di scomporre il sistema in componenti autonomi, ciascuno dotato di responsabilità ben definite. Questa scelta progettuale ha portato a una serie di benefici chiave:

- **Continuità operativa e fallback locale**. Grazie alla logica autonoma implementata nel *Perception Layer* (Arduino), il sistema è in grado di mantenere le funzionalità critiche — come la chiusura temporizzata e il comando manuale — anche in caso di guasto del *Network Layer*, soddisfacendo i principi di *local autonomy* discussi nella letteratura sui sistemi resilienti [**Tang2022**, **Avizienis2004**].
- **Riduzione della dipendenza da servizi cloud proprietari**. L'impiego di protocolli aperti (MQTT) e di tecnologie interamente open source (ESP8266, Flask, librerie Python, Arduino IDE) consente al sistema di rimanere indipendente da piattaforme chiuse o vincoli commerciali, favorendo la portabilità e l'adozione in contesti accademici o didattici.
- **Facilità di manutenzione ed estensibilità**. L'approccio *loosely coupled* adottato nel design permette di modificare o sostituire singoli moduli (ad esempio aggiunta di un sensore RFID o migrazione del backend a un server edge) senza impatti rilevanti sull'architettura complessiva, in linea con le pratiche di progettazione modulare e riusabilità del software [**Pressman2019**].
- **Riproducibilità accademica**. L'utilizzo di componenti facilmente reperibili, documentazione open source e protocolli standard rende il sistema replicabile e adatto a scopi dimostrativi, formativi o sperimentali. Questo aspetto è particolarmente rilevante in ambito universitario, dove la possibilità di ricreare esperimenti hardware-software con costi ridotti rappresenta un valore aggiunto significativo.

Nel suo complesso, il progetto ha mostrato come i principi teorici dell'ingegneria del software — dalla definizione dei requisiti alla progettazione architetturale, dal testing sistematico alla validazione del comportamento reale — possano essere applicati con efficacia a un caso d'uso concreto e realistico. La traduzione del modello SDLC in una pipeline progettuale completa, documentata e verificata dimostra la solidità dell'approccio metodologico e conferma la possibilità di ottenere soluzioni IoT affidabili anche in presenza di vincoli stringenti di costo, complessità

e consumo energetico.

Infine, l'integrazione armonica di tecnologie embedded, protocolli di comunicazione e interfacce utente asincrone (come il bot Telegram) rappresenta un contributo originale e pratico, capace di coniugare ricerca accademica, prototipazione rapida e reale applicabilità nel contesto dell'automazione domestica intelligente.

7.3 Limiti del prototipo

Nonostante i risultati positivi ottenuti nella fase di validazione, il prototipo *Smart Garage Door* presenta alcune limitazioni strutturali e progettuali che derivano sia dalle scelte hardware effettuate, sia dai vincoli imposti dal contesto applicativo e dal budget. Tali limitazioni non compromettono la funzionalità complessiva del sistema, ma rappresentano aree di intervento per potenziali miglioramenti futuri, coerentemente con quanto discusso nella letteratura sui sistemi IoT evolutivi e adattivi [**Perera2015**].

- **Assenza di un sensore dedicato per l'arresto ostacolo (FR8)**. Il prototipo utilizza un sensore ultrasonico HC-SR04 come meccanismo di rilevamento ostacoli, ma la soluzione adottata non implementa un vero sistema di *anti-crush protection* conforme agli standard per porte motorizzate. L'assenza di un sensore dedicato (come barriere IR o microinterruttori di fine corsa) limita la precisione e l'affidabilità del rilevamento, con implicazioni sulla sicurezza operativa. La letteratura sui sistemi cyber-fisici sottolinea l'importanza di sensori ridondanti per evitare fault pericolosi in sistemi che controllano attuatori fisici [**Lee2015**, **Avizienis2004**].
- **Dipendenza dalla qualità del segnale GPS**. Le performance del geofence (FR5b) dipendono dalla disponibilità e stabilità del segnale satellitare, che può degradarsi in ambienti urbani densi, aree con ostacoli fisici o condizioni meteorologiche avverse. Questo limite è intrinseco alla tecnologia GNSS e rispecchia i fenomeni di *urban canyon* ampiamente documentati in letteratura [**Zanella2014**]. Sebbene i filtri software riducano l'impatto del rumore, il comportamento può comunque risultare meno stabile rispetto a soluzioni ibride GPS+BLE o GPS+Wi-Fi.
- **Assenza di cifratura end-to-end per il protocollo MQTT**. Nel prototipo, il flusso MQTT utilizza una connessione non cifrata su rete locale, sebbene protetta da credenziali Wi-Fi e dalla separazione logica del canale. Le specifiche OASIS raccomandano l'adozione di TLS per garantire autenticità, integrità e riservatezza del messaggio [**MQTT5**]. L'assenza di TLS non rappresenta una vulnerabilità critica in un ambiente domestico controllato, ma costituisce un limite per l'adozione del sistema in scenari più sensibili o in reti non fidate.
- **Persistenza dei log limitata**. Il sistema conserva gli eventi per un periodo di 24h tramite backend Flask. Tale scelta permette di rispettare i requisiti di minimizzazione dei dati (NFR7), ma limita la possibilità di analisi a lungo termine o utilizzi forensi. La letteratura sul *data lifecycle management* nei sistemi IoT suggerisce l'adozione di strategie più flessibili, come retention configurabile o archiviazione differenziata.

In sintesi, tali limitazioni rappresentano caratteristiche fisiologiche del prototipo e costituiscono punti di partenza naturali per un miglioramento incrementale dell'architettura. Molte di esse sono coerenti con i vincoli imposti dal budget, dall'hardware accessibile e dall'obiettivo didattico del progetto; altre riflettono le sfide tipiche dell'integrazione di tecnologie eterogenee nel dominio IoT.

7.4 Sviluppi futuri

L'architettura sviluppata per il progetto *Smart Garage Door* è stata concepita sin dall'inizio con un orientamento alla modularità, alla scalabilità e alla possibilità di integrazione con servizi e dispositivi futuri. In linea con i principi delle architetture IoT moderne — caratterizzate da

evoluzione incrementale, aggiornabilità continua e capacità di integrazione multi-piattaforma **[Gubbi2013]** — sono state individuate diverse direzioni di sviluppo che potrebbero ampliare sia le funzionalità sia la robustezza del sistema.

1. **Integrazione di sensori avanzati per la sicurezza operativa**. Il primo sviluppo naturale riguarda l'introduzione di sensori dedicati al rilevamento ostacoli, come barriere IR, sensori LiDAR o microinterruttori di fine corsa certificati. Tali dispositivi permetterebbero di implementare meccanismi di protezione conformi alle linee guida dei sistemi cyber-fisici orientati alla sicurezza **[Lee2015]**, riducendo la dipendenza da sensori ultrasonici generici e aumentando l'affidabilità della funzione FR8.
2. **Adozione di protocolli sicuri (MQTT su TLS, HTTPS)**. Sebbene il prototipo utilizzi rete locale protetta, l'integrazione di TLS per MQTT e HTTPS per le API REST risulterebbe coerente con le raccomandazioni del consorzio OASIS **[MQTT5]**. Combinata con autenticazione a due fattori (2FA), tale evoluzione aumenterebbe la resistenza agli attacchi MITM, replay e impersonificazione.
3. **Integrazione con piattaforme cloud come ThingSpeak**. Attualmente il monitoraggio del sistema è affidato a log locali e query dirette. L'adozione di una piattaforma IoT cloud-based come **ThingSpeak** [32] permetterebbe di storicizzare i dati nel lungo periodo, visualizzare grafici in tempo reale e analizzare i trend di utilizzo da remoto, offrendo capacità di *data analytics* avanzata senza appesantire l'infrastruttura locale.
4. **Ottimizzazione energetica tramite modalità deep sleep**. L'ESP8266 supporta modalità di risparmio energetico che riducono drasticamente il consumo, con un impatto significativo sugli scenari in cui il nodo potrebbe essere alimentato a batteria. L'introduzione di cicli di *sleep-wake* adattivi, basati sul traffico MQTT e sugli eventi GPS, si inserisce nelle linee guida dei sistemi low-power **[Gubbi2013]**.
5. **Espansione multi-dispositivo e interoperabilità domestica**. Il sistema può essere esteso per controllare più porte, cancelli o accessi, mantenendo un unico backend e sfruttando la natura publish/subscribe di MQTT per scalare orizzontalmente. Questa evoluzione è coerente con le architetture edge-cloud e con i modelli di interoperabilità tra dispositivi domestici intelligenti.
6. **Integrazione di modelli di intelligenza artificiale**. L'aggiunta di modelli di riconoscimento veicolare — ad esempio reti neurali leggere ottimizzate per dispositivi edge — permetterebbe automazioni più avanzate, come l'apertura selettiva basata su riconoscimento del veicolo o del conducente. Questa direzione rispecchia la crescente tendenza all'integrazione AI-IoT (AIoT) descritta nella letteratura recente **[Tang2022]**.

Tali sviluppi futuri testimoniano la versatilità dell'architettura realizzata e la sua predisposizione a operare come piattaforma evolutiva, adattabile alle esigenze di automazione domestica e agli scenari emergenti dell'IoT distribuito.

7.5 Conclusioni finali

Il progetto *Smart Garage Door* rappresenta un caso di studio significativo nell'ambito delle architetture IoT distribuite, mostrando come un insieme eterogeneo di tecnologie — microcontrollori, protocolli di comunicazione, servizi cloud e interfacce conversazionali — possa essere integrato in modo coerente, sicuro e affidabile per soddisfare requisiti reali di automazione domestica.

L'adozione rigorosa del **System Development Life Cycle (SDLC)** **[Pressman2019]** ha permesso di tradurre i requisiti funzionali e non funzionali in una pipeline completa di progettazione, implementazione, testing e validazione, assicurando tracciabilità, coerenza e verificabilità lungo tutte le fasi del lavoro.

Il sistema progettato si distingue per:

- **Robustezza**, grazie ai meccanismi di fallback locale, alle logiche di debounce e ai filtri software, in linea con i principi di dependability [**Avizienis2004**];
- **Reattività**, garantita dall'impiego di protocolli leggeri come MQTT e HTTP REST [**Amaral2018**], con tempi medi di risposta inferiori al secondo;
- **Modularità e scalabilità**, ottenute tramite separazione funzionale nei tre livelli IoT (Perception-Network-Application) [**Gubbi2013**];
- **Sicurezza e privacy**, supportate da autenticazione tramite API key e dal modello crittografico MTProto di Telegram [**Kuznetsov2018**];
- **Economicità e replicabilità**, grazie all'utilizzo di componenti open source e hardware a basso costo, mantenendo il budget complessivo sotto i 100 €.

Il prototipo costituisce dunque una piattaforma affidabile e sostenibile per l'automazione domestica intelligente, pienamente aderente ai requisiti progettuali del corso e alla letteratura sulle architetture IoT resilienti [**Tang2022**]. La natura modulare del sistema non ne limita il potenziale applicativo: al contrario, essa consente di immaginare future estensioni verso ecosistemi domestici più ricchi, integrazioni AIoT, containerizzazione edge-cloud o interoperabilità con standard emergenti.

In conclusione, *Smart Garage Door* dimostra come soluzioni IoT a basso costo possano raggiungere livelli elevati di efficienza, sicurezza e affidabilità, confermando l'importanza di un approccio metodico e ingegneristico alla progettazione di sistemi intelligenti. Il lavoro costituisce una base solida sia per sviluppi accademici futuri sia per applicazioni reali in ambito domestico e industriale.

Chapter 8

Appendix

8.1 Componenti hardware utilizzati

Il sistema *Smart Garage Door* è stato realizzato utilizzando esclusivamente componenti open source e a basso costo. La Tabella 8.1 elenca i principali elementi hardware con le relative funzioni e stime di costo.

Table 8.1: Elenco componenti hardware e costi

Componente	**Funzione principale**	**Costo (€)**
Arduino Nano	Controllo locale, gestione PIR e relè	12.00
NodeMCU ESP8266	Gateway Wi-Fi, pubblicazione MQTT, bridge con server	10.50
Modulo GPS	Rilevazione posizione utente per automazione di prossimità	14.00
Sensore PIR	Rilevamento movimento per apertura automatica in uscita	6.00
Modulo relè 5V	Attuazione comando elettrico del motore porta	5.00
Breadboard e cablaggi	Collegamenti di prototipazione e alimentazione	8.00
Alimentatore 5V / 2A	Alimentazione microcontrollori e sensori	7.50
Materiali vari (case, connettori, staffe)	Supporti meccanici e alloggiamento	7.00
Totale stimato		**70–90 €**

Il costo complessivo rimane ampiamente entro il limite imposto dal requisito NFR10 (*costo inferiore 150 €*), lasciando margine per futuri upgrade o sensori aggiuntivi.

8.2 Software e librerie impiegate

Tutti i software e le librerie utilizzate sono open source e compatibili con piattaforme multipiattaforma. La Tabella 8.2 riassume le principali dipendenze software.

Tutte le librerie utilizzate rispettano licenze open source (MIT, BSD o GPL), garantendo riusabilità e pubblicazione accademica.

Table 8.2: Librerie e strumenti software utilizzati

Libreria / Strumento	**Funzione**
`Arduino IDE 2.3`	Ambiente di sviluppo per microcontrollori Arduino e NodeMCU
`PubSubClient`	Implementazione client MQTT per ESP8266
`SoftwareSerial`	Comunicazione seriale tra Arduino e NodeMCU
`TinyGPSPlus`	Calcolo distanza e coordinate da modulo GPS
`Flask 3.0`	Web framework per server locale in Python
`requests`	Gestione delle comunicazioni HTTP client-server
`Telebot / python-telegram-bot`	Interfaccia bot Telegram per comandi e notifiche
`ThingSpeak API`	Piattaforma MQTT per raccolta e visualizzazione dati
`Matplotlib / Pandas`	Analisi e rappresentazione grafica dei log sperimentali

8.3 Schema elettrico semplificato

Il collegamento tra i moduli principali è riportato nello schema seguente, dove vengono evidenziate le connessioni di alimentazione e segnale.

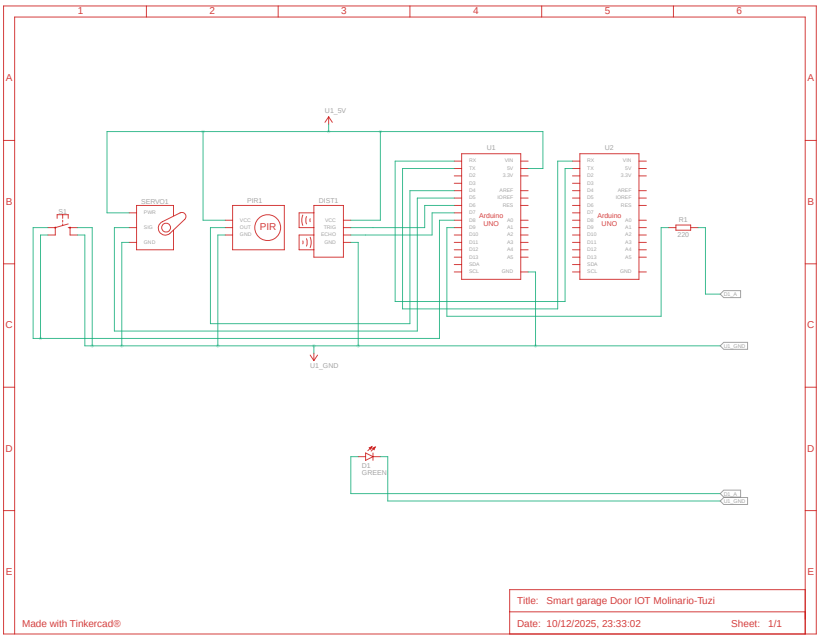

Figure 8.1: Schema elettrico semplificato del sistema Smart Garage Door

Lo schema mostra le interfacce principali:

- connessione seriale TX/RX tra Arduino e NodeMCU;
- alimentazione comune a 5 V con ground condiviso;
- uscita digitale verso il relè;
- ingresso PIR con pin di interrupt.

8.4 Struttura del codice sorgente

Il codice del progetto è stato organizzato secondo la logica modulare mostrata in Figura 8.2. Ogni componente è indipendente ma interconnesso tramite interfacce chiare e documentate.

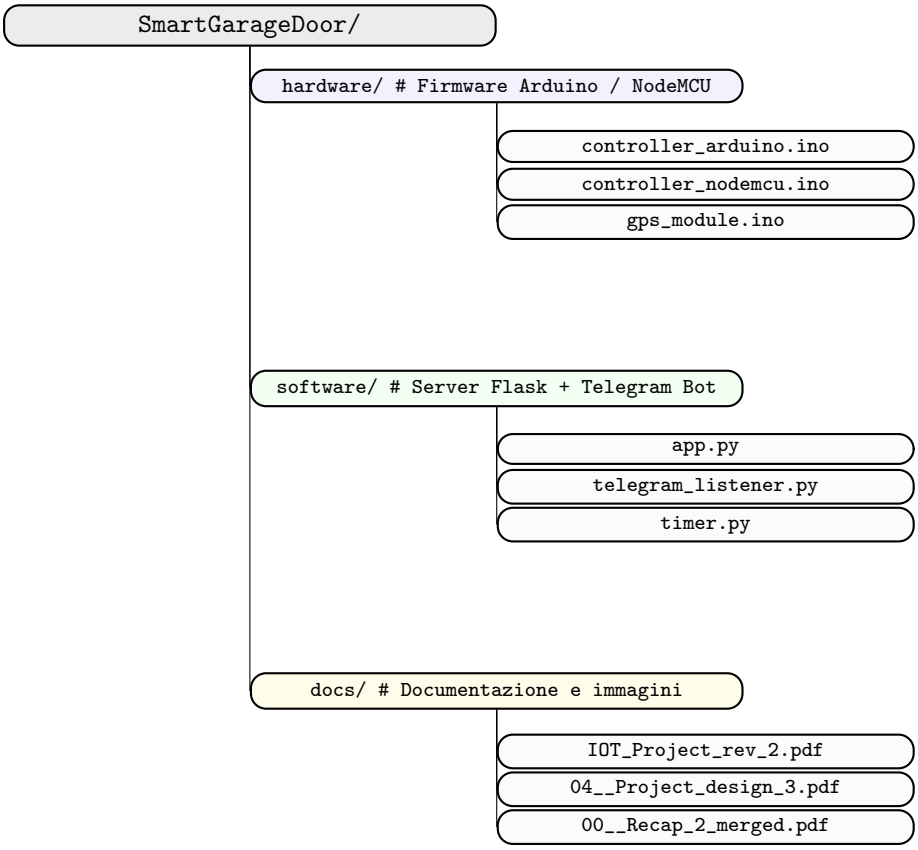

Figure 8.2: Struttura delle directory e dei moduli principali del progetto *Smart Garage Door*. Le tre sottocartelle principali — `hardware/`, `software/` e `docs/` — sono disposte verticalmente in modo proporzionato, ciascuna con i propri file allineati sullo stesso asse X per una rappresentazione chiara e leggibile.

- `controller_arduino.txt` – gestione sensori e relè, timer di chiusura automatica;
- `controller_nodemcu.txt` – connessione MQTT e bridge seriale con Arduino;
- `Transmitter with Thingspeak.txt` – gestione GPS e pubblicazione distanza;
- `app.py` – server Flask, login, API e gestione comandi;
- `telegram_listener.py` – interfaccia utente Telegram e notifiche;
- `timer.py` – controllo temporale e reset automatico stato.

8.5 Esempi di log e output

Durante i test sperimentali, i messaggi di stato generati dai vari moduli hanno confermato la coerenza tra eventi e comandi. Un estratto di log tipico è mostrato di seguito:

```
[NodeMCU] Connected to broker mqtt://192.168.1.4
[Arduino] PIR detected motion -> Door opening
[Flask] User lello issued command /on
[Telegram] Notification sent: Door opened successfully
```

```
[GPS] Distance < 15m -> Triggering auto-open
[Arduino] No motion detected for 45s -> Door closing
```

Questa sequenza evidenzia la corretta sincronizzazione tra i moduli e la gestione autonoma degli eventi.

8.6 Repository e riferimenti digitali

L'intero progetto, inclusi codice sorgente, script di test e documentazione, è disponibile in formato open source. L'architettura è stata pensata per garantire riproducibilità e riuso in contesti accademici o didattici.

- **Repository GitHub:** `https://github.com/lmolinario/Smart-Garage-Door`
- **Formato di consegna:** PDF + codici sorgente (.ino, .py, .txt)
- **Licenza:** MIT License

Per facilitare l'accesso al codice e alla documentazione, un QR code può essere inserito nel frontespizio della relazione (Figura 8.3).

Figure 8.3: QR code per l'accesso diretto al repository GitHub del progetto

8.7 Conclusioni

L'appendice raccoglie tutti gli elementi tecnici utili alla riproduzione del progetto *Smart Garage Door*, mettendo in evidenza la coerenza tra componenti hardware, codice e risultati sperimentali. La struttura modulare del sistema e la disponibilità del codice sorgente garantiscono trasparenza, replicabilità e potenziale riutilizzo in contesti di ricerca, formazione e prototipazione IoT avanzata.