UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Progettazione e implementazione delle componenti hardware e software di uno smart locker

Design and implementation of the hardware and software components of a smart locker

Relatore

Prof. Domenico Ursino

Candidato

Massimiliano Piccinini

ANNO ACCADEMICO 2023-2024

L'unico modo per fare un grande lavoro è amare ciò che fai.
Se non lo hai ancora trovato, continua a cercare.
Non accontentarti.
Come in tutte le questioni del cuore, lo saprai quando lo troverai

Steve Jobs

Sommario

Negli ultimi anni l'Internet of Things (IoT) ha assunto un'importanza sempre maggiore, poiché consente, da un lato, di prendere decisioni operative strategiche in vari settori dell'industria, e dall'altro di agevolare lo svolgimento delle attività quotidiane del grande pubblico. In questa tesi vengono illustrati la progettazione e l'implementazione di uno smart locker, che si occupa dello stoccaggio e del prelievo, da parte del cliente, dei prodotti di una ferramenta. In particolare, ci si è concentrati sulle componenti elettronica, di back-end e dell'applicazione mobile, e nel loro coordinamento. La progettazione è stata condotta tenendo conto delle best practice relative all'ingegneria del software, mentre, per l'implementazione, ci si è basati sulle tecniche di programmazione avanzata. In particolare, oltre la specifica e l'analisi dei requisiti, sono stati definiti i diagrammi delle classi e di flusso, e sono stati scelti i pattern che meglio si prestano per la struttura e per il funzionamento del sistema. Le tecnologie utilizzate principalmente sono Arduino, per la componente elettronica, Firebase per la componente di back-end e Flutter per quella di front-end.

Keyword: Ingegneria del Software, Specifica e analisi dei requisiti, Arduino, IoT, API, Firebase, App cross-platform, Pattern MVC

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Da qualche anno, ormai, il mercato del commercio online è caratterizzato da una forte crescita, in particolare dall'inizio della pandemia di COVID-19, la quale, per necessità, ha spinto molti consumatori verso l'acquisto online a causa delle restrizioni e della chiusura dei negozi fisici.

Tale onda positiva viene cavalcata da moltissime aziende, che, nel frattempo, data la situazione, hanno deciso di investire in questo settore, implementando il proprio sito di e-commerce.

Tra le varie tecnologie moderne spiccano, di certo, la data analytics per monitorare e migliorare i servizi offerti, i sistemi di intelligenza artificiale per la personalizzazione dell'esperienza da parte del cliente, i metodi di pagamento più diversificati e sicuri e i servizi per la logistica, per velocizzare i trasporti, rispettando, ad ogni modo, le norme sulla sostenibilità.

In questa tesi ci concentreremo, in particolare, proprio sul settore della logistica e della consegna, in cui c'è ancora molto da migliorare.

Scendendo nei dettagli, il colosso dell'e-commerce Amazon ha iniziato a sviluppare ed installare, nelle città, degli smart locker, ovvero degli armedietti cosiddetti *intelligenti*, poiché possono essere usati come locazione di consegna dei prodotti acquistati tramite il loro sito web, risolvendo il tipico problema in cui incorrevano numerosissimi utenti, per i quali, durante l'orario di consegna, era impossibile trovarsi a casa e, quindi, ritirare il proprio acquisto.

Tali smart locker, però, possono essere utilizzati solo nel caso in cui l'ordine provenga dal sito web di Amazon, mentre le aziende che oggi possiedono, o stanno implementando, un sito web o applicazione mobile con integrato un portale di e-commerce proprietario, subiscono uno svantaggio competitivo, perdendo, di fatto, la nicchia di clienti che non possono attendere a casa l'arrivo del corriere.

Proprio per questo motivo realizzeremo il nostro sistema, che riuscirà ad integrarsi con i siti di e-commerce esistenti, favorendo, quindi, anche le piccole e medie imprese che possiedono un servizio di vendita online.

Dal punto di vista tecnico, quanto appena detto si traduce nella progettazione ed implementazione di un'interfaccia pubblica del back-end a cui le potenziali imprese, ovvero coloro che saranno interessate a fornire il servizio di consegna nello smart locker, possono collegare il proprio sito di e-commerce ed usufruirne. Ciò consisterà nella pubblicazione di un'API accessibile anche da software esterni.

In sostanza, realizzeremo il sistema complessivo dello smart locker, il quale è composto da tre importanti componenti, ovvero l'elettronica, il back-end ed il front-end. Ciascuna di queste

sezioni sarà creata a partire da una fase di progettazione, seguita, poi, dall'implementazione.

La prima parte è, quindi, dedicata alla costruzione di uno schema elettrico, che sarà grado di leggere segnali in input dai sensori e da un tastierino, e di controllare, in uscita, relay, motori servo ed un display LCD.

Inoltre, nostra premura sarà anche programmare il micro-controllore Arduino, che funge da *cervello* dell'intero sistema elettrico.

Successivamente affronteremo la sezione dedicata alla progettazione e sviluppo del back-end, che consisterà nell'implementazione di un'interfaccia pubblica, locata su un server di nostra proprietà e che rispetta tutti i necessari requisiti di sicurezza. Questa interfaccia verrà, poi, connessa alle altre 2 componenti.

Infine tratteremo la componente di front-end, che, dal punto di vista pratico, consiste in un'applicazione *cross-platform*, ovvero disponibile sia in versione mobile per iOS ed Android, sia fruibile in versione *web app*.

Tale app sarà, poi, usata dagli addetti ai lavori, i quali verranno preventivamente istruiti per gestire, in maniera completamente autonoma ed efficiente, le operazioni di inserimento dei prodotti e prelievo dello smart locker.

Un ulteriore ed importante focus avverrà sulle diverse vie di comunicazione che abbiamo implementato tra le tre componenti sopra citate, per capire come avverrà il coordinamento tra ogni fattore del sistema complessivo.

La presente tesi è composta da otto capitoli strutturati come di seguito specificato:

- Nel Capitolo 1 sarà introdotto il contesto del nostro sistema, e successivamente verranno definiti i termini che saranno utilizzati e gli obiettivi del progetto, analizzando la concorrenza e focalizzandoci sui problemi e le soluzioni che si vorranno proporre.

- Nel Capitolo 2 si descriverà l'architettura del sistema, denotando come esso sarà diviso in una componente elettrica, una componente di back-end ed una di front-end; quindi, si tratterà l'architettura del software e dell'hardware.

- Nel Capitolo 3 si svolgerà la fase di ingegneria dei requisiti, la cui analisi si baserà sulle informazioni ottenute mediante interviste ed etnografie, per ottenere la lista completa dei requisiti di progetto.

- Nel Capitolo 4 si tratterà la progettazione della componente hardware, e quindi dell'elettronica, e di come viene coordinata dalla componente di back-end.

- Nel Capitolo 5 si descriverà la fase della progettazione del software, ovvero dell'applicazione mobile, e di come anch'essa è coordinata alle altre due componenti.

- Nel Capitolo 6 sarà discussa l'implementazione delle tre componenti sopra citate, con un focus relativo alle connessioni tra di esse, ma anche ai problemi riscontrati ed alle soluzioni approntate.

- Nel Capitolo 7 proporremo una discussione riguardante il lavoro svolto, utilizzando la nota analisi SWOT, ed elencando le lezioni apprese.

- Nel Capitolo 8 si esporranno le conclusioni in merito al lavoro svolto e si darà uno sguardo anche ai possibili sviluppi futuri.

CAPITOLO 1

Descrizione del contesto di riferimento

In questo primo capitolo scopriremo, attraverso una panoramica generale, la situazione attuale della ferramenta analizzata, in modo tale da individuarne tutti i problemi e le metodologie che, grazie al software e all'hardware che progetteremo e produrremo, saranno semplificati e risolti.

Per poterci permettere di farlo, abbiamo compreso come prima cosa la situazione attuale, e in seguito abbiamo posto le basi per realizzare l'analisi dei requisiti, che verranno utilizzati nei capitoli seguenti, il tutto utilizzando sempre il linguaggio naturale.

1.1 Premessa

Il nostro compito è quello di progettare e realizzare un armadietto da esterno, con diversi sportelli che possono essere aperti sia da una chiave sia da un codice digitato su un apposito tastierino, per consentire ai clienti di prelevare i prodotti acquistati quando lo ritengono più comodo.

Inoltre ci riserviamo di creare un'applicazione per il responsabile della ferramenta, che permetta di generare, modificare ed eliminare ordini per l'armadietto e di controllare ogni apertura e chiusura passate.

1.2 Descrizione del contesto

Ad oggi, per i clienti che ordinano dei prodotti ma non riescono a passare entro gli orari di apertura, vengono lasciati i loro acquisti all'esterno, con la possibilità che questi vengano rubati o danneggiati dagli agenti atmosferici avversi.

Altri acquirenti potrebbero avere la necessità di comprare dei prodotti rapidamente, essendo di fretta, e il pensiero di dover attendere in coda potrebbe demoralizzarli.

Ancora, i falegnami molto spesso, essendo impegnati col lavoro, mandano un operaio manovale a fare rifornimento degli strumenti di lavoro e delle materie prime; questi manovali però, essendo meno esperti, possono confondere i prodotti e fare acquisti errati.

La nostra soluzione è quindi quella di posizionare all'esterno, ma sempre in uno spazio di proprietà della Ferramenta, e video-sorvegliato, uno smart locker, ovvero un armadietto in lamiera zinco-plastificata, con 6 sportelli di diverse dimensioni, e uno spazio appositamente riservato ad un tastierino e ad un display. Ogni sportello può essere aperto sia da una chiave, di cui un responsabile interno della ferramenta manterrà 2 copie, oppure automaticamente, attraverso l'inserimento di un codice nel tastierino, che scatena un meccanismo di apertura

motorizzata dello stesso. Tale codice, a 6 cifre, viene generato automaticamente nel momento in cui un cliente effettua un ordine e specifica di volerlo prelevare nell'armadietto; esso verrà spedito al cliente per e-mail ed SMS.

1.2.1 Realtà d'interesse

La ferramenta Berti, con sede a Castelfidardo, in provincia di Ancona, è da sempre specializzata nella vendita di ferramenta per mobili ed infissi. Si rivolge a piccoli artigiani e medie imprese che puntano su prodotti all'avanguardia e con buon rapporto qualità-prezzo. Essa dispone di un vasto assortimento di accessori per tende, porte, vernici, abrasivi, zanzariere, casseforti, accessori per serramenti. Essa effettua inoltre, il servizio di duplicazione chiavi.

Anche le esigenze dei privati vengono soddisfatte grazie alla realizzazione di un negozio elettronico e ad un efficiente servizio spedizioni in tutta Italia. La ferramenta Berti dispone di un negozio fisico, in cui i clienti, durante l'orario d'apertura, si riforniscono del materiale che necessitano.

1.2.2 Glossario

Il glossario è la prima cosa che abbiamo definito, nonostante sembri tutto apparentemente scontato, la terminologia è molto importante e a volte, durante la comunicazione, una discrepanza verbale rischia di tramutarsi in un errore di dimensioni maggiori in fase di sviluppo.

Definiamo il glossario della Tabella 1.1.

	Termine	**Significato**
1	Cliente	Persona che usufruisce del servizio, che ha effettuato un ordine, in cui ha specificato di voler prelevare i prodotti nel locker.
2	Responsabile	Personale della ferramenta; colui che mantiene una doppia copia di ogni chiave ed è responsabile, in una prima fase, dell'utilizzo dell'app.
3	App	Applicazione desktop e mobile, riservata al Responsabile, che permette di gestire e monitorare gli ordini.
4	Locker	Armadietto, nel suo completo.
5	Cabinato	Armadietto esanime, dal punto di vista della struttura.
6	Locker Door	Singolo sportello dell'armadietto.
7	Ordine	Informazioni riguardanti un ordine di prodotti al quale vengono associate un sottoinsieme di label, un codice di sblocco, e del personale addetto
8	Label	Etichetta che distingue una locker door da un'altra.
9	Opening	Informazioni riguardo l'apertura o chiusura di una locker door.

Tabella 1.1: Glossario relativo alla realtà d'interesse

1.2.3 Vincoli

Come sempre, non si ha illimitata libertà progettuale, ma, una volta conosciuta la realtà gestionale dell'azienda, abbiamo capito che c'erano alcuni vincoli da definire ancor prima di iniziare il progetto vero e proprio. Questi vincoli possono essere di qualsiasi tipologia; nel nostro caso, sono di diversa natura.

Essi vincoli sono presentati nella Tabella 1.2.

	Vincoli
1	Grafica dell'app semplice e intuitiva, con colori rosso, blu e bianco.
2	Almeno una locker door di altezza interna di 2.40 m, per le cornici.
3	Posizionamento del locker all'esterno della ferramenta, in un luogo ripreso dalla videosorveglianza, e vicino ad una presa della corrente.
4	Protezione delle componenti elettroniche dagli agenti atmosferici avversi.
5	Scrittura su ogni locker door, della corrispettiva label.
6	Stampa di un QR code sul fianco del locker, che, se inquadrato dalla fotocamera di un cellulare, porta al sito `https://www.bertiferramenta.it` .
7	Rialzo con piedini del locker, per proteggere gli sportelli più bassi dalla pioggia.
8	Visibilità notturna, sia del locker stesso, sia del display, che sarà retroilluminato, sia dei tasti del tastierino.
9	L'elettronica e i cavi utilizzati devono essere il meno visibili possibile per il cliente, che in ogni modo non deve poterli facilmente manomettere.

Tabella 1.2: Vincoli relativi alla realtà d'interesse

1.3 Analisi della concorrenza

Per non partire da zero, con la genuina curiosità di un ingegnere che, di fronte ad un nuovo strumento, si chiede sempre "com'è fatto?" e "come funziona?", abbiamo analizzato altre aziende, molto più grandi, che realizzano smart locker da anni, e che abbiamo constatato essere affidabili e duraturi nel tempo. Sicuramente non possiamo non considerare i sempre più popolari Amazon Locker che, però, sono utilizzabili solo per gli ordini effettuatitramite il corrispettivo sito di e-commerce.

Inoltre, approfondendo la ricerca, abbiamo esplorato il mondo, anch'esso in via di sviluppo, degli smart locker da appartamento, e, infine, quelli che somigliano più di tutti al nostro caso, ovvero gli armadietti intelligenti da azienda, e ne abbiamo compreso le caratteristiche che li rendono popolari ed efficienti.

In particolare, abbiamo constatato che la facilità e comodità d'uso sono fondamentali; altre caratteristiche fondamentali sono una reliability decisamente alta ed ad un largo uso delle più recenti tecnologie IoT.

1.4 Definizione di problemi e soluzioni

Date le nostre capacità, abbiamo elencato i problemi che il nostro smart locker può risolvere e, per ciascuno di essi, abbiamo spiegato in che modo lo stesso possa diventare, invece, un punto di forza dell'azienda che decide di usare l'armadietto.

In particolare, i problemi identificati sono i seguenti:

- I potenziali clienti che non riescono a passare in ferramenta entro l'orario di chiusura possono acquistare altrove.
- I clienti che acquistano ma non possono passare in ferramenta entro l'orario di chiusura, possono trovare il pacco danneggiato, o possono non trovarlo, lamentandosi e lasciando recensioni negative.
- I potenziali clienti che sono spinti dalla fretta possono decidere di acquistare in altri negozi ciò di cui necessitano, per evitare di rimanere in coda per qualche minuto.
- I falegnami occupati spesso delegano agli operai manovali l'acquisto di strumenti e materie prime rischiando, però, acquisti errati per la minore esperienza di questi ultimi.

La nostra soluzione consiste in un armadietto dove i dipendenti possano lasciare i prodotti ordinati, al sicuro a riparo da malviventi e intemperie; tali prodotti possono essere ritirati dai clienti comodamente. I clienti esterni, possono, inoltre, acquistare un prodotto e delegare il prelievo a qualcun altro senza preoccupazioni per possibili acquisti errati.

Tale smart locker ha anche la particolare caratteristica di poter collaborare, come vedremo in seguito, con i sistemi già in funzione nella ferramenta Berti.

CAPITOLO 2

Architettura hardware e software del sistema

In questo capitolo daremo uno sguardo all'architettura del sistema realizzato. In particolare divideremo il capitolo nell'analisi delle componenti hardware e delle componenti software.

Nella prima sezione, osserveremo il modello e l'interazione con l'utente.

Nella seconda sezione, invece, analizzeremo i pattern architetturali utilizzati per il server e per l'app, introducendo, innanzitutto, la motivazione fondamentale per la quale si utilizzano i pattern.

In seguito, approfondiremo il pattern MVC, che ha contribuito alla realizzazione della struttura del nostro sistema, spiegando il motivo che ci ha spinto ad usarlo. Infine, vedremo in che modo tale pattern è stato implementato, quali modifiche, seppur leggere, ha subito, e cosa ci ha spinto ad effettuarle.

2.1 Architettura generale del sistema complessivo

Il sistema complessivo che stiamo progettando è strutturato in tre componenti principali, ovvero:

- un server;
- un dispositivo;
- un'applicazione mobile.

Questi elementi lavorano in sinergia per garantire l'efficacia e l'efficienza del sistema.

Il server rappresenta il cuore del sistema, in quanto è responsabile della gestione delle connessioni, dell'elaborazione dei dati e della comunicazione sia con il dispositivo Arduino che con l'app.

Il server è ospitato su una piattaforma cloud che, nel nostro caso, corrisponde a Firebase, permettendo, così, scalabilità e accessibilità. È, inoltre, incaricato della raccolta e dell'analisi dei dati inviati dal dispositivo Arduino, fornendo funzionalità di back-end come il database per lo storage e strumenti analitici per l'elaborazione dei dati reali.

Lo smart locker agisce come nodo sensore nel sistema IoT. Equipaggiato con diversi sensori, il dispositivo è in grado di intercettare informazioni dall'ambiente che lo circonda.

Questi dati vengono quindi trasmessi al server attraverso una connessione Internet, che può essere realizzata via Wi-Fi o tramite protocolli di comunicazione IoT, come l'HTTP.

Il dispositivo è programmato per operare autonomamente, ma può ricevere aggiornamenti e comandi dal server per adeguarsi a nuove configurazioni o parametri operativi.

L'applicazione mobile cross-platform offre un'interfaccia utente per interagire con il sistema. Attraverso l'app, gli amministratori possono visualizzare in tempo reale i dati necessari.

L'app può anche inviare comandi specifici al server che, a sua volta, li inoltra allo smart locker. Ciò permette un controllo diretto e personalizzato del comportamento del dispositivo IoT, aumentando l'interattività per l'azienda.

La struttura generale appena descritta è rappresentata schematicamente nella Figura 2.1.

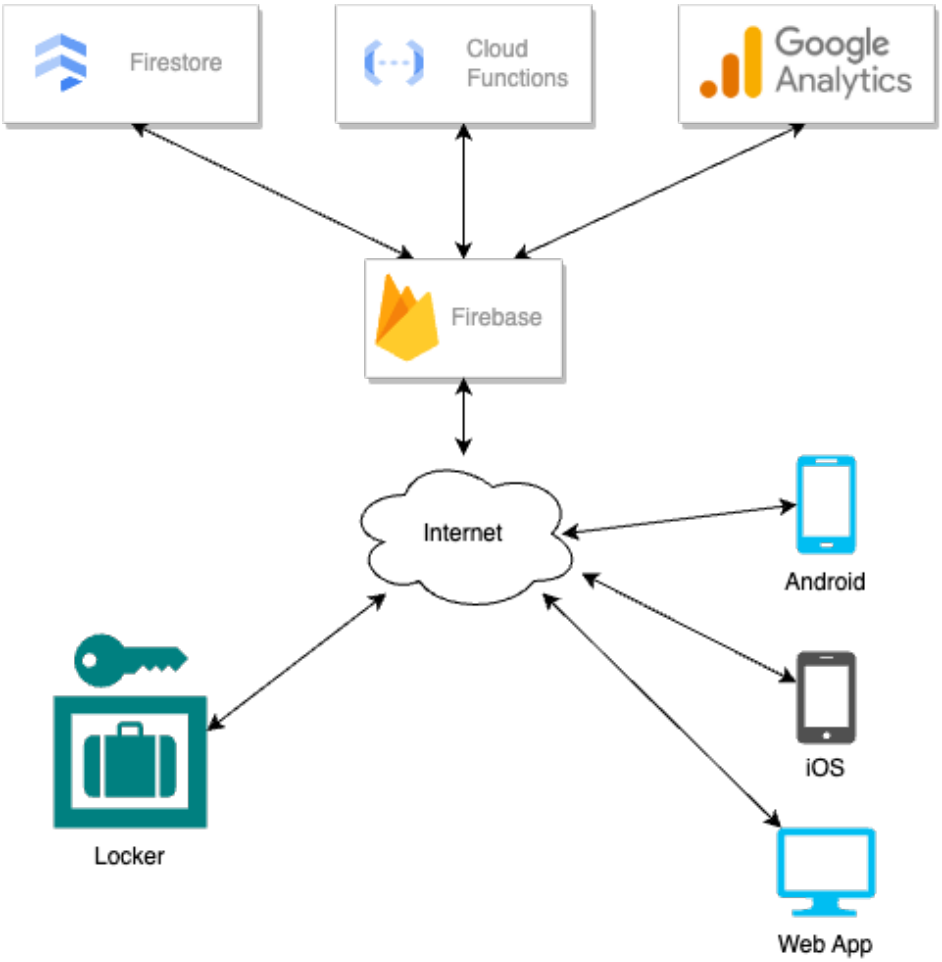


Figura 2.1: Diagramma dell'architettura generale

2.2 Architettura hardware del sistema

L'architettura del sistema hardware è stata realizzata con uno schema detto "a stella", in cui l'unità centrale di controllo basata su Arduino funge da nodo centrale, o "centro stella".

Considerando, tuttavia, la limitazione dei pin disponibili sull'Arduino scelto per gestire un numero elevato di periferiche, il sistema sfrutta il protocollo di comunicazione I2C per estendere la capacità di connessione. Questa soluzione include l'uso di espansori di I/O, come il PCF8574, e switch/multiplexer I2C, come il TCA9548A, per collegare e gestire le periferiche.

Questo modello è molto comune per i casi di questa tipologia, e viene scelto per la semplicità di gestione e di controllo centralizzati di tutte le periferiche e delle componenti del sistema.

2.2.1 Descrizione del modello hardware

Nel modello a stella, il centro della stella è il fulcro del sistema, mediante il quale passano tutte le comunicazioni attraverso il bus. Nel nostro progetto, tale importante ruolo è affidato ad Arduino. Questo microcontrollore non solo gestisce direttamente le comunicazioni con le periferiche di input e output, ma serve anche come ponte di connessione tra queste e il server centrale.

Il centro della stella ha, però, vari compiti che spesso risultano essere critici, essi sono:

- *gestione dei dati*: ottiene i dati dai sensori, che corrispondono alle periferiche di input, li elabora e determina le azioni da svolgere, quindi controlla gli attuatori, ovvero le periferiche di output;
- *comunicazione con il server centrale*: ogni periodo determinato di tempo, scarica i dati necessari dal server, e invia a quest'ultimo gli aggiornamenti sui dati di cui ha bisogno;
- *controllo delle operazioni*: coordina le periferiche di input e output in modo che il dispositivo continui ad operare in maniera più fluida ed efficiente possibile.

D'altro canto l'architettura a stella possiede numerosi vantaggi, che motivano la nostra scelta, tra i quali la semplificazione della gestione della rete, la facilità di aggiornamento e la centralizzazione del controllo.

Per quanto riguarda il modello software del controllore, è stato usato il template tipico di Arduino, per cui sono presenti due funzioni generiche, di cui una che viene eseguita solo all'avvio, per le operazioni di setup; e l'altra che viene eseguita ciclicamente, ad intervalli predeterminati di tempo.

L'approccio proposto non solo risolve il problema del numero di pin limitati su Arduino ma offre anche una struttura estremamente modulare e flessibile per l'espansione futura.

Utilizzando il protocollo I2C e componenti come il PCF8574 e il TCA9548A, il sistema può gestire una vasta gamma di periferiche in modo efficiente e affidabile, rendendolo ideale per complessi progetti IoT.

2.2.2 Descrizione dell'interazione con l'utente

L'utente si interfaccia all'hardware del sistema mediante due vie:

- *input*: tastierino numerico;
- *output*: display LCD.

Per quanto riguarda il tastierino numerico, esso viene usato per digitare il codice di sblocco dello sportello, ricevuto per email e SMS. Per facilitare l'utente sono previsti anche dei tasti per cancellazione delle cifre digitate.

L'utente riceve informazioni tramite un display LCD di medie dimensioni, posizionato poco sotto l'altezza degli occhi, che fornisce costantemente informazioni, in maniera riassuntiva, sullo stato del locker, nonché le istruzioni all'utente per utilizzare il dispositivo nella maniera corretta.

2.3 Architettura software del sistema

Nel nostro progetto è fondamentale riconoscere che il sistema non è governato da un unico software, ma da due componenti software distinti per il funzionamento complessivo, ovvero:

- il server;
- l'applicazione mobile.

Questa distinzione sarà applicata in tutte le future descrizioni e documentazioni del progetto per garantire chiarezza nella comprensione delle diverse funzioni e responsabilità di ciascun componente software.

Le due componenti, seppur comunicanti, sono progettate con tecniche diverse, e tipiche per il sistema che rappresentano.

2.3.1 Descrizione generale dei pattern

I design pattern rappresentano soluzioni già pronte, che permettono di realizzare il principio del riuso, uno dei principi base dell'ingegneria del software non solo a livello di codice (librerie di classi) ma anche a livello di progettazione, infatti essi ci forniscono dei pattern, cioè degli schemi vuoti, che possono essere adattati allo schema delle classi che vogliamo realizzare.

Oggigiorno, i pattern esistono per qualsiasi livello di progettazione e programmazione, infatti, abbiamo per esempio:

- *I pattern creazionali*: astraggono il processo di istanziazione. Più nel dettaglio, essi consentono di rendere il sistema indipendente da come gli oggetti sono creati e rappresentati nonché dalle relazioni di composizione tra essi.
- *I pattern strutturali*: si occupano di come comporre le classi e gli oggetti per formare delle strutture più complesse. Abbiamo sia pattern basati su classi che basati su oggetti. La differenza sta nel fatto che, in quelli basati su classi, le interfacce e le implementazioni vengono composte mediante l'ereditarietà. I pattern basati su oggetti descrivono la modalità secondo cui comporre gli oggetti per realizzare nuove funzionalità; ovviamente, in questo modo, è possibile fornire maggiore flessibilità.
- *I pattern comportamentali*: attribuiscono le responsabilità ad oggetti che comunicano tra loro, a runtime. Si usano quando si hanno flussi di informazioni difficili da seguire, e quando vogliamo studiare le relazioni dinamiche tra oggetti. Quei pochi pattern basati su classi utilizzano l'ereditarietà, mentre quelli basati su oggetti utilizzano la composizione. Con l'uso dei pattern si riesce a comunicare, modificare e quant'altro, mantenendo molto basso il livello di accoppiamento.

Infine, ci sono i pattern architetturali, a cui appartiene il pattern MVC di cui parleremo in seguito.

Questi pattern si collocano ad un livello più in alto rispetto a quelli visti precedentemente; in particolare descrivono lo schema organizzativo della struttura che caratterizza un sistema software.

In genere, individuano le parti del sistema a cui sono associate responsabilità omogenee e le relazioni che esistono tra i diversi sottosistemi.

2.3.2 Architettura del server

L'architettura del server nel nostro progetto è pensata principalmente per essere robusta, modulare e, soprattutto, facilmente scalabile ed estendibile a nuove funzioni, incorporando componenti essenziali, quali un indice delle API, dei controlles, dei model, e connessioni a un database, un sistema di gestione delle autenticazioni e un sistema di storage.

L'architettura descritta viene presentata nella Figura 2.2.

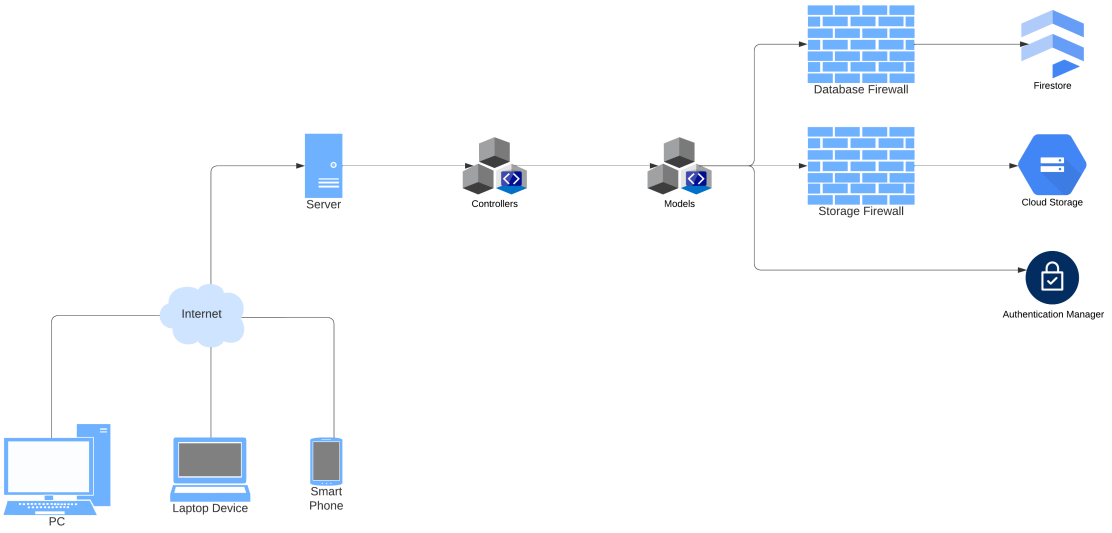


Figura 2.2: Schema dell'architettura del server

Descrizione dei pattern utilizzati

Come pattern architetturale per il software del server è stato scelto il noto e ormai consolidato modello MVC, che viene adattato allo sviluppo del back-end secondo quanto descritto nella prossima sezione.

MVC sta per Model-View-Controller, e consiste in un framework architetturale che distingue le applicazioni in tre componenti logici: Model, View e Controller. Ciascuno di questi componenti è progettato per gestire responsabilità specifiche all'interno dell'applicazione.

Nello specifico, il nostro pattern separa il livello di presentazione dalla logica di business. Ciò garantisce che lo sviluppo del modulo di presentazione possa essere eseguito indipendentemente da quello della logica di business.

In un contesto di programmazione moderno, in cui l'elaborazione parallela sta diventando sempre più prevalente per ottimizzare l'uso delle risorse e migliorare l'efficienza, l'approccio MVC risponde efficacemente a queste esigenze.

Tale approccio facilita decisamente la modularità e riduce le dipendenze tra i diversi moduli dell'applicazione, consentendo, così, uno sviluppo più agile e meno suscettibile agli errori dovuti all'integrazione di diverse funzionalità.

Model

La componente Model del pattern MVC in un server gioca un ruolo cruciale, essendo responsabile della gestione dei dati e della logica di business.

Nel contesto di un server, per quanto riguarda l'implementazione, ogni model corrisponde ad una classe che implementa i metodi necessari al funzionamento, soprattutto per quanto riguarda il collegamento alle altre componenti, quali database, storage a gestore dell'autenticazione.

Controller

La componente Controller serve come intermediario tra il Model, che gestisce i dati e la logica di business, e l'utente finale, che interagisce direttamente con l'interfaccia grafica. Questi Controller sono fondamentali per elaborare le richieste in entrata, manipolare i dati e restituire le risposte appropriate.

I Controller ricevono le richieste in entrata dalle API, estrapolano i parametri necessari e decidono quale azione intraprendere. Sono i primi a processare l'input dell'utente, ma anche del dispositivo IoT, e ad invocare le funzionalità indicate dal sistema.

Una volta ricevuta una richiesta, il Controller si interfaccia con il Model per richiedere i dati o per effettuare operazioni che riflettono la logica di business. Esso non esegue queste operazioni direttamente, ma si serve del Model per assicurarsi che le interazioni con il database siano gestite correttamente e che le regole di business siano applicate a dovere.

Il Model risponderà alla richiesta con delle informazioni, e di conseguenza il Controller può elaborare ulteriormente questi dati prima di inviarli all'utente o ad altri sistemi. Questo processo può includere la formattazione dei dati per adattarli ai requisiti specifici del client che ha fatto la richiesta.

Infine, il Controller crea il pacchetto della risposta finale che sarà inviata all'utente o al sistema che ha fatto la richiesta. Questa risposta può essere un semplice successo o fallimento, un messaggio di errore, o un set complesso di dati, a seconda del risultato delle operazioni effettuate.

Contestualmente al server, così come i Model, anche i Controller vengono implementati come classi i cui metodi svolgono le operazioni sopra elencate.

2.3.3 Architettura dell'app

Anche in questo caso si prende in prestito il modello MVC già visto nelle pagine precedenti, in quanto è largamente usato e consolidato nello sviluppo delle applicazioni per smartphone.

La filosofia è molto simile all'architettura del server descritta precedentemente; tuttavia, in questo caso, le due componenti assumono un ruolo diverso, e viene introdotta la View.

L'architettura dell'app appena descritta viene presentata nella Figura 2.3.

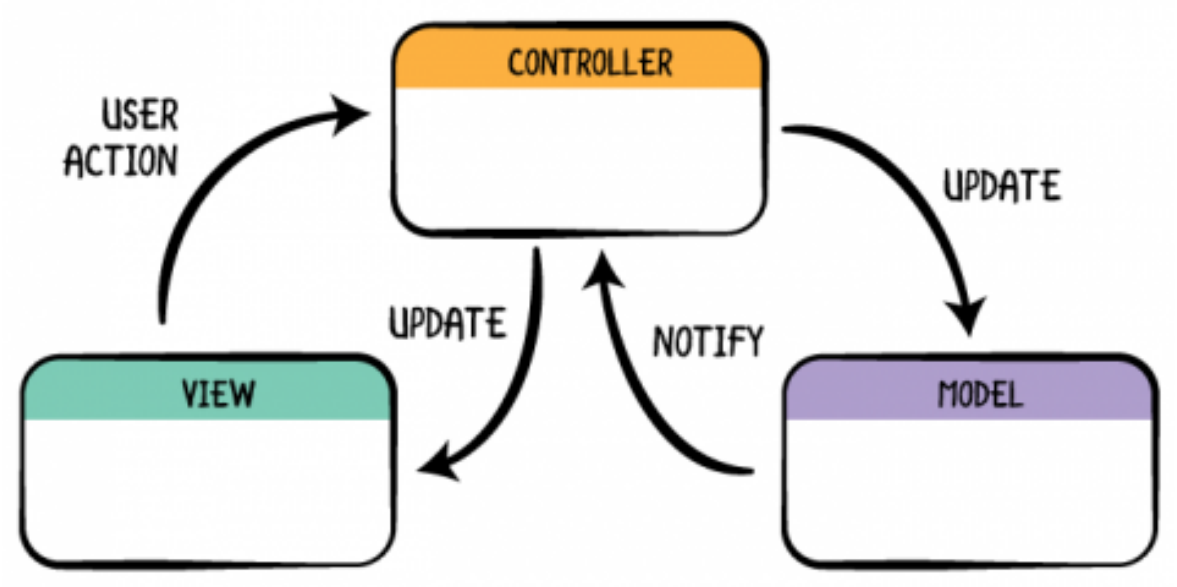


Figura 2.3: Schema dell'architettura dell'app

Model

Il Model è leggermente differente; infatti nell'ambito mobile serve per ottenere i dati dal Controller, e aggiorna i dati richiamando le API del server descritte precedentemente, dalle

quali preleva le informazioni che reindirizza al Controller dell'app.

Controller

Anche in questo caso il Controller, agisce come intermediario tra il Model e la View. La differenza principale consiste nel fatto che riceve gli input dall'utente tramite la View, li elabora utilizzando il Model e poi aggiorna la View con i nuovi dati necessari.

View

La View è la componente che gestisce l'output visivo e l'interazione con l'utente. Si occupa di presentare i dati all'utente e di raccogliere i suoi input. Nel contesto dello sviluppo mobile, le View sono tipicamente rappresentate da schermate e componenti UI (User Interface) come pulsanti, textbox, liste, etc. Le View inviano le azioni dell'utente al Controller, ma non elaborano i dati, garantendo, così, che la presentazione dei dati sia separata dalla logica di business.

CAPITOLO 3

Analisi dei requisiti

Prima di iniziare a sviluppare un applicativo di qualsiasi tipo, sappiamo che è una buona norma capire, e successivamente mettere per iscritto, quali sono, nello specifico, i requisiti che il committente desidera. Negli ultimi 50 anni questa fase è diventata, giustamente, sempre più importante e più dispendiosa, perché, come avviene in moltissimi ambiti, avere chiare le disposizioni fin dall'inizio evita, o comunque riduce notevolmente, gli errori durante lo sviluppo, che, se esistenti, causano una reazione a catena. Dunque, è nata una vera a propria disciplina, l'ingegneria dei requisiti, in cui gli ingegneri si occupano proprio di ottenere i requisiti in linguaggio naturale e poi formalizzarli, in modo tale da renderli leggibili e comprensibili anche dagli sviluppatori.

3.1 Ingegneria dei requisiti

L'Ingegneria dei requisiti si occupa dell'analisi dei requisiti, che, in linea teorica, dovrebbero essere quanto più dettagliati possibile, per far sì che i progettisti capiscano esattamente come fare il proprio lavoro, anche se nelle piccole realtà, come quella oggetto della nostra tesi, queste figure coincidono.

Dunque il nostro lavoro di ingegneria dei requisiti consiste dapprima nello scrivere in linguaggio naturale i requisiti chiesti dal committente, e successivamente realizzare il diagramma dei casi d'uso, che serve a formalizzare il complesso.

3.2 Analisi dei requisiti

Per ottenere i requisiti necessari, abbiamo utilizzato le pratiche dell'intervista aperta e dell'etnografia, che approfondiremo in questa sezione.

Una volta ottenuti tali requisiti, è stata nostra premura analizzarli specificamente, mediante uno studio di fattibilità in primis, e poi con uno studio di realizzazione.

Nel nostro caso lo studio di fattibilità è stato relativamente complicato, perché sia le richieste, sia ciò che abbiamo creativamente aggiunto, sono, ad oggi, recenti e moderne, e quindi abbiamo avuto poche occasioni per prendere spunto dalle tecnologie esistenti.

Per quanto riguarda lo studio di realizzazione, ci siamo affidati alle nostre conoscenze; quindi, essendo ormai il nostro lavoro, ci siamo mossi in maniera molto sicura decidendo in che modo ogni requisito andasse soddisfatto.

3.2.1 Intervista

Conoscendo molto bene i titolari della Ferramenta, abbiamo avuto diversi modi di confrontarci con loro che, molto gentilmente, sono stati sempre a nostra disposizione per la prima progettazione e per ogni dubbio successivo.

Per interfacciarci con gli amministratori, abbiamo utilizzato entrambe le tecniche che abbiamo studiato, cioè l'intervista, di cui in particolare l'intervista aperta, e l'etnografia.

Intervista Aperta

Come prima cosa, non essendo molto esperti nel mondo della ferramenta, abbiamo dovuto porre domande "banali" per comprendere il working set in cui ci saremmo trovati; quindi, inizialmente, l'intervista consisteva di una spiegazione, da parte degli amministratori, del loro lavoro e di ciò che pensavano si potesse semplificare.

Una volta compreso l'ambito lavorativo, abbiamo iniziato a porre domande sempre più specifiche sul metodo che utilizzavano fino a quel momento per la vendita dei prodotti e la gestione di ordini e utenti, che sarebbe, poi, diventato ciò di cui ci saremmo occupati.

3.2.2 Etnografia

Successivamente siamo stati gentilmente ospitati negli uffici amministrativi, dove abbiamo assistito al procedimento di gestione degli ordini per i clienti che non possono passare a ritirare i propri prodotti durante il normale orario di lavoro.

Questo procedimento era molto informale, e consisteva nel lasciare un pacco nei pressi del cancello, con il rischio di furti e danneggiamento da intemperie. Con le attuali tecnologie, abbiamo capito che questo aspetto poteva essere nettamente migliorato, creando un dispositivo che risolvesse questo problema.

3.2.3 Componenti dei requisiti

Entriamo, ora, finalmente nel dettaglio dei requisiti, analizzando uno ad uno i nostri componenti che entreranno in gioco.

Iniziamo dagli attori, cioè gli utenti attivi del nostro sistema software. Nel nostro caso gli attori sono di due tipologie, ovvero il responsabile della ferramenta, che utilizza l'applicazione mobile, e il cliente, che usufruisce del locker.

I casi d'uso, invece, sono tutte le attività che le diverse tipologie di utenti possono svolgere, e corrispondono esattamente ai dodici requisiti funzionali che vedremo nella prossima sezione.

3.2.4 Lista dei requisiti

Riportiamo la lista dei requisiti realizzata durante il processo di ingegneria dei requisiti. Differenziamo tra requisiti funzionali e non funzionali.

Requisiti funzionali

I requisiti funzionali definiscono le funzionalità fornite da un sistema. Nel complesso, essi definiscono il comportamento di quest'ultimo. I requisiti funzionali da noi individuati sono i seguenti:

- RF1: Il responsabile visualizza gli sportelli liberi.

- RF2: Il responsabile crea un ordine.
- RF3: Il responsabile aggiorna un ordine.
- RF4: Il responsabile elimina un ordine.
- RF5: Il responsabile visualizza gli ordini.
- RF6: Il responsabile visualizza le aperture e le chiusure.
- RF7: Il responsabile visualizza gli ordini passati.
- RF8: Il responsabile carica i prodotti.
- RF9: Il cliente visualizza le istruzioni.
- RF10: Il cliente digita il codice.
- RF11: Il cliente preleva i prodotti.
- RF12: Il cliente chiude lo sportello.

Requisiti non funzionali

I requisiti non funzionali descrivono gli aspetti del sistema che non sono direttamente legati al suo comportamento. Essi rappresentano piuttosto dei vincoli che caratterizzano il sistema.

I requisiti non funzionali relativi al nostro sistema sono i seguenti:

- RNF1: Il sistema utilizza Firebase come DBMS.
- RNF2: L'applicazione mobile deve essere realizzata con i colori blu, rosso e bianco.
- RNF3: Il cabinato deve essere alto almeno 2.50 metri.
- RNF4: Il cabinato deve essere zinco-plastificato, garantire una protezione dalla ruggine per 10 anni, ed essere posizionato all'esterno.

3.2.5 Descrizione dei requisiti

Per ogni requisito funzionale o non funzionale individuato, tracceremo una descrizione testuale, che sarà molto utile per progettare e sviluppare il flusso delle operazioni previste dal suddetto requisito.

Tale descrizione, seppur informale, ci dà una chiara idea sul funzionamento di ogni operazione che l'attore può svolgere, oltre che descrivere come verranno presentate all'utente le eventuali eccezioni.

Requisiti funzionali

- RF1: Il responsabile visualizza gli sportelli liberi.

  Il responsabile, aprendo l'applicazione mobile, visualizza nella home page una rappresentazione stilizzata dell'armadietto fisico, in cui vengono separati graficamente i vari sportelli.

  Ogni sportello raffigurato presenta un numero identificativo, chiamato comunemente label, o etichetta, e le dimensioni dello sportello stesso. Per quanto riguarda queste

ultime, vengono mostrate solo, in ordine, l'altezza e la larghezza, in quanto sono le uniche che variano tra uno sportello ed un altro, dato che la profondità è uguale per tutti gli sportelli dell'armadietto.

Ogni riquadro raffigurante uno sportello possiede un colore di sfondo, che può essere, alternativamente, verde o rosso, dove il verde indica che tale sportello è libero e pronto per essere utilizzato, mentre il colore rosso indica che il relativo sportello è occupato dall'ordine di un cliente.

I dati da cui si ricavano le informazioni vengono aggiornati ogni volta che la home page viene caricata, o, in alternativa, quando il responsabile preme il pulsante di refresh situato in alto a sinistra. Tale operazione prevede il caricamento delle informazioni, durante il quale compare al centro dello schermo un indicatore dell'attesa, che viene sostituito dal contenuto non appena quest'ultimo è stato ottenuto con successo.

Nella Tabella 3.1 viene riportata una scheda del requisito.

Nome	**RF1. Il responsabile visualizza gli sportelli liberi**
Attori	Responsabile
Flusso	1. Il responsabile apre l'app. 2. Vengono scaricate le informazioni dal server. 3. Il responsabile visualizza la raffigurazione dell'armadietto.
Output	Schermata dell'app dove viene raffigurato l'armadietto, in cui lo sfondo di ogni sportello viene colorato di verde, se libero, o di rosso, se occupato.
Eccezioni	1. Il server non risponde. Dopo il tempo di timeout (30 secondi) viene mostrato un messaggio di errore al responsabile.

Tabella 3.1: Scheda del requisito RF1

- RF2: Il responsabile crea un ordine.

  Il responsabile, una volta ricevuto un ordine per cui è richiesto l'uso del locker, che sia telefonicamente, per email o tramite un messaggio, può creare l'ordine sull'applicazione.

  Il primo passo consiste nell'aprire l'app, e, una volta caricata la home page, premere il pulsante in fondo a destra, meglio noto come Floating Action Button, tipico dello stile Google chiamato Material, che, come icona, presenta il simbolo dell'addizione.

  Il responsabile apre, così, una seconda schermata, in cui si presenta una checklist che elenca i 6 sportelli, e le relative dimensioni. Il nostro attore deve selezionare lo sportello, o gli sportelli, in cui inserirà fisicamente i prodotti appartenenti all'ordine.

  L'applicazione, automaticamente, impedisce al responsabile di selezionare uno sportello già occupato, sia mostrando il testo in rosso, sia rendendo disabilitata la relativa checkbox.

  Dopodiché il responsabile deve selezionare il proprio nome, o quello di un delegato che si occupa effettivamente di preparare l'ordine, tra una lista di nomi prevista dal software.

  A questo punto compare un pulsante per aggiungere l'ordine al sistema, che, se premuto, mostra al responsabile un popup di successo. Tale popup contiene anche il

codice numerico a sei cifre da comunicare all'utente, e, mediante un nuovo pulsante appartenente al popup, il responsabile può inviare direttamente questo codice al cliente.

Per compiere tale operazione, il responsabile viene reindirizzato su una nuova pagina, in cui può inserire il nome del cliente, insieme alla sua email od al suo numero di telefono, al quale o ai quali verrà inviato un messaggio preimpostato contenete il codice da usare fisicamente e le istruzioni.

Nella Tabella 3.2 viene riportata una scheda del requisito.

Nome	**RF2. Il responsabile crea un ordine**
Attori	Responsabile
Flusso	1. Il responsabile si trova nella home page. 2. Il responsabile apre la schermata del nuovo ordine. 3. Il responsabile seleziona lo sportello, o gli sportelli, e il proprio nome. 4. Il sistema crea l'ordine e comunica il codice al responsabile 5. Il responsabile apre la schermata di invio del codice. 6. Il responsabile inserisce i dati del cliente. 7. Il responsabile invia il codice e le istruzioni al cliente.
Output	Popup con conferma di creazione dell'ordine e con il codice.
Eccezioni	1. Il server non risponde. Dopo il tempo di timeout (30 secondi) viene mostrato un messaggio di errore al responsabile.

Tabella 3.2: Scheda del requisito RF2

- RF3: Il responsabile aggiorna un ordine.

  Il responsabile può anche aggiornare un ordine che è già stato creato, e ciò può avvenire quando cambia il delegato oppure ci si accorge solo in seguito che gli sportelli selezionati in fase di creazione dell'ordine non sono sufficienti per contenere i prodotti destinati al cliente.

  Nella home page, il responsabile ha un pulsante, che, se premuto, reindirizza ad una nuova schermata contenente la lista di tutti gli ordini che il sistema ha registrato e che sono, in quel momento, attivi. Tale lista è riordinata dall'ordine più recente al meno recente; di ogni ordine è visibile il codice di rilevanza, seguito dalla data di creazione dell'ordine, del nome del responsabile e dell'elenco degli sportelli di cui prevede l'uso.

  Il responsabile clicca sull'ordine da modificare, quindi, si apre una schermata identica a quella di creazione di un nuovo ordine, con la differenza che essa è già compilata con i dati inseriti in fase di creazione.

  A questo punto il responsabile effettua le dovute modifiche e, se preme sul pulsante "Aggiorna" in fondo alla pagina, comunica al sistema l'aggiornamento.

  Di conseguenza, il sistema mostra un popup con l'informazione del cambiamento avvenuto con successo, e poi reindirizza il responsabile alla pagina precedente.

  Nella Tabella 3.3 viene riportata una scheda del requisito.

- RF4: Il responsabile elimina un ordine.

  Il responsabile può eliminare un ordine su richiesta del cliente.

Nome	**RF3. Il responsabile aggiorna un ordine**
Attori	Responsabile
Flusso	1. Il responsabile si trova nella home page. 2. Il responsabile apre la schermata della lista di tutti gli ordini attivi. 3. Il responsabile seleziona l'ordine che intende modificare. 4. Il responsabile effettua le dovute modifiche. 5. Il sistema comunica l'avvenuto cambiamento, e reindirizza alla pagina precedente.
Output	Popup con conferma di aggiornamento dell'ordine.
Eccezioni	1. Il server non risponde. Dopo il tempo di timeout (30 secondi) viene mostrato un messaggio di errore al responsabile.

Tabella 3.3: Scheda del requisito RF3

Nella home page, il responsabile ha un pulsante, che, se premuto, reindirizza ad una nuova schermata contenente la lista di tutti gli ordini che il sistema ha registrato e che sono, in quel momento, attivi. Tale lista è riordinata dall'ordine più recente al meno recente; di ogni ordine è visibile il codice di rilevanza, seguito dalla data di creazione dell'ordine, del nome del responsabile e dell'elenco degli sportelli di cui prevede l'uso.

Il responsabile clicca sull'ordine da eliminare, quindi, si apre una schermata identica a quella di creazione di un nuovo ordine, con la differenza che essa è già compilata con i dati inseriti in fase di creazione.

A questo punto il responsabile preme sul pulsante "Elimina" in fondo alla pagina, a seguito di ciò appare un popup che chiede conferma dell'eliminazione al responsabile, per evitare errori.

Se quest'ultimo conferma nuovamente, viene comunicata l'eliminazione al server, che è irreversibile. A tal punto il sistema risponde mostrando un popup con l'informazione dell'avvenuta eliminazione, e poi reindirizza il responsabile alla pagina precedente, in cui non sarà più visibile l'ordine appena eliminato.

Nella Tabella 3.4 viene riportata una scheda del requisito.

- RF5: Il responsabile visualizza gli ordini.

  Nella home page, il responsabile ha un pulsante, che, se premuto, reindirizza ad una nuova schermata contenente la lista di tutti gli ordini che il sistema ha registrato e che sono, in quel momento, attivi.

  Un ordine è definito attivo dal momento della sua creazione fino a quando il cliente non ritira i prodotti ordinati e chiude lo sportello, o gli sportelli, in cui essi si trovavano.

  Questa lista è riordinata dall'ordine più recente al meno recente; di ogni ordine è visibile il codice il rilevanza, seguito dalla data di creazione dell'ordine, del nome del responsabile e dell'elenco degli sportelli di cui prevede l'uso.

  Nella Tabella 3.5 viene riportata una scheda del requisito.

- RF6: Il responsabile visualizza le aperture e le chiusure.

Nome	**RF4. Il responsabile elimina un ordine**
Attori	Responsabile
Flusso	1. Il responsabile si trova nella home page. 2. Il responsabile apre la schermata della lista di tutti gli ordini attivi. 3. Il responsabile seleziona l'ordine che intende eliminare. 4. Il responsabile effettua l'eliminazione, confermandola. 5. Il sistema comunica l'avvenuta rimozione, e reindirizza alla pagina precedente.
Output	Popup con conferma di eliminazione dell'ordine.
Eccezioni	1. Il server non risponde. Dopo il tempo di timeout (30 secondi) viene mostrato un messaggio di errore al responsabile.

Tabella 3.4: Scheda del requisito RF4

Nome	**RF5. Il responsabile visualizza gli ordini**
Attori	Responsabile
Flusso	1. Il responsabile si trova nella home page. 2. Il responsabile preme sul pulsante per visualizzare gli ordini. 3. Il responsabile visualizza la schermata della lista di tutti gli ordini attivi.
Output	Schermata dell'app contenente la lista degli ordini attivi.
Eccezioni	1. Il server non risponde. Dopo il tempo di timeout (30 secondi) viene mostrato un messaggio di errore al responsabile.

Tabella 3.5: Scheda del requisito RF5

Il reponsabile, nella home page, vede una raffigurazione dell'armadietto, suddiviso per sportelli.

Selezionando uno sportello si apre una nuova schermata contenente la lista completa di ogni apertura e di ogni chiusura registrate, che coinvolgono tale sportello. La lista è ordinata dalle attività più recenti a quelle meno recenti.

Questa sezione dell'applicazione è fondamentale per valutare a posteriori le responsabilità di errori, gli avvenuti malfunzionamenti o, nel caso peggiore, gli eventuali atti di vandalismo e furto.

Ogni elemento della lista, non selezionabile, presenta un'informazione sulla tipologia, che può essere, mutualmente, un'apertura o una chiusura, seguita dalla data e ora, dall'id con cui può essere cercata nel database e dall'id dell'ordine, anch'esso utile per approfondire le dinamiche della situazione.

Nella Tabella 3.6 viene riportata una scheda del requisito.

- RF7: Il responsabile visualizza gli ordini passati.

  Nella home page, il responsabile ha un pulsante, che, se premuto, reindirizza ad una nuova schermata contenente la lista di tutti gli ordini che il sistema ha registrato e che sono passivi.

Nome	**RF6. Il responsabile visualizza le aperture e le chiusure**
Attori	Responsabile
Flusso	1. Il responsabile si trova nella home page. 2. Il responsabile seleziona uno sportello. 3. Il responsabile visualizza la schermata della lista di tutte le aperture e chiusure relative a tale sportello.
Output	Schermata dell'app contenente la lista delle aperture e delle chiusure di uno sportello.
Eccezioni	1. Il server non risponde. Dopo il tempo di timeout (30 secondi) viene mostrato un messaggio di errore al responsabile.

Tabella 3.6: Scheda del requisito RF6

Un ordine è definito passivo dal momento in cui cliente ritira i prodotti ordinati e chiude lo sportello, o gli sportelli, in cui essi si trovavano.

Questa lista è riordinata dall'ordine più recente al meno recente; di ogni ordine è visibile il codice il rilevanza, seguito dalla data di creazione dell'ordine, del nome del responsabile e dell'elenco degli sportelli di cui prevede l'uso; ogni elemento della lista non è cliccabile, dal momento che non vi possono essere effettuate modifiche.

Questa lista potrebbe essere molto lunga; quindi, per precauzione, vengono inizialmente caricati e mostrati dal sistema solo i 10 ordini più recenti, per non appesantire l'app. Ovviamente, se il responsabile lo richiede, può premere sul Floating Action Button in fondo a destra per estendere tale lista e visualizzare tutti gli ordini passati.

Nella Tabella 3.7 viene riportata una scheda del requisito.

Nome	**RF7. Il responsabile visualizza gli ordini passati**
Attori	Responsabile
Flusso	1. Il responsabile si trova nella home page. 2. Il responsabile preme sul pulsante per visualizzare gli ordini passati. 3. Il responsabile visualizza la schermata della lista dei dieci ordini passati effettuati più recentemente. 4. Il responsabile preme sul pulsante per estendere il contenuto. 5. Il responsabile visualizza la schermata della lista di tutti gli ordini passati.
Output	Schermata dell'app contenente la lista degli ordini passati.
Eccezioni	1. Il server non risponde. Dopo il tempo di timeout (30 secondi) viene mostrato un messaggio di errore al responsabile.

Tabella 3.7: Scheda del requisito RF7

- RF8: Il responsabile carica i prodotti.

  Il responsabile possiede le chiavi fisiche, ed eventualmente anche una copia, di ogni sportello, con le quali apre gli sportelli vuoti in cui deve inserire l'ordine, per poi richiuderli.

Il sistema dell'armadietto, in automatico, registra le aperture e le chiusure avvenute, per ogni sportello coinvolto.

Nella Tabella 3.8 viene riportata una scheda del requisito.

Nome	
Attori	Responsabile
Flusso	1. Il responsabile apre gli sportelli interessati. 2. Il responsabile inserisce i prodotti. 3. Il responsabile chiude gli sportelli.
Output	Nessuno
Eccezioni	Nessuna

Tabella 3.8: Scheda del requisito RF8

- RF9: Il cliente visualizza le istruzioni.

  Il cliente si reca sulla locazione dell'armadietto per prelevare i prodotti ordinati, e, soprattutto se è la sua prima volta, ha necessità di leggere le istruzioni per l'uso corretto del dispositivo.

  Ha già ricevuto tali istruzioni per email o via sms, ma, per ulteriore comodità, può leggerle sul display retroilluminato collocato al centro dell'armadietto.

  Di norma, la prima istruzione invita il cliente a digitare il codice ricevuto nel tastierino, per poi aggiornarsi in automatico man mano che l'utente compie le operazioni.

  Infine, una volta terminato il prelievo, ricorda al cliente di richiudere gli sportelli, e lo saluta, ringraziandolo per l'acquisto.

  Nella Tabella 3.9 viene riportata una scheda del requisito.

Nome	**RF9. Il cliente visualizza le istruzioni**
Attori	Cliente
Flusso	1. Il display dell'armadietto invita il cliente a digitare il codice ricevuto. 2. Il display mostra, in diretta, i numeri digitati sul tastierino. 3. Il display mostra una scritta con l'avvenuta corrispondenza del codice oppure con l'eventuale errore. 4. Il display comunica le etichette degli sportelli che si sono automaticamente aperti. 5. Il display invita il cliente a richiudere tutti gli sportelli. 6. Il display ringrazia il cliente.
Output	Sequenza di istruzioni.
Eccezioni	1. Se il cliente commette 3 errori di seguito, il sistema si blocca per 30 secondi, mostrando un messaggio di errore seguito da un countdown.

Tabella 3.9: Scheda del requisito RF9

- RF10: Il cliente digita il codice.

Il cliente, dopo aver letto le istruzioni, può inserire il codice ricevuto usando l'apposito tastierino numerico, situato nelle immediate vicinanze del display.

Ogni volta che il cliente digita una cifra, oltre ad essere memorizzata dal sistema, essa viene visualizzata sul display nello schermo, così che il cliente veda esattamente, ed in tempo reale, ciò che sta digitando.

Se il cliente si accorge di aver commesso un errore, può usufruire del tastino con la lettera C, per cancellare l'ultima cifra inserita, ed, ovviamente, può ripetere questa operazione fino a cancellare tutto il numero.

Quando il cliente ha digitato un numero a sei cifre, il sistema, in automatico, senza la necessità di alcun tasto di conferma, confronta il codice inserito con ogni codice degli ordini attivi che conosce, e, se c'è una corrispondenza, provvede ad aprire gli sportelli associati a tale ordine.

In alternativa, se non c'è alcuna corrispondenza, il sistema incrementa un contatore degli errori e ripropone al cliente di inserire il codice dall'inizio.

Se il contatore degli errori arriva a tre, il sistema si blocca, per questioni di sicurezza, per 30 secondi, dopodiché il contatore di errori si azzera e il cliente può riprovare ad inserire il proprio codice.

Nella Tabella 3.10 viene riportata una scheda del requisito.

Nome	**RF10. Il cliente digita il codice**
Attori	Cliente
Flusso	1. Il cliente digita il codice. 2. Il sistema mostra ogni cifra inserita. 3. Il sistema comunica che il codice è corretto.
Output	Apertura degli sportelli previsti per l'ordine.
Eccezioni	1. Se il cliente commette 3 errori di seguito, il sistema si blocca per 30 secondi, mostrando un messaggio di errore seguito da un countdown. 2. Terminato il countdown, il sistema si sblocca e chiede al cliente di inserire nuovamente il codice.

Tabella 3.10: Scheda del requisito RF10

- RF11: Il cliente preleva i prodotti.

  Il cliente, una volta digitato il codice corretto, vede aprirsi in automatico lo sportello o gli sportelli contenenti i prodotti acquistati, appartenenti all'ordine.

  L'apertura consiste nell'azionamento del motore relativo allo sportello, che apre la serratura. Una piccola molla fa sì che lo sportello si apra leggermente, per dare un input visivo al cliente che, quindi, capisce istantaneamente quali sono gli sportelli in cui si trovano i suoi prodotti.

  Il sistema registra autonomamente l'avvenuta apertura, per ogni sportello coinvolto.

  Nella Tabella 3.11 viene riportata una scheda del requisito.

- RF12: Il cliente chiude lo sportello.

  Il cliente preleva il contenuto degli sportelli aperti, e poi chiude questi ultimi.

Nome	**RF11. Il cliente preleva i prodotti**
Attori	Cliente
Flusso	1. Il sistema attiva i motori che sbloccano le serrature. 2. La molla applica una forza sullo sportello che lo apre leggermente.
Output	Nessuno
Eccezioni	Nessuna

Tabella 3.11: Scheda del requisito RF11

Per effettuare correttamente la manovra di chiusura, è necessario appoggiare delicatamente ogni sportello in posizione di chiusura.

Un elettromagnete attivatosi automaticamente facilita lo stazionamento dello sportello in tale posizione, e, dopo un secondo di attesa, il motore si attiva e aziona la chiusura della serratura. Dopodiché l'elettromagnete smette di funzionare.

L'avvenuta chiusura viene registrata dal sistema.

Nella Tabella 3.12 viene riportata una scheda del requisito.

Nome	**RF12. Il cliente chiude lo sportello**
Attori	Cliente
Flusso	1. Il cliente posiziona ogni sportello in posizione di chiusura. 2. L'elettromagnete mantiene chiuso lo sportello. 3. Il motore blocca la serratura. 4. L'elettromagnete si spegne. 5. Il display ringrazia il cliente.
Output	Il display ringrazia il cliente.
Eccezioni	Nessuna

Tabella 3.12: Scheda del requisito RF12

Requisiti non funzionali

- RNF1. Il sistema utilizza Firebase come DBMS.

  Il sistema di back-end della ferramenta esiste da tempo, e vi sono già salvati molti dati aziendali.

  Di conseguenza il software che stiamo sviluppando consiste in un altro modulo che viene aggiunto al sistema complessivo, e come tale, deve poter collaborare in maniera efficace e diretta, per quanto possibile.

  Attualmente il server utilizzato è Firebase, quindi il nostro vincolo consiste nel salvare i dati nel Firebase Cloud Storage, usare il servizio di autenticazione di Firebase Authentication, memorizzare i file sul Firebase Storage e soprattutto fare il deploy delle funzioni su Firebase Functions.

- RNF2. L'applicazione mobile deve essere realizzata con i colori blu, rosso e bianco.

  L'applicazione mobile deve rispecchiare, almeno graficamente, la ferramenta. Come colori sono stati scelti quelli del logo, ovvero il blu, il rosso e il bianco.

Il blu è generalmente associato a sentimenti di calma, stabilità e affidabilità, ed è spesso utilizzato in contesti che richiedono fiducia, e quindi lo risaltiamo nell'app usandolo come colore principale rispetto al rosso, che invece evoca forti emozioni, ed è legato alla passione, all'energia e all'urgenza.

- RNF3. Il cabinato deve essere alto almeno 2.50 metri.

  Il cabinato deve poter contenere tutti i possibili prodotti, e, ad oggi, quelli più ingombranti sono le cornici delle finestre, che, seppur essendo fine e strette, sono lunghe più di due metri.

  Considerando che il cabinato ha bisogno di uno spessore in basso e uno spessore in alto, abbiamo optato per la scelta dell'altezza complessiva del dispositivo di due metri e mezzo.

- RNF4. Il cabinato deve essere zinco-plastificato, e garantire una protezione dalla ruggine per 10 anni, ed essere posizionato all'esterno.

  Per poter posizionare il cabinato all'esterno, servono numerosi trattamenti chimici alla lamiera di cui è composto.

  Il principale problema sono le intemperie metereologiche, infatti esse possono danneggiare la lamiera, penetrando al suo interno e causando l'arrugginimento. Per ovviare a ciò, abbiamo selezionato, come possibili fornitori, solo aziende che ci garantiscono dieci anni di protezione, grazie al trattamento per il quale la lamiera viene zinco-plastificata.

  Ovviamente l'interezza di questa operazione deve essere eseguita a monte del montaggio della componentistica, e quindi la prima parte del progetto prevede la progettazione dei fori all'interno dei quali passeranno i cavi.

3.3 Diagramma dei casi d'uso

Il diagramma dei casi d'uso esprime un comportamento, offerto o desiderato, sulla base dei suoi risultati osservabili. Serve per visualizzare chiaramente chi o che cosa ha a che fare con il sistema (attore) e che cosa viene fatto (caso d'uso).

Il diagramma dei casi d'uso relativo al nostro sistema viene riportato in Figura 3.1.

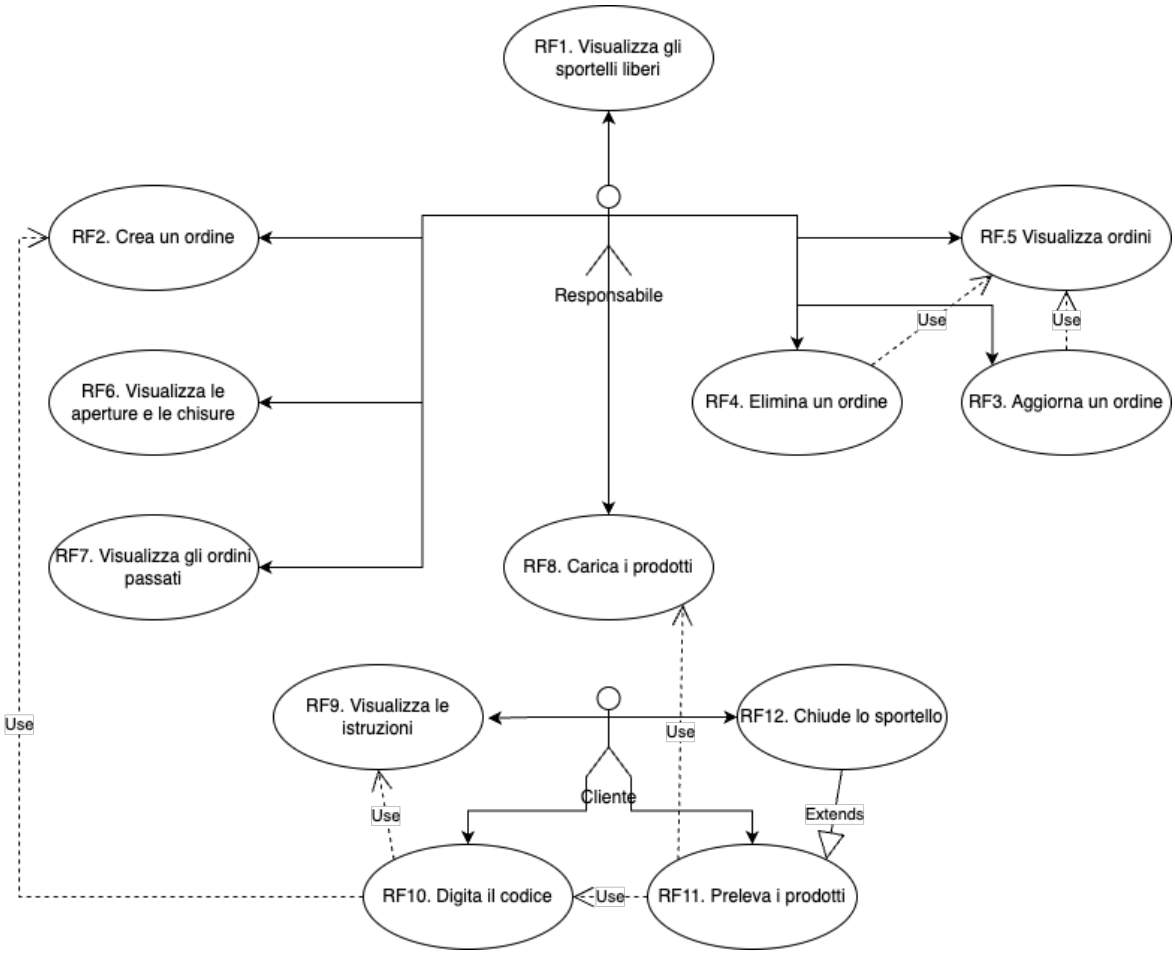


Figura 3.1: Diagramma dei casi d'uso

CAPITOLO 4

Progettazione dell'hardware e del back-end

Un qualsiasi software che operi nella rete deve poter salvare i dati necessari in una locazione condivisa, alla quale, con determinate autorizzazioni e coi relativi permessi, vi si può accedere da più di un dispositivo.

Il nostro prodotto rientra, evidentemente, in questa tipologia di applicativi; quindi, abbiamo ritenuto opportuno progettare sia la componente di back-end che quella di front-end. In questo capitolo analizzeremo in particolare la componente di back-end.

La componente elettronica, inoltre, gioca un ruolo fondamentale nei progetti di Internet of Things (IoT). Essa permette di raccogliere e trasmettere dati da diversi dispositivi connessi in rete, come sensori, attuatori e micro-controllori.

Tali dispositivi elettronici possono essere utilizzati per monitorare e controllare diversi aspetti dell'ambiente circostante, come la temperatura, l'umidità, la pressione e molti altri fattori.

Inoltre, la componente elettronica consente di elaborare i dati raccolti e di utilizzarli per prendere decisioni automatiche e migliorare l'efficienza del sistema.

4.1 Progettazione dell'hardware

Iniziamo il capitolo trattando la progettazione dell'hardware, di cui analizzeremo in primis la struttura e l'architettura, per poi passare alla descrizione di ogni componente ed infine capire come esse collaborano tra loro per raggiungere lo scopo finale.

4.1.1 Struttura generale

L'architettura di un progetto di Internet of Things (IoT) comprende diversi elementi che operano insieme per fornire funzionalità complete.

Nel caso specifico descritto, c'è un micro-controllore che svolge il ruolo di cuore del sistema e controlla i motori collegati alle serrature degli sportelli.

I motori sono azionati dal micro-controllore per aprire o chiudere le serrature a seconda delle esigenze.

Inoltre, il micro-controllore è in grado di raccogliere informazioni da diversi sensori di contatto, che indicano se gli sportelli sono aperti o chiusi. Gli elettromagneti possono essere utilizzati per rafforzare la sicurezza delle serrature e impedire l'apertura non autorizzata.

Il sistema include anche un tastierino per l'inserimento della password e un LCD screen che fornisce informazioni sullo stato del sistema stesso e sulle operazioni effettuate.

Il micro-controllore è in grado di gestire la comunicazione tra questi diversi componenti e di elaborare le informazioni raccolte per prendere decisioni e agire di conseguenza.

In sintesi, l'architettura di un progetto IoT comprende un micro-controllore che controlla i motori delle serrature, i sensori di contatto, gli elettromagneti, un tastierino e un LCD screen, e garantisce la raccolta, l'elaborazione e la trasmissione dei dati in modo efficiente.

Tale architettura è progettata per fornire una soluzione integrata e personalizzabile per la gestione delle serrature degli sportelli e il monitoraggio dello stato dei sistemi.

4.1.2 Architettura dell'elettronica

L'architettura del nostro sistema IoT è basata su un micro-controllore che controlla i motori delle serrature, i sensori di contatto, gli elettromagneti, un tastierino e un LCD screen ed è caratterizzata da una topologia di rete a stella.

Il micro-controllore costituisce il nodo centrale della rete e si occupa di controllare e coordinare l'interazione tra tutti i componenti collegati.

I motori delle serrature e gli elettromagneti costituiscono i nodi finali della rete e sono controllati dal micro-controllore tramite connessioni cablate.

I sensori di contatto, il tastierino e l'LCD screen costituiscono, invece, i nodi intermedi e sono connessi al micro-controllore tramite interfacce di comunicazione specifiche.

Il micro-controllore, che costituisce il cuore del sistema, è dotato di una serie di funzionalità di elaborazione dei dati, tra cui la raccolta, l'analisi e l'elaborazione di informazioni provenienti dai vari sensori connessi alla rete.

Grazie a queste funzionalità, esso è in grado di prendere decisioni e di eseguire azioni in tempo reale, garantendo un alto grado di efficienza e precisione.

L'architettura a stella permette di creare un sistema scalabile e personalizzabile, in cui è possibile aggiungere o rimuovere componenti in base alle specifiche esigenze.

Inoltre, l'utilizzo di connessioni cablate garantisce una maggiore sicurezza e stabilità del sistema, minimizzando il rischio di interferenze o malfunzionamenti.

4.1.3 Componenti

Nel seguito di questo elaborato, procederemo con un'analisi dettagliata delle componenti elettroniche che costituiscono l'architettura del nostro progetto.

Questa sezione avrà lo scopo di elencare e descrivere ogni singolo elemento hardware utilizzato, illustrando le specifiche tecniche e le funzionalità che rendono possibile il funzionamento del sistema nel suo complesso.

Scheda forata

La scheda forata è stata a lungo un componente fondamentale nella realizzazione di progetti di elettronica. Questa scheda è costituita da una base di materiale isolante, di solito fenolico o fibra di vetro, con un reticolo di fori regolari disposti in modo da consentire l'inserimento di componenti elettronici, come resistori, condensatori, diodi e integrati.

Grazie alla sua flessibilità e alla facilità di utilizzo, la scheda forata è stata utilizzata per molti anni come piattaforma di prototipazione per progetti di IoT.

Per la realizzazione di un progetto di questo tipo, la scheda forata viene utilizzata per saldare i componenti elettronici necessari per il sistema IoT, creando un circuito personalizzato e su misura per le specifiche esigenze del progetto.

La scheda forata è ancora oggi un'opzione popolare per i progettisti di IoT che cercano una soluzione semplice ed economica per la realizzazione di prototipi. Tuttavia, essa ha alcune limitazioni, come, ad esempio, la difficoltà di realizzare circuiti molto complessi o la mancanza di precisione nell'assemblaggio.

Per questo motivo, molte aziende stanno ora utilizzando tecnologie più avanzate, come le schede a circuito stampato (PCB), per la realizzazione di progetti di IoT più sofisticati e avanzati.

La Figura 4.1 mostra un esempio di scheda forata.

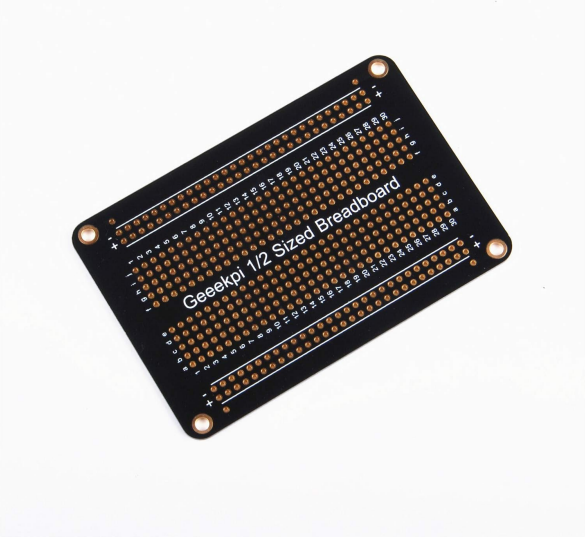


Figura 4.1: Raffigurazione di una scheda forata

Arduino Nano IoT 33

Arduino Nano IoT 33 è una scheda a micro-controllore sviluppata da Arduino, progettata specificamente per applicazioni IoT. Tale scheda presenta un design compatto e leggero, che la rende adatta per applicazioni mobili e wearable.

Il Nano IoT 33 è basato sul micro-controllore Arm Cortex-M0+ SAMD21 di Atmel e su un modulo wireless a banda larga (LTE-M e NB-IoT) integrato, che consente la connessione a Internet e la comunicazione con altri dispositivi IoT.

Inoltre, esso è dotato di un accelerometro integrato, una memoria flash da 256 KB e 32 KB di RAM, nonché di un connettore per schede microSD per la memorizzazione di dati aggiuntivi.

La scheda supporta l'IDE di Arduino, una piattaforma open-source che permette di scrivere e caricare facilmente il codice su dispositivi Arduino, semplificando notevolmente la creazione di applicazioni IoT.

Inoltre, Arduino Nano IoT 33 è compatibile con molte librerie di terze parti e include anche alcune librerie predefinite per semplificare la programmazione di funzioni comuni.

Grazie alle sue funzionalità avanzate e alla facilità di utilizzo, Arduino Nano IoT 33 è diventato uno strumento popolare per la creazione di progetti IoT, dai prototipi alle applicazioni commerciali.

La Figura 4.2 mostra un esempio di Arduino Nano IoT 33.

Schermo LCD

Lo schermo LCD con connettore I2C è un dispositivo che offre una visualizzazione a caratteri o a grafica su un display a cristalli liquidi (LCD) e una connessione I2C per comunicare con altri dispositivi. L'interfaccia I2C, che significa "Inter-Integrated Circuit", è un protocollo di comunicazione seriale a due fili che consente la comunicazione tra micro-controllori e dispositivi periferici.

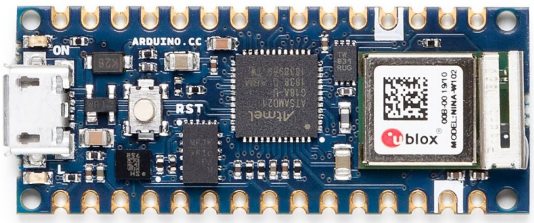


Figura 4.2: Raffigurazione di una scheda Arduino Nano IoT 33

La connessione I2C semplifica la connessione di dispositivi all'interno di un sistema, poiché utilizza solo due fili per la comunicazione dati e la sincronizzazione, riducendo il numero di pin necessari per la connessione. Questo rende l'implementazione dello schermo LCD con connettore I2C particolarmente adatto per progetti IoT e wearable.

Lo schermo LCD con connettore I2C è disponibile in diverse dimensioni e risoluzioni e può essere facilmente programmato per visualizzare testo, numeri, grafici e icone. Questi schermi possono essere utilizzati per visualizzare informazioni in tempo reale.

In sintesi, lo schermo LCD con connettore I2C è un componente versatile e utile in molti progetti di IoT, in grado di visualizzare informazioni importanti e di comunicare con altri dispositivi attraverso la connessione I2C.

La Figura 4.3 mostra un esempio di uno schermo LCD.

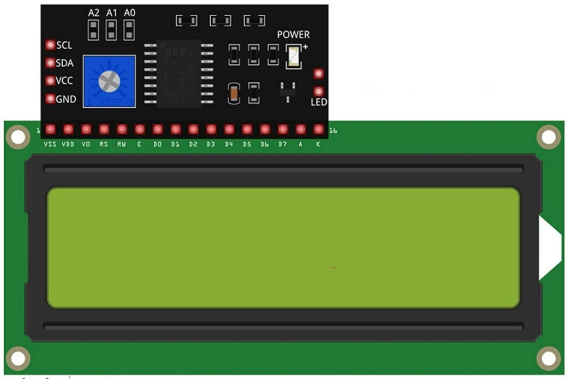


Figura 4.3: Raffigurazione di uno schermo LCD

Tastiera

Un tastierino numerico con 8 connettori matriciali è un tipo di tastiera che consente di inserire numeri e simboli utilizzando una matrice di pulsanti.

Questo tipo di tastierino può essere utilizzato in diversi contesti, ad esempio per inserire codici di accesso, password o numeri di telefono.

La matrice di pulsanti è organizzata in righe e colonne, e ogni pulsante è collegato a due connettori diversi.

Ciò permette di ridurre il numero di connettori necessari per collegare la tastiera al computer o al dispositivo di destinazione, rendendola una soluzione più economica e conveniente.

Inoltre, la disposizione matriciale dei pulsanti consente di ridurre le dimensioni complessive del tastierino, rendendolo compatto e facile da utilizzare.

La Figura 4.4 mostra un esempio di tastierino.

Figura 4.4: Raffigurazione di un tastierino

Scheda d'espansione TCA9548A

La scheda d'espansione TCA9548A è un dispositivo che consente di espandere il numero di dispositivi I2C che possono essere collegati a un micro-controllore o a un microprocessore.

Questa scheda è dotata di un multiplexer I2C a 8 canali, che permette di selezionare uno dei dispositivi I2C collegati alla scheda e di indirizzarlo in modo selettivo.

Questo significa che è possibile collegare più dispositivi I2C a un singolo bus, riducendo il numero di pin necessari per collegare i dispositivi al micro-controllore o al microprocessore.

La scheda d'espansione TCA9548A è particolarmente utile in applicazioni in cui è necessario controllare più dispositivi I2C, come, ad esempio, nei sistemi di controllo e di monitoraggio industriale, nei sistemi di automazione domestica, o nei sistemi di misurazione e di acquisizione dati.

Grazie alla sua flessibilità e alla sua semplicità d'uso, la scheda d'espansione TCA9548A è diventata uno strumento essenziale per molti progettisti e ingegneri che lavorano con dispositivi I2C.

La Figura 4.5 mostra un esempio di TCA9548A.

Scheda d'espansione PCF8574

La scheda d'espansione PCF8574 è un dispositivo che consente di espandere il numero di porte I/O disponibili su un micro-controllore o su un microprocessore.

Questa scheda è dotata di un expander I2C a 8 bit, che permette di controllare fino a 8 porte di input/output. Ciò significa che è possibile collegare più dispositivi elettronici, come, ad esempio, sensori, attuatori o display, a un singolo micro-controllore o microprocessore, senza dover aggiungere ulteriori porte di input/output sul dispositivo.

La scheda d'espansione PCF8574 è particolarmente utile in applicazioni in cui è necessario controllare più dispositivi elettronici contemporaneamente, come, ad esempio, nei sistemi di

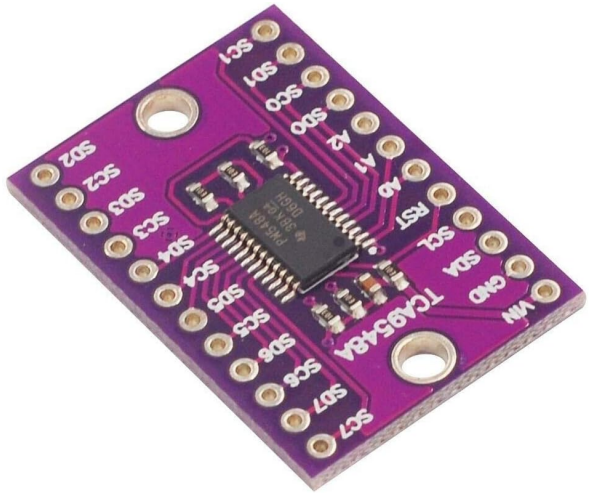


Figura 4.5: Raffigurazione di una scheda TCA9548A

controllo e di automazione industriale, nei sistemi di monitoraggio ambientale, o nei sistemi di controllo dell'illuminazione.

La scheda d'espansione PCF8574 è facile da utilizzare e da integrare nei circuiti elettronici esistenti. Essa comunica con il micro-controllore o microprocessore tramite il protocollo di comunicazione seriale I2C, che consente di trasmettere i dati in modo efficiente e affidabile.

Inoltre, la scheda è dotata di un indirizzo I2C programmabile, che consente di utilizzare fino a 8 schede d'espansione PCF8574 su un singolo bus I2C. Ciò significa che è possibile controllare fino a 64 porte di input/output, utilizzando soltanto due pin del micro-controllore o microprocessore.

La Figura 4.6 mostra un esempio di PCF8574.

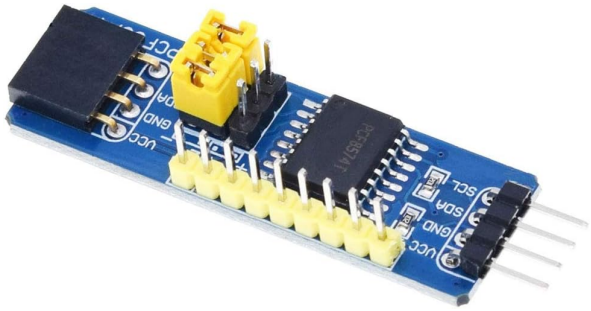


Figura 4.6: Raffigurazione di una scheda PCF8574

Cavi elettrici

I cavi elettrici per progetti di IoT sono un componente fondamentale per la corretta alimentazione dei dispositivi elettronici.

La scelta del cavo elettrico giusto dipende dalle specifiche esigenze del progetto in questione; tale scelta comprende come la lunghezza del cavo, la corrente massima che deve essere trasmessa, il tipo di connettori necessari e la compatibilità con gli standard di sicurezza elettrica.

Inoltre, la scelta del cavo elettrico deve considerare l'ambiente in cui il progetto verrà utilizzato; per questo motivo, è importante scegliere cavi elettrici di alta qualità, resistenti e adatti alle specifiche esigenze del progetto.

Nascondi-cavi

Le fascette nascondi-cavi sono un accessorio molto utile per organizzare e nascondere i cavi elettrici in modo ordinato e pulito. Queste fascette sono costituite da una guaina in tessuto o in PVC, che copre i cavi e li mantiene insieme, proteggendoli da eventuali danni o interruzioni.

Grazie alla loro versatilità e alla loro facilità d'uso, le fascette nascondi-cavi sono un accessorio molto apprezzato sia in ambito domestico che professionale, per organizzare e nascondere i cavi in modo pulito e sicuro.

Inoltre, esse rappresentano una soluzione estetica e funzionale per ridurre l'ingombro dei cavi e migliorare la fruibilità degli spazi.

La Figura 4.7 mostra un esempio di nascondi-cavi.

Figura 4.7: Raffigurazione di una fascia nascondi-cavi

Ferma-cavi

I ferma-cavi sono un accessorio molto utile per evitare che i cavi elettrici si muovano o si aggroviglino, creando disordine e causando potenzialmente danni ai dispositivi elettronici. Questi dispositivi sono costituiti da un supporto in plastica o in metallo, dotato di apposite clip o ganci, che consentono di fissare i cavi elettrici in modo ordinato e sicuro.

Grazie alla loro facilità d'uso e alla loro efficacia nel mantenere i cavi in ordine, i ferma-cavi rappresentano un accessorio molto apprezzato sia in ambito domestico che professionale, per organizzare e proteggere i cavi elettrici.

Inoltre, l'utilizzo dei ferma-cavi è molto importante per garantire la sicurezza dei dispositivi elettronici, poiché consente di evitare che i cavi si muovano o si aggroviglino, riducendo il rischio di cortocircuiti o di interruzioni di corrente.

In questo modo, i ferma-cavi contribuiscono anche a prolungare la durata dei dispositivi elettronici, evitando l'usura dei cavi e dei connettori e prevenendo eventuali danni ai componenti elettronici.

La Figura 4.8 mostra un esempio di ferma-cavi.

Trasformatore

I trasformatori da 220V a 5V sono dispositivi elettronici che consentono di convertire la tensione di rete (220V) in una bassa tensione (5V), utilizzata comunemente per alimentare dispositivi elettronici, come smartphone, tablet, dispositivi IoT e altri dispositivi elettronici a bassa potenza. Questi trasformatori sono dotati di un circuito di regolazione della tensione che consente di mantenere la tensione a 5V anche in presenza di variazioni della tensione di ingresso.

I trasformatori da 220V a 5V sono disponibili in diverse forme e dimensioni, a seconda dell'applicazione per cui sono destinati.

Figura 4.8: Raffigurazione di un ferma-cavi

Alcuni modelli sono dotati di un connettore USB, che consente di collegare direttamente i dispositivi elettronici, mentre altri modelli prevedono l'utilizzo di un cavo di alimentazione per il collegamento ai dispositivi. Inoltre, esistono modelli di trasformatori da 220V a 5V con diverse potenze di uscita, in modo da poter alimentare sia dispositivi a bassa potenza che dispositivi più esigenti.

I trasformatori da 220V a 5V rappresentano un componente molto importante per i progetti di IoT e per l'alimentazione dei dispositivi elettronici in generale. Grazie alla loro capacità di convertire la tensione di rete in una bassa tensione, questi trasformatori consentono di alimentare i dispositivi elettronici in modo sicuro ed efficiente, senza rischi di surriscaldamento o di sovratensioni.

Dunque, essendo dotati di un circuito di regolazione della tensione, essi consentono di mantenere la tensione costante, garantendo la stabilità e la sicurezza dell'alimentazione elettrica dei dispositivi.

La Figura 4.9 mostra un esempio di trasformatore.

Figura 4.9: Raffigurazione di un trasformatore

Motore servo

I motori servo sono dispositivi elettronici che consentono di controllare in modo preciso la posizione e la velocità di un albero rotante. Questi motori sono costituiti da un motore

elettrico, un sistema di ingranaggi e un circuito di controllo, che consente di comandare la posizione e la velocità dell'albero rotante con una precisione elevata.

I motori servo sono utilizzati in molteplici applicazioni, come robotica, automazione industriale, stampanti 3D, droni e molti altri dispositivi che richiedono un controllo preciso del movimento.

I servo motori sono dotati di un feedback di posizione, che consente di rilevare la posizione effettiva dell'albero rotante in tempo reale. Questo feedback viene utilizzato dal circuito di controllo per regolare la posizione dell'albero rotante e per mantenere la posizione desiderata con una precisione elevata.

Grazie a tale caratteristica, i motori servo consentono di controllare in modo preciso la posizione e la velocità dell'albero rotante, anche in presenza di eventuali interferenze esterne.

Essi sono disponibili in diverse forme e dimensioni, a seconda dell'applicazione per cui sono destinati. Inoltre, esistono motori servo con diverse caratteristiche di coppia e di velocità, in modo da poter soddisfare le diverse esigenze applicative.

La Figura 4.10 mostra un esempio di motore servo.

Figura 4.10: Raffigurazione di un motore servo

Relay

I relay sono dispositivi elettronici utilizzati per controllare la commutazione di un segnale o di una tensione elettrica da un circuito all'altro.

Questi componenti sono costituiti da una bobina di avvolgimento, che viene alimentata da un segnale elettrico, e da un contatto meccanico, che consente di chiudere o aprire il circuito in base alla presenza o all'assenza del segnale di alimentazione. Grazie a questa caratteristica, i relay consentono di controllare in modo preciso l'apertura e la chiusura di circuiti elettrici anche in presenza di alte tensioni o correnti.

I relay sono utilizzati in molteplici applicazioni, come l'automazione industriale, l'elettronica di potenza, il controllo di motori e molti altri dispositivi che richiedono un controllo preciso della commutazione di segnali elettrici.

Grazie alla loro capacità di isolare completamente il circuito di controllo dal circuito di potenza, i relay consentono di garantire un elevato livello di sicurezza e di affidabilità, evitando eventuali danni o interferenze.

I relay sono disponibili in diverse forme e dimensioni, a seconda dell'applicazione per cui sono destinati. Inoltre, esistono relay con diverse caratteristiche di tensione di alimentazione, di corrente massima commutabile e di tempo di risposta, in modo da poter soddisfare le diverse esigenze applicative.

La Figura 4.11 mostra un esempio di relay.

Figura 4.11: Raffigurazione di un relay

Sensore di fine corsa

I sensori di fine corsa sono dispositivi utilizzati per rilevare la posizione di un oggetto o di una parte mobile di una macchina elettronica o meccanica.

Questi sensori sono costituiti da una coppia di contatti metallici che si chiudono o si aprono in base alla posizione dell'oggetto o della parte mobile.

Quando il contatto si chiude, viene generato un segnale elettrico che consente di interrompere o modificare il funzionamento della macchina o del dispositivo.

I sensori di fine corsa sono utilizzati in molte applicazioni, come robotica, automazione industriale, stampanti 3D e molti altri dispositivi che richiedono un controllo preciso della posizione e del movimento.

Grazie alla loro capacità di rilevare la posizione con precisione elevata, i sensori di fine corsa consentono di prevenire eventuali malfunzionamenti o danni, garantendo un elevato livello di sicurezza e affidabilità.

I sensori di fine corsa sono disponibili in diverse forme e dimensioni, a seconda dell'applicazione per cui sono destinati. Inoltre, esistono sensori di fine corsa con diverse caratteristiche di contatto, come normalmente aperto o normalmente chiuso, in modo da poter soddisfare le diverse esigenze applicative.

La Figura 4.12 mostra un esempio di sensore di fine corsa.

Elettromagnete

Gli elettromagneti sono dispositivi che utilizzano il principio dell'elettromagnetismo per generare un campo magnetico.

Questi componenti sono costituiti da un nucleo di materiale ferromagnetico, come il ferro o il nichel, avvolto da un filo di rame o di un altro materiale conduttore. Quando viene fatta passare una corrente elettrica attraverso il filo, si genera un campo magnetico che circonda il nucleo, il quale diventa magnetizzato.

Gli elettromagneti sono utilizzati in molte applicazioni, come nell'automazione industriale, nei motori elettrici, nei dispositivi di sicurezza e in molti altri dispositivi. Grazie alla loro capacità di generare un campo magnetico controllato e preciso, gli elettromagneti consentono di controllare il movimento di oggetti o parti mobili e di attivare o disattivare circuiti elettrici.

Gli elettromagneti sono disponibili in diverse forme e dimensioni, a seconda dell'applicazione per cui sono destinati.

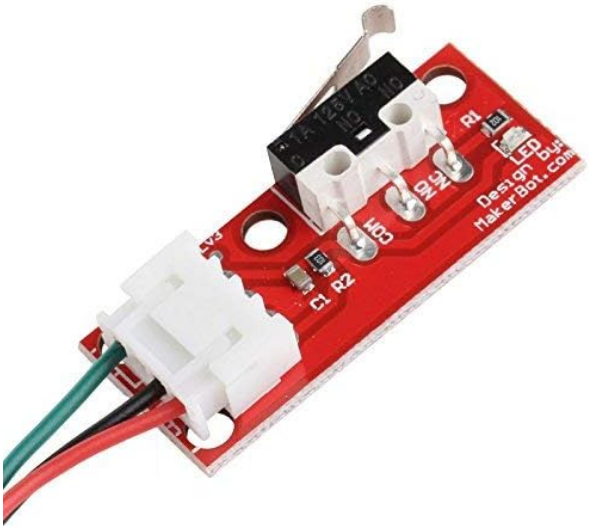


Figura 4.12: Raffigurazione di un sensore di fine corsa

Inoltre, esistono elettromagneti con diverse caratteristiche di tensione di alimentazione, di intensità di corrente e di forza di attrazione, in modo da poter soddisfare le diverse esigenze applicative.

La Figura 4.13 mostra un esempio di elettromagnete.

Figura 4.13: Raffigurazione di un elettromagnete

4.1.4 Funzionamento

Arduino Nano IoT 33 svolge il ruolo di centro stella, quindi lo poniamo anche, idealmente, al centro del nostro circuito, collegato alla corrente di alimentazione, di 5V.

Ogni 5 minuti, Arduino aggiorna i codici di sblocco delle locker door, conoscendo, quindi, a partire da un codice, a quali locker door esso si riferisce.

Diagramma di flusso

I diagrammi di flusso sono strumenti grafici utilizzati per rappresentare visivamente il flusso di controllo o il processo logico all'interno di un sistema o di un programma. Tali diagrammi utilizzano una varietà di simboli standardizzati, come rettangoli, ovali e rombi, per rappresentare diverse operazioni, decisioni e processi.

I rettangoli indicano le operazioni o le istruzioni, gli ovali rappresentano l'inizio o la fine del processo, mentre i rombi simboleggiano i punti di decisione. Le linee di connessione

mostrano la direzione del flusso e collegano i vari elementi del diagramma, illustrando il percorso che segue l'algoritmo o il processo.

I diagrammi di flusso sono strumenti preziosi per la progettazione, l'analisi e la documentazione dei sistemi informatici e dei processi aziendali, facilitando la comprensione e la comunicazione delle dinamiche operative sia per gli sviluppatori che per gli stakeholder.

Nella Figura 4.14 viene riportato il diagramma di flusso del nostro progetto.

Flusso di esecuzione degli sportelli

Quando un utente inserisce un codice nel tastierino, se questo è errato avviene un'attesa, di 30 secondi, che serve principalmente ad evitare attacchi di tipo brute-force.

Se, invece, il codice è riconosciuto da Arduino, ques'ultimo individua le relative locker door e inizializza la procedura d'apertura per ciascuno di essi.

Tale operazione si svolge in diversi passi:

1. Accensione del relay, con conseguente ed immediata accensione del sensore di fine-corsa, del servo motore e dell'elettromagnete.
2. Attesa di 500 ms.
3. Rotazione del servo motore in posizione apertura.
4. Attesa di 500 ms.
5. Spegnimento del relay, con conseguente ed immediato spegnimento del sensore di fine-corsa, del servo motore e dell'elettromagnete.
6. Attesa di 4000 ms, ovvero il tempo minimo con cui si prevede che l'utente prelevi i prodotti.
7. Accensione del relay, con conseguente ed immediata accensione del sensore di fine-corsa, del servo motore e dell'elettromagnete.

Per la chiusura sicura di una locker door, il cliente deve solo appoggiare lo sportello a battente; quindi il sensore di fine-corsa intercetta la chiusura; a seguito di ciò vengono effettuati i seguenti passi:

1. Attesa di 500 ms.
2. Posizionamento del servo motore in chiusura.
3. Attesa di 500 ms.
4. Spegnimento del relay, con conseguente ed immediato spegnimento del sensore di fine-corsa, del servo motore e dell'elettromagnete.

4.1.5 Protocolli di comunicazione

I protocolli di comunicazione tra le componenti che usiamo sono 4, ovvero:

- PWM;
- digitale;
- comunicazione seriale;

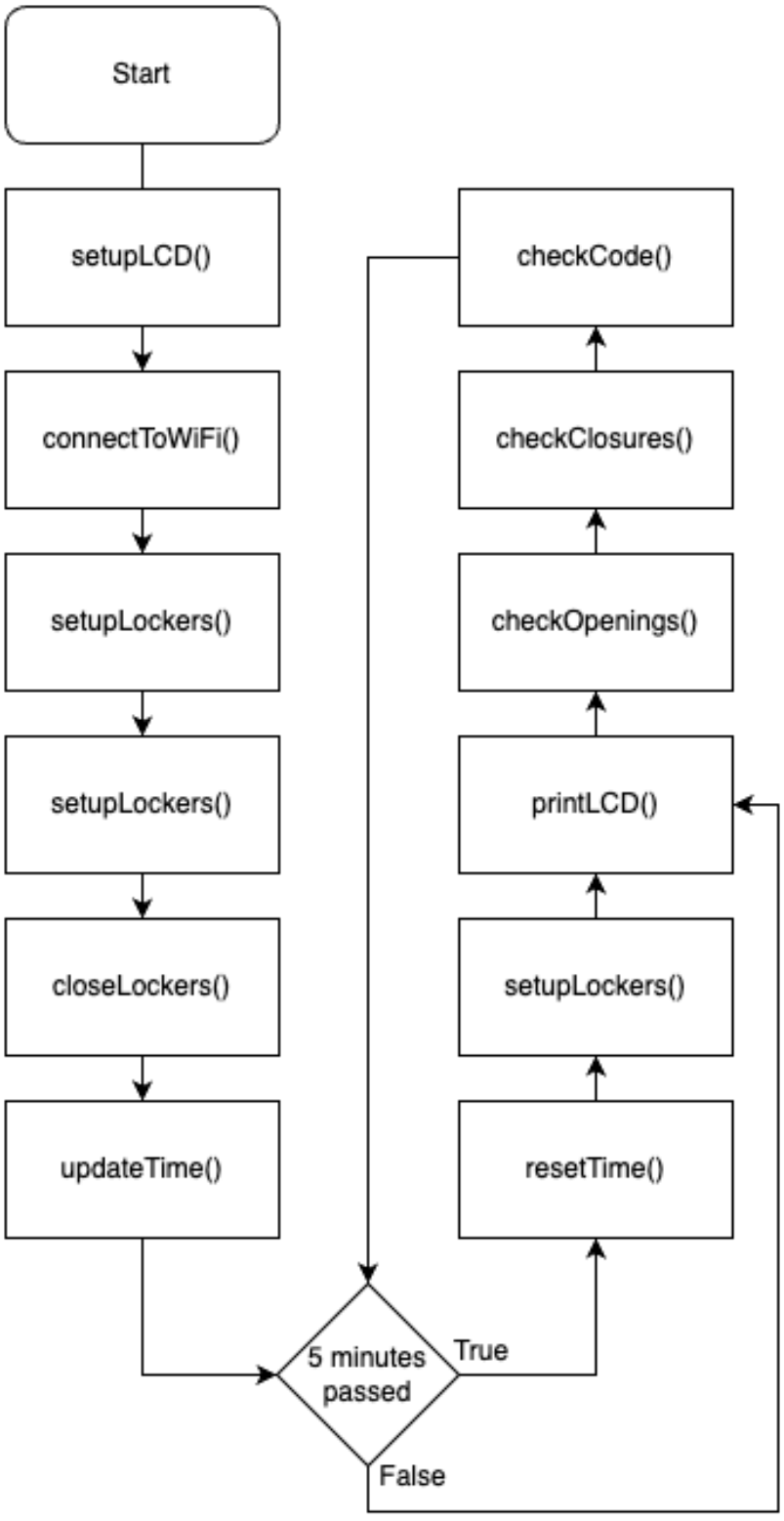


Figura 4.14: Diagramma di flusso del software del controllore

- I2C.

Nelle prossime sottosezioni vengono presentate, la descrizione, le modalità e le motivazioni d'uso dei protocolli di comunicazione nel nostro progetto.

PWM

Il protocollo di modulazione di larghezza dell'impulso (PWM) è un protocollo di comunicazione analogico utilizzato da Arduino. In pratica, il segnale PWM consiste in una serie di impulsi di tensione di larghezza variabile, dove la larghezza dell'impulso rappresenta la percentuale di tempo in cui il segnale è attivo rispetto al tempo totale.

Ad esempio, se il segnale è attivo al 50 per cento del tempo, la tensione in uscita sarà circa la metà del massimo. Arduino utilizza il protocollo PWM tramite i suoi pin di uscita analogici, dove i valori di tensione vengono modulati per controllare il comportamento dei dispositivi esterni.

In generale, la tecnologia PWM è una soluzione efficace ed economica per controllare la velocità e la luminosità dei dispositivi analogici in modo preciso e regolabile; per questa ragione viene utilizzata ampiamente nell'ambito di progetti elettronici. Nel nostro progetto, i pin analogici A4 e A5 di Arduino Nano 33 IoT vengono usati non per modulare la lunghezza d'impulso ma per il protocollo I2C.

Digitale

Il protocollo digitale di Arduino è basato sull'utilizzo di due stati di segnale: HIGH e LOW. Il segnale HIGH rappresenta il valore di tensione logico "1", mentre il segnale LOW rappresenta il valore di tensione logico "0".

Questo protocollo è utilizzato per controllare dispositivi digitali come i LED, i relè, i display a sette segmenti e altri componenti elettronici che funzionano con logica binaria.

Esso viene utilizzato anche per comunicare con altri dispositivi esterni tramite la comunicazione seriale, dove i dati vengono trasmessi utilizzando una sequenza di bit in cui ogni bit rappresenta un valore logico HIGH o LOW.

Grazie a questo protocollo, gli sviluppatori di Arduino possono controllare in modo preciso il comportamento dei dispositivi digitali e comunicare con altri dispositivi in modo affidabile ed efficiente.

Nel nostro progetto, 8 pin digitali di Arduino Nano 33 IoT sono usati per collegarsi al tastierino.

Comunicazione seriale

Il protocollo di comunicazione seriale è uno dei protocolli più utilizzati da Arduino per comunicare con altri dispositivi.

Esso prevede la trasmissione di dati attraverso un singolo filo di comunicazione, che viene utilizzato sia per inviare che per ricevere i dati.

La comunicazione seriale di Arduino utilizza un formato di trasmissione di dati noto come "asincrono", dove i dati vengono trasmessi in sequenza, uno dopo l'altro, senza un segnale di clock separato.

In pratica, i dati vengono inviati come una sequenza di bit, dove ogni bit rappresenta un valore logico HIGH o LOW.

Il protocollo seriale di Arduino prevede l'utilizzo di diversi parametri di configurazione, tra cui la velocità di trasmissione (baud rate), il formato dei dati e il tipo di controllo di flusso.

Noi utilizziamo questo protocollo solo in fase di debugging, per monitorare il comportamento del software in real-time. La velocità di trasmissione utilizzata è pari a 9600 bauds.

I2C

Il protocollo I2C (Inter-Integrated Circuit) è un protocollo di comunicazione seriale utilizzato da Arduino per consentire ai suoi micro-controllori di comunicare con altri dispositivi.

Esso prevede l'utilizzo di due fili di comunicazione: uno per la trasmissione dei dati (SDA) e l'altro per il segnale di clock (SCL). Il protocollo I2C utilizza un approccio "a due vie", dove i dispositivi possono inviare e ricevere dati in entrambe le direzioni.

Inoltre, questo protocollo prevede l'utilizzo di un indirizzo univoco per ogni dispositivo connesso alla linea di comunicazione, consentendo a più dispositivi di essere connessi alla stessa linea senza conflitti di indirizzo.

Nel nostro progetto questo protocollo è largamente usato, per una questione di compattezza, nel senso che i pin di Arduino Nano 33 IoT non sarebbero bastati per gestire ogni periferica.

Quindi usiamo una scheda d'espansione PCF8574 per ogni locker door; ad esso colleghiamo il sensore di fine-corsa e il relay.

Successivamente, le 6 schede d'espansione PCF8574, e il display LCD, vengono collegate ad un'unica scheda d'espansione TCA9548A che, a sua volta, viene collegata alle porte A4 e A5 di Arduino.

4.1.6 Alimentazioni

L'alimentazione di corrente è essenziale per far funzionare la maggior parte dei dispositivi elettrici, dai semplici elettrodomestici alle grandi macchine industriali.

La fornitura di corrente affidabile è fondamentale per garantire il corretto funzionamento di questi dispositivi, e per evitare danni costosi o potenzialmente pericolosi. Inoltre, la corretta alimentazione della corrente può anche contribuire a ridurre i costi energetici e a migliorare l'efficienza degli impianti elettrici.

Quando si tratta di installare prese elettriche all'esterno, la protezione dalle intemperie diventa un fattore ancora più critico. La pioggia, la neve, il vento e il sole possono tutti causare danni alle prese elettriche, che potrebbero interrompere l'alimentazione elettrica e danneggiare i dispositivi connessi. Per proteggere le prese elettriche all'esterno dalle intemperie, ci sono alcune misure che possono essere adottate.

In primo luogo, è importante scegliere prese elettriche specificamente progettate per l'uso esterno. Queste prese sono solitamente realizzate con materiali resistenti alle intemperie, come il PVC o l'acciaio inossidabile, e sono dotate di coperchi protettivi per impedire l'ingresso di acqua e polvere. Inoltre, è possibile installare prese elettriche con grado di protezione IP, che garantiscono una maggiore protezione contro l'acqua e la polvere.

In secondo luogo, è importante proteggere le prese elettriche esterne con dispositivi di protezione, come le scatole di derivazione o le cassette di protezione. Questi dispositivi possono essere utilizzati per proteggere le prese dalle intemperie, ma anche per prevenire danni causati da urti o cadute accidentali.

Il locker è dotato di una sola presa d'alimentazione, che deve essere posizionata sul retro dell'armadietto, e opportunamente riparata dalla pioggia o da altre intemperie.

Tale presa è collegata internamente ad un trasformatore, che trasforma la corrente da 220V AC a 5V DC; ed esso svolge un ruolo duplice, ovvero alimenta, utilizzando una presa usb-micro, direttamente Arduino Nano 33 IoT, ed è connesso ad ogni altra periferica che necessita di alimentazione; in particolare abbiamo creato una linea sulla scheda forata con la

5V e un'altra, parallela, con la GND. Quindi, ogni periferica che deve essere alimentata, è connessa a queste due linee.

Nello specifico, tali periferiche sono:

- 6 relay;
- 6 schede d'espansione TCA9548A;
- 1 scheda d'espansione PCF8574;
- 6 elettromagneti, ma solo a massa (GND);
- 1 schermo LCD.

4.2 Progettazione del back-end

In questa sezione tratteremo la progettazione del backend, un componente cruciale per il funzionamento di qualsiasi applicazione web o sistema software complesso.

Analizzeremo, innanzitutto, la struttura e l'architettura del backend, esplorando i vari modelli utilizzati per organizzare e gestire i dati, le logiche di business e le interazioni con i frontend e altri servizi.

Successivamente, procederemo con una descrizione dettagliata di ciascuno dei componenti principali, quindi il server, il database, l'API e i middleware. Approfondiremo poi il ruolo di ciascun componente, esaminando le sue caratteristiche, funzionalità e il modo in cui contribuiscono all'intero sistema.

Infine, dedicheremo una parte significativa alla comprensione di come tutti questi componenti collaborano tra loro per raggiungere l'obiettivo finale. Esamineremo i flussi di dati, le interazioni tra i vari servizi e le strategie di integrazione utilizzate per garantire che il backend funzioni in modo efficiente e senza intoppi.

Questa analisi ci aiuterà a cogliere l'importanza della coesione e del coordinamento tra i vari elementi del backend, evidenziando come una progettazione attenta e ben strutturata possa migliorare significativamente l'efficacia e l'affidabilità del sistema nel suo complesso.

4.2.1 Struttura generale

Un backend basato su Firebase con Node.js e TypeScript è strutturato per pubblicare API che facilitano l'interazione tra l'applicazione e l'hardware.

Firebase fornisce un ambiente serverless, gestendo l'hosting, l'autenticazione, il database in tempo reale e le funzioni cloud. Node.js, invece, offre il runtime per l'esecuzione di codice JavaScript lato server, mentre TypeScript aggiunge un livello di tipizzazione statica che migliora la robustezza e la manutenibilità del codice.

Le API sono create come funzioni cloud di Firebase, scritte in TypeScript, rispondendo alle richieste HTTP. Esse si occupano della logica di business interagendo con Firestore. Questo sistema è progettato per essere scalabile, sicuro e altamente responsivo, garantendo una comunicazione fluida tra il frontend, l'hardware e il backend.

Tali API sono pubbliche, e quindi accessibili dai dispositivi terminali, che possono essere gli smartphone con sistema operativo iOS o Android, attraverso l'app appositamente realizzata; esse, però, sono anche accessibili da altri dispositivi, quali l'armadietto da noi realizzato.

Questa interconnessione è resa possibile dalla rete Internet, che diventa, quindi, un requisito fondamentale per i dispositivi che devono ususfruire del servizio.

4.2.2 Architettura del back-end

Scendiamo più in profondità ed analizziamo dettagliatamente la componente di back-end.

Per farlo, iniziamo con la visualizzazione del diagramma dell'architettura, molto utilizzato nell'ambito dell'ingegneria del software, perché ci permette di fare una panoramica generale sulle relazioni di tutte le componenti fisiche che intervengono durante la fruizione del programma.

Nella Figura 4.16 viene riportato il diagramma dell'architettura fisica del nostro back-end.

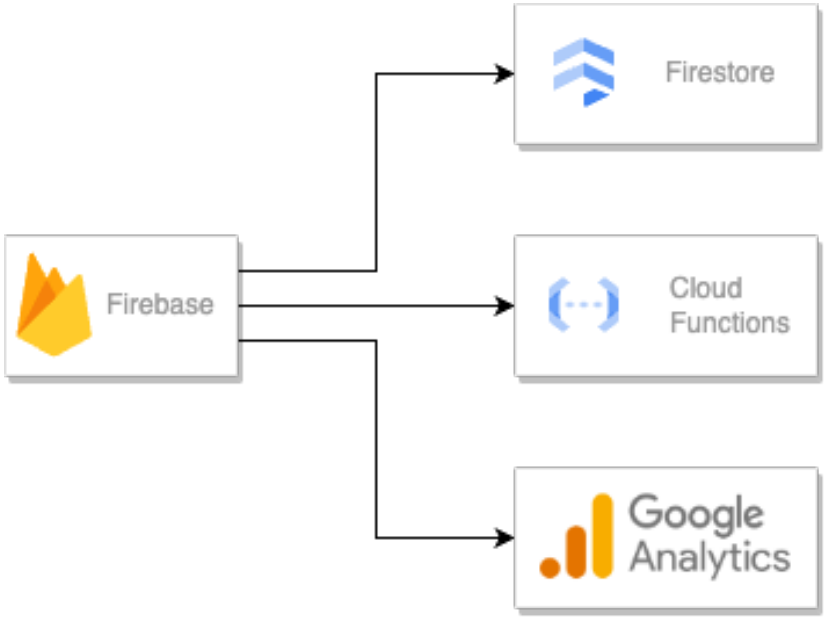


Figura 4.15: Diagramma dell'architettura del back-end

4.2.3 Firebase

Gran parte del lavoro viene svolto da Firebase. Quest'ultimo consiste in una piattaforma di sviluppo web e mobile nata nel 2011 e successivamente acquisita da Google nel 2014. Esso offre numerosi servizi semplicemente con l'implementazione di poche righe di codice all'interno del proprio progetto. È possibile integrare Firebase in siti web, in app Android o iOS e in giochi.

Firebase offre la possibilità di semplificare numerosi aspetti che spesso per i programmatori rappresentano un ostacolo.

A ciò si aggiunge l'importante opzione di analisi dei dati per reagire in modo coerente alle impressioni degli utenti aumentandone il flusso o servendosi della pubblicità per trarne profitto.

Per realizzare ciò, Firebase fornisce a noi sviluppatori diversi pacchetti con svariate funzionalità; di esse, per ora, ne utilizziamo solo cinque, ma in futuro, ci piacerebbe espandere la nostra conoscenza approfondendone altre.

Firebase Cloud Firestore

Cloud Firestore è un database flessibile e scalabile per lo sviluppo di dispositivi mobili, Web e server da Firebase e Google Cloud.

Esso mantiene i dati sincronizzati tra le app client tramite listener in tempo reale e offre supporto offline per dispositivi mobili e Web in modo da poter creare app reattive che funzionano indipendentemente dalla latenza di rete o dalla connettività Internet.

Cloud Firestore offre inoltre una perfetta integrazione con altri prodotti Firebase e Google Cloud, incluse le funzioni Cloud.

Qui risiedono i dati relativi a utenti, prodotti, ordini e altro. Cloud Firestore non offre un DBMS relazionale, bensì un DBMS noSQL, il che, dal nostro punto di vista può essere preferibile, in quanto consente una maggior scalabilità e una notevole flessibilità.

Firebase Functions

Cloud Functions for Firebase è un framework serverless che consente di eseguire automaticamente il codice back-end in risposta agli eventi attivati dalle funzionalità Firebase e dalle richieste HTTPS.

Il codice JavaScript o TypeScript viene archiviato nel cloud di Google e viene eseguito in un ambiente gestito.

Non è necessario gestire e ridimensionare i propri server.

Tale strumento, per il quale abbiamo dovuto sottoscrivere un abbonamento a pagamento, anche se per una cifra molto bassa, è proprio quello che ci consente di ridimensionare le immagini, come abbiamo visto per Firebase Storage.

Firebase ci offre, inoltre, svariate funzionalità già implementate; quindi, possiamo semplicemente aggiungerle al nostro progetto, dopo averne modificato i principali parametri, senza dover scrivere alcuna riga di codice.

Molte delle implementazioni future avverranno proprio qui, dal momento che, per la gestione dei pagamenti, ci sono delle collaborazioni con Stripe, un servizio di fama mondiale che consente, per l'appunto, di effettuare transazioni.

Google Analytics

Google Analytics è una soluzione di misurazione delle app o dei siti web, disponibile gratuitamente, che fornisce informazioni dettagliate sul loro utilizzo e sul coinvolgimento degli utenti.

Al centro di Firebase c'è Google Analytics, una soluzione di analisi illimitata disponibile gratuitamente. Analytics si integra tra le funzionalità di Firebase e fornisce rapporti illimitati per un massimo di 500 eventi distinti, ampiamente sufficienti per i nostri obiettivi.

I rapporti di analisi ci aiutano a capire chiaramente come si comportano i nostri utenti, il che ci consente di prendere decisioni in merito all'utilizzo dell'app e dell'armadietto, oltre che all'ottimizzazione delle prestazioni.

4.2.4 Elaborazione dati

Abbiamo, dunque, visto come viene progettata l'architettura delle componenti fisiche per il nostro software; sarà, ora, fondamentale capire come questi provider comunicheranno tra loro.

Analizzeremo nel prossimo capitolo i dettagli, oltre agli aspetti relativi alla sicurezza.

Per il momento possiamo, però, dire che Firebase, per ciascuno dei suoi servizi, mette a disposizione delle API (Application Programming Interface), ottime perché possono essere richiamate direttamente da codice, in modo asincrono.

La documentazione di queste API è ottima, ben strutturata e lascia poco spazio ai dubbi.

Il tutto, come vedremo in seguito, è realizzato utilizzando il linguaggio di programmazione denominato TypeScript (TS), usato, quindi, sia per il front-end che per il back-end.

Avendo già utilizzato i servizi di Firebase per altri software sviluppati in precedenza, seppur usando altri linguaggi di programmazione, ci è risultato molto facile e intuitivo utilizzare nuovamente tali funzioni.

CAPITOLO 5

Progettazione del front-end

Oggigiorno, gran parte delle operazioni che si svolgono all'interno di un'azienda, vengono registrate per poter essere monitorate ed analizzate.

Diventa fondamentale, quindi, avere sempre a disposizione un portale in cui inserire, automaticamente, se possibile, i dati, le azioni compiute e i risultati ottenuti da ogni singola operazione.

Esistono diversi tipi di portali, come, ad esempio, le applicazioni per desktop, i siti web, ed, ultimamente, le applicazioni per smarthphone.

Questo fenomeno accade perché ci stiamo abituando ad avere sempre tutto a portata di mano, in qualsiasi posto e in qualsiasi momento della giornata.

Cavalcando questa onda, abbiamo deciso che, per l'accesso, il nostro sistema utilizzi un'applicazione per smartphone, che può anche essere usufruita nella versione di web app.

Nel presente capitolo tratteremo della progettazione del software che compone il sistema complessivo.

Per diversi motivi, che discuteremo in seguito, abbiamo scelto di realizzare un'applicazione mobile, semplice ed intuitiva, che il responsabile della ferramenta può utilizzare per gestire gli ordini e monitorare le operazioni del locker, e, volendo, anche di ogni locker door.

5.1 Struttura generale

Ad oggi, prima di progettare un'applicazione front-end, è necessario, come prima attività, determinare il dispositivo e il sistema operativo in cui tale software dovrà funzionare.

Questa scelta è molto importante, perché per ogni dispositivo target si possono scegliere diversi framework, pattern e linguaggi di programmazione che non sarebbero disponibili se il dispositivo scelto fosse stato un altro.

Tale limite è molto forte, e per questo motivo, a volte accade l'inverso, ovvero il progettista o il programmatore sceglie il dispositivo di deploy in base ai framework o ai linguaggi di programmazione che conosce meglio.

Per non porci ostacoli, abbiamo deciso di utilizzare un nuovo framework, denominato Flutter, che, a differenza delle altre tecniche viste precedentemente, consente di sviluppare, con un unico codice, lo stesso software per più dispositivi e sistemi operativi differenti.

Nel nostro caso, scrivendo un solo codice abbiamo creato un'applicazione mobile per il sistema operativo di casa Apple denominato iOS, per Android e anche per i browser web, rendendo, quindi, possibile, per il responsabile della ferramenta, avere sempre a disposizione una piattaforma per dialogare con il locker.

5.2 Architettura del software

La struttura del nostro software si basa sul framework di Flutter, che è organizzato a strati, i quali conferiscono una netta separazione delle responsabilità di ciascun modulo, oltre a garantire una maggior scalabilità.

In particolare, possiamo vedere Flutter sotto tre aspetti differenti, ovvero il framework, l'engine e l'embedder.

Il framework è completamente scritto in Dart, un linguaggio di programmazione che riesce a compilare sia in codice nativo sia in JavaScript, garantendo altissime prestazioni in termini di velocità, e anche tempi di sviluppo ridotti rispetto ad altri linguaggi classici. Tale framework rende disponibili le principali API per la costruzione dei widget, per la gestione dello stato ma anche del rendering dell'interfaccia utente.

L'engine, invece, è implementato in un linguaggio più a basso livello, ovvero il C++, e gestisce il rendering, le animazioni, la composizione di grafica e le interazioni di basso livello con la piattaforma.

Infine, l'embedder è la componente che gestisce il collegamento tra Flutter e la piattaforma nativa su cui viene eseguito il codice, che può essere iOS, macOS, Android, Windows, Linux o un browser web.

Dal punto di vista, invece, dell'architettura, abbiamo, anche in questo caso, utilizzato il pattern MVC, già descritto nel capitolo precedente, ma implementato con alcune differenze dettate dalle caratteristiche del front-end.

Nella Figura 5.1 viene rappresentata l'architettura di Flutter.

5.3 Flutter

Flutter è un toolkit dell'interfaccia utente cross-platform pensato per consentire il riutilizzo del codice su differenti sistemi operativi, consentendo, inoltre, alle applicazioni sviluppate con esso, di interfacciarsi direttamente con i servizi della piattaforma nativa.

L'obiettivo è permettere agli sviluppatori di realizzare applicazioni ad alte prestazioni che risultino adatte a piattaforme diverse, considerando certamente le differenze laddove esistono e condividendo, però, quanto più codice possibile.

Durante lo sviluppo, le app Flutter vengono eseguite in una macchina virtuale che offre l'aggiornamento e la visualizzazione istantanei delle modifiche, senza bisogno di una ricompilazione completa di tutto il codice.

Per il rilascio, le app Flutter vengono compilate direttamente nel codice macchina, siano esse istruzioni Intel x64 o ARM, o in JavaScript, se destinate al Web.

5.3.1 Framework

Il framework di Flutter è open source, con una licenza BSD permissiva e dispone di un ecosistema di pacchetti di terze parti che integrano le funzionalità della libreria principale.

Abbiamo già visto come funziona l'architettura di Flutter nella sezione precedente, quindi di seguito analizzeremo quali sono le novità introdotte da Flutter, seguite dalle principali motivazioni che ci hanno spinto ad usarlo.

In particolare, considereremo in dettaglio i seguenti aspetti:

- anatomia di un'app;
- interfacce utente reattive;
- widget;

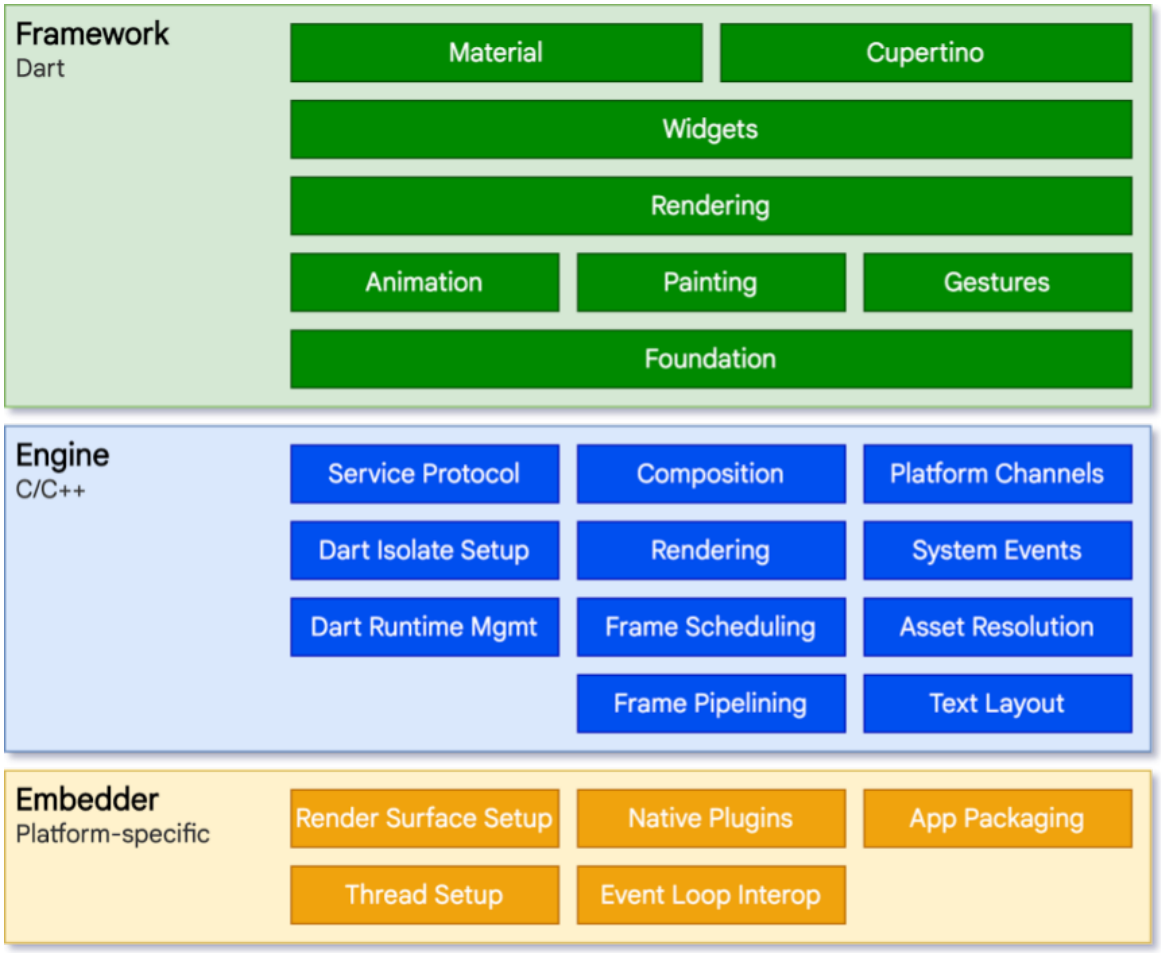


Figura 5.1: Rappresentazione dell'architettura di Flutter

- processo di rendering;
- integrazione con altro codice;
- supporto per il web.

Anatomia di un'app

La Figura 5.2 fornisce una panoramica dei componenti di un'app Flutter creata con il comando 'flutter create'.

Essa illustra la posizione del Flutter engine all'interno di questa pila, mette in evidenza quali sono i confini dell'API ed elenca le sezioni in cui risiedono i vari componenti.

È evidente come l'architettura di un'app preveda la presenza di 5 sezioni, che sono:

- *app Dart*: compone i widget nell'interfaccia utente e implementa la logica aziendale, viene realizzata dallo sviluppatore;
- *framework*: fornisce API al livello superiore per creare app di alta qualità; compone l'albero dei widget dell'app in una scena;
- *engine*: è il responsabile della rasterizzazione di scene composite e fornisce l'implementazione di basso livello delle API;

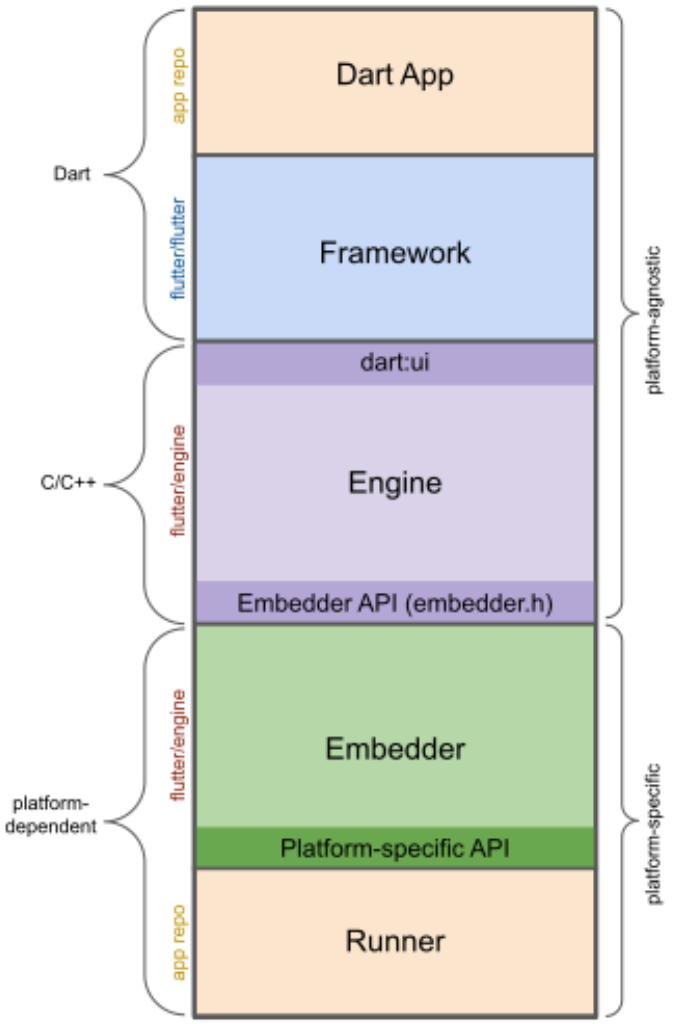


Figura 5.2: Raffigurazione delle componenti di un'app Flutter

- *embedder*: serve per coordinare il sistema operativo del dispositivo con i servizi quali il rendering, l'accessibilità e l'input;
- *runner*: compone l'API specifica della piattaforma dell'embedder in un pacchetto di app che è eseguibile direttamente sul dispositivo.

Interfacce utente reattive

Dal punto di vista dell'utente, Flutter è un framework con UI reattiva e dichiarativa. In esso, lo sviluppatore definisce un mapping tra lo stato dell'applicazione e lo stato dell'interfaccia utente; una volta fatto ciò, Flutter si occupa di aggiornare l'interfaccia durante l'esecuzione non appena lo stato dell'applicazione cambia.

Questo modello prende ispirazione dal framework React di Facebook, che anch'esso ha rivoluzionato molti dei principi di design tradizionali.

Nei framework UI tradizionali, invece, lo stato iniziale dell'interfaccia utente viene definito una sola volta e poi aggiornato separatamente dal codice dell'utente durante l'esecuzione, in risposta agli eventi.

Una delle sfide di questo approccio è che, con l'aumentare della complessità dell'applicazione, lo sviluppatore deve tenere traccia di come i cambiamenti di stato influenzano l'interfaccia utente.

Widget

Flutter mette decisamente in risalto i widget come unità fondamentali di composizione grafica. Essi rappresentano i componenti principali dell'interfaccia utente di un'app Flutter; in particolare ciascun widget rappresenta una dichiarazione immutabile di una parte dell'interfaccia utente.

I widget formano una gerarchia basata sulla composizione. Ogni widget è annidato all'interno del suo genitore e può ricevere contesto da esso; questa struttura si estende fino al widget radice, che ospita l'intera app; generalmente si tratta del MaterialApp o del CupertinoApp.

Flutter implementa internamente ogni controllo dell'interfaccia utente, invece di utilizzare quelli forniti dal sistema operativo.

Tale approccio offre una serie di numerosi vantaggi, tra cui l'estensibilità illimitata, migliori prestazioni e l'indipendenza dal sistema operativo.

Processo di rendering

Flutter, per effettuare il rendering, invoca prima il codice Java del framework Android. Le librerie di sistema Android forniscono i componenti necessari per disegnare su un oggetto Canvas, che viene poi renderizzato da Android tramite Skia, un motore grafico scritto in C/C++ che utilizza la CPU o la GPU per completare il disegno sul dispositivo.

I framework cross-platform tradizionali creano, di solito, un livello di astrazione sopra le librerie dell'interfaccia utente native dei principali sistemi operativi nativi, cercando di uniformare le differenze tra le rappresentazioni delle due piattaforme.

Il codice delle app, spesso scritto in un linguaggio interpretato come JavaScript, deve interagire con le librerie di sistema iOS, basate su Swift, o Android, basate su Kotlin, per visualizzare l'interfaccia utente.

Questo processo, però, introduce un sovraccarico significativo, soprattutto se c'è una frequente interazione tra l'interfaccia utente e la logica dell'applicazione.

Flutter, invece, minimizza queste astrazioni evitando di utilizzare le librerie di widget dell'interfaccia utente di sistema, preferendo un proprio insieme di widget.

Il codice Dart utilizzato per disegnare le immagini in Flutter viene compilato in codice nativo, che utilizza Skia per realizzare il rendering.

Flutter include la propria copia di Skia come parte del suo motore, permettendo agli sviluppatori di aggiornare le proprie app con gli ultimi miglioramenti delle prestazioni, indipendentemente dagli aggiornamenti del sistema operativo del dispositivo.

Possiamo vedere graficamente la pipeline del processo di rendering appena descritto nella Figura 5.3.

Integrazione con altro codice

Flutter offre anche dei meccanismi di interoperabilità tra diversi linguaggi, permettendo l'accesso al codice e ad API scritte in Kotlin o in Swift, la chiamata ad API native basate su C, l'integrazione di controlli nativi in un'app Flutter e l'inclusione di Flutter stesso all'interno di un'applicazione già esistente.

Nella Figura 5.4 vediamo come Flutter riesce ad operare sia con le API di una tipica app nativa, sia essa Android o iOS.

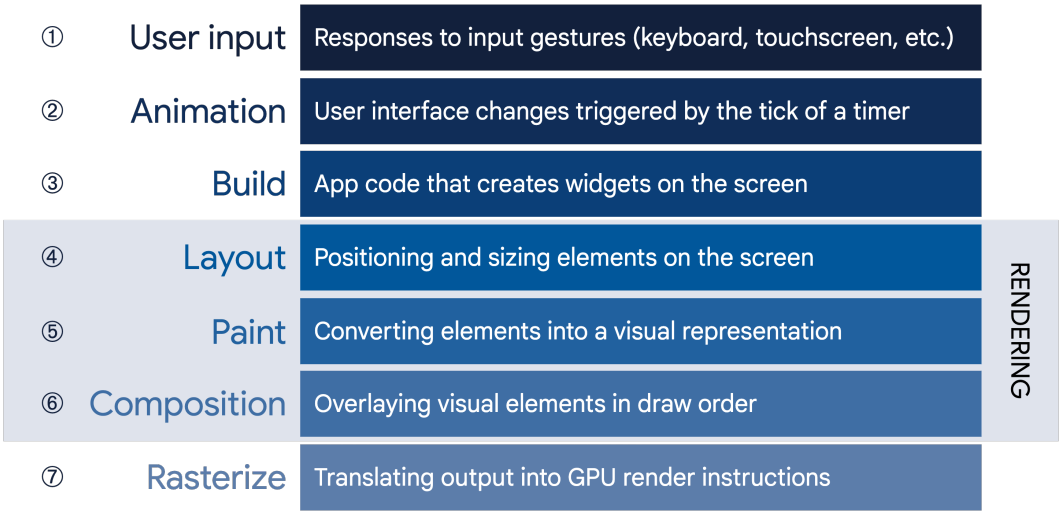


Figura 5.3: Raffigurazione della pipeline di rendering di un'app Flutter

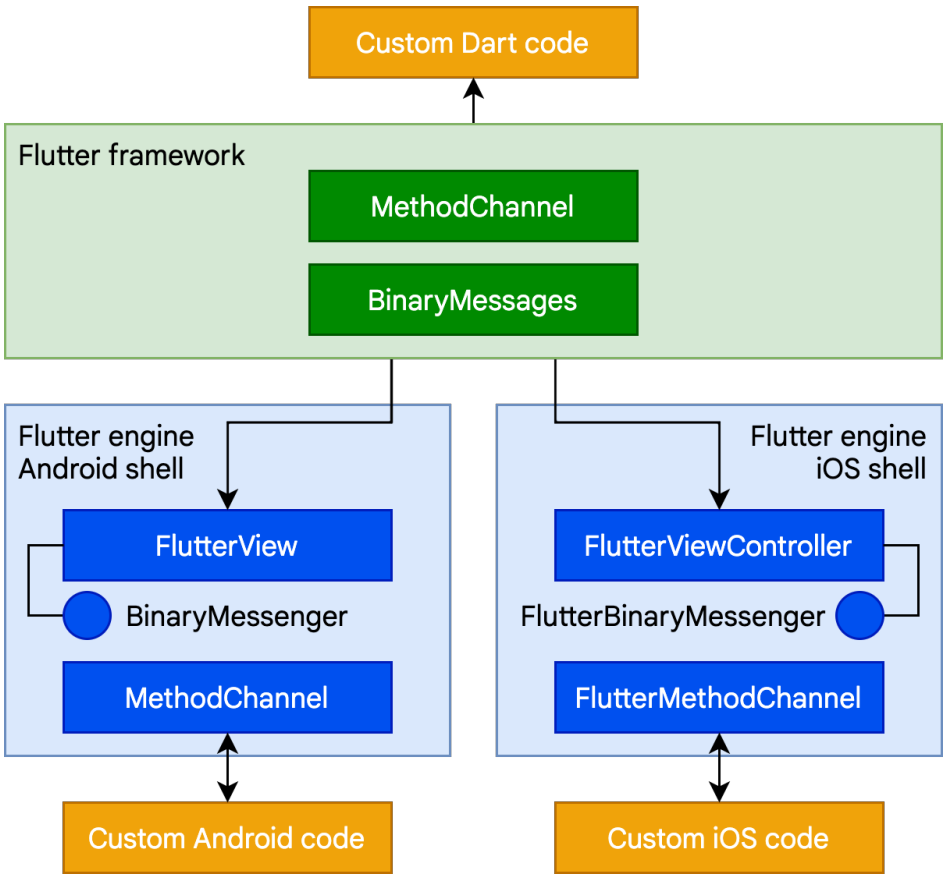


Figura 5.4: Schema di interoperabilità con Android e iOS di Flutter

Supporto per il web

Poiché il framework Flutter è scritto in Dart, la sua compilazione in JavaScript è relativamente semplice.

L'engine di Flutter, infatti, è scritto in C++, ed è progettato per interfacciarsi con il sistema operativo sottostante piuttosto che con un browser web.

Di conseguenza, serve un approccio diverso per il web; quindi, per il rendering dei

contenuti, Flutter esegue una reimplementazione dell'engine che usufruisce delle API tipiche dei web browser.

Esso utilizza HTML, CSS, Canvas ed SVG per il rendering, per ottenere un codice molto leggero e velocemente interpretabile dai browser web.

Nella Figura 5.5 possiamo vedere com'è strutturata l'architettura di supporto per il web di Flutter.

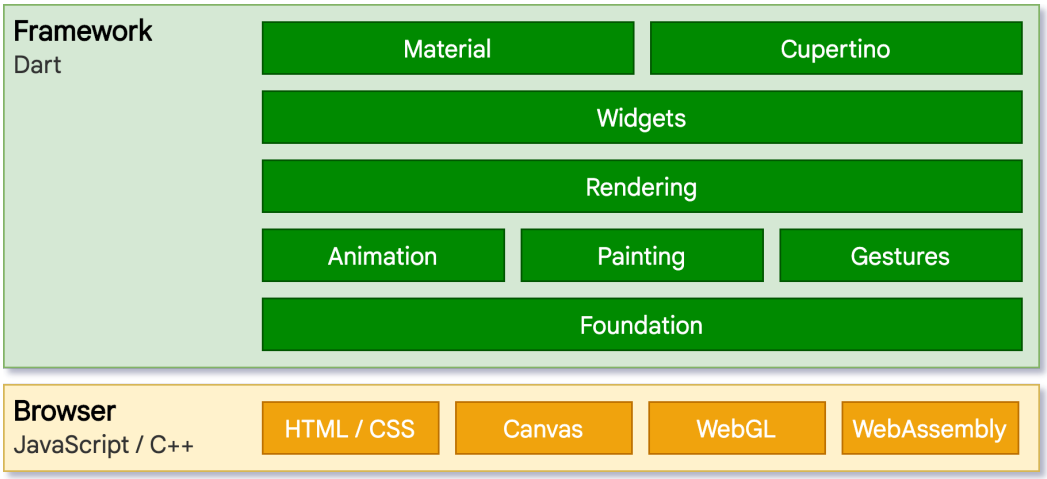


Figura 5.5: Raffigurazione dell'architettura di supporto al web di Flutter

5.3.2 Architettura

Come anticipato, nella nostra applicazione di front-end abbiamo deciso di usare, come pattern, il classico Model-View-Controller, noto per la sua netta divisione delle responsabilità, la facilità di manutenzione e la scalabilità.

Nonostante il pattern venga definito a livello universale, ad oggi, in realtà, ogni tipologia di progettazione del software fornisce una sua interpretazione, che può variare leggermente dal prototipo globale.

Nel nostro caso abbiamo implementato il pattern Model-View-Controller considerando che l'appicazione debba essere utilizzata in maniera veloce e intuitiva, quindi dando molta priorità alla sua estetica e alla sua semplicità.

Di conseguenza, la componente più sviluppata di tutta l'app è, sicuramente, la view, mentre i controller si limitano a effettuare chiamate API al server e i model corrispondono alle classi del nostro programma, che sono molto simili, quasi un mirroring, alle classi che troviamo nel sistema del server.

Nella prossima sezione seguente analizzeremo nel dettaglio le modalità con cui sono state e progettate le tre diverse componenti del pattern MVC, tenendo, ovviamente, conto delle considerazioni fatte precedentemente.

Model

Nel progetto dell'applicazione mobile, il model corrisponde alle classi del programma, e serve per creare una rappresentazione su Dart del suo gemello che è stato, invece, implementato nel server.

Pertanto, abbiamo un model contenuto in una cartella di progetto che si chiama *models* ed è costituito dalle classi:

1. *locker_order.dart*;

2. *opening.dart*.

Il documento *locker_order.dart* contiene la classe relativa all'ordine effettuato telefonicamente o sul sito di e-commerce della ferramenta, e che richiede che i prodotti vengano posizionati all'interno del locker.

Il file *opening.dart*, invece, presenta la classe relativa ad ogni movimento di uno sportello, che sia esso di apertura o di chiusura.

Controller

Il controller della nostra app risulta, come da progetto, essere molto limitato; infatti si limita ad effettuare chiamate API al server, come vedremo nella sezione successiva, su richiesta della view, e quindi dell'utente.

Per rispettare il pattern MVC, abbiamo creato, dunque, la cartella denominata *controllers*, che, al suo interno, ha:

- *locker_orders.dart*;
- *openings.dart*.

Questi due file contengono i metodi CRUD (Create, Read, Update e Delete) per leggere, creare, aggiornare ed eliminare, rispettivamente, un locker order e un opening, che sia esso di apertura o di chiusura.

Ciascuno dei metodi sopra elencati lavora direttamente con le istanze delle classi che abbiamo descritto nella sottosezione precedente, in modo da rendere la comunicazione più completa ed efficace.

View

La componente view, ovvero l'insieme delle viste grafiche, è, sicuramente, la più sviluppata all'interno del nostro progetto.

Ciò è vero sia perché l'app è formata da molte viste, sia perché, come anticipato, in Flutter la componente view incorpora anche delle funzioni che, in altri modelli MVC, apparterrebbero al controller.

Ciò è dovuto al fatto che i widget, ovvero i componenti grafici alla base di Flutter che abbiamo descritto precedentemente, sono in grado di interagire con l'utente, con, ad esempio, il click, ma anche con altre gesture meno note.

Le viste della nostra applicazione si trovano all'interno della cartella chiamata *views*; esse sono:

1. *orders.dart*;
2. *add_order.dart*;
3. *update_order.dart*;
4. *opening.dart*;
5. *send_code.dart*.

Senza entrare troppo nel dettaglio di ogni vista, come invece faremo nel capitolo successivo, possiamo affermare che esse rappresentano l'interfaccia grafica che viene utilizzata dal responsabile della ferramenta.

Inoltre, sfruttando pienamente il principio della composizione dei widget di Flutter, abbiamo anche creato una cartella *components*, interna a *views*, che contiene i widget che vengono utilizzati all'interno di altre schermate.

In particolare, tale cartella contiene il file *door.dart*, che corrisponde al widget di rappresentazione grafica di uno sportello.

Main

Infine vediamo il file *main.dart*, necessario per la creazione di ogni progetto Dart. Questo file contiene una classe, inizializzata di default durante la creazione del nostro progetto, la quale è l'unico punto di accesso di tutta l'app.

È responsabilità del main, dunque, invocare la home page, e creare i collegamenti alle pagine seguenti.

Possiamo vedere la struttura della cartella *lib* del nostro progetto Flutter nella Figura 5.6.

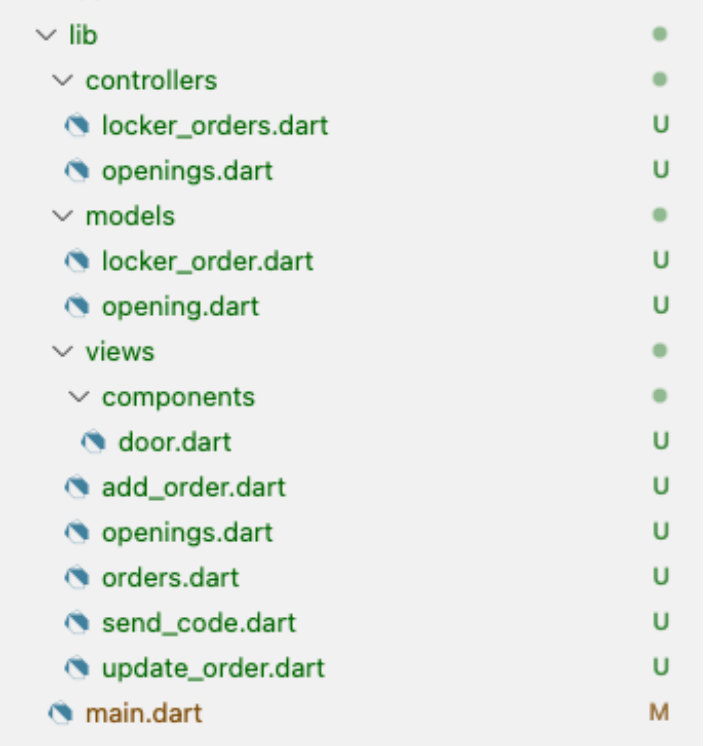


Figura 5.6: Struttura della cartella *lib* del progetto

5.4 Connessione al server

La nostra applicazione deve, obbligatoriamente, mantenere una comunicazione attiva e costante con il server, che abbiamo già descritto precedentemente.

Per realizzare ciò, tramite il controller dell'applicazione, effettuiamo delle chiamate RESTful all'API che abbiamo reso pubblica dal server di cui disponiamo.

Tale server è di proprietà di Firebase, e ci ha fornito un collegamento ipertestuale tramite il quale possiamo effettuare le nostre chiamate, ottenendo, ove necessario, anche le risposte.

La Figura 5.7 mostra la base del percorso assoluto dell'API, da cui si possono comporre le chiamate ad essa.

```
'https://us-central1-ferramenta-berti.cloudfunctions.net/app/
```

Figura 5.7: Base del percorso assoluto dell'API

Appendendo il nome della funzione dell'API a questo link, ed effettuando una chiamata riusciamo a comunicare in maniera sicura con il server, che, infatti, dispone del protocollo HTTPS.

Tale protocollo, il cui acronimo è Hypertext Transfer Protocol Secure, è fondamentale per la protezione delle comunicazioni su Internet.

Esso consiste nell'evoluzione, sicura, dell'HTTP, arricchito dall'aggiunta del protocollo SSL/TLS (Secure Socket Layer/Transport Layer Security) per cifrare i dati che vengono trasmessi nella rete, la quale, ricordiamo, è di pubblico dominio.

Questa cifratura impedisce che le informazioni siano intercettate o manipolate durante il loro trasferimento tra il client e il server.

HTTPS utilizza un vasto sistema di certificati digitali, che autenticano l'identità dei siti web, garantendo ai client che il sito a cui stanno accedendo è legittimo e non una replica fraudolenta.

Le chiamate, nel nostro sistema, possono essere effettuate mediante 4 possibili metodi, ovvero "GET", "POST", "PUT" e "DELETE"; in base anche al metodo scelto, si ottiene la risposta dal server.

Il server restituisce al client chiamante un numero che corrisponde allo stato della risposta stessa; tale numero può rappresentare un evento avvenuto con successo oppure, in alternativa, un errore.

Ogni errore è sempre seguito dal messaggio che lo descrive, e ciò è utile sia all'utente finale per poter capire dove ha sbagliato, sia a noi, in qualità di sviluppatori, per eseguire in maniera più professionale la fase di debug.

Nel caso, invece, di un evento avvenuto con successo, la risposta del server contiene sia lo stato, come abbiamo appena visto, sia altre meta-informazioni, sia i dati restituiti, eventualmente, dal nostro server, che poi il controller elabora e converte in dati utili per il funzionamento di tutta l'app.

La comunicazione bilaterale tra il client, sia essa l'applicazione mobile, la web app o l'armadietto, e il server avviene, quindi, mediante un passaggio di dati, i quali contengono le informazioni strettamente necessarie per eseguire le funzionalità desiderate.

Tali dati vengono strutturati mediante un formato chiamato JSON (JavaScript Object Notation) che è molto leggero e ampiamente utilizzato per la sua semplicità e facilità di lettura sia per gli umani che per le macchine.

JSON è un formato standard indipendente dal linguaggio, supportato, ormai, da quasi la totalità delle tecnologie di programmazione.

Nella Figura 5.8 mostriamo un esempio di una chiamata API al nostro server, simulata mediante il software applicativo Postman.

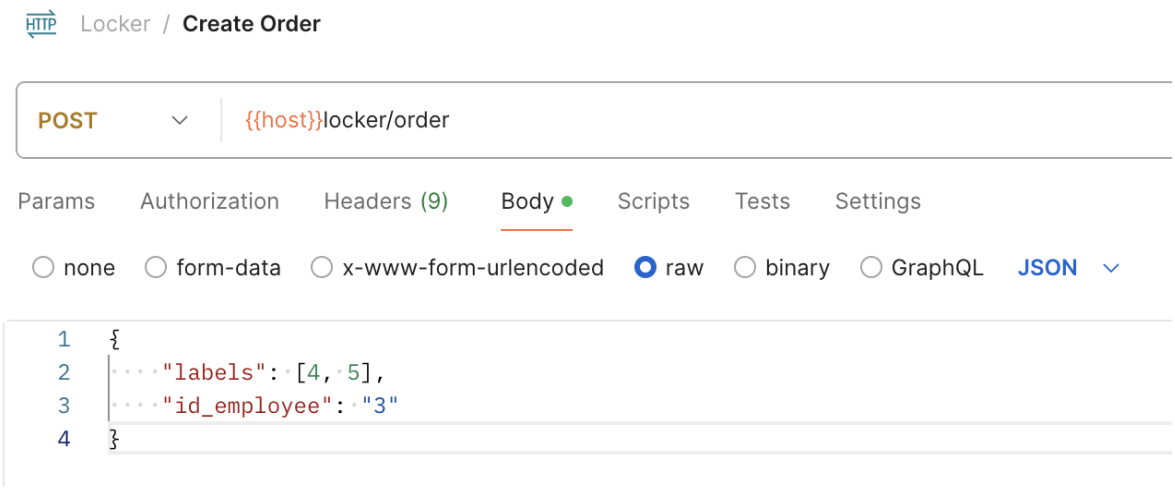


Figura 5.8: Simulazione di una chiamata all'API del server da Postman

CAPITOLO 6

Implementazione

In questo capitolo vedremo, in dettaglio, la parte meno progettuale, ma più pragmatica, del nostro lavoro.

L'implementazione, nel nostro ambito, consiste nella programmazione di codice sorgente e nella costruzione di un circuito elettrico, il cui scopo è quello di fornire all'utente finale un'esperienza gradevole, intuitiva, e, soprattutto, funzionale.

Analizziamo, quindi, come è stato codificato il tutto; in particolare, vediamo come il programma è stato diviso nelle sue parti.

6.1 Implementazione dell'hardware

In questa sezione considereremo esclusivamente la componente hardware del sistema realizzato, e vedremo, scendendo nei particolari, la sua implementazione.

Per svolgere tale attività, abbiamo cercato di seguire, un passo dopo l'altro, gli schemi realizzati durante la progettazione, incontrando, inevitabilmente, alcuni problemi che hanno ostacolato il nostro cammino, ma che, applicando al meglio le nostre conoscenze interdisciplinari, abbiamo saputo superare con successo.

6.1.1 Firmware

Come anticipato nel capitolo relativo alla progettazione dell'hardware, il nostro sistema è architetturalmente paragonabile ad una stella, con il centro stella che corrisponde al dispositivo chiamato Arduino.

Tale Arduino è, quindi, il *cervello* di tutto il nostro meccanismo, e per svolgere le funzioni ha bisogno di essere programmato da degli sviluppatori.

Inizia, quindi, la fase in cui si descrive com'è avvenuto il processo per il quale si è istruito Arduino per svolgere le attività per cui è stato progettato.

Come primo passo, abbiamo dovuto scaricare ed installare il compilatore ufficiale di Arduino, chiamato Arduino IDE (Integrated Development Environment), rilasciato dalla stessa azienda produttrice dell'hardware. Tale software si presenta in due versioni, la 1.* e la 2.*, più moderna e veloce, e proprio per tali motivi quest'ultima è stata scelta da noi per lo sviluppo.

Questa piattaforma è molto *minimal* ed efficiente, e presenta, come possiamo vedere nella Figura 6.1, diverse funzionalità, che sono:

Figura 6.1: Panoramica delle funzionalità di Arduino IDE 2

- *verify / upload*: compila e carica il codice sulla scheda Arduino;
- *seleziona scheda e porta*: le schede Arduino rilevate vengono visualizzate in automatico in questa combo box, insieme al numero di porta;
- *sketchbook*: sezione in cui viene visualizzata la lista dei progetti personali, è collegato ad una specifica cartella locale;
- *gestore delle schede*: sfoglia i pacchetti Arduino e di terze parti che possono essere installati;
- *gestore delle librerie*: sfoglia migliaia di librerie Arduino, realizzate sia da Arduino che dagli utenti della community;
- *debugger*: esegue test e debug dei programmi, in tempo reale;
- *cerca*: aiuta a trovare una parola all'interno di tutti i file del programma;
- *monitor e plotter seriali*: apre gli strumenti di visualizzazione.

Per poter caricare uno *sketch* sulla nostra scheda Arduino, i passi da eseguire, dopo aver collegato fisicamente la *board* al computer, sono la verifica e il caricamento.

Il codice sorgente è scritto in un linguaggio molto simile al C++, che prende, anch'esso, il nome di Arduino. Tale codice può essere compilato in qualsiasi momento, ed effettua semplicemente le operazioni di controllo degli errori e di compilazione.

Il caricamento del codice prevede, automaticamente, una prima fase di verifica, a seguito della quale, in caso di esito positivo, viene installato il codice sorgente scritto direttamente sulla scheda Arduino connessa e selezionata, la quale inizierà ad eseguirlo in tempo reale.

Ogni *sketch* di Arduino si compone nativamente di due funzioni. La prima, chiamata *Setup*, esegue il corpo contenuto al suo interno una sola volta, non appena viene data corrente al dispositivo. La seconda, invece, che prende il nome di *Loop*, come ci suggerisce il nome, inizia non appena finisce la fase di *Setup* ed è l'equivalente di un ciclo infinito, ovvero del codice seguente:

```
while(true){

}
```

Di conseguenza, la funzione si interrompe solo se viene tolta l'alimentazione al dispositivo.

Moduli

Prima di passare all'implementazione del codice, è necessario specificare che, per questioni di praticità, esso è stato diviso in ben 9 moduli, ognuno dei quali si focalizza su una caratteristica fisica del sistema complessivo. Questi moduli sono descritti nella Tabella 6.1, e verranno approfonditi nelle sottosezioni successive.

Modulo	Descrizione
Locker.ino	File principale, contiene le funzioni di *setup* e di *loop*.
HTTPS.h	Ha, al suo interno, tutte le primitive per dialogare con il server.
Keys.h	Presenta le funzioni necessarie all'attivazione e all'uso del tastierino.
LCD.h	Possiede le funzionalità di inizializzazione e scrittura sul display LCD.
Locker.h	È la classe che rappresenta uno sportello fisico.
LockerOrder.h	È una classe che modella un Locker Order.
WiFiNet.h	Contiene le primitive per connettersi ad una rete WiFi.
Communications.h	Gestisce tutte le comunicazioni tra i dispositivi che fanno uso del protocollo I2C.
trust_anchors.h	File auto-generato del certificato utile per l'uso del protocollo HTTPS.

Tabella 6.1: Moduli di cui si compone il nostro software

Locker.ino Si tratta del file principale, che mette insieme ogni altro modulo del programma complessivo. Inizializza le variabili globali ed esegue, nell'ordine prestabilito, le funzioni di *setup* e di *loop*. Contiene al suo interno anche altre funzioni atte a completare le attività richieste, le quali sono molto generali e non possono essere inserite, a loro volta, in un altro modulo. Possiamo vedere la sezione di importazione delle librerie e dei moduli e della dichiarazione delle variabili globali nella Figura 6.2.

HTTPS.h In questo file sono contenute le due funzioni principali che il modulo WiFi presente sulla scheda Arduino usa per effettuare le chiamate all'API che abbiamo realizzato, e queste due funzioni sono:

```
bool connectToServer();

String call(String method, String path, String body);
```

In particolare la funzione *call* è fondamentale per poter comunicare con il server al quale si è connessi, per poi ottenerne, validare ed utilizzarne la risposta.

Possiamo vedere il codice sorgente di HTTPS.h nella Figura 6.3.

Keys.h In questo header file sono contenute solo due funzioni, di inizializzazione del tastierino e di lettura; esse sono:

```
void setupKeyPad();

char keyPressed();
```

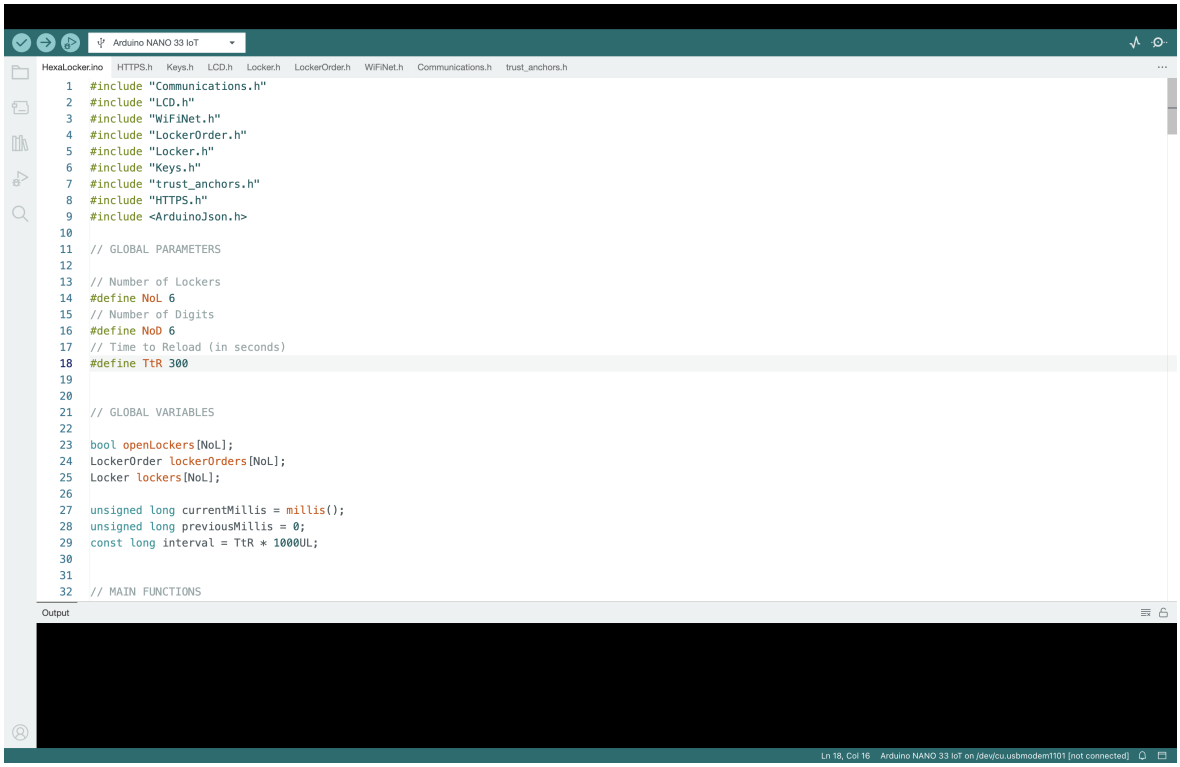


Figura 6.2: Implementazione del modulo Locker.ino

In particolare la funzione *keyPressed* è utile per capire quando il relativo sportello è aperto o chiuso.

Possiamo vedere il codice sorgente di Keys.h nella Figura 6.4.

LCD.h In questo header file sono contenute tre funzioni, per inizializzare l'LCD e per scrivere su di esso, anche nel caso di messaggi che occupano due righe, di cui il nostro display è provvisto; queste funzioni sono:

```
void setupLCD();

String* splitMessage(String message)

void printLCD(String message, bool isLong);
```

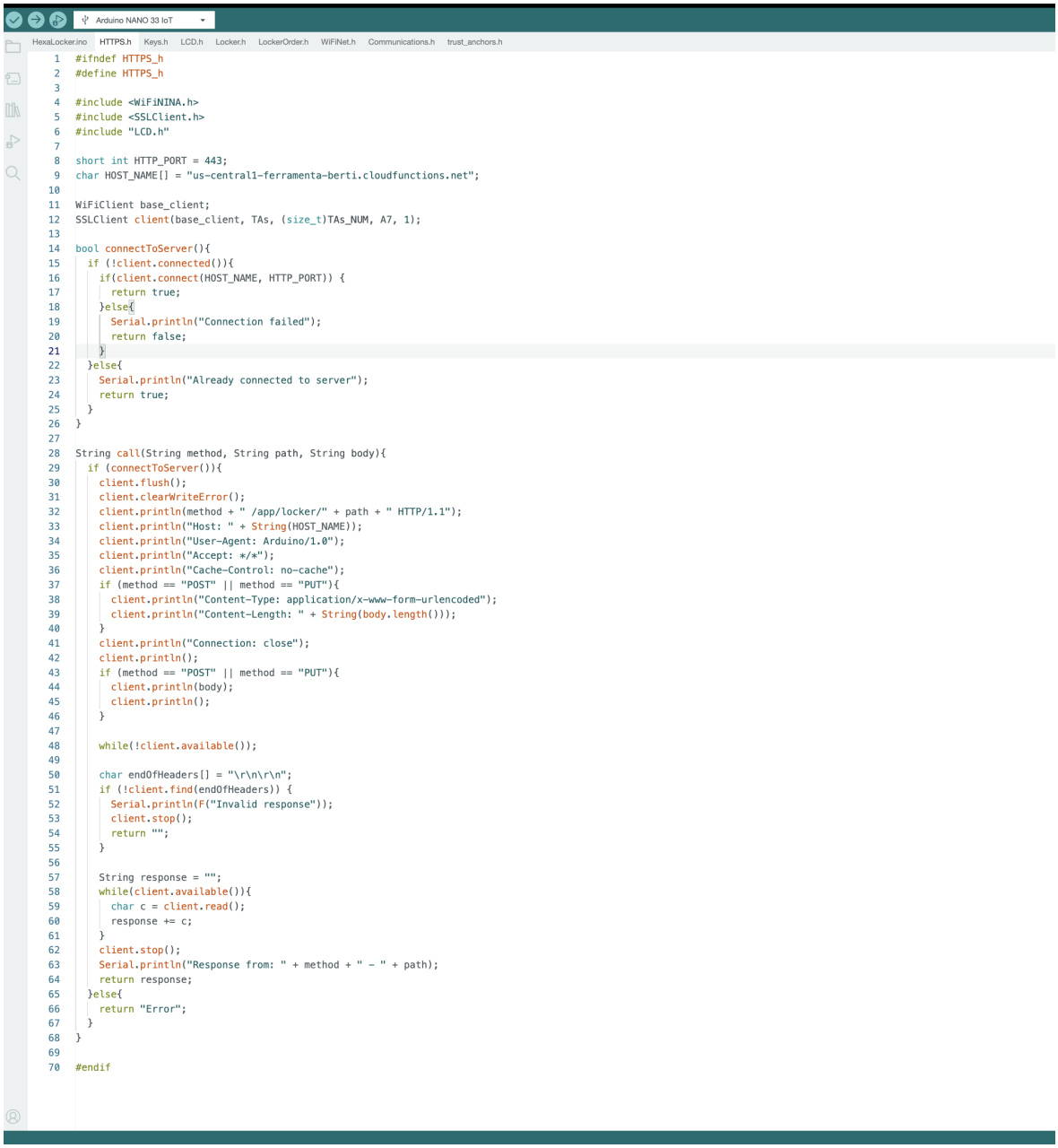
In particolare la funzione *printLCD* prevede, oltre al messaggio da stampare, anche un valore booleano che, se impostato a vero dal chiamante, fa restare più a lungo la scritta. Si usa questa opzione nei messaggi più importanti.

Possiamo vedere il codice sorgente di LCD.h nella Figura 6.5.

Locker.h Questa classe rappresenta uno sportello fisico dell'armadietto, è costituita dal metodo costrotture, i metodi *getter* e *setter*, implementati per questioni di sicurezza, e da un metodo per poter aprire o chiudere la serratura.

Possiamo vedere il codice sorgente di Locker.h nella Figura 6.6.

LockerOrder.h La classe LockerOrder rappresenta un ordine, in maniera similare a come lo è nell'app o nel server, e presenta il metodo costruttore, i metodi *getter* e *setter* e svariate



```
#ifndef HTTPS_h
#define HTTPS_h

#include <WiFiNINA.h>
#include <SSLClient.h>
#include "LCD.h"

short int HTTP_PORT = 443;
char HOST_NAME[] = "us-central1-ferramenta-berti.cloudfunctions.net";

WiFiClient base_client;
SSLClient client(base_client, TAs, (size_t)TAs_NUM, A7, 1);

bool connectToServer(){
  if (!client.connected()){
    if(client.connect(HOST_NAME, HTTP_PORT)) {
      return true;
    }else{
      Serial.println("Connection failed");
      return false;
    }
  }else{
    Serial.println("Already connected to server");
    return true;
  }
}

String call(String method, String path, String body){
  if (connectToServer()){
    client.flush();
    client.clearWriteError();
    client.println(method + " /app/locker/" + path + " HTTP/1.1");
    client.println("Host: " + String(HOST_NAME));
    client.println("User-Agent: Arduino/1.0");
    client.println("Accept: */*");
    client.println("Cache-Control: no-cache");
    if (method == "POST" || method == "PUT"){
      client.println("Content-Type: application/x-www-form-urlencoded");
      client.println("Content-Length: " + String(body.length()));
    }
    client.println("Connection: close");
    client.println();
    if (method == "POST" || method == "PUT"){
      client.println(body);
      client.println();
    }

    while(!client.available());

    char endOfHeaders[] = "\r\n\r\n";
    if (!client.find(endOfHeaders)) {
      Serial.println(F("Invalid response"));
      client.stop();
      return "";
    }

    String response = "";
    while(client.available()){
      char c = client.read();
      response += c;
    }
    client.stop();
    Serial.println("Response from: " + method + " - " + path);
    return response;
  }else{
    return "Error";
  }
}

#endif
```

Figura 6.3: Implementazione del modulo HTTPS.h

funzioni per ottenere, controllare la presenza o rimuovere un'etichetta, oltre ad un metodo per ottenere la lista degli sportelli a partire dall'ordine stesso.

Possiamo vedere il codice sorgente di LockerOrder.h nella Figura 6.7.

WiFiNet.h Questo header file possiede la funzione che consente ad Arduino di connettersi ad una rete WiFi, anche se protetta con password, la quale ovviamente deve essere nota. Questa funzione verifica la presenza di una rete e si connette, altrimenti mostra un messaggio di errore, e ritenta. L'unica funzione che possiede è:

```
void connectToWiFi();
```

Possiamo vedere il codice sorgente di WiFiNet.h nella Figura 6.8.

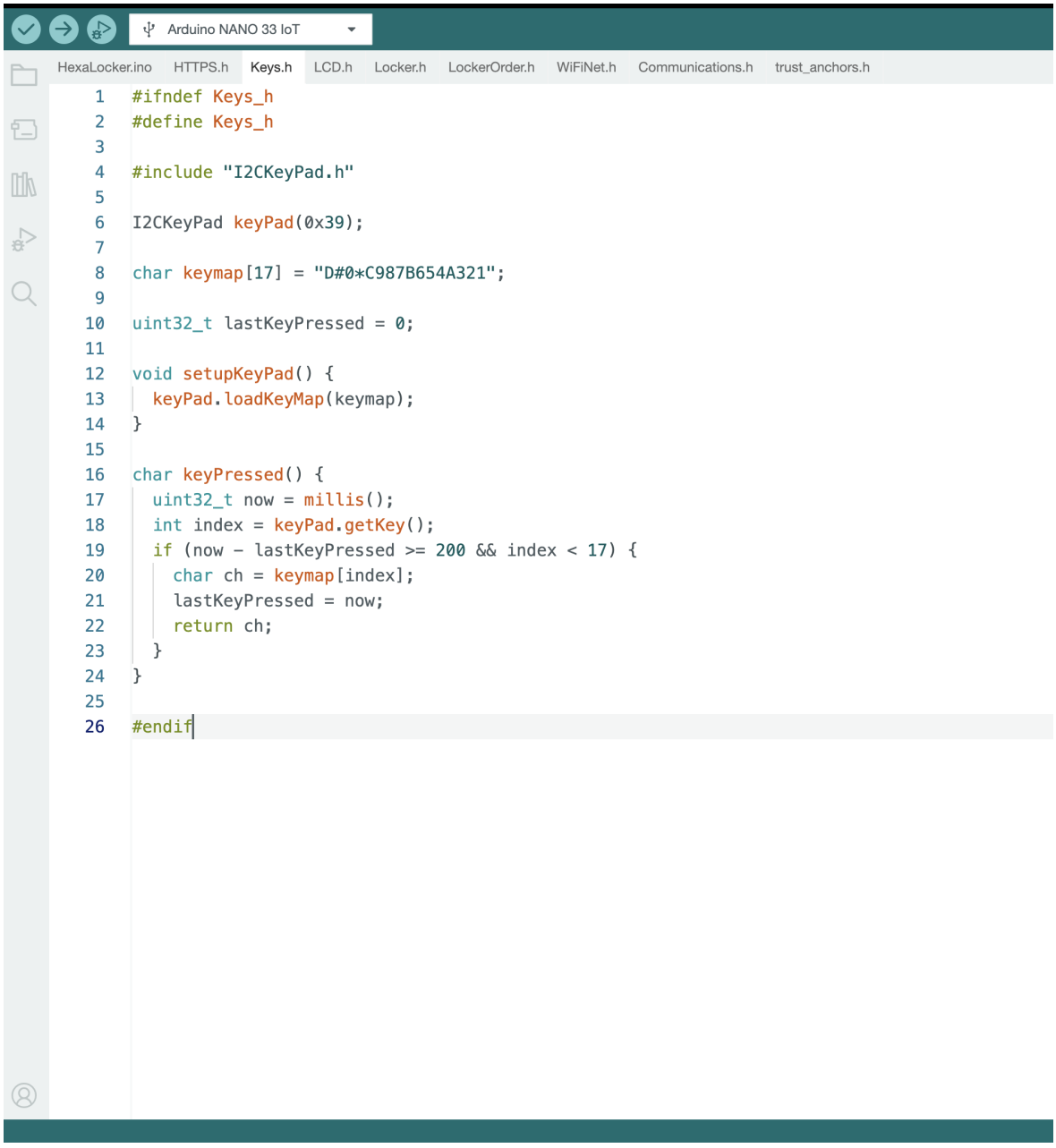


Figura 6.4: Implementazione del modulo Keys.h

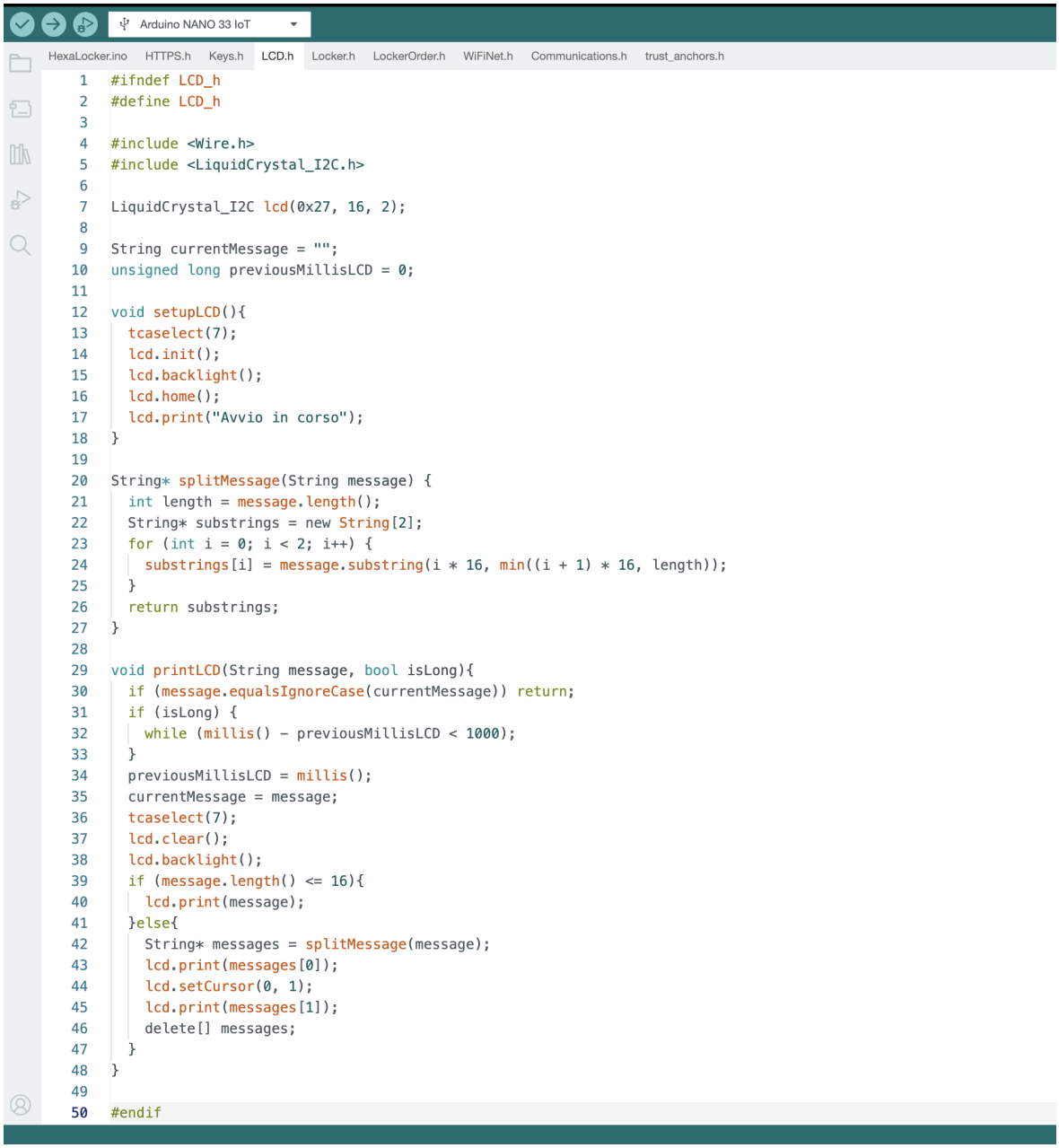
Communications.h Questo file gestisce le comunicazioni tra i dispositivi che fanno uso del protocollo I2C; in particolare, utilizza in maniera attiva il modulo TCA9584A, selezionando con quale dispositivo PCF8574 comunicare, dato che se ne può attivare solo uno alla volta. Le funzioni implementate sono, dunque:

```
setupCommunications;

void tcaselect(uint8_t i);
```

In particolare la funzione *tcaselect* riceve un numero tra 1 e 8, che rappresenta l'indice dello sportello con cui comunicare. Il funzionamento, nel dettaglio, verrà descritto nella sottosezione successiva.

Possiamo vedere il codice sorgente di Communications.h nella Figura 6.9.




```
#ifndef LCD_h
#define LCD_h

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

String currentMessage = "";
unsigned long previousMillisLCD = 0;

void setupLCD(){
  tcaselect(7);
  lcd.init();
  lcd.backlight();
  lcd.home();
  lcd.print("Avvio in corso");
}

String* splitMessage(String message) {
  int length = message.length();
  String* substrings = new String[2];
  for (int i = 0; i < 2; i++) {
    substrings[i] = message.substring(i * 16, min((i + 1) * 16, length));
  }
  return substrings;
}

void printLCD(String message, bool isLong){
  if (message.equalsIgnoreCase(currentMessage)) return;
  if (isLong) {
    while (millis() - previousMillisLCD < 1000);
  }
  previousMillisLCD = millis();
  currentMessage = message;
  tcaselect(7);
  lcd.clear();
  lcd.backlight();
  if (message.length() <= 16){
    lcd.print(message);
  }else{
    String* messages = splitMessage(message);
    lcd.print(messages[0]);
    lcd.setCursor(0, 1);
    lcd.print(messages[1]);
    delete[] messages;
  }
}

#endif
```

Figura 6.5: Implementazione del modulo LCD.h

Setup

La funzione *setup* viene utilizzata per effettuare tutte le inizializzazioni dei dispositivi e dei moduli usati.

Si può trovare la sua implementazione nella Figura 6.10.

Come primo step, come si è soliti fare, si inizializza la comunicazione con il monitor seriale, che è molto utile soprattutto per la fase di debug del codice.

Dopodiché si prepara il modulo per le comunicazioni, e solo allora si possono connettere il tastierino e il display LCD.

A questo punto di effettua la connessione alla WiFi, necessaria per aggiornare il sistema scaricando i dati sugli ordini; dopo aver istanziato i 6 oggetti della classe Locker, viene chiusa ogni serratura per precauzione.

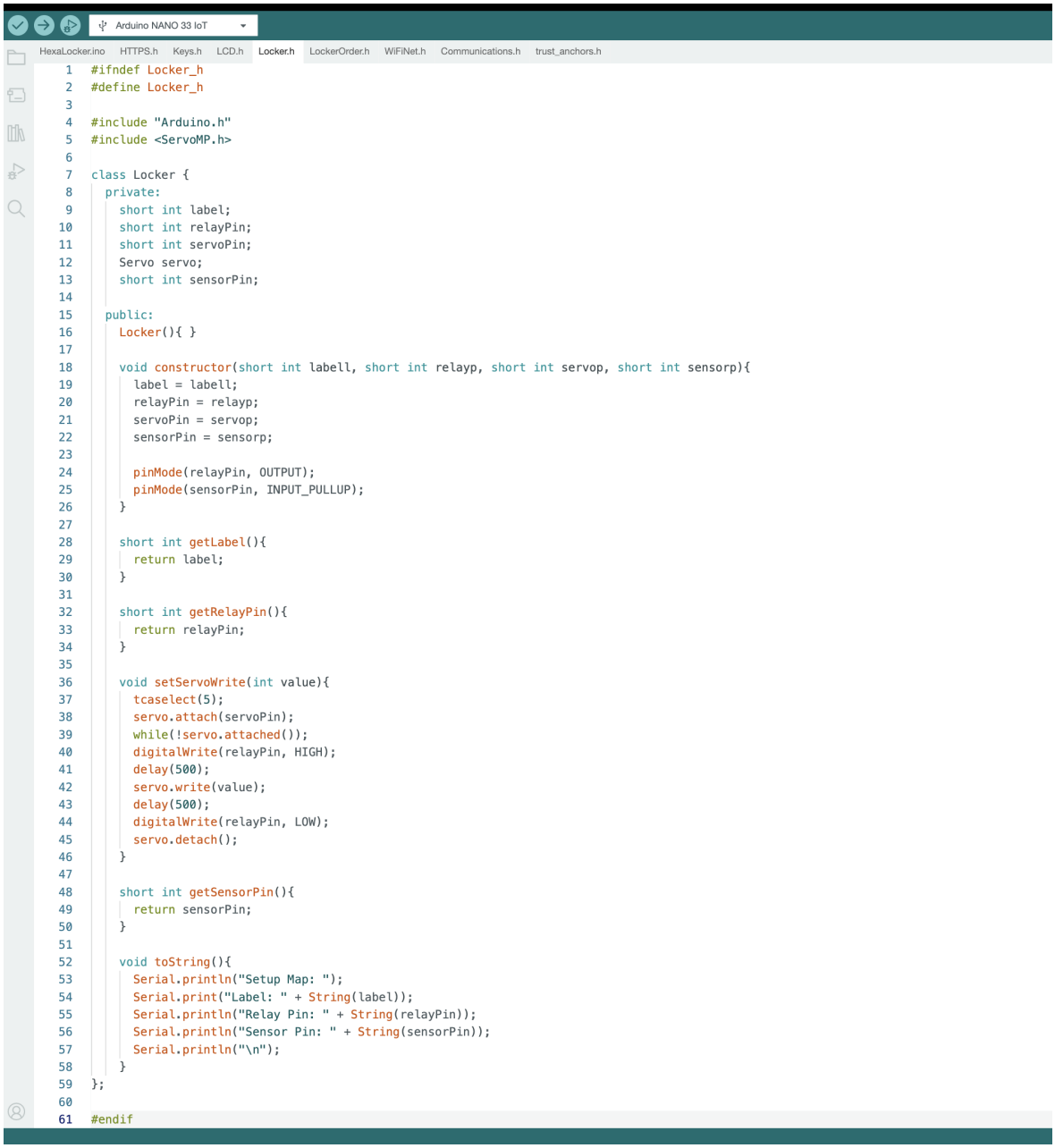


Figura 6.6: Implementazione del modulo Locker.h

Solo a tal punto, se la catena di operazioni è andata a buon fine, l'armadietto comunica, attraverso il display, che è tutto "Pronto!" e si passa alla fase di *loop*.

Loop

La funzione di *loop* consiste in un ciclo infinito, che può terminare solo se viene tolta l'alimentazione ad Arduino. Per questo motivo, in essa abbiamo implementato le funzioni, che sappiamo verranno eseguite una dopo l'altra, a catena, e periodicamente.

Ogni 5 minuti viene aggiornato il sistema scaricando i nuovi ordini, mentre ad ogni istante si controllano sia l'apertura o la chiusura di ogni armadietto, sia se l'utente sta digitando il codice.

Arduino NANO 33 IoT

HexaLocker.ino | HTTPS.h | Keys.h | LCD.h | Locker.h | LockerOrder.h | WiFiNet.h | Communications.h | trust_anchors.h

```
#ifndef LockerOrder_h
#define LockerOrder_h

#include "Arduino.h"

#define NoL 1

class LockerOrder {
  private:
    String id;
    int code;
    short int labels[NoL];

  public:
    LockerOrder() { }

    void constructor(String docid, int cod, short int labelsarray[]) {
      id = docid;
      code = cod;
      for (short int i = 0; i < NoL; i++) {
        labels[i] = labelsarray[i];
      }
    }

    String getId() {
      return id;
    }

    int getCode() {
      return code;
    }

    short int* getLabels() {
      return labels;
    }

    short int getLabel(short int index) {
      return labels[index];
    }

    bool hasLabel(short int label) {
      for (short int i = 0; i < NoL; i++) {
        if (labels[i] == label) {
          return true;
        }
      }
      return false;
    }

    void unsetLabel(short int label){
      for (short int i = 0; i < NoL; i++) {
        if (labels[i] == label){
          labels[i] = -1;
        }
      }
    }

    short int getNumberOfLockers() {
      short int c = 0;
      for (short int i = 0; i < NoL; i++) {
        if (labels[i] >= 0) {
          c++;
        }
      }
      return c;
    }

    void toString() {
      if (code > 0) {
        Serial.println("Locker Order: ");
        Serial.println("ID: " + id);
        Serial.println("Code: " + String(code));
        Serial.print("Labels: ");
        for (short int i = 0; i < NoL; i++) {
          if (labels[i] >= 0){
            Serial.print(String(labels[i]) + ", ");
          }
        }
        Serial.println("\n");
      }
    }
};

#endif
```

Figura 6.7: Implementazione del modulo LockerOrder.h

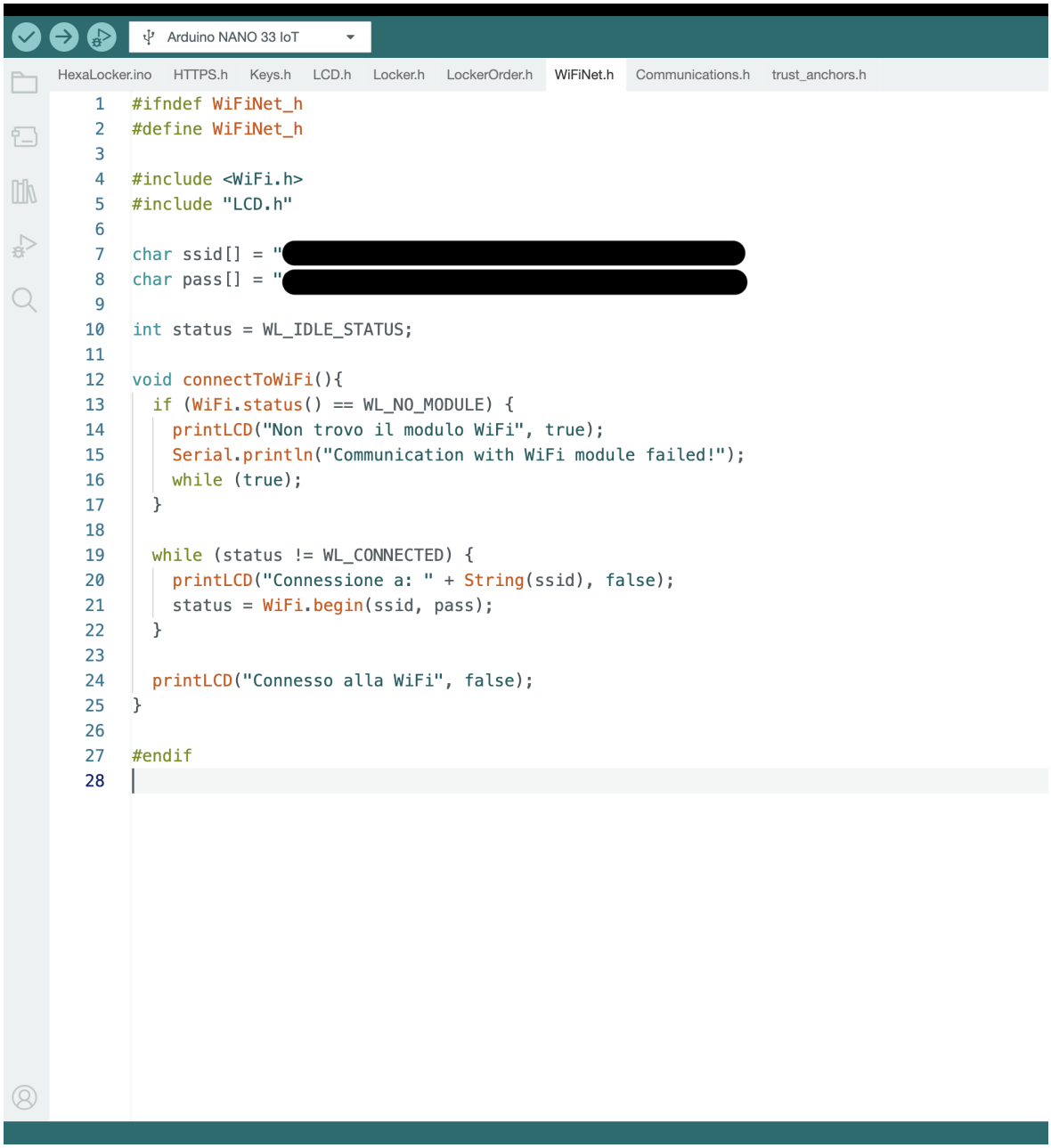


Figura 6.8: Implementazione del modulo WiFiNet.h

In quest'ultimo caso, il sistema entra in una modalità dedicata, in cui attende l'inserimento di 6 cifre, che, se avviene, è seguito da un controllo di validità del codice, che, a sua volta, se superato, provvede ad aprire gli sportelli appartenenti all'ordine avente il codice inserito.

Se il codice è errato, il sistema costringe l'utente ad attendere 10 secondi; questa precauzione è stata presa per evitare il *brute-forcing*.

Se l'utente non ha inserito 6 cifre, e passano più di 10 secondi dall'istante in cui ha inserito la sua ultima cifra, il sistema torna nel suo flusso di base.

Di default, il display del locker mostra la frase "Ciao! Inserisci il tuo Codice".

Possiamo trovare, nella Figura 6.11, il codice dell'implementazione della funzione di *loop*.

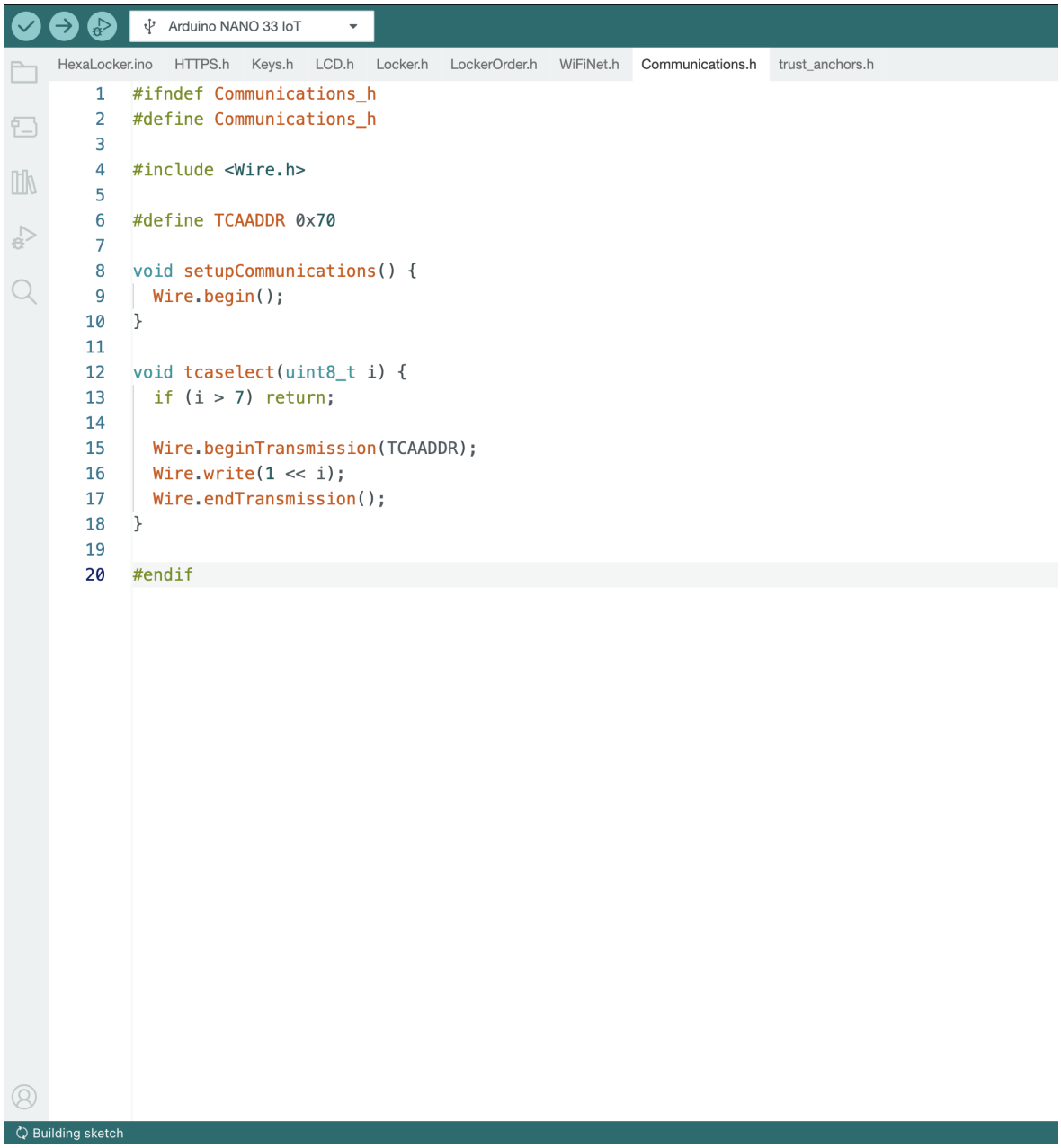


Figura 6.9: Implementazione del modulo Communications.h

6.1.2 Connessioni

Come abbiamo già visto nella fase progettuale, il nostro sistema ha bisogno di svariate connessioni, soprattutto tra Arduino e i diversi dispositivi di input ed output.

La scheda Arduino, appartenendo alla famiglia *Nano*, dispone di pochi pin; dunque abbiamo bisogno di un sistema che 'estenda' tale pinout.

Tra i protocolli per le comunicazioni seriali, che abbiamo preso in considerazione per questo scopo, ci sono l'SPI e l'I2C.

L'SPI sarebbe stata sicuramente una valida alternativa, avendo un'architettura di tipo *master-slave* ed un'alta velocità, ma, di contro, richiede più linee di comunicazione rispetto all'I2C, su cui, invece, abbiamo optato, non avendo bisogno di elevate velocità di comunicazione.

```
266  void setup() {
267    Serial.begin(9600);
268    setupCommunications();
269    setupKeyPad();
270    setupLCD();
271    connectToWiFi();
272    setupLockers();
273    closeLockers();
274    setupLockerOrders();
275    printLCD("Pronto!", false);
276  }
```

Figura 6.10: Implementazione della funzione di Setup

```
278  void loop() {
279    currentMillis = millis();
280    if (currentMillis - previousMillis >= interval) {
281      previousMillis = currentMillis;
282      setupLockerOrders();
283    }
284    printLCD("Ciao! Inserisci il tuo Codice", true);
285    checkOpenings();
286    checkClosures();
287    checkCode();
288  }
```

Figura 6.11: Implementazione della funzione di Loop

Il protocollo I2C (Inter-Integrated Circuit) è un bus seriale sincrono, progettato per permettere la comunicazione tra vari dispositivi integrati internamente ad un sistema. È ampiamente utilizzato in ambito elettronico ed è caratterizzato dalla sua semplicità e versatilità.

Il funzionamento è basato su 2 linee di comunicazione, che sono bi-direzionali e open-drain, quindi necessitano di resistori di pull-up. Le due linee di comunicazione sono:

- SDA (Serial DAta line): linea per il trasferimento dei dati;
- SCL (Serial CLock line): linea per il segnale di clock.

Anche il protocollo I2C usufruisce di un'architettura master slave; il master e lo slave, però, non sono fissi, ma, anzi, possono scambiarsi di ruolo nel tempo, anche sulla stessa linea del bus.

Questo principio è fondamentale nel nostro progetto, per poter coordinare sia i dispositivi di input, sia i dispositivi di output.

Dunque, ricordando che ogni sportello ha bisogno di un elettromagnete, un servo motore, un sensore di contatto ed un relay, abbiamo deciso di collegare ognuno di questi componenti, relativi ad ogni sportello, ad un dispositivo visto precedentemente quale il PCF8574, che riceve ed invia segnali ad Arduino proprio mediante il protocollo I2C.

A questo punto, però, notiamo che, dal momento che abbiamo 6 di questi dispositivi, non possiamo connetterli tutti ad Arduino, che dispone solo di una linea composta da SDA e SCL, grazie ai pin, rispettivamente, A4 e A5.

Di conseguenza, abbiamo utilizzato il componente TCA9548A, che è, in sostanza, un multiplexer I2C. Esso dispone di una linea per comunicare con Arduino e di 8 linee per

comunicare con altri dispositivi, sempre, ovviamente, usando il protocollo I2C. Nel nostro progetto utilizziamo tutte e 8 queste linee; in particolare, utilizziamo:

- 6 linee per i 6 PCF8574;
- 1 linea per il display LCD;
- 1 linea per il keypad.

Come abbiamo visto nella sottosezione dedicata al file *Communications.h*, abbiamo realizzato una funzione chiamata *tcaselect* che è atta a selezionare con quale dispositivo, tra gli 8 disponibili, Arduino deve comunicare.

6.2 Implementazione del back-end

In questa sezione approfondiamo la pipeline dell'implementazione del back-end del nostro sistema, avvenuta seguendo le attività descritte in fase di progettazione.

Conosciamo, quindi, come avviene il setup di un progetto su Firebase, il servizio che abbiamo usato per gestire il server; in particolare ci focalizziamo sulle sue componenti utilizzate, ovvero Cloud Firestore e Functions.

Infine, vediamo come sono stati realizzati i test sull'API risultante dall'implementazione.

6.2.1 Firebase

Firebase è la componente principale di back-end del nostro software; esso, infatti, svolge il ruolo di database, di storage, e molto altro.

La prima cosa da fare è stata quella di creare un progetto, che abbiamo nominato Ferramenta Berti, scegliendo un identificatore di progetto univoco.

La Figura 6.12 mostra i dettagli di un progetto.

Successivamente, abbiamo collegato il nostro sito web a questa piattaforma, attraverso un apposito metodo messo a disposizione da Firebase.

La Figura 6.13 mostra una fase della connessione tra un'app o un sito, e la piattaforma di Firebase.

Proseguendo, abbiamo connesso ciascuna componente al progetto.

6.2.2 Firebase Cloud Firestore

Firebase Cloud Firestore è un DBMS NOSQL,che memorizza i dati sotto forma di collezioni e documenti.

Abbiamo collocato il nostro database a Francoforte, in Germania, considerando sia l'aspetto economico, sia la sua centralità in Europa, dove risiede il 99% della clientela della Ferramenta.

La vicinanza, in software che offrono delle funzionalità live, cioè in diretta, è fondamentale, perché, al diminuire della latenza, aumenta l'efficienza e si riduce, quindi, la possibilità che due o più operazioni vengano prese in carico contemporaneamente, garantendone, quindi, la mutua esclusione.

I dati che abbiamo deciso di salvare all'interno di Firestore, e che, quindi, sono delle collection all'interno del database, sono:

- locker;
- locker orders;

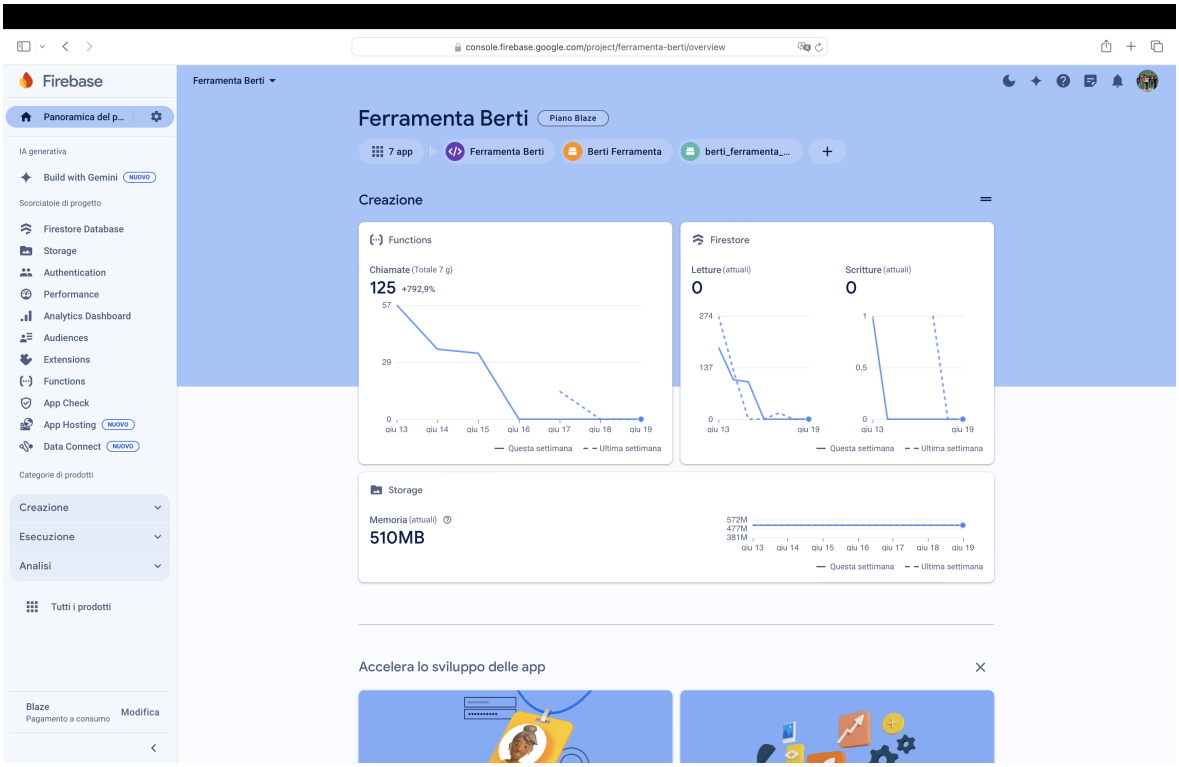


Figura 6.12: Home Page di un progetto su Firebase

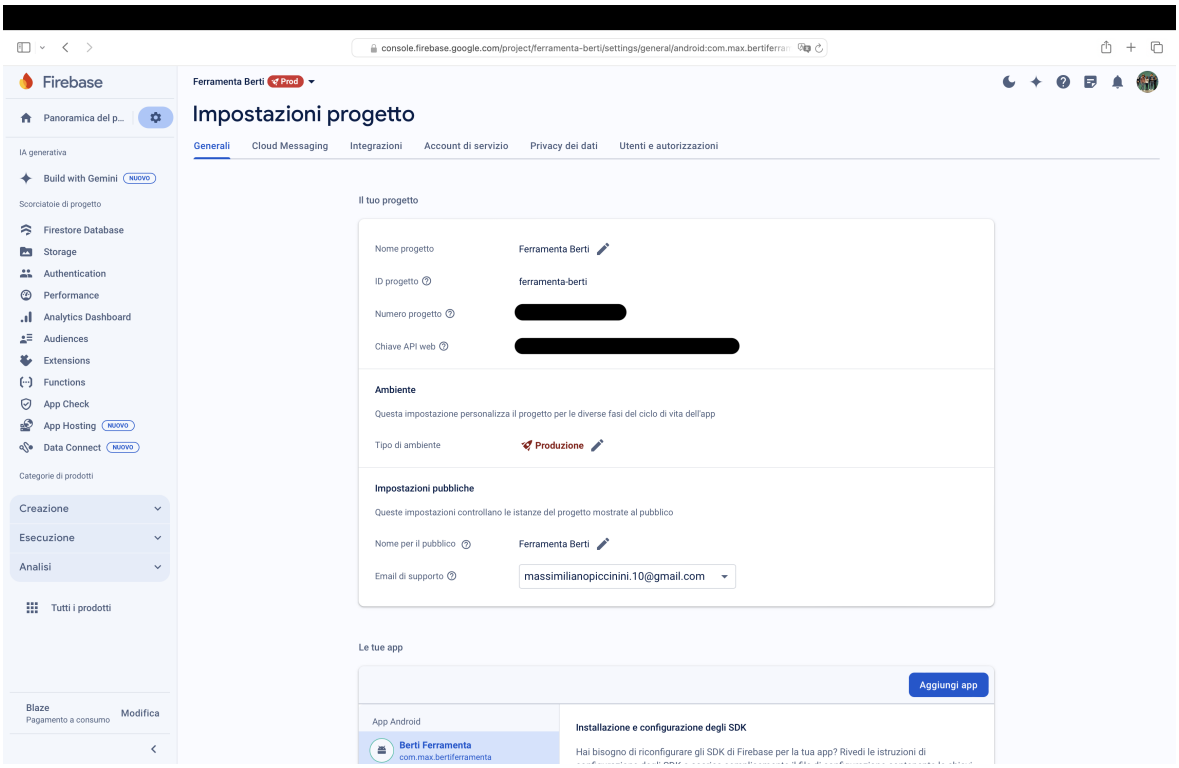


Figura 6.13: Impostazioni di un progetto su Firebase

- user locker orders.

Approfondiamo ora ciascuna di queste collection, per capire quali dati possono contenere;

esse sono illustrate nelle Figure 6.14, 6.15 e 6.16.

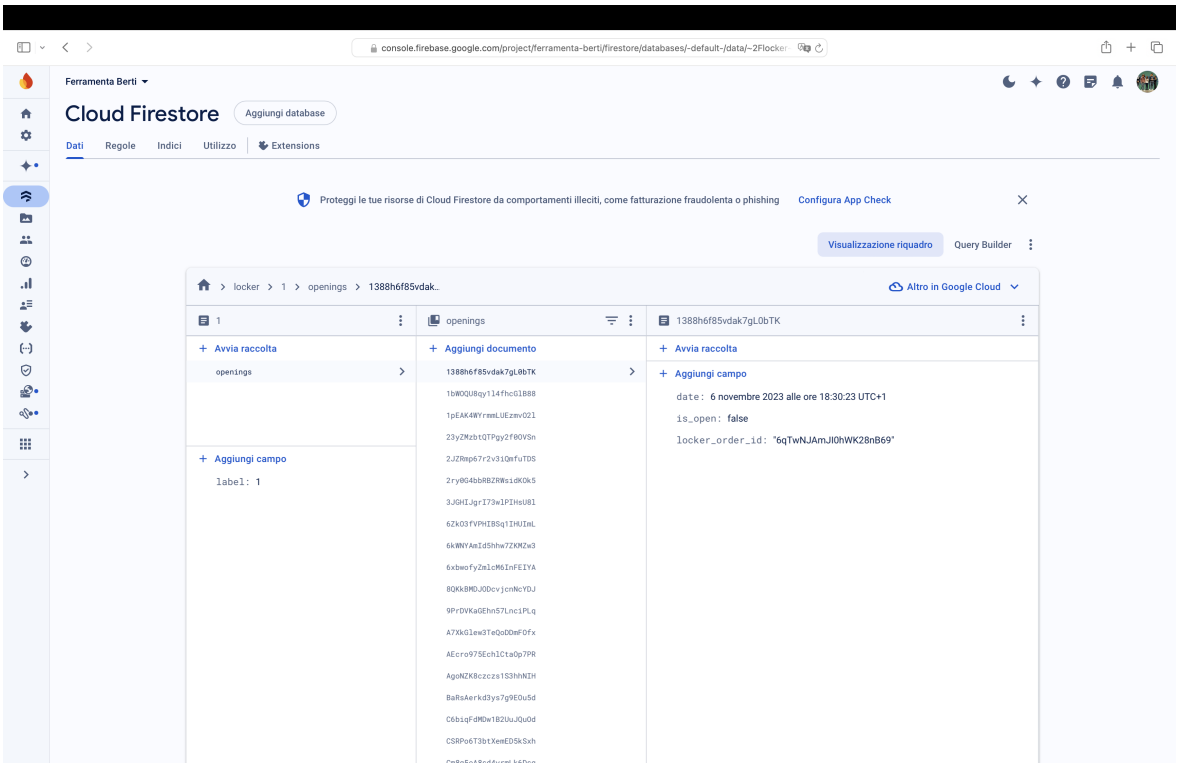


Figura 6.14: Struttura di un locker su Firebase Cloud Firestore

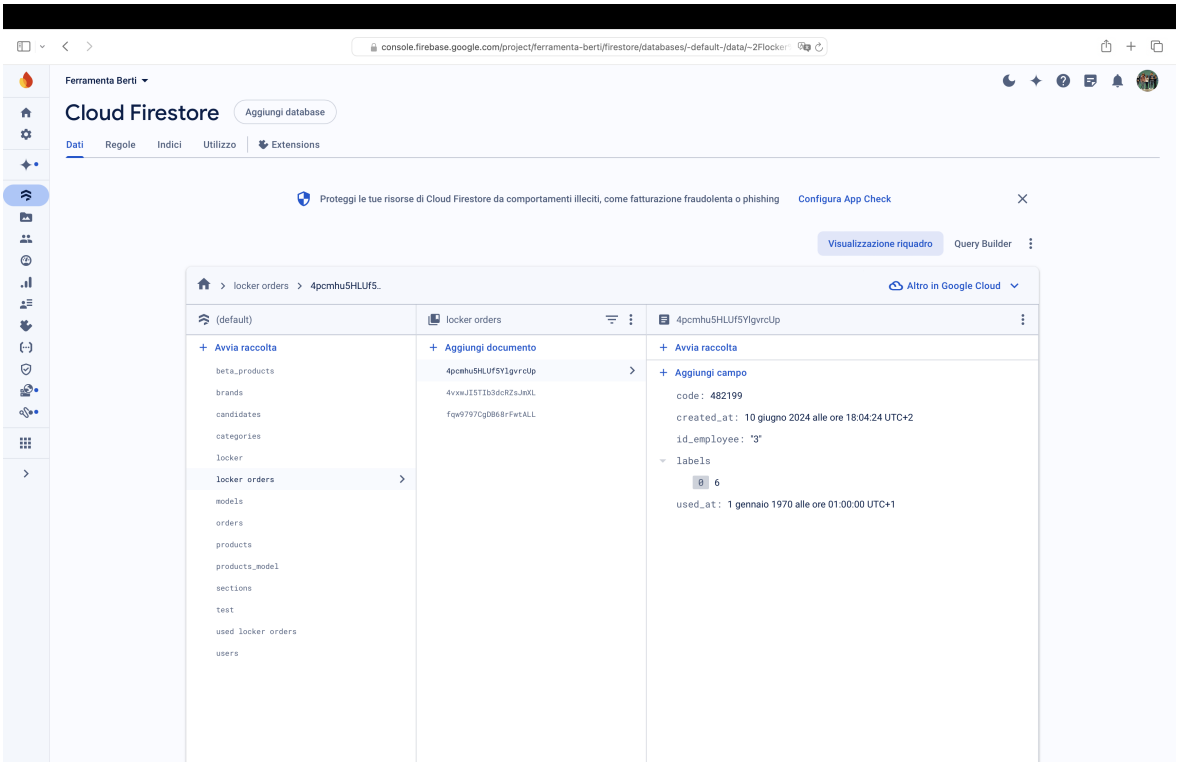


Figura 6.15: Struttura di un locker order su Firebase Cloud Firestore

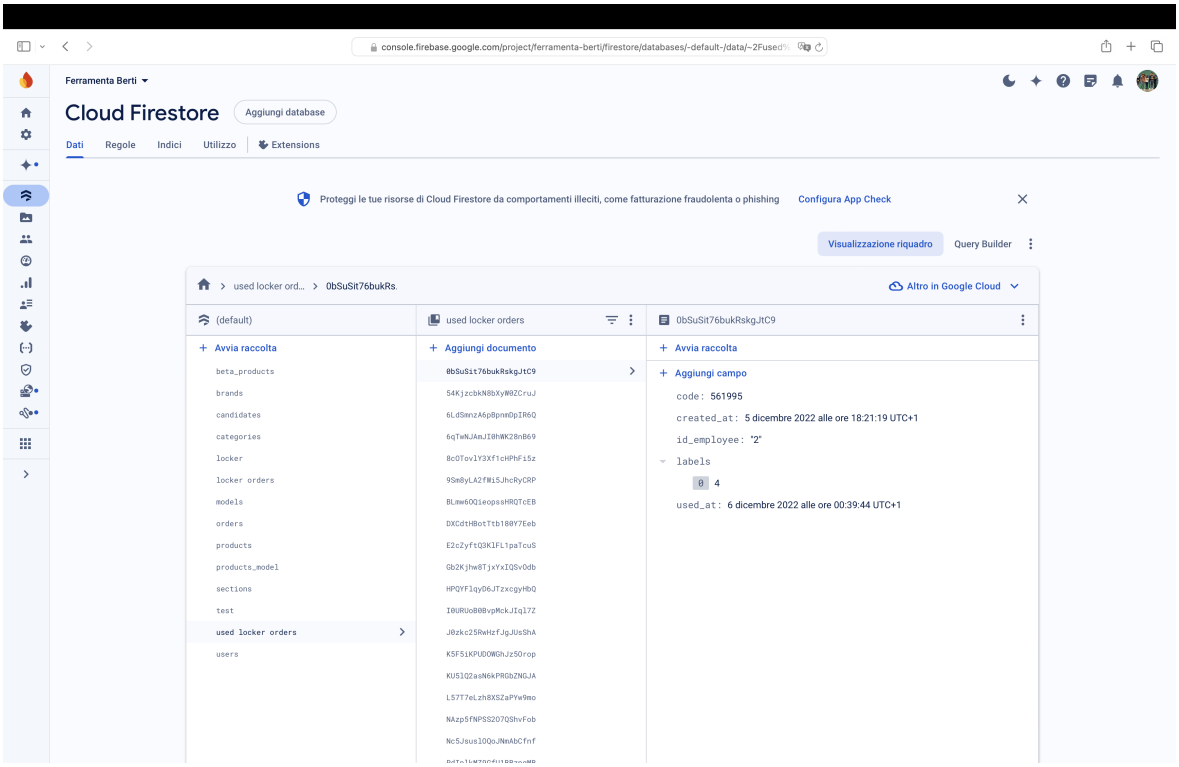


Figura 6.16: Struttura di uno used locker order su Firebase Cloud Firestore

Come possiamo vedere, i dati non sono strutturati in vere e proprie classi, dunque non è stato possibile realizzare il diagramma delle classi, ne' ci troviamo di fronte ad un database tabulare, e, quindi, non abbiamo potuto costruire il diagramma E/R (Entità / Relazione).

I locker order attualmente validi sono stati posti in una collection differente rispetto a quelli già utilizzati, per rendere più efficiente, anche nel lungo periodo, la ricerca da parte dell'app. Avremmo, certamente, potuto creare una sola collection ed aggiungere ad ogni locker order un attributo booleano che rappresentasse l'avvenuto utilizzo o meno; abbiamo usato, invece, questo approccio, oltre che per risparmiare memoria, per far si che i locker order attualmente attivi, che sono senza dubbio quelli più richiesti dai nostri client, permettano un loro accesso decisamente più veloce.

6.2.3 Firebase Functions

Questa sezione è fondamentale perché contiene tutte le funzioni, e il sistema retrostante, che i nostri front-end usano quotidianamente. Per il nostro scopo, Firebase Functions ci ha permesso di realizzare la nostra API (Application Public Interface), come possiamo vedere nella Figura 6.17

Per sviluppare il codice sorgente delle nostre funzioni appartenenti all'API, abbiamo usufruito dell'IDE di casa Microsoft chiamato Visual Studio Code, che è un editor di codice sorgente gratuito, open-source e multipiattaforma. Esso è stato progettato per essere leggero e potente, offrendo un ambiente di sviluppo integrato con numerose funzionalità per il coding, il debugging e il version control.

Il linguaggio di programmazione scelto è TypeScript (TS), anch'esso sviluppato da Microsoft, pensato per migliorare e ampliare le funzionalità di JavaScript.

TS introduce un sistema di tipi statico opzionale, che consente di verificare il tipo di variabili e le funzioni durante la fase di compilazione, piuttosto che a runtime. Questa

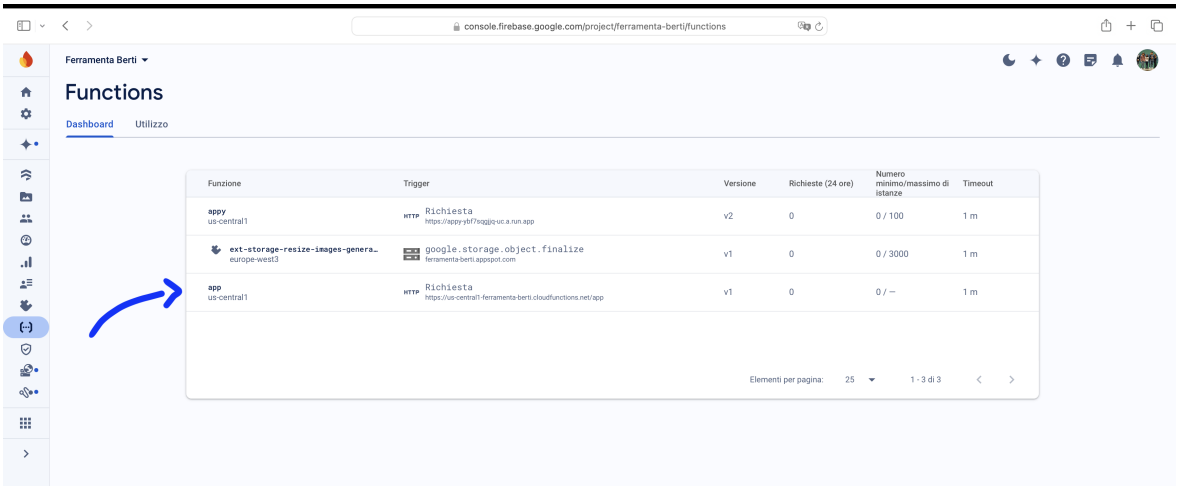


Figura 6.17: API su Firebase Functions

caratteristica ci aiuta molto a prevenire errori e a rendere il codice più robusto e, soprattutto, manutenibile.

All'interno della cartella principale, si presentano i file:

- index.ts;
- config.ts.

e le cartelle:

- classes;
- models;
- views;
- controllers;
- middlewares;
- utils;
- media;
- config.

Il file *index.ts*, oltre all'inizializzazione del server mediante il servizio chiamato *express*, definisce le funzioni pubbliche di cui i client possono usufruire, e che possiamo vedere nella Figura 6.18.

Una volta attivata una delle funzioni dell'index, la scena passa al controller incaricato, il quale, se necessario, effettua operazioni in coordinamento con altri controller, oppure con il model di riferimento.

Il tutto viene realizzato utilizzando oggetti istanziati dalle classi appropriate, che sono *Locker* e *LockerOrder*.

Tali classi implementano le policy tipiche di sicurezza, quindi possiedono attributi privati, un costruttore pubblico e i metodi *getter* e *setter*; inoltre sono stati sviluppati due metodi, che vediamo di seguito.

```
//Locker
app.get('/locker/labels', lockerController.getUsedLockerLabels)
app.get('/locker/orders/:used/:limit', lockerController.getLockerOrders)
app.get('/locker/codes', lockerController.getCodes)
app.get('/locker/order/:label', lockerController.getLockerOrder)
app.post('/locker/order', lockerController.createLockerOrder)
app.put('/locker/order', lockerController.updateLockerOrder)
app.delete('/locker/order/:id', lockerController.deleteLockerOrder)
app.get('/locker/openings/:label/:limit', lockerController.getOpenings)
app.post('/locker/opening', lockerController.createOpening)
```

Figura 6.18: Lista delle funzioni pubbliche dal file index.ts

```
    formatted(): any
    buildFromJson(data: any) : void
```

Questi due metodi permettono, rispettivamente, di convertire un'istanza dell'oggetto nella struttura dati designata al database, e viceversa.

Nel complesso, viene implementato il pattern architetturale Model-View-Controller, che è stato già descritto precedentemente nell'apposito capitolo dedicato alla progettazione.

6.3 Implementazione del front-end

Nella sezione corrente viene illustrata l'implementazione del front-end, che, nel nostro sistema, corrisponde all'applicazione mobile.

Abbiamo già affrontato l'argomento riguardante il framework da noi utilizzato, ovvero Flutter; quindi approfondiamo come sono state realizzate, a livello pratico, le tre sezioni del Model-View-Controller.

6.3.1 Model

Le due classi del componente model, ovvero *locker_order.dart* e *opening.dart*, sono progettate ed implementate nella maniera classica, ma, possiedono un metodo che, similarmente a quanto visto nelle classi del back-end, ci consente di convertire in un'istanza della classe un oggetto di tipo *json*, ricevuto come risposta da una chiamata ad un'API. Il metodo in questione per la classe Locker, che prendiamo da esempio, è il seguente:

```
LockerOrder.fromJSON(json){
    id = json["id"];
    code = json["code"];
    createdAt = DateTime.fromMillisecondsSinceEpoch(
        json["created_at"]["_seconds"] * 1000
    );
    usedAt = DateTime.fromMillisecondsSinceEpoch(
        json["used_at"]["_seconds"] * 1000
    );
    idEmployee = json["id_employee"];
    labels = json["labels"];
}
```

6.3.2 Controller

I controller, anch'essi a lungo descritti nel capitolo dedicato alla progettazione del front-end, si occupano di mettere in comunicazione la componente delle view con i dati presenti sul server; per realizzare tale task, un controller effettua operazioni con gli altri controller e mette in atto le chiamate all'API descritta in precedenza.

Ogni controller possiede i metodi CRUD per poter creare, leggere, modificare ed eliminare un'istanza della classe, sia nella memoria locale, sia sul database condiviso, ovvero su Firebase Cloud Firestore.

Vediamo, nella Figura 6.19 un esempio di un metodo di lettura, che coinvolge l'invocazione di una funzione della nostra API, la quale, se ha successo, restituisce alla vista un'istanza della classe Locker.

```
Future<LockerOrder> getLockerOrder(String id) async {
  final response = await http.get(
      Uri.parse('https://us-central1-ferramenta-berti.cloudfunctions.net/app/locker/order/$id'),
      headers: <String, String>{
        "Content-Type": "application/json"
      }
  );
  if (response.statusCode == 200) {
    final body = jsonDecode(response.body);
    final data = body["data"];
    final String id = data["id"] as String;
    final int code = data["code"] as int;
    List<dynamic> labels = data["labels"];
    DateTime createdAt = DateTime.fromMillisecondsSinceEpoch(data["created_at"]["_seconds"] * 1000);
    DateTime usedAt = DateTime.fromMillisecondsSinceEpoch(data["used_at"]["_seconds"] * 1000);
    String idEmployee = data["id_employee"];
    final LockerOrder lockerOrder = LockerOrder(id, code, createdAt, usedAt, idEmployee, labels);
    return lockerOrder;
  } else {
    return LockerOrder.invalid();
  }
}
```

Figura 6.19: Funzione di lettura del controller locker_order.dart

6.3.3 View

La view svolge, nell'applicazione front-end, un ruolo molto importante; in questa sezione vediamo com'è stata realizzata.

Si parte, ovviamente, dalla Home Page, il cui elemento padre è uno Scaffold, che rappresenta la struttura di base di una pagina, fornendo numerose funzionalità standard per la progettazione di app, come l'appBar, il body e il floatingActionButton (detto FAB).

Nella Home Page viene messo in partica uno dei principi cardine di Flutter, ovvero, l'incapsulamento. Infatti, ogni sportello viene disegnato richiamando un'altra classe denominata *door*, assegnandole, però, parametri diversi, in base all'etichietta e alle dimensioni dello stesso sportello.

Possiamo vedere il risultato di tale vista nella Figura 6.20.

L'utente dell'applicazione può, a partire dalla Home Page, visualizzare la lista degli ordini che sono stati creati e che sono attualmente validi, ma anche una lista di quelli già usufruiti. Si possono vedere queste due schermate nelle Figure 6.21 e 6.22.

Inoltre, l'utente ha a disposizione lo storico delle operazioni fisiche di apertura e chiusura di ogni sportello, che può visualizzare, semplicemente, premendo sopra la rappresentazione di uno di essi, come possiamo vedere nella Figura 6.23

Passiamo, ora, al punto di forza dell'app, che consiste nel dare la possibilità all'utente di creare un nuovo ordine, selezionando un sottoinsieme di sportelli liberi e l'operatore

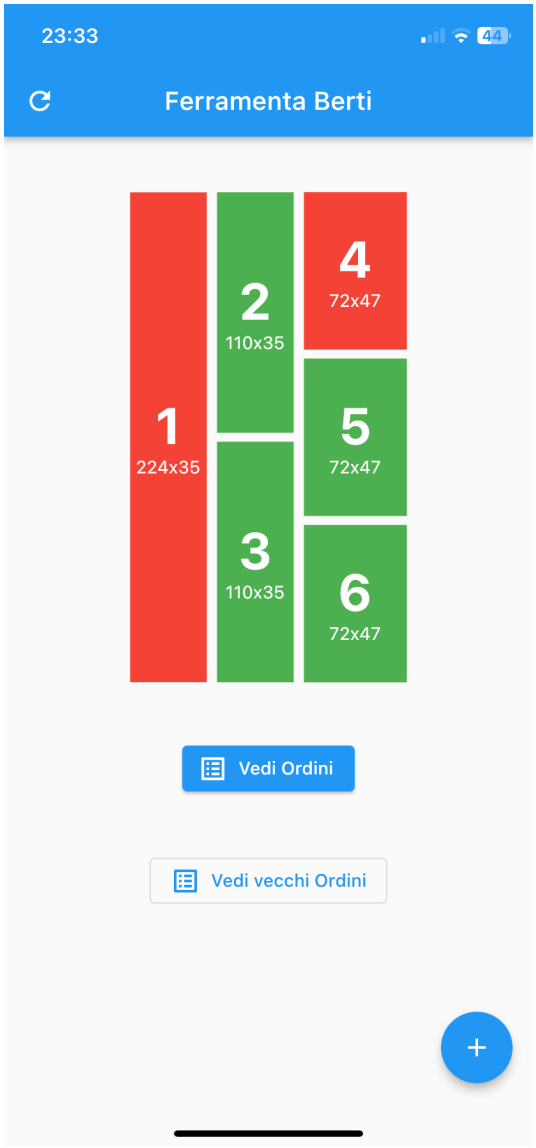


Figura 6.20: Home Page dell'app

incaricato. Tale schermata è visualizzabile premendo sul Floating Action Button della Home Page, ed il risultato è riportato nella Figura 6.24.

Dopodiché, se l'utente preme sul tasto *Salva*, l'ordine viene creato sul database, e quindi, entro il tempo massimo di cinque minuti, il locker ne verrà a conoscenza. La schermata comunica il successo dell'operazione all'utente, insieme al codice da comunicare al cliente, come vediamo nella Figura 6.25.

Tale codice può, poi, essere inoltrato a voce, oppure in automatico, mediante un'apposita funzione di invio delle istruzioni per email o per SMS. Vediamo, nella Figura 6.26, un esempio di email ricevuta dal cliente.

6.4 Test

In questa sezione si discutono i test effettuati per il back-end, svolti mediante il software chiamato Postman.

Postman è uno strumento per la costruzione, il test e anche la documentazione delle API.

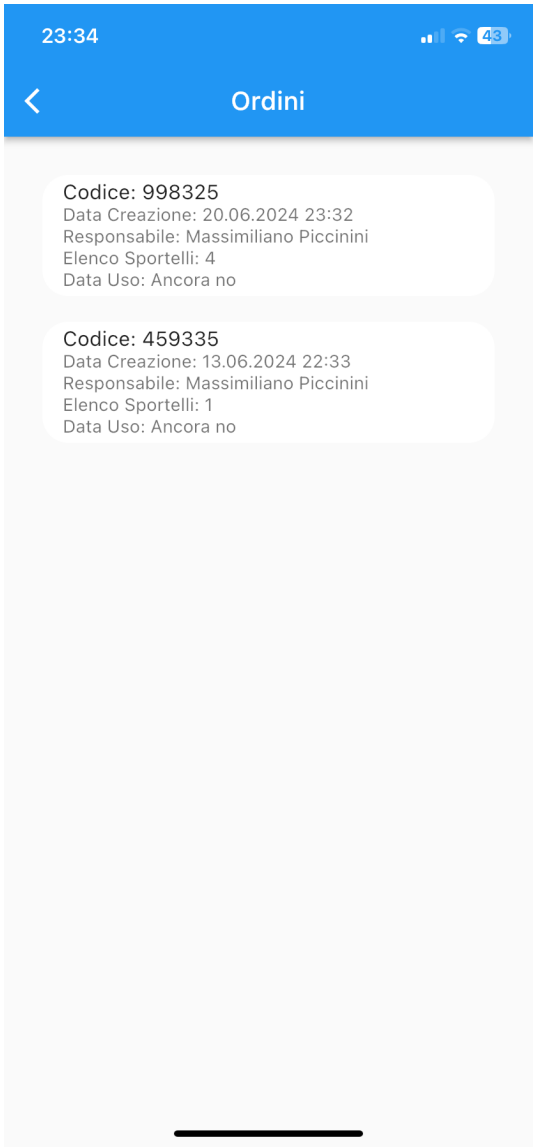


Figura 6.21: Schermata degli ordini validi

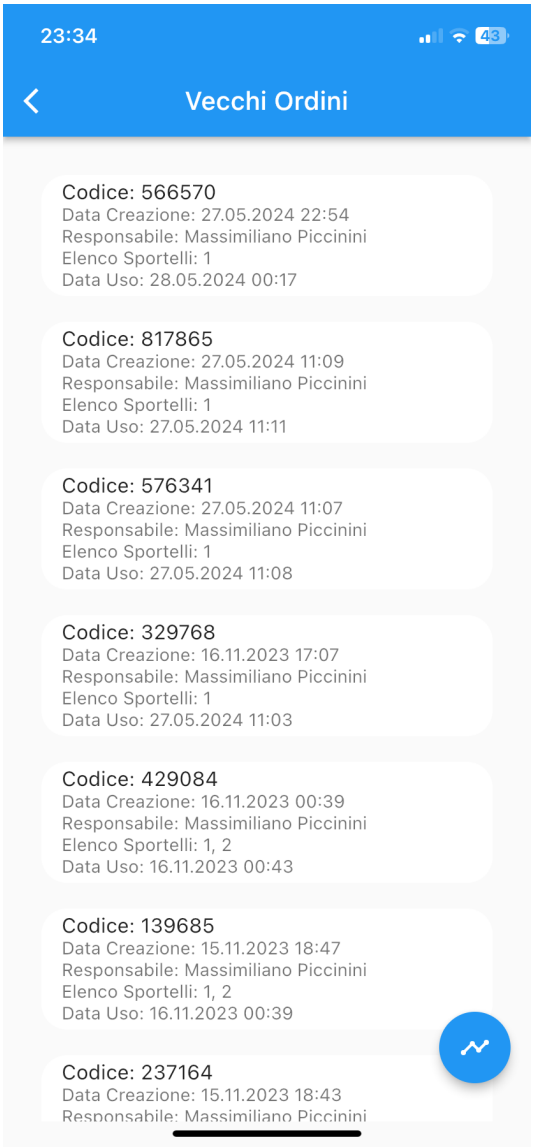


Figura 6.22: Schermata dei vecchi ordini

Dopo la prima pubblicazione dell'API, per scopi di debug, abbiamo provveduto a testare ogni funzione che potesse essere chiamata dai front-end, inviando al server parametri funzionanti, con i quali abbiamo ottenuto messaggi di successo, ma anche parametri non funzionanti, grazie ai quali abbiamo potuto verificare che si scatenassero le corrette sequenze di errori.

6.4.1 API

Attraverso l'app Postman abbiamo creato una cartella dedicata ai test del Locker e abbiamo provveduto a testare ogni singola funzione pubblica appartenente all'API creata.

Nella Figura 6.27 vengono visualizzati l'organizzazione in cartelle e l'elenco delle funzioni testate.

Nella Figura 6.28 vediamo tutti i dettagli che caratterizzano una funzione di esempio; in questo caso si tratta della comunicazione al database, da parte del locker, dell'avvenuta apertura di uno sportello. In tale figura possiamo anche visualizzare la risposta che il server restituisce al client chiamante.

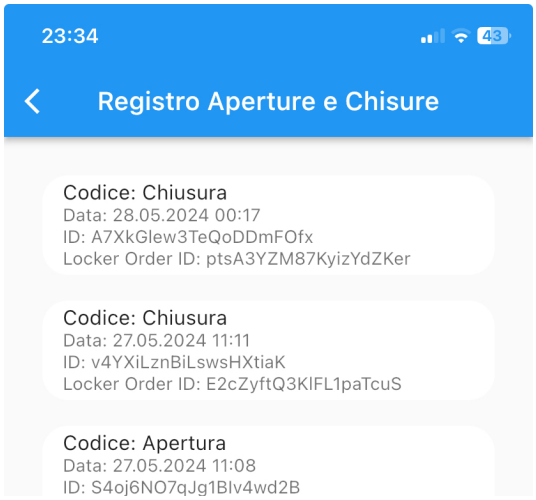


Figura 6.23: Registro delle aperture e chiusure di uno sportello

Notiamo che i fattori fondamentali da conoscere per poter richiamare una funzione sono:

- *il metodo*: può essere GET, POST, PUT o DELETE;
- *il percorso*: url della funzione;
- *i parametri*, nel caso di GET o DELETE;
- *il body*, nel caso di POST o PUT;

6.4.2 Sicurezza

Proteggere le persone, le aziende e le informazioni è parte integrante di una strategia in cui convergono sistemi di videosorveglianza, telecontrollo, antintrusione, antieffrazione, ma anche di protezione da tutte le derive del cybercrime, che colpisce gli utenti in azienda oppure in mobilità, a casa come in automobile, in treno o a piedi.

In Italia si parla genericamente di sicurezza, ma la suddivisione anglosassone specifica le differenze: Safety è la sicurezza dei lavoratori mentre Security è la sicurezza dei cittadini.

Security

Security by design è un concetto base dell'ingegneria del software; esso definisce un software progettato espressamente per essere sicuro, anticipando e minimizzando a priori gli impatti delle vulnerabilità che potrà manifestare in produzione.

Nel nostro software abbiamo adottato diverse pratiche per far sì che il sistema sia il meno vulnerabile possibile.

L'utilizzo di Firebase svolge in ciò il ruolo chiave, infatti, il nostro DBMS, che si trova a Francoforte, è di proprietà di Google, l'azienda che oggigiorno è la più sicura del mondo, e che è abituata a gestire miliardi di database di utenti provenienti da ovunque.

Se Google venisse violata, l'intera sicurezza mondiale verrebbe messa seriamente alla prova.

Google ci protegge anche da altre tipologie di attacchi, come DDoS, o altri attacchi ancora più seri, per merito dell'elevato numero di operatori di cybersecurity qualificati, e grazie alle infrastrutture tecniche migliori al mondo.

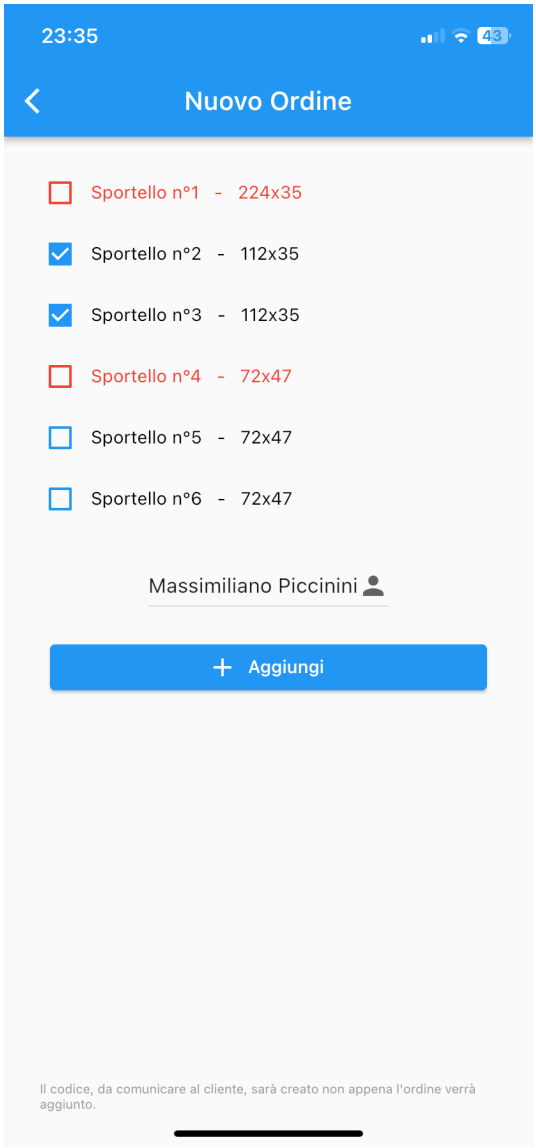


Figura 6.24: Creazione di un nuovo ordine

Safety

Nel nostro sistema, sempre grazie a Firebase, a Google Analytics e a Google Cloud Platform, possiamo monitorare ogni aspetto; per esempio siamo in grado di vedere su quale schermata ogni utente passa del tempo, ma riusciamo anche a monitorare l'andamento dell'utilizzo delle varie funzionalità, per individuare degli errori che potrebbero rendere l'esperienza dell'utente meno gradevole.

A livello di grafica dell'app front-end, per l'utente abbiamo previsto un alert che appare in ogni caso d'errore, mostrando l'id dell'errore e il messaggio, in modo tale da fornire all'utente stesso informazioni a riguardo.

Qualora l'alert fosse dovuto a qualche errore degli sviluppatori, gli utenti possono sempre contattare questi ultimi fornendo informazioni dettagliate sull'errore.

Come ultima cosa, una buona pratica che abbiamo adottato proviene dalla seguente dicitura:

> "nel web, se è distruttivo, allora richiede una conferma"

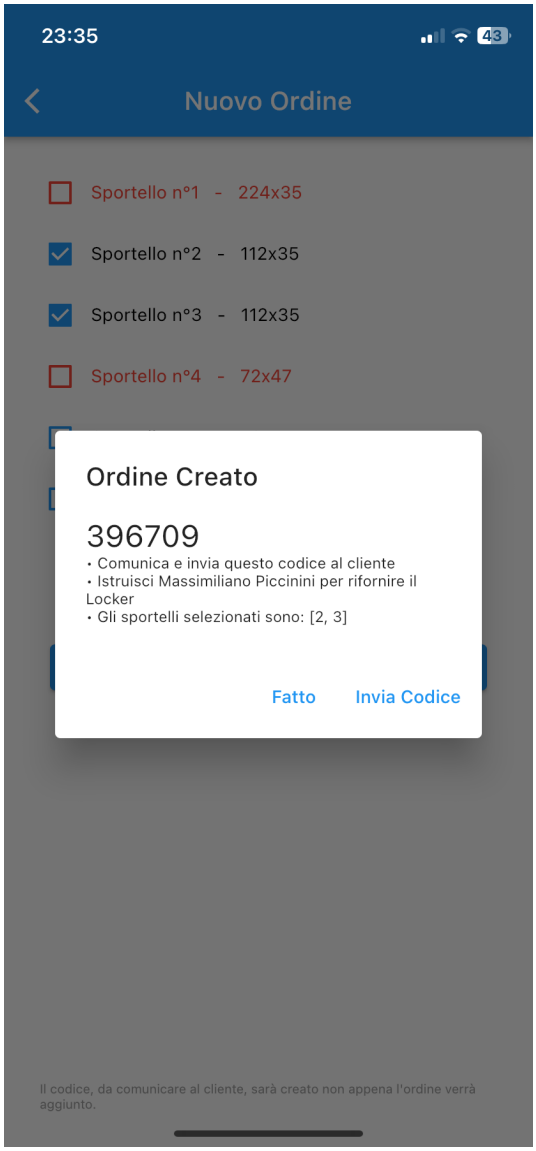


Figura 6.25: Schermata di feedback dell'ordine creato con successo

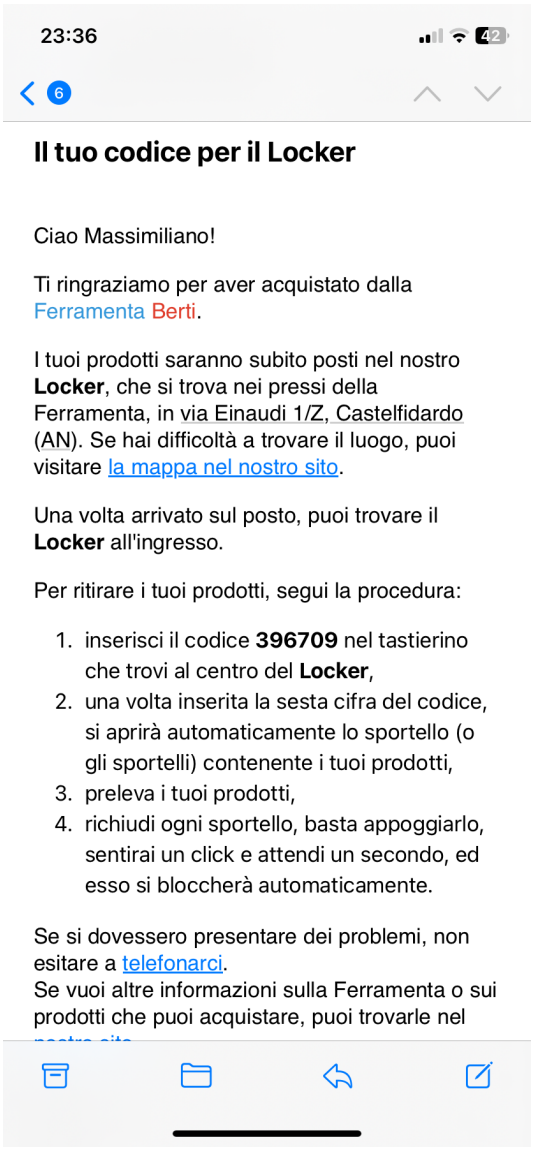


Figura 6.26: Email con istruzioni ricevuta dal cliente

Tale regola sta ad indicare che, per ogni operazione che cancella dei dati in maniera permanente, l'utente deve confermare l'operazione due volte; abbiamo realizzato questo meccanismo tramite un alert.

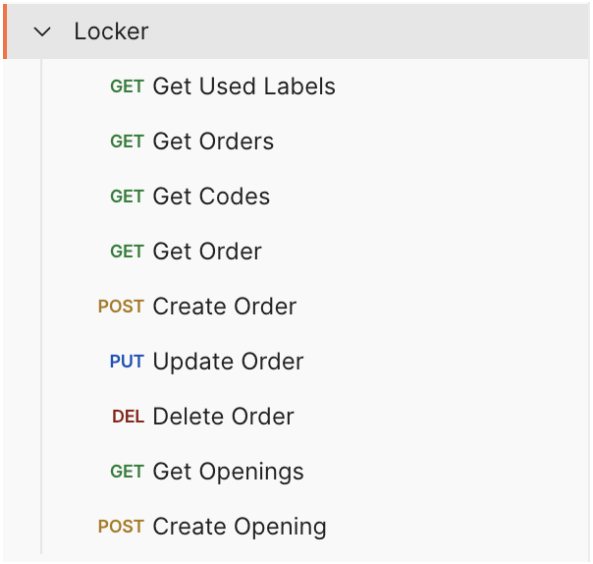


Figura 6.27: Collection 'Locker' di Postman

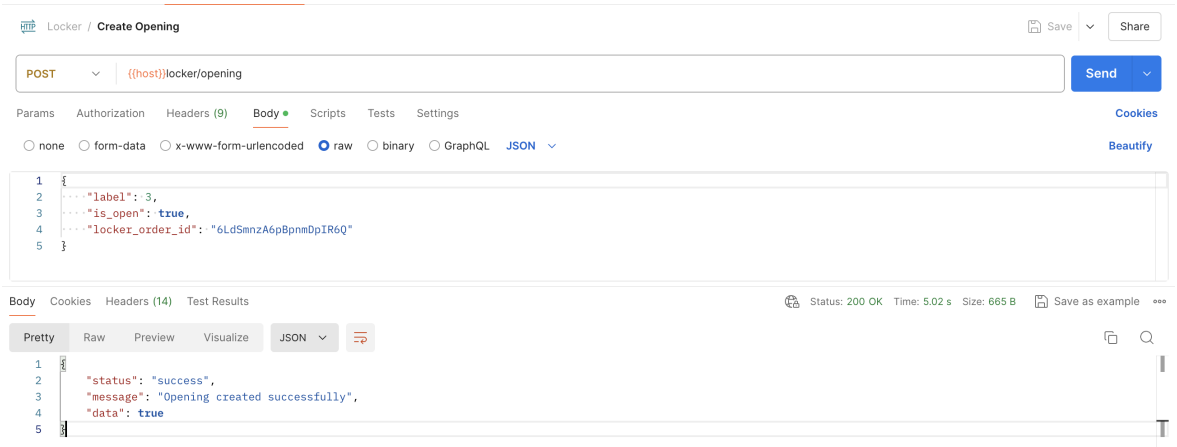


Figura 6.28: Invocazione della funzione 'Create Opening' da Postman

CAPITOLO 7

Discussione

In questo capitolo vedremo un aspetto molto più concettuale rispetto ai capitoli precedenti.

Abbiamo, infatti, terminato l'analisi tecnica del software realizzato, ed ora procederemo discutendo in merito a due importanti aspetti, ovvero la SWOT analysis e le lezioni apprese.

Il primo serve per valutare, avvalendosi di metodologie ben note, alcuni aspetti del nostro progetto, mentre il secondo, che è personale, è necessario per fare un'autocritica, fondamentale per i futuri progetti, che sicuramente prenderanno spunto da questi, ma possono essere gestiti in maniera ancora più efficiente.

7.1 Analisi SWOT

L'analisi SWOT è una tecnica utilizzata per identificare i punti di forza, di debolezza, le opportunità e le minacce di un'azienda o anche di un progetto specifico.

Essa è ampiamente usata da molte organizzazioni, dalle piccole imprese agli enti non-profit fino alle grandi imprese, e può essere utilizzata sia per scopi personali che professionali. Per quanto semplice, l'analisi SWOT è un potente strumento che aiuta a identificare opportunità competitive di miglioramento.

In poche parole, SWOT sta per Strengths (punti di forza), Weaknesses (punti di debolezza), Opportunities (opportunità) e Threats (minacce).

Ciascuno di questi fattori va esaminato attentamente per pianificare adeguatamente la crescita dell'organizzazione, perché è qui che entra in gioco l'analisi.

Se analizzato insieme, il framework SWOT presenta un quadro più ampio della condizione attuale e di come arrivare alla fase successiva.

Entreremo nel dettaglio riguardo ciascuno di questi termini e di come possono contribuire a identificare le aree di miglioramento.

7.1.1 Punti di forza

I punti di forza nell'analisi SWOT sono quelle iniziative interne che danno buoni risultati. Nel nostro caso, queste ultime sono molteplici.

In questo ultimo periodo in cui siamo entrati in maniera più decisa nel mondo dell'industria dell'informatica e dell'automazione, non potevamo non notare che si tratta di un settore in forte crescita.

Sebbene l'informatica risulti essere ancora ostile alla maggior parte delle persone, l'automazione sembra essere vista di buon occhio un po' da tutti, e la spiegazione, secondo noi, è che quest'ultima sa essere *autonoma* ed, inoltre, molto più tangibile, e che, quindi, alle aziende, può comportare un guadagno più diretto di quello che, invece, potrebbe fare un software informatico.

Come conseguenza, un prodotto che funziona in autonomo come il nostro ha maggiori chance di essere preso in considerazione per l'acquisto da parte delle piccole e medie imprese, alle quali si consente di risparmiare tempo e di fornire un servizio in più al cliente.

Dal punto di vista tecnico, invece, possiamo garantire che le prestazioni, la disponibilità e la sicurezza, sono i punti di forza del nostro sistema, in quanto, per la realizzazione di tali servizi, ci siamo affidati a Google, che è la leader indiscussa del settore.

Un altro punto di forza, sotto diversi aspetti, è l'artigianalità del software.

Con "artigianalità" intendiamo il fatto che, a livello del software, esso è stato progettato e realizzato al 100% da noi; quindi ne conosciamo bene ogni aspetto, sia per quanto riguarda la grafica sia per quanto riguarda la logica.

Possiamo garantire l'efficienza più totale di queste componenti, non dovendo caricare librerie esterne, che molto spesso sono decisamente pesanti, e realizzate per una moltitudine di scopi.

Conoscere bene il proprio software, nel nostro ambito, è fondamentale, e può far emergere il prodotto rispetto alla concorrenza, in quanto abbiamo imparato a personalizzare ogni minimo aspetto.

7.1.2 Punti di debolezza

I punti deboli nell'analisi SWOT si riferiscono a iniziative interne che non rendono quanto dovrebbero.

Quest'ultimo aspetto, visto nei punti di forza, rappresenta, per ora, ancora una debolezza sotto certi punti di vista.

Infatti, per realizzare ogni componente del software, abbiamo impiegato molto tempo, che non sempre il cliente è disposto ad attendere; mentre, ad oggi, ci sarebbero numerose soluzioni *chiavi in mano* che avremmo potuto implementare, al posto di progettare ed implementare le nostre soluzioni.

Un discusso punto di debolezza può essere l'utilizzo di Arduino, il quale è, specificatamente, destinato a progetti *casalinghi* e, quindi, consigliato come strumento da hobby più che industriale.

L'alternativa che può essere presa in considerazione è, senza dubbio, il PLC.

Avendo avuto modo di colloquiare con produttori di PLC, abbiamo modo di credere che nelle situazioni aziendali assimilabili a domestiche, queste soluzioni sono sufficienti.

Chiaramente, i parametri vitali del sistema, per come si presenta oggi, vanno costantemente monitorati, per poter, in seguito, trarre conclusioni più decise.

7.1.3 Opportunità

Le opportunità nell'analisi SWOT sono il risultato degli attuali punti di forza e debolezza, insieme a qualsiasi iniziativa esterna che ci metterà in una posizione competitiva più forte.

Una possibile opportunità sarebbe, per noi, pubblicare la libreria che consente di interagire con un motore servo utilizzando il protocollo I2C, che, per come si sta muovendo il mercato, potrà essere utile a migliaia di sviluppatori futuri.

Al giorno d'oggi esiste, infatti, una libreria generica per gestire tali motori servo, così come ne esistono altre per poter utilizzare in maniera agile dispositivi come il tastierino o il display LCD.

Per questi ultimi, però, esistono anche librerie per utilizzarli mediante il protocollo di comunicazione I2C, mentre, per i motori servo, non si trova niente sul mercato. Facendo diverse ricerche su Internet, sembra sia impossibile muovere un motore servo via software e I2C, e c'è chi, addirittura, vende un dispositivo hardware apposito.

Di conseguenza, la pubblicazione della libreria da noi realizzata ed utilizzata con successo potrebbe garantire ad altri sviluppatori di risparmiare tempo e denaro.

L'artigianalità del software ci consente, soprattutto, di poterlo suddividere in moduli e rivenderlo anche ad altre aziende, non necessariamente solo di ferramenta, personalizzando gli aspetti richiesti dal committente.

Grazie, quindi, a questo "farsi tutto in casa" sarà possibile integrare il nostro sistema con altri moduli, come l'e-commerce, senza dover sviluppare nuovamente le API interne.

Ciò porterebbe un enorme vantaggio ai clienti della ferramenta, che potranno ordinare i loro strumenti online e farseli inserire direttamente nel locker più vicino a casa, evitando, quindi, che il corriere passi senza trovare nessuno, ed, anzi, prelevando i prodotti quando lo trovano più comodo.

7.1.4 Minacce

Le minacce nell'analisi SWOT si riferiscono ad aree che potenzialmente potrebbero creare problemi.

Nel nostro ambito la concorrenza esiste già da anni; quindi è inevitabile perdere potenziali clienti a causa delle tempistiche di realizzazione del software.

Per quanto riguarda le minacce di sicurezza, il sistema non è stato supervisionato da nessun esperto, né ci riteniamo tali, e, anche se crediamo di averle pensate tutte, qualche utente più esperto di noi potrebbe sempre metterci in difficoltà.

Inoltre, siamo consapevoli che la sicurezza informatica non è sufficiente, ed in questi casi è più importante la sicurezza fisica; quindi, sebbene il locker è monitorato giorno e notte da telecamere, si potrebbe intervenire solo a posteriori dopo un eventuale danneggiamento del dispositivo.

7.2 Lezioni apprese

Durante la progettazione e lo sviluppo del software, numerose sono state le lezioni che ci sentiamo di aver appreso.

Molte di esse derivano dal fatto che siamo entrati nel mondo del lavoro dopo essere stati per molti anni all'interno dell'ambiente protetto scolastico, in cui abbiamo notato che sono necessarie molte competenze trasversali, che si possono ottenere solo con l'esperienza maturata nel settore in cui ci si trova.

7.2.1 Non reinventare la ruota

Data la nostra passione verso l'arte delle progettazione e della programmazione, ci divertiamo molto ad *inventare* nuove soluzioni apparentemente geniali per i problemi che si pongono nel corso della nostra strada; ma per quanto possiamo essere abili, ci sono molti casi in cui, grazie al principio dell'*open-source* di cui l'informatica si fa portavoce, la soluzione era già disponibile ed aperta a tutti.

In questi casi, finché si tratta di hobbistica non si ha alcuna conseguenza, ma quando si passa al mondo della produzione, ogni risparmio di tempo diventa fondamentale, ed avere una soluzione già testata e comprovata da altri può sicuramente avvantaggiarci.

7.2.2 L'abito fa il monaco

Sebbene il nostro lavoro si è focalizzato sull'ottimizzazione pratica, ad oggi, molte persone scelgono cosa acquistare in base all'estetica. La scelta di un prodotto da parte di un acquirente viene indottrinata dalle emozioni, e solo successivamente giustificata con la ragione.

Per questo motivo bisogna spostare una buona parte della nostra concentrazione sull'estetica del locker, di cui questa tesi, però, non tratta.

7.2.3 Gestione degli errori

La gestione degli errori non è assolutamente da sottovalutare, ma anzi da progettare con spirito critico e cercando di prevedere ogni possibile eccezione.

Nel nostro caso ciò si riflette in particolare sul firmware prodotto, che deve implementare a dovere la caratteristica della disponibilità.

7.2.4 Monitora e analizza i dati con spirito critico

Grazie alla Data Science, oggi, ma ancora di più domani, è possibile analizzare moli di dati eterogenee e ricavarne informazione utile per il futuro.

Per questo motivo abbiamo deciso di raccogliere quanti più dati e misure possibile, memorizzandole su un database; la loro analisi sarà sicuramente sfruttata per correzioni o miglioramenti futuri.

7.2.5 Impara l'arte e mettila da parte

Durante la nostra esperienza accademica abbiamo studiato molteplici aspetti del settore informatico e dell'automazione, anche se taluni, all'apparenza, sembravano completamente scorrelati tra loro.

Facendoci esperienza nel mondo del lavoro, abbiamo imparato che, invece, le conoscenze apprese in tutti questi ambiti si relazionano tra loro e, anzi, permettono un notevole vantaggio competitivo rispetto a chi ha, per esempio, fatto solo qualche corso di programmazione.

Per la nostra esperienza, oggi, gli informatici, in generale, sono visti come coloro che svolgono un mestiere difficile ma freddo ed apatico; il mercato di questo settore offre moltissimi programmatori, capaci di trasformare in codice sorgente le volontà di un manager.

Passione per la scoperta, curiosità del perché e visione dell'insieme: se questi tre aspetti riescono a convivere, allora ci si può distinguere, emergendo; allora, ragionando con la propria testa si può migliorare un prodotto, velocizzare un servizio e progredire verso un futuro in cui l'informatica e l'automazione saranno al servizio di chiunque ne abbracci i valori.

CAPITOLO 8

Conclusioni

In questo capitolo traiamo le conclusioni riguardanti il nostro progetto, dando poi uno sguardo ai possibili sviluppi futuri.

Possiamo sicuramente affermare che l'esperienza vissuta è stata molto positiva, dato che, essendo appassionati della materia, abbiamo fatto del nostro meglio, soprattutto dal punto di vista della ricerca e sviluppo, in quanto non ci siamo accontentati di realizzare un prodotto funzionante, ma abbiamo cercato di renderlo quanto più efficiente possibile, anche se, sicuramente, ci sono ancora ampi margini di miglioramento.

Quanto appena detto si traduce nel fatto che più volte, durante la progettazione e, successivamente, anche durante lo sviluppo, abbiamo deciso di cancellare parte dell'operato per adottare una soluzione più efficace o più efficiente.

Ciò, chiaramente, ci ha causato un'elongazione del lasso temporale di produzione, ma possiamo affermare con certezza che ne è valsa la pena, e per di più, se d'ora in poi ci capitassero progetti o situazioni similari, sapremmo come muoverci molto più agilmente, verso una direzione certa e con meno dubbi.

Una particolare esperienza positiva è stata la fase in cui non riuscivamo a trovare una soluzione per comandare, attraverso la PWM, il servo motore mediante I2C, che abbiamo risolto creando una nostra versione della libreria Servo, realizzata direttamente dalla casa madre Arduino, che premiamo di pubblicare per aiutare chiunque, come è successo a noi, si troverà in una situazione simile.

In sostanza, il sistema è stato realizzato a scopo dimostrativo, e con alcune ulteriori implementazioni potrebbe migliorare ancora.

Dato che il mercato del commercio online e della logistica sta continuando a crescere esponenzialmente, potremmo immaginare delle implementazioni future, le quali, se integrate con il sistema attualmente progettato e sviluppato, potrebbero incrementare le possibilità che esso sia introdotto nel mercato, nei modi che vedremo in seguito.

Si potranno, sicuramente, migliorare gli aspetti della sicurezza informatica, ottenendo dei certificati che ne assicurano la qualità e la robustezza alle intrusioni da parte di malintenzionati.

Inoltre, potremmo procedere creando una soluzione *chiavi in mano*, flessibile ed adattabile a quante più esigenze possibili che ogni cliente potrebbe avere, garantendo, quindi, il processo completo, dall'implementazione di automazioni personalizzate all'installazione fisica in loco, ma anche di una caratterizzazione della logica o della grafica dell'applicazione in base alle richieste specifiche del committente.

Un argomento di cui, purtroppo, dobbiamo ammettere di conoscere ben poco, sono le leggi che governano il mondo del web e delle vendite.

A causa di questo saremo costretti ad affidarci ad un servizio di terze parti a pagamento, come Iubenda, che è in grado di generare automaticamente i Termini e le Condizioni, e le Politiche di Privacy, basandosi sulle informazioni fornite. In un futuro ancora più posteriore, prevediamo di rivolgerci ad un commercialista.

Come ultimo aspetto, sarebbe interessante realizzare un'analisi di mercato per capire come poter realizzare una rete di *smart locker* distribuita sul territorio, di cui ogni azienda che possiede un proprio sito *e-commerce* ne possa usufruire, garantendo, così, un'esperienza al cliente finale decisamente migliore.

Bibliografia

ALESSANDRIA, S. (2021), *Flutter Cookbook: Over 100 proven techniques and solutions for app development with Flutter 2.2 and Dart*.

BLOOM, J. (2019), *Exploring Arduino: Tools and Techniques for Engineering Wizardry*.

BRYANT, D. (2022), *Mastering API Architecture*.

FARLEY, D. (2020), *Modern Software Engineering: Doing What Works to Build Better Software Faster*.

GRUHN, V. (2018), *The Essence of Software Engineering*.

HAJIAN, M. (2024), *Flutter Engineering*.

JIN, B. (2018), *Designing Web APIs: Building APIs That Developers Love*.

LOUBSER, N. (2020), *Software Engineering for Absolute Beginners: Your Guide to Creating Software Products*.

MARGOLIS, M. (2020), *Arduino Cookbook: Recipes to Begin, Expand, and Enhance Your Projects*.

NAKAMOTO, S. (2023), *ARDUINO SIMULATION AND BLOCK CODING: A Comprehensive Guide to Mobile Communication Projects A Step-by-Step Guide for Beginners*.

NATHEEM, A. (2021), *Handbook of Arduino: 100+ Arduino Projects learn by doing practical guides for beginners and inventors.*

NELSON, C. (2024), *Software Engineering for Data Scientists: From Notebooks to Scalable Systems*.

RICHARDS, M. (2020), *Fundamentals of Software Architecture: An Engineering Approach*.

ROSE, R. (2023), *Flutter and Dart Cookbook: Developing Full-Stack Applications for the Cloud*.

THOMAS, H. (2020), *Distributed Systems with Node.js: Building Enterprise-Ready Backend Services*.

Sitografia

- Stack Overflow – `https://stackoverflow.com`
- W3Schools – `https://www.w3schools.com`
- Flutter – `https://flutter.dev`
- Firebase – `https://firebase.google.com`
- Arduino – `https://www.arduino.cc`
- AZ-Delivery – `https://www.az-delivery.de/it`

Ringraziamenti

Ringrazio il Prof. Ursino, per avermi guidato e supportato nella fase più importante del mio percorso accademico.

Ringrazio la Ferramenta Berti per avermi dato la possibilità di svolgere il mio lavoro di tesi in un luogo interessante e dinamico, che mi ha permesso di mettermi in gioco e fare un'esperienza che sarà preziosa per il mio futuro.

Alla mia famiglia, i miei amici, i miei compagni di università e a tutti quelli che hanno incrociato la loro vita con la mia lasciandomi qualcosa di buono. Grazie per essere stati miei complici, ognuno a suo modo, in questo percorso intenso ed entusiasmante, nel bene e nel male. Sono così tanti i ricordi che mi passano per la testa che è impossibile trovare le parole giuste per onorarli. A farlo saranno le mie emozioni, i miei sorrisi e le mie lacrime che insieme si mescolano in un bagaglio di affetto sincero e gratitudine per tutti voi.

Grazie per aver reso il mio traguardo davvero speciale!