UNIVERSITÀ DEGLI STUDI DI FERRARA
DIPARTIMENTO DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA
E DELL'AUTOMAZIONE

Studio e progettazione di una piattaforma IoT per la gestione automatizzata di parcheggi

Fogli Federico

Fontana Luca

Anno Accademico 20222/2023

Introduzione

La presente relazione descrive una piattaforma per la gestione di un multi-parcheggio automatizzato e intelligente (*SmartPark*), il cui obiettivo è di coordinare l'accesso e rilevare i posti occupati al suo interno.
Il sistema, basato su di un'infrastruttura IoT, prevede l'accesso degli utenti e degli amministratori, che potranno accedere alle funzionalità fornite tramite un'interfaccia Web.

Requisiti funzionali

La piattaforma deve soddisfare i seguenti requisiti funzionali:

1. **Gestione degli accessi.** Gli utilizzatori possono accedere al parcheggio dopo essersi autenticati inserendo targa e password.
2. **Gestione dei permessi**. L'accesso potrà essere effettuato con i privilegi da amministratore o come utente.
3. **Visualizzazione della mappa.** Deve essere possibile localizzare i vari SmartPark in una mappa interattiva.
4. **Rilevamento in tempo reale dei posti occupati**. Sarà necessario rilevare automaticamente e in tempo reale i posti occupati all'interno dello SmartPark.
5. **Visualizzazione delle soste.** Gli utenti devono poter visualizzare le soste effettuate (completate o ancora attive) e le relative informazioni: prezzo, orario di ingresso e di uscita, e la targa del veicolo.
6. **Filtraggio delle informazioni.** Deve essere consentito, sia agli amministratori che agli utenti, di filtrare le informazioni visualizzate in base a diverse categorie, come ad esempio la data e la targa del veicolo.
7. **Gestione degli SmartPark.** Gli amministratori potranno visualizzare le soste completate e le statistiche giornaliere del parcheggio, inoltre sarà possibile gestire le tariffe e inviare/recuperare la telemetria dei dispositivi.
8. **Gestione dei prezzi.** Gli amministratori potranno definire i prezzi dei parcheggi, aggiungendone, modificandone o eliminandone uno esistente.

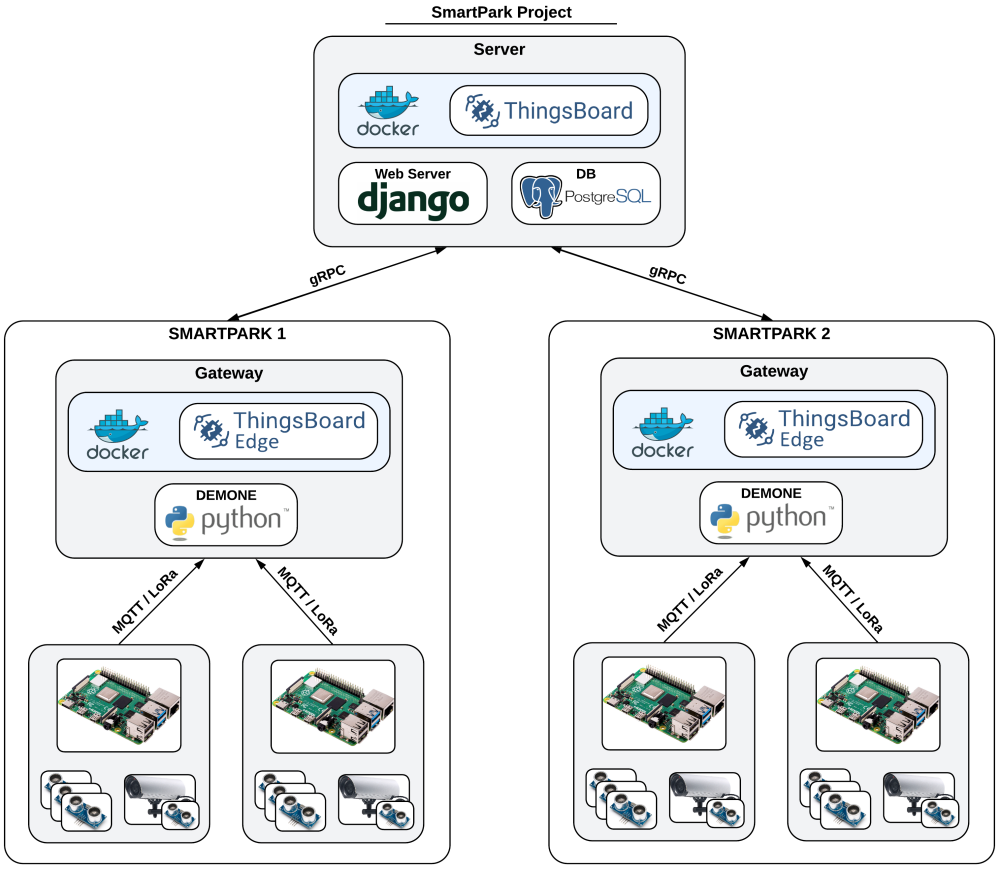
Requisiti non funzionali

L'infrastruttura deve soddisfare i seguenti requisiti non funzionali:

1. **Sicurezza.** Bisogna garantire un buon livello di sicurezza, proteggendo i dati degli utenti, impedendo l'accesso non autorizzato al parcheggio oppure un'uscita senza pagamento.
2. **Usabilità.** Gli utilizzatori, anche quelli meno esperti, devono poter usufruire delle funzionalità in modo semplice e intuitivo.
3. **Scalabilità.** L'applicativo deve essere in grado di supportare un grande numero di utenti e di parcheggi senza rallentamenti.
4. **Bassa Latenza.** Data la possibilità di avere una moltitudine di parcheggi e utenti sarà necessario implementare soluzioni di *Edge Computing* per ridurre la latenza di comunicazione e il carico sul server.

Architettura

La piattaforma é caratterizzata da un'infrastruttura IoT basata su una serie di sensori di prossimità e telecamere che collaborano per gestire il multi parcheggio intelligente con gestione degli accessi e rilevamento dei posti occupati.
All'esterno di ogni SmartPark, sia all'ingresso che all'uscita, saranno presenti un sensore di prossimità per rilevare la presenza di veicoli e una telecamera per leggere la targa. Invece, all'interno sltri sensori di prossimità permetteranno di rilevare se lo stato dei parcheggi (occupato/libero).
I dati provenienti dai sensori di ogni parcheggio vengono inviati al gateway (uno per SmartPark) in cui é presente un'istanza di Thingsboard Edge, quest'ultima processa localmente i dati e invia al server centrale le informazioni riassuntive giornaliere.
La comunicazione tra dispositivi e gateway avviene tramite di alcune RaspberryPI che effettuano delle chiamate ad API REST.
Il server ospita un'istanza dokerizzata di Thingsboard Server che, oltre ad interfacciarsi con i gateway, comunica con un'applicativo web per permettere agli utenti di poter ususfruire del servizio di parcheggio.
I dati non sono memorizzati in maniera centralizzata, infatti i dati relativi ai sensori vengono archiviati all'interno del database di Thingsboard Server, invece i dati utili alla WebApp sono presenti all'interno di un database PostgreSQL (database differenti ma stesso DBMS).



Le funzionalità edge migliorano la scalabilità e la latenza, in quanto una importante parte della computazione viene svolta dai gateway, un aggiunta di un nuovo SmartPark infatti non aggiunge un carico eccessivo al server.

Funzionamento

Il funzionamento del software prevede l'utilizzo di una telecamera e un sensore di prossimità per ogni accesso, in grado di rilevare le auto e le relative targhe, l'accesso viene registrato e utilizzato per il pagamento, che prevede diverse fasce orarie. L'utente può visualizzare la sosta e il relativo importo in qualsiasi momento e pagare. Entro 15 minuti dal pagamento, l'utente deve presentarsi all'uscita, dove la telecamera leggerà la targa e salverà su database la fermata compresa di ora di uscita, permettendo all'utente di uscire. Il sistema permette di uscire solo previo pagamento, e in caso siano passati 15 minuti dal pagamento, sarà necessario contattare l'assistenza.

Database

Il database rappresenta uno dei componenti fondamentali dell'applicazione e nella nostra soluzione abbiamo utilizzato due DB diversi interscambiabili: PostgreSQL e Oracle. Entrambi sono dei database relazionali molto popolari che offrono elevate prestazioni, funzionalità avanzate e sono altamente affidabili.

In particolare, all'interno del nostro database abbiamo definito 5 tabelle principali:

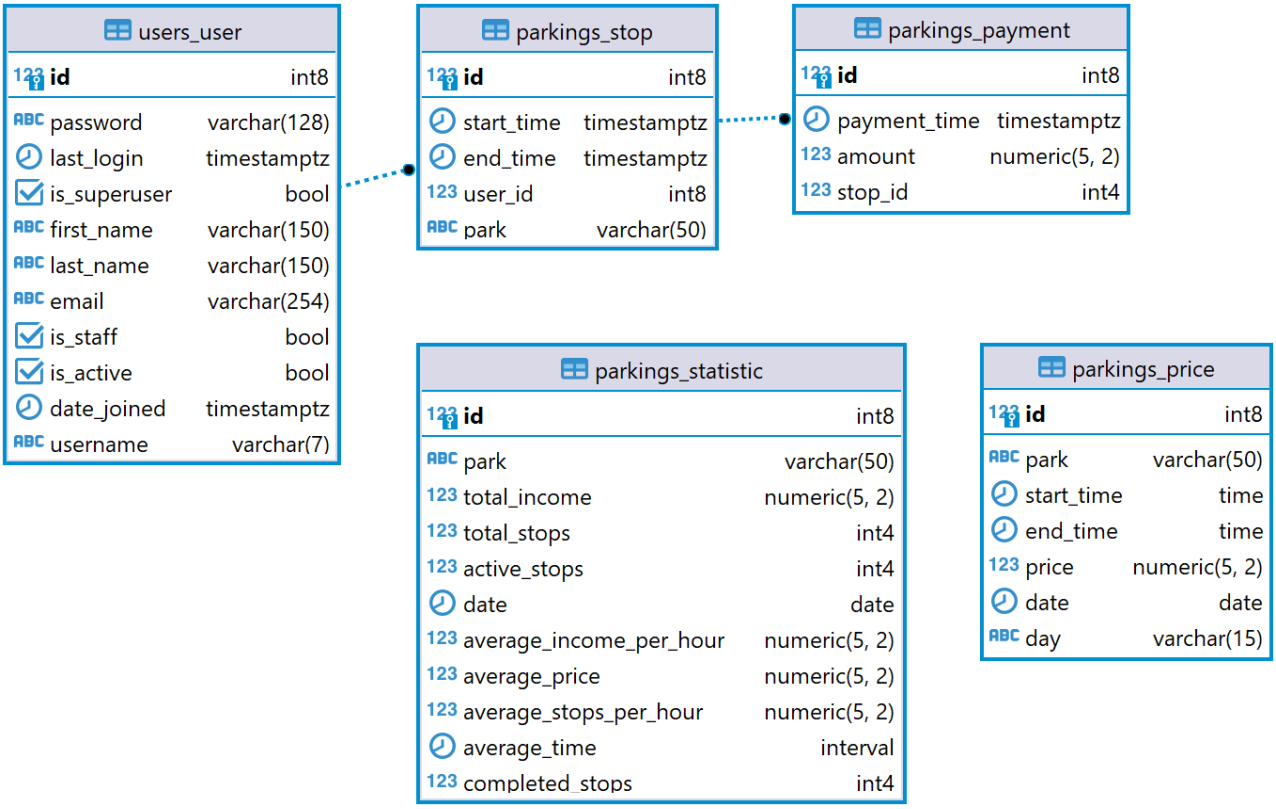
1. **users_user:** contiene gli utenti registrati all'applicazione. Questa tabella memorizza informazioni come l'username dell'utente, che corrisponde alla targa ed è rappresentato da un varchar di 7 caratteri, l'email, la password memorizzata con hash e salt, e la data di iscrizione.

2. **parkings_stop::** contiene le informazioni relative alle soste degli utenti. Questa tabella memorizza le fermate degli utenti, comprensive di data di inizio e fine, parcheggio di riferimento e una chiave esterna che punta all'utente.

3. **parkings_payment:** contiene le informazioni relative ai pagamenti per le soste. Questa tabella memorizza l'ora e l'importo dei pagamenti riferiti alle soste.

4. **parkings_statistic:** contiene le statistiche giornaliere del parcheggio, come il numero di soste totali o il guadagno medio orario.

5. **parkings_price:** contiene i record relativi alla tariffazione oraria. Questa tabella specifica la tariffazione oraria per una singola data (date) o per un giorno della settimana (day), ad esempio ogni giorno o ogni domenica, comprensiva dell'orario.



Strumenti Utilizzati

Questa sezione fornirà un elenco degli strumenti utilizzati per lo sviluppo del software descritto e una breve descrizione di come sono stati utilizzati.

Hardware

- Raspberry: il Raspberry Pi è stato utilizzato per le funzioni di server, e acquisizione dati e tramite una VPN sul server consente la connessione tra i dispositivi situati in reti diverse.
- Sensori di prossimità (ultrasuoni): i sensori di prossimità sono stati utilizzati per rilevare la presenza di veicoli e inviare i dati tramite MQTT/LoRa al server.
- Telecamere: le telecamere sono state utilizzate per leggere le targhe dei veicoli e registrare l'ora di entrata e uscita.

Software

- **Django**: Django è stato utilizzato come framework web per lo sviluppo dell'applicazione web. È stato utilizzato per gestire le richieste e la logica dell'applicazione.
- **Tailwind e Flowbite**: Tailwind e Flowbite sono stati utilizzati per la realizzazione dell'interfaccia grafica web.

- **PostgreSQL / Oracle**: PostgreSQL e Oracle sono stati utilizzati come database relazionale per memorizzare i dati del sistema.
- **Django per il testing**: Django è stato utilizzato per il testing grazie alle librerie che sfruttano un metodo basato su classi derivate dal modulo unittest di Python. Sono stati utilizzati per testare le funzioni di sistema e il funzionamento delle view.
- **Thingsboard**: Thingsboard è uno strumento fondamentale per la realizzazione e gestione dell'infrastruttura. Sono state utilizzate entrambe le istanze (server ed edge), containerizzate con Docker, per elaborare e trasferire i dati tra i dispositivi.
- **Git e Github**: Git e Github sono stati utilizzati come piattaforma di hosting per la collaborazione e il controllo versione del codice.
- **API Thingsboard**: È stata fatta una fork di una repository Github deprecata e opportunamente aggiornata (github.com/Fedefirewall/thingsboard_python_rest) per utilizzare le API Thingsboard e inviare i dati raccolti dal Raspberry Pi al server Thingsboard.
- **Yolov5 + Tesseract**: Yolov5 e Tesseract sono stati utilizzati per il riconoscimento della targa. È stato creato un dataset personalizzato per il rilevamento e successivamente addestrata la rete per leggere le targhe dei veicoli tramite la combinazione di localizzazione e OCR.
- **VPN (Wireguard)**: Wireguard è stato utilizzato per la connessione tra i dispositivi situati in reti diverse e garantire la sicurezza delle comunicazioni.

Lettura targa

La lettura della targa è una delle funzionalità fondamentali del sistema di gestione del parcheggio. Per garantire una buona efficienza del riconoscimento della targa è stato creato un dataset specifico, che comprende molte immagini di targa riprese da diverse angolazioni, con diverse luminosità e posizioni.

Il processo di riconoscimento della targa avviene tramite l'utilizzo di una rete neurale basata sull'algoritmo Yolo e il software di OCR Tesseract. In particolare, l'algoritmo Yolo viene utilizzato per identificare le regioni dell'immagine in cui è presente la targa, mentre Tesseract viene utilizzato per estrarre il testo dalle regioni identificate.

Il dataset appositamente creato viene utilizzato per addestrare la rete neurale, in modo da rendere il sistema di riconoscimento della targa il più accurato possibile.

L'addestramento della rete neurale richiede tempo e risorse, ma è essenziale per ottenere risultati di riconoscimento della targa affidabili e precisi.

Testing

Nella fase di testing del software, sono state verificate diverse caratteristiche del funzionamento del sistema, tra cui il comportamento delle view sia per gli utenti loggati che per quelli non loggati.

Per quanto riguarda le view, sono state testate le pagine di risposta e il loro contenuto, in particolare verificando la presenza di informazioni come le soste effettuate o i prezzi calcolati. Inoltre, sono stati controllati i cookie utilizzati dal sistema e la loro corretta gestione.

Inoltre, sono state testate le funzioni di sistema, in particolare confermando che i risultati ottenuti siano quelli previsti. Ad esempio, è stato verificato il corretto funzionamento del calcolo del prezzo in base alle diverse fasce orarie e il corretto utilizzo dei filtri presenti nelle form.

Per effettuare questi test, è stato utilizzato il testing automatizzato di Django, che ha permesso di eseguire i test in modo rapido ed efficace. Grazie a questa metodologia, è stato possibile individuare eventuali problemi o malfunzionamenti del sistema, consentendo di apportare le necessarie correzioni in modo tempestivo.

In sintesi, la fase di testing del software ha permesso di verificare il corretto funzionamento delle view e delle funzioni di sistema, garantendo che il software soddisfi pienamente le esigenze e le aspettative.

Possibili miglioramenti e Problematiche

1. **Migliorare l'efficienza del sistema di riconoscimento targa.** Il sistema di riconoscimento targa è uno dei componenti principali del sistema di parcheggio intelligente. Per migliorare la sua efficienza e velocità, è possibile utilizzare hardware più performante e modelli di reti neurali più affidabili.
2. **Implementare un sistema di prenotazione dei parcheggi.** Un sistema di prenotazione dei parcheggi consentirebbe agli utenti di prenotare un posto auto in anticipo. Ciò sarebbe possibile tramite l'interfaccia web e richiede un sistema che impedisca l' utilizzo di parcheggi prenotati ma non ancora occupati.

3. **Migliorare l'infrastruttura.** L'infrastruttura del sistema di parcheggio intelligente dovrebbe essere robusta e resiliente. Si potrebbe considerare l'utilizzo di replicazione hardware o software per garantire la disponibilità del sistema.
4. **Integrare il sistema con i servizi di navigazione.** L'integrazione del sistema con i servizi di navigazione GPS potrebbe aiutare gli utenti a trovare facilmente parcheggi disponibili. Inoltre, ciò potrebbe consentire al sistema di parcheggio di comunicare con i veicoli per fornire indicazioni sulla disponibilità dei parcheggi vicini.
5. **Autenticazione con targhe multiple.** Consentire ad un utente di registrare e utilizzare diverse targhe, e consente a due utenti diversi di utilizzare la stessa auto, implementando ad esempio u sistema di autenticazione che permetta ad ogni ingresso di associare correttamente la targa letta all' utente corretto.
6. **Migliorare l'esperienza utente.** Infine, è possibile migliorare l'esperienza utente tramite l'introduzione di ulteriori funzionalità come la possibilità di localizzare la propria auto all'interno del parcheggio, o di fornire feedback.
7. **Comunicazione a lungo raggio.** L'uso di dispositivi come sensori o telecamere per la rilevazione dei parcheggi dovrebbe essere pensato in modo da garantire la comunicazione anche in caso di parcheggi con dimensioni elevate. Una possibile soluzione potrebbe essere l'utilizzo di tecnologie di comunicazione a lungo raggio come LoRa per garantire la trasmissione dei dati anche su distanze maggiori.
8. **Dispositivi special purpose.** È importante utilizzare sensori e sistemi di acquisizione immagini con caratteristiche adatte ad un contesto reale, come la resistenza agli agenti atmosferici, l'affidabilità o il consumo ridotto. Ciò consentirà di ridurre i costi di manutenzione e garantire una maggiore efficienza del sistema.
9. **Telecamere in live streaming.** Gli amministratori del sistema dovrebbero avere la possibilità di visualizzare in diretta il video acquisito dalle telecamere. Questo consentirà loro di monitorare il flusso del traffico e identificare eventuali problemi o situazioni di emergenza.

Conclusioni

Il software di gestione di un multi parcheggio intelligente con gestione degli accessi e rilevamento dei posti occupati è un'applicazione complessa che richiede una solida architettura IoT per funzionare correttamente.
Le telecamere e i sensori di prossimità che inviano i dati tramite MQTT/LoRa ad un

gateway edge, costituiscono la parte di sensoristica del sistema, mentre le tecnologie come Yolo+Tesseract, Django, Thingsboard, PostgreSQL, API REST per la comunicazione tra webserver e Thingsboard forniscono il supporto necessario per la gestione degli accessi e il rilevamento dei posti occupati.
L'uso del testing automatizzato di Django per lo sviluppo di diversi test, sia per le diverse *view* che per le diverse funzioni di sistema, migliora il livello di qualità del software.

Tuttavia, il processo di sviluppo del sistema non è stato privo di sfide.
La gestione della scalabilità e il coordinamento tra le diverse tecnologie utilizzate hanno rappresentato un altro fattore critico per garantire il corretto funzionamento, inoltre la creazione del dataset personalizzato per il riconoscimento della targa e l'addestramento della rete hanno richiesto tempo e risorse significative.

In definitiva, la realizzazione di un sistema di gestione del parcheggio come quello descritto richiede un'attenta pianificazione e una conoscenza approfondita delle tecnologie utilizzate, al fine di garantire la scalabilità, l'affidabilità e la disponibilità del sistema. Tuttavia, quando gestito in modo efficiente, può fornire importanti vantaggi in termini di automazione e miglioramento dell'esperienza degli utenti del parcheggio.