Relazione Parcheggio intelligente - Progettazione e Implementazione di Sistemi Software in Rete (P.I.S.S.I.R)

Componenti del gruppo di lavoro:

Andrea Benedetto, Matricola: 20044519

Gabriele Magenta Biasina, Matricola: 20044231

Federico Mondelli, Matricola: 20044357

Il progetto in questione è un sistema di parcheggio intelligente progettato per gestire e ottimizzare l'occupazione dei posti auto in un'area di parcheggio per veicoli elettrici. Il sistema utilizza una combinazione di telecamere, sbarre, sensori e bot per monitorare e controllare gli accessi e le uscite dei veicoli, nonché per gestire le operazioni di ricarica dei veicoli elettrici.

L'obiettivo principale del progetto è migliorare l'efficienza e la sicurezza del parcheggio, riducendo i tempi di attesa e ottimizzando l'utilizzo dei posti auto disponibili. Questo viene realizzato attraverso un'infrastruttura IoT (Internet of Things) che consente di raccogliere e analizzare dati in tempo reale, fornendo agli utenti e agli operatori informazioni precise e aggiornate.

Per avviare il sistema, è necessario seguire i seguenti passaggi:

- Verificare che il database sia configurato correttamente e che tutte le tabelle richieste siano state create. Noi abbiamo utilizzato PostgreSQL con l'applicativo pgAdmin e abbiamo inserito il file script.SQL .
- Assicurarsi di aver installato il broker MQTT Mosquitto, e avviarlo da terminale, in questo modo si metterà in ascolto sulla porta 1883 e comunicherà con il Backend.
- Assicurarsi che il server sia configurato e in esecuzione, ascoltando sulla porta designata http://localhost:8080. Per avviare il Backend sarà necessario recarsi nella cartella backend-java-spark -> avviare un terminale-> digitare "gradle run".
- A questo punto rimane da avviare il Frontend, ovvero il sito web, dove l'utente potrà comunicare direttamente con i sistemi del parcheggio e monitorare i vari stati. Per avviare il frontend sarà dunque necessario recarsi nella cartella frontend-razor -> GestioneParcheggio->aprire un terminale e digitale "dotnet watch run" questo caricherà il frontend e aprirà direttamente la pagina web.
- Se si desidera testare l'ingresso, l'uscita e visualizzare il monitor dei parcheggi e lo stato del bot, basterà recarsi nella cartella sottosistemaIoT-> aprire un terminale e digitare "python simulatore.py "questo avvierà il simulatore. Una volta all'interno del simulatore sarà necessario inserire le credenziali di Amministratore, ovvero come username: "admin" e come password: "admin". Fatto ciò il simulatore sarà abilitato e pronto a ricevere le richieste.

Componenti del sistema e le loro funzioni:

- **BACKEND (Java con Spark):**
  Il backend è principalmente impostato su due classi:
  Controller (RouteHandler): Gestisce le richieste http in ingresso e invia risposte. Utilizza Spark per definire le route API e per gestire le richieste. Utilizza i metodi dalla classe
  (DAO) per eseguire operazioni di database.
  Model (DAO): Fornisce metodi per eseguire operazioni sul database. Utilizza JDCB per connettersi al database ed eseguire query SQL.

- **FRONTEND (Razor Pages):**

    - **Struttura delle Pagine:** Le pagine sono state strutturate utilizzando una combinazione di HTML e C# per generare contenuti dinamici. La struttura HTML è stata definita nelle pagine Razor (.cshtml) con la combinazione di C# per gestire la logica di business e per recuperare i dati. I file (.cshtml.cs) sono i file di backend associati alle rispettive pagina Razor (.cshtml) e segue il modello "code-behind" che contiene la logica per gestire le richieste http e interagire con i dati
    - **Progettazione Responsiva:** L'applicazione è stata progettata per essere responsiva, il che significa che si adatta a diverse dimensioni dello schermo e dispositivi.
    - **Protezione delle Pagine:** Le pagine sono state protette utilizzando un sistema di autenticazione basato su token. Quando un utente si autentica, un token JSON Web (JWT) viene generato dal backend e memorizzato nei cookie della risposta HTTP. Il token verrà quindi utilizzato per autenticare le richieste future. (Nel token ho inserito anche il tipo utente perché mi serviva per capire quali card in IndexUser mostrare all'utente: es: se utente è di tipo Premium può visualizzare la pagina per effettuare una prenotazione, viene nascosta altrimenti)
    - **Comunicazione con il Backend**: Il frontend comunica con l'API backend utilizzando API RESTful. Gli endpoint dell'API sono stati definiti nel backend e il frontend invia richieste HTTP a questi endpoint per recuperare o inviare dati.
    - **Verifica del Token:** Nel momento dell'autenticazione dell'utente al sito web, il backend .NET riceve dal backend spark il token. "AuthService" si occupa, attraverso i suoi metodi, di recuperarlo dai cookie della richiesta http, di verificarne la validità, e estrarre informazioni come nome utente. Tipo utente, qrcode (id dell'utente)... "TokenValidator" invece di occupa della validazione effettiva del token assicurandosi che esso non sia scaduto e che posso essere decodificato correttamente per recupere le informazioni
    - **Gestione degli Errori:** La gestione degli errori è stata implementata utilizzando blocchi try-catch e i messaggi di errore sono stati visualizzati all'utente utilizzando finestre modali.
    - **Validazione dei Form:** La validazione dei form è stata implementata utilizzando JavaScript. I form sono stati validati sul lato client prima di inviare i dati all'API backend.

Relazione tra i Vari Componenti:

I componenti del sistema sono interconnessi attraverso un'architettura di microservizi. Il simulatore IoT e il MWBot comunicano con il backend tramite API REST per inviare e ricevere dati relativi agli accessi, uscite e operazioni di ricarica. Il broker MQTT funge da hub centrale per le notifiche in tempo reale, trasmettendo informazioni tra il backend, i sensori e il MWBot.

Scelte Implementative e Business Rule:

Le scelte implementative sono state guidate da requisiti di scalabilità, affidabilità e sicurezza. L'uso di un'architettura basata su microservizi consente di scalare i componenti del sistema in modo indipendente, mentre l'uso di MQTT per la comunicazione in tempo reale riduce la latenza.

Tipologia di Utenti e Permessi:

- Utente non registrato: ogni utente per accedere a questo servizio deve registrarsi inserendo delle credenziali e una carta di credito valida, altrimenti non può usufruire dei servizi che questo parcheggio offre.
- Utente Base: può registrarsi, accedere e utilizzare i servizi di parcheggio e ricarica. Non ha accesso a funzioni di amministrazione, non può prenotare un parcheggio, ma può effettuare l'upgrade ad

utente premium pagando una piccola somma di 10€ come abbonamento. Può eventualmente modificare la carta con cui effettuerà i vari pagamenti. Tramite la pagina “Transazioni effettuate” vedrà lo storico delle operazioni di sosta o di ricarica che ha effettuato. Una volta effettuata la richiesta di ricarica di una propria auto all’interno del parcheggio, sarà possibile tramite la pagina “Notifica ricarica” visualizzare quando quest’ultima è pronta.

- Utente Premium: può fare le stesse operazioni dell’utente base, ma gli è consentito di prenotare un parcheggio, a patto che rispetti l’orario di arrivo e di uscita con una tolleranza di 15min, altrimenti verrà multato. L’utente premium ha già scelto appena viene iscritto come tale di pagare un abbonamento di 10€. Le prenotazioni che effettua possono essere visualizzate e annullate tramite la pagina “Mie prenotazioni”.
- Utente Admin: può modificare i prezzi riguardanti sia la sosta che la ricarica. Inoltre, può monitorare in qualsiasi momento lo stato di occupazione dei posti auto e visualizzare tutte le transazioni degli utenti.

Funzionamento simulatore IoT:

Il simulatore IoT è un componente cruciale del sistema, che emula il comportamento delle telecamere e dei sensori del parcheggio. Il tempo simulato è accelerato rispetto al tempo reale, utilizzando un fattore di scala di 15 (1 secondo reale equivale a 15 secondi simulati). Questo permette di testare e visualizzare scenari di parcheggio e di traffico in tempi più brevi.

Il simulatore interagisce con il backend per sincronizzare lo stato dei posti auto e le operazioni di ricarica, assicurando che le informazioni visualizzate agli utenti siano sempre aggiornate. La gestione del tempo simulato è cruciale per coordinare le operazioni del MWBot e per garantire che i dati siano coerenti attraverso il sistema.

Documentazione API REST:

URL	Metodo	Parametri	Body	Codici stato	Descrizione
/api/check_ricarica_effettuata	GET	req(id_ricarica), res	/	200, 404, 500	Verifica se la ricarica è stata effettuata con successo e restituisce i dettagli.
/api/signin	POST	req,res	username, password, tipoCarta, numeroCarta, dataScadenza, cvv, userType,	200,400,409, 422,500	Registra un nuovo utente e crea un account

/api/login	POST	Req,res	Username, password	200,400,401, 500	Autentica un utente e genera un Token JWT
/api/currentPrices	GET	Req, res	/	200,404,500	Recupera i prezzi attuali di sosta e ricarica
/api/currentCard	GET	Req(qrCode), res	/	200,404,500	Recupera i dettagli della carta di credito dell'utente
/api/upgradeToPremium	POST	Req,res	/	200,401,402, 404,422, 500	Aggiorna un utente esistente al tipo di utente premium
/api/upgradeToCard	POST	Req,res	tipoCarta,numeroCarta, dataScadenza,cvv	200,400,401, 404,500	Aggiorna i dati della carta di credito di un utente
/api/modificaPrezzi	POST	Req,res	CostoSosta,CostoRicarica	200,400,500	Modifica i prezzi di sosta e ricarica
/api/cancellaPrenotazione	DELETE	Req, res	Id_prenotazione	200,400,500	Cancella una prenotazione specifica
/api/checkParkedVehiclesOfUser	GET	Req, res	qrCode	200,400,500	Verifica se un utente ha veicoli attualmente parcheggiati
/api/transazioniUtenti	POST	Req,res	dataOraInizio,DataOraFine,filters	200,400,404, 500	Recupera le transazioni filtrate degli utenti
/api/ricariche	GET	Req,res	/	200,500	Recupera tutte le ricariche registrate
/api/confermaRicarica	POST	Req, res	percentualeRicarica,veicolo	200,400,401, 404,500	Conferma una richiesta di ricarica per un veicolo
/api/prenotaposto	POST	Req,res	Ingresso, uscita	200,400,409, 500	Effettua una prenotazione di un posto auto

/api/veicoliNelParcheggio	GET	Req,res	qrCode	200,400	Recupera i veicoli parcheggiati associati all'utente
/api/cardTypes	GET	Req,res	/	200	Recupera i tipi di carte di credito supportate
/api/richiediRicarica	POST	Req,res	/	200,500	Richiede la prossima ricarica per il veicolo
/api/sensore_occupazione	POST	Req,res	Id_sensore, stato_occupazione, username, targa	200,400,401, 500	Registra lo stato di occupazione di un sensore
/api/ricarica_effettuata	POST	Req,res	Id_ricarica, timestamp, kw_usati	200,401,500	Registra il completamento di una ricarica
/api/verifica_ingresso	POST	Req,res	Username, targa, timestamp	200,400,401, 404,409,500	Verifica l'ingresso di un veicolo nel parcheggio
/api/verifica_uscita	POST	Req, res	Username, targa, timestamp	200,400,401, 404,409,500	Verifica l'uscita di un veicolo nel parcheggio
/api/leave	POST	Req,res	postoAuto	200,409,500	Permette a un veicolo di lasciare il parcheggio
/api/transazioniUtente	GET	Req, res	qrCode	200,500	Recupera tutte le transazioni associate a un utente specifico
/api/statoParcheggio	GET	Req, res	/	200,500	Recupera lo stato di tutti i posti auto nel parcheggio
/api/mieprenotazioni	GET	Req, res	qrCode	200,500	Recupera tutte le prenotazioni attive associate a un utente specifico

/api/all_active_reservations	GET	Req, res	/	200,500	Recupera tutte le prenotazioni attive senza veicolo associato
/api/all_parking_spots_active_with_vehicle	GET	Req, res	/	200,500	Recupera tutte le prenotazioni attive con veicolo associato
/api/aggiornamento_ricarica	POST	Req, res	Id_veicolo, capacità, modello, id_ricarica, durata, id_posto_auto	200,401, 500	Aggiorna i dettagli di una ricarica in corso
/api/all_parking_spots	GET	Req,res	/	200,500	Recupera tutti i posti auto disponibili nel parcheggio
/api/parking/:spot_id	GET	Req, res	:spot_id	200,500	Recupera lo stato di occupazione di un posto auto specifico

Documentazione MQTT:

TOPIC	Formato del messaggio	Descrizione
parcheggio/verifica_ingresso	{"tipo",”username", "targa", "timestamp", "token"}	Verifica l'ingresso di un veicolo nel parcheggio
parcheggio/verifica_uscita	{"tipo",”username", "targa", "timestamp", "token"}	Verifica l'uscita di un veicolo nel parcheggio
parcheggio/sensore_occupazione	{"id_sensore",”stato_occupazione”,”username", "targa", "timestamp", "token"}	Notifica lo stato di occupazione di un sensore
parcheggio/richiesta_tutte_prenotazioni_attive	{“message (Richiesta prenotazioni attive)”,” token”}	Richiede tutte le prenotazioni attive nel parcheggio

parcheggio/richiesta_tutte_prenotazioni_attive_auto_dentro	{“message (Richiesta prenotazioni attive dentro)”,” token”}	Richiede tutte le prenotazioni attive per veicoli già dentro il parcheggio
parcheggio/richiesta_ricariche	{“message (richiesta ricariche)”,” token”}	Richiede la lista delle ricariche da effettuare
parcheggio/ricarica_effettuata	{"id_ricarica",” timestamp”,”kw_usati", "token"}	Notifica che una ricarica è stata completata
parcheggio/multe	{"multe",” username", "targa", "timestamp_multa", "token"}	Notifica l'emissione di multe per veicoli
parcheggio/aggiornamento_ricarica	{“id_veicolo",” capacità", "modello", "id_ricarica",” durata”,”id_posto_auto”, "token"}	Aggiorna lo stato di una ricarica in corso
parcheggio/presa_in_carico_bot	{“message (Bot preso in carico)”,”id_veicolo”,”id_posto_auto”, “token”}	Notifica che il bot ha preso in carico una ricarica o operazione di parcheggio