

Progetto Pissir 2023-2024

Guido Lorenzo Broglio (20043973)
 Matteo Cartieri (20044003)
 Alessandro Fatone (20019780)

1 Descrizione progetto

Immaginiamo che esistano ricaricatori wireless mobili (MWbot), capaci autonomamente di spostarsi sotto le auto elettriche e ricaricarle per induzione. Ciò apre ad una gestione agile della ricarica in cui gli utenti possono lasciare l'auto parcheggiata e richiedere che venga caricata mentre fanno le loro commissioni.

Quando la batteria del auto raggiunge la percentuale di carica richiesta dal utente, l'MWbot può spostarsi per caricare altre auto. Si suppone che l'MWbot possa riconoscere il modello d'auto da ricaricare e in particolare la capacità in kW della batteria. Inoltre ogni posto auto è dotato di sensori per monitorarne l'occupazione.

Obiettivo del progetto è la progettazione e l'implementazione di un sistema di gestione di un parcheggio smart dotato di MWbot, per semplicità si può assumere la presenza di un singolo MWbot al interno del parcheggio.

Gli utenti del sistema sono di tre tipi: utenti base che usufruiscono dei servizi di parcheggio e ricarica, utenti premium che possono in aggiunta prenotare preventivamente il posto, e l'amministratore che da remoto può monitorare i vari componenti del sistema e lo stato di occupazione del parcheggio.

2 Diagramma delle classi di dominio

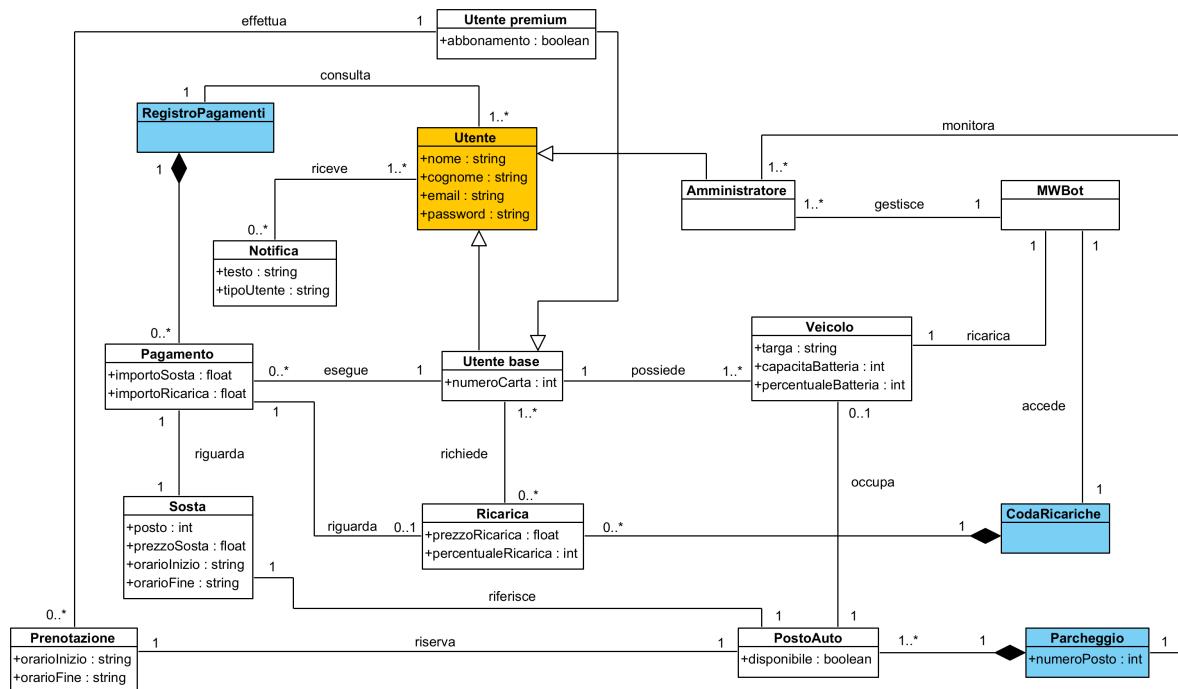


Figura 1: Diagramma delle classi

3 Diagramma dei casi d'uso

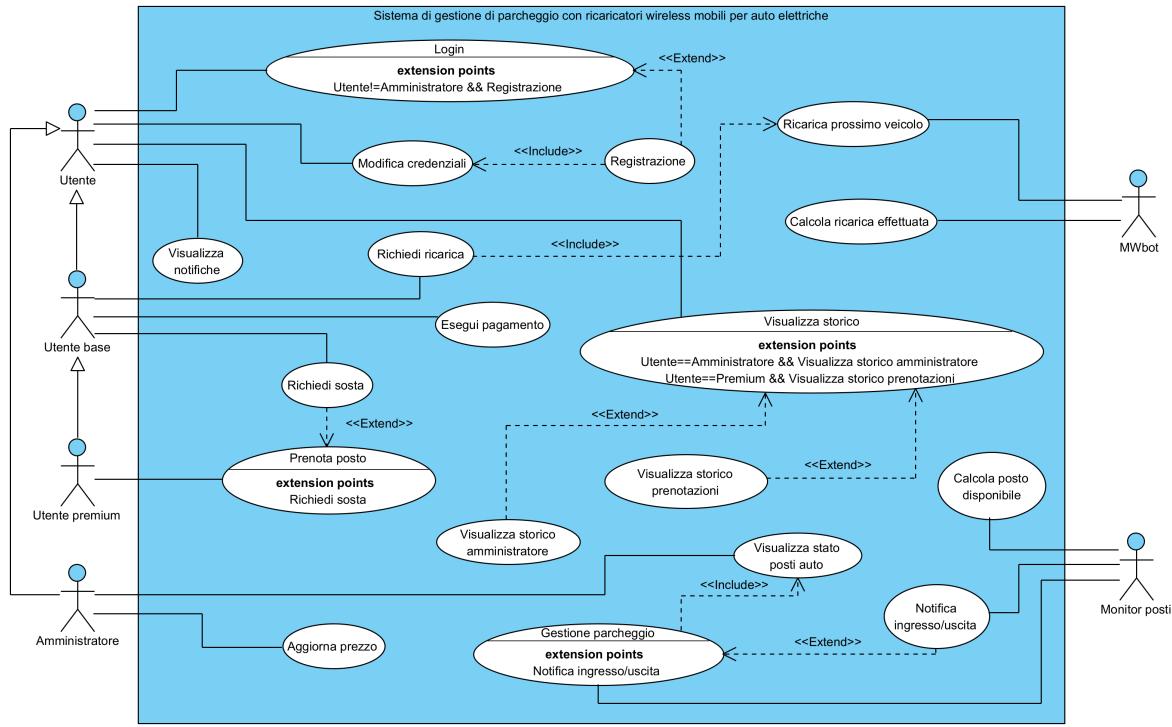


Figura 2: Diagramma dei casi d'uso

3.1 Login, Registrazione, e Modifica credenziali

Gli utenti inseriscono il proprio username e la propria password per accedere o registrarsi al sistema. Per registrarsi, sarà obbligatorio inserire anche email, codice della carta di credito e la targa di almeno un veicolo. Modifica credenziali viene richiamata solo durante la registrazione (quando le credenziali passano dallo stato nullo a quello esistente) oppure quando l'utente ne fa richiesta.

3.2 Richiesta sosta

Gli utenti base e premium hanno la possibilità di richiedere la sosta presentandosi all'ingresso del parcheggio. Il veicolo degli utenti viene identificato all'entrata, attraverso sensori in grado di rilevare la targa. Se il veicolo non è associato all'account dell'utente, la sbarra presente all'ingresso rimarrà abbassata e l'utente non potrà entrare. Ovviamente, nel caso in cui tutti i posti fossero occupati, l'entrata non sarà disponibile e nessun utente potrà accedere al parcheggio.

3.3 Richiesta ricarica

Supponendo che il veicolo degli utenti sia elettrico, quando essi sostano nel parcheggio possono avere la possibilità di ricaricare la batteria del proprio veicolo completamente, oppure fino ad una certa percentuale prestabilita. Quando l'utente richiede la ricarica, la lista d'attesa di "Ricarica prossimo veicolo" viene incrementata. La lista è utilizzata dal MWbot per permettergli di gestire la ricarica dei vari veicoli.

3.4 Visualizza notifiche

Gli utenti possono decidere di essere notificati per i pagamenti che effettuano, per la percentuale di ricarica raggiunta, per lo stato e la data di prenotazione di un posto, o per altri tipi di eventi. Gli amministratori, invece, ricevono notifiche sullo stato del parcheggio come lo stato dei dispositivi, dei posti auto e delle prenotazioni relative ad un posto.

3.5 Visualizza storico

Lista dello storico di tutti i movimenti relativi agli utenti che hanno usufruito dei servizi del parcheggio. Si potranno visualizzare la data, il posto occupato, il costo della sosta e dell'eventuale ricarica.

3.6 Visualizza storico prenotazioni

Se un utente premium ha effettuato delle prenotazioni, quando accede allo storico può anche visualizzare le sue prenotazioni passate. Un utente base è in grado di vedere le prenotazioni solo nel caso in cui sia stato premium in passato (e abbia effettuato delle prenotazioni).

3.7 Visualizza storico amministratore

L'utente amministratore può accedere e visualizzare lo storico di tutti gli utenti, che include tutte le transazioni effettuate.

3.8 Esegui pagamento

Gli utenti base e premium pagano la sosta, e anche l'eventuale ricarica se essi usufruiscono del servizio. Gli utenti possono anche pagare un abbonamento, e con ciò è possibile ottenere o mantenere la funzionalità della prenotazione del posto auto (e permette all'utente base di diventare premium). Il pagamento può essere effettuato solamente online e gli utenti non possono uscire dal parcheggio se non pagano la sosta (e l'eventuale ricarica se è stata richiesta).

3.9 Prenota posto

L'utente premium riserva, in anticipo, un posto auto per la sosta.

3.10 Aggiorna prezzo

L'amministratore può modificare il costo €/ora della sosta e €/Kw della ricarica.

3.11 Visualizza stato posti auto

L'amministratore vede quali posti sono occupati e da quali veicoli.

3.12 Notifica ingresso/uscita

Il sensore rileva il veicolo appena entrato o uscito dal parcheggio, tenendo traccia della sua targa. Una volta che il sistema ha la conferma del pagamento da parte dell'utente, libererà il parcheggio occupato e lo renderà nuovamente disponibile.

3.13 Gestione parcheggio

Il sensore tiene traccia dello stato dei posti, se sono occupati e da che tipo di veicolo è occupato. I dati raccolti sono passati all'utente amministratore, al fine di monitorare tutti gli stati del parcheggio.

3.14 Ricarica prossimo veicolo

L'Mwbots inizia la ricarica del veicolo in testa alla coda dei veicoli che ne hanno fatto richiesta.

3.15 Calcola ricarica effettuata

L'Mwbots restituisce il valore in kW della ricarica appena effettuata su un veicolo, in modo che si possa calcolarne il prezzo.

3.16 Calcola posto disponibile

Il sistema calcola i posti disponibili (e di conseguenza anche quelli occupati) al fine di restituirli sul monitor all'ingresso del parcheggio.

4 Diagrammi di architettura

I diagrammi di architettura illustrano la struttura generale del sistema di gestione del parcheggio con il ricaricatore wireless mobile (MWbot). Questi diagrammi forniscono una visione d'insieme dei vari componenti software e hardware del sistema, mostrando come essi interagiscono tra di loro per fornire le funzionalità richieste.

Il sistema è progettato seguendo un'architettura modulare, basata su microservizi, al fine di garantire scalabilità e manutenibilità. Ogni modulo è responsabile di un aspetto specifico del sistema, come la gestione delle prenotazioni, la gestione delle ricariche, o la gestione dei pagamenti. L'interazione tra i moduli avviene tramite API REST e messaggistica MQTT, assicurando una comunicazione efficiente e asincrona tra i componenti.

4.1 Diagramma esagonale

Il diagramma esagonale rappresenta l'architettura del sistema. Nel Back-end, la parte software è suddivisa in "moduli" (microservizi), che garantiscono una gestione scalabile e affidabile delle operazioni, e all'interno del sistema essi possono avere un ruolo centrale oppure un ruolo periferico. I moduli centrali compongono la Business logic, che è definita da SmartParkCore (che gestisce l'accesso al database tramite il pattern DAO per le operazioni CRUD), SmartParkManager (che coordina la logica interna dell'applicazione, gestendo le prenotazioni e i pagamenti) e SmartParkIoT (che gestisce e coordina la comunicazione con i dispositivi fisici del parcheggio attraverso l'utilizzo del protocollo MQTT per monitorare, per esempio, l'occupazione dei posti auto e lo stato del MWBot).

Nei moduli periferici, invece, è presente SmartParkPhilipsHue che fa da ponte e interagisce con le lampadine Hue (simulatori per i posti del parcheggio) per notificare lo stato dei posti tramite luci. Sono anche presenti MWBot (robot responsabile per le ricariche dei veicoli elettrici) e MonitorPosti (che visualizza l'occupazione dei posti), che sono i dispositivi fisici IoT presenti all'interno del Back-end, e come già accennato essi interagiscono direttamente o indirettamente con SmartParkIoT tramite MQTT (sempre attraverso l'utilizzo del broker Mosquitto) in modo asincrono, al fine di raccogliere i dati dei sensori e attivare gli attuatori. Anche SmartParkWebAPI è un modulo periferico che funge da intermediario tra il Back-end e il Front-end, ed esso consente la comunicazione con l'applicazione web (SmartParkWebApp) attraverso l'utilizzo di chiamate API REST.

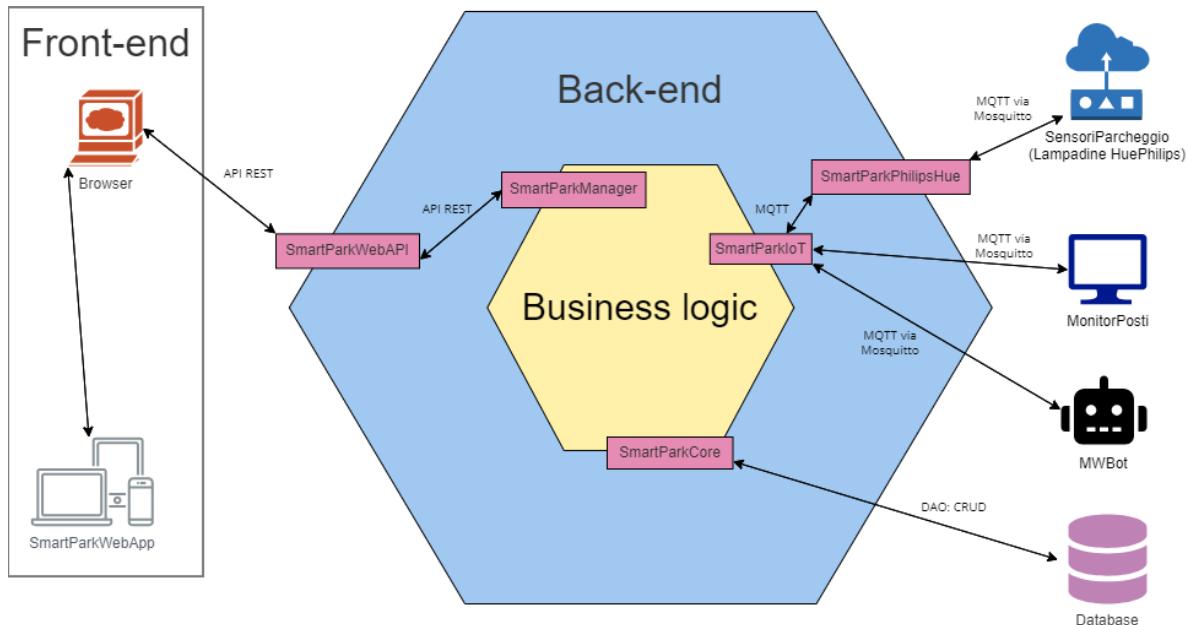


Figura 3: Diagramma esagonale per l'architettura del sistema

4.2 Diagramma dei componenti

Il diagramma dei componenti descrive i principali elementi del sistema e le loro interazioni. Ognuno di essi è responsabile di una specifica funzionalità, ad esempio, il componente "GestorePrenotazioni" si occupa di ricevere e processare le richieste di prenotazione dei posti auto, mentre il componente di "GestoreRicariche" gestisce la coda di veicoli in attesa di ricarica e comunica con l'MWbot per avviare il processo di ricarica. La comunicazione con i componenti avviene, principalmente, con il database centrale, attraverso le API REST per i componenti dell'interfaccia web, e tramite l'utilizzo del broker MQTT per inviare-ricevere notifiche e aggiornamenti di stato dai sensori e dagli attuatori del sistema.

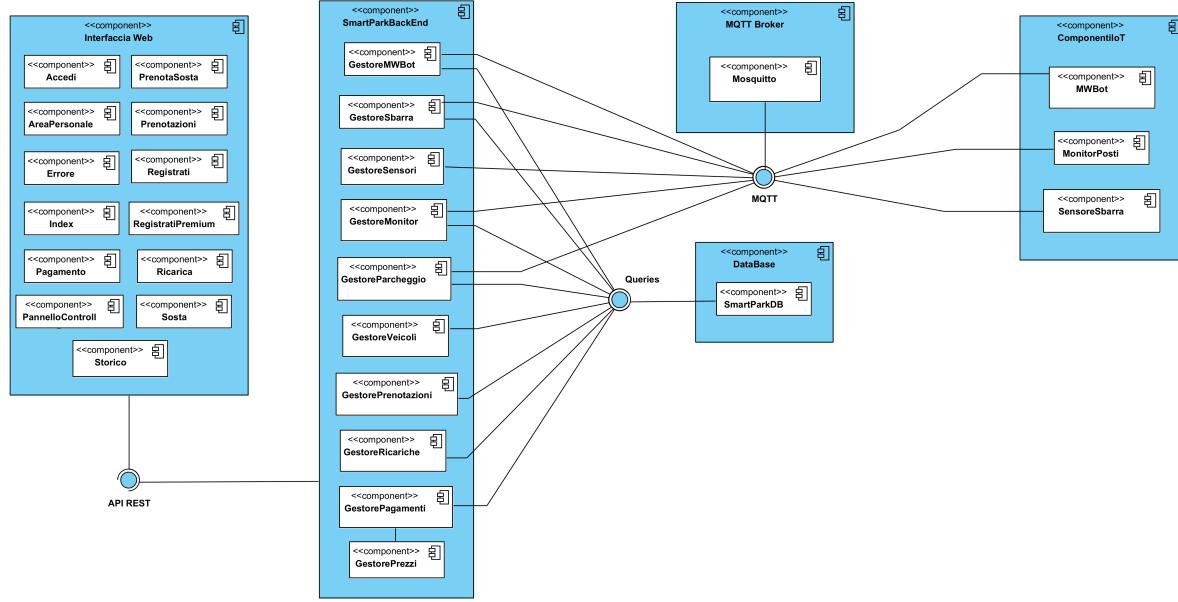


Figura 4: Diagramma dei componenti che definiscono il sistema

4.3 Diagramma dei package

Il diagramma dei package rappresenta la struttura organizzativa della soluzione del sistema. I package principali rappresentano ogni microservizio (progetto) all'interno della soluzione, mentre i sottopackage raffigurano la struttura interna del progetto, che raggruppa le classi (definite nei diagrammi delle classi) e le interfacce che implementano specifiche funzionalità, come la gestione della sessione per gli utenti, la logica di business per le prenotazioni e le ricariche, e l'interfaccia di comunicazione con i dispositivi IoT. Questo diagramma aiuta a visualizzare la separazione delle responsabilità e facilita la manutenibilità e l'estensibilità del codice.

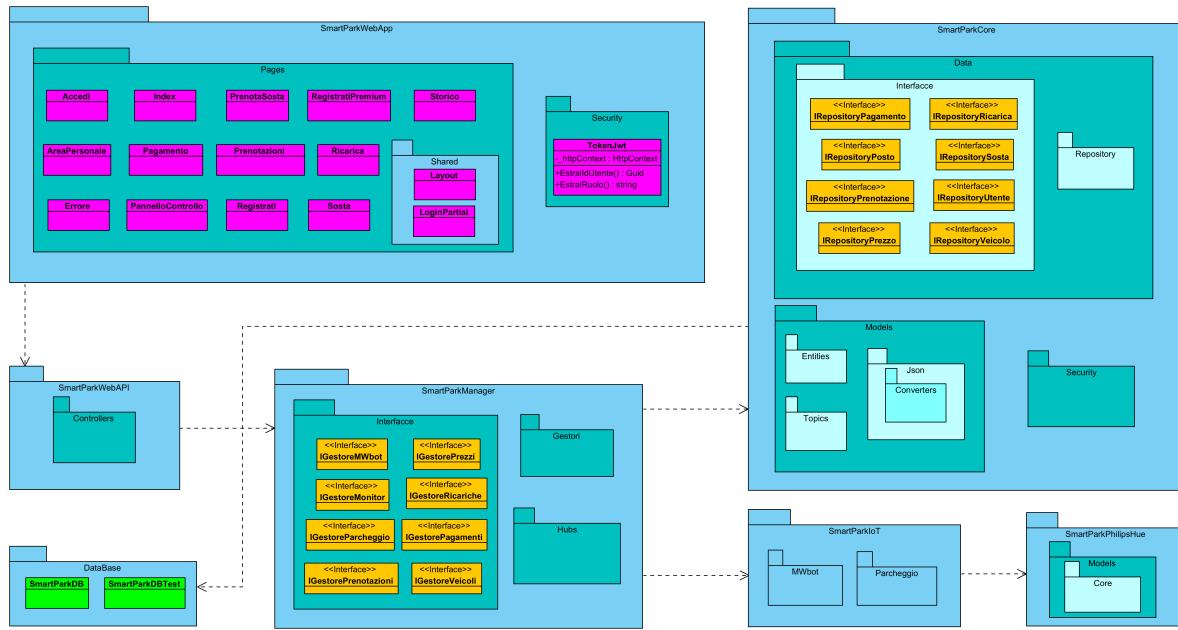


Figura 5: Diagramma dei package

5 Diagrammi delle classi

I diagrammi delle classi rappresentano la struttura e le relazioni tra le classi all'interno di ciascun progetto della soluzione. Come già accennato, ogni progetto contenuto nella soluzione è un microservizio, quindi è un'unità indipendente e modulare che si occupa di un determinato compito specifico all'interno dell'architettura. Questo permette un alto livello di coesione e bassa accoppiatura tra i vari componenti del sistema, con maggiore scalabilità, manutenibilità e facilità di aggiornamento o sostituzione di singoli moduli, evitando di influenzare il funzionamento generale dell'applicazione.

Ogni microservizio ha una sua responsabilità ben definita: ad esempio, SmartParkCore gestisce le entità e le operazioni CRUD principali, SmartParkIoT si occupa della gestione dei dispositivi IoT nel parcheggio, SmartParkManager coordina le attività di gestione, SmartParkPhilipsHue si concentra sull'integrazione con i dispositivi Philips Hue, e SmartParkWebAPI fornisce le API necessarie per l'interazione con il client e client esterni.

5.1 SmartParkCore

Il progetto SmartParkCore contiene le classi fondamentali e le entità che rappresentano i dati e i servizi principali dell'applicazione, così come le entità e le operazioni CRUD.

Il progetto è rappresentato attraverso l'utilizzo di molteplici diagrammi, in modo da descrivere dettagliatamente tutti i package e tutte le classi contenute al suo interno.

5.1.1 Repository

Il primo diagramma rappresenta le classi contenute nel package Repository, ed esse interagiscono con il database per le operazioni CRUD (Create, Read, Update, Delete). Ogni classe del Repository implementa la propria interfaccia, il che garantisce un'architettura più modulare e flessibile. Le classi come RepositoryUtente, RepositoryVeicolo e RepositorySosta gestiscono l'interazione con il database per gli utenti, i veicoli e le soste, mentre RepositoryRicarica, RepositoryPagamento e RepositoryPrenotazione si occupano di gestire ricariche, pagamenti e prenotazioni.

Le interfacce, come IRepositoryUtente, IRepositoryVeicolo, e IRepositorySosta, consentono di astrarre i dettagli dell'implementazione, facilitando il testing unitario e rendendo il codice più manutenibile. L'uso delle interfacce assicura che ogni classe del Repository rispetti i contratti predefiniti e renda possibile la sostituzione o il miglioramento delle implementazioni senza influenzare il resto dell'applicazione.

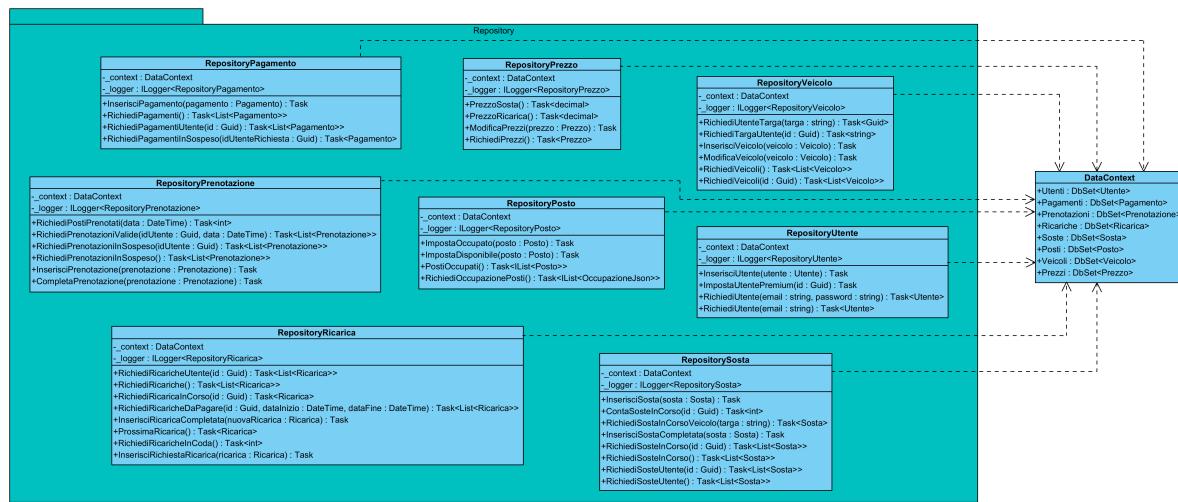


Figura 6: Diagramma delle classi Repository e DataContext

5.1.2 Entities

Il diagramma delle entità mostra le classi principali utilizzate per modellare i dati del dominio, contenute all'interno del package "Entities". Ad esempio, la classe Utente contiene le informazioni dell'utente, come nome, cognome, email, numero di carta e i veicoli associati. La classe Veicolo, associata all'utente, rappresenta le informazioni del veicolo, come la targa e la percentuale di batteria. La classe Prenotazione gestisce i dati relativi alla prenotazione di un posto nel parcheggio. Ci sono anche altre classi come Ricarica, Sosta e Pagamento che si occupano rispettivamente di gestire i dati di una ricarica del veicolo, della sosta in un posto e del pagamento associato.

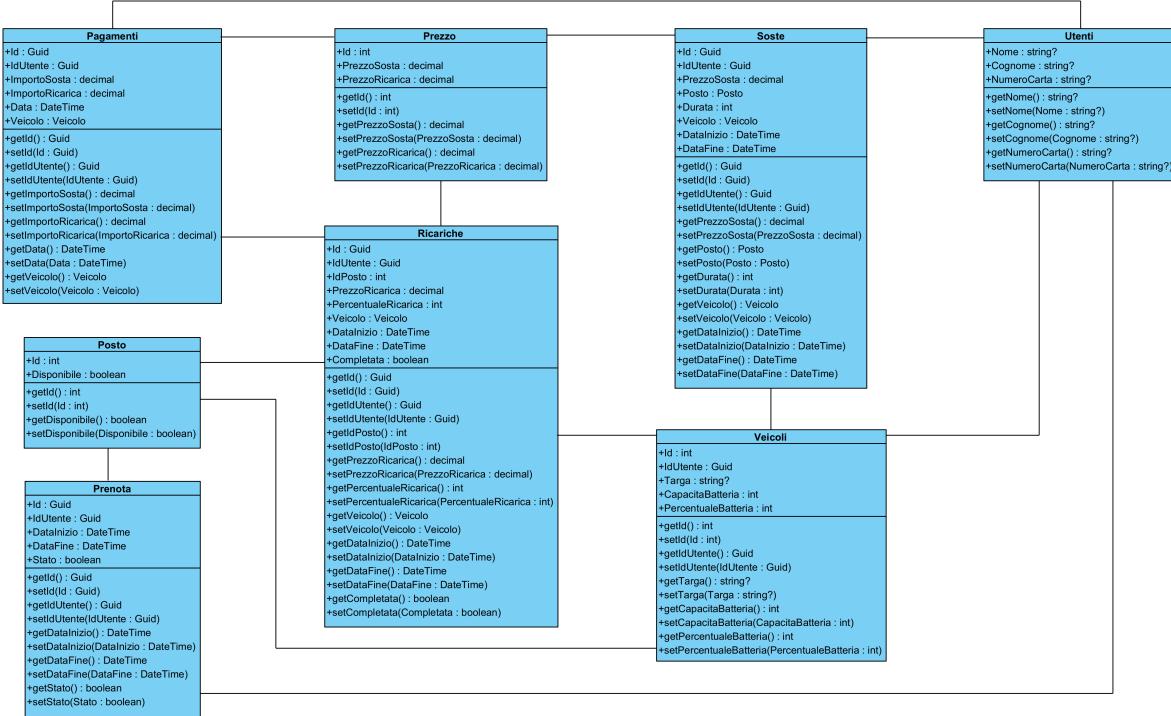


Figura 7: Diagramma per le classi Entities in SmartParkCore

5.1.3 Json, Converters e Topics

Il diagramma successivo rappresenta le classi contenute nei package "Json" e "Topics", presenti all'interno di "Models" che è allo stesso livello di "Entities". Queste classi gestiscono la serializzazione e deserializzazione dei dati in formato JSON, che torna utile per la comunicazione con sistemi esterni come MQTT. Le classi come RicaricaJson, VeicoloJson e UtenteTargaJson vengono utilizzate per rappresentare i dati in formato JSON inviati o ricevuti da altri servizi, e principalmente vengono utilizzate per formattare le risposte del server in modo compatibile con le API REST, garantendo una comunicazione efficiente e standardizzata con il client.

La classe Topics, invece, gestisce gli argomenti MQTT usati per la comunicazione con i dispositivi IoT.

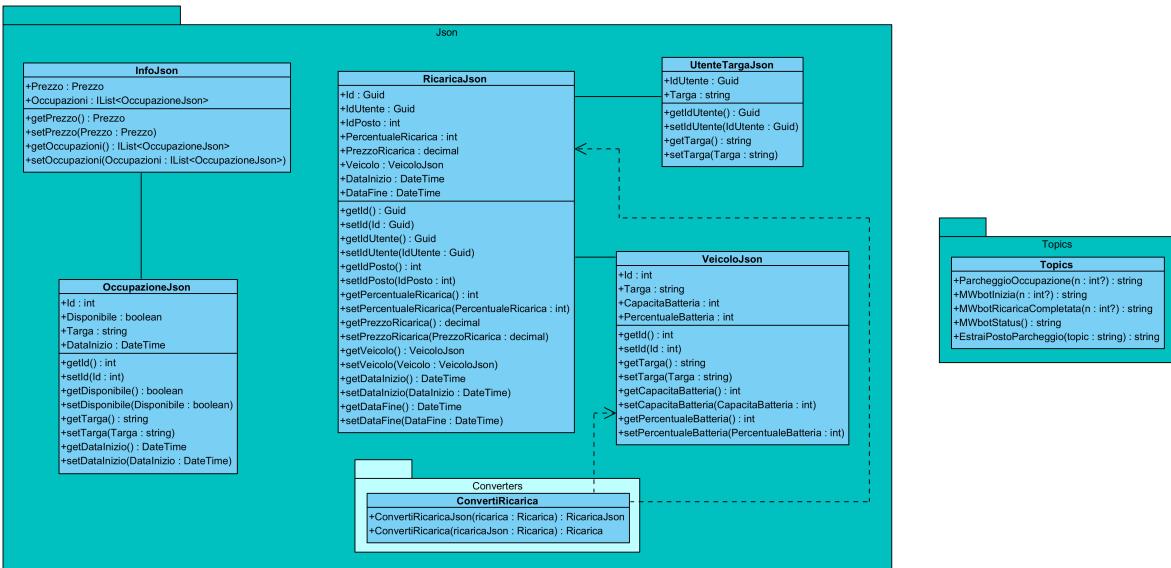


Figura 8: Diagramma per le classi contenute nei package Json, Converters e Topics

5.1.4 Security

Infine, il diagramma "Security" si concentra sulle classi che gestiscono l'accesso e la protezione dei dati del sistema. Le classi Accesso e Registrazione sono utilizzate per gestire la registrazione e l'autenticazione degli utenti. La classe TokenJson rappresenta il token JWT utilizzato per l'autenticazione e l'autorizzazione degli utenti nel sistema.

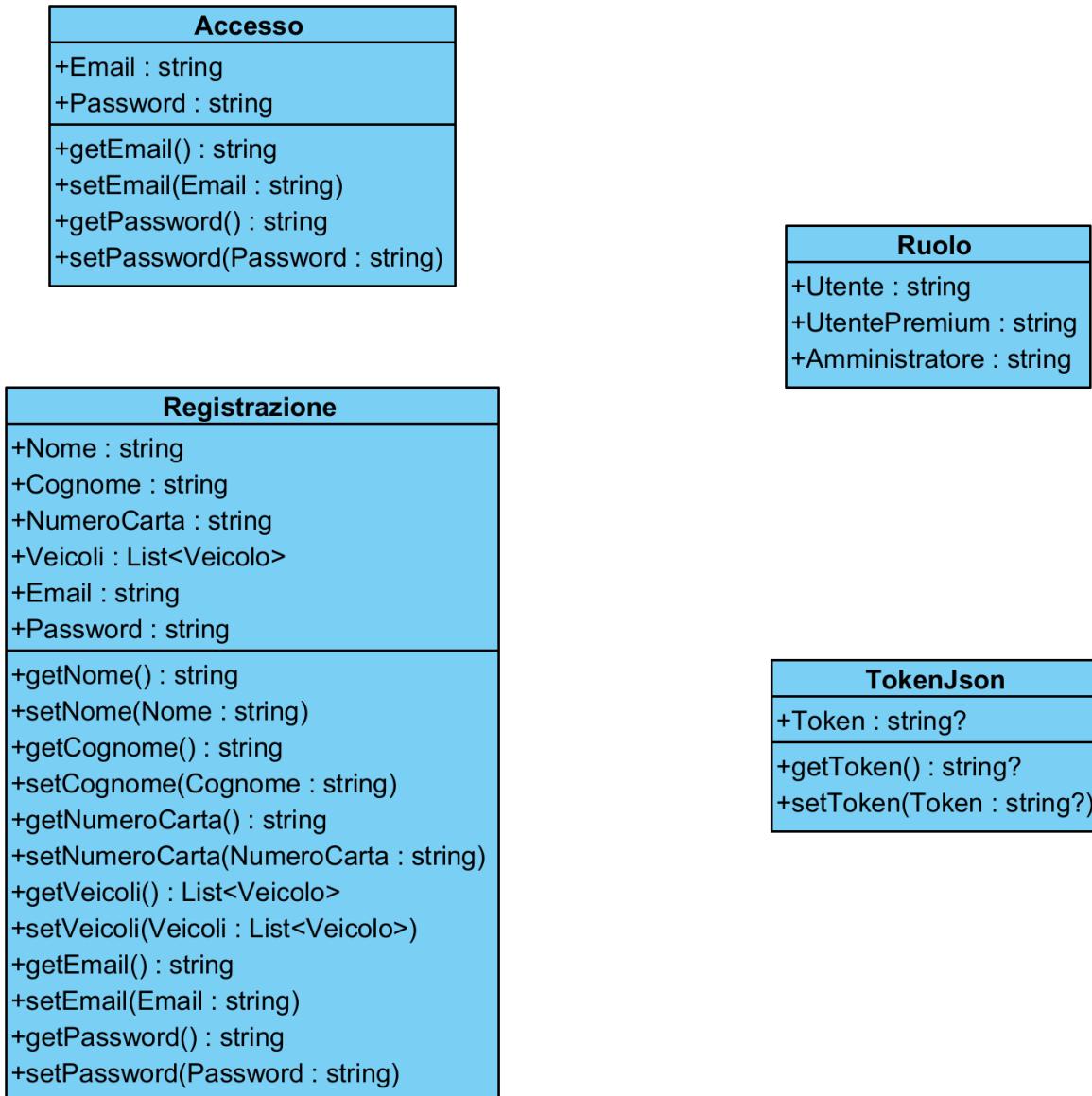


Figura 9: Diagramma che raffigura le classi Security in SmartParkCore

5.2 SmartParkIoT

Il progetto SmartParkIoT gestisce e dirige la comunicazione dei vari dispositivi IoT, al fine di monitorare e controllare lo stato del parcheggio e delle ricariche dei veicoli. Il package "MWbot", che contiene un'unica classe con lo stesso nome, si occupa della ricarica automatizzata dei veicoli elettrici, e interagisce con dispositivi IoT attraverso MQTT per monitorare e gestire lo stato di ricarica dei veicoli.

Nel package "Parcheggio", invece, è contenuta la classe MonitorPosti che tiene traccia del numero di posti occupati nel parcheggio e invia notifiche quando un posto diventa disponibile, facilitando la gestione dei parcheggi in tempo reale.

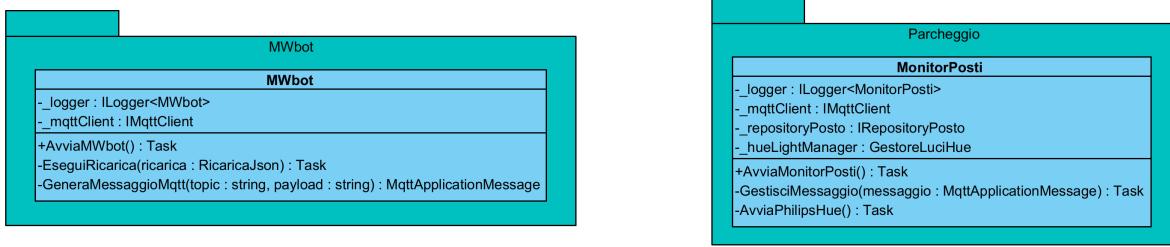


Figura 10: Classi dei package Bot e Parcheggio, che definiscono SmartParkIoT

5.3 SmartParkManager

Il progetto SmartParkManager gestisce la logica principale per la gestione delle ricariche, delle soste e dei pagamenti, coordinando le varie operazioni tra le diverse entità del sistema. Le classi contenute nel package "Gestori" coordinano le operazioni principali nel sistema. Le classi come GestoreParcheggio, GestoreRicariche e GestorePagamenti gestiscono le operazioni di parcheggio, ricarica e pagamento, interfacciandosi con i repository e i servizi IoT per garantire che tutte le operazioni siano eseguite correttamente. La classe GestorePrenotazioni si occupa della gestione delle prenotazioni di posti per gli utenti premium, mentre GestoreMonitor comunica con i sensori per aggiornare lo stato del parcheggio e dei posti, interfacciandosi con il gestore del parcheggio per notificare gli eventi di ingresso e uscita dei veicoli. Come per il Repository, anche ogni classe di SmartParkManager ha la propria interfaccia, (come IGestoreParcheggio, IGestoreRicariche, e così via) che ne permette la sostituzione o l'aggiornamento senza impattare il resto del sistema.

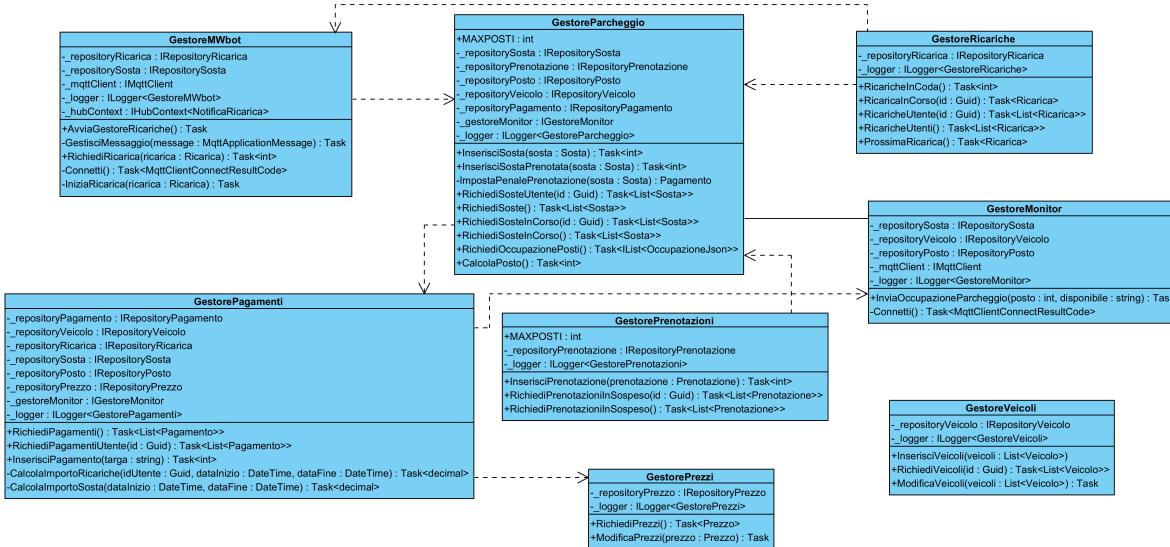


Figura 11: Classi del package Gestori, che definisce SmartParkManager

5.4 SmartParkPhilipsHue

Il progetto SmartParkPhilipsHue integra il sistema di gestione del parcheggio con le luci intelligenti Philips Hue per migliorare l'esperienza dell'utente e la gestione visiva degli spazi del parcheggio. Il diagramma delle classi mostra le interazioni tra la classe GestoriLuciHue, che gestisce il controllo e la configurazione delle luci intelligenti, e le classi che rappresentano le luci e i loro stati. GestoriLuciHue invia comandi alle luci, come il cambio di colore o lo stato di accensione, attraverso le API di Philips Hue.

La classe Luce rappresenta una singola luce intelligente nel sistema, con attributi come Id, Nome e Stato, mentre la classe Stato gestisce lo stato della luce, inclusi attributi come la luminosità, il colore e lo stato di accensione o spegnimento.

La funzione principale di questo modulo è quella di sincronizzare le luci con lo stato del parcheggio, cambiando il colore della luce di un posto libero o occupato, di fatto rappresentando un monitor d'ingresso. Questa integrazione facilita la gestione del parcheggio, fornendo un feedback visivo in tempo reale su posti disponibili o occupati, e migliorando l'esperienza utente complessiva nel parcheggio.

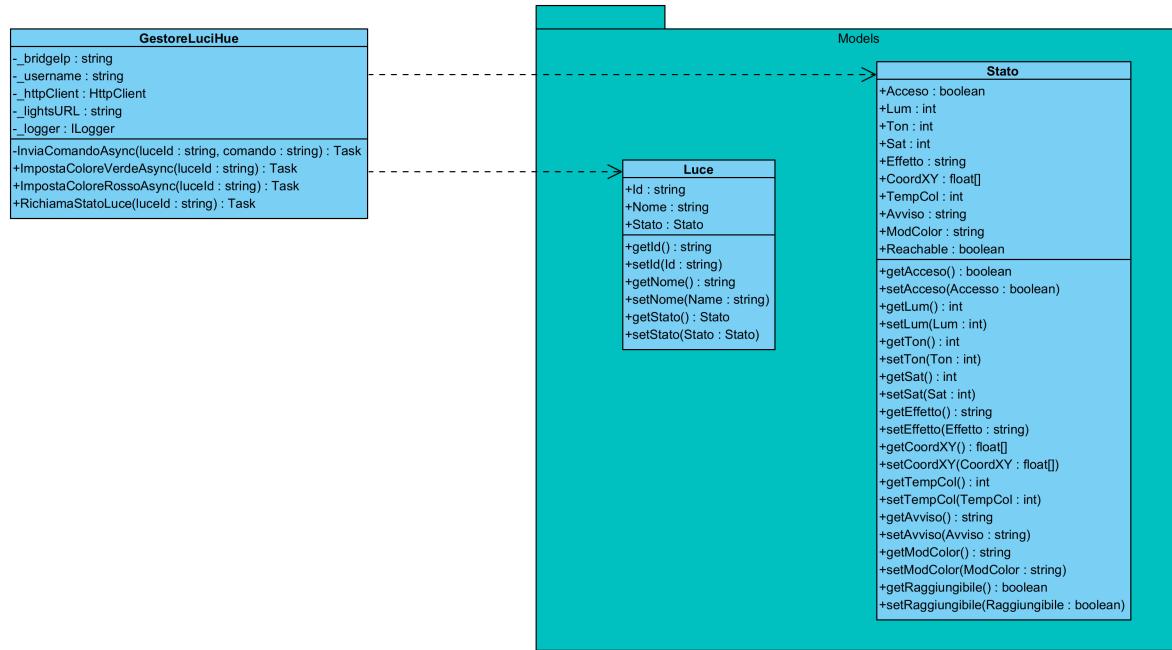


Figura 12: Classi che definiscono il microservizio SmartParkPhilipsHue

5.5 SmartParkWebAPI

Il progetto SmartParkWebAPI fornisce l'interfaccia di comunicazione tra il sistema e il mondo esterno tramite API REST, permettendo agli utenti di interagire con il sistema. Il package dei "Controllers" implementa, attraverso le sue classi, le API principali che espongono i servizi del sistema agli utenti finali. La classe AccessoController gestisce l'accesso e la registrazione degli utenti al sistema. Le classi RicaricheController, SosteController e PagamentiController espongono le operazioni di richiesta di ricarica, gestione delle soste e pagamenti tramite API REST. Queste classi ricevono le richieste HTTP, le elaborano e restituiscono le risposte agli utenti o alle applicazioni client.

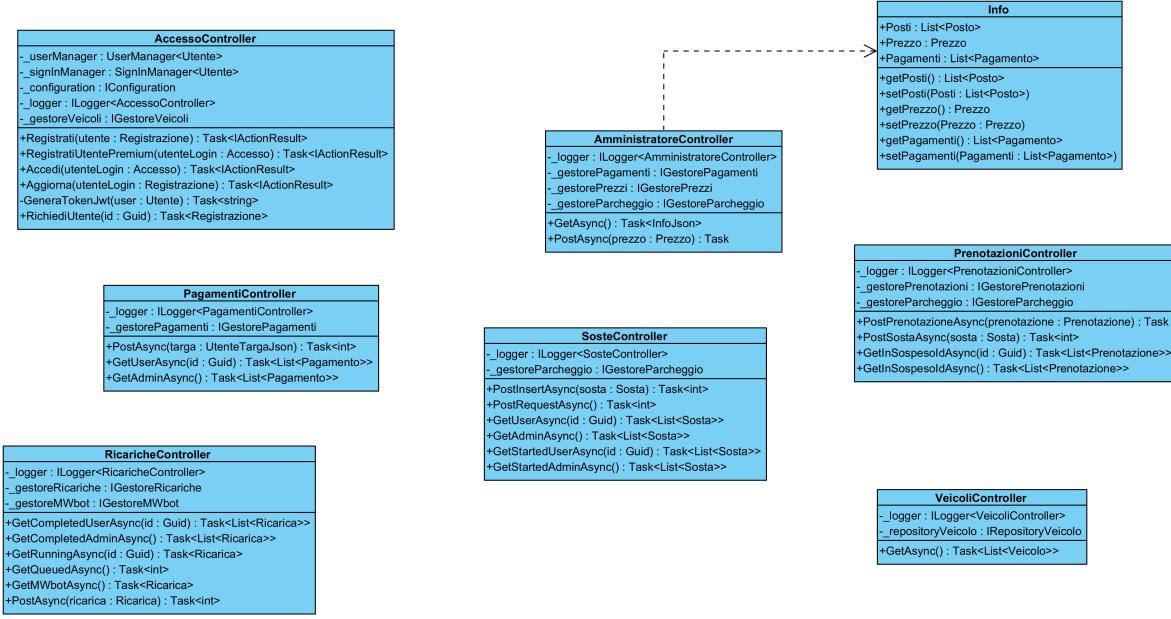


Figura 13: Classi contenute in Controllers, che costituiscono il microservizio SmartParkWebAPI

6 Diagrammi di sequenza

Questa sezione presenta i diagrammi di sequenza che illustrano il flusso delle operazioni all'interno del sistema in risposta a specifici eventi o richieste da parte degli utenti.

I diagrammi di sequenza offrono una visione dettagliata delle interazioni tra i vari componenti del sistema durante l'esecuzione di funzionalità chiave, mostrando l'ordine cronologico dei messaggi scambiati tra i componenti.

6.1 Calcola posto disponibile

Questo diagramma illustra il flusso di interazione tra i componenti del sistema quando un utente richiede di calcolare la disponibilità di posti auto nel parcheggio. Il sistema riceve la richiesta dall'utente tramite l'interfaccia utente, consulta il database per verificare lo stato corrente dei posti auto, e ritorna l'informazione al frontend, che la visualizza all'utente.

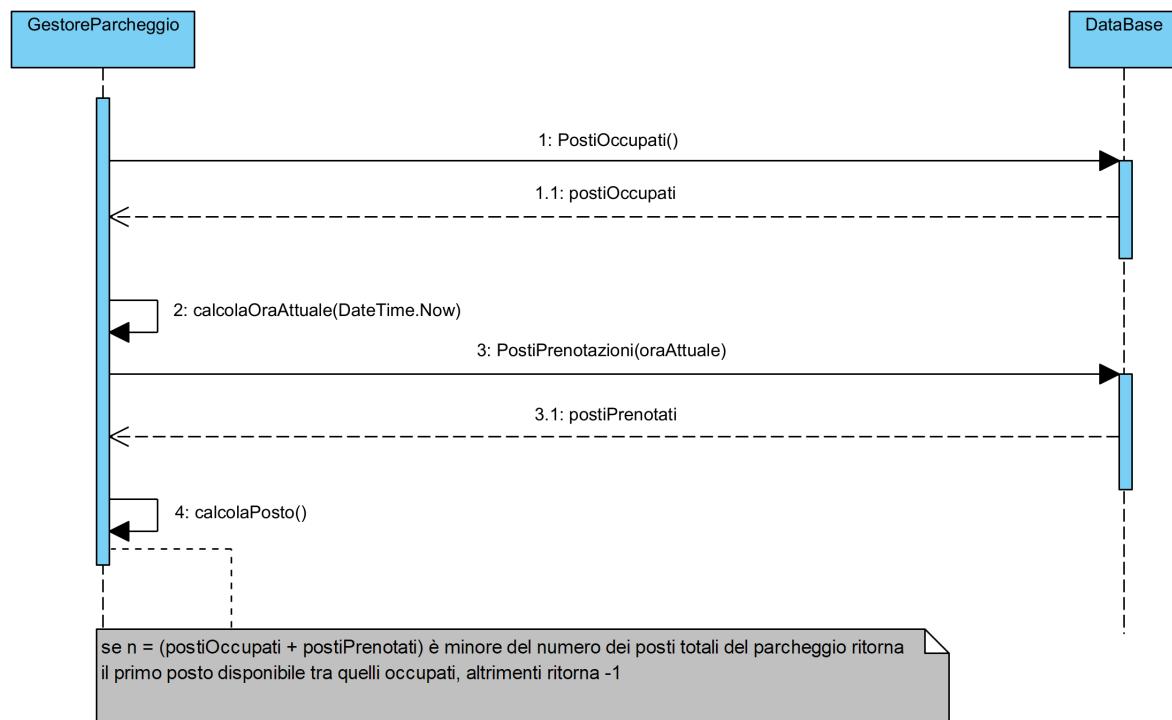


Figura 14: Diagramma di sequenza - Calcola posto disponibile

6.2 Esegui pagamento

Il diagramma di sequenza relativo all'esecuzione del pagamento illustra il flusso completo di una transazione di pagamento richiesta da un utente alla fine di una sosta. Il processo inizia con l'utente che invia una richiesta di pagamento tramite l'interfaccia web. Il sistema verifica la sosta associata al veicolo e calcola sia l'importo delle eventuali ricariche effettuate sia quello della sosta. Se tutti i dati sono corretti, il pagamento viene elaborato e inserito nel database. A pagamento completato, viene inviata una notifica di esito positivo, autorizzando l'utente a lasciare il parcheggio.

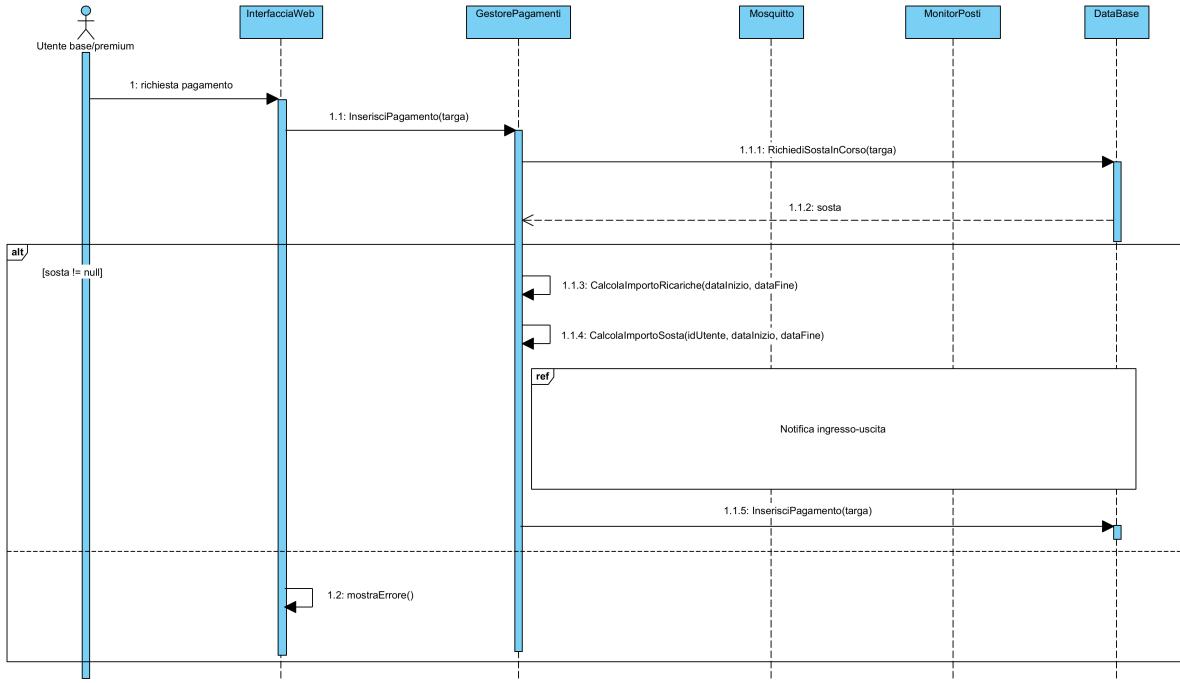


Figura 15: Diagramma di sequenza - Esegui pagamento

6.3 Notifica ingresso/uscita

Questo diagramma illustra il processo di notifica al sistema durante l'entrata o l'uscita di un veicolo dal parcheggio. Quando un veicolo viene rilevato da un sensore, il sistema invia un messaggio tramite il broker MQTT, che comunica con vari componenti. Il sistema aggiorna lo stato del posto auto nel database e gestisce la visualizzazione sull'emulatore Philips Hue, dove la lampadina corrispondente cambia colore (ad esempio, rosso per indicare che il posto è occupato). Inoltre, nel caso di uscita, viene verificato se l'utente ha effettuato il pagamento per la sosta.

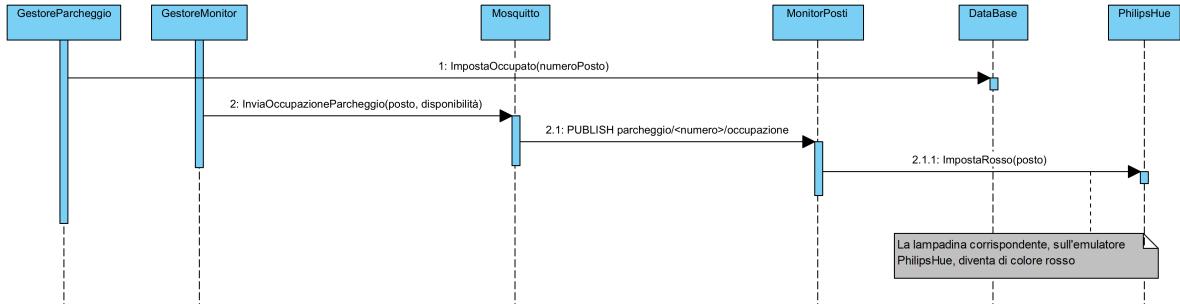


Figura 16: Diagramma di sequenza - Notifica ingresso/uscita

6.4 Prenota posto

Il diagramma raffigura la prenotazione di un posto auto, e mostra come un utente premium possa effettuare una prenotazione prima di arrivare al parcheggio. L'utente invia una richiesta di prenotazione al sistema, che verifica se il numero di prenotazioni è inferiore alla capacità massima dei posti disponibili. Se la prenotazione è possibile, il sistema la conferma, assegnando un posto specifico, e memorizza i dati nel database. L'utente riceve una conferma che include l'identificativo del posto riservato. In caso di errore, il sistema comunica l'impossibilità di effettuare la prenotazione.

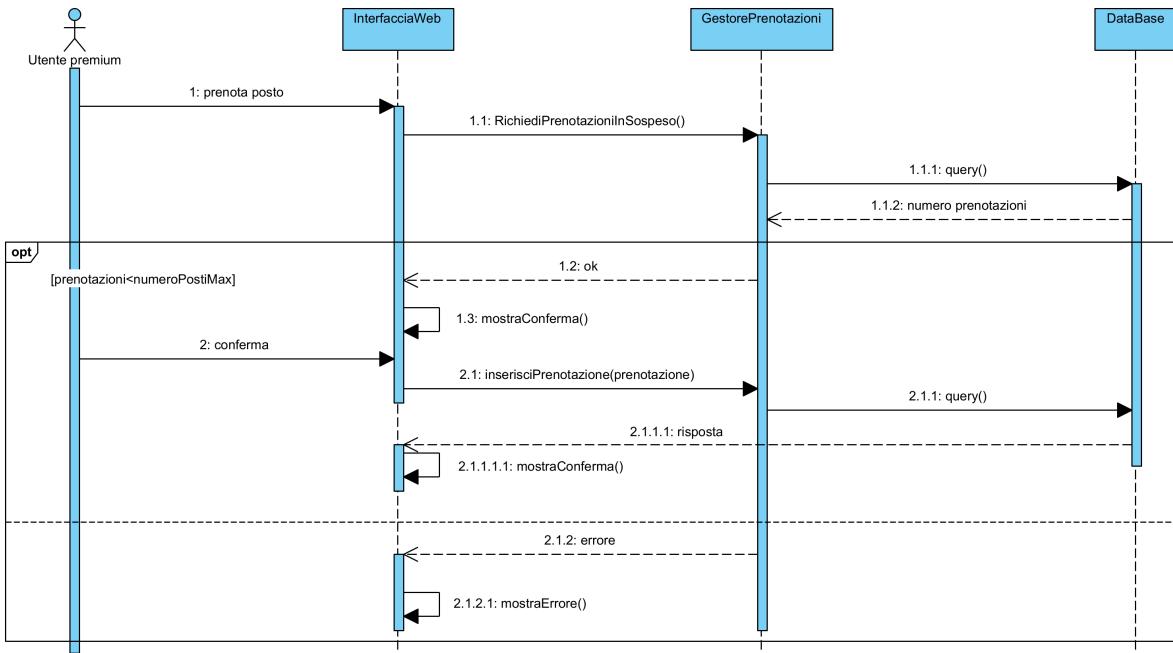


Figura 17: Diagramma di sequenza - Prenota posto

6.5 Richiedi ricarica

Il seguente diagramma di sequenza mostra il flusso di operazioni che coinvolgono l'MWbot quando un veicolo è pronto per la ricarica. Il sistema controlla se il veicolo è attualmente in sosta e, se la condizione è soddisfatta, inserisce una richiesta di ricarica. Quando il veicolo è in coda, viene inviata una richiesta all'MWbot tramite il broker MQTT per avviare la ricarica. L'MWbot esegue la ricarica e, una volta completata, pubblica un messaggio di completamento al sistema. Il database viene aggiornato con i dettagli della ricarica, e l'utente viene notificato della ricarica completata.

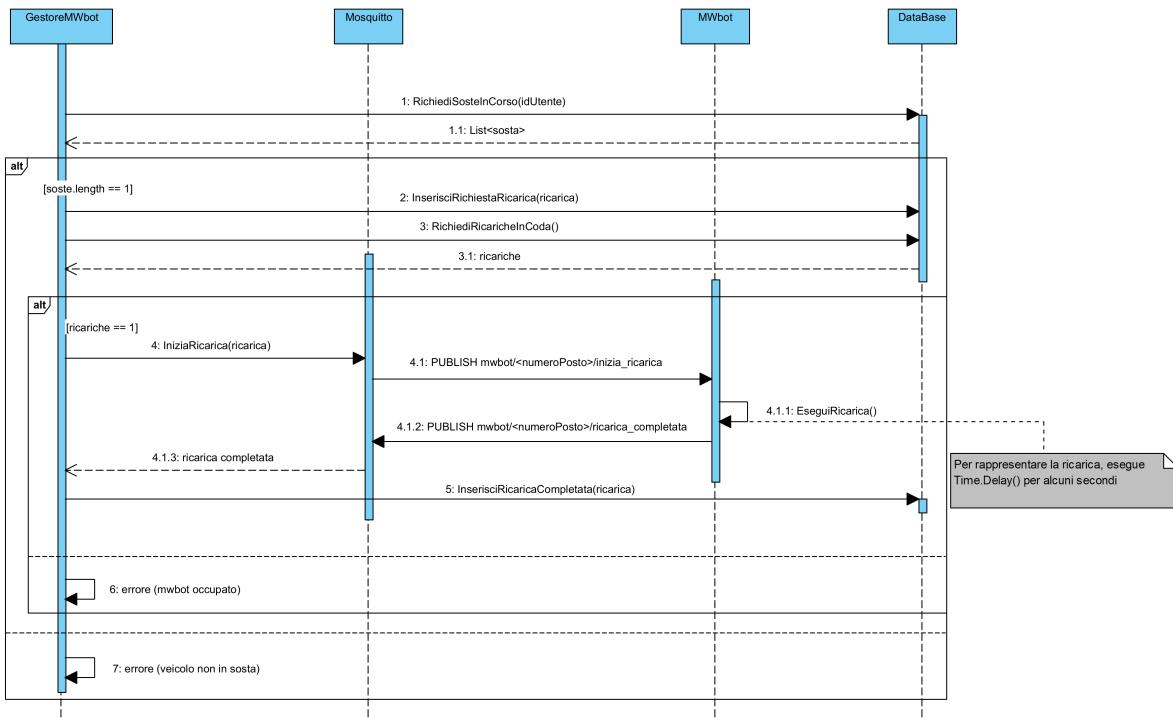


Figura 18: Diagramma di sequenza - Richiedi ricarica

6.6 Richiedi sosta

Il diagramma mostra il flusso di operazioni che avviene quando un utente richiede di sostare all'interno del parcheggio. Il sistema gestisce due scenari principali: l'utente senza prenotazione e l'utente con prenotazione. Nel primo caso, il sistema verifica che l'utente non abbia già una sosta attiva, richiede la targa del veicolo e, se tutto è corretto, inserisce i dettagli della sosta nel database. Nel secondo caso, il sistema verifica che l'utente sia arrivato entro i limiti della prenotazione. Se l'utente è in anticipo o in ritardo vengono generate delle penalità, e la sosta può essere rifiutata.

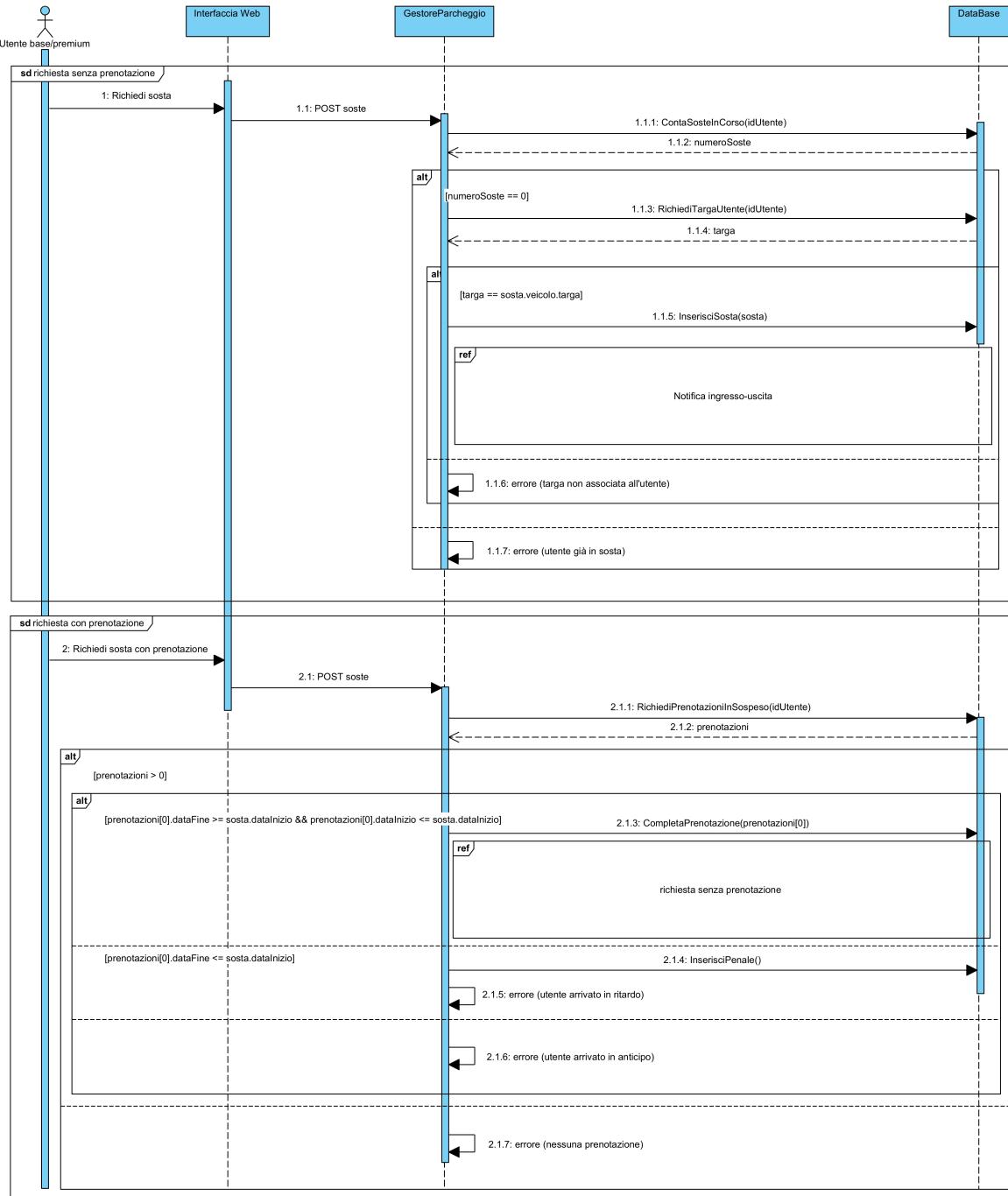


Figura 19: Diagramma di sequenza - Richiedi sosta

7 Istruzioni e suddivisione dei compiti

7.1 Come avviare il progetto

Prima di poter eseguire l'applicazione SmartPark, è necessario attivare i seguenti servizi:

- il broker **Mosquitto**.
- l'**Emulatore** per le lampadine Philips Hue, eseguendo il file `HueEmulator-v0.8.jar` ed importando la configurazione presente in `hue_smartpark.json` (entrambi contenuti nel folder `EmulatorePhilipsHue`). L'emulatore deve essere attivato sulla porta 8000.

Per avviare SmartPark è necessario eseguire i seguenti progetti della soluzione .NET *ProgettoPissir*:

- **SmartParkWebAPI**, che avvia l'API REST.
- **SmartParkWebApp**, che avvia l'applicazione web che utilizza l'API REST.

I progetti possono essere attivati contemporaneamente da Visual Studio (configurando "Progetti di avvio multipli" nelle proprietà della soluzione), oppure lanciando il comando `dotnet run` rispettivamente per SmartParkWebAPI e SmartParkWebApp.

7.2 Esempi di utilizzo

7.2.1 Credenziali utenti

Di seguito le credenziali di alcuni utenti già inseriti nel DataBase:

Email	Password	Targa Veicolo	Ruolo
marco@email.com	Password\$123	-	Amministratore
giulia@email.com	Password\$123	AA111AA	Utente Premium
luca@email.com	Password\$123	BB222BB	Utente

7.2.2 Richiedere una sosta

Se l'utente ha effettuato il login, può richiedere di iniziare una sosta (si assume che si trovi all'ingresso del parcheggio). Il sistema, dopo aver verificato la disponibilità di posti, chiederà di inserire la targa del veicolo con il quale effettuare la sosta. Se l'operazione ha avuto successo, la lampadina dell'emulatore Philips Hue corrispondente al posto occupato si colorerà di rosso, simulando così la visualizzazione del monitor dei posti.

7.2.3 Richiedere una ricarica

Se l'utente ha effettuato il login e ha un veicolo in sosta, può richiedere che l'MWbot ricarichi la batteria del veicolo fino a una certa percentuale (occorre specificare anche la percentuale attuale). Se l'MWbot non è occupato eseguirà la ricarica, altrimenti essa finirà in coda dopo quelle già richieste. Al completamento della ricarica l'utente riceverà una notifica.

7.2.4 Effettuare un pagamento e terminare la sosta

Quando l'utente vuole terminare la sosta deve richiedere un pagamento, che comprende l'importo della sosta e quello delle ricariche eseguite in questo intervallo. Si assume che il pagamento coincida con l'uscita del veicolo dal parcheggio, perciò dopo l'operazione la lampadina corrispondente al posto appena liberato si colorerà di verde.

7.2.5 Prenotare una sosta

Un utente Premium può prenotare una sosta prima di recarsi al parcheggio, specificandone l'intervallo di tempo. Se richiederà la sosta in anticipo non gli verrà permessa (ma potrà comunque effettuarla nel tempo previsto); se la richiederà in ritardo gli verrà addebitata una penale.

7.2.6 Operazioni dell'amministratore

L'amministratore può monitorare l'occupazione del parcheggio, vedere dove si trova l'MWbot, consultare lo storico di tutti gli utenti e aggiornare il prezzo di sosta e di ricarica per il parcheggio.

7.3 Suddivisione dei compiti

I microservizi del progetto sono stati così suddivisi tra i componenti del gruppo:

- Gestore backend: Cartieri, Fatone
- Sensori IoT: Broglio, Cartieri, Fatone
- API REST: Broglio, Cartieri
- Test: Broglio
- Web App: Broglio, Cartieri, Fatone
- Interfaccia grafica: Broglio
- Autenticazione: Cartieri, Fatone