SPECIFICA

Requisiti funzionali

- Login: gli utenti saranno in grado di fare il login inserendo il proprio username e la password
- Crea account: gli utenti potranno creare l'account fornendo nome, cognome, e-mail, username e password.
- Monitoraggio parcheggio: gli amministratori saranno in grado di vedere la situazione dei posti del parcheggio in tempo reale. In altre parole, saranno in grado di vedere quali posti sono occupati e quali no.
- Aggiornamento dei costi: gli amministratori saranno in grado di modificare il prezzo della ricarica e quello della sosta.
- Visualizzazione dei pagamenti: gli amministratori potranno vedere l'elenco contenente tutte le informazioni di ogni singolo pagamento. In aggiunta, potranno selezionare se visualizzare i pagamenti di un giorno in particolare, oppure se visualizzare i pagamenti riguardanti la sosta/ricarica e/o riguardo gli utenti premium/base.
- Richiesta di ricarica: gli utenti base e premium potranno richiedere di far ricaricare la propria auto.
- Esecuzione della ricarica: il MWbot ricarica l'auto.
- Esecuzione di una prenotazione: gli utenti premium potranno prenotare un posto per una certa data e ora.
- Generazione notifiche: Il MWbot, una volta terminata la ricarica, manderà una notifica visualizzabile dall'utente se quest'ultimo l'ha richiesto.
- Visualizzazione delle richieste di ricarica: Il MWbot dovrà essere in grado di visualizzare la coda delle richieste di ricarica ed eseguire una politica di tipo FIFO (First In First Out).

Requisiti non funzionali

- Utilizzo di Spark Java per l'interfaccia e la REST API
- Utilizzo di Java per la creazione del client MQTT
- L'interfaccia e il client MQTT comunicheranno con il backend tramite il framework Spring

PROGETTAZIONE

CLIENT

Non è stato specificato nei diagrammi, ma ogni Route del tipo “serve<nomePagina>Page” ha all’interno una variabile del tipo Map model. Essa permette di creare l’interfaccia grafica.

REST API

Ogni classe della REST API comunica con la classe DAO associata. Ad esempio, la classe User ha associata una classe UserDao. All’interno del DAO sono presenti operazioni del tipo “getUser(String username)” per ottenere le informazioni di un utente in particolare, “addUser(User newUser)” per aggiungere al DB un nuovo utente e così via.

REST API – EndPoint

Azione	URL	Metodo	Informazioni
Ottieni utente	http://localhost:4567/api/v1.0/users?username=?	GET	
Aggiungi utente	http://localhost:4567/api/v1.0/users	POST	Bisogna inserire nel body (in formato JSON): name, surname, email, username, password, type
Ottieni tutte le ricariche	http://localhost:4567/api/v1.0/recharges	GET	
Ottieni le ricariche di un utente	http://localhost:4567/api/v1.0/recharges?username=?	GET	
Ottieni le ricariche	http://localhost:4567/api/v1.0/recharges?username=?&completed=yes	GET	

completate di un utente			
Ottieni le ricariche non completate	http://localhost:4567/api/v1.0/recharges?completed =no	GET	
Modifica una richiesta di ricarica	http://localhost:4567/api/v1.0/recharges?username=?	PUT	Inserire nel body (in formato JSON) tutte le proprietà della classe Recharge
Aggiungi una richiesta di ricarica	http://localhost:4567/api/v1.0/recharges	POST	Inserire nel body (in formato JSON) tutte le proprietà della classe Recharge
Ottieni informazioni auto	http://localhost:4567/api/v1.0/cars/:licensePlate	GET	
Ottieni l'auto di un utente	http://localhost:4567/api/v1.0/users/:username/car	GET	
Aggiungi una macchina al DB	http://localhost:4567/api/v1.0/cars	POST	Inserire nel body (in formato JSON) tutte le proprietà della classe Car
Ottieni tutti i posti auto	http://localhost:4567/api/v1.0/parkingSpots	GET	È possibile aggiungere i filtri "limit" e "offset" per

			non ottenere la lista completa
Modifica stato di un posto auto	http://localhost:4567/api/v1.0/parkingSpots?id=?&value=?	PUT	Value può essere 1 (occupato) o 0 (libero)
Ottieni tutti i prezzi	http://localhost:4567/api/v1.0/prices	GET	Se si inserisce il filtro "service" si ottiene il prezzo di un servizio in particolare
Ottieni tutte le prenotazioni	http://localhost:4567/api/v1.0/reservations	GET	
Aggiungi prenotazione	http://localhost:4567/api/v1.0/reservations	POST	Inserire nel body (in formato JSON) tutte le proprietà della classe Reservation
Ottieni pagamenti	http://localhost:4567/api/v1.0/payments	GET	È possibile aggiungere i filtri "limit" e "offset" per non ottenere la lista completa
Aggiungi pagamento	http://localhost:4567/api/v1.0/payments	POST	Inserire nel body (in formato

			JSON) tutte le proprietà della classe Payment
Elimina ricarica di un utente	http://localhost:4567/api/v1.0/users/:username/recharges	DELETE	
Elimina prenotazione di un utente	http://localhost:4567/api/v1.0/users/:username/reservations	DELETE	
Aggiungi notifica	http://localhost:4567/api/v1.0/notifications	POST	Aggiungere nel body (in formato JSON) tutte le proprietà della classe Notification
Ottieni notifiche di un utente	http://localhost:4567/api/v1.0/notifications?username=?	GET	È possibile aggiungere i filtri “limit” e “offset” per non ottenere la lista completa

MQTTX

I messaggi MQTT vengono creati manualmente tramite il client MQTTX. I formati dei messaggi sono stati descritti all'interno del diagramma.