Progetto d’esame: laboratorio di PISSIR

Anno accademico 2023/2024

DESCRIZIONE PROGETTO

Sistema di gestione di parcheggio con ricaricatori wireless mobili per auto elettriche: requisiti funzionali

Immaginiamo che esistano ricaricatori wireless mobili (MWbot)[^1] 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 dell'auto raggiunge la percentuale di carica richiesta dall'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 all'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.

Gli automobilisti tramite un'applicazione multi-piattaforma (smartphone o quadro strumenti digitale dell'auto) possono effettuare le seguenti operazioni:

- solo gli utenti premium, possono consultare la disponibilità di posti auto nel parcheggio ed eventualmente, se disponibile, prenotarne preventivamente uno fornendo le proprie credenziali e i dati della carta di credito, e indicando un determinato tempo di arrivo e durata permanenza previsti. A seguito di queste operazioni riceveranno l'identificativo del posto prenotato. Nel caso la prenotazione non venga cancellata e l'utente non arrivi entro il tempo di arrivo più un certo tempo di tolleranza (es. 15-30 minuti), la prenotazione viene automaticamente cancellata ed è addebitata una determinata somma per la mancata utilizzazione del posto. Per semplicità supponiamo che tutti gli utenti rispettino la durata dichiarata (con un margine simile a quello del tempo di arrivo) in modo da poter accettare più prenotazioni non sovrapposte temporalmente per uno stesso posto.
- Gli utenti base o premium, possono richiedere che una volta arrivati la loro auto venga ricaricata fino ad una desiderata percentuale di carica; in tal caso verranno informati del numero di auto da ricaricare prima della loro, o di una stima del tempo di attesa prima del completamento della ricarica e a quel punto potranno accettare o rifiutare la prestazione
- un utente che abbia richiesto ed accettato il servizio di ricarica, può richiedere di ricevere una notifica (nella realtà questa potrebbe essere inviata tramite messaggio whatsapp, SMS, mail, nel vostro sistema semplicemente inviando un messaggio tramite MQTT) quando la batteria raggiunge il livello di carica desiderato.

[^1]: prototipi simili esistono già: https://insideevs.it/news/592196/ziggy-colonnina-mobile-robot-parcheggi/ https://www.newsauto.it/guide/ricarica-wireless-auto-elettriche-2-2022-382355/

All'uscita dal parcheggio, all'automobilista è addebitato il costo per il tempo di sosta e il costo per l'eventuale tempo totale di ricarica.

L'utente amministratore può:

- monitorare lo stato di occupazione di ogni posto auto
- aggiornare il costo €/ora della sosta e €/Kw della ricarica
- visualizzare i pagamenti effettuati in un determinato intervallo di giorni/ore, eventualmente distinti tra quelli per la sosta e quelli per la ricarica, e tra le diverse tipologie di utenti base e premium

Il sistema deve:

- autenticare gli automobilisti tramite login e password o con Oauth2 ed eventualmente, per chi l'ha richiesto, indicare il posto prenotato
- tener traccia del numero di auto all'interno del parcheggio ed aggiornare l'occupazione corrente su un monitor all'ingresso
- gestire le prenotazioni degli automobilisti, la ricarica, le eventuali notifiche ed il pagamento dei servizi erogati
- gestire la coda di auto in attesa di ricarica: non è necessario gestire il movimento dell'MWbot all'interno del parcheggio, ma bisogna indicare all'MWbot quale sarà il successivo utente/posto/auto (a seconda di come si decide di identificarli) da ricaricare

Struttura del sistema (progettazione)

Il sistema dovrà essere composto da

- un “backend” che espone un'interfaccia di tipo REST e permette di inserire in un database i dati rilevanti sulle prenotazioni, soste, ricariche e pagamenti, ed inoltre permette di modificare o consultare tali dati. Il backend ha accesso esclusivo al DB (chiunque voglia aggiornare o leggere i dati deve chiederlo al backend).
- Una interfaccia utente (un'app, una web-app o una mobile app) che permetterà agli utenti di interagire con il backend (tramite le API-REST di quest'ultimo). Questa potrà essere una semplice interfaccia testuale oppure una interfaccia a finestre (per esempio eseguibile nel browser, quindi una web-app) o una mobile app (se avete seguito il corso di applicazioni mobili).
- Un gestore del sottosistema IoT: si tratta di un componente che può entrare in comunicazione con i sensori di occupazione posto, il monitor all'ingresso del parcheggio e l'MWbot. Il gestore dei sottosistema IoT dovrà interagire con gli altri sottosistemi tramite broker MQTT (per esempio per trasmettere le notifiche sull'occupazione/liberazione del posto auto, sullo stato dell'MWbot (libero, in movimento, in erogazione), o altro.

COMPONENTI ACCESSORI (facoltativi)

Il sistema potrebbe inglobare anche alcuni componenti accessori: seguono alcuni esempi.

- Un sistema esterno di autenticazione / autorizzazione (es. keycloak) da utilizzare per permettere l'accesso ai soli automobilisti autorizzati (previa autenticazione);

- un sistema esterno di esecuzione delle transazioni per il pagamento dei servizi usufruiti dall'utente;
- supponendo l'uso di molteplici MWbot all'interno di un singolo parcheggio e che gli stessi necessitino periodicamente di essere ricaricati, un sistema di ottimizzazione che permetta di ricaricare i MWbot garantendo, per quanto possibile, la fruizione continua del servizio di ricarica auto.
- Alla ricezione della notifica di raggiungimento del livello di carica desiderata, gli utenti possono eventualmente richiedere un'ulteriore ricarica fino ad un altro livello. In tal caso la nuova richiesta verrà inserita in fondo all'eventuale coda di richieste di ricarica e nuovamente l'utente sarà informato della sua lunghezza, o del tempo complessivo d'attesa stimato per il completamento

SENSORI E ATTUATORI

i sensori e gli attuatori potranno essere reali o emulati, in base alla disponibilità di strumenti fisici. In ogni caso per poter effettuare più facilmente i test sul sottosistema IoT e i test d'integrazione occorrerà predisporre delle piccole applicazioni in grado di fornire misure fittizie (che siano rappresentative di scenari realistici, e che potrebbero essere memorizzate e quindi lette da un file: ciò sarebbe molto utile sia per lo svolgimento dei test che in occasione della demo da mostrare all'esame) ma anche di recepire i comandi per gli attuatori e visualizzarli in formato testo o grafico (una possibilità è usare l'emulatore delle Philips Hue assegnando a ciascuna lampadina emulata un significato, come rappresentante di un attuatore fisico, e in base al comando ricevuto accenderla, spegnerla o farle cambiare colore).

FASI DEL LAVORO

Specifica

in base ai requisiti elencati sopra, eventualmente dettagliati grazie a interviste che i gruppi potranno organizzare con il committente (i docenti del corso), occorrerà definire con sufficiente dettaglio il dominio e i casi d'uso. Per evitare ambiguità si dovranno usare schemi chiari preferibilmente utilizzando i diagrammi UML (diagramma delle classi del dominio e diagramma dei casi d'uso, corredati di brevi commenti testuali complementari ai diagrammi che spiegano in cosa consiste il caso d'uso; a vostra discrezione potrete usare altri tipi di diagrammi per specificare meglio l'articolazione in termini di successione di passi dei diversi casi d'uso).

Progettazione

in questa fase si dovrà scegliere il tipo di architettura (che dovrà essere basata su microservizi), si dovranno individuare i sottosistemi, le loro interfacce e le interazioni. Inoltre all'interno di ogni gruppo si dovranno definire dei sottogruppi a cui attribuire la responsabilità della progettazione di 1 o 2 sottosistemi (AI GRUPPI PIU' NUMEROSI SARA' RICHIESTO DI REALIZZARE UNA DELLE FUNZIONALITA' ACCESSORIE). Per ciascun sottosistema dovrà essere preparato un documento che definisca in modo sufficientemente preciso le classi che si dovranno implementare nella fase successiva: per evitare ambiguità la documentazione dovrà contenere schemi chiari, possibilmente utilizzando diagrammi UML. In particolare dovrete schematizzare il diagramma delle classi (questa volta sarà il diagramma delle classi che dovranno essere implementate) e dei package, e i diagrammi di sequenza per descrivere le interazioni tra i componenti (sia all'interno di uno stesso microservizio, sia tra microservizi diversi). In questa fase si dovranno definire sia gli end-point delle API REST, con gli eventuali parametri

e il formato delle informazioni scambiate, sia i topic MQTT con il formato dei messaggi inviati sui diversi topic

Implementazione

L'implementazione delle classi definite in fase di progettazione può essere essere fatta in Java, ma potete usare altri linguaggi a voi più congeniali (il supporto da parte dei docenti potrebbe non essere ugualmente proficuo se si scelgono altri linguaggi). I diversi microservizi possono essere implementati con linguaggi diversi (ciascun sottogruppo può implementare nel linguaggio preferito il proprio microservizio: l'importante è che le interfacce siano rispettate).

Test

Lo sviluppo dovrà prevedere anche il test dei componenti e dei sottosistemi in isolamento. Quando il gruppo avrà completato tutti i sottosistemi dovrà svolgere anche test di integrazione. I test dovranno coprire un numero adeguato di casistiche tipiche, e saranno anche la base per la dimostrazione di funzionamento che dovrà essere illustrata durante la parte dell'esame relativa al progetto.

Demo

durante l'esame sulla parte pratica dovrà essere preparata una demo che permetta di mostrare tutte le funzionalità implementate e che sia esempio significativo di alcuni dei test più rilevanti realizzati prima della consegna.

Progetto d'esame: laboratorio di PISSIR

Progettazione concettuale

Sommario

Componenti del gruppo

Relatrice: Franceschinis Giuliana, giuliana.franceschinis@uniupo.it

1. Palmieri Matteo, 20038775, 20038775@studenti.uniupo.it
2. Galasso Veronica, 20038821, 20038821@studenti.uniupo.it

1.1 Lettura requisiti

> Immaginiamo che esistano ricaricatori wireless mobili (MWbot) capaci autonomamente di spostarsi sotto le auto elettriche e ricaricarle per induzione.

- Capiamo che esiste questa entità che compie determinate azioni da sé, quindi non dovremmo implementare da zero nella fase di progettazione, immagino che ci abbiano già pensate gli ingegneri elettronici che hanno creato appunto questo **MWbot**

> 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.

- Dovremmo implementare il servizio di ricarica alle macchine elettriche chiedendo al proprietario la **percentuale di batteria** da raggiungere

> Quando la batteria dell'auto raggiunge la percentuale di carica richiesta dall'utente, l'MWbot può spostarsi per caricare altre auto.

- **DOMANDA**: é possibile interrompere la ricarica prima di raggiungere alla percentuale desiderata?
  - **F.G.**: Qualche gruppo ha implementato questa possibilità ma potete decidere di NON implementarla (dovete scriverlo con precisione nella documentazione)
- alla luce della risposta della prof.ssa, supponiamo che l'utente che richiede il servizio di ricarica rispetti la percentuale richiesta senza interromperlo prima

> Si suppone che l'MWbot possa riconoscere il modello d'auto da ricaricare e in particolare la capacità in KW della batteria.

- Ciò lascia a noi programmatori a gestire non il "come" ma "quanto" caricare. Da questa frase mi fa capire che avremo due paramentri importanti dalla macchina da ricaricare, ovvero i **KW attuali** della batteria e i **KW massimi** della batteria, così da calcolare la percentuale di carica

> Inoltre ogni posto auto è dotato di sensori per monitorarne l'occupazione.

- Ci saranno tanti **sensori di parcheggio** quanti sono i posti auto capaci di mandare un'informazioni simil-booleana: **"posto libero"** e **"posto occupato"**

> 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 all'interno del parcheggio.

- **DOMANDA**: Dovremmo progettare in modo che é possibile ampliare facilmente da un singolo parcheggio con un MWbot? Nel senso aggiungere un altro parcheggio con un altro MWbot o mettere più MWbot in un parcheggio
    - **F.G.**: Anche questa è una caratteristica che sarebbe utile ma, soprattutto se siete solo 2 nel gruppo, potete decidere di NON implementare. Anche in questo caso precisare questi dettagli nella documentazione
- Visto il codice JS dell'interfaccia GUI per gli utenti di sistema di Veronica, che implementa di già questa complessità, abbiamo deciso di togliere questa caratteristica dallo scenario di molteplici parcheggi con almeno un MWbot e di rimanere con uno solo parcheggio con un MWbot.

> 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.

- Esistono tre utenti: **base**, **premium** e **amministratore**.
    - l'utente base può usufruire dei servizi di parcheggio e di ricarica
    - l'utente premium può usufruire, oltre a quelli dell'utente base, a prenotare il posto auto da remoto
    - l'amministratore è l'unico tipo di utente che è "unicum", ovvero esiste un solo utente amministratore nell'intero sistema di gestione del parcheggio
    - l'amministratore non usufruisce dei servizi degli utenti base e premium, ma può monitorare i vari componenti del sistema (da definire) e lo stato dei sensori di parcheggio

> Gli automobilisti tramite un'applicazione multi-piattaforma (smartphone o quadro strumenti digitale dell'auto) possono effettuare le seguenti operazioni:

- **DOMANDA**: per **applicazione multi-piattaforma**, si intende una interfaccia (CLI o GUI) dedicato ai soli utenti base e premium? Puó andare bene un website Node.js based?
    - **F.G.**: Sì va bene.
- nell'architettura del progetto, avremo un server Node.js con il middleware Express.js che si occuperà soltanto di fornire un'interfaccia grafica intuitiva agli utenti di sistema e gestire i dati in input da poi inoltrare al backend

> - solo gli utenti premium, possono consultare la disponibilità di posti auto nel parcheggio ed eventualmente, se disponibile, prenotarne preventivamente uno fornendo le proprie credenziali e i dati della carta di credito, e indicando un determinato tempo di arrivo e durata permanenza previsti. A seguito di queste operazioni riceveranno l'identificativo del posto prenotato. Nel caso la prenotazione non venga cancellata e l'utente non arrivi entro il tempo di arrivo più un certo tempo di tolleranza (es. 15-30 minuti), la prenotazione viene automaticamente cancellata ed è addebitata una determinata somma per la mancata utilizzazione del posto. Per semplicità supponiamo che tutti gli utenti rispettino la durata dichiarata (con un margine simile a quello del tempo di arrivo) in modo da poter accettare più prenotazioni non sovrapposte temporalmente per uno stesso posto.

- l'utente premium può consultare, tramite applicazione multi-piattaforma, i posti auto liberi e di prenotare da remoto il posto, per fare ciò bisogna essere autenticato e con una carta di credito valida.
- I posti auto sono identificati in modo univoco da un codice seriale (un semplice unsigned integer oppure una lettera alfabetica per indicare la corsia e un numero a due cifre, es. A01, C21, ...)
- dovremmo stabilire in modo chiaro il tempo di tolleranza riguardo al tempo di arrivo e alla durata di permanenza
- Il posto auto viene scelto dal backend (presumo casualmente tra quelli liberi) e viene assegnato all'utente
- potremmo impostare che se uno slot è occupato fino alle 12:30, un altro utente premium che vuole occupare quello slot, può farlo dalle 12:30 avvisando che può esserci una sovrapposizione di tot. tempo di tolleranza

> - Gli utenti base o premium, possono richiedere che una volta arrivati la loro auto venga ricaricata fino ad una desiderata percentuale di carica; in tal caso verranno informati del numero di auto da ricaricare prima della loro, o di una stima del tempo di attesa prima del completamento della ricarica e a quel punto potranno accettare o rifiutare la prestazione

- dovremmo chiedere prima all'utente se desidera il servizio di ricarica, poi successivamente fare il resoconto del tempo necessario a ricaricare la sua macchina fino alla percentuale di batteria desiderata
- Ciò mi porta a credere che anche l'utente base verrà assegnato un posto auto dal backend come utente premium per la prenotazione a distanza ma effettuato tramite il "totem" all'entrata del parcheggio
- Infine, per qualsiasi richiesta di sosta, è sempre il backend a scegliere il posto auto da assegnare all'utente richiedente del servizio di sosta

> - un utente che abbia richiesto ed accettato il servizio di ricarica, può richiedere di ricevere una notifica (nella realtà questa potrebbe essere inviata tramite messaggio whatsapp, SMS, mail, nel vostro sistema semplicemente inviando un messaggio tramite MQTT) quando la batteria raggiunge il livello di carica desiderato.

- dovremmo mettere alla fine della richiesta del servizio di ricarica se vuole ricevere il messaggio MQTT

> All'uscita dal parcheggio, all'automobilista è addebitato il costo per il tempo di sosta e il costo per l'eventuale tempo totale di ricarica.

- **DOMANDA**: per tempo di sosta dovremmo memorizzare il tempo di entrata e il tempo di uscita dal parcheggio o in base allo stato occupazionale del posto auto dal sensore di parcheggio?
    - **F.G**: Non sono sicura di aver capito bene la domanda. Potete memorizzare il tempo di entrata e poi all'uscita calcolare la differenza tempo di uscita - tempo di entrata per il calcolo del tempo di sosta (e il calcolo dell'addebito corrispondente)
- Il tempo di sosta riferito nell'addebito quando l'utente esce dal parcheggio è semplicemente la differenza dell'orario di uscita e quello di entrata nella stazione di parcheggio

> L'utente amministratore può:
>
> - monitorare lo stato di occupazione di ogni posto auto
> - aggiornare il costo €/ora della sosta e €/Kw della ricarica
> - visualizzare i pagamenti effettuati in un determinato intervallo di giorni/ore, eventualmente distinti tra quelli per la sosta e quelli per la ricarica, e tra le diverse tipologie di utenti base e premium

- Per ora, l'amministratore può compiere le seguenti azioni:
    - GET: stato di occupazione di ogni posto auto
    - GET/SET: costo €/ora della sosta
    - GET/SET: costo €/ora della ricarica
    - GET: pagamenti effettuati in un intervallo di giorni/ore, distinti tra quelli per la sosta e quelli per la ricarica, e tra utenti base e premium

> Il sistema deve:
>
> - autenticare gli automobilisti tramite login e password o con Oauth2 ed eventualmente, per chi l'ha richiesto, indicare il posto prenotato
> - tener traccia del numero di auto all'interno del parcheggio ed aggiornare l'occupazione corrente su un monitor all'ingresso
> - gestire le prenotazioni degli automobilisti, la ricarica, le eventuali notifiche ed il pagamento dei servizi erogati
> - gestire la coda di auto in attesa di ricarica: non è necessario gestire il movimento dell'MWbot all'interno del parcheggio, ma bisogna indicare all'MWbot quale sarà il successivo utente/posto/auto (a seconda di come si decide di identificarli) da ricaricare

- Per ora, il sistema può compiere le seguenti azioni:
    - autenticare gli automobilisti tramite login classico con email e password
    - indicare il posto auto prenotato all'utente premium autenticato
    - tenere aggiornato l'occupazione corrente (numero di auto presenti al suo interno) su un monitor all'ingresso del parcheggio
    - gestire la prenotazione del posto auto richiesto da un utente premium autenticato
    - gestire il servizio di ricarica agli utenti base e premium autenticati
    - notificare gli automobilisti riguardo mancato arrivo al parcheggio
        - gestire il use-case riguardo alla multa riguardo mancato arrivo
    - notificare gli automobilisti riguardo la ricarica completata
    - gestire il pagamento relativo al tempo di sosta all'utente all'uscita del parcheggio
    - gestire il pagamento relativo al tempo di ricarica all'utente all'uscita del parcheggio

    - manovrare il MWbot impostando la prossima macchina da ricaricare in base all'ID del posto auto

> Il sistema dovrà essere composto da
>
> - un “backend” che espone un'interfaccia di tipo REST e permette di inserire in un database i dati rilevanti sulle prenotazioni, soste, ricariche e pagamenti, ed inoltre permette di modificare o consultare tali dati. Il backend ha accesso esclusivo al DB (chiunque voglia aggiornare o leggere i dati deve chiederlo al backend).
> - Una interfaccia utente (un'app, una web-app o una mobile app) che permetterà agli utenti di interagire con il backend (tramite le API-REST di quest'ultimo). Questa potrà essere una semplice interfaccia testuale oppure una interfaccia a finestre (per esempio eseguibile nel browser, quindi una web-app) o una mobile app (se avete seguito il corso di applicazioni mobili).
> - Un gestore del sottosistema IoT: si tratta di un componente che può entrare in comunicazione con i sensori di occupazione posto, il monitor all'ingresso del parcheggio e l'MWbot. Il gestore dei sottosistema IoT dovrà interagire con gli altri sottosistemi tramite broker MQTT (per esempio per trasmettere le notifiche sull'occupazione/liberazione del posto auto, sullo stato dell'MWbot (libero, in movimento, in erogazione), o altro).

- L'**architettura del progetto** è di tipo client-server riguardo alla comunicazione tra i clienti (utenti base e premium) e il server/backend con in aggiunta un server MQTT per gestire i numerosi sensori di parcheggio e il MWbot, in breve l'achitettura è composta da tre parti:
    - **BACKEND**: web server con interfaccia REST documentato, l'unico ad aver accesso in lettura/scrittura diretto al DB
    - **FRONTEND**: interfaccIA CLI o GUI per utenti base e premium, una interfaccia dedicata solo all'amministratore; queste due interfacce comunicano al BACKEND tramite richieste REST
    - **MQTT**: un sottosistema IoT con al centro il MQTT broker dove incanala tutte le informazioni ricavabili dai dispositivi collegati, ovvero i sensori di parcheggio, un MWbot e il monitor all'entrata del parcheggio

> Il sistema potrebbe inglobare anche alcuni componenti accessori: seguono alcuni esempi.
>
> - Un sistema esterno di autenticazione / autorizzazione (es. keycloak) da utilizzare per permettere l'accesso ai soli automobilisti autorizzati (previa autenticazione);
> - un sistema esterno di esecuzione delle transazioni per il pagamento dei servizi usufruiti dall'utente;
> - supponendo l'uso di molteplici MWbot all'interno di un singolo parcheggio e che gli stessi necessitino periodicamente di essere ricaricati, un sistema di ottimizzazione che permetta di ricaricare i MWbot garantendo, per quanto possibile, la fruizione continua del servizio di ricarica auto.
> - Alla ricezione della notifica di raggiungimento del livello di carica desiderata, gli utenti possono eventualmente richiedere un'ulteriore ricarica fino ad un altro livello. In tal caso la nuova richiesta verrà inserita in fondo all'eventuale coda di richieste di ricarica e nuovamente l'utente sarà informato della sua lunghezza, o del tempo complessivo d'attesa stimato per il completamento

- Essendo un gruppo composto da 2 persone, nessuno di queste componenti aggiuntive verranno implementate

> i sensori e gli attuatori potranno essere reali o emulati, in base alla disponibilità di strumenti fisici. In ogni caso per poter effettuare più facilmente i test sul sottosistema IoT e i test d'integrazione occorrerà predisporre delle piccole applicazioni in grado di fornire misure fittizie (che siano rappresentative di scenari realistici, e che potrebbero essere memorizzate e quindi lette da un file: ciò sarebbe molto utile sia per lo svolgimento dei test che in occasione della demo da mostrare all'esame) ma anche di recepire i comandi per gli attuatori e visualizzarli in formato testo o grafico (una possibilità è usare l'emulatore delle Philips Hue assegnando a ciascuna lampadina emulata un significato, come rappresentante di un attuatore fisico, e in base al comando ricevuto accenderla, spegnerla o farle cambiare colore).

- Per facilità, creeremo una simulazione dei sensori di parcheggio così ogni sensori ha il suo corso di vita durante l'esecuzione del progetto

1.2 Requisiti riscritti

1.2.1 Sistema di gestione parcheggio con ricaricatori wireless mobili

L'obiettivo è progettare un sistema per gestire in modo intelligente una stazione di parcheggio con un MWbot che ricarica le auto elettriche.

La stazione di parcheggio, per brevità "parcheggio", è composto da numerosi posti auto "slots" dove i veicoli possono sostare e sono identificati univocamente.

Ogni slot del parcheggio è dotato di un suo "sensore" (di occupazione) che segnala al sistema tramite MQTT lo stato occupazionale per intendere la presenza/assenza di un veicolo sullo slot corrispondente.

Il sistema prevede l'uso di un ricaricatore wireless mobile (MWbot) che si sposta autonomamente sotto le auto elettriche in sosta su un posto auto per ricaricarle per induzione. L'MWbot è capace di riconoscere il modello d'auto da ricaricare e la capacità in KW della batteria. Quando la batteria dell'auto raggiunge la percentuale di carica richiesta dall'utente, l'MWbot notifica il completamento e può spostarsi per caricare altre auto elettriche.

Gli utenti di sistema sono di tre tipi:

- **Utenti base**: usufruiscono dei servizi di parcheggio e ricarica
    - richiedono la ricarica fino a una percentuale desiderata, viene informato del numero di auto da ricaricare prima della loro, se accettano, possono richiedere di ricevere una notifica al loro cellulare tramite MQTT al completamento della ricarica.
    - richiedono la sosta, viene assegnato un ID dello slot scelto dal backend.
    - può passare ad essere un utente premium, per poter richiedere la sosta da remoto
- **Utenti premium**: in aggiunta ai servizi dell'utente base, possono
    - prenotare a distanza una sosta da remoto se disponibile, forniscono
        - credenziali
        - dati di pagamento,

        - l'orario di arrivo
        - durata della sosta

      ricevono un ID del posto auto scelto dal backend.

      La prenotazione scade se l'utente non arriva entro il orario di arrivo stabilito più un margine di 10 min, in più viene stilato una multa

      $$ multa = €/h_\mathrm{sosta} * \frac{N^\circ minuti_\mathrm{durata}}{60} $$

    - può passare ad essere un utente base

- **Amministratore**: in aggiunta ai servizi dell'utente premium, possono

    - Monitorare lo stato di occupazione dei posti auto.
    - Aggiornare i costi €/ora della sosta e €/Kw della ricarica.
    - Visualizzare i pagamenti effettuati applicando dei filtri (anche combinati) per una migliore ricerca, ovvero:
        - intervallo di giorni/ore.
        - distinguere i pagamenti tra quelli per la sosta e quelli per la ricarica
        - pagato da un certo tipo di utente (base e/o premium).

Quando l'utente raggiunge l'uscita del parcheggio, il sistema addebita i costi per il tempo di sosta e per la ricarica se ne ha usufruito:

- **Addebito per il servizio di sosta**: differenza in minuti tra entrata e uscita dal parcheggio per il costo orario della sosta

  $$ €/h_\mathrm{sosta} * (timedate_\mathrm{uscita} - timedate_\mathrm{entrata}) $$

- **Addebito per il servizio di ricarica**: numero di Kw caricati per il costo al Kw della ricarica

  $$ €/Kw_\mathrm{ricarica} * Kw_\mathrm{erogati} $$

Il sistema deve:

- Autenticare gli utenti tramite login con email e password
- Indicare il posto auto scelto dal backend all'utente dopo l'accertata prenotazione o arrivo dell'utente al parcheggio.
- Aggiornare lo stato occupazionale dei posti auto corrente su un monitor all'ingresso del parcheggio.
- Gestire prenotazioni, pagamenti e ricariche con eventuali notifiche.
- Gestire la coda di ricarica, indicando all'MWbot il prossimo veicolo da caricare.

1.2.2 Struttura del sistema

Il sistema è composto da:

- **Backend**: server con interfaccia REST che gestisce prenotazioni, soste, ricariche e pagamenti interagendo con il database.
- **Interfaccia utente**: permette agli utenti di interagire con il backend tramite una interfaccia GUI.

- **Gestore del sottosistema IoT**: comunica con sensori di occupazione, il monitor, l'MWbot e backend tramite broker MQTT per notificare l'un l'altro...

1.2.3 Componenti accessori (facoltativi)

Visto l'esigua dimensione del gruppo di lavoro, i componenti accessori non verranno svolti.

1.2.4 Sensori e attuatori

I sensori e gli attuatori saranno emulati, per effettuare più facilmente i test sul sottosistema IoT e test d'integrazione.

Queste applicazioni dovranno anche ricevere comandi per gli attuatori e mostrarli in formato testo o grafico tramite una pagina web completamente distaccata dal resto del sistema.

Progetto d'esame: laboratorio di PISSIR

Progettazione logica

Sommario

Architettura del sistema

Il sistema è composto da più architetture che interagiscono tra loro:

1. **Architettura Client-Server:**
   - **Frontend:** è scritto in JavaScript e utilizza Node.js per l'esecuzione e con il framework Express.js mostriamo un'interfaccia grafica realizzata come pagina web. Inoltre Express funge da **client REST** per il backend. Gli utenti interagiscono con il sistema tramite il frontend, il quale effettua chiamate REST al server backend.
   - **Backend:** Il server backend è scritto in Java utilizzando il framework **Javalin**. Questo server espone delle API REST per ricevere richieste dal frontend e interagire con il database SQLite.
   - **Database:** Il database utilizzato è **SQLite**, che viene manipolato esclusivamente dal backend tramite le API REST.
2. **Architettura IoT (Internet of Things):**
   - È presente un sistema IoT, costituito da numerosi dispositivi che si collegano a un broker **MQTT Mosquitto**.
   - I dispositivi IoT, **sensor, monitor, mobile e mwbot**, sono tutti client MQTT conformi alla versione **MQTTv3** e utilizzano la libreria **Eclipse Paho** in Java per la gestione della comunicazione.
   - **Comunicazione IoT:** Tutti i dispositivi (client MQTT) inviano e ricevono dati dal broker Mosquitto. La comunicazione tra il backend e i dispositivi IoT avviene tramite MQTT per operazioni in tempo reale.

Schema dell'architettura:

- **Client-Server**: Frontend (Client REST) <-> API REST <-> Backend (Java Javalin, SQLite)
- **IoT**: Dispositivi IoT (Sensori, Monitor, MWBot, Mobile) <-> MQTT Broker (Mosquitto)

Diagramma UML

Dettagli dei Moduli

- **Frontend (Node.js, Express.js):** Gestisce le richieste degli utenti e le interazioni via interfaccia Web, comunicando con il backend tramite RESTful API.
- **Backend (Java, Javalin):** Espone API REST per il frontend e interagisce con il database SQLite. Si occupa delle operazioni di backend.
- **MQTT Broker (Mosquitto):** Centralizza la comunicazione tra i dispositivi IoT (sensori, monitor, mobile, mwbot).
- **Sensori, Monitor, MWBot, Mobile (Java, Eclipse Paho Client MQTT):** Dispositivi che inviano e ricevono messaggi al broker MQTT.

Moduli di Comunicazione

- **Frontend e Backend (REST API):** Il frontend comunica con il backend tramite chiamate RESTful API. Queste API sono implementate nel server backend (Java con Javalin) e consentono l'interscambio di dati tra l'interfaccia utente e la logica di business del server. Il backend interagisce con il database SQLite per ottenere o salvare i dati.

- **IoT (MQTT):** Il sistema IoT è gestito tramite un broker MQTT (Mosquitto). I dispositivi IoT (sensori, mobile, monitor, mwbot) sono client MQTT, utilizzando la libreria **Eclipse Paho** per inviare e ricevere messaggi al broker. Il backend può interagire con questi dispositivi IoT in tempo reale, ricevendo dati da sensori o inviando comandi tramite il broker MQTT.

Tecnologie e Strumenti utilizati

- **Javalin:** Javalin è un framework per Java che ti consente di creare API REST veloci ed efficienti, successore di SparkJava Core, utilizzato da Backend e Debug Gradle applications.
- **Node.js e Express.js:** Il server frontend è scritto in Node.js con il middleware Express.js, che gestisce le richieste HTTP da parte degli utenti tramite l'interfaccia Web, le elabora per inoltrarle al server backend tramite le chiamate REST.
- **SQLite:** è il DBMS utilizzato per la sua semplicità e la capacità di essere facilmente integrato in applicazioni leggere. Tutte le operazioni sul database vengono eseguite dal backend tramite API REST.
- **MQTT:** I dispositivi IoT e il backend utilizzano questo protocollo per la comunicazione in tempo reale con il broker **Mosquitto**. **Eclipse Paho** è la libreria Java scelta per gestire la connessione MQTT dei vari client.
- **Autenticazione:** Gli utenti si autenticano con **email e password** per un semplice accesso al sistema.
- **Deployment:** Il sistema è composto da diverse applicazioni Java, tutte gestite tramite **Gradle**. Per semplificare il lancio del sistema, ho creato uno **script bash** che permette di avviare tutte le applicazioni (i vari **JAR files**) separatamente, simulando l'ambiente di produzione.

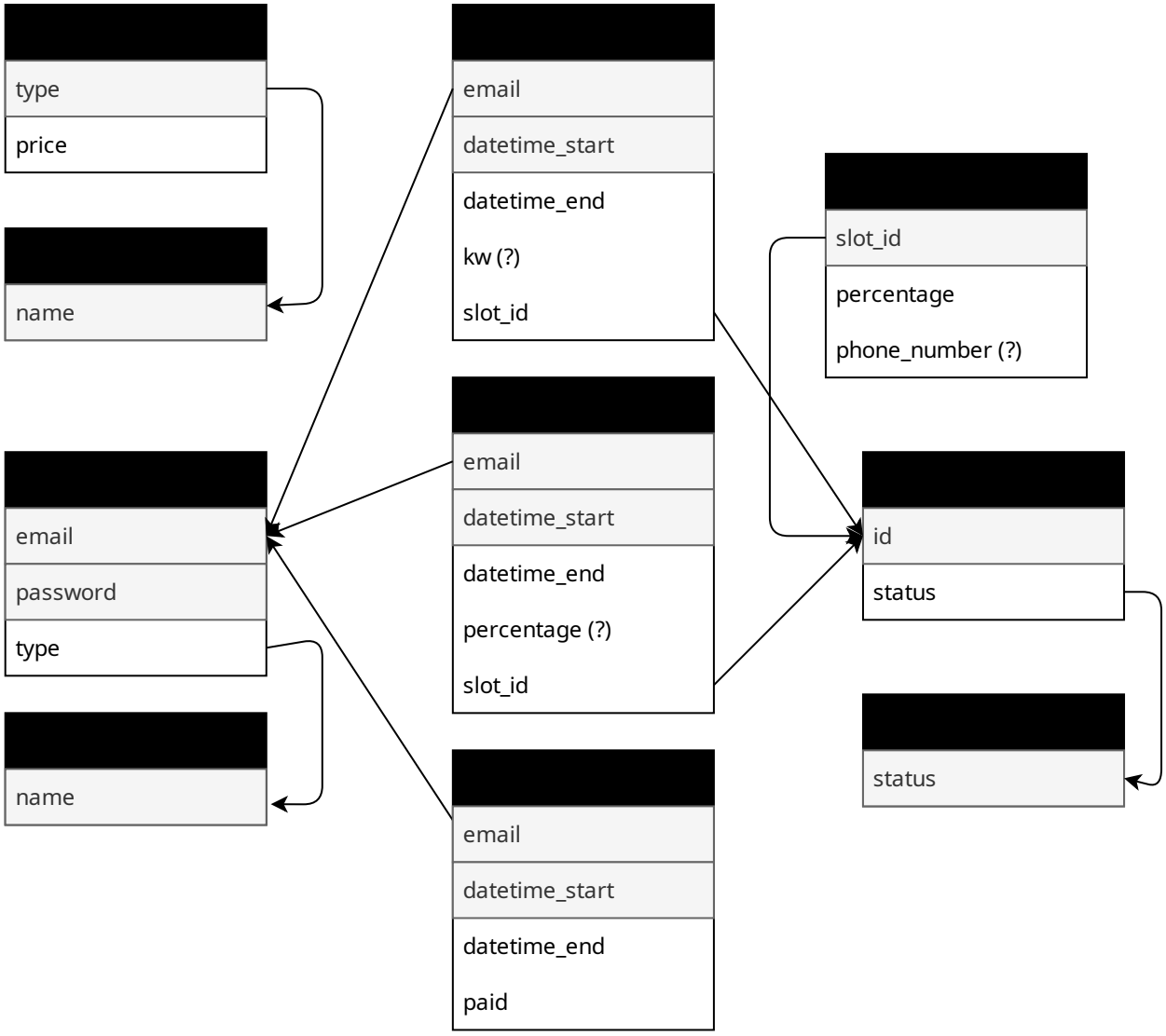
Sicurezza e Affidabilità

Ho implementato un'app Gradle atto a monitorare per tenere traccia dello stato dei vari client MQTT in tempo reale.

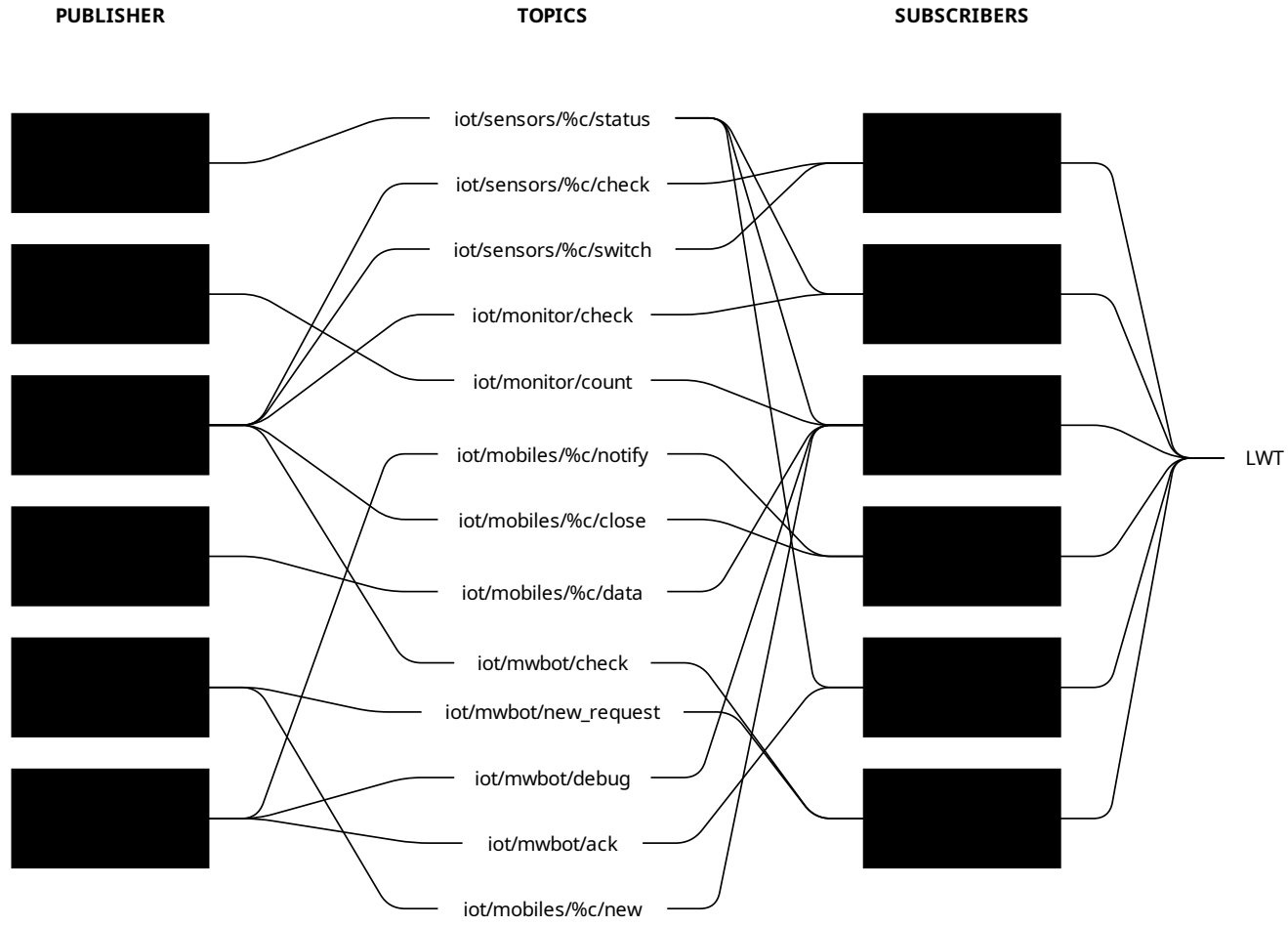
Sviluppo

DB

- **Schema DB:** smartparkingDB.schema


type
price
name
email
password
type
name
email
datetime_start
datetime_end
kw (?)
slot_id
email
datetime_start
datetime_end
percentage (?)
slot_id
email
datetime_start
datetime_end
paid
slot_id
percentage
phone_number (?)
id
status
status

Topic



Topic messages

- **iot/sensors/sensor_[1-9][0-9]*/status**: sensore pubblica il proprio stato occupazionale.

```
{
  "slot_id": "[1-9][0-9]+",
  "status": "FREE" | "OCCUPIED"
}
```

- **iot/sensors/sensor_[1-9][0-9]*/check**: topic utilizzato per "pingare" il sensore per avere il suo stato occupazionale.

```

```

- **iot/sensors/sensor_[1-9][0-9]*/switch**: topic utilizzato per "pingare" il sensore di cambiare stato occupazionale da `FREE` a `OCCUPIED` o viceversa.

```

```

- **iot/monitor/check**: topic utilizzato per "pingare" il monitor per avere il numero di posti liberi.

```

```

```
```

- **iot/monitor/count**: monitor pubblica il numero di posti liberi.

```
{
  "id": "monitor",
  "count": "[0-9]+"
}
```

- **iot/mobiles/[0-9]{10}/notify**: topic utilizzato per "notificare" il mobile/cellulare che l'MWbot ha finito di ricaricare la macchina.

```
{
  "message": "Car charged to ([0-9]|([1-9][0-9])|(100))% in slot [1-9][0-
9]* with 0|[1-9][0-9]* KW"
}
```

- **iot/mwbot/[0-9]{10}/close**: topic utilizzato per "chiudere" il processo mobile/cellulare.

```
```

- **iot/mwbot/[0-9]{10}/data**: topic utilizzato dai dispositivi mobile/cellulare per inviare dati al broker MQTT. (utile per il debug)

```
{
  "id": "[0-9]{10}",
  "message": ""
}
```

- **iot/mwbot/check**: topic utilizzato per "pingare" il MWBot per avere il suo stato lavorativo e altre informazioni.

```
```

- **iot/mwbot/new_request**: topic utilizzato per notificare l'MWbot di una nuova richiesta dagli utenti da aggiungere alla coda di attesa.

```
{
  "slot_id": "[1-9][0-9]*",
  "percentage": ([0-9]|([1-9][0-9])|(100)),
  "phone_number": "[0-9]{10}" | ""
}
```

- **iot/mwbot/debug**: topic utilizzato per scopi di debug per l'MWbot per avere il suo stato lavorativo e altre informazioni.

```
{
  "status": "IDLE" | "MOVING" | "CHARGING",
  "position": [1-9][0-9]*,
  "model": "<model name car>>",
  "percentage": ([0-9]|([1-9][0-9])|(100))
}
```

- **iot/mwbot/ack**: topic utilizzato per l'MWbot di notificare il backend che una richiesta di ricarica è stata completata.

```
{
  "slotId": [1-9][0-9]*,
  "kw": 0|[1-9][0-9]*
}
```

- **iot/mobiles/[0-9]{10}/new**: topic utilizzato dal debug per la creazione automatica dei dispositivi mobile/cellulare.

```
{
  "phoneNumber": "[0-9]{10}"
}
```