1. SPECIFICA
   1. DOMINIO

Commenti su?

* 1. CASI D’USO

Commenti sul diagramma dei casi d’uso.

1. PROGETTAZIONE
   1. Sottoinsiemi, interfacce e interazioni.

Diagrammi di sequenza

* 1. Definizione ENDPOINT API-REST e documentazione, AZIONI ed Errori

Parametri e formato delle informazioni scambiate

* 1. TOPIC MQTT

Formati dei messaggi inviati sui diversi topic

Abbiamo deciso di gestire il parcheggio formato da posti dedicati alla sosta e posti dedicati alla ricarica utilizzando un bot che gestirà la ricarica di un’auto alla volta.

Vi saranno 3 tipologie di utenti: BASE, PREMIUM e ADMIN che avranno rispettivamente permessi differenti per operare. Ogni utente per poter accedere all’applicazione dovrà prima effettuare il login e in caso non siano registrato devono effettuare prima la registrazione del suo account inserendo nome utente, e-mail e password utilizzando un username e una email che non siano presenti già nel database, inoltre email e password dovranno essere conformi agli standard di validità.

In seguito, potrà effettuare il login e a seguito anche il logout se è necessario.

L’utente BASE avrà la possibilità di cliccare il bottone “Entra ora” se vuole accedere al parcheggio sul subito se vi sono tutte queste condizioni:

1. Abbia un saldo sufficiente per la sosta o la ricarica desiderata, nel caso prima deve ricarica il saldo, associando oppure no una carta di credito
2. Abbia una macchina associata al suo account, nel caso prima deve creare una macchina con dati validi e che nel caso abbia scelto la ricarica che sia una macchina elettrica anche
3. Ci sia almeno un posto libero per la ricarica o per la sosta, che nel caso non vi sia verrà visualizzato quando sarà disponibile un posto.
4. Non vi siano delle prenotazioni riferite all’utente a cui vi è associato un ritardo, nel caso devono essere pagate tutte prima

L’utente BASE avrà la possibilità anche di fare un abbonamento e di diventare così utente PREMIUM, solamente se ha il saldo disponibile per farlo e solamente se non è già PREMIUM oppure ADMIN.

L’utente PREMIUM a differenza di quello BASE potrà effettuare la prenotazione del parcheggio cliccando il bottone “Prenota” per prenotare la sosta o la ricarica nella data e ora desiderata e valgono le stesse regole che vi sono sopra, in più verrà controllato anche che il tempo selezionato dall’utente per la sosta oppure, in caso di ricarica, il tempo di ricarica calcolato in base alla percentuale di ricarica che l’utente aveva inserito non si sovrappongano con altre prenotazioni.

Ogni utente, inoltre, se vorrà potrà cliccare “Utente” e verrà fuori una tendina con la voce “Esci Prima dal parcheggio” e nel caso vi sia una macchina all’interno del parcheggio riferita all’utente, l’utente potrà così uscire dal parcheggio.

Ogni utente inoltre potrà modificare i dati di una macchina a lui associata oppure cancellarla.

Infine, vi è l’utente ADMIN, il quale può anche lui fare le operazioni che fa l’utente PREMIUM,

ma ha altre operazioni accessorie alla sua figura, ovvero può stampare tutti i pagamenti per sosta, ricarica, oppure stampare tutti i pagamenti per categoria di utente (ADMIN, PREMIUM, BASE).

L’ADMIN potrà cancellare tutto il database oppure può cancellare tutto ma mantenendo gli utenti e lo storico tramite il bottone “Cancella prenotazioni” nella tendina ADMIN.

Inoltre, potrà attraverso il bottone “Sposta le soste” cancellare le prenotazioni finite e le inserisce all’interno dello Storico.

L’ADMIN inoltre potrà registrare un account ADMIN nel caso vi sia necessità di un altro account amministratore.

Poi potrà anche aggiornare i costi della sosta e della ricarica del parcheggio.

Infine, abbiamo inserito un bottone per poter impostare un ritardo su delle prenotazioni che scegliamo in modo da poter testare il funzionamento del bot sulla gestione dei ritardi e la gestione generale del parcheggio, in modo che se una prenotazione non fosse stata in ritardo sarebbe stata messa, una volta completata, all’interno dello storico; altrimenti la prenotazione veniva inserita nella tabella delle prenotazioni in ritardo, ovvero che hanno sforato le tempistiche di uscita dal parcheggio.

COMPONENTI

Abbiamo 2 cartelle che sono “parcheggio” e “ParcheggioGUI”, il primo è per la parte di

back-end e conterrà oltre al codice riguardante il back-end, anche la parte che riguarda la Sbarra e l’emulatore, il secondo riguarda invece tutta la parte grafica che permetterà all’utente di poter operare sul parcheggio.

Nella cartella Parcheggio c’è la suddivisione di tutta la parte che gestisce le richieste tramite API, difatti quando l’utente sceglierà quale operazione eseguire in base a quelle proposte sul sito, verrà richiamata l’API dedicata che è definita dentro ad una delle 7 classi che fanno da controller e difatti si trovano all’interno della classe “Controllers”:

* AdminController 🡪 API per le operazioni dell’ADMIN
* ChargeRequestController 🡪 API che riguardano le richieste di carica
* DelayedReservationController 🡪 API che riguardano le prenotazioni in ritardo
* MacchineController
* PaymentController
* ReservationController 🡪 API che riguardano le prenotazioni
* UserController

Al link <https://localhost:8443/swagger-ui/index.html#/> vi è la documentazione riguardante le API che si trovano all’interno dei controller sopra citati.

Ad ogni API è riferito un metodo e le classi Controller andranno a richiamare dei metodi delle classi Service che faranno dei controlli e eseguiranno delle operazioni sui dati che vengono passati, infine verranno poi chiamati i relativi metodi delle classi Repository, i quali ad ognuno verrà associata una query che verranno eseguite sul database.

I Service:

* ChargeRequestService
* MacchineService
* PaymentService
* ReservationService
* StoricoService
* UserService

I Repository:

* ChargeRequestRepository
* DelayedReservationRepository
* MacchinaRepository
* ParkingSpotRepository
* PaymentRepository
* ReservationRepository
* StoricoRepository
* UserRepository

All’Interno della cartella Parcheggio vi è la cartella Security dove vi sono rispettivamente i seguenti file:

* HttpsRestTemplateConfig: configura un RestTemplate per effettuare chiamate con HTTPS e legge da un trust store(keystore.jks) per validare le connessioni HTTPS. Utilizza **Apache HttpClient** per creare un client HTTP personalizzato con supporto SSL/TLS. Quindi con il RestTemplate risultante possiamo effettuare richieste in maniera sicura e affidabile.
* JwtAuthenticationFilter: permette di filtrare tutte le richieste in modo da vedere ad esempio se contengono un Token JWT e nel caso validarlo.
* JwtUtil: contiene tutti quei metodi per creare, gestire e manipolare il token
* SecurityConfig: permette di configurare la sicurezza della’applicazione realizzata tramite SpringBoot utilizzando Spring Security e come prima cosa viene utilizzato un filtro di autenticazione basato su JWT che viene eseguito prima del filtro di autenticazione con username e password. Inoltre vengono definite le politiche di accesso ai vari end-point, configura il CORS permettondo così richieste anche da altri domini specificati, richiede HTTPS per le richieste e inoltre da il permesso di utlizzo dell’interfaccia H2 per la gestione del database.

MQTT

FARE!

BOT

Abbiamo deciso di implementare un bot che in base alla coda delle prenotazioni, lui prenda quella in testa, e simuli la ricarica della macchina aspettando il tempo di carica che era stato calcolato in precendenza, poi viene notificato tramite MQTT l’inizio e la fine della carica della macchina, l’apertura o la chiusura della sbarra e infine anche lo stato di occupazione del parcheggio invocando anche la funzione per l’aggiornamento dello stato delle lampadine.

Il bot controllerà costantemente se vi sono auto da ricaricare, e se non vi sono auto attende un certo periodo di tempo prima di riprovare.

Il bot è eseguito in maniera asincrona in modo da permetterne l’esecuzione su un thread separato senza bloccare l’applicazione principale.

SBARRA

La componente Sbarra è formata principalmente da un UI che rapprensenterà la sbarra a livello grafico, un model chiamato “BarrierState”, ed infine un controller che ha all’interno due API che sono mappate nel seguente modo:

* /api/barrier/open 🡪 per l’apertura della sbarra
* /api/barrier/close 🡪 pr la chiusura della sbarra

Al link <http://localhost:9090/swagger-ui/index.html#/> vi è la documentazione riguardante le API di Sbarra.

HUE EMULATOR

PARCHEGGIOGUI

Riguarda il front-end ed è formato da tutti quegli elementi che permetteranno all’utente di accedere alla home del parcheggio e così poter prenotare il parcheggio e svolgere altre operazioni. L’utente potrà vedere nella home lo stato del parcheggio vedendo quante auto sono in carica e quante in sosta ed inoltre vedrà la posizione del parcheggio sulla mappa.

L’interfaccia grafica dell’applicazione è sviluppata e organizzata secondo il framework ASP.NET Core Razor Pages / Blazor e avremo una cartella chiamata “Pages” dove vi saranno i vari file dei quali ognuno svolgerà un operazione differente del parcheggio, manipolando i dati e richiamando le API del back-end attraverso il codice C# e si occuperà quindi anche della rappresentazione delle pagine definendo l’HTML con codice Razor.

H2-CONSOLE

Abbiamo deciso di utilizzare h2 perché è un database leggero che funziona all’interno dell’applicazione senza bisogno di un server separato, inoltre ha una console web che permette di eseguire query, visualizzare dati e gestire il database. Inoltre, non richiede un’installazione complessa e si può integrare con Spring Boot in pochi passi.

DIAGRAMMI DI SEQUENZA

1.EntraOraSeq e 2.PrenotaSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l’username dell’utente tramite token e in seguito controlliamo se all’utente sono associate delle prenotazioni in ritardo, nel caso vi siano l’utente non potrà proseguire e non potrà così entrare nel parcheggio; se non vi sono così prenotazioni in ritardo associate alll’utente potrà selezionare come prima cosa se vuole che avvenga la ricarica della macchina cosa potrà inserire i kw altrimenti selezionerà la sosta e così potrà inserire il tempo di sosta. Verrano visualizzate in seguito solo le macchine elettriche se è richiesta la ricarica, altriementi tutte le macchine associate dall’utente. Nel caso l’utente non abbia macchine a lui associate non potrà proseguire fino a che non ne registrerà una. Dopo aver selezionato la macchina viene verificata poi la disponibilità per controllare se vi è almeno un posto libero per la macchina dell’utente, se non vi è disponibilità viene visualizzato l’orario in cui l’utente potrà riprovare per accedere al parcheggio. Nel caso vi sia disponibilità allora viene creata la richiesta e così l’utente può entrare nell’immediato nel parcheggio.

1. AbbonamentoSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l’username e ruolo dell’utente tramite token, in seguito andiamo a controllare se l’utente è BASE e in questo caso potrà proseguire e si controllerà se il saldo dell’utente è sufficiente per fare l’abbonamento; in caso non sia sufficiente verrà chiesto di ricarica il saldo e di riprovare successivamente l’operazione, se invece è sufficiente si chiamerà la relativa API e l’utente passerà da BASE a PREMIUM.

1. Aggiorna-costoSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l’username dell’utente tramite token. L’utente che in questo caso è l’ADMIN potrà modificare i costi per la ricarica oppure per la sosta, nel caso il valore del costo dovrà essere uguale o maggiore di 0 e solamente in questo caso verrà richiamata la relativa API per andare a modificare i costi.

1. AggiornaCartaSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l’username dell’utente tramite token. In seguito l’utente inserirà il numero della propria carta e verrà verificato che il numero fornito sia valido, nel caso si si andrà a richiamare l’API appropriata per andare ad assegnare il relativo numero di carta all’utente, altrimenti verrà segnalato che il numero di carta inserito non è valido.

1. AggiornasaldoSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l’username dell’utente tramite token. Subito dopo verrà verificato se l’utente abbia un numero di carta associato, solamente se ha una carta asscoiata potrà proseguire con l’aggiornamento del saldo e l’utente dovrà inserire un saldo che sia maggiore di 0 altrimenti verrà segnalato errore fino a che il saldo non è corretto. Dopo aver inserito un valore corretto per il saldo si andrà così ad aggiornare il saldo dell’utente.

1. CancellaprenotazioniSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l’username dell’utente tramite token. In seguito sia andranno ad eliminare le prenotazioni, le richieste di carica e i pagamenti.

1. CancellatuttoSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l’username dell’utente tramite token. Poi vengono chiamate le Api relative alla cancellazione delle tabelle Reservation, DelayedReservation, Payment, ChargeRequest , Macchina, User.

1. CreaEModificaMacchinaSeq

Come prima cosa tramite la OnInitializedAsync() andiamo ad ottenere l’username dell’utente tramite token. Si controlla poi che la targa inserita dall’utente sia valida, nel caso quindi l’utente inserisce il modello e i kw della batteria se l’auto è elettrica.

1. EliminaMacchinaSeq

Come prima cosa l’utente seleziona la targa della macchina a lui associata che vuole eliminare, dopo aver scelto la targa clicca il bottone appropriato per l’eliminazione della macchina, alche verrà richiamata l’API e si procederà all’eliminazione della macchina. Se l’utente non ha una macchina associata comparirà un messaggio che dirà che finché non aggiungerà una macchina non potrà eliminarne alcuna.

1. EsciPrimaSeq

Vengono mostrate a video tutte le prenotazioni che l’utente ha fatto e in quel caso l’utente clicca il bottone riferito alla riga della prenotazioni riferita alla macchina che vuole che esca prima.

1. LoginSeq

Dopo che l’utente avrà inserito la mail e la password corretti verrà chiamata l’API relativa al login che permetterà quindi l’accesso alle funzioni dell’applicazione all’utente ora che si è autenticato e ritornerà un token il quale sarà salvato nel LocalStorage in modo da essere utilizzato per le chiamate delle API che richiederanno nell’intestazione il token.

1. OccupazioneSeq

Verrà visualizzato lo stato attuale del parcheggio chiamando l’API per l’occupazione del parcheggio.

1. PagaRitardoSeq

Come prima cosa si ottiene il token dell’utente, poi vengono caricate le prenotazioni in ritardo che sono associate all’utente, l’utente selezionerà la prenotazione la prenotazione che vuole di cui vuole pagare la multa e prima di poter pagare verrà controllato se ha saldo sufficiente, e solamente nel caso in cui ce l’abbia, ovvero che sia almeno uguale al valore della multa, potrà pagare la multa, altrimenti dovrà prima aggiornare il saldo.

1. RegistraSeq

L’utente inserirà username e email che devono essere unici e non già esistenti ed inoltre dovrà inserire un password che sia valida.

1. SetRitardoSeq

Verranno carica le prenotazioni esistenti e l’ADMIN selezionerà la prenotazione a cui vuole assegnare un ritardo.

1. SpostaSosteSeq

Al click del bottone le soste che sono state ultimate verranno salvate all’interno dello Storico ed è eliminate dalla tabella Reservation, se la prenotazione era in ritardo viene anche salvata all’interno della DelayedReservation