Relazione sul Funzionamento del Progetto di Applicazione Web

Autore: *Federico Mondelli 20044357*

CONFIGURAZIONE e SETUP

Per configurare ed eseguire il progetto, seguire le seguenti istruzioni nel **seguente ordine**:

DATABASE:

Utilizzare un'interfaccia grafica per amministrare database PostreSQL (noi abbiamo usato pgAdmin 4) ed eseguire i seguenti comandi:

Creare nuovo DB

- Inserire:

Nome Database: "GestioneParcheggio", utente: "postgres" e password: "1234" (per maggiori info guardare "\util\DatabaseConnection.java" che gestisce la connessione al database

- Eseguire lo script.sql presente nella cartella: "*aa23-24-gruppo04*"

Nello script sono già presenti alcuni inserimenti:

utente admin: username: admin, password: admin (in cui si accede sempre dal form di Login)

2 ***parcheggi*** per utenti base e premium, 2 parcheggi accessibili solo tramite prenotazioni)

Alcuni utenti base e premium per testare le varie funzionalità che possono utilizzare gli utenti nell'applicazione web

Costi di ***sosta e ricarica*** che admin può modificare tramite la pagina dedicata

MOSQUITTO (se si vuole utilizzare il simulatore):

- *Installare mosquitto*
- Dal Terminale, digitare: *mosquitto -v*

BACKEND (Java con Spark):

Navigare nella cartella "*aa23-24-gruppo04 \backend-java*" ed eseguire i seguenti comandi:

- *gradle build*
- *gradle run* (per avviare l'applicazione)

Il server Spark ascolterà ascolterà sulla porta 8080

FRONTEND (Razor Page):

Navigare nella cartella "*aa23-24-gruppo04\frontend-razor\GestioneParcheggio*" ed eseguire i seguenti comandi:

- *dotnet restore*
- *dotnet run* (per avviare l'applicazione)

Aprire il browser web e digitare l'indirizzo: *https://localhost:7211*

SIMULATORE MQTT (Python):

Navigare nella cartella "*aa23-24-gruppo04\sottosistemaIoT*"

Eseguire il comando:

- *Terminale 2: python simulatore.py*

(senza mosquitto avviato, non esegue nulla)

SCELTE IMPLEMENTATIVE e NOTE

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.

SIMULATORE (Python):

Questo programma è un sistema automatizzato che simula il sistema di parcheggio attraverso l'uso di MQTT per la comunicazione tra i vari sottosistemi. E' composto da una Centralina che coordina il sistema, gestisce la comunicazione MQTT, controlla le operazione e i rilevamenti simulati dei vari attuatori (sensori occupazione, bot, fotocamera ingresso e uscita) e da un Simulatore che permette di simulare gli eventi e di visualizzare lo stato del parcheggio. La Centralina di connette al broker MQTT e avvia i componenti del sistema, mentre il Simulatore fornisce l'interfaccia utente.

Flussi del Simulatore:

<u>Ingresso:</u>

All'arrivo la fotocamera legge qrcode (per semplificare chiedo username) e targa e verifica la disponibilità di un posto auto controllato l'occupazione dei vari sensori (uno per ogni

posto auto). Se c’è un posto disponibile, l’ingresso è permesso e il sensore rileva l’occupazione del suo posto auto.

Se entra attraverso una prenotazione la centralina recupera il posto auto assegnatoli.

Durante la sosta:

Durante la sosta può richiedere una ricarica tramite il sito web e riceve una notifica, in una sezione dedicata nel sito, al completamento della ricarica effettuata dal bot e viene creata immediatamente la transazione.

Uscita:

Il sensore comunica la liberazione del posto auto e al momento dell’uscita, quando la fotocamera di uscita rileva qrcode e targa, viene effettuata la transazione della sosta.