Relazione per il progetto di Progettazione e Implementazione dei Sistemi Software in Rete

Luca Benetti 20043903
Anton Borislavov Iliev 20035170
Linda Monfermoso 20028464

27 ottobre 2024

1 Introduzione

Come previsto dal progetto presentato durante il corso, abbiamo creato un applicativo web pensato per gestire le ricariche di automobili elettriche, in più parcheggi, tramite robot autonomi.

Il progetto è stato realizzato in ASP.NET Core (linguaggio C#), su .NET Core 8 e Entity Framework. Per l'interfaccia grafica è stato utilizzato Razor, mentre per il database Sqlite (con Entity Framework Core) e LinQ. La comunicazione con il database da parte delle varie funzioni e servizi avviene tramite repository pattern.

I programmi CamSimulator e MonitorSimulator sono stati realizzati tramite Windows Form, e le funzionalità dei MwBot sono state implementate tramite protocollo MQTT, grazie alla libreria MQTTnet.

Per comunicare con gli utenti è stata utilizzata l'API di Telegram.

2 Specifica

2.1 Casi d'uso e requisiti

Il diagramma dei casi d'uso è disponibile nel file PDF apposito.

2.1.1 Descrizione casi d'uso e requisiti funzionali

1	Login	L'utente si autentica alla piattaforma
2	Creazione utente	Viene creato un utente nella piattaforma
3	Aggiorna dati utente	L'utente aggiorna i propri dati
4	Effettua pagamento	L'utente effettua un pagamento in seguito a una ricarica

5	Visualizza pagamento	L'utente visualizza il pagamento da effettuare
6	Creazione pagamento	Viene creato un pagamento in seguito a una ricarica finita
7	Crea prenotazione	L'utente premium crea una prenotazione
8	Elimina prenotazione	L'utente premium elimina una prenotazione
9	Modifica prenotazione	L'utente premium modifica una prenotazione
10	Elenco prenotazioni	L'amministratore visualizza elenco delle prenotazioni
11	Invio messaggio utente	Il sistema invia un messaggio (via Telegram) all'utente
12	Invio messaggio robot	Il sistema invia un messaggio (via MQTT) all'utente
13	Aggiunta/rimozione robot	L'amministratore aggiunge o rimuove un MWbot
14	Monitoraggio occupazione	Il sistema monitora le macchine in entrata e uscita per determinare occupazione dei posti
15	Aggiunta/rimozione auto	L'utente rimuove o aggiunge un'automobile
16	Aggiungi/rimuovi macchina coda ricariche	Il sistema aggiorna lo stato delle macchine in coda per ricarica
17	Elenco coda ricariche	L'amministratore visualizza l'elenco delle ricariche in coda
18	Aggiornamento costi ricarica	L'amministratore aggiorna i costi delle ricariche
20	Elenco pagamenti	L'amministratore visualizza l'elenco dei pagamenti
21	Rimozione utente	L'amministratore rimuove un utente dalla piattaforma
22	Elenco posteggi	L'amministratore visualizza un elenco dei posteggi di un parcheggio
23	Elenco auto	L'amministratore visualizza un elenco delle auto registrate alla piattaforma
24	Ricerca utente	L'amministratore ricerca un utente registrato
25	Ricerca parcheggio	L'amministratore ricerca un parcheggio

2.1.2 Requisiti non funzionali

- L'interfaccia è grafica e realizzata con Razor Pages
- Il database è realizzato con SQLite

- Le specifiche di progettazione sono realizzate con diagrammi UML
- Il sistema è implementato in .NET e EntityFramework
- La password è lunga 8 caratteri, con maiusole, minuscole e numero
- Le date sono memorizzate nel formato standard UTC
- Lo scambio di messaggi con MWbot avviene tramite MQTT
- Lo scambio di messaggi tra sistema di prenotazione e ricariche e utente avviene tramite Telegram
- Il pagamento avviene tramite PayPal
- La registrazione e l'autenticazione sono gestite dalla libreria Identity
- Il rilevamento dei posti occupati è gestito da un sensore posto all'entrata del parcheggio

2.2 Diagramma delle classi di dominio

Il diagramma delle classi di dominio è disponibile nel file PDF apposito.

3 Progettazione

3.1 Diagramma delle classi

I diagrammi di tutte le classi rilevanti per il progetto sono presenti nella cartella `Diagrammi Classe`.

3.2 Documentazione API

Per documentare le API abbiamo utilizzato Swagger. La lista di API utilizzate nel progetto è visibile alla pagina `https://localhost:7237/swagger/index.html` quando questo è in esecuzione. È anche presente un documento in formato PDF (api-documentation) qualora non fosse possibile visualizzare la pagina online.

3.3 MQTT

La gestione e comunicazione con i robot MWbot è stata implementata mediante il protocollo MQTT. All'avvio, l'applicativo verifica gli MwBot che sono attualmente online e li istanzia come client, connettendoli al broker ed effettuando l'inizializzazione dei parametri.

Come da protocollo, la nostra implementazione prevede la presenza di un broker e di più client, che rappresentano gli MWbot. In particolare, il broker è responsabile della ricezione e distribuzione di messaggi tra client MWbot e ne gestisce le richieste. Il client, invece, comunica con il broker pubblicando sui topic pertinenti.

3.3.1 Struttura messaggi MQTT

Classe rappresentante un messaggio MQTT:

```
    public class MqttClientMessage : MwBot
{
    public MessageType MessageType { get; set; }
    public int? ParkingSlotId { get; set; }
    public ParkingSlot? ParkingSlot { get; set; }
    public decimal? CurrentCarCharge { get; set; }
    public decimal? TargetBatteryPercentage { get;
        set; }
    public string? UserId { get; set; }
    public string? CarPlate { get; set; }
    public CurrentlyCharging? CurrentlyCharging {
        get; set; }
    public int? ImmediateRequestId { get; set; }
    public ImmediateRequest? ImmediateRequest { get;
        set; }
}
```

Tipi di messaggi MQTT da client a broker:

- RequestCharge: il MwBot chiede al broker di ricaricare un'auto
- CompleteCharge: il MwBot riferisce al broker che l'auto è completamente carica
- UpdateCharging: il MwBot riferisce al broker la sua percentuale di carica
- UpdateMwBot: il MwBot riferisce al broker il suo stato (StandBy, MovingToSlot, MovingToDock, Offline, ChargingCar, Recharging)
- RequestRecharge: il MwBot richiede al broker di ricaricarsi al dock
- DisconnectClient: il MwBot chiede di essere disconnesso
- RequestMwBot: connette client a MwBot

Tipi di messaggi MQTT da broker a client:

- StartCharging: indica al MwBot di iniziare la ricarica di un'auto
- StartRecharge: indica al MwBot di iniziare la ricarica della propria batteria al dock
- ChargeCompleted: indica al MwBot che la ricarica dell'auto è completa, in quanto percentuale specificata dall'utente
- ReturnMwBot: indica al MwBot di ritornare al dock
- StopCharging: indica al MwBot di interrompere il processo di ricarica

L'emulazione del movimento, della ricarica delle auto e della ricarica degli MwBot è simulato nella classe `MqttMwBotClient`, nelle funzioni `SimulateMovement`, `SimulateChargingProcess` e `SimulateRechargingProcess`.

4 Implementazione

4.1 Istruzioni di installazione

Si consiglia l'utilizzo del programma Visual Studio per avviare l'applicativo web e i due programmi che permettono di emulare l'entrata delle auto nel parcheggio (CamSimulator) e l'occupazione dei posti auto (MonitorSimulator).

Una volta aperta la soluzione con Visual Studio, è necessario aggiungere i segreti utente. Per farlo, cliccare su `Progetto.App`, selezionare `Gestione segreti utente`, e aggiungere il codice presente nel file segreti.json.

Aggiunti i segreti, è sufficiente fare click destro su `Solution Pissir.Progetto` nella schermata `Solution Explorer`, selezionare `Configure startp projects...`, selezionare tutti i progetti e premere su `Start` nella schermata principale.

Avviata la soluzione, è possibile utilizzare l'applicativo web recandosi all'indirizzo `https://localhost:7237`, emulare l'entrata/uscita delle auto tramite CamSimulator e visionare lo stato di occupazione dei posteggi tramite MonitorSimulator.

Utenze:

- admin@admin.it - Admin1234$
- utente01@utente.com - bHw2Hs3!4bzq-aQ
- utentepremium01@upremium.com - SgRE5GZp_rj6s2E

4.2 Progetto.App

Progetto.App implementa l'applicativo web. Contiene configurazione all'avvio, log, controller con i vari endpoint con cui interfacciarsi per le richieste e le pagine front-end di interfaccia.

Sono presenti tre tipi di utente:

- Utente base: può visionare i parcheggi disponibili, il loro stato d'occupazione ed effettuare ricariche e/o soste scegliendo dal menù "Dashboard > Services" quando la sua auto viene rilevata dalla telecamera in entrata;
- Utente premium: come utente base, ma può prenotare una ricarica tramite il menù "Dashboard > Reservation";
- Utente amministratore: ha il controllo completo sui parcheggi; può quindi effettuare alcune operazioni CRUD non-critical tramite i vari menù, accendere / spegnere i robot, filtrare i pagamenti effettuati.

4.3 Progetto.App.Core

Progetto.App Core (domain + application) contiene la configurazione del database, i modelli utilizzati dall'applicativo web, le migration, i validator dei modelli, le repository per interfacciarsi con il database e i servizi.

4.4 CamSimulator

CamSimulator è un'applicazione di tipo Windows Form che rappresenta la telecamera con rilevamento targa presente all'entrata di uno specifico parcheggio (da selezionare). Rileva targhe in entrata / uscita: in seguito all'entrata, viene richiesto all'utente sulla pagina Dashboard/Servizi se effettuare una sosta o una ricarica. Si interfaccia con l'applicazione web tramite Post e Get a due endpoint ApiRest presenti in un controller dedicato.

4.5 MonitorSimulator

Monitor Simulator è un'applicazione di tipo Windows Form che rappresenta i sensori presenti nei posti auto. Tramite essa è possibile controllare l'occupazione dei posti auto in vari parcheggi.

4.6 PayPal.REST

Implementa la logica dei pagamenti tramite PayPal. È una API di tipo REST.