



UNIVERSITÀ DEGLI STUDI DI GENOVA

SCUOLA DI SCIENZE MATEMATICHE, FISICHE E NATURALI

CORSO DI LAUREA IN INFORMATICA

ANNO ACCADEMICO 2021/2022

Sviluppo di un portale web per il censimento e controllo di colonnine di ricarica per veicoli elettrici

Marzo 2023

Candidato
Timossi Luigi

Relatore
Prof. Delzanno Giorgio

Dibris

Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi

Abstract

La transizione verso la mobilità elettrica è un fenomeno ormai avviato e in rapida crescita. I dati presentati di recente all'ExpoMove 2021 ne sono una prova e i dati degli ultimi anni lo dimostrano. Nonostante i numeri del mercato dei veicoli elettrici in Italia siano ancora limitati, le tendenze dimostrano una chiara tendenza alla crescita: +89% di auto elettriche immatricolate nella prima metà del 2018, +750 punti di ricarica pubblici durante il 2017 (e-mobility report 2018, Politecnico di Milano).

In questo scenario diventa sempre maggiore la necessità di spingere sulla contemporanea diffusione delle infrastrutture per supportare la mobilità elettrica, indirizzandosi sulla installazione di colonnine di ricarica, in particolar modo sulle vie ad alto scorrimento e nelle aree urbane.

In collaborazione con l'azienda italiana Gruppo SIGLA S.r.l. del gruppo RELATECH si è scelto di svolgere un lavoro di tesi incentrato sul censimento di colonnine elettriche creando un portale web accessibile sia da clienti che da amministratori.

Indice

Acronimi	iii
1 Introduzione	1
1.1 Scopo del progetto	1
2 Architettura	3
2.1 Architettura di riferimento	3
2.2 Roaming	5
2.3 Protocollo OCPP	6
2.4 Protocollo OCPI	7
2.5 Architettura progetto	8
3 Creazione database	10
3.1 Differenze tipologie database	11
3.2 SQL Server	13
4 Backend	17
4.1 Set Up	17
4.2 HashPasswordService	18
4.3 ServerDB	19
4.4 ServerAPI	21
4.4.1 Impostazioni del progetto - CORS	21
4.4.2 Impostazioni del progetto - Dependency Injection	21
4.4.3 Impostazioni del progetto - Autorizzazione e JWT	22
4.4.4 Controllers API	24
4.4.5 Mapping usato	27
4.5 Swagger per il Testing delle API	27
5 Frontend	30
5.1 Angular	30
5.1.1 PrimeNG	31
5.2 Login	32

5.2.1	Pagina di login	33
5.2.2	Invio token	35
5.3	Servizi Angular	36
5.3.1	auth.service	36
5.3.2	view-chargepoint.service	37
5.3.3	view-data.service	38
5.3.4	view-selected.service	39
5.3.5	edit-data.service	40
5.4	Route Guard per il controllo	41
5.4.1	authGuard	41
5.4.2	adminGuard	42
5.5	Dashboard	43
5.6	Geolocation	44
5.6.1	Tabella	44
5.6.2	Mappa	47
5.7	Pagina informazioni colonnine	48
5.8	Modifica colonnine	50
5.8.1	modifica delle informazioni generali	51
5.8.2	modifica degli orari di apertura	51
5.8.3	modifica dei dati energetici	52
5.8.4	modifica degli EVSE e connettori	52
5.9	Aggiunta colonnine	53
6	Conclusioni e ringraziamenti	55
6.1	Conclusioni	55
6.2	Ringraziamenti	55
	Elenco delle figure	57
	Bibliografia	59

Acronimi

CORS - Cross-origin resource sharing. 21

CPO - Charge Point Operator. 5–8

CSMS - Charging Station Management System. 4

DSO - Distribution System Operator. 4

eMSP - e-Mobility Service Provider. 5, 6

ESMIG - European Smart Metering Industry Group. 7

EV - Electric Vehicle. 3

EVSE - Electric Vehicle Supply Equipment. 4, 7, 8, 38, 45, 49, 50, 52, 53

HTML - Hypertext markup language. 6

JWT - JSON Web Token. 22–25, 57

OCPI - Open Charge Point Interface. 5, 7–9, 13

OCPP - Open Charge Point Protocol. 4, 6

RFID - Radio-Frequency IDentification. 4

SOAP - Simple Object Access Protocol. 6

XML - Extensible Markup Language. 6

Capitolo 1

Introduzione

1.1 Scopo del progetto

La transizione verso la mobilità elettrica è un fenomeno ormai avviato e in rapida crescita. I dati presentati di recente all'ExpoMove 2021 ne sono una prova e i dati degli ultimi anni lo dimostrano. Nonostante i numeri del mercato dei veicoli elettrici in Italia siano ancora limitati, le tendenze dimostrano una chiara tendenza alla crescita: +89% di auto elettriche immatricolate nella prima metà del 2018, +750 punti di ricarica pubblici durante il 2017 (e-mobility report 2018, Politecnico di Milano). Il 2020 non è stato da meno e ha visto un incremento costante nelle vendite delle auto elettriche e ibride con una crescita del 237% negli ultimi tre mesi dell'anno. Inoltre, Tra il 2020 e il 2022 le vendite sono oltre che raddoppiate. In questo scenario diventa sempre maggiore la necessità di spingere sulla contemporanea diffusione delle infrastrutture per supportare la mobilità elettrica, indirizzandosi sulla installazione di colonnine di ricarica, in particolar modo sulle vie ad alto scorrimento e nelle aree urbane. In questo contesto si focalizza l'attività di tirocinio e tesi proposta da un'azienda genovese del settore ICT, Gruppo SIGLA S.r.l. (www.grupposigla.it) parte del gruppo RELATECH, molto attiva in azioni di innovazione a livello nazionale ed internazionale.

L'azienda ha un gruppo di lavoro dedicato ad attività nell'ambito della mobilità elettrica che lavora ad ampio spettro su diversi elementi di quest'ultimo dal controllo delle infrastrutture di ricarica a quella dei protocolli di comunicazione, integrazione e controllo delle singole postazioni di ricarica agli applicativi di gestione del servizio coi clienti finali e gli utenti delle stazioni di ricarica. L'attività proposta è collaterale alle attività che Gruppo SIGLA svolge all'interno di molti dei suoi progetti commerciali e fa riferimento, più in generale, alle proprie pratiche di implementazione software e rispecchia le scelte tecnologiche aziendali nell'ambito della mobilità elettrica. Dunque, è questo il contesto nel quale si colloca l'attività proposta e che vuole eseguire una serie di attività esplorative complementari rispetto ai temi illustrati.

Lo scopo del progetto è quello di creare un portale web che funga da hub tra i fornitori e gestori di colonnine elettriche, al fine di raggruppare le informazioni in modo che i clienti non si ritrovino divisi tra i vari gestori e fornitori. Questo progetto si basa sulle scelte tecnologiche dell'azienda nell'ambito della mobilità elettrica e vuole essere di supporto nel rifacimento e nell'ammodernamento delle proprie

soluzioni software dedicate a questo settore.

Il progetto si concentra sull'integrazione e il controllo delle singole postazioni di ricarica, sui protocolli di comunicazione, sugli applicativi di gestione del servizio coi clienti finali e gli utenti delle stazioni di ricarica. L'obiettivo del lavoro svolto vuole essere in prima istanza l'analisi e l'implementazione degli applicativi di controllo e monitoraggio delle infrastrutture di ricarica dei veicoli elettrici in via di rinnovamento da parte di Gruppo SIGLA e l'individuazione di ulteriori elementi degli stessi discussi insieme al laureando. In definitiva, il lavoro vuole essere di supporto all'azienda nel rifacimento e nell'ammodernamento delle proprie soluzioni software dedicate al mondo della mobilità elettrica. Il documento e i capitoli che seguono descrivono nel dettaglio il lavoro svolto, sottolineandone i singoli passi e raccontando il processo di analisi e le scelte tecnologiche per eseguire l'attività finalizzata, come detto, a implementare un applicativo back-end e front-end che funga da hub per la fornitura di servizi di mobilità elettrica.

Capitolo 2

Architettura

In questo capitolo verrà presentata l'architettura su cui è basato l'intero progetto di laurea. Verranno descritte le infrastrutture già esistenti e i protocolli standard usati per la realizzazione di una rete di ricarica efficiente e affidabile. Inoltre, questo capitolo fornirà una panoramica completa di cosa sia una colonnina di ricarica elettrica, di quali componenti è formata e di come si interfaccia con l'esterno.

2.1 Architettura di riferimento

Per la creazione del portale web, ci siamo affidati all'infrastruttura di mobilità elettrica già esistente. A tal proposito, si può far riferimento alla Figura 2.2 in cui viene mostrata un'architettura di riferimento. Come si può notare, sono presenti molti attori e vengono usati diverse tipologie di comunicazioni tra le varie entità. Abbiamo un EV, che viene rappresentato dall'icona di una macchina elettrica, che si collega tramite un connettore alla stazione di ricarica.

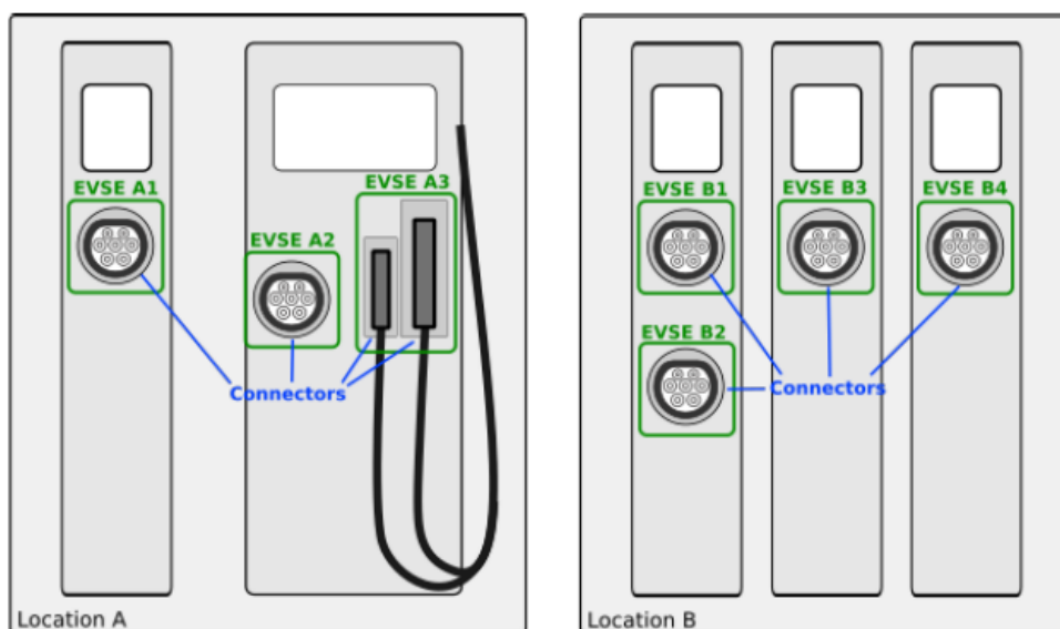


Figura 2.1: Esempio dei componenti principali di una stazione di ricarica

Una stazione di ricarica può essere formata da varie componenti, quali per esempio: uno schermo, un tastierino o un lettore RFID. Le componenti che ci interessano in questo momento sono tre:

- EVSE, ovvero un modulo della stazione di ricarica gestita e operata in modo indipendente, contiene uno o più connettori ma può fornire energia solo ad un veicolo alla volta;
- Connettore, la presa elettrica sulla stazione di ricarica gestita e utilizzata indipendentemente. Corrisponde a un connettore singolo fisico. In molti casi un EVSE può avere diversi tipi di presa fisica e/o cavi collegati per facilitare diversi tipi di veicoli;
- Local Controller, dispositivo logico tra il CSMS e la colonnina. Ha l'abilità di controllare la carica di un gruppo di stazioni di ricarica;

Oltre alla colonnina si hanno altri attori esterni che completano il quadro dell'infrastruttura: Gli operatori di sistema di distribuzione, o DSO, sono le entità incaricate di distribuire e gestire l'energia lungo il percorso che va dalla fonte all'utente finale. Essi comunicano con il sistema centrale fornendo previsioni sulla capacità di determinate zone e stabilendo dei limiti sull'energia/corrente utilizzabile.

il sistema centrale (CSMS) comunica end-to-end con la colonnina attraverso il protocollo OCPP (verrà illustrato nei prossimi capitolo) e comunica usando un altro protocollo con il DSO. Infine, abbiamo un possibile HUB che impartisce direttive al CSMS permettendo la supervisione e il controllo di un possibile supervisore o amministratore.

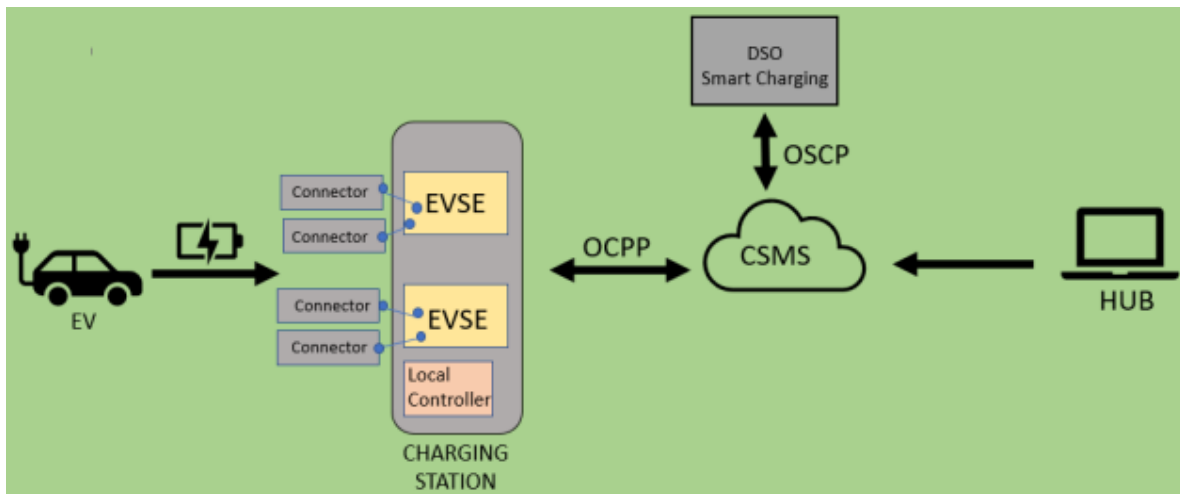


Figura 2.2: Schema base di una possibile infrastruttura di ricarica elettrica.

La comunicazione tra stazione di ricarica e sistema centrale è solitamente diretta. Tuttavia, possono esistere altre configurazioni che prevedono un'entità nel mezzo (ad es. proxy) che utilizzano diversi protocolli di comunicazione.

2.2 Roaming

[1] Essendoci più fornitori di servizi di ricarica è stato inserito il meccanismo di roaming usato appunto per collegare diversi eMSP (e-Mobility Service Provider). Questo meccanismo è considerato fondamentale poiché:

- migliora l'esperienza degli utenti, permettendo loro di utilizzare i punti di ricarica non gestiti dal proprio fornitore di servizi;
- migliora la comunicazione tra vari eMSP, creando uno standard per lo scambio di informazioni.

In questo scenario oltre al eMSP abbiamo un altro attore importante:

il CPO (Charge Point Operator) è colui che ha il ruolo di gestore delle colonnine e permette agli utenti di ricaricare i propri veicoli.

I due ruoli non sono esclusivi, esistono diverse topologie in cui possono esserci piattaforme che operano esclusivamente come CPO o eMSP oppure piattaforme che possono assumere tutti e due i ruoli.

Il protocollo OCPI prevede una connessione diretta fra i server dei gestori con scambio di specifici messaggi. In alternativa esistono Hubs e Girevi, i quali agiscono come hub/accentratori tra i gestori definendo un proprio protocollo e gestendo le fatturazioni. L'approccio più usato risulta essere OCPI in quanto permette una connessione diretta senza dover passare per terze parti che gestiscono la comunicazione, anche se nelle ultime versioni del protocollo viene prevista la connessione di piattaforme attraverso hub.

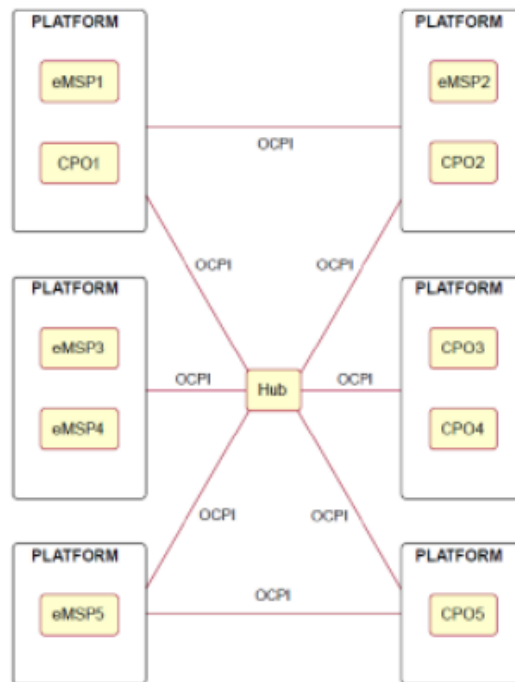


Figura 2.3: Topologia generale OCPI.

In figura 2.3 possiamo veder un esempio di topologia generale in cui sono presenti diverse piattaforme (piattaforme solo CPO/eMSP e piattaforme che assumono tutti e due i ruoli) dove le comunicazioni avvengono sia direttamente che tramite hub.

2.3 Protocollo OCPP

Il protocollo Open Charge Point Protocol come spiegato nel sito [2] è utilizzato per la comunicazione tra le colonnine di ricarica e il sistema centrale. Esso definisce le procedure e gli scambi di messaggi per autenticazione, autorizzazioni, transazioni, controllo remoto, visualizzazione della disponibilità e prenotazione delle colonnine, nonché per l'aggiornamento del firmware e la diagnostica. Ci sono due modalità di ricarica: quella fisica, in cui l'utente effettua azioni dirette sulla colonnina, e quella remota, tramite un'applicazione del gestore della stazione di ricarica. In entrambi i casi, il sistema centrale verifica l'abilitazione dell'utente alla ricarica e, se accettata, avvia la sessione di ricarica.

Col tempo sono state definite più versioni di OCPP tra cui:

- Versione 1.5: dove lo standard utilizza il Simple Object Access Protocol (SOAP) come mezzo per inviare messaggi tra componenti su Internet. SOAP viene usato come framework che semplifica l'invio e la ricezione di messaggi, rendendo veloce l'implementazione. I messaggi SOAP sono scritti in Extensible Markup Language (XML), un linguaggio simile all'HTML utilizzato per la navigazione su Internet. I messaggi XML possono contenere non solo testo ma anche immagini e codice eseguibile. Il vantaggio principale di questo formato è che i messaggi sono leggibili e comprensibili.
- Versione 1.6: Il protocollo OCPP 1.6 offre nuove caratteristiche rispetto alla versione precedente, tra cui lo Smart Charging, l'utilizzo di JSON su WebSockets, maggiori possibilità di diagnostica, più stati dei punti di ricarica e il TriggerMessage. Con l'introduzione di OCPP 1.6, ci sono due varianti del protocollo: quella basata su SOAP e quella basata su JSON (più compatta). Nella comunicazione per identificare l'implementazione utilizzata, si utilizzano i suffissi -J e -S per indicare JSON o SOAP rispettivamente.
- Versione 2.0.1: in questa ultima versione la struttura e il metodo di segnalazione delle transazioni sono stati semplificati, inoltre non vi è più un supporto del protocollo SOAP. In OCPP 1.X, la segnalazione dei dati delle transazioni era suddivisa in messaggi separati come StartTransaction, StopTransaction, MeterValue e StatusNotification. Con l'evoluzione del mercato verso una programmazione più avanzata, è emersa la necessità di una gestione più sofisticata dei dati delle transazioni. Tutti i messaggi StartTransaction, StopTransaction, MeterValue e StatusNotification relativi alle transazioni sono stati sostituiti dal messaggio 'TransactionEvent'. Il messaggio StatusNotification esiste ancora, ma solo per le notifiche di stato non legate alle transazioni, come la disponibilità del connettore.

Non entrerò più nel dettaglio poiché è stato necessario studiare il protocollo solo ed esclusivamente per conoscere l'architettura delle colonnine elettriche e come comunicano con l'esterno (per eventualmente rendere più completo il programma) e non mi è stato utile per l'implementazione del progetto.

2.4 Protocollo OCPI

Il protocollo OCPI (Open Charge Point Interface) come spiegato nel sito [3] è utilizzato per la comunicazione tra sistemi di ricarica per veicoli elettrici e sistemi di gestione dell'energia. Serve a garantire la interoperabilità tra diversi sistemi di ricarica, indipendentemente dal produttore o dal fornitore, in modo che i veicoli elettrici possano ricaricarsi in qualsiasi punto di ricarica compatibile con il protocollo. In pratica OCPI consente ai veicoli elettrici di comunicare con le colonnine di ricarica e ai gestori delle colonnine di comunicare con i gestori dell'energia, permettendo di gestire la ricarica, la fatturazione e la gestione delle sessioni di ricarica in modo semplice ed efficiente. OCPI è stato sviluppato dall'associazione europea dell'energia elettrica ESMIG (European Smart Metering Industry Group) e supportato da molte aziende del settore della mobilità elettrica e dell'energia.

La versione presa in esame la 2.1.1 e include funzionalità come l'autorizzazione, lo scambio di dati tra colonnine di ricarica, la gestione delle sessioni di ricarica, i comandi remoti alle colonnine e lo scambio di informazioni per lo smart-charging tra le parti interessate.

Tra le sue principali caratteristiche ci sono il sistema di roaming bilaterale o tramite hub, l'accesso in tempo reale a informazioni su posizione, disponibilità e prezzo e la procedura di scambio dati prima, durante e dopo la transazione. Il protocollo si basa su HTTP e utilizza il formato JSON per le comunicazioni, protette a livello di trasporto tramite TLS e autenticazione token-based. Non è richiesto il certificato lato client, solo lato server. La topologia di ricarica consiste in tre entità:

- Il connettore (presa o cavo specifico per il veicolo elettrico);
- L'EVSE (che controlla l'alimentazione per un singolo veicolo in una singola sessione e può fornire più connettori, ma solo uno alla volta può essere attivo);
- La location (gruppo di uno o più EVSE legati tra loro per posizione geografica o spaziale, può essere una postazione specifica o l'ingresso di un parcheggio o di una comunità recintata).

Il protocollo OCPI utilizza queste tre entità per creare un'ecosistema standard e utilizzabile nella pratica: per esempio un EVSE è definito come un oggetto formato da:

- **uid**: stringa di massimo 39 caratteri che definisce un identificativo univoco tra tutte le piattaforme dei CPO;
- **evse.id**: stringa di massimo 48 caratteri che definisce l'identificativo dell'EVSE secondo la specifica indicata in "eMI3 standard version V1.0" (<http://emi3group.com/documents-links/>);
- **status**: enum che indica lo stato corrente dell'EVSE

- **status_schedule**: lista di oggetti a se, indica gli stati futuri pianificati
- **capabilities**: lista di enum che indicano le varie funzionalità presenti
- **connectors**: lista oggetti a se, sono i connettori presenti nell'EVSE
- **floor_levels**: stringa di massimo 4 caratteri che indica il piano dove è situato l'EVSE
- **coordinates**: oggetto a se che definisce le coordinate dell'EVSE
- **physical_reference**: stringa di massimo di 16 caratteri che indica il numero stampato sulla superfice dell'EVSE per l'identificazione da parte del cliente
- **directions**: lista di oggetti a se che fungono da informazioni per localizzare meglio l'EVSE
- **parking_restrictions**: lista di enum che indicano le varie restrizioni per il parcheggio
- **images**: lista di oggetti a se contengono una o più immagini raffiguranti l'EVSE
- **last_updated** Timestamp dell'ultimo aggiornamento

Spetta al CPO utilizzare quello che ha più senso in una situazione specifica. Mi sono basato sul Protocollo OCPI 2.1.1 per la creazione delle entità e relazioni del database.

2.5 Architettura progetto

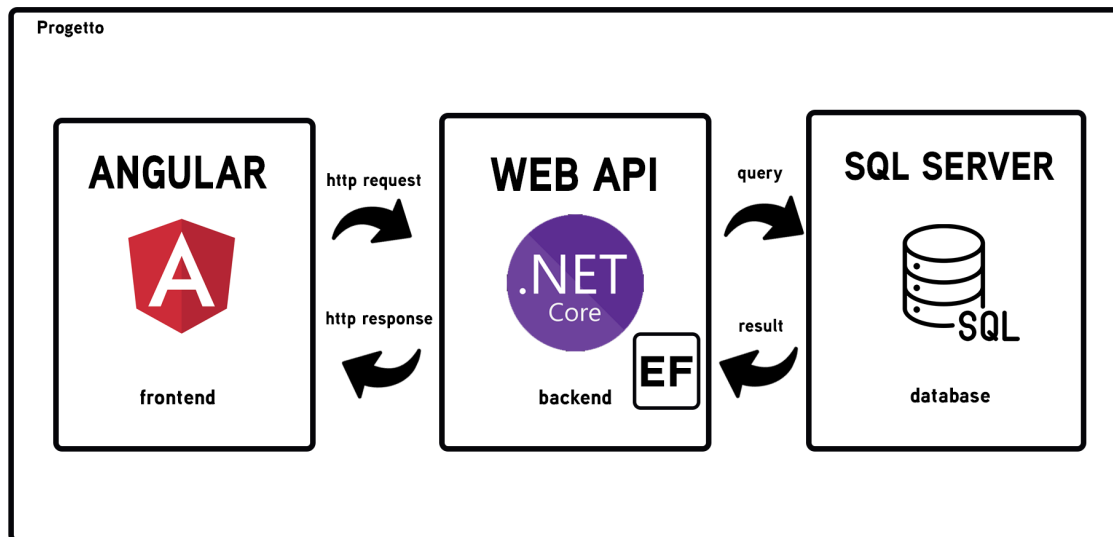


Figura 2.4: Architettura progetto

Il progetto, come spiegato precedentemente, è una soluzione complessa basata su tecnologie web, con un'architettura a servizi, adibita al censimento delle colonnine per la ricarica di veicoli elettrici. La soluzione progettata e realizzata è composta da tre livelli architetturali:

- Database: modello relazionale che definisce le varie entità e proprietà utilizzate nel protocollo OCPI, inoltre contiene le informazioni dei vari utenti iscritti al portale;
- Backend: Web API in dotnet 5 utilizzate per filtrare e controllare le comunicazioni tra portale e base di dati utilizzando entity framework, fornisce le operazioni disponibili in forma di API, controlli sui vari tipi di autenticazione e permessi ed è facilmente scalabile;
- Frontend: sito web realizzato in Angular dove gli utenti, dopo aver effettuato l'accesso, possono eseguire varie azioni inerenti al censimento, in base ai permessi a loro concessi.

Il Backend funge da ponte tra il portale web e il database, applicando filtri, controlli e operazioni prestabilite. La struttura base di una richiesta è la seguente: il frontend invia una richiesta HTTP (il frontend può effettuare due tipologie di requests: le requests che non richiedono una modifica al database, ovvero richieste di lettura, e le requests che richiedono una modifica del database, ovvero le richieste di scrittura) al webserver dove è situato il backend, una volta ricevuto il pacchetto HTTP il backend deve controllare:

- La validità della richiesta: ovvero se esiste un web token inserito nell'header http (necessario per accedere a qualsiasi API che non sia di login o di registrazione) e verificare che sia valido,
- I permessi della richiesta: ovvero se l'utente ha i permessi adeguati per interfacciarsi con l'API che ha richiesto (i permessi sono salvati nel web token).

Dopo aver completato i controlli (in caso l'utente non avesse superato tali controlli, il backend risponderà con il codice 400 se il web token non è presente nella richiesta o se sono sbagliati/mancanti alcuni campi, e con il codice 401 se il web token è scaduto, invalido o se non si ha il permesso per utilizzare l'API) il backend elaborerà la richiesta e nel caso eseguirà delle query sul database utilizzando entity framework come supporto. Una volta elaborata la richiesta, e se non ci sono stati errori nell'elaborazione, il backend invierà la risposta HTTP al frontend con i dati richiesti.

Capitolo 3

Creazione database

Per il progetto era necessario un database per archiviare i dati degli utenti e delle stazioni di ricarica. Inizialmente, è stato utilizzato MongoDB, ma successivamente è stato riscritto interamente appoggiandosi a SQL Server.

In un primo momento ho scelto di utilizzare MongoDB in funzione di alcune considerazioni: in primo luogo per le sue caratteristiche tecniche precipue (ad es. possibilità di archiviare i dati in forma di documenti JSON, assai comodo per implementare il collegamento con le colonnine di ricarica che utilizzano appunto il protocollo OCPP e JSON per comunicare); in secondo luogo, perchè inizialmente suggerito dalle fonti documentali utilizzate per approfondire dotnet e l'utilizzo delle WebAPI. In un secondo momento però mi è stato consigliato dall'azienda di utilizzare un database relazionale come SQL Server dato che era utile avere delle relazioni controllate tra i vari oggetti, la capacità di stabilire relazioni controllate tra oggetti diversi e la necessità di garantire che ogni dato avesse attributi specifici. Quindi utilizzare un database relazionale era più idoneo rispetto a un database schemaless e flessibile come MongoDB.

```

_id: ObjectId('6363e88391d75981bd5b29b5')
adress: "via pinco pallo 2"
city: "genova"
provider: "enelElectric"
heartbeat: 2022-11-01T12:07:40.000+00:00
name: "colonnina1"
h24: false
evse: Array
  0: Object
    status: "free"
    connectors: Array
      0: Object
        models: "tipoA"
      1: Object
      2: Object
    id_evse: "aa1"
latitude: 41.7738706
longitude: 12.6332836
closing_times: "23:00"
credit_card_payable: true
firmware_last_update: 2022-10-21T12:07:40.000+00:00
firmware_version: "15.22a"
floor_level: "T"
green_energy: true
opening_times: "9:00"
parking_restriction: "nessuna"
rfid_reader: true
zip_code: 16100
operator_name: "operatoreMagico"
type_location: "stazione normale"
version_connection: "1.6"

```

Figura 3.1: Esempio di un documento MongoDB

parliamo in generale delle principali differenze tra le due tipologie di database, dei pro, dei contro e dei motivi per cui preferire uno rispetto all'altro:

3.1 Differenze tipologie database

La differenza principale tra i database relazionali e non relazionali è la modalità di archiviazione dei dati. I database relazionali (come SQL) utilizzano tabelle con colonne e righe per rappresentare i dati, mentre i database non relazionali (come mongoDB) utilizzano documenti, chiavi-valore o grafi.

La scelta tra un database relazionale o non relazionale dipende dalle esigenze specifiche del progetto. I database relazionali sono adatti ai sistemi di gestione di grandi quantità di dati strutturati e all'esecuzione di query complesse, mentre i database non relazionali sono più flessibili e scalabili e sono adatti ai sistemi che gestiscono grandi quantità di dati semi-strutturati o non strutturati, in sintesi:

Pro dei database relazionali:

- Struttura rigida per i dati: i dati vengono archiviati in tabelle con colonne e righe ben definite, il che facilita la gestione dei dati e la creazione di relazioni tra di essi.
- Query complesse: i database relazionali supportano le query complesse e offrono un'alta performance per le operazioni di ricerca.
- Integrità dei dati: i database relazionali hanno un sistema di integrità dei dati molto rigoroso che aiuta a garantire che i dati siano sempre corretti e coerenti.

Contro dei database relazionali:

- Scalabilità: i database relazionali possono diventare lenti e poco flessibili quando le quantità di dati aumentano.
- Dati semi-strutturati o non strutturati: i database relazionali possono avere difficoltà a gestire i dati semi-strutturati o non strutturati.

Pro dei database non relazionali:

- Scalabilità: i database non relazionali sono progettati per scalare in modo fluido e gestire grandi quantità di dati.
- Dati semi-strutturati o non strutturati: i database non relazionali sono molto flessibili e possono gestire facilmente i dati semi-strutturati o non strutturati.
- Performance: i database non relazionali offrono prestazioni elevate per le operazioni di lettura e scrittura.

Contro dei database non relazionali:

- Query complesse: i database non relazionali possono avere difficoltà a gestire le query complesse.
- Integrità dei dati: la gestione dell'integrità dei dati in un database non relazionale può essere più difficile rispetto a un database relazionale.

In generale, se si ha bisogno di una struttura rigida per i dati e di un supporto robusto per le query complesse, è preferibile utilizzare un database relazionale. Se si ha bisogno di una maggiore flessibilità e scalabilità, è preferibile utilizzare un database non relazionale.

3.2 SQL Server

In figura 3.2 vi è illustrato lo schema ER del database utilizzato, è basato su OCPI 2.1 ed è formato da due macroparti:

- Tabelle utilizzate per archiviare i dati degli utenti;
- Tabelle utilizzate per archiviare i dati delle stazioni di ricarica e i vari attributi legati.

La tabella utilizzata per archiviare gli utenti è la tabella "User" che contiene i campi utilizzati per il login, il ruolo dell'utente e i token utilizzati nell'autenticazione. il resto delle tabelle sono state create per replicare gli oggetti del protocollo in questione e sono conformi ai vari vincoli da esso descritti. per semplicità e necessità sono stati aggiunte altre tabelle e campi:

- Le tabelle aventi il prefisso "Has" sono da considerarsi tabelle relazionali utilizzate per collegare due entità in una relazione molti a molti, quindi generalmente contengono le chiavi esterne delle due entità a cui sono connesse come chiave primaria;
- Le tabelle aventi il prefisso "C" seguito da un'altra lettera maiuscola sono le tabelle utilizzate per archiviare gli enum utilizzati nel protocollo: la chiave primaria è un numero intero che rappresenta la posizione nell'array e la descrizione è il nominativo utilizzato per riconoscere tale valore (campo non presente nel protocollo dato che nel suddetto vengono usati solo i nomi del tipo di dato e non i valori numerici).

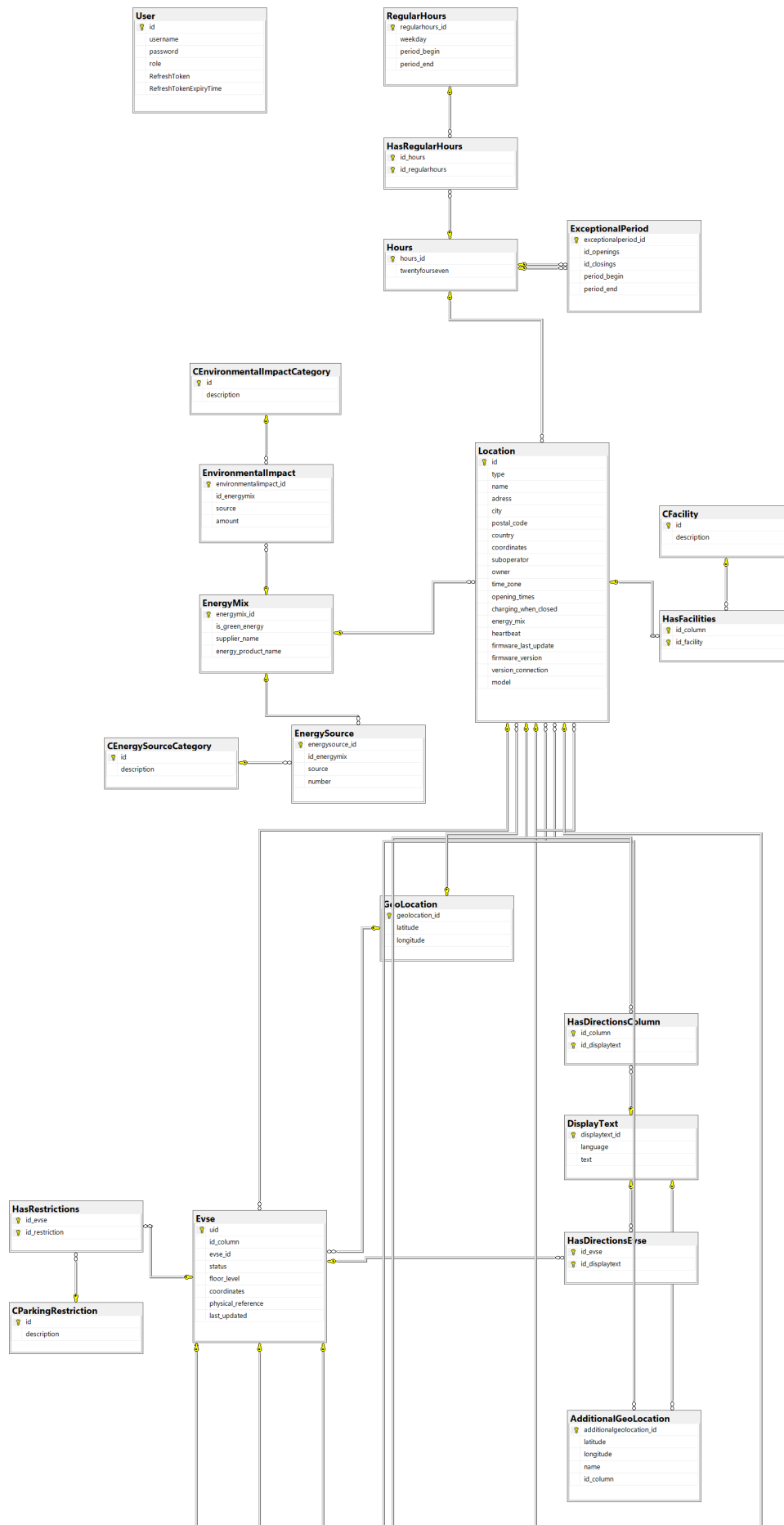
Un'altra differenza rispetto al protocollo è data dalla tabella "RegularHours" dove, per evitare gli stetti dati reiterati (es: colonnina che apre alle 9:00 e chiude alle 23:00) è stata implementata come relazione molti a molti.

Per una corretta distinzione tra identificativi di chiave primaria e identificativi di chiave esterna è stato deciso di adottare il seguente protocollo: La chiave primaria è denominata come "nometabella" _id mentre la chiave esterna della tabella "altratabella" è denominata come id_"altratabella" eccenzion fatta per:

- Le tabelle che indicano un enum, dove l'identificativo è denominato con solo "id";
- Le tabelle dove già nel protocollo veniva definito un identificatore univoco;
- Le tabelle dove ci sono più chiavi esterne della stessa tabella (per esempio nella tabella "ExceptionalPeriod" si hanno due chiavi esterne della tabella "Hours" una però si relaziona con la riga che identifica l'orario di apertura, l'altra si relaziona con la riga che identifica l'orario di chiusura).

Col senno di poi la scelta di impostare il database seguendo quasi scrupolosamente la struttura indicata nel protocollo OCPI 2.1 potrebbe rilevarsi svantaggiosa in futuro: In primo luogo poiché, essendo la struttura così dipendente e legata dalla versione 2.1, in un futuro aggiornamento del protocollo la

struttura proposta potrebbe risultare difficile da adattare alle nuove specifiche richieste; In secondo luogo non vi è stato un processo di astrazione il quale avrebbe sicuramente potuto rendere meno complicata la gestione del sistema e la manutenzione nel tempo. Allo stesso tempo però seguire la struttura indicata nel protocollo garantisce una maggiore coerenza e uniformità nell'organizzazione dei dati, rendendo più facile l'interoperabilità con altri sistemi che utilizzano lo stesso protocollo.



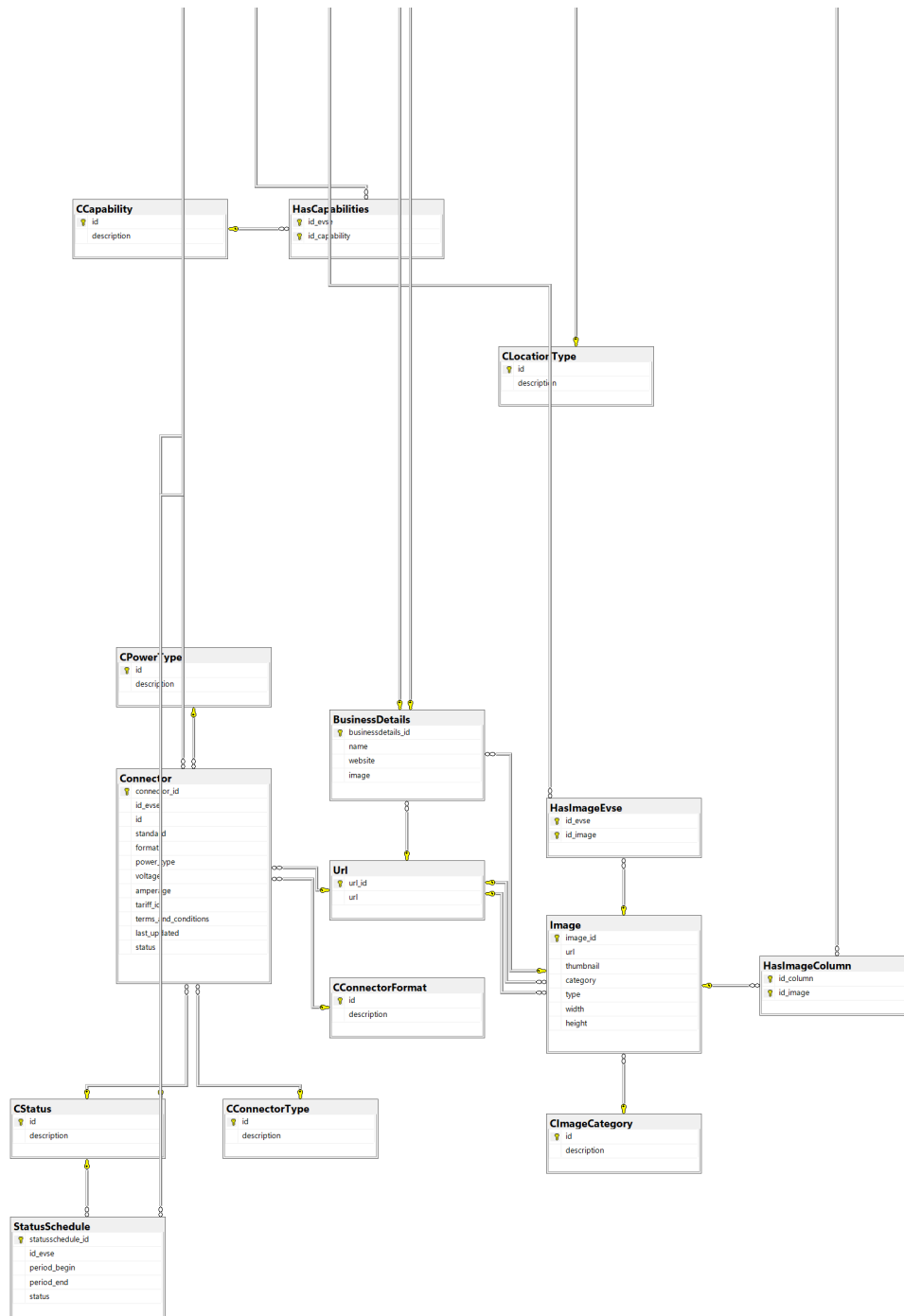


Figura 3.2: Schema ER del Database SQL

Capitolo 4

Backend

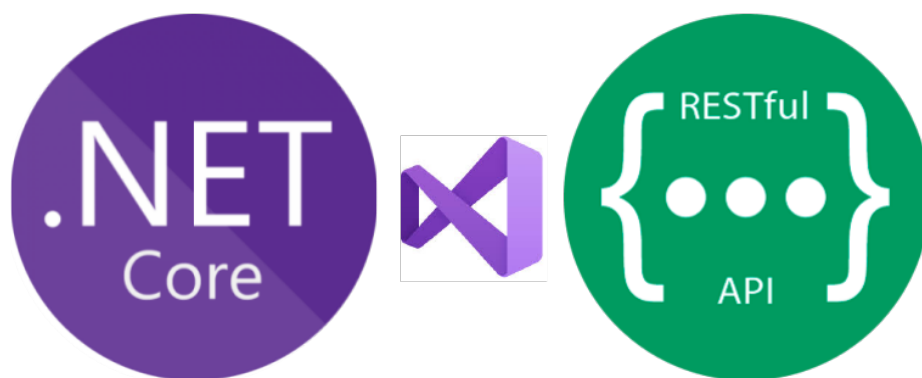


Figura 4.1: DotNET e API REST

4.1 Set Up

Come scritto in precedenza, mi è stato consigliato dall'azienda l'utilizzo di .NET come framework di sviluppo per il Backend del progetto, in questa prima sezione voglio spiegare cosa sia .NET e come ho impostato il backend.

[4] .NET è un framework di sviluppo software sviluppato da Microsoft che supporta molti linguaggi di programmazione. Viene utilizzato per sviluppare una vasta gamma di applicazioni, tra cui desktop, mobile, web e cloud. DotNet è strettamente integrato con le tecnologie Microsoft e offre una potente piattaforma di sviluppo che consente di creare API web efficienti e scalabili. Inoltre ha una vasta comunità di sviluppatori che contribuiscono alla creazione di strumenti, librerie e documentazione, il che rende più semplice per gli sviluppatori ottenere aiuto e trovare soluzioni ai problemi.

Come prima cosa ho creato un nuovo progetto API Web ASP.NET CORE, utilizzando Visual Studio, il quale inizializza molteplici file e dipendenze creando fin da subito un ambiente di sviluppo funzionante installando anche swagger, una suite di strumenti per testare e documentare le API REST. La solution di visual studio che implementa il Backend del progetto di tesi è a sua volta divisa in altri tre sottoprogetti indipendenti uno dall'altro:

- un progetto che si occupa del servizio di hash delle password chiamato "HashPasswordService";
- un progetto che si occupa del collegamento con il database e dell'incapsulamento dei suoi dati tramite Entity Framework, fornendo una interfaccia con operazioni utilizzabili chiamato "ServerDB";
- un progetto che implementa le API e che si interfaccia con il Frontend chiamato "ServerAPI".

4.2 HashPasswordService

Utilizzare funzioni di hash per archiviare le password in un database è ormai una caratteristica imprescindibile di ogni database poiché le funzioni di hash sono una forma unidirezionale di crittografia che trasformano una stringa in chiaro (La password effettiva) in una differente stringa di lunghezza fissa (chiamata "hash"). Questo rende, in caso di compromissione del database, difficile per gli aggressori ottenere le password originali poiché non è possibile invertire la funzione di hash. Il progetto "HashPasswordService" è formato da una sola classe chiamata "HashPassword": La classe "HashPassword" è una classe che fornisce funzionalità per creare e verificare password hashate:

- La funzione "HashPasswordV3" riceve in input una password in chiaro e la restituisce in formato hash utilizzando l'algoritmo "PBKDF2" con "HMACSHA512" come funzione di derivazione della chiave e "RandomNumberGenerator" come generatore di numeri casuali, utilizza la funzione privata "HashPasswordV3";
- La funzione "VerifyHashedPasswordV3" riceve in input una password hashata e una password in chiaro, e restituisce un valore booleano che indica se la password in chiaro corrisponde a quella hashata. La verifica viene effettuata tramite l'algoritmo "PBKDF2";
- Le funzioni private "WriteNetworkByteOrder" e "ReadNetworkByteOrder" scrivono e leggono rispettivamente un intero in formato di rete (big-endian);
- La funzione privata "HashPasswordV3" esegue l'effettiva creazione della password hashata utilizzando l'algoritmo "PBKDF2".

PBKDF2 (Password-Based Key Derivation Function 2) è un algoritmo di hash, il quale crea una "chiave derivata" a partire dalla password in input e dall'utilizzo di un "sale" (un valore univoco associato all'utente che viene utilizzato per rendere più complesso il processo di cracking). L'algoritmo esegue un elevato numero di iterazioni sulla password e sul sale (nel mio caso 1000), rendendo difficile per gli attaccanti decifrare la password originale anche se hanno accesso all'hash memorizzato nel database. PBKDF2 è preferibile rispetto ad altri algoritmi di hash come MD5 o SHA-1 perché è progettato per essere lento e resistente agli attacchi di forza bruta, il che rende più difficile per un possibile attaccante decifrare le password.

4.3 ServerDB

Il mapping di oggetti, o object-relational mapping (ORM), è una tecnica utilizzata in programmazione per gestire l'accesso ai dati tra il codice dell'applicazione e un database relazionale. L'ORM consente di separare la logica dell'applicazione dalla gestione dei dati, fornendo una maggiore astrazione dei dati, consentendo di manipolare oggetti e collezioni di oggetti piuttosto che operare direttamente sulle tabelle del database. Ciò semplifica la gestione dei dati, rende il codice più leggibile e facilmente mantenibile, migliorando le prestazioni dell'applicazione. Ho scelto di utilizzare Entity Framework come mapper per il database SQL sia per i motivi sopracitati che per la mia precedente esperienza con questo strumento, acquisita durante il corso di tecniche avanzate di programmazione.

Esistono due tipologie di approcci per l'impostazione di un progetto con database: "Code First" e "Database First".

Nell'approccio "Code First", si inizia scrivendo il codice dell'applicazione e poi si crea il database a partire da esso. Con questo approccio, si utilizzano le annotazioni o i mapping del codice per definire le tabelle, i campi e le relazioni del database. Questo approccio è molto utile quando si vuole avere il controllo completo sulla struttura del database e su come viene rappresentato il codice.

L'approccio "Database First" invece inizia con la creazione del database e quindi si genera il codice a partire da esso. Questo approccio è utile quando si ha già un database esistente o quando si vuole utilizzare un database di terze parti per la propria applicazione. Con questo approccio, si possono generare automaticamente le classi del codice a partire dalle tabelle del database, che possono essere personalizzate in seguito.

Avendo già creato il database del progetto ho optato per l'approccio "Database First", utilizzando uno script di Code-based Migration e collegando il database salvato in SQL Server utilizzando la `ConnectionString`:

Per prima cosa ho aperto la console di gestione pacchetti nuget e ho abilitato la migration con il comando "enable-migrations" successivamente ho aggiunto la migration utilizzando il comando "Add-Migration" e infine ho creato il mapping del database usando il comando "update-database -verbose". Questo ha creato un file per ogni tabella del database con all'interno la classe entity framework corrispettiva, con tutte le proprietà, tuttavia ho ritenuto utile eseguire alcune modifiche:

- inserire alcuni limiti di dominio, come da specifica del protocollo
- creare degli enum logici per gestirli con il linguaggio di programmazione e il suo compilatore, non esistendo gli enum su SQL, nel database sono salvati come byte (quindi un numero) pertanto ogni volta che utilizzo questi dati per un controllo maggiore ho necessità di fare un cast a enum, mentre per eseguire modifiche e salvarle sul database eseguirò un cast a byte.
- creare una interfaccia con tutte le dichiarazioni di metodi da usare poi con le API (dependency injection) che poi verrà implementata dal contesto SQL dove verranno implementati i metodi utilizzando EF e SQL, e verrà successivamente usata dal progetto ServerAPI.

La classe `CPDataContextSQL`, che implementa l'interfaccia generale del contesto ed estende la classe contesto di Entity Framework, è la classe più importante del sottoprogetto. In questa classe vengono effettivamente implementate le query che poi verranno usate per interrogare il database. Per la realizzazione delle query ho utilizzato la funzionalità LINQ:

LINQ (Language Integrated Query) ed è una funzionalità di programmazione presente in diversi linguaggi di programmazione, tra cui appunto C#. LINQ consente ai programmatori di scrivere query utilizzando un'unica sintassi unificata per più raccolte di oggetti differenti. In Entity Framework, LINQ viene utilizzato per scrivere query che recuperano dati dal database. Queste query utilizzano la sintassi LINQ per definire le operazioni di selezione, proiezione, ordinamento, filtro e aggregazione dei dati.

I metodi `Get` di questa classe sono abbastanza autoesplicativi: recuperano dal database i dati di una determinata tabella salvandoli in una classe dell'entity Framework creata in precedenza. Alcuni di questi metodi, in genere quelli che restituiscono un oggetto senza navigation property importanti, sono semplici e utilizzano una sola query; Altri metodi invece possono essere molto più complessi e richiamare all'interno altri metodi della classe. Facciamo un esempio per un metodo `Get` semplice:

```
public User? getUser(string username) {  
    return Users.FirstOrDefault(u => u.UserName == username);  
}
```

Il metodo di cui sopra è utilizzato per verificare la presenza di un utente all'interno del database, basandosi sul suo username. Nel caso in cui venga trovata una corrispondenza, verrà restituito un oggetto Entity Framework associato all'utente, consentendo di accedere ai dati richiesti. Ed ecco un esempio di un metodo più complesso:

```
public BusinessDetail getBusinessDetailsFull(int? id) {  
    if (id == null) return null;  
    var businesAux = BusinessDetails.Where(b => b.businessdetails_id == id)  
        .ToList();  
    foreach (BusinessDetail bu in businesAux) {  
        bu.imageNavigation = getImageFull(bu.image);  
        bu.websiteNavigation = Urls.FirstOrDefault(u => u.url_id == bu.website);  
    }  
    return businesAux.First();  
}
```

Il metodo di cui sopra è utilizzato per recuperare tutti i dati di un determinato fornitore/distributore (Business) dato il suo identificativo. Dato che i dati di un Business sono distribuiti su più tabelle nel database è necessario completare i dati di un oggetto riempiendo le sue navigation property in modo adeguato richiamando altri metodi `get` e utilizzando altre query.

Il prossimo capitolo si concentrerà sul `ServerAPI`, che ci permetterà di esporre le funzionalità del nostro database tramite chiamate API e di creare un'interfaccia di comunicazione tra il front-end e

il back-end della nostra applicazione. In questo modo, potremo accedere e gestire i dati del nostro database in modo efficiente, sicuro e controllato.

4.4 ServerAPI

ServerAPI è il terzo e ultimo progetto della solution del backend e possiamo dividerlo in 3 parti:

- La parte di Set Up, e di impostazione del progetto
- I controllers dove sono implementate le API
- i modelli e le classi utilizzate dalle API

4.4.1 Impostazioni del progetto - CORS

il file "Startup.cs" contiene informazioni sulle varie impostazioni e dei servizi utilizzati dalle API e dalla solution in generale: per utilizzare il progetto in una prima fase ho impostato CORS poiché avevo bisogno di utilizzare "Access-Control-Allow-Origin".

"Access-Control-Allow-Origin" è un header HTTP che viene utilizzato per specificare gli indirizzi che sono autorizzati a effettuare richieste cross-origin. In altre parole, questo header viene utilizzato per indicare che un'origine (come un'applicazione web) è autorizzata ad accedere ai contenuti di un altro dominio. È necessario utilizzare "Access-Control-Allow-Origin" quando si effettuano richieste tra domini diversi, ad esempio, come nel mio caso, quando si effettua una richiesta da un'applicazione client-side a un API back-end su un dominio diverso. Senza questo header, il browser impedirà la richiesta in quanto potrebbe rappresentare una minaccia per la sicurezza, poiché potrebbe rappresentare un tentativo di accedere a informazioni riservate o sensibili.

4.4.2 Impostazioni del progetto - Dependency Injection

Ho poi aggiunto un extension method per poter iniettare nella interfaccia offerta dal "ServerDB" un'implementazione diversa in base al tipo di database utilizzato (Nel mio caso inietto l'unica implementazione scritta da me ovvero quella per un database sql) e quindi utilizzare la Dependency Injection.

```
namespace ServerAPI {  
    0 riferimenti | Luigi Timossi, 71 giorni fa | 1 autore, 2 modifiche  
    public static class SelectorDBClass {  
        1 riferimento | Luigi Timossi, 71 giorni fa | 1 autore, 2 modifiche  
        public static void SelectorDB(this IServiceCollection service, IConfiguration Configuration) {  
            service.AddScoped<ICPDataContext>(provider => provider.GetService<CPDataContextSQL>());  
            service.AddDbContext<CPDataContextSQL>(options => options.UseSqlServer(Configuration.GetConnectionString("CPData")));  
        }  
    }  
}
```

Figura 4.2: Extension method utilizzata per Dependency injection

La Dependency Injection (DI) è un pattern di progettazione che consiste nell'iniettare dipendenze esterne all'interno di un componente, in modo che questo possa funzionare correttamente. In prati-

ca, DI consiste nel fornire a un componente tutto ciò di cui ha bisogno per svolgere il suo compito, tramite l'iniezione di dipendenze esterne come servizi, helper o altri componenti. Per poter utilizzare la DI bisogna installare un framework DI come Microsoft.Extensions.DependencyInjection, in questo framework le dipendenze vengono registrate all'interno di un contenitore di dipendenze, che è responsabile di creare e fornire le istanze richieste ai componenti che ne hanno bisogno. L'utilizzo della DI ha diversi vantaggi:

1. Migliore separazione delle responsabilità: la DI aiuta a separare le responsabilità dei componenti, in modo che ognuno di essi sia responsabile solo di una parte specifica del sistema.
2. Maggiore modularità e testabilità: i componenti che utilizzano la DI sono più facili da testare, poiché le dipendenze esterne possono essere sostituite con mock durante i test.
3. Maggiore flessibilità e manutenibilità: la DI rende più facile la modifica e l'aggiornamento delle dipendenze esterne, poiché queste possono essere gestite in modo centralizzato.

In questo caso la DI viene impostata per poter utilizzare diverse tipologie di database senza dover cambiare il codice, dato che il codice si basa su una interfaccia generale.

4.4.3 Impostazioni del progetto - Autorizzazione e JWT

Per il processo di autorizzazione e il controllo dei permessi di un determinato utente, ho scelto di utilizzare JSON Web Token: Uno standard aperto (RFC 7519) per rappresentare in modo sicuro le informazioni usando stringhe JSON. L'utilizzo di JWT per l'autenticazione delle richieste API Web è un'alternativa alle sessioni basate sui cookies. Se si utilizza un JWT l'autenticazione può essere effettuata in modo stateless, rendendo le API più scalabili e riducendo il carico del server.

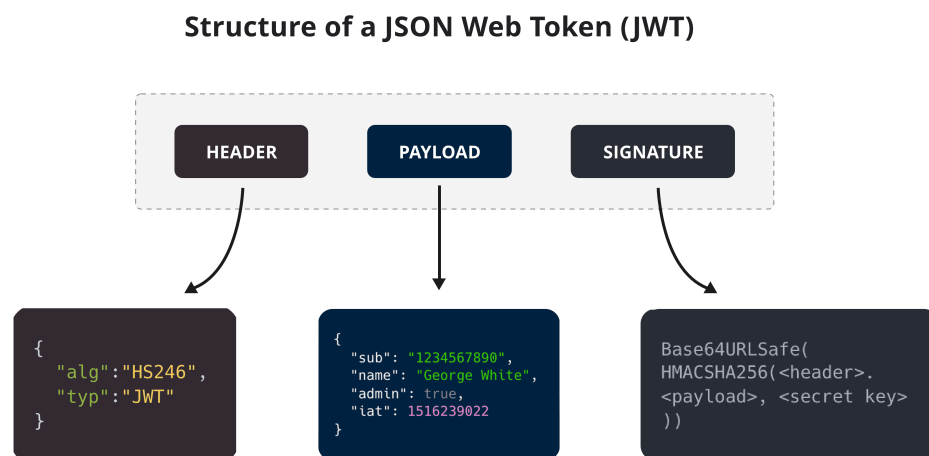


Figura 4.3: Struttura di un JWT

Un JSON Web Token è costituito da tre parti principali: una intestazione (header), un payload (carico utile) e una firma (signature). L'intestazione identifica l'algoritmo di crittografia utilizzato per generare la firma, il payload contiene le informazioni che vengono trasportate e la firma viene utilizzata per verificarne l'autenticità.

Il processo di creazione di un JSON Web Token è il seguente:

1. Il mittente (chiamato "emittente" o "issuer") crea un payload contenente le informazioni che desidera condividere con la parte destinataria.
2. L'emittente codifica il payload in una stringa JSON.
3. L'emittente crea un'intestazione che contiene l'algoritmo di crittografia utilizzato per generare la firma.
4. L'emittente codifica l'intestazione in una stringa JSON.
5. L'emittente utilizza l'algoritmo di crittografia specificato nell'intestazione per generare una firma per il token.
6. L'emittente concatena la stringa codificata dell'intestazione, la stringa codificata del payload e la firma per creare il token.
7. La parte destinataria può quindi utilizzare la firma per verificare l'autenticità del token e accedere alle informazioni nel payload.

I JSON Web Token sono spesso utilizzati per l'autenticazione e l'autorizzazione nelle applicazioni web e mobili. Come nel nostro caso, il server di autenticazione crea un token JWT contenente le informazioni dell'utente autenticato e lo invia al client. Il client può quindi inviare il token JWT al server per ogni richiesta successiva, consentendo al server di verificare rapidamente l'autenticità dell'utente e fornire l'accesso alle risorse protette.

Nel nostro caso il JWT è stato implementato seguendo lo standard, ovvero creando due Token: uno di accesso e l'altro di refresh. Il primo token viene generato dal Backend in fase di Login se e solo se il Login ha avuto successo, insieme ad esso viene generato un refresh token e vengono inviati entrambi come risposta alla richiesta di Login.

Nel payload del token di accesso vengono archiviate alcune informazioni:

1. username dell'utente
2. ruolo dell'utente
3. data di scadenza del token

Ogni qualvolta un utente effettuerà una chiamata API (tranne Login e Registrazione), dovrà inserire nell'header della richiesta il JWT ricevuto in fase di Login:

- Nel caso non venga inserito il JWT o non venga inserito correttamente il Backend risponderà con un messaggio di Errore 403.
- Se in una richiesta viene inviato un JWT valido e non scaduto ma il livello di permessi del JWT non soddisfa quelli richiesti dalla chiamata API, in questo caso il Backend risponderà con un messaggio di Errore 401.

Per controllare al meglio i permessi di un JWT, sono stati installati nel progetto due pacchetti NuGet: `Microsoft.AspNetCore.Authentication` e `Microsoft.AspNetCore.Authentication.JwtBearer`. Il primo fornisce funzionalità di autenticazione per le applicazioni ASP.NET Core, inclusi i middleware necessari per configurare l'autenticazione dell'applicazione, il secondo consente ad un'applicazione ASP.NET Core di verificare l'autenticità dei JWT ricevuti dai client e di autorizzare gli utenti in base alle informazioni contenute nel token. Il controllo dei permessi viene effettuato a livello di codice inserendo un attributo prima della funzione API:

inserendo `[Authorize]` si controlla se il JWT sia valido e non scaduto;

inserendo `[Authorize(Roles="ruoli")]` si controlla se il JWT sia valido, non scaduto e che abbia come attributo Roles almeno una stringa corrispondente tra quelle indicate.

Il refresh token rimane valido per molto più tempo rispetto al token principale e viene utilizzato per evitare di imporre il Login dell'utente ogni qualvolta il token principale scade: Quando il token principale scade, il Frontend invia una richiesta HTTP all'API che elabora la richiesta di "refresh" inserendo nell'header il refresh token. Se il refresh token è valido e non scaduto il Backend genererà un nuovo token e un nuovo refresh token da inviare al Frontend, se non è valido o scaduto risponderà con un errore, in quel caso l'utente sarà costretto ad effettuare nuovamente il Login per ricevere nuovi token validi.

4.4.4 Controllers API

Un Controller API è un componente software che gestisce le richieste di servizio da parte dei client, elaborandole e fornendo risposte in formato JSON o XML. Il controller si occupa anche della validazione delle richieste in ingresso, dell'autenticazione e autorizzazione degli utenti e della gestione degli errori. Il progetto "ServerAPI" contiene tre diversi Controller che differiscono tra loro in base alle funzioni che offrono:

- `TokenController` che contiene le varie operazioni che servono per gestire il refresh o il revoke dei JWT;
- `UserController` che contiene le varie operazioni che si occupano della gestione degli User nel database e le varie operazioni di registrazione e login
- `ChargePointController` che contiene le varie operazioni che si interfacciano con la parte di database che si occupa di archiviare i dati riguardanti i ChargePoint;

TokenController

La classe `TokenController` rappresenta un controller di una API REST che serve per la gestione e la creazione di un refresh token, quindi verrà usata nel caso sia scaduto il JWT al client. Il controller è formato da due metodi:

1. Il primo metodo pubblico presente nella classe è `RefreshAsync`. Questo metodo viene utilizzato per gestire la richiesta di refresh token, ovvero per ottenere un nuovo token di accesso a partire da un token di accesso scaduto e un refresh token valido. Il metodo prende come argomento un oggetto di tipo `TokenApiModel` che rappresenta la richiesta effettuata dal client.

Il metodo controlla innanzitutto se il parametro `tokenApiModel` è nullo; in tal caso, viene restituito un errore di tipo "Invalid client request". Successivamente, il metodo estrae il token di accesso e il refresh token dalla richiesta e utilizza il servizio `_tokenService` per ottenere l'utente associato al token di accesso scaduto.

Se l'utente non viene trovato, se il refresh token non è valido o se il refresh token è scaduto, il metodo restituisce un errore di tipo "Invalid client request". In caso contrario, il metodo genera un nuovo token di accesso e un nuovo refresh token e aggiorna il token di refresh dell'utente nel database tramite il servizio `_userService`. Infine, il metodo restituisce un oggetto JSON contenente la durata di validità del nuovo token di accesso, il nuovo token di accesso e il nuovo refresh token.

2. Il secondo metodo pubblico presente nella classe è `RevokeAsync`. Questo metodo viene utilizzato per revocare il token di accesso e il refresh token di un utente autenticato.

Il metodo controlla innanzitutto se l'utente autenticato esiste nel database; in tal caso, viene aggiornato il token di refresh dell'utente con i valori nulli. Infine, il metodo restituisce una risposta HTTP con lo status code 204 `NoContent()`, che indica che la richiesta è stata elaborata con successo ma non è necessario restituire alcun contenuto al client.

UserController

La classe `UserController` rappresenta un controller di una API REST in cui gli utenti possono registrarsi, accedere e modificare il proprio ruolo. Il controller è formato da tre metodi:

1. Il primo metodo pubblico della classe è `ChangeRole`. Questo metodo permette di modificare il ruolo di un utente nel database. Questo metodo è disponibile solo per gli utenti autenticati. Questo metodo è stato usato solo in fase di test e non è utilizzabile direttamente dal Frontend.
2. Il secondo metodo pubblico della classe è `Register`. Questo metodo permette agli utenti di registrarsi all'applicazione. Questo metodo prende come parametro il nome utente e la password, che vengono salvati nel database utilizzando il metodo `createUser` di `ICPDataContext`.
3. Il terzo metodo pubblico è il metodo usato per il Login. Questo metodo consente agli utenti di effettuare l'accesso all'applicazione. Questo metodo prende come parametri il nome utente e la

password. Se il nome utente non esiste nel database, viene restituito un errore 401. Se la password inserita non corrisponde a quella nel database (controllando se gli hash corrispondono dato che la password nel database non è salvata in chiaro), viene restituito un errore 401. Altrimenti, viene generato un token di accesso con all'interno alcuni dati dell'utente come il suo ruolo e un token di refresh per l'utente. Per generare i token di accesso e di refresh, il metodo Login chiama il metodo `GenerateAccessToken` e `GenerateRefreshToken` di `ITokenService`. Inoltre, il metodo utilizza il metodo `UpdateToken` di `ICPDataContext` per aggiornare il token di refresh dell'utente nel database. All'utente infine vengono inviati i due token generati, la data di scadenza del token di accesso, il ruolo e l'username dell'utente.

ChargePointController

Questa classe rappresenta un controller ASP.NET Core che gestisce le richieste per le risorse relative ai charge point, ovvero i punti di ricarica per veicoli elettrici. questo Controller è il fulcro del progetto ed è il controller da cui escono ed entrano la maggior parte di richieste HTTP dell'intero progetto. Il controller è formato da cinque metodi:

1. Il primo metodo pubblico della classe è chiamato `GetAllLocations()`. Questo metodo viene utilizzato per ottenere tutte le informazioni dei charge point presenti nel sistema. Il metodo restituisce una lista di oggetti di tipo `Location` (classe EF del serverDB), Questo metodo è stato usato solo in fase di test e non è raggiungibile direttamente dal Frontend.
2. Il secondo metodo pubblico della classe è chiamato `GetBusiness()`. Questo metodo viene utilizzato per ottenere una lista di tutti i nomi dei fornitori/distributori (`Business`) salvati nel database. Il metodo è accessibile solo dagli utenti con ruolo "admin" e restituisce una lista di stringhe. Questo metodo viene utilizzato dal Frontend in una pagina accessibile solo dagli admin poiché nella pagina è possibile aggiungere o modificare i vari attributi e impostazioni di una charge point.
3. Il terzo metodo pubblico della classe è chiamato `GetColonnine()`. Questo metodo è utilizzato per ottenere tutte le informazioni non private sui charge point presenti nel database. Il metodo è accessibile solo dagli utenti autenticati e restituisce una lista di oggetti di tipo `RLocation`, che sono stati creati a partire dagli oggetti di tipo `Location` ottenuti dal metodo `GetAllLocations()` di `ICPDataContext`. La classe `RLocation` (`RespondLocation`) rappresenta la versione dell'oggetto `Location` che il Frontend si aspetta di ricevere con alcuni nomi di proprietà cambiati e oggetti più semplificati e astratti.
4. Il quarto metodo pubblico della classe è chiamato `GetColonnineAdmin()`. Questo metodo è utilizzato solo dagli utenti con ruolo "admin" per ottenere tutte le informazioni sui charge point presenti nel database. Il metodo restituisce una lista di oggetti di tipo `RLocationAdmin`, che sono stati creati a partire dagli oggetti di tipo `Location` ottenuti dal metodo `GetAllLocations()` di

ICPDataContext. La classe RLocationAdmin rappresenta una versione più completa dell'oggetto RLocation e comprende anche le informazioni private dei vari charge point.

5. Il quinto e ultimo metodo pubblico della classe è chiamato PushColonnina(). Questo è il metodo utilizzato per creare o aggiornare i charge point nel database. Questa azione è accessibile solo agli utenti con il ruolo "admin" poiché richiede il potere di modificare il database. La richiesta HTTP contiene un oggetto JSON, che rappresenta i dati della colonnina di ricarica inviati dal Frontend, che viene poi deserializzato automaticamente in un oggetto ALocation (AnswerLocation) per essere utilizzato dal backend. Il metodo inizia verificando se esiste già una charge point corrispondente nel database. Se esiste, il metodo si occuperà di aggiornare i dati dell'entità esistente, altrimenti creerà un nuovo charge point con i dati in input. Successivamente, l'azione verifica se esistono già gli oggetti Evse ricevuti in input nel database. In caso affermativo, gli oggetti verranno aggiornati con i dati forniti. In caso contrario, verrà creato un nuovo oggetto. Vengono quindi creati o aggiornati gli oggetti Connector e le relazioni HasCapabilities, HasParkingRestrictions e HasTermsConditions corrispondenti per ogni charge point. Infine, il metodo aggiorna i dati rimanenti come il fuso orario, gli orari di apertura, i mix di energia, i dettagli tecnici e i dettagli dei vari business.

4.4.5 Mapping usato

Come scritto nella sezione precedente, ho scelto di usare, oltre a Entity Framework per il database, altri due classi di mapping: ALocation e RLocation (e tutte le altre classi ad esse collegate). Questi due modelli si distinguono facilmente dal prefisso del loro nome, che significa in un caso "Domanda" e nell'altro caso "Risposta". Quindi sono entrambi oggetti popolati da dati di charge point ma usati in momenti differenti della comunicazione tra Frontend e Backend:

- ALocation rappresenta i dati dei charge point inviati insieme alla "Domanda" ovvero ad una richiesta HTTP inviata dal Frontend, quindi rappresentano sia l'oggetto JSON presente nel body del messaggio, sia il modo con cui il Frontend salva in memoria i dati dei vari charge point.
- RLocation rappresenta i dati dei charge point inviati insieme alla "Risposta" ovvero alla risposta HTTP inviata dal Backend al Frontend, quindi rappresentano l'oggetto JSON che il Frontend si aspetta di ricevere, un oggetto più semplice ed astratto rispetto all'oggetto Entity Framework che rappresenta il database.

4.5 Swagger per il Testing delle API

Swagger è uno strumento molto utile per la documentazione e il testing delle API REST, rende più semplice il testing anche grazie ad una UI semplice ed intuitiva. Una volta avviato il progetto con Visual Studio, si aprirà una pagina web all'indirizzo <https://localhost:44340/swagger/index.html>. A questo indirizzo è stata generata una pagina HTML di Swagger UI dove poter testare direttamente dal browser le varie API sviluppate.

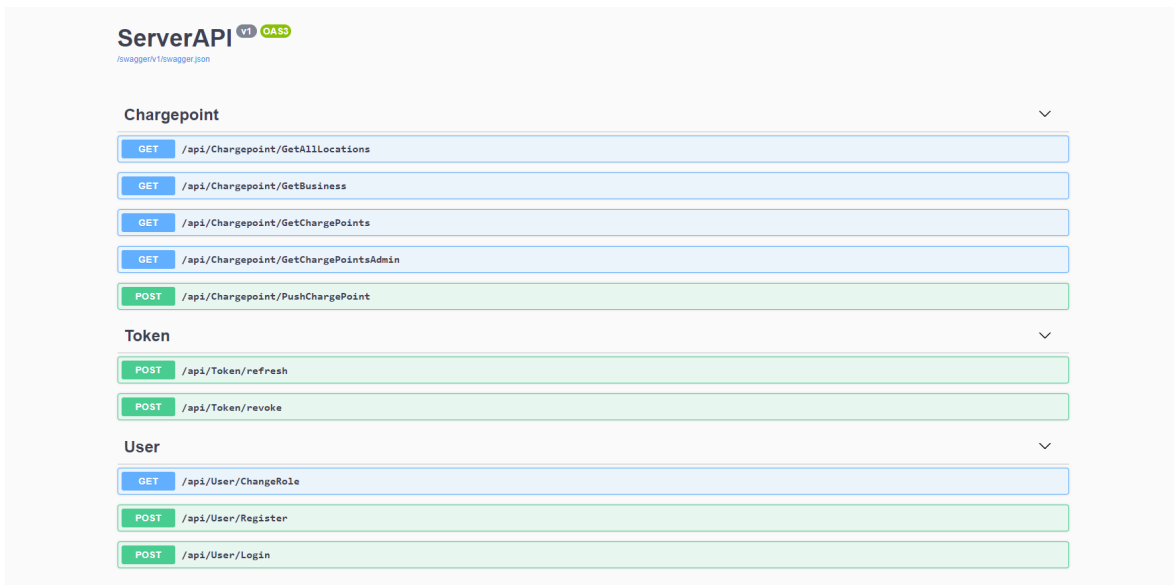


Figura 4.4: Pagina principale di Swagger

Come osservabile in figura 4.4 le API vengono etichettate come operazioni POST o GET in base a quanto definito nel codice, inoltre vengono anche suddivise in Set corrispondenti ai vari Controller creati. Per testare una API sarà semplicemente necessario cliccare l'operation block di riferimento con il nome della API da testare. A questo punto il blocco si aprirà rivelando i vari dati da inserire in input. Per questa demo proveremo l'API di Login

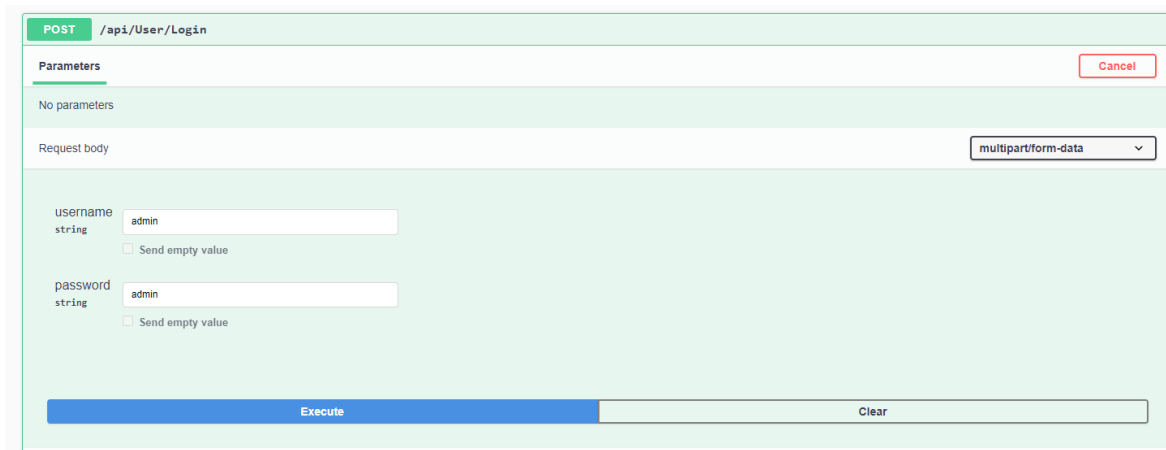


Figura 4.5: Test API di Login

In figura 4.5 Possiamo vedere in che modo possiamo interfacciarci con la nostra API. In questo caso abbiamo da riempire (non obbligatoriamente se per caso vogliamo testare come si potrebbe comportare la API con argomenti mancanti) due variabili: una di nome "username" mentre l'altra di nome "password" entrambe stringhe. Una volta riempiti i campi di input si può cliccare il pulsante di esecuzione della chiamata API.

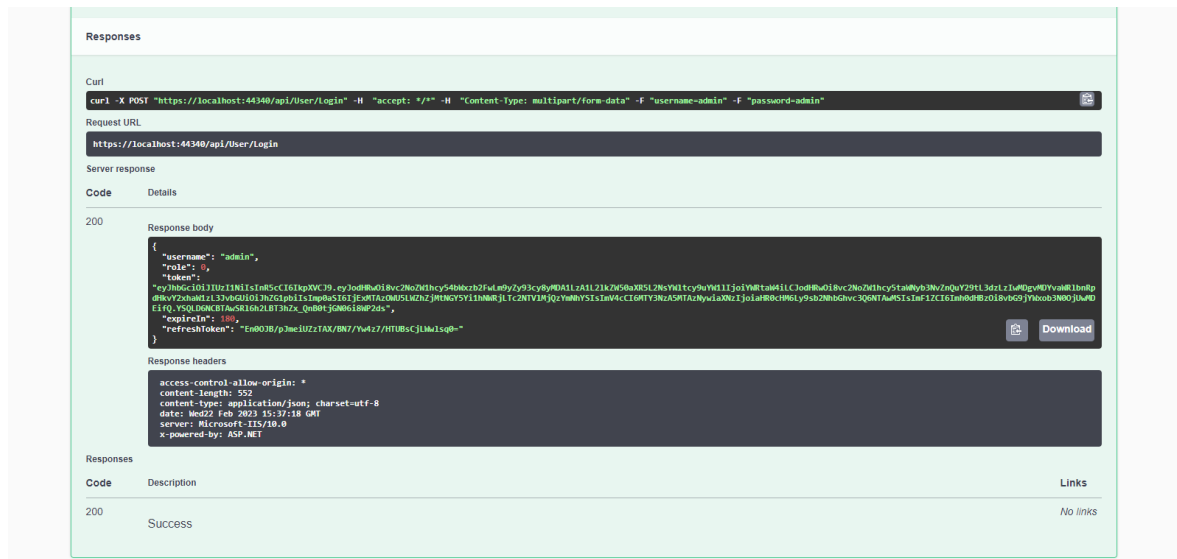


Figura 4.6: Test API di Login - Risposta

Il Test delle API avviene attraverso operazioni CURL, quindi vengono effettivamente eseguite richieste HTTP. Dopo aver inviato la richiesta, possiamo analizzare la stringa CURL inviata, l'indirizzo web dove è stata inviata la richiesta HTTP e una volta ricevuta, la risposta HTTP restituita dall'API. In questo caso possiamo vedere che abbiamo ricevuto una risposta con un codice di successo: 200 OK. dato che la richiesta è andata a buon fine, nel body è presente un oggetto JSON con i dati richiesti che serviranno al Frontend per autenticare ogni prossima richiesta HTTP.

In questo progetto ho scelto di non documentare le API in modo troppo preciso, dato che non era il principale scopo del progetto, tuttavia Swagger è un ottimo strumento anche per la documentazione delle API ed è importante farlo. Documentare le API semplifica la collaborazione e la manutenzione, mostra lo scopo, i tipi di ritorno, i messaggi di errore e semplifica il testing.

Capitolo 5

Frontend



Figura 5.1: Angular

5.1 Angular

Come anticipato in precedenza, per la parte Frontend del progetto mi è stato consigliato dall'azienda l'utilizzo di Angular.

[5] Angular è un framework JavaScript open-source sviluppato da Google per creare applicazioni web a singola pagina (SPA) complesse e dinamiche. Con Angular, gli sviluppatori possono utilizzare il markup HTML per definire la struttura dell'interfaccia utente (UI) dell'applicazione, utilizzare il linguaggio di programmazione TypeScript per definire il comportamento e l'interattività dell'UI, e utilizzare Angular CLI (Command Line Interface, basata sulla CLI di Node.js) per generare, testare e distribuire l'applicazione.

Angular utilizza il pattern architetturale Model-View-Controller (MVC) o Model-View-ViewModel (MVVM) per separare la logica di business dall'interfaccia utente, fornendo così una struttura scalabile e dalla facile manutenzione per le applicazioni web complesse. Angular fornisce anche una vasta gamma

di funzionalità come la gestione degli eventi, l'iniezione di dipendenze, la validazione dei moduli, il routing, la gestione dello stato dell'applicazione, la gestione delle animazioni, e molte altre.

Per quanto riguarda l'interfacciamento dei vari componenti, Angular utilizza il meccanismo di binding dei dati per collegare i dati tra i componenti. I componenti padre passano i dati ai componenti figli attraverso gli input, mentre i componenti figli comunicano con i componenti padre attraverso gli eventi. In questo modo, i dati e gli eventi possono essere scambiati tra i componenti in modo efficiente e coerente.

Con Angular si possono definire anche dei servizi. I servizi sono utilizzati per fornire funzionalità condivise tra più componenti dell'applicazione, contengono la logica dell'applicazione e possono essere utilizzati per accedere ai dati da un server remoto o per fornire funzionalità come la gestione dell'autenticazione o la gestione delle notifiche. I servizi possono essere iniettati nei componenti tramite il meccanismo di dependency injection di Angular.

Come scritto, la parte logica e comportamentale di Angular si basa su TypeScript. TypeScript è un linguaggio di programmazione open-source sviluppato da Microsoft e basato su JavaScript. È stato creato per fornire agli sviluppatori uno strumento per scrivere codice JavaScript di alta qualità e facilmente manutenibile, con l'aggiunta di caratteristiche tipiche dei linguaggi di programmazione orientati agli oggetti come il tipaggio statico e l'ereditarietà delle classi.

5.1.1 PrimeNG



Figura 5.2: PrimeNG

Per facilitare lo sviluppo delle varie pagine mi è stato consigliato l'utilizzo della libreria PrimeNG, [6] una libreria di componenti UI open-source per Angular, sviluppata da PrimeTek Informatics. La libreria PrimeNG fornisce una vasta gamma di componenti UI predefiniti, come pulsanti, campi di input, tabelle, grafici, menu, modali e molti altri. Questi componenti sono progettati per essere facili da usare e facilmente personalizzabili, permettendo agli sviluppatori di creare interfacce utente professionali e accattivanti per le loro applicazioni web.

PrimeNG è basato su Angular e utilizza le sue funzionalità principali come il binding dei dati, le direttive e gli eventi. La libreria PrimeNG include anche un'ampia documentazione, un supporto per temi personalizzati e una vasta comunità di sviluppatori che forniscono supporto e feedback.

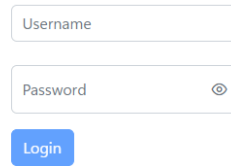
5.2 Login

Angular ha una struttura a moduli, dove ogni modulo può avere un modulo padre, uno o più moduli figli e uno o più moduli fratelli. L'albero risultante viene chiamato Module Tree dove la radice è il file `app.module.ts`. In questo file radice è presente il seguente codice:

```
ngOnInit(): void {  
  if (localStorage.getItem('user')) {  
    const user = JSON.parse(localStorage.getItem('user')!);  
    this.auth.createUser(  
      user.username,  
      user.role,  
      user._token,  
      user.__expirationDate  
    );  
    this.router.navigate(['dashboard']);  
  } else {  
    this.router.navigate(['login']);  
  }  
}
```

Questa funzione viene eseguita solo quando il modulo viene generato per la prima volta dal browser, In questo caso la funzione controlla se nel Local Storage esiste un oggetto chiamato "user" (dove vengono salvati i token e gli altri dati ricevuti in fase di login), se esiste richiama la funzione CreateUser() del servizio "auth", ovvero il servizio che si occupa della authorization, e successivamente ridireziona la "route" (l'indirizzo) del sito all'indirizzo "dashboard", ovvero la pagina visualizzabile solo dopo aver effettuato l'accesso. nel caso non esista l'oggetto "user" nel Local Storage, vuol dire che non è stato eseguito l'accesso e quindi si viene ridirezionati verso la pagina di login.

5.2.1 Pagina di login



A login form consisting of two text input fields stacked vertically. The top field is labeled 'Username' and the bottom field is labeled 'Password'. To the right of the 'Password' field is a small circular icon with an eye, used for toggling password visibility. Below these fields is a blue rectangular button with the text 'Login' in white.

Figura 5.3: Pagina di Login

La pagina contenente il form di login si presenta come in figura 5.3, abbiamo due caselle di input di testo dove possiamo inserire username e password e un pulsante dove poter inviare al server la richiesta. Nel dettaglio, una volta cliccato il pulsante viene chiamata questa funzione

```
onSubmitD(form: NgForm): void {  
  let formData = new FormData();  
  formData.append('username', this.userN);  
  formData.append('password', this.pass);  
  form.reset();  
  this.auth.login(formData).subscribe({  
    next: (data: any) => {  
      //console.log(data);  
      this.onLoginSuccess(data);  
      this.messageService.add({  
        key: 'bc',  
        severity: 'success',  
        summary: 'Login successful',  
        detail: 'token valid for ' + data.expireIn + ' minutes',  
      });  
      this.router.navigate(['dashboard/geolocation']);  
    },  
    error: (error: any) => {  
      //Error callback
```

```
this.messageService.add({
  key: 'bc',
  severity: 'error',
  summary: 'Error',
  detail: error.error.description,
});
//console.log(error);
},
});
}
```

Questa funzione raccoglie i dati del form, ovvero il nome utente e la password inseriti dall'utente, e li inserisce in un oggetto FormData. L'oggetto FormData è un'interfaccia che consente di creare un set di coppie chiave-valore, dove la chiave rappresenta il nome del campo nel form e il valore rappresenta il valore associato a quel campo.

Successivamente, la funzione chiama il metodo di login del servizio auth (il servizio di autorizzazione), passando l'oggetto FormData appena creato come parametro. Il metodo di login restituisce un'observable, che viene sottoscritto utilizzando il metodo subscribe(). Il metodo subscribe() accetta due callback come argomenti: la callback next() viene chiamata quando l'observable emette un valore, ovvero quando il login è stato eseguito con successo, mentre la callback error() viene chiamata quando si verifica un errore durante il processo di login.

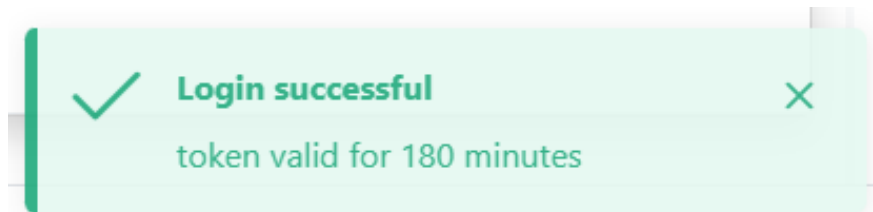


Figura 5.4: Login eseguito correttamente

Se il login ha successo, la funzione chiama la funzione onLoginSuccess() con i dati restituiti dall'observable. In questa funzione viene salvato nella Local Storage un nuovo oggetto "user2" con i dati ricevuti dalla risposta HTTP. Successivamente, viene aggiunto un messaggio di successo alla messageService, un servizio di PrimeNG che consente di visualizzare dei messaggi nell'interfaccia utente tramite Toast a scomparsa. Infine, si viene ridirezionati al link `dashboard/geolocation`.

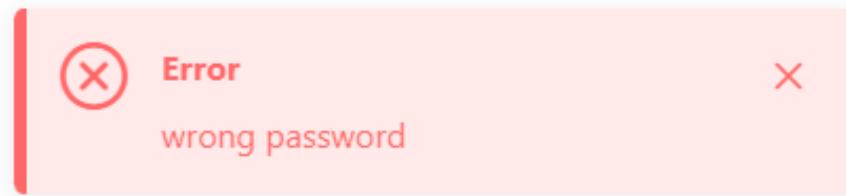


Figura 5.5: Errore nel Login

Se si verifica un errore durante il processo di login, la funzione aggiunge un messaggio di errore al `messageService`, che indica che si è verificato un errore e fornisce una descrizione dell'errore. In entrambi i casi, i campi del form vengono resettati utilizzando il metodo `reset()` della classe `NgForm`.

5.2.2 Invio token

Dopo aver ricevuto il token per l'autenticazione e l'autorizzazione, per poter essere autenticati dal Backend è necessario che ogni chiamata HTTP contenga nell'header il token ricevuto. Per intercettare in automatico ogni chiamata HTTP per inserire nell'header questo campo, nel file `app.module.ts` (questo file rappresenta il modulo radice, viene caricato all'avvio dell'applicazione e ha il compito di inizializzare l'intera struttura dell'applicazione, fornendo informazioni su come caricare e organizzare tutti i moduli e i componenti necessari) ho inserito tra gli import il seguente script

```
JwtModule.forRoot({
  config: {
    tokenGetter: (request) => {
      if (request!.url.includes('Login')) {
        return '';
      }
      return JSON.parse(localStorage.getItem('user'))!._token;
    },
    allowedDomains: ['localhost:44340'], //da modificare in productions
    //disallowedRoutes: ["http://example.com/examplebadroute/"],
  },
}),
```

in questo codice in particolare, viene configurato il modulo `JwtModule` attraverso il metodo `forRoot()`, passandogli un oggetto di configurazione. Il modulo `JwtModule` è utilizzato per gestire i token JWT (JSON Web Token) utilizzati per l'autenticazione e l'autorizzazione nelle applicazioni web. L'oggetto di configurazione contiene tre proprietà:

- `config` è l'oggetto di configurazione effettivo per il modulo `JwtModule`. Qui vengono definite le opzioni per la gestione dei token JWT.

- `tokenGetter` è una funzione che viene chiamata per ottenere il token JWT da utilizzare nelle richieste API. In questo caso, la funzione controlla se la richiesta corrente include la parola 'Login' nell'URL. Se sì, la funzione restituisce una stringa vuota. In caso contrario, viene recuperato il token dall'oggetto dell'utente salvato nella `localStorage`.
- `allowedDomains` è un array di domini consentiti per l'utilizzo dei token JWT. In questo caso, viene consentito solo il dominio `localhost:44340`, ma questo valore deve essere modificato in produzione.

5.3 Servizi Angular

Un servizio in Angular è un tipo di oggetto che fornisce funzionalità riutilizzabili all'interno di un'applicazione. I servizi sono spesso utilizzati per gestire l'accesso ai dati, comunicare con i server e implementare logica di business, possono essere creati utilizzando il decorator `@Injectable`: Questo decorator consente ad Angular di iniettare automaticamente le dipendenze dei servizi quando vengono utilizzati all'interno di altri componenti o altri servizi. L'iniezione delle dipendenze permette di creare servizi modulari e facili da testare.

Un servizio in Angular può essere definito in diversi modi. Ad esempio, un servizio può essere creato come classe TypeScript, come una funzione o come un oggetto JavaScript. In entrambi i casi, il servizio deve implementare il decorator `@Injectable` per indicare ad Angular che si tratta di un servizio e per abilitare l'iniezione delle dipendenze la quale, consente ad Angular di identificare le dipendenze di un componente (o di un servizio) e di fornirle automaticamente quando il componente (o il servizio) viene creato. In questo modo, il servizio può essere utilizzato all'interno del componente (o del servizio) senza dover essere esplicitamente importato.

Nel progetto ho utilizzato 5 servizi per scopi ed utilizzi diversi:

1. `auth.service`
2. `view-changepoint.service`
3. `view-data.service`
4. `view-selected.service`
5. `edit-data.service`

5.3.1 `auth.service`

Questo è un servizio per gestire l'autenticazione tramite comunicazione con il Backend, utilizza il modulo `HttpClient` di Angular per fare richieste HTTP e gestire le risposte. È anche decorato con il decoratore `Injectable`, che lo rende disponibile per l'iniezione di dipendenze in tutta l'applicazione.

Il servizio ha diversi metodi:

- `isAuthenticated()`: questo metodo controlla se l'utente è autenticato verificando se esiste un oggetto "user" nel local Storage del browser e restituisce un valore booleano.
- `createUser()`: questo metodo viene utilizzato per creare una nuova variabile utente che possiede gli stessi dati salvati nel Local Storage.
- `login()`: questo metodo fa una richiesta HTTP POST all'URL `API_LOGIN_URL` per effettuare l'accesso. L'input del metodo è un oggetto `formData` che contiene le credenziali dell'utente e restituisce l'Observable della richiesta.
- `logout()`: questo metodo rimuove i dati dell'utente dal `localStorage` del browser e reimposta la variabile `user` a `null`.
- `refreshToken()`: questo metodo fa una richiesta HTTP POST all'URL `API_REFRESH_TOKEN` per ottenere un nuovo token di accesso utilizzando il token di refresh. L'input del metodo è una stringa di credenziali, che viene inviata nel body e restituisce l'Observable della richiesta.

5.3.2 view-chargepoint.service

Questo è un servizio per la richiesta e la modifica dei dati di una stazione di ricarica di veicoli elettrici.

Il servizio contiene diverse dipendenze:

- `HttpClient`: modulo di Angular per effettuare richieste HTTP.
- `MessageService` di PrimeNG: servizio per la visualizzazione di messaggi a schermo.
- Il servizio `auth.service`: il servizio spiegato precedentemente che contiene operazioni per l'autenticazione.

Il servizio poi utilizza diverse classi importate dal file `../models/chargepoint.ts`, tra cui `ChargePoint`, `Evses`, `Connectors`, `Coordinates`, `Imageinfo`, `Operator`, `StatusSchedule`, `Regular`, `Hours`, `EnergyMix`, `EnergySource` e `EnvironmentalImpact`.

Il servizio ha anche variabili importanti da spiegare:

- `chargePointsValue`: Array pubblico di `ChargePoint` che viene riempito con i dati provenienti dal server, verrà bindato dai moduli per accedere sempre alla versione più aggiornata dei dati
- `available`: Un soggetto (un tipo di Observable castabile) che viene aggiornato subito dopo che l'Array `ChargePointsValue` viene finito di riempire, viene utilizzato per informare gli utenti quando i dati dei punti di ricarica sono disponibili.
- `availableObs`: Observable su `available` per poter richiamare la `subscribe` e quindi sapere quando è disponibile l'Array di dati

Il servizio ha diversi metodi:

- `getBusiness()`: questo metodo fa una richiesta HTTP GET all'URL `API.GET.BUISINESS` e restituisce un Observable della richiesta.
- `getChargePoints()`: questo metodo controlla se siamo autenticati, in caso positivo fa una richiesta HTTP GET all'URL `API.GET.CHARGEPOINTS.ADMIN` se abbiamo i permessi di admin, altrimenti all'URL `API.GET.CHARGEPOINTS`; In caso di successo la risposta della richiesta viene passata al metodo `updateData()` per aggiornare i dati, altrimenti utilizza il servizio `MessageService` per indicare un errore nel raggiungere il server.
- `updateData()`: questo metodo azzerà l'array `chargePointsValue` e poi itera su tutti gli elementi dell'oggetto `data` passato come parametro. Per ogni elemento, vengono creati oggetti `Coordinates`, `Evses`, `StatusSchedule`, `Capability`, `ParkingRestriction`, `Imageinfo` e `Connectors`, che vengono utilizzati per costruire un oggetto `ChargePoint` e aggiungerlo all'array `chargePointsValue`. Alla fine del metodo, viene emesso un valore tramite l'oggetto `Subject` `available` per segnalare che i dati dei punti di ricarica sono disponibili.

5.3.3 view-data.service

Questo è un servizio utilizzato per convertire i tipi di dati, tra cui enum, in stringhe da usare nell'interfaccia grafica. Il servizio, essendo molto semplice e settoriale, non contiene dipendenze esterne.

Il servizio usa variabili ausiliari quali i due array `nameStatus` e `weekDaysNames` che rispettivamente vengono usati per salvare in una stringa il nome di un certo stato e salvare in una stringa i giorni della settimana.

Il servizio possiede i seguenti metodi:

- `StatusDot()`: questo metodo dato in input un array di EVSE ne restituisce lo stato generale del charge point: per esempio se almeno un EVSE è `AVAILABLE`, lo stato generale sarà `AVAILABLE`, anche se altri EVSE non rispettano tale stato;
- `SingleStatusDot()`: questo metodo replica il metodo `StatusDot()` ma per un solo EVSE;
- `valuetDot()`: questo metodo dato in input un numero, restituisce la stringa per descrivere lo stato che quel numero indica;
- `parkingRestrictionString()`: questo metodo viene usato per descrivere tramite stringa una restrizione per il parcheggio;
- `RestrictionsString()`: questo metodo viene usato per descrivere tramite stringa più restrizioni per il parcheggio;
- `FacilitiesString()`: questo metodo viene usato per descrivere tramite stringa le strutture vicinoad una certa stazione di ricarica;
- `PowerString()`: questo metodo, dato un connettore in input, restituisce la stringa descrittiva di Amperaggio e Voltaggio;

- `getStringSources()`: questo metodo viene usato per descrivere tramite stringa una sorgente di energia, dato un `EnergySourceCategory` in input;
- `getStringImpact()`: questo metodo viene usato per descrivere tramite stringa un tipo di impatto ambientale dovuto all'energia, dato un `EnvironmentalImpactCategory` in input;
- `getStringType()`: questo metodo viene usato per descrivere tramite stringa il tipo di luogo dove è ubicato il parcheggio;
- `getStringOpTimes()`: questo metodo si occupa della stampa, tramite stringa, degli orari di apertura suddivisi per giorno di un determinato charge point;
- `getStringOpTimesAux()`: questo metodo è un metodo ausiliario usato da `getStringOpTimes()`;
- `EnergyString()`, `LocationString()`, `FormatString()`, `StatusString()`, `PowerTypeString()`, `SingleFormatString()`: questi metodi, dato in input un tipo enum, restituiscono il valore di quel enum come stringa

5.3.4 view-selected.service

Questo è un servizio molto importante per il funzionamento della dashboard. Ha 2 principali funzioni:

- salva in una variabile i dati del charge point selezionato, in modo da tenere uno "screenshot" dell'oggetto, disponibile in ogni pagina, che rappresenta la colonnina scelta dall'utente;
- si occupa della comunicazioni tra due componenti importanti: la tabella dei charge point e la mappa

Questo servizio non possiede metodi ma il fulcro della sua utilità può essere trovato nelle sue variabili e dalla sua classe interna:

- `MsgRow`: è una classe interna utilizzata come struttura di un pacchetto di comunicazione tra mappa e tabella, contiene tre property:
 1. `id`: questo campo può contenere due valori: nel caso sia una stringa, allora il campo contiene l'id del charge point che è stato selezionato dall'utente, così da comunicarlo anche alla sua controparte; nel caso di un valore nullo invece significa che è stata deselezionato il charge point;
 2. `lat`: latitudine espressa in numero del charge point selezionato, in caso di id null questo campo viene ignorato
 3. `lon`: longitudine espressa in numero del charge point selezionato, in caso di id null questo campo viene ignorato
- `rowSel`: Un soggetto (un tipo di `Observable` castabile) che contiene i messaggi `MsgRow` in arrivo dalla tabella alla mappa (il nome indica che è stata selezionata una riga della tabella e bisogna informare la mappa)

- rowObs: Observable su rowSel per poter richiamare la subscribe e quindi sapere quando arriva un nuovo messaggio
- mapSel: Un soggetto (un tipo di Observable castabile) che contiene i messaggi MsgRow in arrivo dalla mappa alla tabella (il nome indica che è stata selezionata un punto nella mappa e bisogna informare la tabella)
- mapObs: Observable su mapSel per poter richiamare la subscribe e quindi sapere quando arriva un nuovo messaggio

5.3.5 edit-data.service

Questo servizio si occupa della modifica dei dati di una stazione di ricarica di veicoli elettrici. Il servizio contiene diverse dipendenze:

- HttpClient: modulo di Angular per effettuare richieste HTTP;
- MessageService di PrimeNG: servizio per la visualizzazione di messaggi;
- ViewChargePointsService: servizio per la richiesta e la modifica dei dati di una stazione di ricarica.

Il servizio ha anche variabili importanti da spiegare:

- send: Un soggetto (un tipo di Observable castabile) che viene utilizzato per far comunicare due componenti padre-figlio che poi verranno spiegati più avanti, segue la falsariga di MsgRow e ha due possibili valori : "save" e "reset"
- generalinfo: Un valore booleano che viene usato rendere cliccabile un bottone nel caso il form di modifica/aggiunta venga completato correttamente
- isNew: Un valore booleano che viene usato per indicare se si sta modificando o aggiungendo un nuovo charge point

Il servizio possiede i seguenti metodi:

- resetControls(): questo metodo reimposta la proprietà generalinfo a false;
- copyChargePoint(): questo metodo crea una copia del charge point selezionato in modo da rendere disponibile la modifica senza ripercussioni sul dato originale;
- enumsString() e enumKeys(): questi metodi, dato un enum in input, creano l'array di coppie label-value di quel enum
- pushChargePoints(): questo metodo fa una richiesta HTTP POST all'URL API.PUSH_CHARGE_POINT con la colonnina modificata nel body, per modificare o aggiungere i dati. In caso di successo farà visualizzare tramite MessageService un messaggio di successo e aggiornerà i dati dei charge point locali richiamando il metodo getChargePoints() di ViewChargePointsService. In caso di errore nella richiesta farà visualizzare il messaggio di errore all'utente.

5.4 Route Guard per il controllo

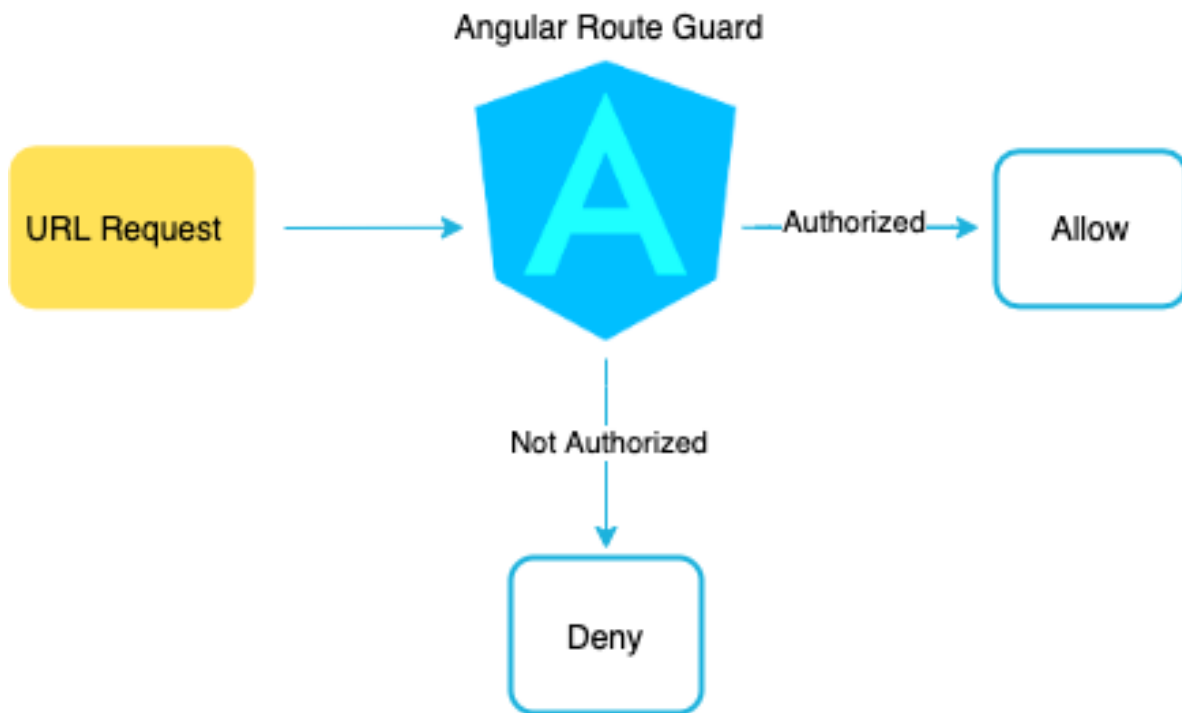


Figura 5.6: schema di reindirizzamento

Una route guard in Angular è un meccanismo utilizzato per proteggere le route dell'applicazione e limitare l'accesso alle pagine in base alle autorizzazioni dell'utente. I guards consentono di gestire la navigazione dell'utente in modo sicuro, impedendo l'accesso alle pagine riservate solo agli utenti autorizzati.

In pratica, una route guard in Angular è una classe TypeScript che implementa l'interfaccia `CanActivate` o altre interfacce correlate come `CanLoad`, `CanActivateChild` o `CanDeactivate`. Queste interfacce definiscono i metodi che vengono eseguiti quando un utente tenta di accedere a una determinata rotta dell'applicazione.

Il metodo `CanActivate`, ad esempio, viene eseguito quando un utente tenta di accedere a una route specifica. La route guard può quindi verificare se l'utente ha l'autorizzazione per accedere alla route e restituire un booleano che indica se l'accesso deve essere consentito o meno. Se la route guard restituisce `true`, l'utente può accedere alla route. Se invece restituisce `false`, l'utente viene reindirizzato a una pagina di errore o alla pagina di login.

In questo progetto ho implementato due route guard diversi: `authGuard` e `adminGuard`

5.4.1 `authGuard`

la route guard `authGuard` ha il compito di verificare se l'utente è autenticato. La classe riceve l'iniezione di dipendenze di quattro servizi:

- AuthService: questo servizio è utilizzato per gestire l'autenticazione tramite comunicazione con il Backend;
- JwtHelperService: questo servizio viene utilizzato per decodificare e verificare la scadenza del token JWT;
- Router: questo servizio viene utilizzato per reindirizzare l'utente alla pagina di accesso quando non è autenticato o il token è scaduto;
- MessageService: questo servizio viene utilizzato per far visualizzare messaggi all'utente in caso di errore o di successo.

Il metodo principale di AuthGuard è il metodo `canActivate()`. Questo metodo viene chiamata ogni volta che un utente tenta di accedere a una rotta protetta (in questo caso ogni route che non sia la pagina di login). La funzione controlla se l'utente è autenticato verificando se esiste un token JWT valido. Se l'utente è autenticato, la funzione restituisce `true` e l'accesso alla rotta viene consentito. Se l'utente non è autenticato o il token è scaduto, la funzione tenta di aggiornare il token utilizzando la funzione `tryRefreshingTokens()`. Se l'aggiornamento del token ha successo, la funzione aggiorna il token nell'oggetto `user` del servizio `AuthService` e salva i nuovi token nel Local Storage del browser. La funzione poi mostra un messaggio di successo all'utente e reindirizza l'utente alla pagina di dashboard consentendone la rotta. Se l'aggiornamento del token fallisce, la funzione mostra un messaggio di errore all'utente, restituisce `false` e l'utente viene reindirizzato alla pagina di accesso.

5.4.2 adminGuard

La route guard `adminGuard` ha il compito di verificare se l'utente autenticato ha il ruolo di amministratore (cioè ha il valore `0` nell'attributo `role` dell'oggetto `user` memorizzato nel `localStorage`). In caso contrario, l'utente viene reindirizzato alla pagina `"notauthorized"`.

La classe riceve l'iniezione di dipendenze di tre servizi:

- AuthService: questo servizio è utilizzato per gestire l'autenticazione tramite comunicazione con il Backend;
- JwtHelperService: questo servizio viene utilizzato per decodificare e verificare la scadenza del token JWT;
- Router: questo servizio viene utilizzato per reindirizzare l'utente alla pagina di accesso quando non è autenticato o il token è scaduto;

Il metodo `canActivate` della classe viene chiamato automaticamente ogni volta che si tenta di navigare verso una rotta protetta (solo la pagina di modifica e di aggiunta di charge point). In questo metodo, viene prima verificato se l'utente è autenticato. In caso contrario, viene restituito il valore `false` per impedire l'accesso alla rotta. Se l'utente è autenticato, viene verificato se il token memorizzato nel `localStorage` è ancora valido utilizzando il servizio `JwtHelperService`. Se il token è valido, viene

verificato se l'utente ha il ruolo di amministratore e, in caso affermativo, viene restituito il valore `true` per permettere l'accesso alla rotta. In caso contrario, viene eseguito un reindirizzamento alla pagina "notauthorized" e restituito il valore `false`.

5.5 Dashboard

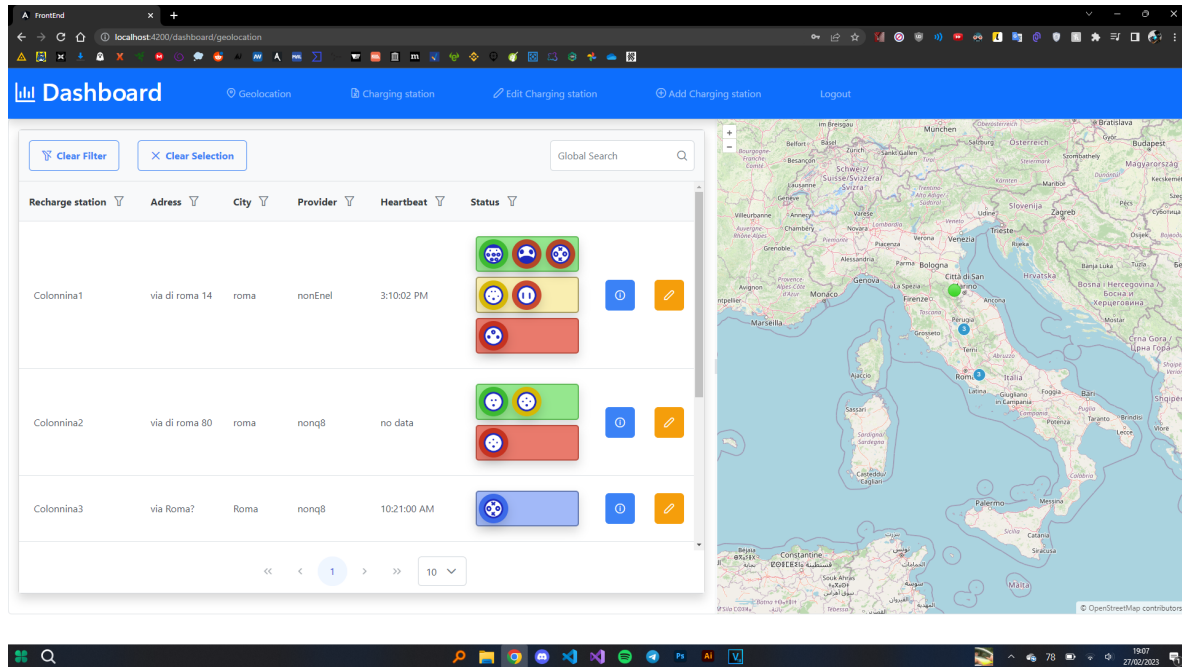


Figura 5.7: Dashboard - pagina principale Geolocation

Una volta effettuato l'accesso si viene reindirizzati nella dashboard del portale. La dashboard contiene due pagine che sono accessibili a tutti e due pagine che sono accessibili solo agli utenti con diritti da amministratore. Le due pagine accessibili a tutti sono:

- **Geolocation:** Pagina principale divisa in due componenti: Una tabella contenente le informazioni principali di tutte le charging station e una mappa interattiva dove si può visionare il luogo di ubicazione di ogni charging station;
- **Charging station:** Pagina dove vengono rese disponibili tutte le informazioni di una charging station selezionata dall'utente. Una versione molto più completa delle informazioni rappresentate nella tabella di Geolocation, nascondendo però alcune informazioni, riservate ai non amministratori.

Le due pagine accessibili solo agli amministratori sono:

- **Edit charging station:** Pagina riservata agli amministratori dove è possibile modificare qualsiasi informazione, valore e dato di una charging station selezionata;
- **Add chargin station:** Pagina riservata agli amministratori dove è possibile aggiungere una nuova charging station nel database inserendo tutti i vari dati e informazioni.

5.6 Geolocation

Geolocation è la pagina principale della dashboard, quella a cui si viene reindirizzati una volta completata la fase di login. Questa pagina ha lo scopo di censire i vari charge point, quindi mostrare informazioni sul numero, il luogo e su diverse caratteristiche di ogni stazione di ricarica. La pagina è formata da due principali componenti che comunicano tra loro. Le funzioni principali di questa pagina sono il filtro e la ricerca: Utilizzando la tabella per filtrare in base a determinati valori, cercare in base a determinate stringhe o la mappa per viaggiare attraverso luoghi e visionare la posizione delle charging station, si può trovare una stazione di interesse (che magari rispetta alcune caratteristiche scelte) e si può selezionare cliccando o la riga nella tabella, o il punto nella mappa. In entrambi i casi l'altro componente selezionerà automaticamente la stessa charging station, quindi per esempio: se selezioniamo la Colonnina3 nella tabella, la mappa si sposterà e zoomerà fino a mostrarci la posizione della stazione di ricarica scelta. Questa selezione sarà consistente tra le pagine e potrà essere deselectata in qualsiasi momento in qualsiasi pagina cliccando l'apposito pulsante. Nella creazione di questo componente viene inviata la richiesta HTTP all'API per richiedere i dati delle charging station utilizzando il servizio precedentemente descritto. In questo modo chiediamo i dati al Backend solo dopo aver autenticato l'utente, evitando problemi di sicurezza.

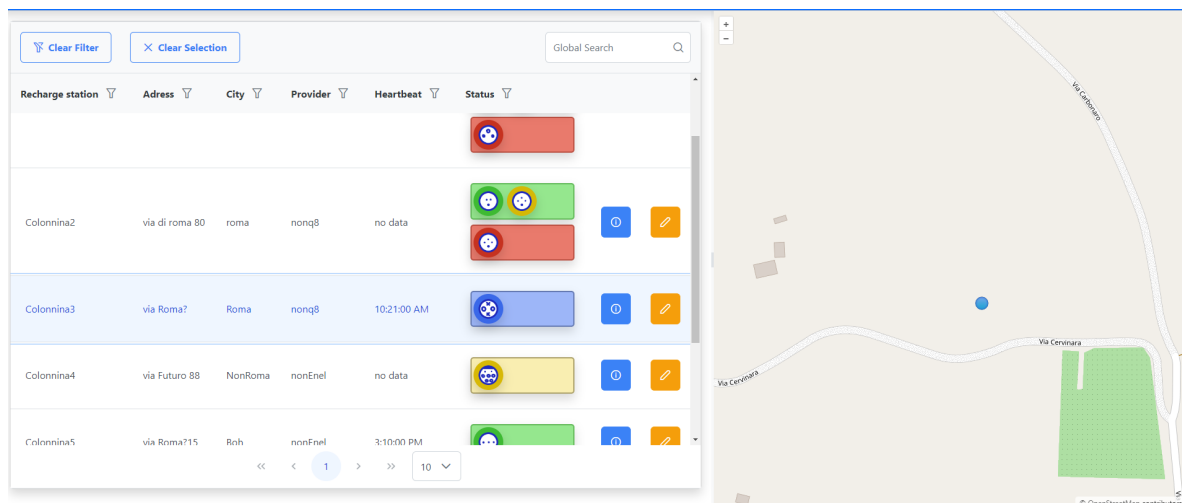


Figura 5.8: Selezione di un charge point

5.6.1 Tabella

La tabella è uno dei due componenti principali della pagina Geolocation. La tabella è stata implementata utilizzando il componente di PrimeNG "ip-table", per renderla dinamica e aggiungere funzionalità. Si presenta come una tabella a 6 colonne + 2 per i bottoni su ogni riga. Ad ogni riga corrisponde una charging station del database. Le colonne sono:

1. Recharge Station: nome della colonnina di ricarica
2. Address: indirizzo completo della colonnina di ricarica

3. City: città dove risiede la colonnina di ricarica
4. Provider: Nome del provider della colonnina di ricarica
5. Heartbeat: timestamp dell'ultimo heartbeat della colonnina di ricarica (ovvero timestamp dell'ultima connessione della colonnina con il sistema centrale)
6. Status: Lo stato è la colonna più complicata poiché non rappresenta un singolo dato, ma più dati contemporaneamente. Ogni colore utilizzato rappresenta un tipo di stato:
 - Verde: Disponibile
 - Blu: In carica
 - Giallo: Prenotato, riservato
 - Rosso: non disponibile

Ogni blocco colorato rappresenta un EVSE e ogni cerchio interno rappresenta un connettore di quell'EVSE. Ogni connettore poi ha un'icona che rappresenta il tipo di presa utilizzata.

I due pulsanti sono il pulsante "info" e il pulsante "edit". Il pulsante "edit" è disponibile solo per gli utenti con permessi di amministratore. Cliccando il pulsante "info" si verrà ridirezionati alla pagina di informazione della colonnina corrispondente alla riga della tabella contenente il pulsante cliccato. Un comportamento analogo viene usato con il pulsante "edit" dove differisce solo il reindirizzamento, il quale rimanda alla pagina di modifica della colonnina di ricarica scelta.

La tabella è dinamica: supporta il filtering su ogni colonna e ha un filtering globale. Il filter globale funziona come una ricerca di una sottostringa su una stringa maggiore e interessa solo le prime 4 colonne (ovvero le colonne che contengono stringhe)

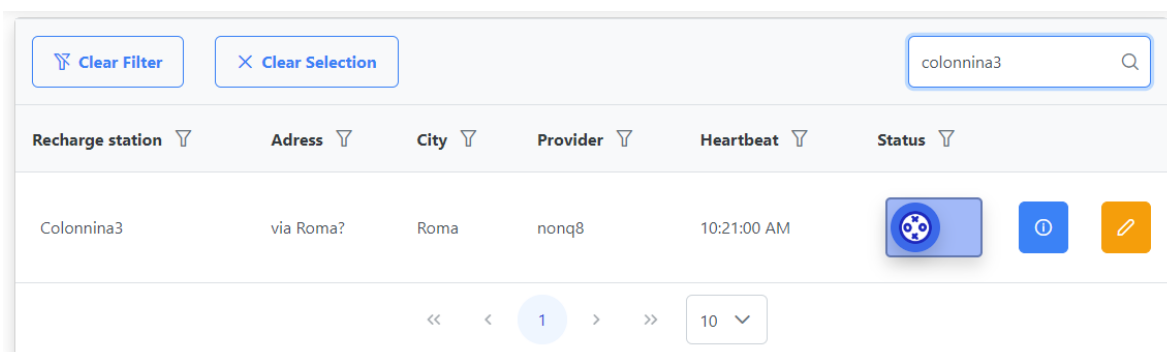
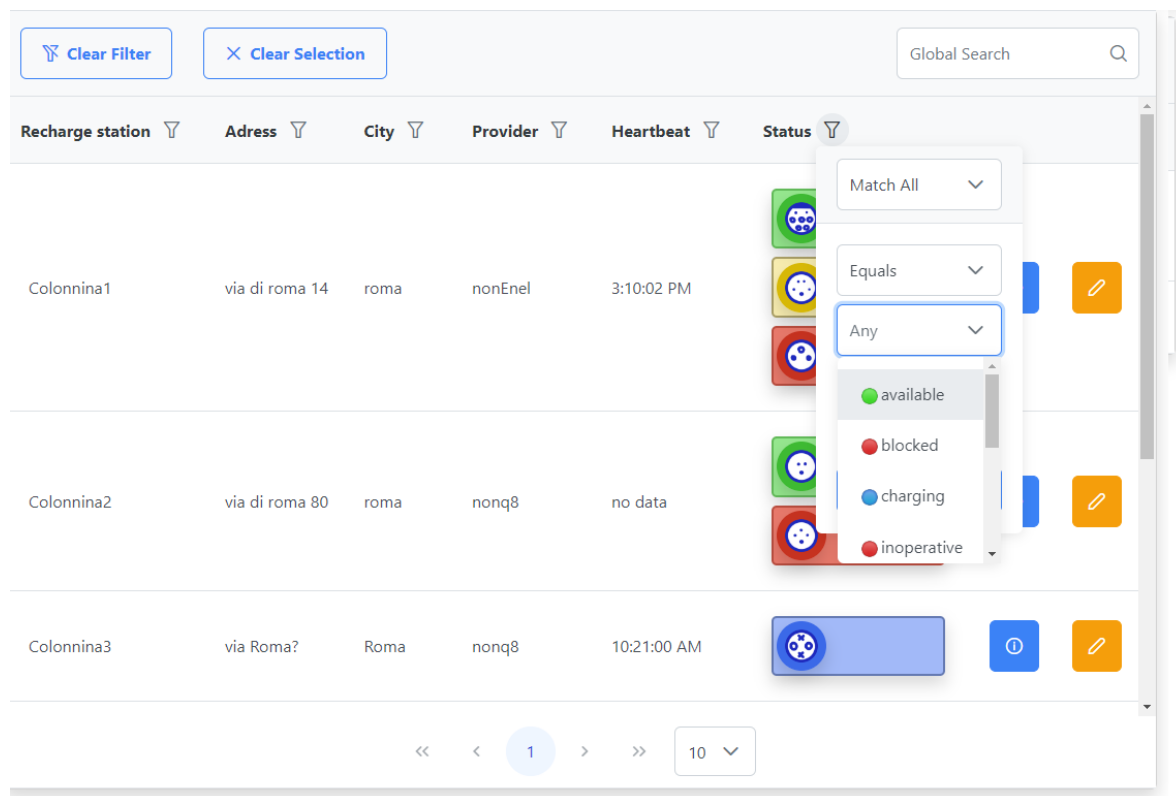


Figura 5.9: Filtro globale tabella

Invece, il filter specifico sulla colonna varia da colonna a colonna: sulle prime 4 colonne è un filtro di sottostringa su una stringa, nella quinta colonna il filtro è di tipo "prima di" o "dopo di" seguito da un valore temporale e infine l'ultimo filtro per lo stato è la selezione da un menù a cascata di un possibile stato tra quelli disponibili.

Nella parte superiore della tabella ci sono due pulsanti: Il primo consente di annullare tutti i filtri inseriti, riportando a poter visionare tutte le righe; il secondo cancella la selezione di un charge point. La tabella ha anche un sistema di pagine per poter rappresentare al meglio più dati possibili.



The screenshot shows a web application interface with a table of charging stations. At the top, there are two buttons: 'Clear Filter' and 'Clear Selection', and a 'Global Search' input field. The table has columns: 'Recharge station', 'Adress', 'City', 'Provider', 'Heartbeat', and 'Status'. A dropdown menu is open for the 'Status' column, showing options: 'Match All', 'Equals', 'Any', 'available', 'blocked', 'charging', and 'inoperative'. The table contains three rows of data. At the bottom, there is a pagination control showing page 1 of 10.

Recharge station	Adress	City	Provider	Heartbeat	Status
Colonnina1	via di roma 14	roma	nonEnel	3:10:02 PM	
Colonnina2	via di roma 80	roma	nonq8	no data	
Colonnina3	via Roma?	Roma	nonq8	10:21:00 AM	

Figura 5.10: Filtro specifico tabella

5.6.2 Mappa

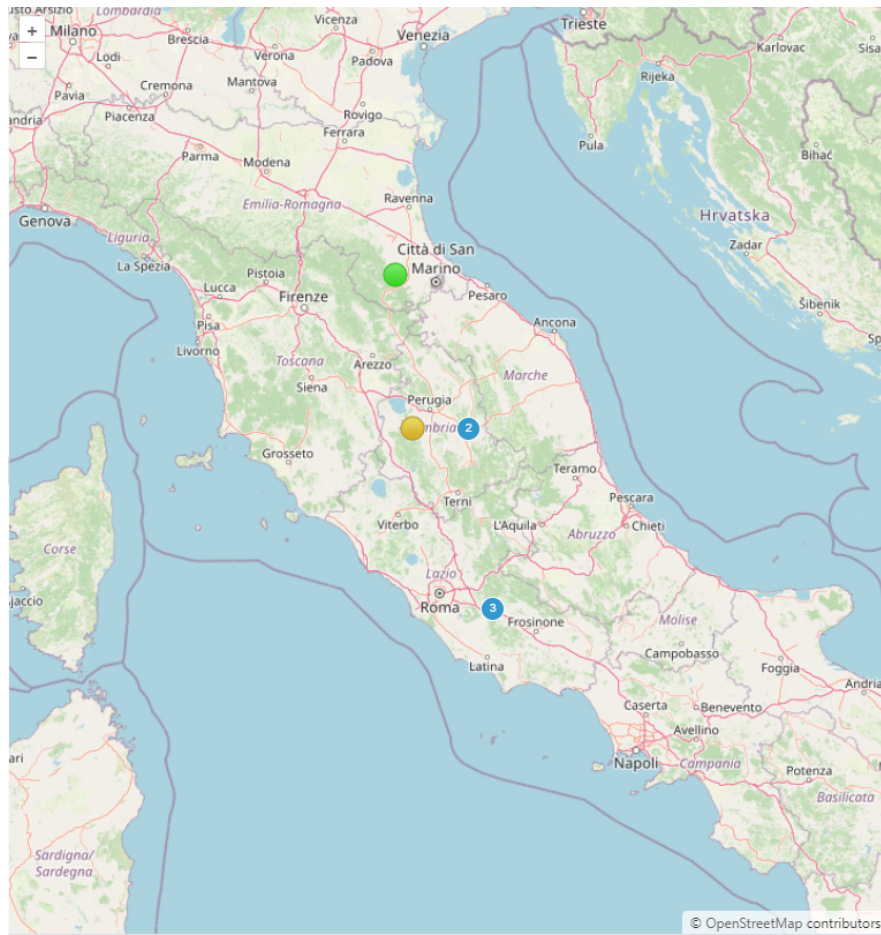


Figura 5.11: Mappa

Inizialmente la mappa era stata implementata utilizzando Gmap di Google, Era una mappa molto dinamica e semplice da usare. Purtroppo leggendo la documentazione viene espressamente indicato che in fase di production per utilizzare la mappa si ha bisogno di una API KEY a pagamento, quindi ho dovuto per forza di cose cambiare implementazione. La scelta è ricaduta sulla mappa OpenLayers. La mappa JavaScript OpenLayers è una libreria open source basata su OpenStreetMap che fornisce strumenti per la creazione e la gestione di mappe interattive su pagine web. Tra le principali funzionalità di OpenLayers si includono:

- Visualizzazione della mappa: OpenLayers consente di visualizzare mappe interattive su pagine web, utilizzando diverse fonti di dati geografici come OpenStreetMap, Bing Maps, Google Maps, mappe WMS e altre. È possibile personalizzare la visualizzazione della mappa impostando il livello di zoom, la posizione iniziale e lo stile dei dati sulla mappa.
- Gestione dei layers: OpenLayers permette di sovrapporre diverse fonti di dati geografici sulla mappa, utilizzando i layers. È possibile aggiungere o rimuovere i layers in modo dinamico, modificare l'ordine di visualizzazione dei layers, e personalizzare lo stile dei dati sulla mappa.

- Interazione con la mappa: OpenLayers offre diverse opzioni per l'interazione con la mappa, come ad esempio lo zoom con il mouse, lo spostamento della mappa tramite drag and drop, l'aggiunta di marker e di forme, e altro ancora.
- Supporto per diverse proiezioni: OpenLayers supporta diverse proiezioni geografiche, come la proiezione Mercatore e la proiezione di Peters, e permette di convertire le coordinate da una proiezione all'altra.
- Supporto per dispositivi mobili: OpenLayers è compatibile con dispositivi mobili come smart-phone e tablet, e consente di creare mappe interattive per applicazioni mobile.
- OpenLayers utilizza JavaScript per creare le mappe, e si integra facilmente con altre tecnologie web come HTML e CSS. La libreria è utilizzata in diverse applicazioni web che richiedono la visualizzazione di dati geografici, come ad esempio applicazioni per la gestione del territorio, applicazioni per la navigazione, applicazioni per il turismo, e altro ancora.

Purtroppo, a differenza di Gmap, OpenLayers ha una documentazione abbastanza raffazzonata e per capire il funzionamento ho dovuto studiare la pagina di esempi di utilizzo fornita dal sito e fare un po' di reverse engineering in alcune parti. La mappa ha due tipi di Marker: I cluster e i marker singoli.

I cluster sono gruppi di marker singoli relativamente vicini e che, con un certo livello di zoom, andrebbero a sovrapporsi o comunque ad essere troppo vicini. Ogni cluster ha una label numerica che rappresenta il numero di marker presenti in quel cluster e, se selezionato, la visuale della mappa andrà a posizionarsi al centro del cluster effettuando uno zoom per mostrare i marker o i sottocluster presenti.

I marker singoli rappresentano la posizione nella mappa delle singole colonnine di ricarica. il colore dei marker rappresenta lo stato generale della colonnina e, una volta cliccato, verrà selezionata la colonnina corrispondente.

5.7 Pagina informazioni colonnine

La pagina Charging Station è la pagina che contiene tutte le informazione di un charge point. Nel caso si finisse in questa pagina senza aver selezionato alcuna colonnina, sarà visibile solo il menù a tendina di scelta e un messaggio che ci invita a selezionare una charging station.

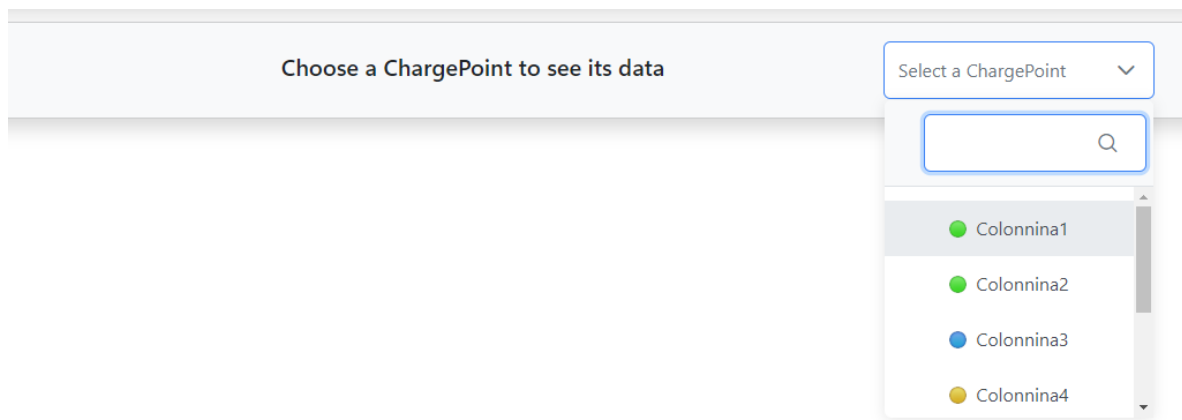


Figura 5.12: Selezione di un charge point

La pagina di informazioni è divisa in due metà, come mostrato in figura 5.13:

Nella metà di sinistra abbiamo le informazioni relative alla charging station quali nome, coordinate, operatore, le varie facilities e caratteristiche; sotto abbiamo due menù Breadcrumb: Nel primo abbiamo gli orari di apertura settimanali, l'identità, il tipo di luogo in cui è situata e il numero di EVSE della colonna; nel secondo abbiamo due tabelle che ci mostrano rispettivamente le sorgenti di energia utilizzate (e le loro percentuali di utilizzo) e l'impatto ambientale della colonna.

Nella metà di destra abbiamo le informazioni sugli EVSE della colonna. sono presenti tanti Breadcrumb menù quanti EVSE. Dentro ogni menù abbiamo un blocco colorato che rappresenta l'EVSE (il colore rappresenta lo stato) con all'interno i vari dati quali id, numero fisico, stato, numero di connettori, livello di ubicazione, le sue varie capabilities e le restrizioni sul parcheggio; Inoltre per ogni menù di un EVSE ci sono tanti Breadcrumb menù quanti connettori ha quel determinato EVSE. Ogni menù del connettore ha i suoi dati specifici quali tipo di connettore, tipo di corrente, potenza e stato.

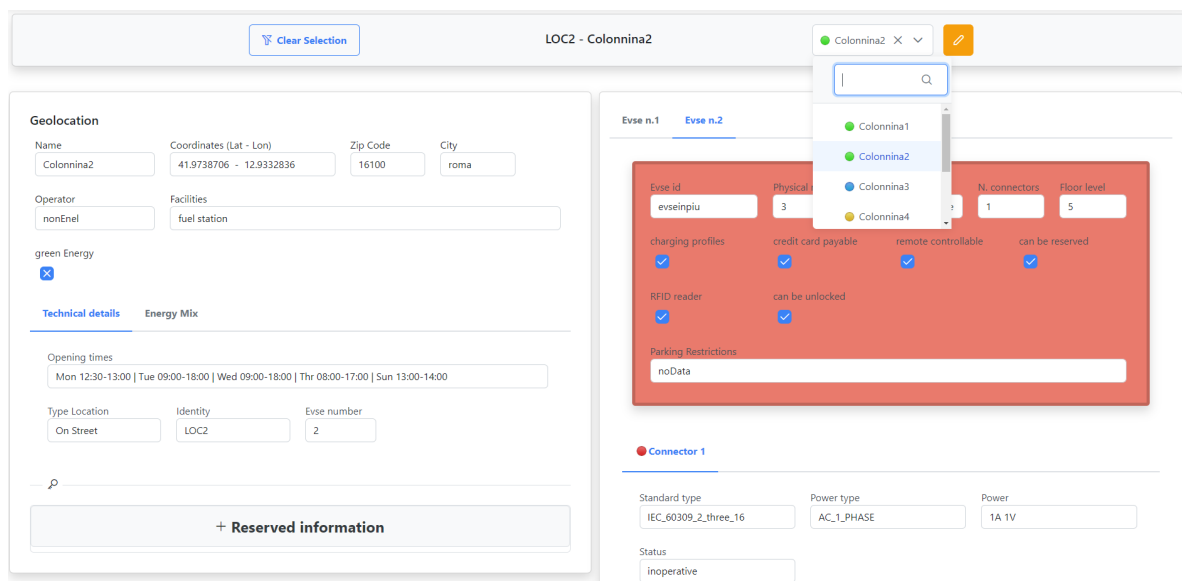


Figura 5.13: Pagina Charging Station per le informazioni specifiche

Se l'utente è un amministratore sarà presente in alto a destra il pulsante "edit", per passare velocemente alla pagina di modifica di questa colonnina selezionata e in basso a sinistra un accordion menù con le informazioni riservate.

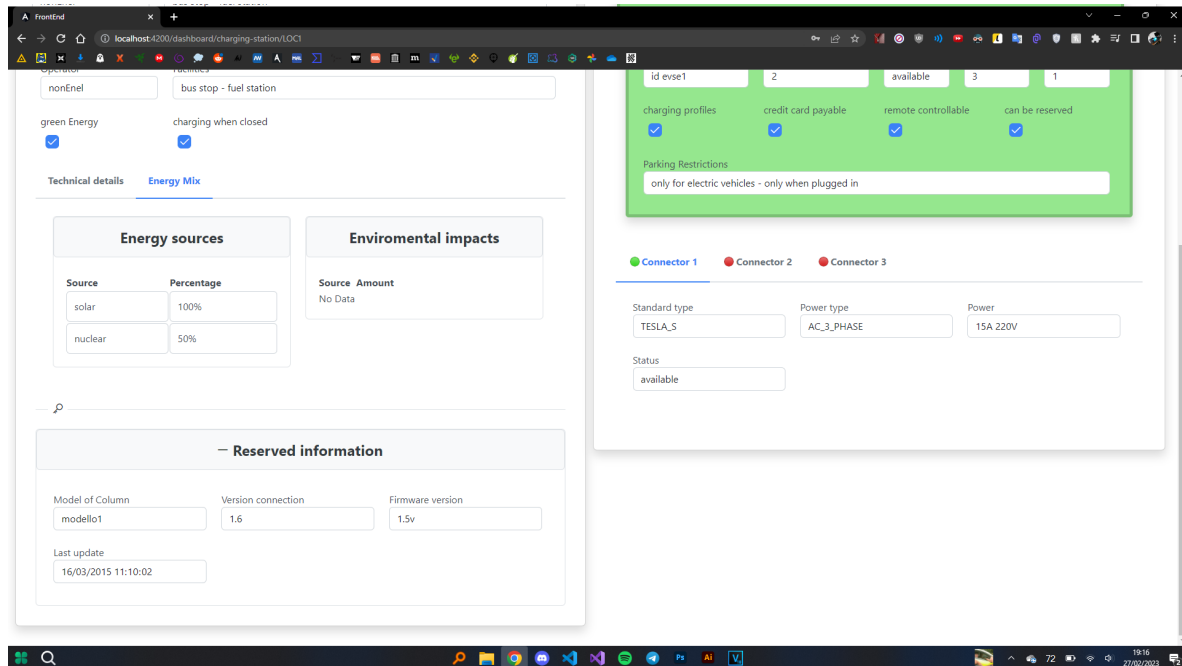


Figura 5.14: Informazioni riservate

5.8 Modifica colonnine

La pagina di modifica colonnine è una delle due pagine accessibili solo da utenti con privilegi di amministrazione. In questa pagina è possibile vedere e modificare tutti i dati dei vari charge point e inviarli al Backend per la modifica effettiva sul database. La pagina è formata da più form, ogni singolo form deve essere validato, ovvero ogni campo obbligatorio (contrassegnato con un asterisco) deve essere riempito e ogni campo deve avere un tipo di dato accettato, per poter inviare le modifiche al database.

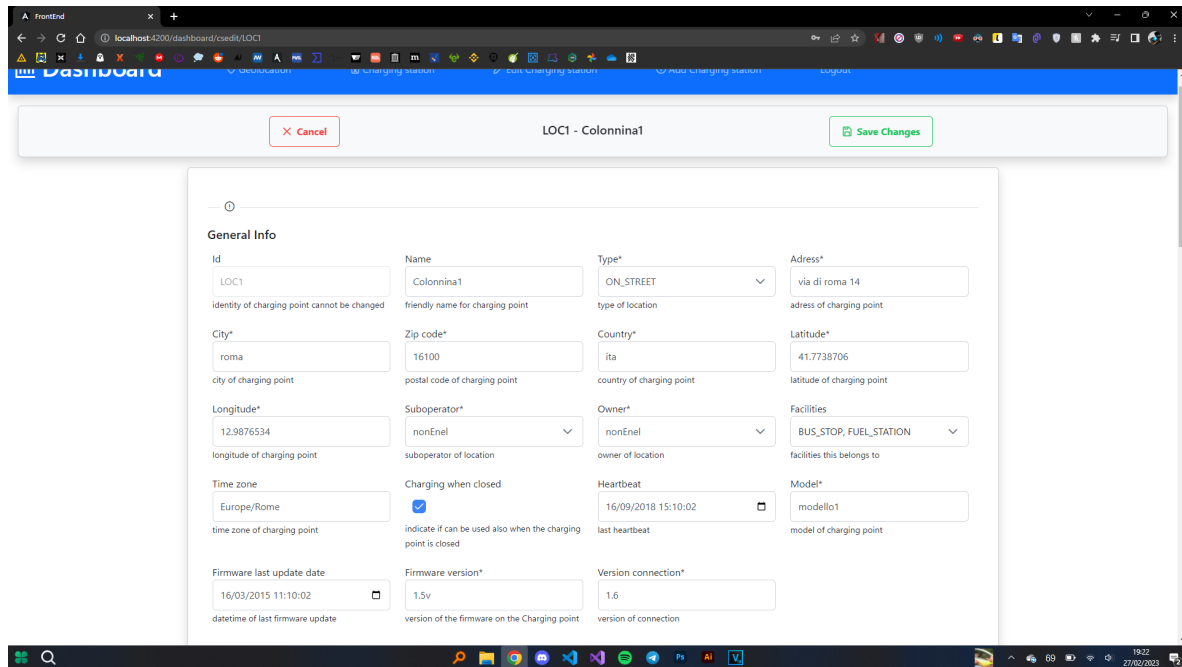
La modifica dei dati viene effettuata su una copia dell'oggetto charge point creata dal servizio edit-data, quindi è sempre possibile annullare le varie modifiche utilizzando il pulsante "Cancel" situato in alto a sinistra. Il pulsante in alto a destra (attivo solo se i form sono validati) invece si occuperà di effettuare, tramite servizio, una chiamata HTTP all'API di modifica delle colonnine inviando nel body della richiesta la copia del charge point modificata.

Possiamo suddividere questa pagina in 4 parti:

1. modifica delle informazioni generali;
2. modifica degli orari di apertura;
3. modifica dei dati energetici;
4. modifica degli EVSE e connettori.

5.8.1 modifica delle informazioni generali

Questa prima parte è composta da un form semplice contenente vari campi di input con le informazioni generali come indirizzo, nome, coordinate, facilities ecc. Il campo Id non può essere modificato poiché viene utilizzato come identificativo anche nel database e la modifica di questo campo comporterebbe problemi di consistenza.



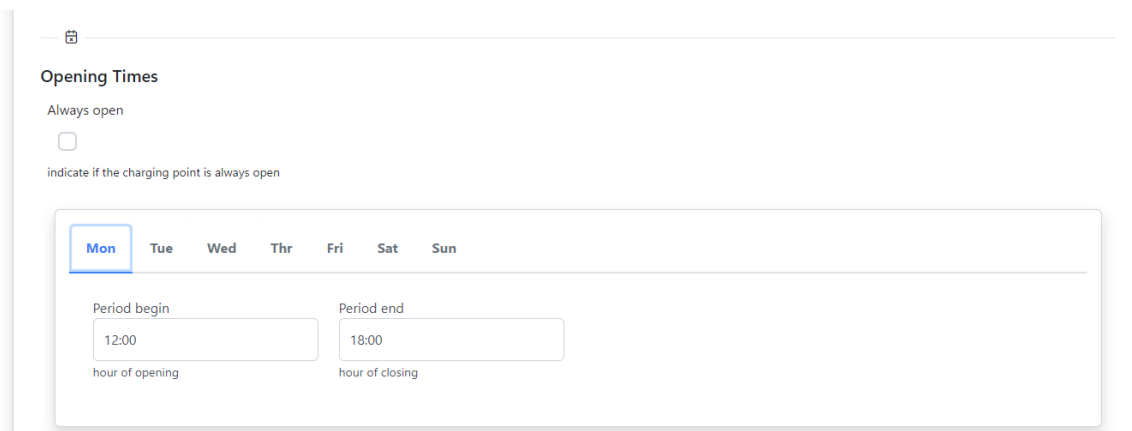
The screenshot shows a web browser window displaying a dashboard with a form titled "LOC1 - Colonnina1". The form is divided into sections, with the "General Info" section visible. It contains various input fields and dropdown menus for charging point details. At the top of the form, there are "Cancel" and "Save Changes" buttons. The browser's address bar shows the URL "localhost:4200/dashboard/cedit/LOC1".

General Info			
Id LOC1 <small>identity of charging point cannot be changed</small>	Name Colonnina1 <small>friendly name for charging point</small>	Type* ON_STREET <small>type of location</small>	Adress* via di roma 14 <small>adress of charging point</small>
City* roma <small>city of charging point</small>	Zip code* 16100 <small>postal code of charging point</small>	Country* ita <small>country of charging point</small>	Latitude* 41.7738706 <small>latitude of charging point</small>
Longitude* 12.9876534 <small>longitude of charging point</small>	Suboperator* nonEnel <small>suboperator of location</small>	Owner* nonEnel <small>owner of location</small>	Facilities BUS_STOP, FUEL_STATION <small>facilities this belongs to</small>
Time zone Europe/Rome <small>time zone of charging point</small>	Charging when closed <input checked="" type="checkbox"/> <small>indicate if can be used also when the charging point is closed</small>	Heartbeat 16/09/2018 15:10:02 <small>last heartbeat</small>	Model* modello1 <small>model of charging point</small>
Firmware last update date 16/03/2015 11:10:02 <small>datetime of last firmware update</small>	Firmware version* 1.5v <small>version of the firmware on the Charging point</small>	Version connection* 1.6 <small>version of connection</small>	

Figura 5.15: Pagina modifica charge point - informazioni generali

5.8.2 modifica degli orari di apertura

La modifica degli orari di apertura è impostata in modo da poter inserire per ogni giorno della settimana gli orari di apertura e di chiusura. Nel caso si volesse impostare una colonnina aperta sette giorni su sette, ventiquattro ore su ventiquattro, è possibile selezionare la checkbox "Always open" in questo modo il menù della settimana scomparirà e la colonnina sarà impostata per essere sempre aperta.



The screenshot shows a web browser window displaying a dashboard with a form titled "Opening Times". The form includes a checkbox for "Always open" and a table for setting opening and closing times for each day of the week. The "Mon" tab is currently selected.

Opening Times						
Always open <input type="checkbox"/> <small>indicate if the charging point is always open</small>						
Mon	Tue	Wed	Thr	Fri	Sat	Sun
Period begin 12:00 <small>hour of opening</small>						
Period end 18:00 <small>hour of closing</small>						

Figura 5.16: Pagina modifica charge point - orari di apertura

5.8.3 modifica dei dati energetici

La modifica dei dati energetici ha relativamente pochi campi, però a differenza delle parti sopracitate, ha dei sottoform nascosti usati per l'inserimento di nuove sorgenti e dati sull'impatto ambientale a cui si può accedere tramite il pulsante "add" nei due accordion menù dedicati, questo aprirà un dialog menù dove si potrà inserire il dato desiderato e, solo dopo aver validato quel sottoform, aggiungere il dato alla colonnina in modifica. È anche possibile eliminare sorgenti energetiche e dati sull'impatto ambientale grazie agli appositi pulsanti rossi con l'icona del cestino.

Energy Mix

Green energy ☒

Indicate if the charging point use green energy

Energy sources

Source	Percentage	
SOLAR	100%	
NUCLEAR	50%	

Environmental impacts

SourceAmount

Supplier name: greenInc
supplier of the energy

Energy product name: energia solare
name of the energy suppliers product/tariff plan

Figura 5.17: Pagina modifica charge point - sorgente energetica

Add Sources

New source*

New amount*

Submit

Figura 5.18: Pagina modifica charge point - dialog aggiunta impatto ambientale

5.8.4 modifica degli EVSE e connettori

La modifica degli EVSE e dei connettori utilizza Breadcrumb menù per mantenere un certo ordine nella pagine e non rendere troppo lungo lo scorrimento tra EVSE diversi. Ogni EVSE ha 8 campi di cui 7 modificabili (id non è modificabile per come scritto prima) e ha i suoi connector anchessi in un Breadcrumb menù. Gli EVSE e i connettori non possono essere cancellati per questioni di storico

degli ordini. Cliccando il pulsante "Add Evse" si aprirà un dialog menù simile a quello per le sorgenti energetiche, dove si potranno inserire i nuovi dati del nuovo EVSE e solo dopo aver validato il sottoform, aggiungere il nuovo EVSE alla charging station.

The screenshot shows the 'Evses' management interface. At the top, there's a header with a hamburger menu icon and the title 'Evses'. Below this, a blue button labeled 'Add Evse +' is visible. The main content area has three tabs: 'Evse n.1', 'Evse n.2', and 'Evse n.3'. The 'Evse n.1' tab is selected. The form contains several input fields and dropdown menus: 'Uid' (text input with value 3256), 'Id' (text input with value id evse1), 'Status*' (dropdown menu with value AVAILABLE), 'Capabilities' (dropdown menu with value CHARGING_PROFILE_CA...), 'Floor level*' (text input with value 1), 'Physical reference*' (text input with value 2), 'Parking restrictions' (dropdown menu with value EV_ONLY, PLUGGED), and 'Last update date' (text input with value 01/03/2016 20:45:05). Below the form, there's a blue button labeled 'Add Connector +'. The form also includes descriptive text for each field, such as 'unique id of evse, cannot be changed' for Uid and 'id from eMI3 standard version V1.0' for Id.

Figura 5.19: Pagina modifica charge point - evse

La modifica dei connettori avviene in modo analogo a quella degli EVSE

The screenshot shows the 'Connectors' management interface. At the top, there's a header with a hamburger menu icon and the title 'Connectors'. Below this, a blue button labeled 'Add Connector +' is visible. The main content area has two tabs: 'Connector 1' and 'Connector 2'. The 'Connector 1' tab is selected. The form contains several input fields and dropdown menus: 'Id connector*' (text input with value 1), 'Standard*' (dropdown menu with value IEC_60309_2_three_32), 'Format type*' (dropdown menu with value CABLE), 'Power type*' (dropdown menu with value AC_1_PHASE), 'Voltage*' (text input with value 220V), 'Amperage*' (text input with value 15A), 'Status*' (dropdown menu with value RESERVED), 'Id tariff*' (text input with value 12), and 'Last update' (text input with value 2015-03-16T12:35:05). Below the form, there's a blue button labeled 'Add Connector +'. The form also includes descriptive text for each field, such as 'id of connector in the evse, cannot be changed' for Id connector* and 'the standard of the installed connector' for Standard*.

Figura 5.20: Pagina modifica charge point - connettori

5.9 Aggiunta colonnine

L'aggiunta delle colonnine è la seconda pagina accessibile solo dagli utenti con permessi di amministrazione. In questa pagina, come per la modifica, sono presenti più form da validare per poter inviare i dati al Backend. L'aggiunta dei dati viene effettuata su un oggetto chargepoint vuoto creato al caricamento della pagina ed è possibile annullare il processo in qualsiasi momento.

Il componente del form è lo stesso della modifica, solo che i dati vengono boundati ad un oggetto vuoto. l'id della colonnina viene scelto dal Backend per evitare problemi di consistenza, mentre gli altri campi possono essere modificati a piacimento.

Una volta validati i form, sarà possibile inviare la richiesta HTTP all'API del Backend che si occupa della modifica e aggiunta delle charging station.

The screenshot shows the 'Create new ChargePoint' form with the 'General Info' section expanded. The form has a header bar with a red 'Reset' button, the title 'Create new ChargePoint', and a green 'Save Changes' button. The form fields are organized into a grid:

- Id**: Text input, note: 'identity of charging point cannot be changed'.
- Name**: Text input, note: 'friendly name for charging point'.
- Type***: Dropdown menu, note: 'type of location'.
- Adress***: Text input, note: 'adress of charging point'.
- City***: Text input, note: 'city of charging point'.
- Zip code***: Text input, note: 'postal code of charging point'.
- Country***: Text input, note: 'country of charging point'.
- Latitude***: Text input with '0' pre-filled, note: 'latitude of charging point'.
- Longitude***: Text input with '0' pre-filled, note: 'longitude of charging point'.
- Suboperator***: Dropdown menu, note: 'suboperator of location'.
- Owner***: Dropdown menu, note: 'owner of location'.
- Facilities**: Dropdown menu, note: 'facilities this belongs to'.
- Time zone**: Text input, note: 'time zone of charging point'.
- Charging when closed**: Checkbox, note: 'indicate if can be used also when the charging point is closed'.
- Heartbeat**: Text input with 'gg/mm/aaaa --:--' pre-filled, note: 'last heartbeat'.
- Model***: Text input, note: 'model of charging point'.
- Firmware last update date**: Text input with 'gg/mm/aaaa --:--' pre-filled, note: 'datetime of last firmware update'.
- Firmware version***: Text input, note: 'version of the firmware on the Charging point'.
- Version connection***: Text input, note: 'version of connection'.

Figura 5.21: Pagina creazione nuovo charge point - informazioni generali

The screenshot shows the 'Create new ChargePoint' form with the 'Opening Times', 'Energy Mix', and 'Evses' sections expanded.

- Opening Times**:
 - Always open**: Checked checkbox, note: 'indicate if the charging point is always open'.
- Energy Mix**:
 - Green energy**: Unchecked checkbox, note: 'indicate if the charging point use green energy'.
 - Energy sources**: A box with an 'Add' button and a '+' icon, note: 'Source Percentage'.
 - Enviromental impacts**: A box with an 'Add' button and a '+' icon, note: 'Source Amount'.
 - Supplier name**: Text input, note: 'supplier of the energy'.
 - Energy product name**: Text input, note: 'name of the energy suppliers product/tariff plan'.
- Evses**:
 - Add Evse**: A button with a '+' icon.

Figura 5.22: Pagina creazione nuovo charge point - campi speciali da completare

Capitolo 6

Conclusioni e ringraziamenti

6.1 Conclusioni

La realizzazione del portale web è stata un'attività molto complessa e impegnativa. Solo grazie all'utilizzo di tutte le tecnologie riportate è stato possibile implementare un sistema completo e integrato per supportare la mobilità elettrica.

Il risultato finale è stato un'applicazione web moderna e intuitiva, in grado di fornire una soluzione completa per la gestione delle colonnine di ricarica per i fornitori e gestori, e una piattaforma user-friendly per gli utenti finali.

L'attività di tirocinio e tesi ha fornito un'ottima occasione per apprendere nuove competenze tecniche e lavorare a stretto contatto con un'azienda all'avanguardia nel settore dell'ICT. Inoltre, ha permesso di approfondire la conoscenza della mobilità elettrica e delle tecnologie ad essa correlate, acquisendo competenze preziose e spendibili in futuro.

In conclusione, l'implementazione di un portale web per la gestione delle colonnine di ricarica rappresenta un passo importante verso la diffusione della mobilità elettrica e la riduzione delle emissioni inquinanti. L'applicazione sviluppata nel contesto di questo progetto è stata un'importante contributo all'innovazione del settore, dimostrando la possibilità di utilizzare tecnologie avanzate per creare soluzioni efficaci e funzionali per le sfide del futuro.

6.2 Ringraziamenti

Desidero ringraziare di cuore tutte le persone che mi hanno supportato e incoraggiato durante il mio percorso accademico. In primo luogo, vorrei ringraziare i miei genitori e la mia famiglia per il loro amore, la loro pazienza e il loro sostegno incondizionato.

Vorrei inoltre ringraziare i miei amici, sia quelli di lunga data che quelli incontrati durante gli studi universitari, per le risate, le discussioni e il supporto reciproco che ci siamo dati. Un ringraziamento speciale va ai miei colleghi universitari con cui ho condiviso i momenti più intensi di questo percorso, dalle notti a ripassare prima degli esami alle discussioni accademiche.

Desidero inoltre ringraziare il Gruppo SIGLA per avermi offerto l'opportunità di effettuare il tirocinio curricolare presso la loro azienda e per avermi guidato nella implementazione del progetto.

Infine, desidero ringraziare il mio relatore e tutti i professori che mi hanno seguito nel mio percorso universitario, per la loro competenza, disponibilità e preziosi consigli. Senza il supporto di tutte queste persone, non sarei arrivato fino a qui. Grazie di cuore a tutti!

Elenco delle figure

2.1	Esempio dei componenti principali di una stazione di ricarica	3
2.2	Schema base di una possibile infrastruttura di ricarica elettrica.	4
2.3	Topologia generale OCPI.	5
2.4	Architettura progetto	8
3.1	Esempio di un documento MongoDB	11
3.2	Schema ER del Database SQL	16
4.1	DotNET e API REST	17
4.2	Extension method utilizzata per Dependency injection	21
4.3	Struttura di un JWT	22
4.4	Pagina principale di Swagger	28
4.5	Test API di Login	28
4.6	Test API di Login - Risposta	29
5.1	Angular	30
5.2	PrimeNG	31
5.3	Pagina di Login	33
5.4	Login eseguito correttamente	34
5.5	Errore nel Login	35
5.6	schema di reindirizzamento	41
5.7	Dashboard - pagina principale Geolocation	43
5.8	Selezione di un charge point	44
5.9	Filtro globale tabella	45
5.10	Filtro specifico tabella	46
5.11	Mappa	47
5.12	Selezione di un charge point	49
5.13	Pagina Charging Station per le informazioni specifiche	49
5.14	Informazioni riservate	50
5.15	Pagina modifica charge point - informazioni generali	51
5.16	Pagina modifica charge point - orari di apertura	51

5.17	Pagina modifica charge point - sorgente energetica	52
5.18	Pagina modifica charge point - dialog aggiunta impatto ambientale	52
5.19	Pagina modifica charge point - evse	53
5.20	Pagina modifica charge point - connettori	53
5.21	Pagina creazione nuovo charge point - informazioni generali	54
5.22	Pagina creazione nuovo charge point - campi speciali da completare	54

Bibliografia

- [1] “Electric vehicle recharging and roaming protocols: A brazilian electromobility case study,” Springer, 2022.
- [2] eopen charge alliance, “Ocpp 1.x,” 2017.
- [3] eco movement, “Ocp1,” 2017.
- [4] Microsoft, “Net docs,” 2022.
- [5] Google, “Angular docs,” 2022.
- [6] PrimeFaces, “Primeng docs,” 2022.