TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Backend implementace Open Charge Point Protokolu verze 1.5

Bakalářská práce

Studijní program: B0613A140005 Informační technologie
Studijní obor: Aplikovaná informatika

Autor práce: **David Šoltés**
Vedoucí práce: Ing. Lukáš Krčmář
Ústav mechatroniky a technické informatiky

Konzultant práce: Ing. Pavel Jandura, Ph.D.
Ústav mechatroniky a technické informatiky

Liberec 2022

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Backend implementace Open Charge Point Protokolu verze 1.5

Jméno a příjmení: **David Šoltés**
Osobní číslo: M20000048
Studijní program: B0613A140005 Informační technologie
Specializace: Aplikovaná informatika
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2021/2022**

Zásady pro vypracování:

1. Seznamte se s nabíjecí stanicí CPC50 od společnosti Simens.
2. Seznamte se s jazykem C# a technologiemi .NET, ve kterých budete backendovou část realizovat.
3. Navrhněte a realizujte API s implementací protokolu OCPP 1.5 pro dobíjecí stanici.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30–40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Alcaraz, Cristina & Lopez, Javier & Wolthusen, Stephen. (2017). OCPP Protocol: Security Threats and Challenges. IEEE Transactions on Smart Grid. PP. 1-1. 10.1109/TSG.2017.2669647.

[2] Open Charge Point Protocol: Interface description between Charge Point and Central System. OPEN CHARGE ALLIANCE [online]. 2010 [cit. 2021-10-12]. Dostupné z: https://www.openchargealliance.org/

[3] Albahari, Joseph: C# 7.0 in a Nutshell: The Definitive Reference, O'Reilly Media; 1 edition (October 28, 2017), ISBN: 978-1491987650

Vedoucí práce:	Ing. Lukáš Krčmář Ústav mechatroniky a technické informatiky
Konzultant práce:	Ing. Pavel Jandura, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	29. září 2021
Předpokládaný termín odevzdání:	16. května 2022

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Černohorský, Ph.D.
vedoucí ústavu

V Liberci dne 29. září 2021

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

11. května 2022 David Šoltés

Poděkování

Rád bych poděkoval vedoucímu Ing. Lukášovi Krčmářovi za vytvoření a prosazení zadání této bakalářské práce. Dále za jeho odborné vedení a získání znalostí, které vedly k dokončení této bakalářské práce.

Dále bych poděkoval panu Dr. Pavlovi Jandurovi za jeho rady a čas, který mi v rámci implementace věnoval.

V poslední řadě bych chtěl poděkovat své přítelkyni Ing. Tereze Šubrové a rodině, která mi dodávala motivaci práci dodělat.

Abstrakt

Tato bakalářská práce se zabývá návrhem, realizací a implementací backendového řešení pro dobíjecí stanici. V rámci práce vznikl nový systém, který využívá části komunikačního protokolu OCPP verze 1.5, relační databázi, .NET API, SOAP API a WPF pro desktopovou aplikaci. Cílem práce je umožnit administrátorovi vzdálenou správu dobíjecích karet a zobrazení statistik dobíjení. Potřebné části systému již byly aplikovány a vedlejší jsou navrženy tak, aby dobře podporovaly budoucí vývoj.

Klíčová slova: API, OCPP 1.5, SOAP API, Entity framework, WPF, Material design, dobíjecí stanice

Abstract

This bachelor thesis deals with the design, implementation and implementation of a backend solution for a charging station. The thesis developed a new system that uses parts of the OCPP version 1.5 communication protocol, relational database, .NET API, SOAP API and WPF for desktop application. The goal of the work is to allow the administrator to remotely manage the recharge cards and view recharge statistics. The necessary parts of the system have already been applied and the secondary parts are designed to support future development as well.

Keywords: API, OCPP 1.5, SOAP API, Entity framework, WPF, Material design, charging station

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Seznam zkratek

SW	Software
SPZ	Státní poznávací značka
BC	Backend
Namespace	Jmenný prostor
WPF	Windows Presentation Foundation, knihovna tříd pro tvorbu grafického rozhraní
API	Aplication programable interface, aplikační programovatelné rozhraní
SOAP	Simple object access protocol, jednoduchý objektov
HMI	Human machine interface, rozhraní člověk-stroj
Endpoint	Koncový bod
GSM	Global System for Mobile communication, globální systém pro mobilní komunikaci
RFID	Radio Frequency Identification, rádio frekvenční identifikátor
SLA	Service Level Agreements, úrovně smluvních podmínek
Gb	Giga byte
DTO	Data transfer object, Objekt pro přenosu dat

Úvod

Elektrických vozidel je v České republice rok od roku více. Taková vozidla lze na první pohled poznat podle písmen “EL” ve státní poznávací značce. Pro právo nárokovat danou SPZ musí vozidlo splňovat dvě podmínky. Druh paliva – čistě elektromotor nebo kombinovaný pohon (tzv. hybrid) a emise nižší jak 50 g CO_2 na 100 kilometrů.

Elektromobily mají v ČR určité výhody – osvobození od placení dálniční známky a parkování v modrých a jinak barevně značených zónách zdarma. Hlavní výhodou elektromobility je levnější běžná údržba, není potřebná nákladná údržba jako u spalovacího motoru. Dále mezi výhody elektromobility je obecně fakt, že díky nim není potřeba platit silniční daň.

Dobíjecí stanice nejsou úplně všude dobře dostupné. Dobrá dostupnost dobíjecích stanic je převážně kolem hlavních tahů a u vybraných nákupních center. Cena za dobíjení elektrických vozidel se odvíjí od rychlosti dobíjení: čím rychleji se vozidlo bude nabíjet, tím vyšší bude sazba za kWh. Nejjednodušší dobíjení je v domácím prostředí pomocí AC dobíjecího adaptéru. Nejjednodušší je připojení do jednofázové zásuvky. V době platnosti nízkého tarifu (dříve noční proud) je cena elektrické energie značně výhodnější, může být méně než 5 Kč za kWh. Průměrný elektromobil na ujetí 100 km vzdálenosti spotřebuje 15 kWh, tak se dostaneme na méně než 75 Kč za 100 km, což je oproti spalovacímu pohonu výrazně méně.

Cílem této bakalářské práce je vytvořit backendovou část pro dobíjecí stanici, která umožní správci dobíjecí stanice sledovat, kdo kolik energie natankoval a přidávat nebo odebírat oprávnění tankovat za použití RFID karet. Část centrálního systému bude realizována pomocí implementace potřebných částí protokolu OCPP verze 1.5.

Rešerše

1. Dobíjecí stanice

Dobíjecí stanice je zařízení určené k dobíjení baterií (akumulátorů) elektromobilů, elektrokol, elektroskútrů a jiných elektrických dopravních prostředků [7]. Je v podstatě obdobou klasické čerpací stanice pro běžná vozidla na benzín nebo naftu. Existuje několik druhů dobíjecích stanic. Hlavní dělení je na běžné dobíjecí stanice a vysoce výkonné (rychlodobíjecí stanice). Dále se dobíjecí stanice dělí z hlediska velikosti na sestavu více stojanů, s jedním stojanem nebo pouze bez stojanu připevněné zařízení na zdi [7].

Existují dva typy nabíjení. AC nabíjení (alternating current – nabíjení střídavým proudem) a DC (direct current – nabíjení stejnosměrným proudem). Jednofázové či třífázové pro nebo pomalé AC nabíjecí stanice (wallbox) poskytují střídavý proud vozidlu, které si ho musí převést na stejnosměrný proud a předat akumulátoru. K tomuto převodu slouží palubní nabíječka a její výkon a konfigurace ve skutečnosti rozhoduje, jak rychle se auto nabije. Pokud je na pomalé AC nabíjecí stanici napsán výkon 22 kW, ale palubní nabíječka elektromobilu má pouze 7,2 kW, bude se vozidlo nabíjet maximálně tímto výkonem [8].

Při nabíjení stejnosměrným proudem může být měnič výrazně větší, protože je umístěn mimo vozidlo. Vzhledem k tomu, že proud je již přeměněn na stejnosměrný v okamžiku, kdy dorazí do vozidla, je možné dodávat větší výkon a rychleji.

Díky této odlišné technice nabíjení mohou stejnosměrné stanice poskytovat až 350 kW výkonu a plně nabít elektromobil za 15 minut (pokud to elektromobil umožňuje). Díky svým schopnostem rychlého nabíjení jsou rychlonabíječky stejnosměrného proudu ideální pro místa s krátkými zastávkami, například nákupní centra, hotely, dálniční odpočívadla nebo v okolí frekventovaných lokalit. Jedna ze známějších výkonných dobíjecích stanic je Tesla Supercharger zobrazena na obrázku č. 1 od společnosti Tesla Inc s výkonem až 350 kW. V tak vysokých výkonech je už limitačním prvkem dobíjecí kabel stanice.

Obrázek 1: Dobíjecí stanice Tesla Supercharger [14]

Dobíjecí stanice lze dobře poznat kvůli výraznému označení. Nejčastěji je stání vedle stanice nabarveno zelenou barvou a označeno dopravní značkou. Zelená barva znázorňuje „čistější" energii. Příkladem takové stanice je stanice v areálu Technické univerzity v Liberci na obrázku č. 2.

Obrázek 2: Dobíjecí stanice CPC50 v areálu TUL

Vizuálně se stanice CPC50 skládá z následujících prvků

- Ovládací panel / HMI
    - HMI je zkratkou pro Human Machine Interface neboli rozhraní člověk-stroj. Jedná se o zařízení, které zprostředkovává komunikaci uživatele s dobíjecí stanicí.
- RFID čtečka
- Tlačítko NOUZOVÉHO ZASTAVENÍ
- Konektor CCS Combo 2 - Phoenix Contact
- Konektor CHAdeMO Sumitomo
- Konektor AC typ 2 pro nabíjení střídavým proudem
- Zamykání, páka pro otevření nabíjecí stanice
- Osvětlení LED (i na každém kabelovém výstupu)

Obrázek 3: Dobíjecí stanice CPC50 detail

Obecná Technická specifikace dobíjecí stanice CPC50 viz tabulka č.1

Tabulka 1: Technické parametry dobíjecí stanice CPC50

Technické parametry dobíjecí stanice CPC50	
Provozní teplota	-30 °C až 50 °C
Skladovací teplota	-40 °C až 85 °C
Relativní vlhkost	5 % až 95 % (bez kondenzace)
Třída krytí IP	IP54 (vnitřní i venkovní použití)
Rozměry (V x Š x H)	1929 mm x 822 mm x 618 mm
Hmotnost	Cca 650 kg
Hladina hluku (při plné zátěži)	<55 dB

Tabulka 2: Připojení stanice CPC50

Vstup AC	
Připojení na hlavní rozvod	3fázové + N + PE
Vstupní napětí	400 V AC ± 10 %
Vstupní proud	3 x 32 A AC – 3 x 150 A AC
Kmitočet	47 Hz – 63 Hz

Stanice má v klidovém režimu spotřebu 130 wattů ale může být vyšší z důvodu, že si udržuje vnitřní provozní teplotu. Tím pádem v zimě provoz stojí více než v létě. Stanice disponuje třemi nabíjecími konektory.

1.1. CCS (Systém kombinovaného nabíjení)

Projekt kombinovaného systému nabíjení (Combined Charging System) byl zahájen v roce 2009. Německý automobilový průmysl se rozhodl pro vývoj vlastní koncepce pro DC nabíjení. Základní myšlenkou bylo vytvořit společné rozhraní pro nabíjení střídavým a stejnosměrným proudem. Iniciativa založená společnostmi Carmec, Audi, Porsche, Daimler, BMW, Volkswagen, Opel a Phoenix Contact vyvinula některé počáteční koncepty. První relevantní návrh byl předložen výborům IEC v roce 2011 [9].

Na obrázku č. 4 je zobrazen CCS konektor. Horní část konektoru obsahuje kontakty pro AC nabíjení + komunikační kontakty a spodní část zásuvky obsahuje kontakty pro DC nabíjení. Toto je jeden z nejdůležitějších aspektů, kterým se CCS liší od CHAdeMO (obě metody nabíjení AC i DC jsou zpracovávány v jedné zásuvce) [9].

Obrázek 4: CCS konektor

1.2. CHAdeMO

Nabíjecí systém CHAdeMO byl vyvinut v Japonsku v roce 2005. Jeho název kombinuje slova „nabíjení“ a „pohyb“ a zhruba znamená „nabíjení pro pohyb“. První dobíjecí stanice byly uvedeny do provozu v roce 2009. Mnoho vozidel, která jsou v současné době k dispozici, např. Mitsubishi i MiEV, Nissan Leaf a Toyota eQ mají rozhraní pro nabíjení CHAdeMO. Technologii využívají především asijští výrobci [9].

Obrázek 5: CHAdeMO konektor

1.3. Typ 2 AC

Typ 2 AC, někdy označovaný jako Mennekes kvůli tomu, že firma Mennekes jako první přišla s návrhem a je jeden z nejvýznamnějších výrobců daného typu.

Obrázek 6: Typ 2 AC konektor

CHAdeMO a CSS jako výstup DC jsou specifikované v následující tabulce. Jedná se o maximální hodnoty. Stanice může nabíjet 125 A pokud stanice nepřekročí 50 kW výkon.

Tabulka 3: vlastnosti konektoru CSS A CHAdeMO

Konektor 1: CSS a Konektor 2: CHAdeMO	
Výkon	50 kW
Výstupní napětí	850 V DC
Výstupní proud	125 A
Faktor výkonu při (50% zatížení)	> 0.97
Účinnost	> 94%

Stanice obsahuje modem, který umožňuje připojit stanici pomocí ethernetu nebo GSM do internetu. Stanice (obrázek č. 3), která je nainstalována v areálu TUL, je připojená přes ethernet přímo do LIANE, kde má přidělenou statickou IP adresu a hostname. Přes LIANE má povolenou možnost připojení do internetu.

Stanice umožňuje zprostředkování komunikace pomocí protokolu **OCPP v 1.5**. a přípravu na OCPP 1.6 a 2.0. To znamená, v době, kdy byla stanice uvedena do provozu, tak stávající hardware umožnil pouze připojení pomocí OCPP verze 1.5. V dnešní době by se musel obměnit hardware, aby stanice umožnila připojení pomocí 1.6 a vyšších verzí.

Nabíjení pomocí typů konektoru je dané regionem a výskytem dominantnějšího výrobce elektromobilů. Jestli je nějaký výrobce elektromobilů dominantním výrobcem, tak si určí (nebo se domluví), co se bude používat. Na obrázku č. 7 jsou sepsány konektory a typy nabíjení podle regionu, kde je největší zastoupení daného konektoru.

Typ nabíjení a název konektoru	Region			
	Japonsko	Čína	Amerika	Evropa
AC				
Název	Type 1 - J1772	GB/T	Type 1 - J1772	Type 2
DC				
Název	CHAdeMO	GB/T	CCS - Type 1	CCS - Type 2

Obrázek 7: Nabíjecí konektor podle regionu

2. Použité technologieXML

E**x**tensible **M**arkup **L**anguage zkratkou XML, rozšiřitelný značkovací jazyk, který je kompatibilní s řadou programovacích jazyků a nástrojů. Díky tomu se hodí převážně pro výměnu dat mezi různými aplikacemi. Dokument XML popisuje strukturu předávaných dat. Data jsou označena v souboru jednotlivými tagy [1]. Každý XML soubor obsahuje hlavičku, která obsahuje verzi a kódování a root element. Dokument může vypadat viz. zdrojový kód 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<flowers>
 <flower>
  <name>sunflower</name>
  <color>yellow</color>
 </flower>
<flower>
 <name>rose</name>
 <color>red</color>
 </flower>
</flowers>
```

Zdrojový kód 1: Ukázka XML souboru

Formát se nejčastěji používá pro výměnu dat po internetu. Jednotlivé tagy dávají datům strukturu. Strukturou je soubor podobný HTML, až na to, že nemá předdefinované tagy.

Pravidla pro XML soubor

- obsah tagu (element) musí začínat startovním tagem ve tvaru <jmeno_tagu>
- konec elementu je značeno takzvaným koncovým (ukončovacím) tagem </jmeno_tagu>, který je povinný!
- před ukončením elementu musí být ukončeny všechny jeho vnitřní elementy – každý element tedy musí mít koncový tag uvnitř stejného elementu jako počáteční [2].

- prázdný tag je značen <jmeno_tagu/>
- počáteční tag může obsahovat atributy v podobě <jmeno_tagu atribut1=„aaa“ atrribut2=„abc“>. Atributy se mohou psát i dovnitř tagu <jmeno_tagu>< jmeno_tagu:atribut1=“aaa“/></jmeno_tagu>
- nejvyšší element v hierarchii elementů je nazýván „root element“ a smí být pouze jeden [2].

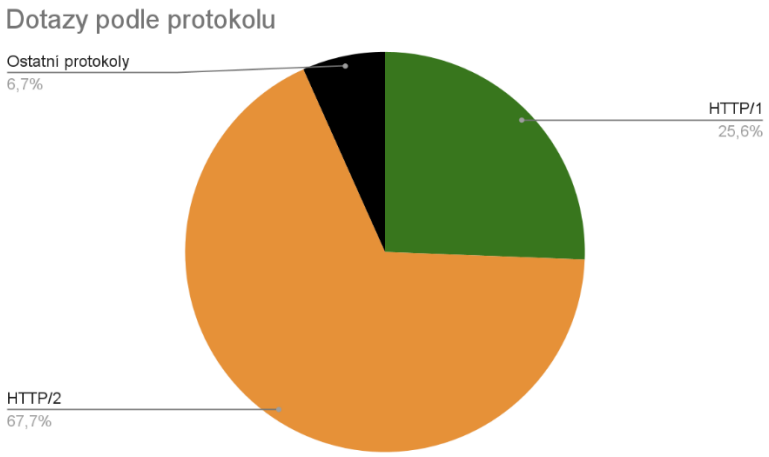
Ze zdrojového kódu 1 lze vygenerovat C# datové třídy zdrojový kód 2 pomocí online konvertorů např. *https://jsonformatter.org/*.

```
public class Flower
{
    public string Name { get; set;}
    public string Color { get; set;}
}
public class Flowers
{
    public List<Flower> Flower { get; set;}
}
```

Zdrojový kód 2: Ukázka C# třídy vygenerované z XML souboru

2.2. HTTP

Hyper **T**ext **T**ransfer **P**rotocol je jednoduchý transportní aplikační protokol, který se využívá nejen k přenosu hypertextových dokumentů a obrázků.



Graf č.1: Zastoupení protokolů [16]

Podle dat z roku 2021 na grafu č. 1 je jeho nejvíce využívaná verze 2.0. Základní princip protokolu je komunikace request – response (dotaz – odpověď). Komunikaci vždy začíná klient. Po zaslání dotazu server vrátí odpověď se stavovým kódem. Kódy jsou rozděleny do 5 skupin. Jedná se o seskupení odpovědí, které mají podobný nebo příbuzný význam. Znalost toho, co to je, vám může pomoci rychle určit obecnou podstatu stavového kódu, než začnete hledat jeho konkrétní význam. Informační odpovědi jsou v rozmezí 100-199. Říkají, že požadavek byl přijat, bylo mu porozuměno a žádost se dále vykonává.

Tabulka 4: Informační HTTP odpovědi [13]

Informační HTTP odpovědi	
Kód	Název
100	Continue
101	Switching Protocols
103	Early Hints

Stavové kódy pro indikaci úspěchu jsou v rozmezí 200-299. Rozumí se tím, že požadavek klienta byl přijat, pochopen a zpracován serverem.

Tabulka 5: Úspěšné HTTP odpovědi [13]

Úspěšné HTTP odpovědi	
Kód	Název
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content

Stavové kódy zaměřující se přesměrováním požadavku jsou v rozmezí 300-399. Kód označuje, že požadavek má více než jednu možnou odpověď. Klient nebo uživatel by si měl vybrat právě jednu z nich.

Tabulka 6: Přesměrovací HTTP odpovědi [13]

Přesměrovací HTTP odpovědi	
Kód	Název
300	Multiple Choice
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect

Stavové kódy zaměřující se chybou ze strany klienta mají rozmezí 400-499 a kategorizují chybu, kterou zavinil klient. Asi nejznámější je 404 Not found. Všichni se s ní už určitě setkali. Vzniká, když se klient odkazuje na neexistující adresu.

Tabulka 7: Chybové HTTP odpovědi klienta [13]

Chybové HTTP odpovědi klienta	
Kód	Název
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Alowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed

Stavové kódy zaměřující se chybou ze strany serveru mají přidělené rozmezí 500-599. Chyba nastane tehdy, když požadavek od klienta byl přijat serverem, ale během vyhodnocování se objeví neočekávaná chyba.

Tabulka 8: Chybové HTTP odpovědi serveru [13]

Chybové HTTP odpovědi serveru	
Kód	Název
500	Internal Serve Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

2.3. URL

Uniform **R**esource **L**ocator („jednotný lokátor zdroje“), běžně webová adresa, je řetězec znaků, který slouží k přesné specifikaci umístění zdrojů informací na internetu. Nejběžnějším zdrojem je webová stránka [3].

Příkladem může být následující formát:

```
protokol://server.doména:port/cesta/název?dotaz=parametr
```

2.4. Metody protokolu HTPP

2.4.1. OPTIONS

Metoda OPTIONS představuje dotaz na možnosti komunikace spojené s uvedeným URL. Metoda umožňuje klientovi určit možnosti a omezení spojené se zdrojem nebo schopnostmi serveru. Pokud je URL v dotazu ve tvaru "*", pak se jedná o dotaz na možnosti serveru jako celku.

2.4.2. GET

Metoda GET představuje požadavek na zaslání dokumentu určeného pomocí URL [4]. V souvislosti s proxy se může metoda GET změnit na "podmíněný GET", která požaduje poslat dokument pouze za určitých podmínek definovaných v hlavičce dotazu.

2.4.3. HEAD

HEAD metoda je identická s metodou GET, server však nemusí posílat tělo odpovědi. Metodu je možné použít k získání doplňkových informací o dokumentu, často se používá k testování hypertextových linek, jejich dostupnosti a poslední modifikace. Klient může získané hlavičky analyzovat a případně požádat o data novým dotazem GET (např. test, zda dokument není příliš dlouhý).

2.4.4. POST

POST metoda se používá v případě, kdy má cílový server přijmout data z požadavku. Skutečná funkce metody závisí na URL s ní spojené. [4] Výsledkem POST metody může být poslání e-mailu, předání dat do procesu, který data zpracuje, rozšíření databáze. Posílaná data nejsou nijak omezená a je možné v hlavičkách tělo zprávy popsat.

2.4.5. PUT

PUT metoda představuje požadavek na uložení zaslaných dat pod specifikované URL na server. Takto uložená data budou dostupná např. následnými dotazy GET. Metoda PUT předpokládá, že uložení dat do souboru na server provádí přímo server, nikoli externí aplikace [4].

2.4.6. DELETE

Požadavek na odstranění dokumentu na serveru. Odstraněný dokument je specifikován v URL.

2.4.7. TRACE

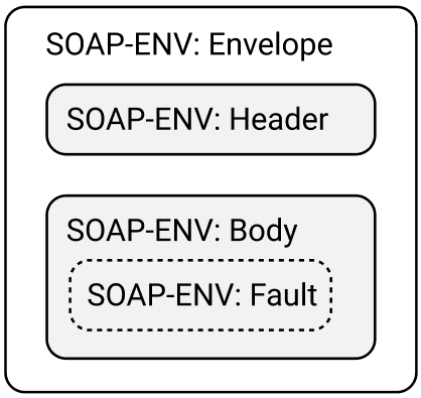
Metoda použitá k testování originálního serveru. Originální server má vrátit klientovi kladnou odpověď bez dat [4].

2.5. SOAP

Simple **O**bject **A**ccess **P**rotocol je komunikační protokol založený na bázi značkovacího jazyka XML určený pro komunikaci na internetu. Může rozšiřovat HTTP o zprávy v XML. Protokol SOAP není závislý na programovacím jazyku nebo platformě, kde se používá.

Přestože SOAP lze použít v různých systémech zasílání zpráv a lze jej doručit prostřednictvím různých přenosových protokolů, prvotním cílem SOAP je vzdálené volání procedur (endpointů) přenášené prostřednictvím HTTP.

Endpointy jsou funkce, dostupné skrze API, které provádí několik akcí [6]. Příklad akce může například být získávání dat z databáze nebo vytvoření nového záznamu v databázi. Mohli bychom říci, že endpoint spouští proceduru, která provádí určitý úkol. Tyto endpointy jsou závislé na předaných parametrech [6].



Obrázek 8: SOAP envelope (obálka)

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAPENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
    </SOAP-ENV:Fault>
```

```
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Zdrojový kód 3: Ukázka SOAP souboru [15]

Zpráva SOAP je dokument XML obsahující prvky, které jsou definované v obrázku č.3. Tag envelope je povinný root element daného souboru. Envelope označuje začátek a konec zprávy, takže příjemce ví, kdy byla přijata celá zpráva. Řeší problém, kdy je zpráva zcela odeslána a je připravená na další zpracování. Jedná se typ obalového mechanismu. Header je nepovinný parametr, ve kterém se dají specifikovat další volitelné parametry pro volání endpointu. Body je parametr, ve kterém jsou data. Podle dat v tomto tagu se vykoná procedura.

```
<?xml version="1.0"?>
<SOAP:Envelope
xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body xmlns:data="[xml předpis url]">
<data:GetTemperatureTownRequest>
  <data:Town>Liberec</m:Town>
  <data:Date>2022-02-06</m:Date>
</data:GetTemperatureTownRequest>
  </SOAP:Body>
</SOAP:Envelope>
```

Zdrojový kód 4: Příklad SOAP request (požadavek)

Zdrojový kód č. 5 je příklad odpovědi na Zdrojový kód č. 4 v požadavku je zadán datum a město. A v odpovědi o dotazované údaje navíc teplota.

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<SOAP:Envelope
xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Body xmlns:data="[xml předpis url]">
    <data:GetTemperatureTownResponse>
      <data:Celsius>3</m:Celsius>
      <data:Town>Liberec</m:Town>
      <data:Date>2022-02-06</m:Date>
    </data:GetTemperatureTownResponse>
  </SOAP:Body>
</SOAP:Envelope>
```

Zdrojový kód 5: Příklad SOAP response (odpověd)

Volitelný prvek fault. Fault je chyba, která je způsobena nesprávným formátem zprávy, problémy se zpracováním záhlaví nebo nekompatibilitou.

3. API

Application **P**rogramming **I**nterface („programovatelné aplikační rozhraní"). Rozhraní API je jako jídelní lístek v restauraci. Jídelní lístek obsahuje seznam jídel, která si lze objednat a popis každého jídla. Když vybereme ty položky, co chceme, kuchyň restaurace odvede práci a poskytne hotové pokrmy. Nevíme přesně, jak restaurace toto jídlo připravuje, a ani to není potřeba. Důležitý je výsledek od zadaného požadavku.

V kontextu API můžeme říct, že API obsahuje instrukce a implementaci procedur, které něco vykonávají. Předpis datových tříd a názvy metod mohou být popsány například pomocí WSDL formátu.

WSDL je notace XML pro popis webové služby. Definice WSDL říká klientovi, jak sestavit požadavek na webovou službu. Dále popisuje rozhraní, které poskytuje poskytovatel webové služby. Definice WSDL je rozdělena na samostatné části, které specifikují logické rozhraní a fyzické detaily webové služby. Fyzické detaily zahrnují jak informace o proceduře (jako je číslo portu HTTP), tak informace o vazbě, které určují, jak je popsán SOAP a jaký transportní protokol je použit. Při importu nebo generování WSDL se WSDL ověřuje podle základního profilu WS-I. Před nasazením aplikace, knihovny nebo sady zpráv je nutné chyby validace opravit. Varování o validaci nebrání nasazení, ale mohou indikovat potenciální problémy s interoperabilitou. Validovaný WSDL se stává nedílnou součástí aplikace, knihovny nebo sady zpráv [6]. Generování tříd z notace (pospáno v příkladu) WSDL je možné pomocí svcutil.exe, kde MerchantService.wsdl je zdrojový soubor. Přepínač /l je požadovaný jazyk výstupu, přepínač /o je cesta k nově vytvořenému souboru a /n je základní namespace, pod kterou bude třída vygenerována. Výsledná třída obsahuje rozhraní (interface) s metodami, které jsou popsány v daném WSDL souboru. Pro programátora poté zbývá napsat implementaci daného rozhraní ve své aplikaci.

```
svcutil.exe D:\MerchantService.wsdl /t:code /l:c#
  /o:"D:\MerchantService.cs" /n:*,NamespaceName
```

Interface neboli rozhraní je předpis skupiny vlastností nějakého objektu nebo třídy, kde jsou definovány názvy metod se vstupními a výstupními parametry. Příkladem může být rozhraní pásový dopravník (zdrojový kód č. 6).

```
public interface IPasovyDopravnik
{
    void Start();
    void Stop();
    void ZvolitSmer(bool smer);
}
```

Zdrojový kód 6: Ukázka C# rozhraní popisující pásový dopravník

Dané rozhraní můžeme použít ve třídě zdrojového kódu č. 7

```
public class Eskalator :  IPasovyDopravnik
{
    public void Start()
    {
        //implementace zapnutí
    }
    public void Stop()
    {
        //implementace vypnutí
    }
    public void ZvolitSmer(bool smer)
    {
  //implementace změny směru
    }
}
```

Zdrojový kód 7: Ukázka C# Třídy s implementací rozhraní IPasovyDopravnik

Implementace nemusí končit třídou `Eskalator` můžeme mít třídu `Travelator`, která bude mít také vlastnosti `IPasovyDopravnik`, ale jinou implementaci daného rozhraní. Definovat rozhraní je žádoucí, když se bude jednat o více objektů s podobnými vlastnostmi.

4. Protokol OCPP

Protokol **O**pen **C**harge **P**oint **P**rotocol původně vznikl iniciativou nizozemské firmy E-laad. Nejprve jako OCPPF, kde f je zkratka pro fórum. Později se přidaly firmy Greenlots (Severní Amerika) a ESB (Irsko), které společně s E-laad založili OCA (Open Charge Aliance). OCA je společenství firem, které udržují a rozvíjí protokol OCPP. Hlavně zachovávají původní vizi: dále rozvíjet OCPP, aby řídil otevřené a flexibilní sítě pro elektromobily po celém světě.

OCPP je komunikační protokol pro komunikaci dobíjecí stanice a centrálního systému. Jsou zde popsány metody pro komunikaci, například o ovládání periferií dobíjecí stanice podle předdefinované logiky. Logika komunikace se postupně vyvíjela s verzemi protokolu. První stabilní verze vypuštěná do ostrého provozu byla verze 1.5. V dnešní době se považuje za funkční pouze s verzí firmware.

4.1. OCPP 1.5

Celkem je ve verzi 1.5 25 operací. Z nich je 10 iniciováno dobíjecí stanicí a 15 centrálním systémem.

Operace iniciovány dobíjecí stanicí:

Authorize, Boot Notification, Data Transfer, Diagnostics Status Notification, Firmware Status Notification, Heartbeat, Meter Values, Start Transaction, Status Notification [10] a Stop Transaction [10].

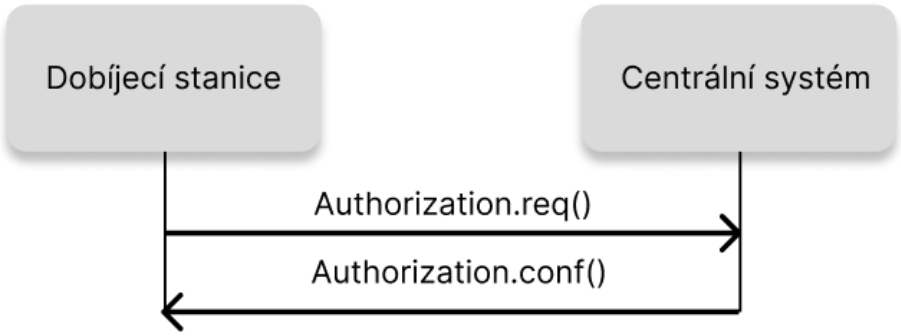
Operace iniciovány centrálním systémem:

Cancel Reservation, Change Availability, Change Configuration, Clear Cache, Data Transfer, Get Configuration, Get Diagnostics, Get Local List Version, Remote Start Transaction, Remote Stop Transaction, Reserve Now, Reset, Send Local List, Unlock Connector [10] a Update Firmware [10].

Komunikace mezi dobíjecí stanicí a centrálním systémem vždy začíná požadavkem. Ve specifikaci OCPP je to popsáno pomocí operationname.req(). Příjemce zprávy vždy odpoví potvrzením. Ve specifikaci OCPP je potvrzení popsáno jako operationname.conf().

Obrázek č. 9 ukazuje komunikační tok pro operaci Authorize. Je z něj patrné, že dobíjecí stanice odešle centrálnímu systému zprávu Authorization.req() a obdrží odpověď ve tvaru

Authorization.conf(). Pro přehlednost jsou uvedeny názvy zpráv a zbytek komunikace [10].



Obrázek 9: Komunikace pro autorizaci dobíjecí stanice s centrálním systémem

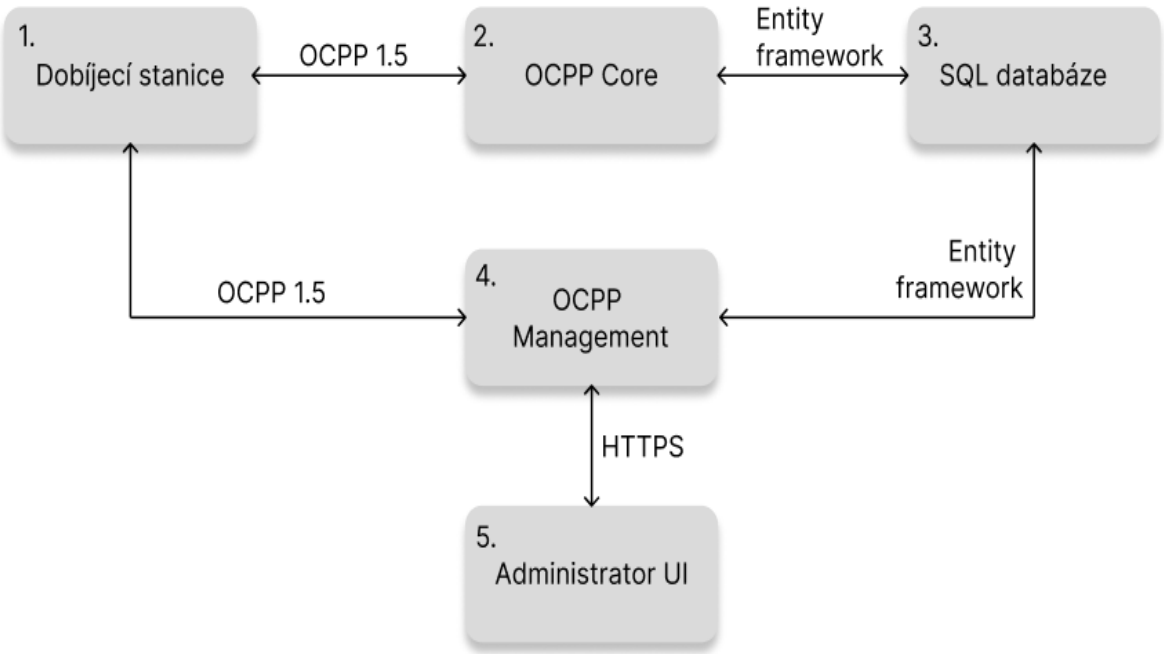
5. Praktická část

5.1. Návrh řešení

Celý systém je složen z nabíjecí stanice a čtyřech k ní přidružených částí, které jsou propojeny vazbami. Vazby jsou znázorněny na obrázku č. 10. Do backendové části spadá blok 2, 3 a 4. Systém využívá k ukládání dat SQL databázi. OCPP Core a OCPP Management pomocí entity frameworku přistupují k databázi.

Entity Framework je moderní objekt – Mapovač databáze pro .NET. Podporuje dotazy LINQ, sledování změn, aktualizace a migrace schématu. EF Core funguje s mnoha databázemi, včetně SQL Database [11].

Rozdělení na právě 4+1 části je z důvodu dostupnosti daných systémů. Rozdělení úrovní pomáhá stanovit prioritu v dostupnosti systémů. V případě nefunkčnosti OCPP Management a Administrator UI se nic neprojeví na chodu dobíjecí stanice. V případě OCPP Core a databáze je to horší. Tyto služby mají největší prioritu, tím pádem větší důraz na dostupnost služeb.



Obrázek 10: Návrh řešení systému

V rámci zachování integrity systémů OCPP Core a SQL je potřeba vybrat hosting (cloud) s co nejvyšším SLA viz. tabulka č. 6. Pro OCPP Management a Administrator UI, které budou využívány méně není potřeba tak vysoké SLA.

Tabulka 9: Přehled dostupnosti služeb

Dostupnost služeb (SLA)			
SLA	Odstávky za týden	Odstávky za měsíc	Odstávky za rok
99 %	**1,68** hodin	**7,2** hodin	**3,65** dní
99,9 %	**10,1** minut	**43,2** minut	**8,76** hodin
99,95 %	**5** minut	**21,6** minut	**4,38** hodin
99,99 %	**1,01** minut	**4,32** minut	**52,56** hodin
99,999 %	**6** sekund	**25,9** sekund	**5,26** minut

Řešení bylo nasazeno a odladěno na dedikovaném počítači. Server s operačním systémem Windows 10, 8Gb RAM a 200Gb úložným prostorem. Server měl dostatečný výkon a splnil potřebnou roli v celém průběhu vývoje systému.

Obecné zabezpečení je řešeno firewallem, komunikaci po protokolu HTTPS a omezením přístupu pouze z lokální sítě. SQL server je zabezpečen přes tzv. Authentication Mode – pro přístup služeb je potřeba mít zohledněné heslo a jméno v rámci konfigurace připojení SQL.

Další možná zabezpečovací metody může být „allowed hosts“ v konfiguračním souboru appsettings.json. U jednotlivých API, se specifikuje, z jaké adresy může na API klient přistupovat.

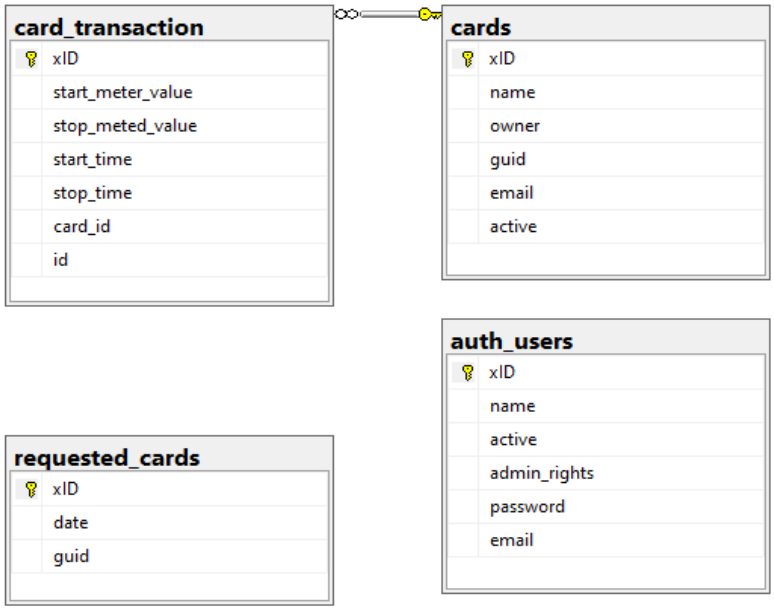
V OCPP Core pouze adresa stanice. A v OCPP Management adresy klientů.

```
//povolení všech
"AllowedHosts": "*",
//NEBO povolení z localhostu a ze specifické adresy
"AllowedHosts": "localhost;192.168.1.0",
```

Zdrojový kód 8: Nastavení přístupů do API

5.2. Databáze

Databázové schéma se skládá ze 4 tabulek viz obrázek č. 11.



Obrázek 11: Databázové schéma

Relace je u tabulky cards na tabulku card_transaction. Jedná se o transakci zahájenou danou aktivní kartou. V tabulce requested_cards jsou uloženy všechny neznámé karty přiložené na RFID čtečku stanice. Tabulka zde plní funkci databáze scanneru, který načte ID uložený v RFID kartě. V auth_users jsou uloženy uživatelé celého systému. V systému se rozlišují 2 typy uživatelů ADMIN a host. Databáze je zabezpečena přes Authentication Mode (autentizační mód) a do databáze mohou přistupovat jen databázoví uživatelé definovaní v SQL serveru.

5.2.1. Návrh na zlepšení

Přidáním jedné tabulky charging_station by se řešení rozšířilo na N možných dobíjecích stanic. K transakci by se ukládal cizí klíč z tabulky charging_station. Ale v rámci stanice TUL to není potřeba.

5.3. OCPP Core

OCPP Core je hlavním funkčním blokem systému. Jedná se o SOAP API psané v jazyce C#, ve kterém jsou implementovány metody iniciované dobíjecí stanicí. Rozhraní bylo vytvořeno z WSDL souboru. V lokální paměti si API udržuje veškeré informace, které stanice do centrálního systému posílá a v pravidelném nastavitelném intervalu je loguje. Do databáze ukládá jen část. Všechny neznámé karty přiložené k RFID čtečce jsou uloženy do databáze s aktuální časovou známkou. Viz zdrojový kód č.8.

```
private void SaveUnknownCard(string idtag)
{
  _ctx.RequestedCards.Add(new() {
    Date = DateTime.Now,
      Guid = idtag
  });
  _ctx.SaveChanges();
}
private Card IsTagInAuthDatabase(string idtag)
{
  var card = _ctx.Cards.SingleOrDefault(x => x.Guid == idtag);
  if (card != null) return card;
  else
    SaveUnknownCard(idtag);
  return null;
}
```

Zdrojový kód 9: Ukázka C# metody obsluhující kontrolu karet vůči databázi

Metoda IsTagInAuthDatabase je volána z metody (Authorization) implementované rozhraním SOAP API. Jedná se o hlavní ověření vůči databázi.

OCPP Core si při startu definuje výchozí id první transakce. Když v databázi existuje již nějaký záznam transakce, tak použije id posledního záznamu inkrementovaného o 1. Tímto je vyřešena konzistence id transakce ve vnitřní paměti OCPP a SQL databáze.

V případě, když je OCPP Core mimo provoz tak stanice dostane chybový kód a nepovolí dobíjení, ale pokouší se stále autentizovat do té doby, než se to povede. Bylo provedeno několik testovacích scénářů. Provedené testovací scénáře jsou popsány v tabulce č. 7. Až na akci č. 2 vše dopadlo dle očekávání.

Tabulka č. 10: Testovací scénář

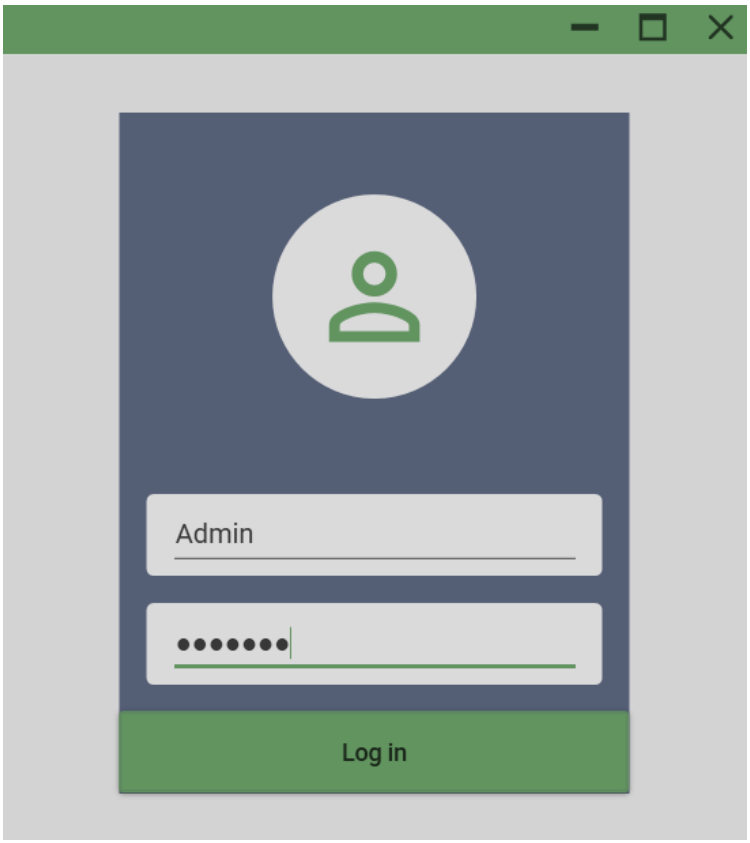
Dostupnost služby OCPP Core				
Akce	**Paramenty**	**Očekávaný výsledek**	**Reálný výsledek**	**Sum**
Autentizace stanice **#1**	Zapnutí stanice	Pokus o navázání autentizace	Pokus o navázání autentizace	OK
Autentizace stanice **#2**	Odpojení z LAN/Vypnutí služby	Opakovaně se pokoušet dotazovat na danou službu (log chyby na UI)	Na HMI stanice není žádná indikace problému	NOK
Autentizace stanice **#3**	Nastavení špatné adresy OCCP	Nepřipojit se	Nepřipojit se	OK
Autentizace stanice **#4**	Nastavení správné adresy	Připojit se	Připojit se	OK
Autentizace stanice **#5**	Propadlý HTTPS certifikát	Nepřipojit se	Nepřipojit se	OK
Přiložení neznámé karty **#6**	RFID karta co není v systému	Uložit do databáze a zamítnout autorizaci	Uložit do databáze a zamítnout autorizaci	OK
Přiložení aktivní karty **#7**	Známá aktivní RFID karta	Povolit dobíjení	Povolit dobíjení	OK
Přiložení neaktivní karty **#8**	Známá neaktivní RFID karta	Nepovolit dobíjení	Nepovolit dobíjení	OK
Zahájit dobíjecí cyklus **#9**	Zahájení aktivní kartou	Umožnit výběr konektoru a začít dobíjecí cyklus	Umožnit výběr konektoru a začít dobíjecí cyklus	OK
Ukončit dobíjecí cyklus **#10**	Zahájení stejnou kartou (z #9)	Ukončit dobíjení	Ukončit dobíjení	OK
Ukončit dobíjecí cyklus **#11**	Ukončení různou kartou od karty, která	Neukončit stávající. Nabídnout nové nabíjení jiným konektorem.	Neukončit stávající. Nabídnout nové nabíjení jiným konektorem.	OK
Ukončit dobíjecí cyklus **#12**	Nedostupný OCPP Core	Umožnit ukončení a v dostupnosti Core zapsat ukončení.	Umožnit ukončení a v dostupnosti Core zapsat ukončení.	OK
Násilné odpojení Typ 2 **#13**	Aktivní nabíjení Typ 2 AC	Uložit dobíjecí cyklus do DB	Uložit dobíjecí cyklus do DB	OK
Log chyby nedostupnosti SQL **#14**	SQL nedostupné	Log chyby + data vstupu	Log chyby + data vstupu	OK
Po restartu Core načíst ID #15	Po restartu načíst z DB ID + 1.	Načíst ID + 1	Načíst ID + 1	OK

5.4. OCPP Management

OCPP Management je mezičlánek mezi databází a uživatelským rozhraním. Jedná se o API psané v C# ASP.NET verze 6.0. Aplikační rozhraní implementuje koncové body ve čtyřech částí.

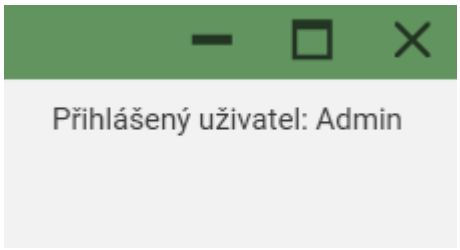
5.5. Autorizační část

Autorizace je pomocí uživatelského jména a hesla. Hesla jsou v databázi (Obrázek č. 11 tabulka auth_users) uložená pomocí hashovacího algoritmu (MD5) nikoliv reálná podoba hesla. Uživatelské účty se dělí na dva druhy. Administrátor a normální uživatel. Administrátor má veškerá práva a normální uživatel má pouze čtení. V aplikaci to znamená, že uživatel může pouze zobrazit historii dobíjení.



Obrázek 12: Obrazovka přihlášení

V pravém horním rohu je zobrazen aktuálně přihlášený uživatel. Po najetí kurzorem se zobrazí, jestli má nebo nemá administrátorské oprávnění.



Obrázek 13: Obrazovka přihlášení detail uživatele

5.6. Část na správu karet

V této části jsou implementovány CRUD operace (Vytvoření, čtení, aktualizace a odstranění) a poslední přiložené karty k RFID čtečce. Jednotlivé metody mají přiřazené atributy vycházející z kapitoly 2.4. Atributy k metodám dané třídy se píšou do hranatých závorek, jak je znázorněno na atributu HttpPut na zdrojovém kódu č. 10.

```
[HttpPut("card/addcard")]
public async Task<AddCardResponse> AddCard
([FromBody] AddCardRequest requset)
{
.
.
.
}
```

Zdrojový kód 10: Atributy v metodě

Databázové datové třídy jsou vygenerované takzvaným scaffoldem. Scaffold databáze usnadňuje práci vývojářům, protože vygeneruje třídy odpovídající schématu databáze. Po inicializaci databázového kontextu při startu API se do databáze jednoduše přistupuje přes LINQ dotazy. Příklad použití kontextu je ve zdrojovém kódu č. 9 (metoda IsTagInAuthDatabase).

Klient by nikdy neměl vidět databázové modely. Při posílání dat z API klientovi je vždy potřeba převést databázový model na DTO. Pro takový převod se používají různé převodníky. Příkladem takového převodníku je AutoMapper. Nebo napsáním statického

převodníku (rozšiřující metody), tento postup je využitý ve všech částí systému, kde to má smysl. V OCCP Management jsou rozšiřující metody v souboru StaticConverters.cs. Rozšiřující metody umožňují rozšířit existující typ o nové metody, aniž by se změnila definice původního typu. Rozšiřující metoda je statická metoda statické třídy, kde je na první parametr aplikován modifikátor this. Typ prvního parametru bude typ, který je rozšířen. [12] Příklad rozšiřující metody je ve zdrojovém kódu č. 11 a použití ve zdrojovém kódu č. 12. Jedná se o koncový bod, který vrací všechny karty v databázi.

```
public static CardDTO Convert(this Card card)
{
    //kontrola validních dat
    if (card == null) return null;
    else return new()
    {
        Email = card.Email,
        Guid = card.Guid,
        Name = card.Name,
        Owner = card.Owner,
        IsActive = card.Active,
        Id = card.XId
    };
}
```

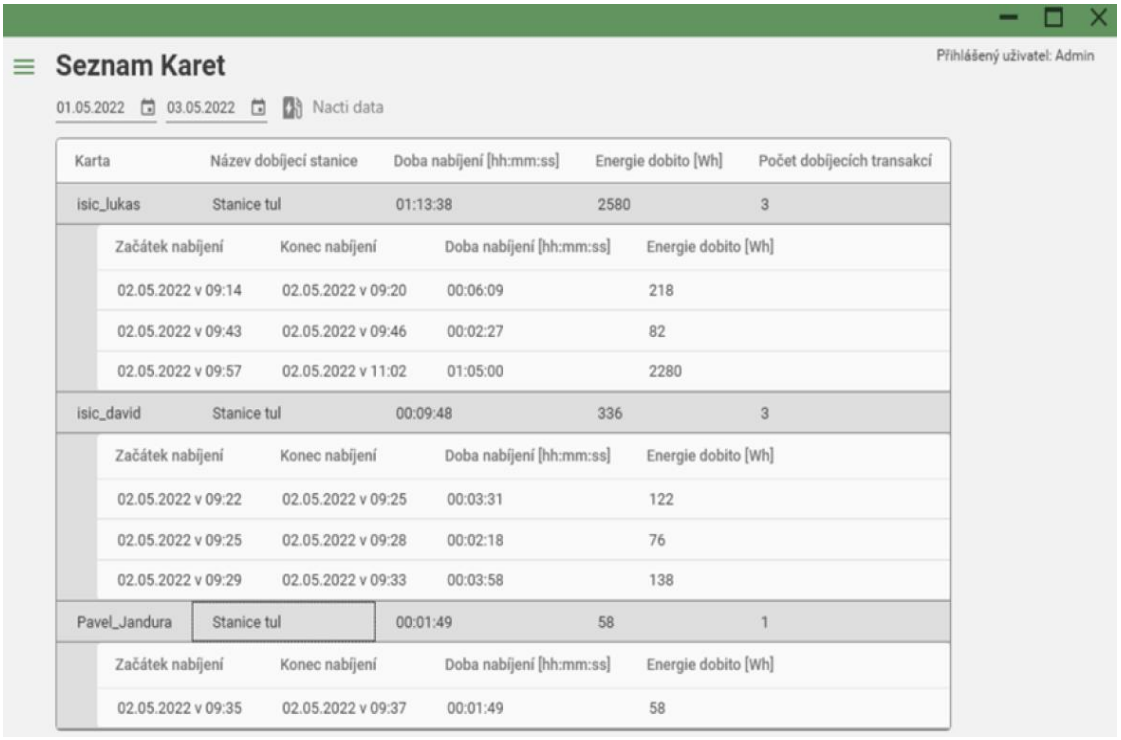
Zdrojový kód 11: Rozšiřující metoda

```
[HttpGet("card/getcards")]
public async Task<GetAllCardsResponse> GetAllCards()
{
    var cards = _ctx.Cards.ToList();
    if(!cards.Any()) return new();
    return new()
    {
        //použití metody Select pro zkonvertování všech záznamů
        Cards = cards.Select(x => x.Convert()).ToList()
    };
}
```

Zdrojový kód 12: Použití rozšiřující metody Convert

5.7. Transakční část

V transakční části je koncový bod, který vrací seznam karet a k nim navázané transakce (nabíjení) v zadaném období. V Administrator UI pak lze zobrazit výstup, který je znázorněn na obrázku č. 14. Nejdůležitějším aspektem bylo přehlednost zobrazovaných dat. Zobrazení je rozděleno na transakce seskupené podle karty a množství. V detailu každé karty jsou pak jednotlivé transakce.



Obrázek 14: Obrazovka transakcí

5.8. OCPP část

V této části jsou v rozhraní vytvořeny metody OCPP, které inicializuje centrální systém. Rozhraní je vygenerováno z oficiálního WSDL souboru. Ve zdrojovém kódu č. 11 je implementována metoda reset stanice. Využívá se zde _server, který je pomocí vkládání závislostí přiřazen v konstruktoru třídy. Server obsahuje třídu Client, kde jsou implementované metody OCPP inicializované centrálním systémem.

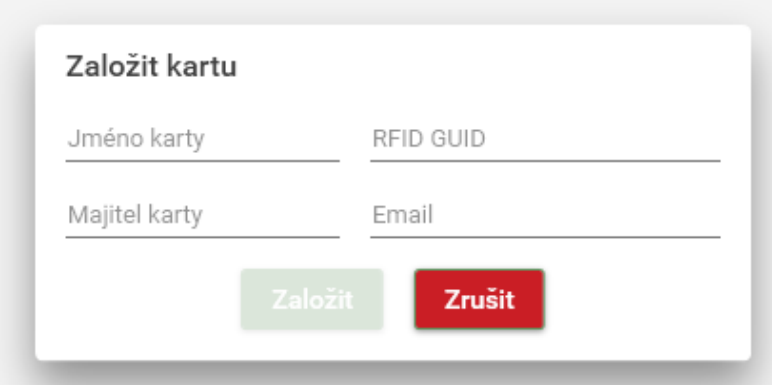
```
[HttpPost("station/reset")]
public async Task<ResetChargeBoxResponse>
ResetStation([FromBody] ResetChargeBoxReq req)
{
    var resetType = req.HardReset ? ResetType.Hard :
    ResetType.Soft;
    var res = await _server.Client.ResetAsync(req.ChargeBoxId,
    resetType);
    string message = res.status == ResetStatus.Accepted ?
    "Reset    OK":
    "Reset NOK";
    _logger.LogInformation(message);
    return new()
    {
        datum = DateTime.Now,
        message = message
    };
}
```

Zdrojový kód 13: Metoda reset stanice

5.9. Administrátor UI

Administrátor UI je klientská desktopová aplikace programovaná v jazyce C# WPF. V aplikaci se zaměřuji na moderní vzhled a využívání osvědčených postupů vývoje aplikací pro windows. Používá se zde grafická knihovna Material Design In XAML, která je podle mých zkušeností nedílnou součástí jakékoliv nové WPF aplikace. V aplikaci je HTTP klient pro komunikaci s OCPP Management. Pro vytvoření rozhraní na komunikaci bylo využito NSwagStudio. NSwagStudio je otevřený nástroj (https://github.com/RicoSuter/NSwag) na generování klienta z referenčního souboru API. Vygeneruje skoro vše potřebné na komunikaci klient server. Normální uživatel nemá v menu (obrázek č. 17) možnost zobrazení „Správa karet“ a „Poslední karty“.

5.9.1. Další obrazovky aplikace



Obrázek 15: Založení nové karty

Obrázek 16: Historie přiložených karet

Obrázek 17: Navigační menu

6. Závěr

Cílem této bakalářské práce bylo navrhnout a realizovat backend pro dobíjecí stanici pomocí komunikačního protokolu OCPP verze 1.5. Dle výstupů je tento cíl splněn. Kromě základních částí backendu bylo realizováno uživatelské rozhraní (desktopová aplikace).

Aplikace umožňuje autorizovaným uživatelům správu identifikačních karet a zobrazení dobíjení ve vybraném termínu. V rámci práce byl splněn požadavek správce stanice na vytvoření dané funkcionality pro dobíjecí stanici TUL. Aplikace umožňuje administrátorovi zobrazit poslední neznámé karty přiložené na čtečku RFID a přidat je do databáze autorizovaných karet. Karty lze aktivovat nebo deaktivovat pomocí administrátorského přístupu.

Největším přínosem této bakalářské práce byl takzvaný „proof of concept“, který potvrdil možnosti, jaké dobíjecí stanice v rámci OCPP nabízí. Po 4 letech se vyřešila otázka autentizace vůči stanici v rámci organizace.

Bylo zapotřebí podniknou hodně kroků. Od přehrání firmwaru, nastavení stanice, zavedení do LIANE, povolení firewallů, nastavení serveru, vývoj a implementace řešení, nasazení do ostrého provozu a odladění uživatelských připomínek.

Do budoucna je plánován dovývoj desktopové aplikace. Přidat veškeré funkce protokolu OCPP verze 1.5, popřípadě vytvořit mobilní aplikaci. OCPP Management lze předělat ze stávajícího REST API do API využívající ke komunikaci WebSocket nebo SignalR. V případě této změny by byl uživatel notifikován v reálném čase od serveru a nemusel by si o data žádat sám.

7. Zdroje

[1] Co je to XML, k čemu se užívá a jaké jsou jeho výhody? [online]. © 2002-2021 NetDirect. [cit. 2022-02-01]. Dostupné z: https://www.fastcentrik.cz/blog/co-je-to-xml,-k-cemu-se-uziva-a-jake-jsou-jeho-vyh

[2] XML – Úvod [online]. © 1998-2022 Internet Info s.r.o. [cit. 2022-12-01]. Dostupné z: https://www.root.cz/clanky/xml-uvod/

[3] Uniform Resource Locator [online] [cit. 2022-26-01]. Dostupné z: https://cs.wikipedia.org/wiki/Uniform_Resource_Locator

[4] HTTP protokol – požadavky a odpovědi [online] [cit. 2022-26-01]. Dostupné z: http://http.stylove.com/

[5] 2. díl: Jak pracovat s REST API ve WordPressu? Návod pro začátečníky [online]. © 2022 MasterDC [cit. 2022-01-02]. Dostupné z: https://www.master.cz/blog/wordpress-rest-api-navod-pro-zacatecniky-druhy-dil/

[6] What is WSDL? [online]. © Copyright IBM Corporation 2018, 2022 [cit. 2022-03-21]. Dostupné z: https://www.ibm.com/docs/en/app-connect/11.0.0?topic=services-what-is-wsdl/

[7] DOBÍJECÍ STANICE PRO ELEKTRICKÁ VOZIDLA [online]. ©2022 Ministerstvo pro místní rozvoj ČR. [cit. 2022-03-21]. Dostupné z: https://www.agentura-api.org/wp-content/uploads/2020/03/metodicka-pomucka-mmr-k-nabijecim-stanicim.pdf

[8] Jak, kde a za kolik nabít elektromobil? [online]. © 2022 24net s.r.o. [cit. 2022-03-21]. Dostupné z: https://fdrive.cz/clanky/jak-kde-a-za-kolik-nabit-elektromobil-kompletni-pruvodce-5005

[9] Kabely a nabíjecí standardy [online]. PHOENIX CONTACT, s.r.o. [cit. 2022-03-27]. Dostupné z: https://www.autonabijecka.cz/kabely-a-nabijeci-standardy/

[10] OCPP v1.5 A functional description [online]. Copyright © 2022 Open Charge Alliance [cit. 2022-04-11]. Dostupné z: https://www.openchargealliance.org/protocols/archive/

[11] Dokumentace k Entity Framework [online]. Copyright © Microsoft 2022 [cit. 2022-05-03]. Dostupné z: https://docs.microsoft.com/cs-cz/ef/

[12] ALBAHARI, Joseph a Ben ALBAHARI. C# 7.0 in a nutshell. 7th edition. Sebastopol: O´Reilly, [2018]. ISBN 978-1491987650.

[13] Roman Kümmel. Cross-Site Scripting v praxi [online] [cit. 2022-05-08] Dostupné z: https://www.soom.cz/data/Cross-Site_Scripting_v_praxi__Roman_Kummel.pdf

[14] Dobíjecí stanice Tesla Supercharger [online]. In: . [cit. 2022-05-09]. Dostupné z: https://i0.wp.com/tallahasseereports.com/wp-content/uploads/2015/09/TELSA2.jpg?fit=701%2C524&ssl=1

[15] Ukázka SOAP souboru [online]. In: . [cit. 2022-05-09]. Dostupné z: http://schemas.xmlsoap.org/soap/envelope/

[16] Zastoupení protokolů [online]. In: . [cit. 2022-05-09]. Dostupné z: https://httparchive.org/