Monitorování parkovacích míst a správa parkoviště - návrh řešení a aplikace

Ing. Jana Sedláčková

Diplomová práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Ing. Jana Sedláčková**
Osobní číslo:	**A18272**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**Kombinovaná**
Téma práce:	**Monitorování parkovacích míst a správa parkoviště – návrh řešení a aplikace**
Téma práce anglicky:	**Monitoring Parking Spaces and Car Park Management – Suggested Solutions, Applications**

Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Zpracujte koncepční návrh a navrhněte architekturu řešení pro monitorování parkovacích míst na parkovišti.
3. Vytvořte aplikaci v jazyce JAVA, která bude zajišťovat přenos dat o dostupnosti parkovacích míst a jejich ukládání na server.
4. Vytvořte REST API v jazyce JAVA a frontendovou JS aplikaci, které umožní vyhodnocovat, spravovat a publikovat data o parkovištích a dostupných parkovacích místech.
5. Návrh hardwarově realizujte (včetně dokumentace) a celý systém otestujte.

Forma zpracování diplomové práce: **Tištěná/elektronická**

Seznam doporučené literatury:

1. SCHILDT, Herbert. Mistrovství – Java. Brno: Computer Press, 2014. Mistrovství. ISBN 978-80-251-4145-8.
2. in28Minutes Official. Master Hibernate and JPA with Spring Boot in 100 Steps. udemy.com [online]. 11/2020. Dostupné z https://www.udemy.com/course/hibernate-jpa-tutorial-for-beginners-in-100-steps/
3. in28Minutes Official. Master Java Web Services and RESTful API with Spring Boot. udemy.com [online]. 11/2020. Dostupné z https://www.udemy.com/course/spring-web-services-tutorial/
4. React. A JavaScript library for building user interfaces [online]. Copyright © 2020 Facebook Inc. Dostupné z https://reactjs.org
5. Demo aplikace Marushka. Geovap [online]. Copyright © 2019 GEOVAP spol. s r.o. Dostupné z https://www.geovap.com/cs/marushka/demo-aplikace-marushka

Vedoucí diplomové práce: **Ing. Tomáš Sysala, Ph.D.**
Ústav automatizace a řídicí techniky

Datum zadání diplomové práce: **15. ledna 2021**
Termín odevzdání diplomové práce: **17. května 2021**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor;
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne Jana Sedláčková v.r.

ABSTRAKT

Cílem této diplomové práce je navrhnout systém pro monitorování parkovacích míst a správu parkoviště, který umožní vyhodnocovat, spravovat a publikovat data o parkovištích a dostupných parkovacích místech. Součástí řešení je navržení systému monitorování parkoviště, a především vytvoření aplikací, které budou zajišťovat ukládání vyhodnocených dat do databáze, zpřístupňovat uložená data pomocí RestAPI tato data vyhodnocovat, spravovat a publikovat pomocí frontendové JavaScriptové aplikace. V DP nebylo řešeno vlastní vyhodnocování obsazenosti pomocí AI. V práci je popsána architektura všech těchto aplikací a také jejich implementace. V závěru bylo celé navržené řešení otestováno sběrem dat na zvoleném testovacím parkovišti.

Klíčová slova: IP kamera, parkoviště, Java, RestAPI, Spring, React, Redux

ABSTRACT

The aim of this diploma thesis is to design a system for monitoring parking places and parking management, which will allow the evaluation, management and publication of data on parking lots and available parking places. Part of the solution is the design of a parking monitoring system, and especially the creation of applications that will store the evaluated data in the database, make the stored data available using RestAPI, evaluate, manage and publish this data using a front-end JavaScript application. The thesis did not solve its own evaluation of occupancy using AI. The work describes the architecture of all these applications and their implementation. In the end, the whole proposed solution was tested by collecting data in the selected test car park.

Keywords: IP kamera, Carpark, Java, RestAPI, Spring, React, Redux

Chtěla bych poděkovat vedoucímu své práce Ing. Tomáši Sysalovi, Ph.D. za odborné vedení a konzultace, které mi poskytl v průběhu zpracování diplomové práce. Dále bych ráda poděkovala Ing. Stanislavu Mudrákovi a Bc. Karlu Szkanderovi za pomoc při technické realizaci řešení a za cenné rady.

OBSAH

ÚVOD

Monitorování parkovacích míst je v současné době, kdy dle údajů Svazu dovozců automobilů připadá na každého obyvatele ČR více než půlka automobilu (údaj z ledna 2021), velmi aktuální téma. K rychlému rozvoji v této oblasti přispěly také pokroky v oblasti umělé inteligence. Vzniká potřeba systémů a aplikací, jejichž cílem je poskytnutí informací o dostupných parkovištích, ať už se to týká statičtějších údajů zahrnujících vlastnosti, množství a rozložení parkovacích míst, údaje o zpoplatnění, otvírací době a dalších, nebo aktuálních údajů o obsazenosti jednotlivých parkovacích míst tak, aby byl k dispozici přehled o možnosti zaparkování.

Cílem této práce je navrhnout základ funkčního systému, který umožní nejen monitorování aktuálního stavu parkoviště, ale také jeho efektivní správu. Pod těmito slovy si můžeme představit uchovávání popisných informací o parkovišti, jednotlivých parkovacích místech, možnost evidence pronájmu parkovacího místa soukromým nebo firemním zájemcům, možnost přikládání dokumentů a souborů k jednotlivým položkám. Zároveň bude tento systém sbírat data o obsazenosti jednotlivých parkovacích míst a tato data prezentovat, včetně souhrnných statistických údajů.

Návrh systému byl vytvořen se snahou o možnost jeho dalšího postupného rozšiřování přidáváním nových částí či modulů tak, aby v budoucnu mohl sloužit zájemcům o jeho využívání. Zároveň je jeho struktura navržena tak, aby mohl být začleněn do již funkčního systému, který zahrnuje např. Správu budov, komunikací, zeleně a dalšího.

I. TEORETICKÁ ČÁST

1 STÁVAJÍCÍ APLIKACE PRO SPRÁVU A MONITORING PARKOVIŠTĚ

V současné době je na trhu větší množství nejrůznějších aplikací, které slouží k monitoringu obsazenosti parkovacích míst a poskytují informace o jednotlivých parkovištích. Podle způsobu detekce obsazenosti se dělí do 2 skupin – první z nich snímá parkoviště pomocí jedné nebo více kamer, u druhého systému jsou instalována čidla do jednotlivých parkovacích míst, přičemž druhý systém má nevýhodu náročnějších úprav v podobě instalace čidel. Některé z aplikací umožňují oba způsoby detekce. K dalším funkcionalitám těchto aplikací patří kromě detekce obsazenosti také správa rezidenčního parkování zahrnující také kontrolu oprávněnosti parkování, navigace na volná parkovací místa pomocí mobilní aplikace včetně plateb parkového nebo statistické výstupy z nasbíraných dat.

V dalším textu uvádím stručný popis několika vybraných českých, ev. slovenských aplikací, které jsou v současné době k dispozici. Z velkého množství zahraničních je možno zmínit např. Car Detection System holadnské firmy LumiGuide Smart Mobility Solutions, který umožňuje detekci parkovacích míst jak ve vnitřních parkovištích (pomocí ultrazvukových senzorů nebo kamer), tak na venkovních parkovištích (pomocí kamer) [1].

1.1 Systém Parkinto firmy Smartiple

Česká fima Smartiple vyvinula parkovací systém Parkinto, který umí detekovat obsazenost parkovacích míst aut nebo lodí na základě obrazu z kamery s využitím algoritmů z oblasti umělé inteligence. Jedná se o cloudovou službu, do které je možné připojit jakýkoli typ kamery. Kromě vlastní detekce obsazenosti poskytuje tento produkt také výstupy v podobě statistik o využití parkoviště a rozpoznání registračních značek. Součástí systému je také mobilní aplikace s navigací na volné parkovací místo. Jako samostatnou možnost nabízí tato aplikace rozhraní RestAPI, které je možno využít k připojení vlastního systému. Typickými příklady využití jsou městské parkovací plochy, hotelová parkoviště, parkoviště u obchodních center, firemní parkoviště nebo také lodní přístavy [21].

Rozhraní RestAPI systému Parkinto bylo využito jako součást systému, prezentovaného v této diplomové práci.

1.2 Systém ParkDots firmy PosAm, spol.s.r.o.

Systém ParkDots je komplexní systém, který zahrnuje monitorování obsazenosti parkovacích míst, správu parkování pro rezidenty, kontrolu oprávněnosti parkování. Poskytuje také statistická data a umožňuje integraci systémů třetích stran.

Jednou ze součástí systému je také úhrada parkovného pomocí mobilní aplikace, Řidič si může vyhledat parkoviště ve svém okolí, zjistit jejich obsazenost a cenu parkovného, na vybrané místo se nechat navigovat a také provést úhradu parkového.

Systém monitoringu parkovacích míst vychází ze senzorů, které jsou osazeny přímo do vozovky, nicméně umožňuje také spolupráci s kamerovými systémy s využitím umělé inteligence (AI), detekujícími registrační značku nebo typ vozidla. Disponuje také integrací na vybrané závorové systémy [20].

1.3 ParkingDetection firmy RCE systems s.r.o.

ParkingDetection je chytrý parkovací systém, který poskytuje informace o vytíženosti parkoviště v reálném čase, měří dobu parkování a pohyb na parkoviště a dá se propojit s parkovacími automaty, což umožňuje spravovat platební systém. Umožňuje detekci nesprávného parkování (např auta na místech autobusů) nebo detekci jízdy v protisměru. Jeho součástí je aplikace, která umožní vzdáleně spravovat jednotlivé kamery a parkovací místa, definovat vlastní parkovací pravidla či vytvářet pokročilé statistiky. a také mobilní aplikace pro řidiče [19]

Monitorování parkoviště probíhá pomocí kamer, přičemž jedna kamera dokáže pokrýt parkovací plochu až o 400 parkovacích místech.

2 MONITOROVÁNÍ PARKOVACÍCH MÍST - TECHNOLOGIE

Vlastní monitorování obsazenosti parkovacích míst probíhá dvěma základními způsoby. Prvním z nich je využití napevno zabudovaných senzorů, které obsazenost vyhodnocují přímo, druhým pak použití kamerových systémů s vyhodnocování pomocí AI.

2.1 Bezdrátové parkovací senzory

Bezdrátové parkovací senzory pracují nejčastěji na principu infračerveného nebo radiového záření, ev. elektromagnetické indukce. Bezdrátový přenos informací je většinou uskutečňován s využitím interní antény pomocí některé sítě internetu věcí (IoT), např. LoRa.

Výhodou bezdrátových parkovacích senzorů je přímá informace o obsazenosti/neobsazenosti parkovacího místa, bez nutnosti použití AI. Některé systémy také umožňují např. integraci RFID čipu, který může evidovat zvláštní parkovací povolení pro invalidy či rezidenty, či jiné doplňující informace. Nevýhodou je potom nutnost instalovat tyto senzory zvlášť na každé parkovací místo.

K zástupcům bezdrátových parkovacích senzorů patří např. Zemní parkovací senzor Siemens [18], Bosh wireless smart parking sensor [17] (Obr. č. 1) nebo LoRaWAN Parking Sensor firmy MOKOSmart [16].

Obrázek 1: Parkovací senzor firmy Bosh [18]

Dalším zajímavým řešením, které také můžeme zahrnout do této skupiny, je umístění radarových senzorů do lamp pouličního osvětlení nebo na fasády budov (Advanced Parking Management firmy Siemens) [18]. Tyto senzory mají nižší rozlišení než běžné monitorovací kamery, což je v tomto případě vzhledem k GDPR výhoda. Odpadá zde nutnost instalace samostatných senzorů pro každé parkovací místo, je však nutno využít složitějšího algoritmu k vyhodnocení obsaszenosti.

2.2 Internetové (IP) kamery

Monitorování obsazenosti parkovacích míst pomocí internetových kamer patří k nejčastěji využívaným řešením. Umožňuje nepřetržité snímání většího množství parkovacích míst (v závislosti na typu kamery a místu její instalace), přičemž záběry z kamery jsou zpřístupněny pomocí Internetu a v pravidelném intervalu vyhodnocovány pomocí algoritmů AI.

IP kamery mají svou vlastní internetovou adresu a komunikují na základě **rodiny protokolů TCP/IP.** Jedná se o sadu protokolů, které definují pravidla pro komunikaci v počítačové síti. Protokoly jsou definovány pro jednotlivé vrstvy čtyřvrstvého TCP/IP modelu. Nejnižší vrstva síťového rozhraní (Network Interface Layer) zajišťuje přenos rámců (frame) mezi dvěma propojenými počítači a vzhledem k tomu, že jsou zde různé přenosové technologie závisející na konkrétní implementaci dané sítě, nejsou pro tuto vrstvu specifikovány žádné TCP/IP protokoly. Na úrovni druhé nejnižší vrstvy, tedy vrstvy síťové (Internet Layer), která slouží především k síťové adresaci, směrování a předávání datagramů, jsou definovány protokoly IP, ICMP, ARP, RARP a další. Internet Protocol (IP) provádí vysílání datagramů na základě síťových IP adres, přičemž se jedná o nespojovaný nespolehlivý protokol s částečnou detekcí chyb. Internet Control Message protokol (ICMP) přenáší zprávy o chybách a řídící zprávy, které jsou součástí IP datagramu. Třetí, transportní vrstva, obsahuje protokoly TCP a UDP. Transmission Control Protocol (TCP) protokol zajišťuje spojení, přes které lze obousměrně přenášet data, přičemž garantuje spolehlivé doručování a doručování ve správném pořadí. V opačném případě, kdy není vyžadována spolehlivost nebo je vyžadována rychlost, je použit protokol User Datagram protokol (UDP), který nevyžaduje navázání spojení. Tento protokol je pro přenos obrazu v reálném čase vhodnější. Ve čtvrté, aplikační vrstvě, jsou provozovány základní aplikace v rámci TCP/IP, kdy zajišťuje přenos a srozumitelnost zpráv. Patří sem protokoly FTP (přenos souborů), SMTP (elektronická pošta), HTTP (přenos hypertextových dokumentů), DNS (systém doménových jmen), DHCP (dynamické přidělování síťových informací) a spousta dalších.

Pro připojení kamery k routeru se využívají 2 základních druhy datového přenosu – drátového nebo Wi-Fi. Drátové připojení se realizuje pomocí LAN kabelu (dle IEEE 802.3). Jedná se o nejlevnější typ připojení, které je ale složitější na instalaci. WiFi připojení (IEEE 802.11) je jednodušší na realizaci, ale trpí častým kolísáním signálu a při výpadku sítě přestává fungovat. V místech, kde není zaveden klasický internet se využívá 3G/4G připojení pomocí vložení SIM karty do kamery, jedná se ale o nejdražší řešení [15].

Dalšími parametry pro výběr vhodné kamery pro využití v rámci monitoringu parkovacích míst jsou kromě typu připojení také její tvar, rozlišení, typ objektivu, funkce pro zlepšení kvality obrazu, velikost interní paměti, možnost externí paměťové karty. Důležitým parametrem je vzhledem k potřebnému využití v nočních hodinách také světelná citlivost, případně funkce nočního vidění.

3 VYHODNOCOVÁNÍ OBSAZENOSTI PARKOVACÍCH MÍST (AI)

Vyhodnocování obsazenosti parkovacích míst se provádí na základě analýzy výstupů z IP kamery pomocí algoritmů umělé inteligence. Vlastní vyhodnocení obsazenosti parkovacích míst není součástí této práce, proto zde uvádím pouze stručný přehled metod, které se pro tento účel využívají.

Algoritmy pro detekci vozidel v obraze lze rozdělit dle principu učení do dvou obecných kategorií, a to na algoritmy s učitelem nebo bez učitele. Pokud je znám přesný nebo alespoň přibližný vzhled detekovaných vozidel, což je i případ detekce na parkovišti, můžeme algoritmus natrénovat pomocí trénovací sady obsahující stovky až tisíce snímků vozidel a pozadí. Použitý algoritmus se na této sadě natrénuje, při samotné detekci pak hledá natrénovaný vzor v obraze. K těmto metodám se řadí AdaBoost (Adaptive Boosting), K-Nearest Neighbor (K-nejbližších sousedů) nebo metoda Support vector machines (SVM). Pro dobře separovatelné objekty je vhodná metoda segmentace obrazu. Další metody používají histogram gradientů nebo kaskádový detektor. Často využívané jsou také neuronové sítě [14].

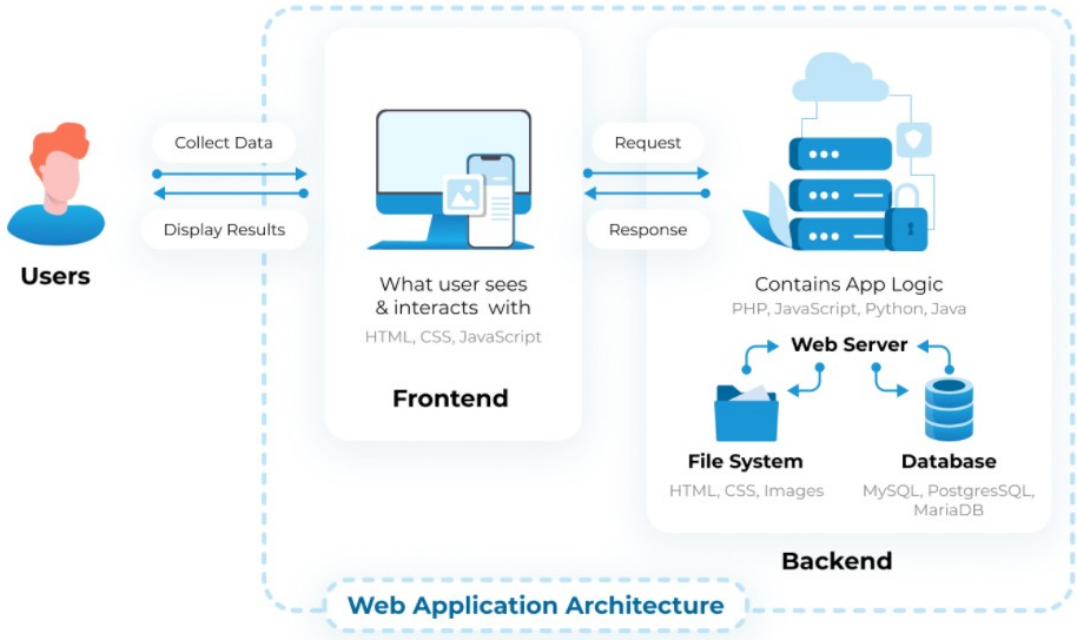
Metody bez učitele se využijí v případě, kdy vzhled detekovaného vozidla není znám, je příliš složitý nebo se pro jednotlivé případy výrazně liší. Mohou se využít např. při vyhodnocování pohybujících se automobilů. Při detekci volných parkovacích míst v reálném čase jsou tyto metody vzhledem k menší rychlosti klasifikace méně vhodné [14].

4 WEBOVÉ APLIKACE

Jednou z nejrozšířenějších architektur webových aplikací je tzv. 3 vrstvá architektura (Obr. č.). Skládá se z následujících vrstev [10] [9]:

- Prezentační vrstva (jinak též „Frontend“) - má na starost komunikaci s uživatelem, většinou ve formě grafického rozhraní, která je zajišťována pomocí webového prohlížeče. Základními technologiemi používanými v rámci této vrstvy jsou HTML, CSS a JavaScript.
- Aplikační nebo také Logická vrstva zahrnuje veškerou logiku aplikace. Přijímá požadavky prohlížeče, zpracovává je, komunikuje se zdroji dat.
- Datová vrstva slouží jako úložiště dat

Aplikační a Databázovou vrstvu společně můžeme označit jako „Backend“.



Obrázek 2: Schema webové architektury [10]

4.1 HTTP protokol

Pro komunikaci v rámci internetu se využívá HTTP protokol. Jedná se o aplikační protokol sloužící pro přenos hypertextových dokumentů, který je implementován nad transportním protokolem TCP a je založen na principu požadavek/odpověď.

Základními požadavky jsou GET (, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT. Jejich souhrn je uveden v tabulce č. 1

Tabulka 1: Přehled metod protokolu HTTP

Název metody	***Popis***
GET	nejpoužívanější metoda, která slouží k vyzvednutí dat. Obvykle nemá tělo a dotaz se posílá v hlavičce.
HEAD	slouží se zjištění existence objektu
POST	odesílá uživatelská data na server v těle protokolu
PUT	používá se pro nahrávání dat na server – je potřeba mít oprávnění
DELETE	smaže data ze serveru – je potřeba mít oprávnění
TRACE	slouží ke zjišťování diagnostických informací
OPTIONS	slouží ke zjištění informací, o tom, jaké dotazy lze zaslat
CONNECT	slouží např. pro navázání šifrovaného spojení

Odpovědí na jednotlivé požadavky jsou stavové kódy, z nichž nejznámější jsou např. 200 OK, 401 Unauthorized, 404 Not Found nebo třeba 500 Server Error.

Samotný HTTP protokol není zabezpečený. Pro komunikaci se proto používá spou s protokolem SSL nebo TLS jako protokol HTTPS, který již umožňuje šifrovanou komunikaci.

4.2 RestApi

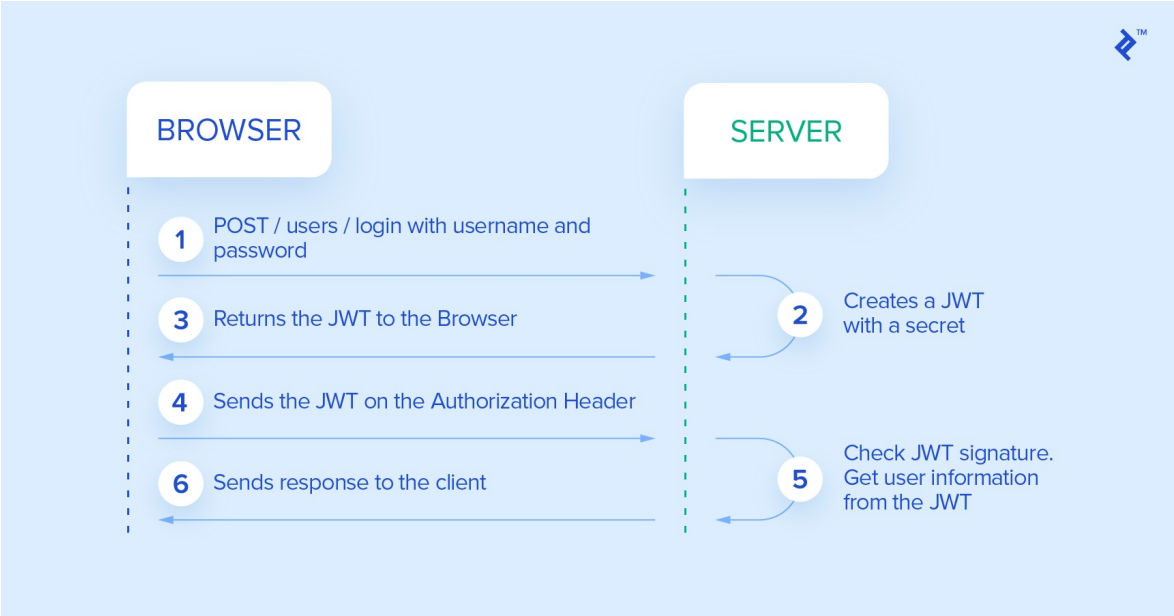
Rest (Representational State Transfer) API je architektura rozhraní, které slouží pro jednotný a snadný přístup ke zdrojům (datům) na určitém místě (všechny zdroje mají vlastní identifikátor URI) pomocí standardních metod HTTP. Rest implementuje 4 základní metody, označné zkratkou CRUD [8]:

1. Create – slouží k vytvoření zdroje. Využívá se HTTP metoda POST, přičemž volání probíhá na obecný endpoint (nový identifikátor zdroje ještě není vytvořen).
2. Retrieve – slouží k získání zdroje. Využívá se metoda GET.
3. Update – slouží k editaci již existujících dat. Používá se metoda PUT
4. Delete – slouží k mazání existujících dat. Používá se metoda DELETE.

Hlavním formátem pro výměnu dat je v současné době JSON, používají se však i jiné formáty (např. XML).

4.2.1 Zabezpečení RestAPI pomocí JWT

Jelikož je REST bezstavový protokol, neumožňuje serveru uchovávat žádné stavy uživatele. Server odpovídá na každý požadavek uživatele, jako by šlo o jeho první. Jako jeden ze způsobů autentizace uživatelů se tak používají tokeny – něco jako bezpečnostní klíče, které mají v sobě uložené zašifrované informace Takovým tokenem je JWT (JSON Web Token). Lze jej použít k předání identity ověřeného uživatele, a může také nést další informace – například autorizační data. Způsob zabezpečení pomocí JWT je zřejmý z Obr. č. 3.



Obrázek 3: Autentifikace pomocí JWT [8]

II. PRAKTICKÁ ČÁST

5 KONCEPČNÍ NÁVRH A ARCHITEKTURA ŘEŠENÍ

Pro zpracování koncepčního návrhu aplikace je nejdříve nutno analyzovat požadavky, které budou na danou aplikaci kladeny, a definovat její funkcionalitu. Tyto požadavky vyplynuly z předpokladu začlenění aplikace do částečně již funkčního systémem správy majetku. K požadovaným funkcionalitám patří:

- evidence popisných informací o parkovišti (např. typ parkoviště, jeho povrch, rozměry, zda je zpoplatněné) a o jednotlivých parkovacích místech (rozměry, možnosti pronájmu, vyhrazenost), s možností editace těchto informací
- evidence o správci parkoviště, o pronájmech parkovacích míst, s možností založení smlouvy o pronájmu včetně jejího generování a tisku
- získání informací o obsazenosti parkoviště v reálném čase, a to pro všechna parkovací místa, a prezentace těchto informací včetně statistických údajů
- možnost využití systému pro více parkovišť
- možnost připojení ke geografickému informačnímu systému (GIS)
- možnost budoucího rozšiřování aplikace o další funkcionalitu

Na základě výše uvedeného bylo definováno několik kroků, které jsou nezbytné k sestavení a zprovoznění celého systému.

1. Instalace systému pro monitorování parkoviště
2. Analýza a vyhodnocení získaného obrazu
3. Vytvoření databáze, do které se budou ukládat veškerá data (popisné, případně i grafické údaje o parkovišti, údaje o obsazenosti parkovacích míst), a základní naplnění databáze
4. Ukládání výsledků analýzy obrazu do databáze
5. Zpřístupnění všech uložených dat pomocí RestAPI
6. Vytvoření frontendové aplikace, pomocí které budou uživatelé přistupovat k získaným datům

5.1 Systém pro monitorování parkoviště

Navržený způsob monitorování parkoviště je vzhledem k předpokladům využití celého systému pro již existující parkoviště, kdy chceme vyloučit složitou instalaci senzorů, pomocí

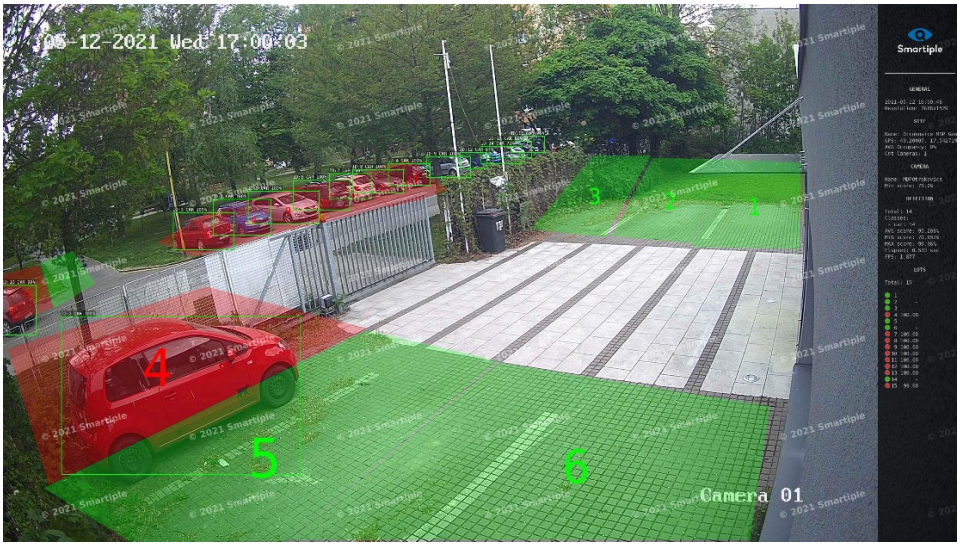
IP kamery. Kamera musí svým záběrem pokrývat celou plochu monitorovaného prostoru. Je potřeba vybrat kameru s vysokou citlivostí, ideálně s funkcí nočního vidění tak, aby byla schopna poskytovat relevantní obraz i v průběhu noci. Pro větší stabilitu je lepší volit drátové připojení pomocí LAN kabelu, ale není vyloučené ani WiFi připojení. V rámci testování aplikace byla zvolena kamera HIKVISION DS-2CD2045FWD-I [12] s funkcí infračerveným přísvitu, která byla připojena drátově přes rozhraní: RJ45 Fast Ethernet s napájeím pomocí kabelu (PoE). (Obr. č. 4).

Obrázek 4: IP kamera
HIKVISION [13]

Kameru je potřeba nakonfigurovat a nastavit k ní přístupy tak, aby výstup z ní ve formě jednotlivých snímků ve stanoveném časovém intervalu nebo přímo ve formě souvislého streamu byl k dispozici jako vstup pro analýzu obrazu.

5.2 Analýza obrazu a její výstupy

Vlastní vyhodnocení obsazenosti parkovacích míst není součástí této práce, předpokládáme využití některé z aplikací třetích stran. V našem případě byl použit systém Parkinto společnosti Smartiple [21] (Obr. č. 5), který jako jednu z možností nabízí vyhodnocení obsazenosti a zpřístupnění výsledků pomocí RestAPI, s aktualizací údajů každou sekundu. Dalším možným řešením by bylo využít některou cloudovou službu nabízející využití algoritmů strojového učení jako je například Azure Custom Vision [13] a natrénovat vlastní model.



Obrázek 5: Vyhodnocení obsazenosti parkoviště

Vstupem pro vlastní analýzu obrazu jsou snímky z IP kamery, jak je uvedeno výše, výstupem pak jsou data zpřístupněná ve formě RestAPI ve formátu JSON (dále uváděné jako RestAPI obsazenosti). Výstupní formát je vzhledem k následnému ukládání dat do databáze závazný (Obr. č. 6). Informace o stavu parkovacího místa (lot) jsou obsaženy v položce "state" (free/occupied), položka „score" udává přesnost výsledků vyhodnocení. Dalším údajem je název parkoviště a časový údaj události v položce „timestamp". Čas je zde uváděn v časovém pásmu UTC.

Obrázek 6: Výstup z vyhodnocení obsazenosti - formát JSON

5.3 Uchovávání získaných dat – databáze

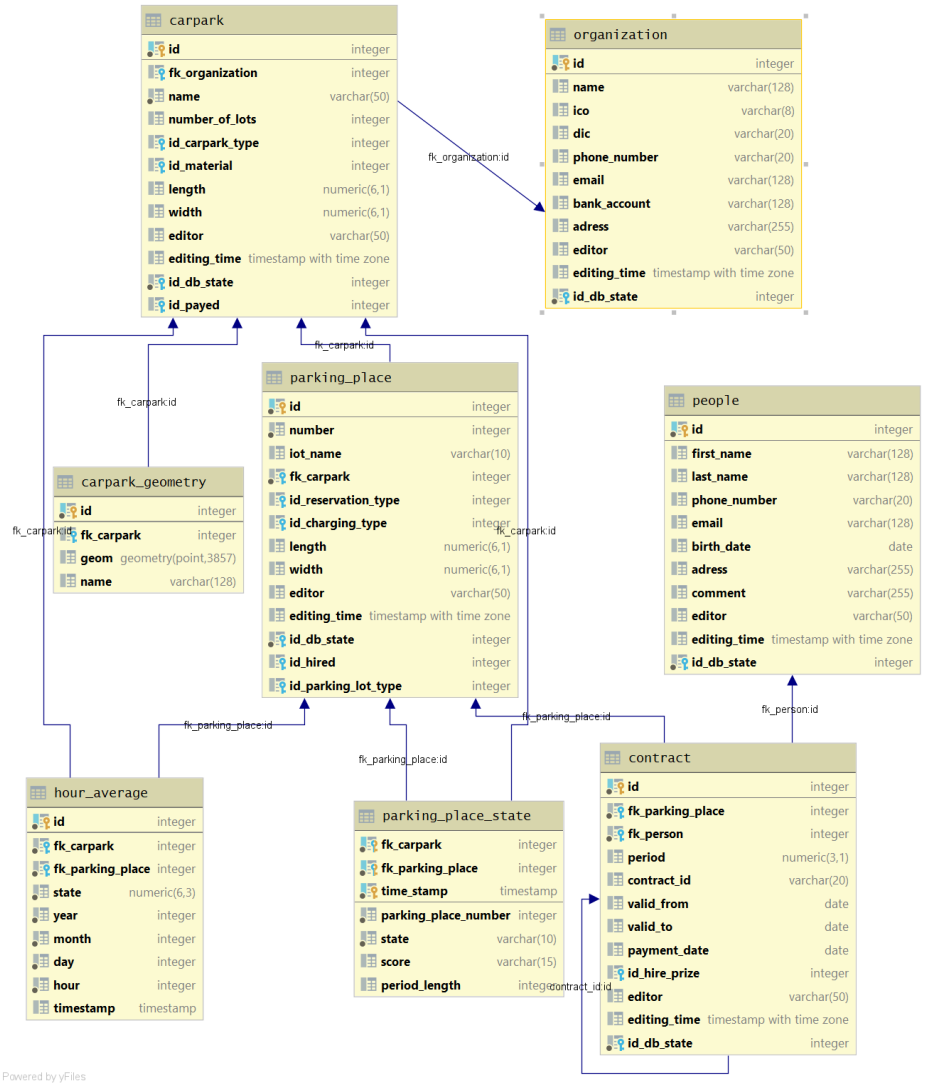
5.3.1 Výběr vhodné databáze

Vzhledem ke strukturovanosti uchovávaných dat byl pro zvolen relační typ databáze. Základním kritériem výběru je umožnění práce s geoprostorovými daty. V rámci předkládaného řešení jsme zvolili databázi PostGreSQL s nadstavbou PostGIS, která má výhodu Open Source řešení. V rámci daného systému je nicméně možno použít i jinou databázi, pokud bude zachována stejná struktura.

5.3.2 Struktura databáze – tabulky, pohledy, triggery

Strukturu základních tabulek přehledně ukazuje Obr. č. 7. Celý systém je navržen tak, aby se mohla uchovávat data z různých parkovišť (tabulka **carpark**), u kterých se předpokládá jeden správce (tabulka **organization**). Tabulka **carpark_geometry** uchovává geoprostorové informace o parkovišti (v současné době pouze ve formě bodu). Tabulka

parking_places obsahuje informace o jednotlivých parkovacích místech. V tabulce **contract** jsou uloženy údaje o smlouvách, které byly uzavřeny mezi správcem a nájemcem (tabulka **people**) na dlouhodobý pronájem parkovacích míst.



Obrázek 7: Struktura základních tabulek databáze

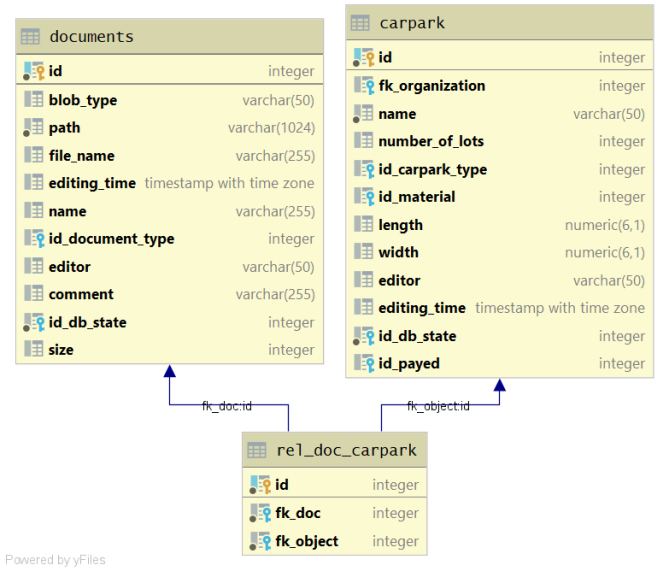
Data obsazenosti parkovacích míst se ukládají do tabulky **parking_place_state**. Tato tabulka se velmi rychle plní, jenom při ukládání stavu každou minutu pro 1 parkoviště o 15 parkovacích místech se každý den uloží 21 600 nových záznamů. Abychom mohli uchovávat statistická data a zároveň nebyli zahlceni množstvím záznamů, byla vytvořena tabulka **hour_average**. Do ní se pomocí vytvořeného triggeru ukládají hodinové průměry obsazenosti pro každé parkovací místo. Zároveň se z tabulky parking_place_state mažou záznamy starší 1 hodinu, která již není potřeba uchovávat. V rámci triggeru je připraveno

také mazání starších záznamů z tabulky hour_average, které však zatím není realizováno. Do budoucna bude nutno otestovat rychlost databáze při větším naplnění této tabulky a najít vhodný kompromis mezi požadavkem na délku období, ze kterého jsou prezentovány statistické údaje, a počtem uchovávaných záznamů.

Tabulka parking_place_state nemá vzhledem k častému mazání starých a vkládání nových položek primární klíč tvořený celočíselnou řadou, ale je vytvořený kombinací údajů z cizích klíčů do tabulek carpark a parking_place a časového údaje záznamu.

Pro prezentaci údajů o aktuální obsazenosti parkovacích míst byl vytvořen pohled view_parking_place_state, při jehož vytváření je vyřešen převod mezi časovým pásmem UTC a místním časem, a zároveň vrací pouze nejaktuálnější časový údaj.

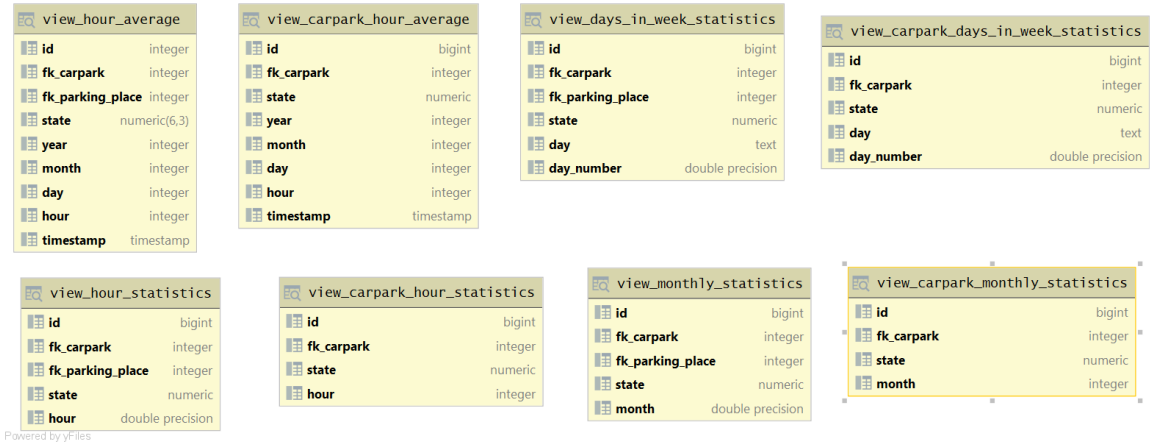
Položky databáze začínající fk_ představují cizí klíče do jiných tabulek, položky začínající id_ jsou cizí klíče do číselníků. Číselníky (lookup table) jsou samostatné tabulky, které uchovávají seznam hodnot; na Obr. č. 7 nebyly pro přehlednost zahrnuty, jejich seznam je však součástí tabulky č. 1. Stejně tak zde není zobrazena tabulka dokumentů a s ní související relační tabulky, mapující jednotlivé dokumenty k příslušným tabulkám. Struktura tabulky dokumentů je patrná včetně relační tabulky do tabulky carpark je patrná z Obr. č. 8. Relační tabulky mezi dokumenty a základními tabulkami jsou vytvořeny také pro tabulky parking_place, organization, contract a people.



Obrázek 8: Tabulka dokumentů s ukázkou propojení na základní tabulku (zde carpark)

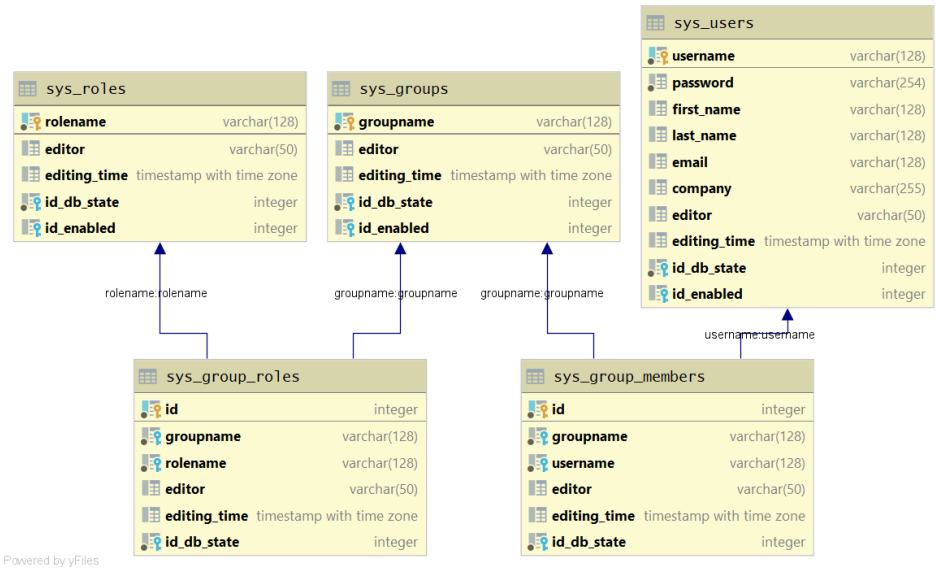
Pro zobrazování statistických údajů o obsazenosti parkoviště byly vytvořeny pohledy (view) (Obr. č. 9), které předávají již zpracovaná data - hodinové průměry v delší časové

řadě, průměry pro jednotlivé hodiny v rámci dne, dny v týdnu a měsíce v roce, oboje jak pro v souhrnu pro celá parkoviště, tak pro jednotlivá parkovací místa v rámci parkoviště.



Obrázek 9: Struktura view pro zobrazování statistik

Další skupinou jsou tabulky, které souvisejí se správou uživatelů (obr. č. 10). Tabulka sys_users uchovává vlastní údaje o jednotlivých uživatelích, ostatní tabulky potom souvisejí s definováním přístupových práv a přiřazení uživatelů do jednotlivých skupin.



Obrázek 10: struktura databázových tabulek pro správu uživatelů

Poslední skupinou jsou tabulky, které souvisejí s definováním obsahu klientské části na straně frontendu. Jednou z funkcionalit celého systému je zobrazování a správa informací o parkovištích, parkovacích místech a dalších objektech ve formě tabulek. Protože počítáme s možností rozšiřování v podobě dalších modulů zahrnujících jiné tabulky, ukládá se rozsah zobrazovaných tabulek včetně jejich vlastností právě v databázi. Součástí tohoto systé-

mu jsou tedy tabulky **client_tree, client_table, client_section a client_column**, včetně potřebných číselníků.

Všechny základní tabulky, které uchovávají popisná data, obsahují položky uchovávající údaje o času poslední editace (editing_time) a uživatelském jménu editora (editor), a také systémovou položku s údajem o platnosti záznamu (sys_db_state). Souhrn všech tabulek a pohledů databáze je uveden v Tabulce č. 2.

Tabulka 2: Přehled databázových tabulek a view

Název	***Typ***	***Účel***
carpark	Základní tabulka	Uchovává popisné údaje o parkovišti
carpark_geometry	Tabulka s geodaty	Uchovává geografické údaje o parkovišti
contract	Základní tabulka	Uchovává popisné údaje o smlouvě
documents	Základní tabulka	Uchovává popisné údaje o dokumentech
organization	Základní tabulka	Uchovává popisné údaje o organizaci (správci)
parking_place	Základní tabulka	Uchovává popisné údaje o parkovacím místu
people	Základní tabulka	Uchovává popisné údaje o nájemcích
parking_place_state	Stavová tabulka	Uchovává aktuální údaje o obsazenosti parkovacích míst
view_parking_place_state	Stavové view	Zobrazuje aktuální údaje o obsazenosti parkovacích míst
hour_average	Stavová tabulka	Uchovává zprůměrované údaje o obsazenosti parkovacích míst
view_carpark_days_in_week_statistics	View se statistikou	Zobrazuje statistické údaje o obsazenosti parkovacích míst pro jednotlivé dny v týdnu pro celé parkoviště
view_carpark_hour_average	View – průměrné hodnoty	Uchovává zprůměrované údaje o obsazenosti parkovacích míst
view_carpark_hour_statistics	View se statistikou	Zobrazuje statistické údaje o obsazenosti parkovacích míst pro jednotlivé dny v týdnu pro celé parkoviště
view_carpark_monthly_statistics	View se statistikou	Zobrazuje statistické údaje o obsazenosti parkovacích míst pro jednotlivé dny v týdnu pro celé parkoviště
view_days_in_week_statistics	View se statistikou	Zobrazuje statistické údaje o obsazenosti parkovacích míst pro jednotlivé dny v týdnu pro celé parkoviště

Název	*Typ*	*Účel*
view_hour_average	View – průměrné hodnoty	Uchovává zprůměrované údaje o obsazenosti parkovacích míst
view_hour_statistics	View se statistikou	Zobrazuje statistické údaje o obsazenosti parkovacích míst pro jednotlivé dny v týdnu pro celé parkoviště
view_monthly_statistics	View se statistikou	Zobrazuje statistické údaje o obsazenosti parkovacích míst pro jednotlivé dny v týdnu pro celé parkoviště
lookup_carpark_type	Číselník – zákl. tab.	Číselník typu parkoviště
lookup_charging_type	Číselník – zákl. tab.	Číselník typu nabíjení
lookup_document_type	Číselník – zákl. tab.	Číselník typu dokumentu
lookup_hire_prize	Číselník – zákl. tab.	Číselník s cenami pronájmu park. míst
lookup_material	Číselník – zákl. tab.	Číselník materiálu povrchu parkoviště
lookup_parking_lot_type	Číselník – zákl. tab.	Číselník typu parkovacího místa
lookup_reservation_type	Číselník – zákl. tab.	Číselník typu rezervace parkovacího místa
lookup_sys_db_state	Číselník – system	Systémový číselník pro stav databázového záznamu
lookup_yes_no	Číselník – zákl. tab.	Číselník s hodnotami ano-ne
rel_doc_carpark	Relační tabulka	Propojení dokumentů k parkovištím
rel_doc_contract	Relační tabulka	Propojení dokumentů ke smlouvám
rel_doc_organization	Relační tabulka	Propojení dokumentů ke správcům
rel_doc_parking_place	Relační tabulka	Propojení dokumentů k parkovacím místům
rel_doc_people	Relační tabulka	Propojení dokumentů k nájemcům
sys_group_members	Uživatelé a přístupy	Přiřazení uživatelů ke skupinám s různými přístupovými právy
sys_group_roles	Uživatelé a přístupy	Přiřazení uživatelských rolí ke skupinám
sys_groups	Uživatelé a přístupy	Seznam uživatelských skupin
sys_roles	Uživatelé a přístupy	Seznam uživatelských rolí
sys_users	Uživatelé a přístupy	Seznam uživatelů
client_columns	Klient – zákl. tab.	Definice sloupce, které se zobrazují uživateli v jednotlivých sekcích tabulek
client_sections	Klient – zákl. tab.	Definice sekce, které se zobrazují uživateli v jednotlivých tabulkách
client_tables	Klient – zákl. tab.	Definuje tabulky, které se zobrazují uživateli
client_tree	Klient – zákl. tab.	Definuje, jak se zobrazují tabulky uživateli v rámci menu výběru tabulek
rel_tree_tables	Relační tabulka	Propojení tabulek s uživatelským menu

Název	*Typ*	*Účel*
lookup_client_branch_type	Číselník - klient	Číselník typu větve
lookup_client_column_type	Číselník - klient	Číselník typu sloupce
lookup_client_table_type	Číselník - klient	Číselník typu tabulky
view_client_relations	View - klient	Zobrazuje vzájemné závislosti jednotlivých tabulek pro zobrazování uživateli

5.4 Ukládání získaných dat – aplikace SaveParking

Aktuální data o obsazenosti parkovacích míst se do databáze ukládají pomocí aplikace, která přistupuje k RestAPI pomocí datazu GET. Získaná data jsou zpracována a uložena do tabulky parking_place_state.

Před vlastním spuštěním aplikace musí správce celého systém vložit do databáze záznam o parkovišti do tabulky carpark a dále záznamy o jednotlivých parkovacích místech do tabulky parking_place. Id parkoviště se potom zadává jako vstupní parametr do aplikace (viz dále), id jednotlivých parkovacích míst jako celočíselná hodnota musí být obsaženo v textu parametru „id“ parkovacího místa v RestAPI obsazenosti (pro id = 4 to bude např: „id“: „lot4“). Pokud se uvedené id nebudou shodovat, data se neuloží.

Vstupními parametry aplikace, které se zadávají jako vlastnosti aplikace v souboru application.properties, jsou údaje unikátní pro každé parkoviště, a dále také časový interval posílání požadavku (Tab. č. 3).

Tabulka 3: Vstupní údaje aplikace saveParking

Název parametru	*Popis*
CARPARK_FK	Id parkoviště v tabulce carpark
URL	Adresa Rest API obsazenosti pro posílání požadavku GET
PERIOD_IN_MILISECONDS	Časový interval posílání požadavku

Vlastní aplikace nemá žádné uživatelské rozhraní. Je spuštěna na webovém serveru a posílá GET požadavek na získání dat v zadaném časovém intervalu pomocí interního časovače. Ukázka implementace je součástí kapitoly č. 6, zdrojový kód aplikace je součástí elektronických příloh na CD.

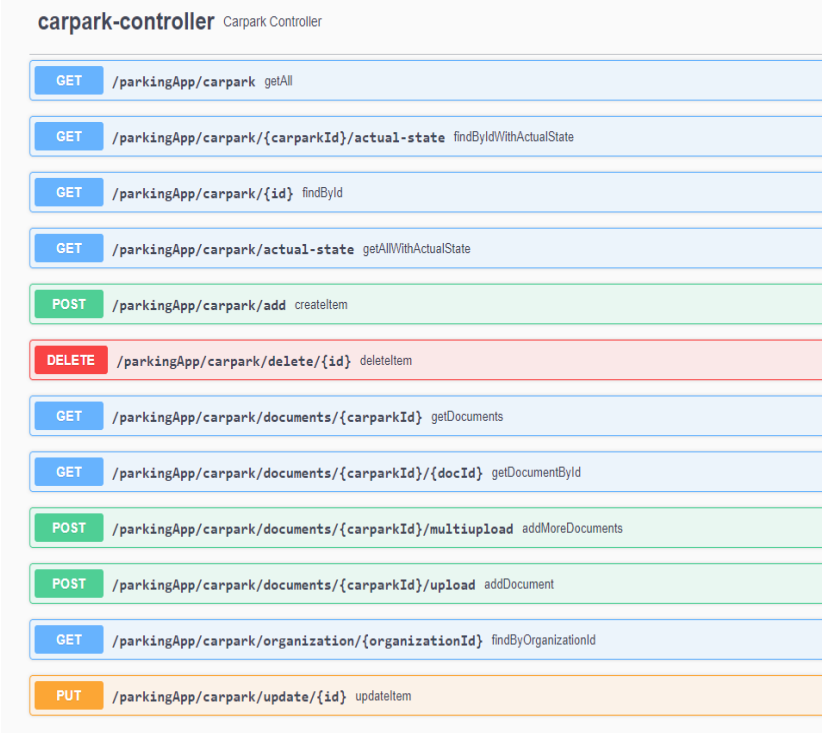
5.5 Zpřístupnění dat – RestAPI parkingApp

Aplikace ParkingApp představuje backendovou část, která zajišťuje logiku celého systému. Komunikuje s databází, zpřístupňuje data, umožňuje jejich vytváření a editaci. Dokumenta-

ce zdrojů API je zpřístupněna pomocí frameworku Swagger [11] na adrese host/parkingApp/swagger-ui/. Výměna dat probíhá ve formátu JSON.

5.5.1 Základní funkcionalita pro práci s tabulkami a číselníky

V rámci práce s tabulkami a číselníky API umožňuje nad databází provádět všechny 4 základní operace CRUD (Create, Read, Update, Delete) – tedy vytváření záznamů, jejich načítání, editaci i mazání. Obr. č. 11 uvádí jako ukázku přehled základních zdrojů pro tabulku parkoviště (carpark) – zdroje pro ostatní tabulky vypadají podobně.



Obrázek 11: Zdroje tabulky carpark

Např. pro tabulku s údaji o parkovišti (carpark) je základní endpoint na adrese *parkingApp/carpark*. Po zaslání dotazu GET na tento endpoint se pomocí metody getAll() vrátí seznam všech parkovišť uložených v databázi (Obr. č. 12 – aktuálně pouze 1 parkoviště).

```
[
  {
    "id": 1,
    "name": "Testovací parkoviště",
    "numberOfLots": 15,
    "length": 26,
    "width": 22,
    "editor": "admin",
    "fkOrganization": 1,
    "editingTime": "2021-03-13T16:10:54.096272",
    "idDbState": 0,
    "idCarparkType": 3,
    "idMaterial": 2,
    "idPayed": 2
  }
]
```


Obrázek 12: Ukázka dat získaných na základě požadavku
GET - endpoint /parkingApp/carpark

Zasíláním požadavků GET na různé endpointy je možno získat různá data – *parkingApp/carpark/{id}* (místo {id} se použije konkrétní hodnota) vrátí pouze 1 záznam s odpovídajícím id parkoviště. Endpoint *parkingApp/carpark/organization/{organizationId}* vrátí seznam (ve formě pole) všech parkovišť, které odpovídají konkrétnímu správci ({organizationId} se nahradí id daného správce). Endpoint *parkingApp/carpark/{carparkId}/actual-state* vrací souhrn popisných údajů o jednom parkovišti s id {carparkId}, doplněný údaji o obsazenosti jak celého parkoviště, tak všech parkovacích míst a byl vytvořen speciálně pro poskytování údajů na nástěnku aplikace (Obr. č. 13 a 22).

```
{
    "id": 1,
    "name": "Testovací parkoviště",
    "numberOfMonitoredLots": 15,
    "carparkTypeName": "kolmé",
    "materialName": "asfalt",
    "payedName": "ne",
    "numberOfFreeLots": 5,
    "numberOfDisabledLots": 1,
    "numberOfFamilyLots": 0,
    "numberOfHiredLots": 1,
    "numberOfChargingLots": 1,
    "parkingPlaceStateDtoList": [
        {
            "fkCarpark": 1,
            "fkParkingPlace": 1,
            "timeStamp": "13.05.2021 18:27:47",
            "state": "volno"
        },
        {
            "fkCarpark": 1,
            "fkParkingPlace": 2,
            "timeStamp": "13.05.2021 18:27:47",
            "state": "obsazeno"
```


Obrázek 13: Ukázka dat získaných na základě požadavku GET - endpoint /parkingApp/carpark/actual-state

Po zaslání požadavku POST na endpoint *parkingApp/carpark/add* se do databáze přidá nové parkoviště, jehož parametry jsou definovány ve formátu JSON v těle příkazu POST. Požadavek PUT na *endpoint parkingApp/carpark/update/{id}* aktualizuje údaje o konkrétním parkovišti dle zadaného id.

Požadavek DELETE na endpoint *parkingApp/carpark/delete/{id}* provede vymazání záznamu parkoviště s daným id. Nedojde však ke skutečnému smazání záznamu, jenom se jeho cizí klíč id_db_state do tabulky číselníku sys_db_state na změní z hodnoty 0 (aktivní záznam) na 1 (záznam označení ke smazání) a v aplikaci se nebude dále zobrazovat.

Geometrické údaje o parkovišti, které se využívají pro jeho situování v mapě, poskytuje zdroj *parkingApp/carpark-geometry*, který vrací pole s prostorovými daty jednotlivých parkovišť.

5.5.2 Dokumenty

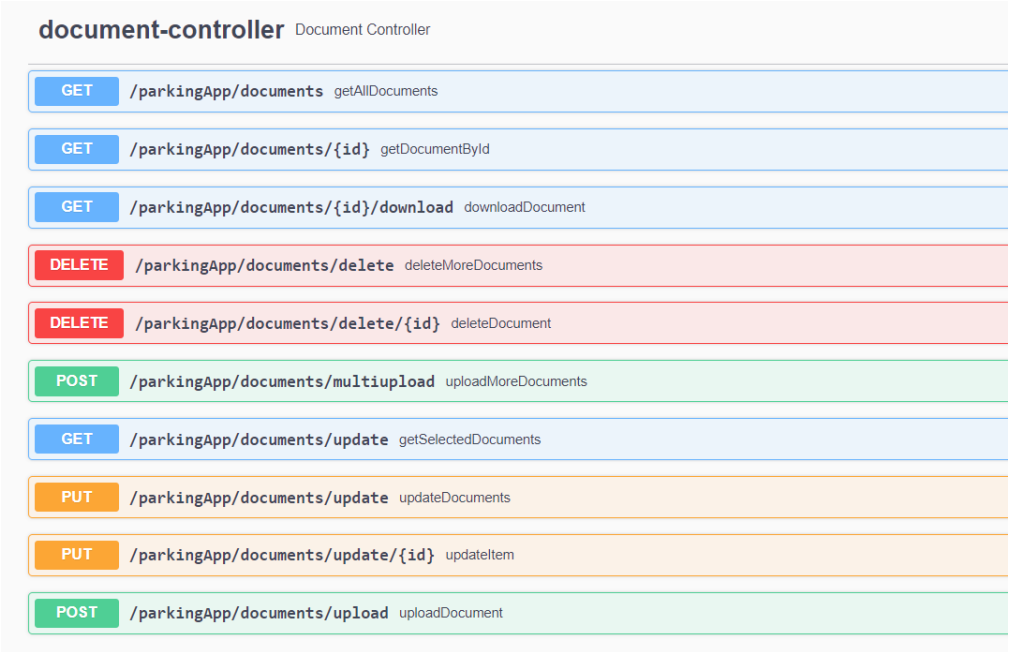
Součástí aplikace je možnost k jednotlivým záznamům základních tabulek databáze ukládat soubory. Jedná se o tabulky organization, carpark, parking_place, contract a people.

Popisná data jako je název souboru, jeho velikost, typ internetového média, typ souboru a další se ukládají do tabulky documents, jejich příslušnost k objektu je uložena v relačních tabulkách spojujících tabulkou dokumentů s příslušnou základní tabulkou. Vlastní soubory jsou ukládány do souborového systému, kdy popisná data obsahují cestu k souboru uloženému mimo databázi. Cesta do adresáře v souborové systému se nastavuje jako vstupní parametr aplikace v souboru application.properties.

Z Obr. č. 11 jsou v přehledu ukázány zdroje tabulky carpark, týkající se dokumentů. Zasláním požadavku GET na endpoint *parkingApp/carpark/documents/{carparkId}* získáme seznam dokumentů (popisná data), které patří k parkovišti s příslušným id, z endpointu *parkingApp/carpark/documents/{carparkId}/{docId}* 1 konkrétní dokument náležející konkrétnímu parkovišti. Jedná se o záznam týkající se dokumentu, nikoliv konkrétní binární soubor.

Po zaslání požadavku POST na endpoint *parkingApp/carpark/documents/{carparkId}upload, resp. parkingApp/carpark/documents/multiupload* se do databáze ke konkrétnímu parkovišti přidá binární soubor, který je vložený v těle příkazu POST.

Další manipulace s dokumenty je prováděna pomocí základního zdroje *parkingApp/documents* (Obr. č. 14). Je zde umožněno zobrazování popisných tabulek dokumentů, stažení konkrétního souboru (*parkingApp/documents/{id}/download),* mazání dokumentů pomocí požadavku DELETE (popisná data i vlastni soubor) nebo editace popisných údajů (požadavek PUT).



Obrázek 14: Zdroje pro manipulaci s dokumenty

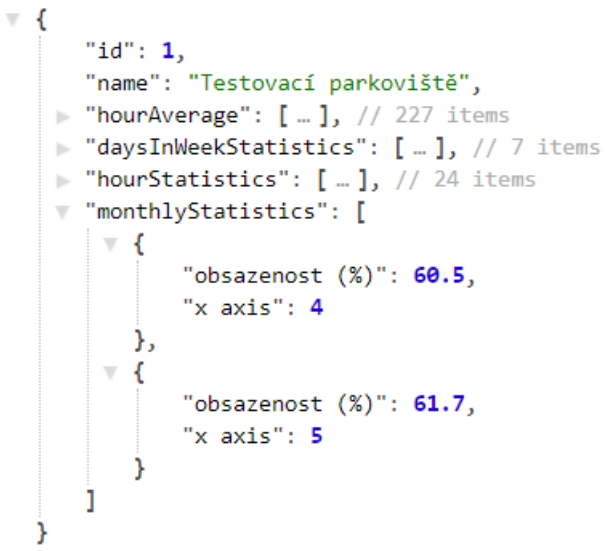
Ukázka dat popisných údajů dokumentů je patrný z Obr. č. 15.

```
[
  {
    "id": 3,
    "blobType": "image/jpeg",
    "fileName": "invalid-2021-05-16T09-03-55.jpg",
    "editingTime": "2021-05-16T09:03:55.964866",
    "size": 40709,
    "idDocumentType": 1,
    "editor": null,
    "comment": "zkouska",
    "idDbState": 0,
    "objectTable": "ParkingPlace",
    "objectId": 1,
    "objectTableId": "441b670b-7377-4de8-81e7-69c9f5cea281"
  },
  {
    "id": 4,
    "blobType": "image/jpeg",
    "fileName": "parkoviste-2021-05-16T09-20-11.jpg",
    "editingTime": "2021-05-16T09:20:11.267935",
    "size": 54662,
    "idDocumentType": 1,
    "editor": null,
    "comment": "zkouska",
    "idDbState": 0,
    "objectTable": "Carparc",
    "objectId": 1,
    "objectTableId": null
  }
]
```


Obrázek 15: Ukázka dat získaných na základě požadavku GET - endpoint /parkingApp/documents

5.5.3 Stav parkovacích míst, statistiky

Statistiky o obsazenosti parkovacích míst, které jsou potřeba pro zobrazení grafů, je možno získat ze zdroje *parkingApp/statistic* pro všechna parkoviště, resp. *parkingApp/statistic/{id}* pro konkrétní parkoviště. Výstupem je JSON (Obr. č. 16) s daty průměrné hodinové obsazenosti (hourAverage) pro každou hodinu (od začátku sledování, max. poslední rok), průměrné hodnoty pro jednotlivé hodiny v rámci dne (hourStatistics), dny v týdnu (daysInWeekStatistics) a měsíce v roce (monthlyStatistics). Grafy generované z těchto dat ukazuje Obr. č. 22



```
{
    "id": 1,
    "name": "Testovací parkoviště",
    "hourAverage": [ … ], // 227 items
    "daysInWeekStatistics": [ … ], // 7 items
    "hourStatistics": [ … ], // 24 items
    "monthlyStatistics": [
        {
            "obsazenost (%)": 60.5,
            "x axis": 4
        },
        {
            "obsazenost (%)": 61.7,
            "x axis": 5
        }
    ]
}
```


Obrázek 16: Ukázka dat získaných na základě požadavku GET - endpoint /parkingApp/statistics

5.5.4 Zabezpečení aplikace, uživatelé, přístupová práva

Popisovaná Rest API obsahuje data, která nejsou volně přístupná. S výjimkou několika zdrojů jsou všechny ostatní zdroje zabezpečeny pomocí JWT autentifikace (viz. kapitola č. 4.2.1. Nezabezpečeny jsou pouze zdroje, které slouží k přihlášení.

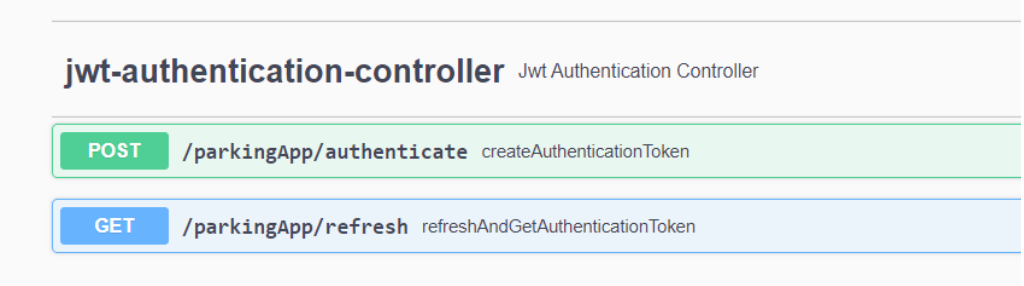
Jednotliví uživatelé jsou přiřazeni do uživatelských skupin, kdy každá ze skupin má přiděleny určité role. Nyní jsou v aplikaci implementovány 3 role, z nichž každá z nich má následující oprávnění (Tabulka č. 4):

Tabulka 4: Přehled uživatelských rolí aplikace

Název role	*Oprávnění*
ROLE_ADMINISTRATOR	Může prohlížet, editovat i mazat údaje. Má přístup k údajům o uživatelích, u nichž může měnit popisná data
ROLE_EDITOR	Může prohlížet a editovat
ROLE_VIEWER	Může pouze prohlížet

Vlastní uživatele přidává do databáze administrátor celého systému, stejně tak je přiděluj do jednotlivých skupin. V rámci aplikace pouze může měnit jejich popisné údaje, ev. je znepřístupnit.

Pro vlastní autentifikaci jsou k dispozici 2 zdroje – pro získání tokenu a pro jeho obnovu (Obr. č. 17).

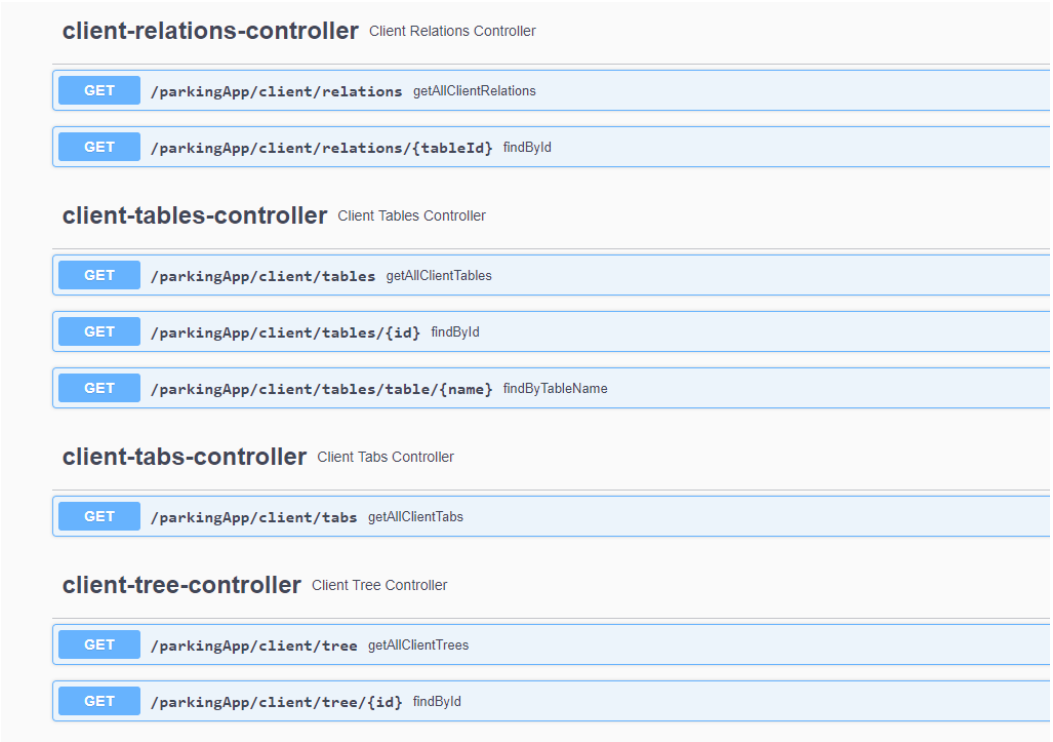


Obrázek 17: Zdroje pro autentifikaci

5.5.5 Klientské tabulky

Vytvořená Rest API zpřístupňuje také zdroje pro vzhled uživatelského prostředí aplikace na straně frontendu. Uživateli se zobrazují nejenom data o aktuální obsazenosti parkovišť, ale také data z jednotlivých popisných tabulek. Aby byl systém co nejpružnější a bylo do něj možno přidávat další tabulky bez nutnosti velkého zásahu do aplikace, definuje se uživatelský vzhled těchto tabulek již v rámci databáze. Forma výstupu z API vychází ze struktury tabulek ve frameworku react-table, který je použit v klientské části. V databázi se tedy definuje jaké tabulky budou zobrazeny, jak budou rozděleny do jednotlivých sekcí, které položky se budou zobrazovat, jejich názvy, pořadí, které tabulky jsou vzájemně propojeny relací.

Byly vytvořeny celkem 4 zdroje – client/tree, client/tables a client/relations. client/tree popisuje strukturu řazení tabulek do nabídky v menu při výběru tabulek, client/tabs určuje, které tabulky jsou spuštěné po načtení aplikace, client/tables popisují jednotlivé tabulky včetně jejich endpointů a client/relations pak vzájemné vztahy mezi tabulkami (Obr. č. 18).



Obrázek 18: Zdroje pro klientské tabulky

struktura dat pro tables a relations je patrná z obrázku č. 19.

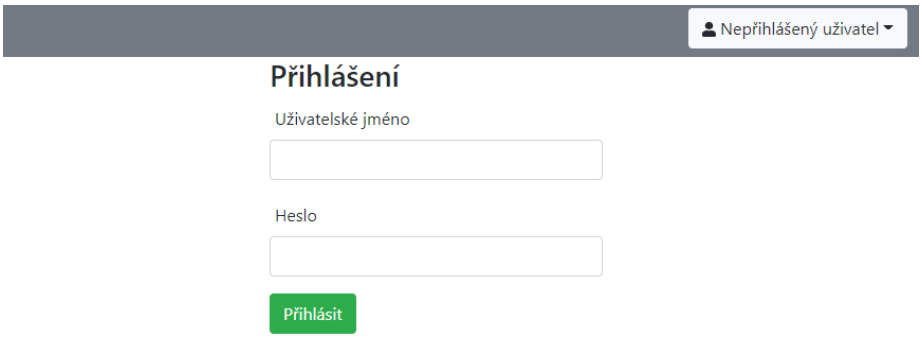
Obrázek 19: Ukázka dat získaných na základě požadavku GET - endpoint /parkingApp/client/tables a /parkingApp/client/relations

5.6 Zpřístupnění a správa dat – aplikace pro uživatele na straně klienta

Zpřístupnění dat uživatelům je realizováno pomocí frontendové aplikace, která je vytvořena pomocí javascriptové knihovny React. Pomocí této aplikace můžou uživatelé přistupovat k vytvořené RestAPI, zobrazovat si dostupné údaje, upravovat je a využívat.

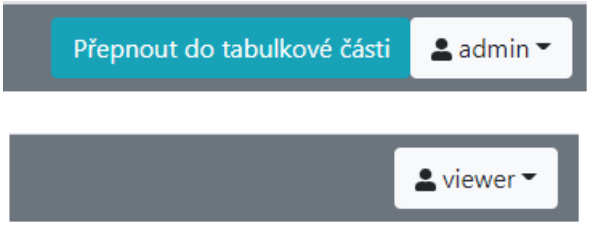
5.6.1 Přihlášení do aplikace

Do aplikace není umožněný volný přístup ani samostatná registrace, správu uživatelů provádí administrátor celého systému, který také vygeneruje každému oprávněnému uživateli jeho přístupové údaje. K přihlášení slouží úvodní stránka celé aplikace (Obr. č. 20).



Obrázek 20: Přihlašovací stránka

Aplikace má 2 základní části – nástěnku, kde se zobrazují informace o obsazenosti parkovišť, a tabulkovou část, kde je možné upravovat data jednotlivých objektů uložených v databázi. Uživatelé s rolí EDITOR a ADMINISTRATOR se do tabulkové části mohou dostat pomocí tlačítka „Přepnout do tabulkové části“ (Obr. č. 21). Uživateli s rolí VIEWER se po přihlášení zpřístupní pouze nástěnka aplikace, bez možnosti zobrazení jednotlivých tabulek. Další rozdíly v přístupu dle uživatelských práv jsou popsány dále v textu.

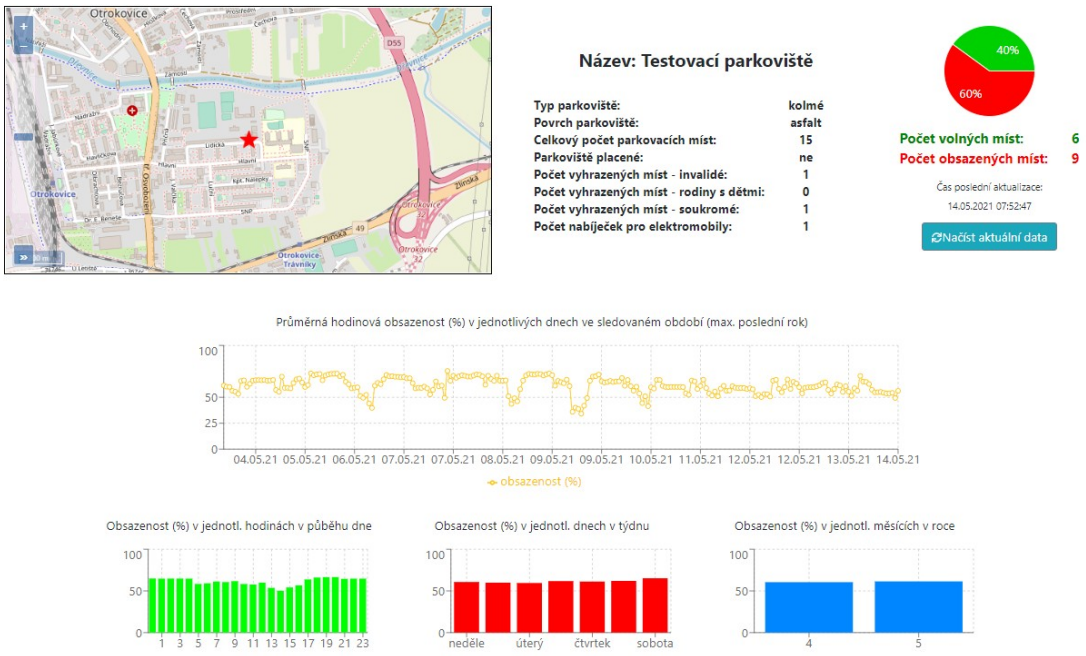


Obrázek 21: Rozdíl v hlavním menu aplikace dle přístupových práv přihlášeného uživatele

Uživatelské role jsou podrobněji popsány v kapitole 5.6.1.

5.6.2 Nástěnka – zobrazování aktuálních dat o obsazenosti

Nástěnka slouží pro zobrazování aktuálních údajů o obsazenosti parkovišť. Zobrazení pro jedno parkoviště má několik částí, jak je vidět na Obr. č. 22:



Obrázek 22: Nástěnka - data pro 1 parkoviště

- Situování parkoviště - je zobrazeno bodově, pomocí interaktivní mapy. Souřadnice se přebírají pomocí RestAPI z geoprostové nástavby databáze.
- Přehled údajů o parkovišti – základní charakteristiky parkoviště
- Aktuální údaje o celkové obsazenosti pomocí koláčového grafu, s časem poslední aktualizace. Data o obsazenosti se aktualizují v pravidelném intervalu, který je v aplikaci nastaven na 1 minutu. Okamžitou aktualizaci je možno provést pomocí tlačítka „Načíst aktuální data“. Interval aktualizace také záleží na nastavení intervalu, ve kterém se data ukládají do databáze (viz Tabulka č. 3 v kapitole 5.4).
- Graf zobrazující vývoj obsazenosti parkoviště v posledním období. Údaje jsou počítány z hodinových průměrů obsazenosti jednotlivých parkovacích míst. Graf je interaktivní, po najetí myší je možno si přečíst aktuální hodnotu obsazenosti (Obr. č. . 23).

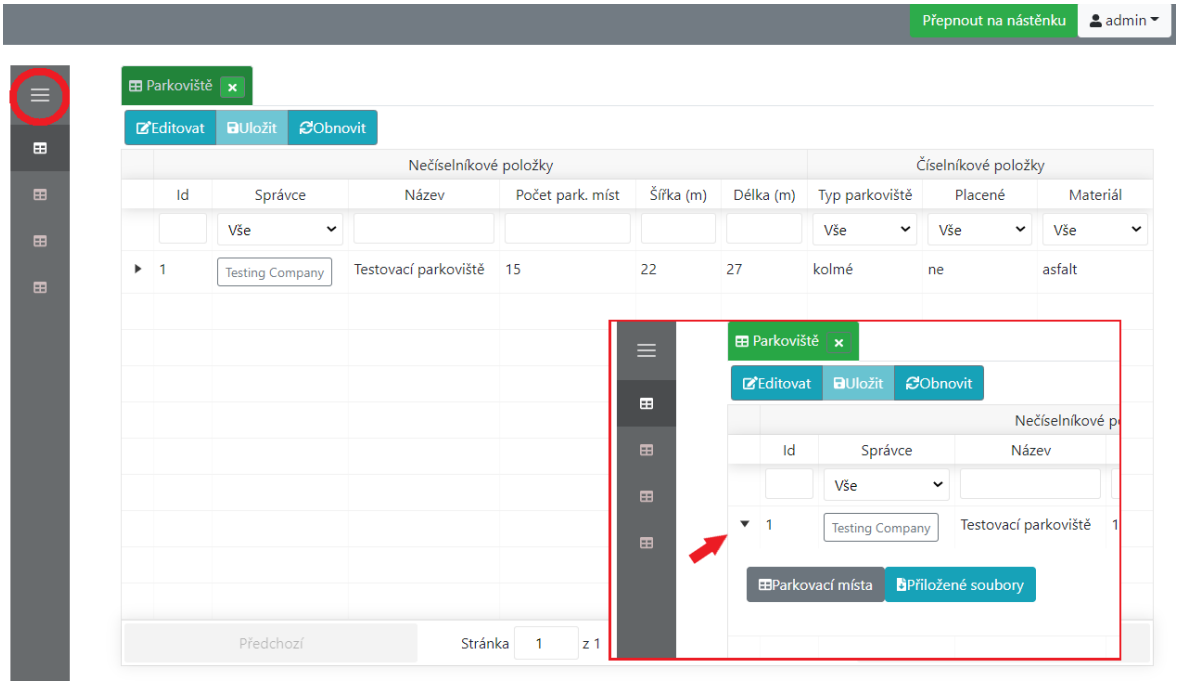
- Grafy, které statisticky vyhodnocují obsazenost parkoviště podle konkrétní hodiny v průběhu dne, podle dne v týdnu nebo měsíce v roce. Také u těchto grafů je možno si zobrazit konkrétní hodnotu po najetí myší.

Obrázek 23: Graf vývoje obsazenosti parkoviště

5.6.3 Tabulková část – správa parkoviště

Do tabulkové části celé aplikace se přepíná pomocí tlačítka „Přepnout do tabulkové části“, které není přístupné běžnému uživateli s oprávněním „VIEWER“. Zpět k přepnutí na nástěnku slouží stejné tlačítko, pouze s upraveným nápisem.

Po přepnutí se zobrazí tabulka Parkoviště, která obsahuje všechny záznamy o parkovištích v systému (Obr. č. 24 - je vložen pouze 1 záznam).

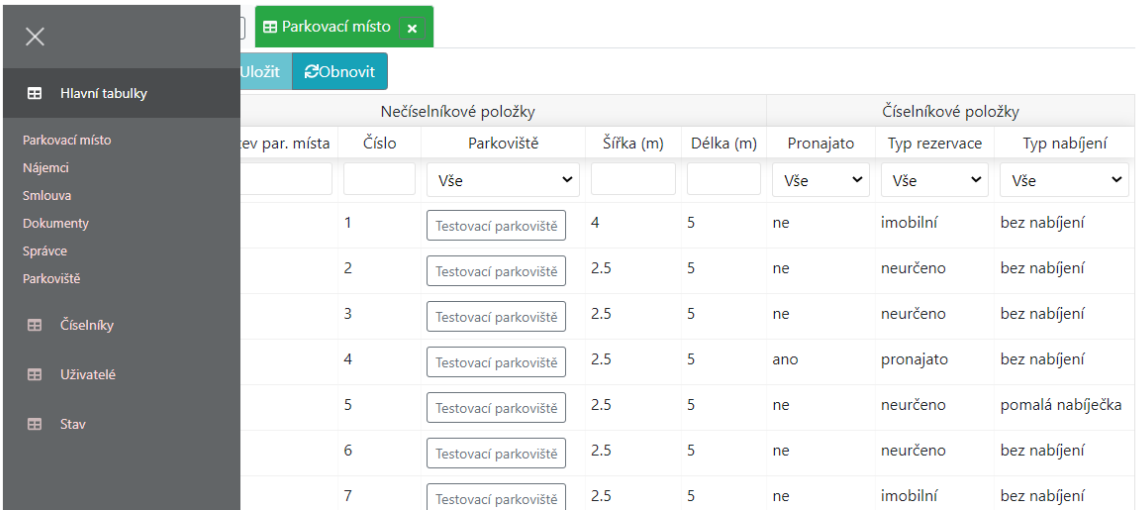


Obrázek 24: Vzhled tabulkové části aplikace - tabulka Parkoviště s výřezem výběru možností u záznamu

Po kliknutí na malou černou šipku u záznamu se otevře menu s výběrem možností daného záznamu (na obrázku výřez v červeném rámečku) – volbou šedého tlačítko s odkazem do podřízené tabulky (zde Parkovací místa) se otevře tabulka Parkovacích míst, modré tlačítko Přiložené dokumenty otevře okno, které umožňuje manipulaci s dokumenty a soubory (viz dále). Nabídka podřízených tabulek je pro každou tabulku jiná, například pro Parkovací místa nebo Nájemníky je to odkaz do tabulky Smlouvy, u Správce je to odkaz do tabulky Parkoviště.

Pokud je v tabulce položka některé nadřízené tabulky, je vložena jako odkaz a po jejím výběru se otevře nadřízená tabulka s vyfiltrovaným konkrétním záznamem. Na Obr. č. 24 je takovým odkazem sloupec správce, ve kterém vidíme název konkrétního správce.

Po levé straně tabulky se nachází lišta s výběrem tabulek, které jsou k dispozici (po najetí myší na symbol tabulky). Pokud vybereme horní tlačítko (na Obr. č. 24 v červeném kolečku), otevře se celá nabídka, ze které můžeme volit (Obr. č. 25). V části „Hlavní tabulky“ jsou zařazeny všechny základní tabulky, v části „Číselníky“ potom tabulky všech číselníků. V části „Uživatelé“, která je přístupná pouze uživateli s rolí administrátora, je možné zobrazit a upravovat tabulky s jednotlivými uživateli. V poslední položce „Stav“ je zařazena tabulka s aktuálním stavem (volno/obsazeno) jednotlivých parkovacích míst v rámci parkoviště.

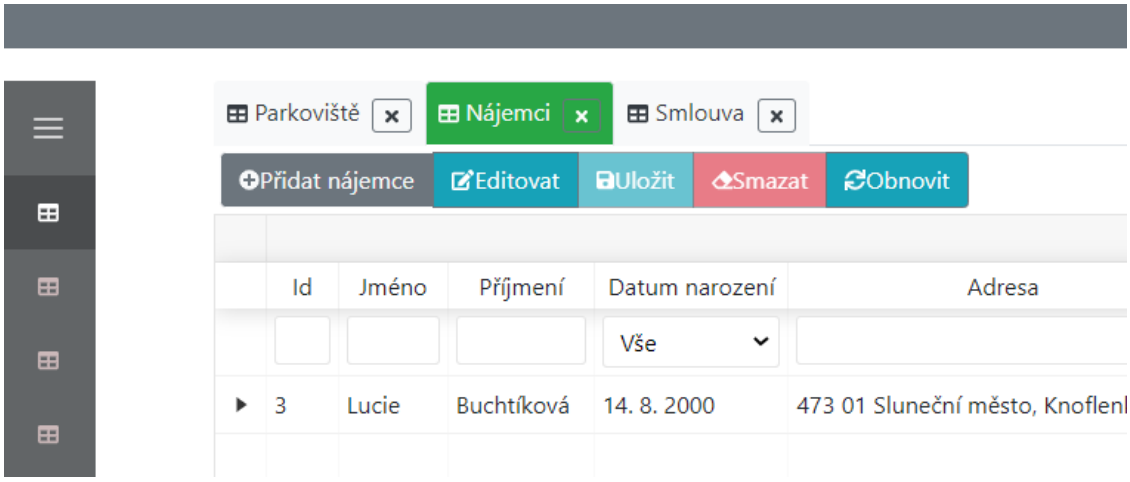


Obrázek 25: Lišta s nabídkou výběru tabulek

Jednotlivé tabulky se otevírají v horní liště. Pokud je chceme zavřít, musíme to udělat pomocí křížku v pravé části záložky.

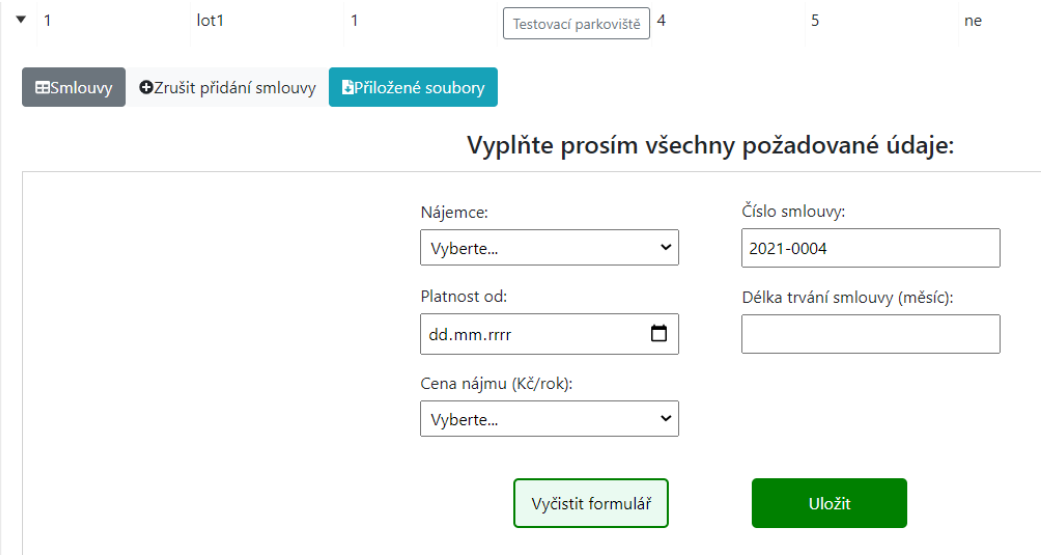
obecný popis, Lišta s tabulkami, jednotlivé tabulky a úpravy v nich, smlouvy, generování Jasper, propojení mezi tabulkami

U každé tabulky je v záhlaví k dispozici menu, které má základní položky stejné pro každou tabulku, přičemž u některých tabulek jsou některé položky navíc. Na Obr. č. 26 u otevřené tabulky Nájemci obsahuje menu následující položky:



Obrázek 26: Uspořádání hlavního menu u tabulek

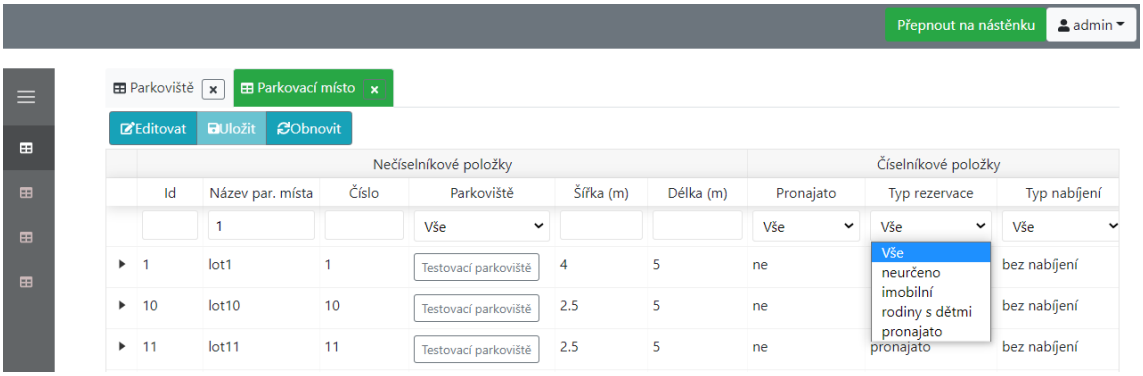
- **„Přidat nájemce“** – po jeho volbě se otevře formulář pro přidávání nového nájemce, který si chce pronajmout některé parkovací místo. Přidávání celého nového záznamu do databáze je možno pouze u tabulky Nájemci, a dále pak jako položku číselníku. Přidávání nového Správce, Parkoviště nebo Parkovacího místa se neprovádí v rámci aplikace, ale systémově, neboť je potřeba zároveň nastavit další části systému. Dále je možno přidat novou smlouvu, což se ale neprovádí z hlavního menu tabulky Smlouvy, ale protože se smlouva vytváří pro konkrétní parkovací místo, vkládá se přímo u tohoto parkovacího místa. Ukázka přidávání nové smlouvy k parkovacímu místu (lot1) je součástí Obr. č. 27 - nájemce se vybírá z uložených v databázi, u ceny nájmu se volí číselníková položka z číselníku Ceník. Nové číslo smlouvy se generuje automaticky z databáze.



Obrázek 27: Ukázka vkládání nové smlouvy k parkovacímu místu

- **„Editovat“** – toto tlačítko zpřístupní editační režim, ve kterém je možno upravovat některé položky záznamu. Pokud se položka upraví, zpřístupní se tlačítko **„Uložit“**, pomocí kterého se provedené změny uloží.
- **„Smazat“** – toto tlačítko je k dispozici pouze uživateli s rolí ADMINISTRATOR, a zároveň je možno ho využít pouze u tabulek Nájemci a Smlouva. Daný záznam se ve skutečnosti nevymaže, ale pouze se u něj nastaví hodnota systémového číselníku na „navrženo k odstranění“ a dále se již v aplikaci nezobrazuje.
- **„Obnovit“** – data dané tabulky se aktualizují na základě hodnot uložených v databázi (např. po přidání nebo vymazání položky).

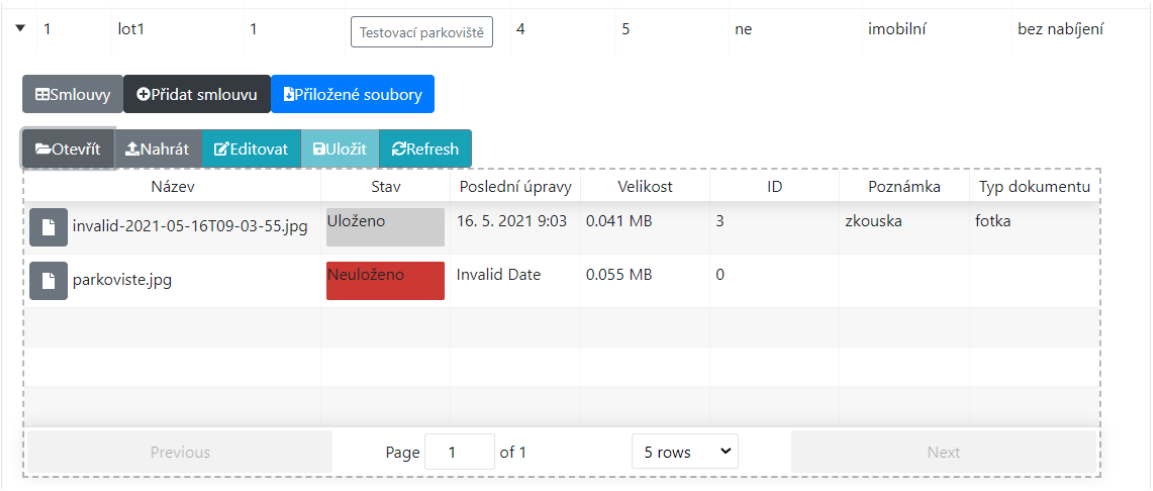
V záhlaví každého sloupce tabulky je samostatný řádek, který je určený pro filtrování (Obr. č. 28). Způsob filtrování se liší u každého typu údaje – textové údaje filtrují všechny záznamy, které obsahují jakýkoliv znak zadaného filtru, bez rozlišení malé – velké písmeno (ve sloupci název park. Místa je do filtru zadána 1), číselné hodnoty podle přesně zadaného čísla, u číselníkových položek je možnost z výběru konkrétního číselníku (sloupec Typ rezervace).



Obrázek 28: Ukázka filtrování v tabulce

Kliknutím na název sloupce dojde k seřazení položek tabulky podle vybraného sloupce.

K další funkcionalitě aplikace patří možnost nahrávání souborů (dokumentů). Dokument lze nahrát k libovolnému záznamu. Po rozkliknutí tlačítka „Přiložené soubory“ v menu pro konkrétní záznam (Obr. č. 24) se otevře tabulka s již nahranými dokumenty, které patří k záznamu, a samostatným menu, umožňujícím přidat nový soubor, nebo ho editovat. Na Obr. č. 29 je k záznamu parkovacího místa s názvem lot1 již přidán 1 soubor s názvem začínajícím invalid-2021, který je možno stáhnout do počítače pomocí ikony v levé části tabulky, a další soubor s názvem parkoviste.jpg byl teprve vybrán, ale ještě nebyl uložen - tomu dojde až po použití tlačítka „Nahrát“.



Obrázek 29: Ukázka přidávání dokumentu k parkovacímu místu

6 IMPLEMENTACE

Pro implementaci celého systému byly vytvořeny 3 aplikace. První dvě, aplikace pro ukládání dat o obsazenosti parkovacích míst do databáze a RestAPI jsou vytvořeny v programovacím jazyce Java, poslední z nich, frontendová aplikace, je implementována pomocí javascriptové knihovny React. Jako databáze byla použita relační databáze PostgreSQL s nadstavbou PostGIS, která přidává podporu pro geografické objekty.

6.1 Použité technologie

6.1.1 Jazyk JAVA

Objektově orientovaný jazyk Java je v současnosti jedním z nejpopulárnějších programovacích jazyků, ve kterých probíhá vývoj webových aplikací. Při vývoji bylo použito OpenJDK 11 ve verzi Zulu.

6.1.2 Spring framework

Spring framework je vícevrstvý aplikační rámec , který usnadňuje vývoj Java webových aplikací. Patří do třídy tzv. lightweight (odlehčených) JEE kontejnerů [7].

Spring je rozdělený do jednotlivých modulů, přičemž je možno vybrat si jen ty moduly, které jsou potřeba. Základ tvoří tzv. „Core“ kontejner s moduly Beans, Core, Context a Expressions, které zahrnují konfigurační model a mechanismus „dependency injection“. Skupina modulů „Data Access/Integration“ obsahuje kromě jiných také modul JDBC, který umožňuje přístup k databázím. K dalším skupinám modulů patří Web, AOP (Aspect Oriented Programming), Instrumentation a Test.

Jádro Springu je postaveno na využití návrhového vzoru Inversion of Control, který řeší vazby mezi jednotlivými komponentami. Díky Spring frameworku je možno vytvářet program jako skupinu vzájemně nezávislých komponent, které se propojí až po spuštění programu. Způsob propojení se určuje pomocí XML souboru, takže jednotlivé závislosti není nutné psát do zdrojového kódu [6].

Pro rychlé spuštění celé aplikace byl využit modul Spring Boot, který slouží k tvorbě soběstačných (stand-alone) aplikací s minimální nutností konfigurace. Tento modul již v sobě obsahuje webserver Tomcat, Jetty nebo Undertow, díky čemuž odpadá nutnost nasazování souboru war. Dále obsahuje startovací konfiguraci POM (Project Object Model) souboru, čímž zjednodušuje konfiguraci Mavenu, a automaticky při každé vhodné příleži-

tosti konfiguruje Spring. Také nabízí předpřipravené nástroje na měření metrik, kontroly aktivity a externí nastavení. Funguje bez nutnosti jakéhokoliv generování kódu a XML konfigurací.

6.1.3 Apache Maven

Maven je nástroj od společnosti Apache Software Foundation, který slouží ke správě a sestavování aplikací postavených nad platformou Java [5]. Jeho využitím odpadá závislost na konkrétním vývojovém prostřed a jeho formát je akceptován všemi vývojovými prostředími Javy.

Základním principem fungování Mavenu je popis projektu pomocí Project Object Model (POM). Tento model popisuje softwarový projekt včetně jeho závislostí na externích knihovnách a různých funkcí s tím spojených. Popis je uložen v souboru pom.xml v kořenovém adresáři projektu. Jednou ze zásadních výhod je řešení závislostí – závislosti na externích knihovnách jsou uložené v tomto souboru v tzv. „Dependencies“ [6].

6.1.4 Hibernate

Hibernate je framework napsaný v jazyce Java, který umožňuje tzv. objektově relační mapování (ORM), a používá se k zajištění perzistence dat. Provádí mapování základních tříd (Entity) na jednotlivé databázové tabulky, a to buď pomocí mapovacích souborů ve formátu XML, nebo pomocí anotací.

6.1.5 React

React je open-source JavaScriptová knihovna, která umožňuje vytvářet uživatelská rozhraní pro webové aplikace. Základním stavebním blokem jsou Komponenty, zapisované pomocí tzv. JSX (JavaScript XML) - kombinace HTML tagů a vlastního Javascriptu. Každá komponenta má svůj vlastní stav, který si sama mění na základě různých událostí nebo pokynů zvenčí, a během svého vývoje prochází různými fázemi, tzv. životním cyklem, kdy každé fázi je přiřazena určitá metoda. Komponenty mohou být třídní nebo funkcionální [4].

Při vývoji v Reactu se využívá prostředí Node.js a NPM (Node Package Manager), který umožňuje instalaci a správu balíčků, které používají javascriptové aplikace.

6.1.6 Redux

Redux je open-source JavaScriptová knihovna, která se používá pro správu stavu aplikace [3]. Nejčastěji se používá společně s React nebo Angularem. Základem Reduxu jsou 3 objekty – store, akce a reducer. Store je jakýsi sklad, ve kterém jsou uložena všechna data aplikace. Akce popisuje změnu stavu, a reducer tuto změnu provede [2].

6.1.7 Vývojové prostředí Intellij IDEA Ultimate

Intellij IDEA Ultimate je vývojové prostředí společnosti JetBrains, které má podporu všech výše zmíněných technologií.

6.2 Ukládání získaných dat – aplikace SaveParking

Tato aplikace nemá žádné uživatelské rozhraní. Slouží pouze k tomu, aby v pravidelném intervalu poslala GET požadavek na RestAPI, která poskytuje data s údaji o obsazenosti jednotlivých parkovacích míst.

K závislostem, které byly v rámci projektu použity, patří kromě SpringBoot frameworku pouze JPA (Java Persistence Api) v implementaci Hibernate a postgreSQL.

Konfigurační údaje pro připojení k databázi se nastavují v souboru application.properties. Kromě těchto údajů se zde zadává také URLa časový interval pro zaslání požadavku GET, a dále id příslušného parkoviště, které je zadáno v databázi (blíže viz. kapitola č. 5.4.

Databázová tabulka parking_place_state je namapována pomocí Entity ParkingPlaceState s kompozitním klíčem ParkingPlaceStateId s anotací @EmbededId. Třída kompozitního klíče má anotaci @Embedable, přepisuje metody equals(Object o) a hashCode() a obsahuje 3 základní proměnné:

```
@Column(name = "fk_carpark")
private Long fkCarpark;

@Column(name = "fk_parking_place")
private Long fkParkingPlace;

@Column(name = "time_stamp")
private LocalDateTime timeStamp;
```

Pro mapování dat z RestAPI ve formátu JSON (Obr. č. 6) byly vytvořeny 3 třídy DTO (Data Transfer Objects) - InputJson, StatusReply a Lots. Komunikace s databází probíhá pomocí Repository ParkingPlaceStateRepository, která rozšiřuje JpaRepository.

Zasílání požadavku GET probíhá pomocí metody s anotací `@Scheduled(fixedRateString = "${PERIOD_IN_MILLISECONDS}")` (org.springframework.scheduling), která zajistí pravidelné vykonávání metody s touto anotací. Časový interval je načítaný z application.properties a je nastaven na 1 minutu.

Vlastní požadavek je definován v třídě RequestService, která vypadá následovně:

```
@Service
public class RequestService {

    @Value("${URL}")
    String url;

    private final HttpClient httpClient = HttpClient.newBuilder()
            .authenticator((Authenticator)(new Authenticator() {
                @NotNull
                protected PasswordAuthentication getPasswordAuthentication() {
                    return new PasswordAuthentication("myusername",
                      mypassword".toCharArray());
                }
            }))
            .version(HttpClient.Version.HTTP_2)
            .build();

    public String sendGet() throws Exception {
        HttpRequest request = HttpRequest.newBuilder()
                .GET()
                .uri(URI.create(url))
                .build();

        HttpResponse<String> response = httpClient.send(request,
               HttpResponse.BodyHandlers.ofString());

        return response.body().toString();
    }
}
```

Vlastní převod načtených dat na formát Entity probíhá pomocí třídy ParkingConventor s anotací @Service, která provede také uložení do databáze.

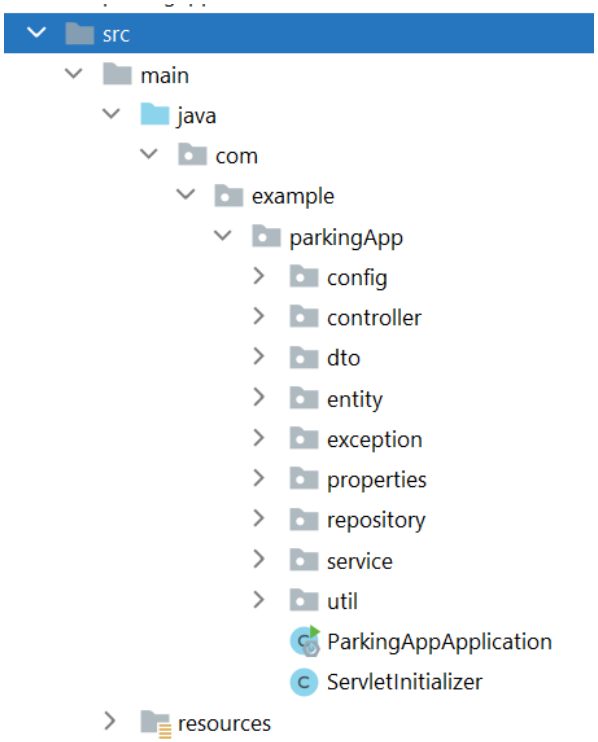
Zdrojový kód aplikace je součástí přílohy na CD.

6.2.1 RestAPI ParkingApp

V souboru Pom.xml, který je konfiguračním souborem nástroje Maven, jsou kromě závislostí uvedených již u předchozí aplikace přidány také další závislosti pro spring boot – security, hibernate-spatial, postgis-jsbc, jackson-datatype-jts, modelmapper, jjwt, validation-api, springfox-swager2, springfox-swagger-ui, lombok, jackson a další.

Další konfigurační údaje, jako je připojení k databázi, nastavení Hibernate, adresář pro ukládání souborů a další se nastavují v souboru application.properties.

Struktura aplikace je rozdělena do základních balíčků (packages) entity, dto, repository, service, controller, config, util a exception (obr. č. 30).



Obrázek 30: Základní struktura balíčků apliace parkingApp

- Package Entity – obsahuje třídy pro mapování databázových tabulek. Kromě základních entit obsahuje dále balíčky client, interfaces, lookups a sys. V entitách jsou definovány jednotlivé proměnné včetně relací do propojených tabulek. Dále uvádíme příklad namapování tabulky parking_place k tabulce carpark a opačně:

```
@LazyCollection(LazyCollectionOption.TRUE)
@OneToMany(mappedBy = "carpark", fetch = FetchType.LAZY)
@JsonIgnore
private List<ParkingPlace> parkingPlaceList = new ArrayList<>();

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "fk_carpark")
private Carpark carpark;
```

- Package Dto – obsahuje třídy, které jsou vytvořené pro úpravu dat z tabulek v databázi tak, aby mohly být předávány uživatelům RestAPI. Jejich struktura vzniká úpravou entit, jejich propojením, sloučením nebo naopak ořezáním.
- Package repository – tyto třídy slouží pro komunikaci s databází. Každá Entita má svoji odpovídající Repository
- Package service – obshuje servisní třídy, která převádějí entity na dto a zpět, dále vytvářejí servis pro ukládání dat do databáze a obsahují další logiku aplikace. Třída

ContractService například mimo jiné obsahuje logiku pro vygenerování nového čísla smlouvy. Třída FileStorageService zase obsahuje metody související s ukládáním a načítáním souoborů.

- Package controller – obsahuje třídy, které slouží k zachytávání požadavků na zdroje. Příkladem může být metoda getAll() ze třídy ContractController, která zachytí požadavek GET, z databáze načte všechny smlouvy, pomocí metody fromEntity(contract) je převede je na výstupní formát dle třídy ContractDto a tato data vrátí uživateli. Obdobným způsobem jsou zpracovány i další metody, odpovídající požadavků POST, PUT nebo DELETE.

```
@GetMapping("")
public List<ContractDto> getAll() {
 final List<ContractDto> contractDtoList = new ArrayList<>();
 final List<Contract> contractList = contractService.getAll();
 for (Contract contract : contractList) {
 contractDtoList.add(contractDtoService.fromEntity(contract));
 }
 return contractDtoList;
}
```

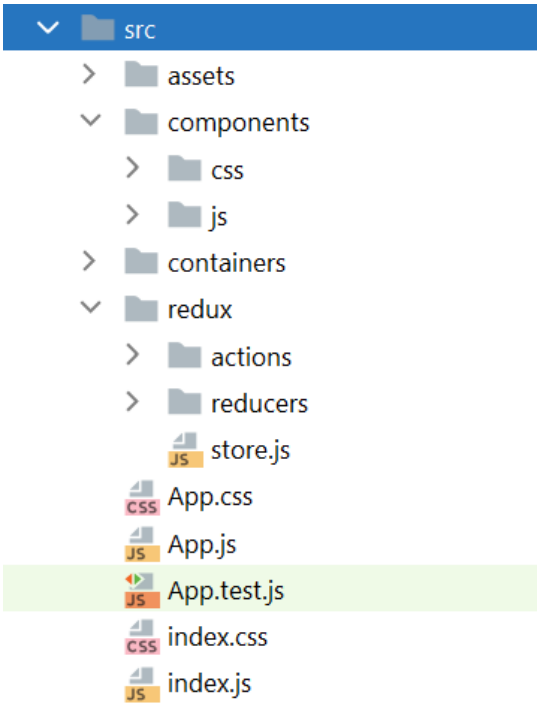
- Package config obsahuje třídy, které se starají o kofiguraci celé aplikace. Patří sem například třída JwtSecurityConfig, kde se nastavuje bezpečnost a přístupová práva, nebo konfigurační soubory pro autentifikaci.

6.3 Zpřístupnění a správa dat – aplikace pro uživatele na straně klienta

Frontendová aplikace pro komunikaci s klientem je vytvořena ve frameworku React, s využitím Reduxu.

V souboru package.json jsou uloženy všechny balíčky, které se používají v rámci aplikace. Jedná se mimo jiné o bootstrap, formik (formuláře), react, react-dom, react-redux, react-scripts, react-table (tvorba tabulek), reactstrap, recharts (grafy), redux.

Struktura vlastní aplikace je patrná z Obr. č. 31.



Obrázek 31: Základní struktura frontendové aplikace

Základním vstupním bodem do aplikace je soubor index.js. Z něj se potom generují další komponenty. V balíčku containers jsou uloženy komponenty, které přebírají stav ze storu a předávají ho dále do svých potomků (balíček components), kteří slouží k vlastnímu generování (neboli renderování) komponent. V balíčku redux jsou potom uloženy jednotlivé akce, reducery a store.

Store je vytvořen následujícím kódem (soubor store.js):

```
import { createStore, applyMiddleware } from "redux";
import thunk from "redux-thunk";
import rootReducer from "./reducers";

export default createStore(
    rootReducer,
    applyMiddleware(thunk)
);
```

Dále uvádím příklad reduceru statistics.js, kdy v úvodu jsou naimportovány akce, které se v rámci tohoto reduceru obsluhují, dále je nadefinován iniciální stav a obsluha jednotlivých akcí.

```
import {
    FETCH_STATISTICS_BEGIN,
    FETCH_STATISTICS_SUCCESS,
    FETCH_STATISTICS_FAILURE,
    FETCH_PARKING_PLACE_STATE_BEGIN,
    FETCH_PARKING_PLACE_STATE_SUCCESS,
    FETCH_PARKING_PLACE_STATE_FAILURE,
} from '../actions/serverTypes';

const initialState = state => ({
    loading: false,
    error: null,
    carparks: [],
    });

const statistics = (state = initialState, action) => {
    switch (action.type) {
        case FETCH_STATISTICS_BEGIN:
            return {...state, loading: false, error: null};
        case FETCH_STATISTICS_SUCCESS:
              return {...state, carparks: action.payload, loading: true, error:
null};
        case FETCH_STATISTICS_FAILURE:
            return {...state, loading: false, error: action.error};
        case FETCH_PARKING_PLACE_STATE_BEGIN:
            return {...state, loadingParkPlState: false, errorParkPlState: null};
        case FETCH_PARKING_PLACE_STATE_SUCCESS:
             return {...state, actualState: action.payload, loadingParkPlState:
true, errorParkPlState: null};
        case FETCH_PARKING_PLACE_STATE_FAILURE:
             return {...state, loadingParkPlState: false, errorParkPlState: acti-
on.error};
        default:
            return state
    }
};
export default statistics
```

V balíčku redux je v podbalíčku actions uložen soubor index.js, ve kterém jsou nadefinovány všechny obslužné metody – např. metoda getJSON, která vyzvedává data z RestAPI.

```
function  getJSON(endpoint) {
    const url = createURL(endpoint);
    return fetch(url, {
        headers: {
            'Authorization': sessionStorage.getItem(TOKEN_SESSION_ATTRIBUTE_NAME)
        }
    })
        .then(handleErrors)
        .then(res => res.json());
}
```

Velmi jednoduchým příkladem kontejneru a následné komponenty je SPost a SignePost, které slouží jako rozcestník mezi nástěnkou a tabulkovou částí.

Spost.js kromě importů obsahuje 2 základní metody – mapStateToProps a mapDispatchToProps, které pomocí metody connect předávají do komponenty SignePost:

```
const mapStateToProps = (state, ownProps) => ({
    location: ownProps.location,
    tabs: state.tabs,
    tree: state.tree,
    tables: state.tables,
    dashboard: state.general.dashboard,
    state: state,
    props: ownProps,
});

const mapDispatchToProps = dispatch => ({
    sendNewForm: (id, endpoint, params) => dispatch(sendNewForm(id, endpoint,
params)),
    toggleTableToolbarForm: (table) => dispatch(toggleTableToolbarForm(table)),
    fetchUserRoles: (username, endpoint) => dispatch(fetchUserRoles(username,
endpoint)),
});

export default connect(mapStateToProps, mapDispatchToProps)(SignePost)
```

SignePost.js v konstruktoru volá metodu fetchUserRoles, pomocí které zjistí, jaké role má přihlášený uživatel, a poté se podle stavu parametru this.props.dashboard (který přejímá z kontejneru Spost) rozhoduje, jestli vygeneruje komponentu Layout nebo komponentu Dashboard:

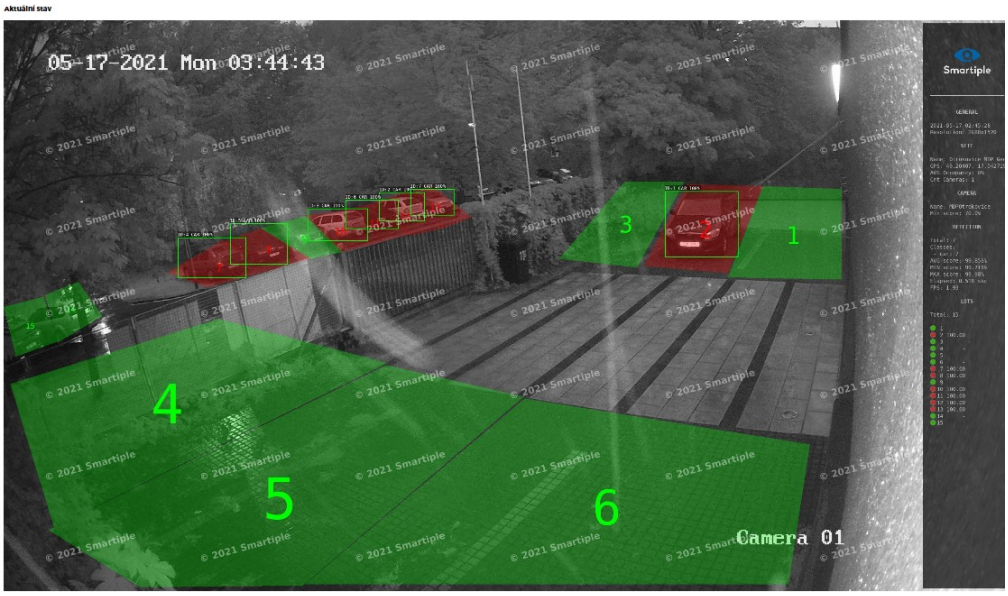
```
class SignePost extends Component {
    constructor(props) {
        super(props);
        this.props.fetchUserRoles("role/user/" +
sessionStorage.getItem("authenticatedUser"));
    }
    render () {
        return (

            <div style={{padding: "20px"}}>
                {!this.props.dashboard &&
                <Layout {...this.props.props}/>
                }
                {this.props.dashboard &&
                <Dashboard {...this.props.props} />
                }
            </div>
        )
    }
}
export default SignePost;
```

V rámci aplikace se generují grafy, pro které byl použit balíček recharts. Pro generování tabulek se používá balíček react-tables a s tím související react-strap (ekvivalent bootstrapu pro react).

7 HARDWAROVÁ REALIZACE A TESTOVÁNÍ

Vlastní realizace celého systému a jeho testování bylo zahájeno 21.4.2021, kdy byla nasazena aplikace saveParking pro ukládání dat do databáze. S předstihem byla instalována IP kamera na vybrané parkoviště, které je umístěno v Otrokovicích u firmy MDP GEO, s.r.o. Jedná se kameru , která je podrobněji popsána v kapitole č. 5.1 a je zobrazena na Obr. č. 4. Kamera je umístěna na čelní stěně budovy a obsáhne 15 parkovacích míst. Po instalaci kamery proběhlo natrénování modelu na určování obsazenosti parkovacích míst, který zajišťuje firma Smartiple. Obraz z kamery je pro model zpřístupněn na zabezpečené URL adrese.

Na obr. č. 32 je vidět již vyhodnocený výstup z kamery v nočních hodinách, kdy je využito infračerveného přísvitu.



Obrázek 32: Výstup z kamery včetně vyhodnocení obsazenosti v nočních hodinách

Po celou dobu sběru dat probíhal vývoj restAPI klientské části aplikace, kdy nasbíraná data sloužila také k testování vyvíjených aplikací. Výsledek celé realizace a testování je vidět na základě prezentace dat v aplikaci.

Z výsledku realizace systému a jeho testování můžeme konstatovat následující skutečnosti:

- Nainstalovaná kamera v rámci daných podmínek pokryje 15 parkovacích míst, u kterých je možno bez problémů provádět vyhodnocení obsazenosti.
- Díky infračervenému přísvitu není problém s vyhodnocováním ani v noci

- Aplikace byla testovaná pouze pro 1 parkoviště. I tak je nasbírané množství údajů, které se ukládají do databáze, velké, a načítání většího množství údajů může být pomalejší. Při využití pro více parkovacích míst, případně pro více parkovišť, bude potřeba aplikaci znovu otestovat z hlediska rychlosti komunikace mezi jednotlivými částmi systému. Je možné, že bude potřeba přehodnotit dobu, po kterou jsou data ponechána v databázi, eventuálně množství dat, která jsou načítána do aplikace pro zobrazování průběžných údajů obsazenosti.
- Na základě testování celého systému můžeme říci, že je pro dané účely dobře využitelný.

ZÁVĚR

Výsledkem této práce je navržení základu funkčního systému, který umožní nejen monitorování aktuálního stavu parkoviště, ale také jeho efektivní správu. Systém bude sbírat data o obsazenosti jednotlivých parkovacích míst a tato data ukládat, spravovat a prezentovat, včetně souhrnných statistických údajů.

Jako první krok po provedení základní rešerše bylo navržení hardwarového řešení, kdy se v podstatě jako jediné řešení ukázalo monitorování pomocí IP kamery. Jelikož vyhodnocování obsazenosti pomocí AI je realizováno externí firmou a není součástí této práce, dalším krokem bylo navržení architektury jednotlivých aplikací a jejich implementace.

Jak je uvedeno již v úvodní části, návrh celého systému byl vytvořen se snahou o možnost jeho dalšího postupného rozšiřování přidáváním nových částí či modulů tak, aby v budoucnu mohl sloužit zájemcům o jeho využívání, případně o jeho začlenění do již funkčního systému, který zahrnuje např. Správu budov, komunikací, zeleně a dalšího. S tímto přístupem byl také proveden výběr vhodné databáze, kdy základním požadavkem bylo umožnění práce s geoprostorovými daty. Z tohoto důvodu byla zvolena databáze PostGreSQL s nadstavbou PostGIS, která má zároveň výhodu Open Source řešení. V rámci daného systému je nicméně možno použít i jinou databázi, pokud bude zachována stejná struktura.

Poté byla vytvořena aplikace pro ukládání dat do databáze, která byla přímo přizpůsobena formátu dat, který je výsledkem vyhodnocení obsazenosti. Jakmile byla aplikace nasazena, byly zároveň k dispozici první údaje, které sloužily pro testování při vývoji RestAPI a následně frontendové aplikace. V průběhu vývoje byla ještě mírně upravena struktura databáze.

Architektura a implementace všech tří aplikací jsou popsány v kapitolách č. 5 a 6. Vzhledem k jejich velkému rozsahu nebylo možno popsat všechny detaily. Kompletní zdrojový kód je součástí příloh.

V současné době je celý systém nasazený a stále probíhá sběr dat.

SEZNAM POUŽITÉ LITERATURY

[1] Car Detection Systems - LumiGuide. LUMIGUIDE Smart Mobility Solutions - LumiGuide [online]. Dostupné z: https://lumi.guide/smart-parking-management/car-detection-system/

[2] Redux + React - Redux - Zdroják. Zdroják - o tvorbě webových stránek a aplikací [online]. Dostupné z: https://zdrojak.cz/clanky/redux-react-23-redux/

[3] Redux - A predictable state container for JavaScript apps. | Redux. Redux - A predictable state container for JavaScript apps. | Redux [online]. Copyright © 2015 [cit. 17.05.2021]. Dostupné z: https://redux.js.org/

[4] React – A JavaScript library for building user interfaces. React – A JavaScript library for building user interfaces [online]. Copyright © 2021 Facebook Inc. [cit. 17.05.2021]. Dostupné z: https://reactjs.org/

[5] Maven – Welcome to Apache Maven. Maven – Welcome to Apache Maven [online]. Dostupné z: http://maven.apache.org/

[6] SEDLÁČKOVÁ, Jana. Internet věcí (IoT) - vzdálený monitoring a analýza dat z hydrogeologického vrtu. Bakaláská práce. Zlín: Univerzita Tomáše Bati ve Zlíně, 2017. Dostupné také z: http://hdl.handle.net/10563/43500. Univerzita Tomáše Bati ve Zlíně. Fakulta aplikované informatiky, Ústav automatizace a řídicí techniky.

[7] Spring Framework Documentation. 301 Moved Permanently [online]. Copyright © 2002 [cit. 17.05.2021]. Dostupné z: https://docs.spring.io/spring-framework/docs/current/reference/html/

[8] 4. Co je to REST API - Node.js tutoriál - Parse-error.cz - nikdy nevíš, kdy to přijde. Parse-error.cz - nikdy nevíš, kdy to přijde [online]. Copyright © 2021 Parse [cit. 16.05.2021]. Dostupné z: https://www.parse-error.cz/nodejs-tutorial/4/co-je-to-rest-api

[9] Web Application Architecture: A Guide Through the Intricate Process of Building an App | LITSLINK Blog. Software Development Company in US - LITSLINK. Top Software Developer [online]. Copyright © 2021 LITSLINK [cit. 16.05.2021]. Dostupné z: https://litslink.com/blog/web-application-architecture

[10] Web Application Architecture: How the Web Works | AltexSoft. AltexSoft - Technology & Solution Consulting Company [online]. Copyright © Copyright [cit.

16.05.2021]. Dostupné z: https://www.altexsoft.com/blog/engineering/web-application-architecture-how-the-web-works/

[11] API Documentation & Design Tools for Teams | Swagger. API Documentation & Design Tools for Teams | Swagger [online]. Copyright © 2021 SmartBear Software. All Rights Reserved. [cit. 13.05.2021]. Dostupné z: https://swagger.io/

[12] VIAKOM | HIKVISION DS-2CD2045FWD-I IP kamera. VIAKOM - DOVOZCE pro KAMEROVÉ SYSTÉMY [online]. Dostupné z: https://www.viakom.cz/hikvision-ds-2cd2045fwd-i-2-8mm/product-5042

[13] Custom Vision | Microsoft Azure. Object moved [online]. Copyright © 2021 Microsoft [cit. 12.05.2021]. Dostupné z: https://azure.microsoft.com/cs-cz/services/cognitive-services/custom-vision-service/

[14] STRÁNSKÝ, Václav. Vizuální systém pro detekci obsazenosti parkoviště pomocí hlubokých neuronových sítí [online]. Brno 2017 [cit 2021-05-12]. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií, Ústav inteligentních systémů. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=159072

[15] Test 10 nejlepších IP kamer 2021 | Recenze & Jak vybrat | ARecenze.cz. Nezávislé recenze a srovnávací testy nejlepších produktů | ARecenze [online]. Copyright © Procontent s.r.o. [cit. 12.05.2021]. Dostupné z: https://www.arecenze.cz/ip-kamery/

[16] LoRaWAN Parking Sensor Detects Events In Real-time - MOKOSmart #1 Smart Device Solution in China. Home - MOKOSmart #1 Smart Device Solution in China [online]. Copyright © MOKOSmart [cit. 11.05.2021]. Dostupné z: https://www.mokosmart.com/lorawan-parking-sensor/?gclid=CjwKCAjw1uiEBhBzEiwAO9B_HS2u1xFPUTnCnjUkfBRRWM4lO8rDR7FRs8cFuVKQ2RVVJVK3JGPKbxoCC5wQAvD_BwE

[17] Smart Parking Lot Sensor | Bosch Connected Devices and Solutions. IoT & Industry 4.0 sensor devices | Bosch Connected Devices and Solutions [online]. Copyright © Bosch Connected Devices and Solutions GmbH [cit. 11.05.2021]. Dostupné z: https://www.bosch-connectivity.com/products/connected-mobility/parking-lot-sensor/

[18] Parkovací systémy | Siemens Česká republika - Czech Republic. 302 Found [online]. Copyright © Siemens, s.r.o. 1996 [cit. 11.05.2021]. Dostupné z: https://www.siemens.cz/smartcities/parkovaci-systemy

[19] Chytré řešení pro parkoviště | ParkingDetection.com. Smart solution for parking lots | ParkingDetection.com [online]. Dostupné z: https://www.parkingdetection.com/cs/domu/

[20] Výhody pre vodičov - ParkDots. Výhody pre vodičov - ParkDots [online]. Copyright © 2017 [cit. 11.05.2021]. Dostupné z: https://parkdots.com/

[21] Smartiple | Poznáme obsazenost parkovišť kamerou.. Smartiple | Poznáme obsazenost parkovišť kamerou. [online]. Copyright © 2021. Všechna práva vyhrazena. [cit. 11.05.2021]. Dostupné z: https://smartiple.com/#parkinto

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AI	Umělá inteligence (Artificial Intelligence)
CSS	Kaskádové styly – Cascading Style Sheets
DTO	Data Transfer Objects
GDPR	Obecné nařízení o ochraně osobních údajů (General Data Protection Regulation)
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet věcí (Internet of Things)
IP	Internet Protocol
JDK	Java Development Kit
JSON	JavaScript Object Notation – formát výměny dat
JWT	JSON Web Token
LAN	Lokální síť (Local Area Network)
RFID	Radio Frequency Identification
SDK	Software Development Kit
SSH	Secure Shell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UTC	Koordinovaný světový čas (Coordinated Universal Time)
WiFi	Označení pro několik standardů IEEE 802.11
XML	Extensible Markup Language

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P 1: CD

Obsah přiloženého CD:

- Diplomová práce v elektronické podobě
- Zdrojový kód programů všech 3 aplikací