FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Správa parkovacího systému/domu
Student:	Tomáš Kovářík
Vedoucí:	Ing. Josef Pavlíček, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce je vytvoření aplikace umožňující správu parkovacích míst.
- Tato aplikace bude obsahovat jednoduché zákaznické webové stránky, kde zákazníci budou moci sledovat aktuální stav naplněnosti parkoviště, mimořádné události, ovládat uživatelskou sekci, sledovat stav zaplacení parkovného a rezervovat si parkovací místo.
- Další částí aplikace bude implementace internetového API, které budou moci využívat především externí aplikace provozovatele. (REST API)
- Součástí aplikace by měla být implementace základní vzorové jednoduché služby (UDP serveru), která bude zpracovávat požadavky jednoduchých externích zařízení (např. výjezdové brány).
- Aplikaci vytvořte, otestujte a definujte závěry.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 13. října 2020 v Praze.

Bakalářská práce

SPRÁVA PARKOVACÍHO SYSTÉMU/DOMU

Tomáš Kovářík

Fakulta informačních technologií ČVUT v Praze
Katedra softwarového inženýrství
Vedoucí: Ing. Josef Pavlíček, Ph.D.
11. května 2021

Odkaz na tuto práci: Tomáš Kovářík. *Správa parkovacího systému/domu*. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Rád bych poděkoval vedoucímu práce Ing. Josefovi Pavlíčkovi,Ph.D. za odborné vedení, všestrannou pomoc a cenné rady při zpracovávání bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 10. května 2020 .....

Abstrakt

Bakalářská práce se zabývá informačním systémem umožňujícím centrální správu parkovišť a parkovacích míst. Analyzuje dostupná řešení, představuje návrh používaného rozhraní a implementuje komplexní nástroj složený z více menších částí, které jsou spojeny pomocí technologie Docker. Automaticky dokumentované aplikační rozhraní, založené na principu REST, slouží ke komunikaci implementovaných služeb, včetně dalšího použití externími systémy provozovatele. Mezi služby využívající veřejně dostupné rozhraní patří statické webové stránky založené na technologii React. Tyto stránky umožňují prezentaci aktuálního stavu spravovaných parkovišť, rezervaci míst a kontrolu plateb pomocí systému uživatelských účtů. Výsledkem práce je jednoduše použitelný systém podporující různé druhy parkování.

Klíčová slova správa parkoviště, rezervační systém, platba parkování, vjezdová brána, Spring, REST, React

Abstract

This thesis deals with an information system allowing centralized management of carparks and parking spaces. It analyzes the available solutions, presents the design of the application interface and implements a comprehensive tool composed of several smaller parts, which are connected using Docker technology. The automatically documented application interface, based on the REST principle, is used for communication between services and it is also available for additional external systems. Among services using publicly available interface includes static websites based on technology React. The site allows the presentation of the current status of managed parking lots, reservation of spots and control of payments using a system of user accounts. The result of the thesis is a quickly deployable system supporting various types of parking.

Keywords parking management, reservation system, parking payment, entry gate, Spring, REST, React

Seznam zkratek

ACL	Access Control List
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BPMN	Business Process Model and Notation
DB	Database
DI	Dependency injection
DOM	Document Object Model
DTO	Data Transfer Object
HATEOAS	Hypermedia as the Engine of Application State
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
ISO-OSI	International organization of Standardization - Open System Interconnection
ITF	Interleaved 2 of 5
IoC	Inversion of control
IoT	Internet of Things
JS	Javascript
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
NFC	Near field communication
REST	Representational state transfer
SPA	Single-page application
SPZ	Státní poznávací značka
SQL	Structured Query Language
SSE	Spring Security Expressions
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UI	User interface
URI	Uniform Resource Identifier
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

Kapitola 1
Úvod

Dostatečná kapacita a dostupnost volných parkovacích míst je problémem, který musí řešit velká města, výrobní podniky i soukromníci. Řešením, nejčastěji preferovaným městy nebo velkými obchodními domy, je stavba vícepatrových nadzemních nebo podzemních parkovacích domů, poskytujících velké množství parkovacích míst. Výrobní podniky i menší společnosti obvykle volí výrazně levnější variantu přízemního nekrytého parkoviště, které se potřebou větší plochy nehodí do městských center. Všechna představená řešení pro své fungování využívají různé druhy informačních systémů, které umožňují centrální správu dostupných parkovacích kapacit.

Ačkoliv většina nákladů spojených se stavbou parkoviště, je použita na stavbu samotnou, tak nezanedbatelná část peněz je využita na technické vybavení, včetně samotného parkovacího systému. Aplikace, která cílí především na menší podniky a soukromníky s nižším celkovým rozpočtem, nabízí možnost výrazného snížení technických nákladů pomocí přímé podpory různých vjezdových zařízení.

Cílem práce je analyzovat dostupná řešení, navrhnout a implementovat elektronický informační systém umožňující správu dostupných parkovišť a souvisejících služeb. Systém bude poskytovat komplexní webové rozhraní a jednoduché stránky, zajišťující přístup zákazníků. Webové rozhraní bude chráněno před neoprávněným použitím pomocí šifrované komunikace (SSL certifikát) a autentizace pomocí uživatelských účtů. Součástí dostupných stránek bude administrátorská sekce pro základní správu systému.

Informační systém, který je rozdělen do více částečně nezávislých celků, umožňuje vyšší modularitu dostupných služeb. Další rozšiřitelnost jednotlivých částí je zaručena použitými frameworky. Jednotlivé části se realizují na základě principů softwarového inženýrství, včetně podpory automatického testování, sestavení a nasazení.

První část práce analyzuje existující a používaná řešení včetně dostupných modulů a subsystémů. V dalších částech jsou popsány použité technologie a jejich rozdělení podle způsobu fungování na *Webové technologie* a *Aplikační technologie*. Následná kapitola se zabývá strukturou datové vrstvy, vlastnostmi klíčových entit, systémem pro správu oprávnění a návrhem vzhledu webových stránek. V kapitole *Implementace* je popisována struktura systému, tvorba dílčích částí a jejich fungování, včetně implementace statických webových stránek. Závěrečné části práce, *Testování* a *Provoz*, se věnují metodikám pro hledání chyb a zálohování systému.

Kapitola 2

Analýza existujících řešení

Společností vyvíjejících produkty a nástroje řešící problematiku spojenou s dostupností a správou parkování je nepřeberné množství. Většina společností, kromě samotného informačního systému, nabízí i vlastní technologická řešení, která výrazně zjednodušují složitost instalace systému a jeho další údržbu. Součástí nabízených technologií často bývají zařízení zvyšující komfort parkujících zákazníků, např. informační tabule a pokročilé platební systémy. Takovéto systémy jsou nejčastěji nabízeny pouze jako služba, *Black-Box*. Neznalost zdrojového kódu výrazně omezuje další možnosti vlastních úprav systému a jeho případné napojení na zákaznické autorizační servery.

2.1 GREEN Center

Podle [1] je česká společnost *GREEN Center* specializovaným výrobcem a dodavatelem parkovacích systémů působící na evropském trhu. Součástí poskytovaných produktů společnosti jsou i přístupové systémy, turnikety a docházkové systémy. Společnost se soustředí na poskytování komplexních zákaznických služeb, včetně servisu a všech nutných stavebních úprav. Parkovací systém, který je obvykle poskytován pouze jako *Black-box*, cílí především na použití v obchodních centrech.

Nástroj umožňuje podporu dočasného nebo trvalého parkování, rezervování místa však není základní technologií podporováno. Prémiová verze systému umožňuje rezervaci míst a vzdálenou správu parkoviště pomocí mobilní aplikace. Jednotlivé části systému používají ke komunikaci Ethernet rozhraní.

2.2 ParkingBOXX

Společnost *ParkingBOXX* je americkým výrobcem a dodavatelem venkovních a vnitřních parkovacích systémů, podporujících správu parkovacích domů a veřejných parkovišť [2]. Kromě nástrojů pro správu parkovišť dodává a navrhuje technologie potřebné pro samotný provoz těchto zařízení, včetně elektronických platebních terminálů. Informační systém společnosti umožňuje všechny druhy parkování, včetně podpory rezervačního systému. Dočasné parkovaní na veřejných parkovištích je zajištěno pomocí speciálních digitálních parkovacích hodin, které lze dodatečně napojit na další zařízení zákazníka.

Ačkoliv je systém obvykle dodáván v podobě uzavřené služby, nejčastěji běžící v cloudu, *Black-Box*, je možné dodatečné zakoupení speciální licence umožňující vlastní zákaznický vývoj [3].

Tabulka 2.1 Parkovací systémy

Název	Typy vjezdů	Zaměření
GREEN Center	Dočasné, Trvalé	Obchodní centra,
ParkingBOXX	Dočasné, Rezervace, Trvalé	Všeobecné
Loxone	Trvalé	Podnikové

2.3 IoT

Nástroje ze světa *IoT* se obvykle nezaměřují na řešení problémů spojených s parkováním, spíše se věnují zabezpečení celých budov. Součástí těchto systémů tak mohou být nástroje, které mohou monitorovat a zabezpečovat vjezdy do samotného parkoviště. Ačkoliv řešení, pomocí technologie IoT, obvykle nenabízí možnost platebních systémů ani rezervaci míst, tak pro základní potřeby menších podniků mohou tyto systémy plně postačovat.

2.3.1 Loxone

Loxone je rakouská společnost, zaměřující se především na produkty pro chytrou domácnost [4]. Kromě produktů pro domácnosti nabízí také řešení pro podniky, včetně restaurací a hotelů, které často vyžadují privátní nebo zákaznické parkoviště. Nabízený vstupní systém, využívající identifikaci pomocí NFC karet, může být použit nejen pro vchodové dveře ale i parkoviště.

Nástroj pro své fungování používá systém speciálně navržených mini serverů, které obsahují webové rozhraní, umožňující vzdálenou správu a kontrolu aktuálního stavu. Systém je vhodný zejména pro malé podniky, které nemají velké nároky na parkovací systém a současně mohou využít široké nabídky dalších dostupných služeb a nástrojů.

Kapitola 3
Webové technologie

V následující kapitole je postupně probíráno téma zabývající se důležitými technologiemi, které umožňují aplikaci komunikovat přes Ethernet síť. Součástí těchto technologií jsou dále standardy, které popisují formu implementovaného aplikačního rozhraní, dále umožňují přihlášení uživatelů a prezentují dokumentaci samotného systému.

3.1 HTTP

HTTP, celým názvem Hypertext Transfer Protocol, je internetový protokol používaný webovými servery a klienty pro vzájemnou komunikaci. Funguje na základě zasílání požadavků a získání příslušných odpovědí po jejich zpracování na serveru. Protokol HTTP patří do 7. (aplikační) vrstvy ISO-OSI modelu, téměř výhradně používá TCP protokol ze 4. (transportní) vrstvy, ačkoliv existují i adaptace protokolu nad transportním protokolem UDP.

Kromě dvou hlavních verzí HTTP/1.0 z roku 1996 a HTTP/2.0 z roku 2015, existuje verze HTTP/1.1 z roku 1999, která je stále nejpoužívanější verzí ačkoliv postupně ustupuje novější verzi 2.0 vyžadující použití protokolu SSL nebo TLS.

3.1.1 Požadavek

Komunikace pomocí protokolu HTTP začíná odesláním požadavku klientem na cílový webový server. Struktura klientského požadavku, striktně definovaná v konkrétní specifikaci protokolu, ve verzi 1.1, vyžaduje přítomnost povinných částí: metoda, URI a verze [5].

Metoda popisuje akci vyžadovanou po webovém serveru. URI popisuje přesnou cestu ke zdroji a verze specifikuje konkrétní použitý protokol, více v ukázce 3.1. Z [5] plyne možnost přidat do klientského požadavku další údaje, jejichž povinnost se liší na základě specifikované metody z ukázky 3.2.

Důležitou součástí požadavku je hlavička, která obsahuje doplňující povinné i nepovinné údaje pozměňující chování a výslednou odpověď serveru při přijetí daného požadavku. Mezi nejčastěji posílané hlavičky patří: Accept, Accept-Charset, Accept-Language, Authorization. Pokud žádost obsahuje tělo, pak hlavička nutně musí obsahovat další doplňující informace: Content-type a Content-Length.

3.1.2 Odpověď

Webový server po přijetí a zpracování klientského požadavku vytvoří a odešle zpět klientovi odpověď. Ačkoliv se odpověď serveru může lišit v závislosti na použité verzi HTTP protokolu,

Výpis kódu 3.1 GET požadavek

```
GET / HTTP/1.1
Host: localhost
```

Výpis kódu 3.2 POST požadavek

```
POST / HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
Authorization: Basic YWRtaW5AZml0LmN2dXQuY3o6QWRtaW4wMDE=

<!doctype html><html lang="en" class="h-100"></html>
```

první řádek odpovědi je vždy stejný, obsahující následující informace: verze, kód a zpráva kódu [5]. Odpověď může nadále obsahovat hlavičku a tělo.

Kód, který je vždy třímístné číslo a k němu přidružená zpráva, jsou ukazatelem stavu zaslaného dotazu, viz ukázka 3.3. Pouze zprávy nesoucí konkrétní kód obsahují ve svém těle, případně hlavičce, žádanou informaci. Zbývající kódy popisují stav samotné komunikace, chyby nebo další informace přímo související se žádostí.

3.2 HTTP Authentication

Protokol HTTP podporuje autentizaci, včetně možnosti explicitního vyžádání webovým serverem. Server může pomocí hlavičky a těla odpovědi vyžádat od klienta nutná data pro autentizaci, klient následně potřebná data přiloží do hlavičky a žádost opakuje [6].

Existují množství různých autentizačních schémat lišících se především způsoby použití a mírou zabezpečení.

3.2.1 HTTP Basic

Základem fungování autentizačního schématu HTTP Basic je přenos informací pro přihlášení, standardně jméno a heslo v hlavičce žádosti [6]. Jelikož schéma žádným způsobem nezabezpečuje přenášené informace ale pouze zprostředkovává přenos informací nutných pro přihlášení, tak by mělo být vždy používáno pouze ve spojení bezpečným protokolem HTTPS.

Ukázka 3.2 obsahuje platnou autentizační hlavičku, jejíž data byla dle standardu schématu zakódována formátem *Base64*, který umožňuje přenos binárních dat pomocí tisknutelných znaků.

Výpis kódu 3.3 OK odpověď

```
HTTP/1.1 200 OK
Server: nginx/1.18.0
Content-Type: text/html
Content-Length: 3110

<!doctype html><html lang="en" class="h-100"></html>
```

Tabulka 3.1 Obecné kódy odpovědí

Kód	Vlastnost
1xx	Informační
2xx	Úspěch
3xx	Přesměrování
4xx	Chyba klienta
5xx	Chyba serveru

3.3 Single-page application

SPA je webová aplikace, která používá jednu HTML stránku jako základ pro ostatní aplikační stránky, kdy uživatelské akce jsou implementovány pomocí JS, HTML a CSS [7]. Po zpracování HTTP žádosti, klient obdrží odpověď obsahující pouze základovou HTML stránku a nezbytné statické soubory. Aplikační data jsou získána později pomocí žádostí na API.

Výhodou SPA je celková nižší potřeba přenesených dat, kdy nedochází k opakovanému odesílání stejných souborů obsahujících různá data, ale pouze přenosu skutečných dat. Velkého snížení je dosaženo pomocí dynamického načítání obsahu, tedy implementací návrhového stylu *on-demand*.

Při používání SPA je delší doba prvotního načtení, vzniklá odděleným načítáním samotné aplikace a aplikačních dat, zásadním negativem. Moderní webové prohlížeče podporující systém cacheování umožňují dočasné uložení statických souborů, čímž mohou omezit negativní vliv při opakovaném načítání webové stránky. Výrazný vliv na celkovou dobu načítání má kvalita návrhu API, která při vyhledávání přes postupné relace může být časově velice náročné.

3.4 REST

Podle [8] je REST architektonickým stylem pro distribuované prostředí, který sdílí bezstavovost HTTP protokolu a současně obsahuje větší množství vrstev (firewall, proxy) mezi klientem a serverem. Jelikož je REST bezstavový, každá žádost musí obsahovat všechny informace nutné pro její zpracování.

Základem návrhového stylu je práce se vším jako se zdrojem, který lze od serveru získat nebo pozměnit dle zaslaného typu požadavku [8]. Nutnou vlastností webového zdroje je jednoznačná identifikace, nejčastěji pomocí primárního klíče. Technologie REST nabízí stejné rozhraní pro všechny dostupné zdroje serveru nebo aplikace.

3.4.1 Typy požadavků

Metody HTTP požadavků určují druh provedené akce nad konkrétně zadaným zdrojem. Některé akce nemusí být nad daným zdrojem přímo dostupné, ale mohou být i zcela nedostupné, např. akce DELETE nad přístupovými záznamy. Kromě přímé dostupnosti mohou být akce podmíněny požadavkem na platnou autorizaci.

Tabulka 3.2 popisuje typy požadavků a nejčastější serverové odpovědi. Kromě zmíněných typů odpovědí může server odpovědět kódy 4xx a 5xx, kde 5xx značí chyby serveru při zpracování žádosti.

3.5 HATEOAS

Cílem principu HATEOAS je poskytnout standardizované rozšíření API, které kromě samotné práce se zdroji dokáže informovat v těle odpovědi o dalších souvisejících zdrojích [9]. Klient

Tabulka 3.2 REST Typy požadavků

Metoda	Popis	Úspěch	Neúspěch
GET	Získání zdroje nebo seznamu zdrojů	200 OK	404 Not Found
POST	Vytvoření nového zdroje	201 Created	400 Bad Request
PUT	Úprava celého zdroje	204 No Content	404 Not Found
PATCH	Úprava části nebo celého zdroje	204 No Content	404 Not Found
DELETE	Smazání zdroje	204 No Content	404 Not Found

Výpis kódu 3.4 HATEOAS JSON

```
{
    "text": "Parking closed",
    "_links": {
        "delete": {
            "method": "DELETE",
            "href": "http://localhost:8080/api/announcement/10"
        },
        "owner": {
            "href": "http://localhost:8080/api/parking/10"
        },
        "self": {
            "href": "https://localhost:8080/api/announcement/10"
        }
    }
}
```

následně může tyto odkazy použít pro získání dalších zdrojů nebo k jejich manipulaci.

Základním principem fungování je přidání odkazů, *LINKs*, do těla odpovědí, které popisují konkrétní zdroj nebo celou kolekci zdrojů, více v 3.4. Kromě základní podoby odkazů lze využít i komplexnějších metod, které dokáží předat v odpovědi větší množství informací nutných pro použití u ostatních HTTP metod [8].

Odpovědí serveru využívající princip HATEOAS se liší od běžné komunikace, struktura HAL. Tento standard odpovědí podporuje používané formáty REST , XML (application/hal+xml) a JSON (application/hal+json) [10].

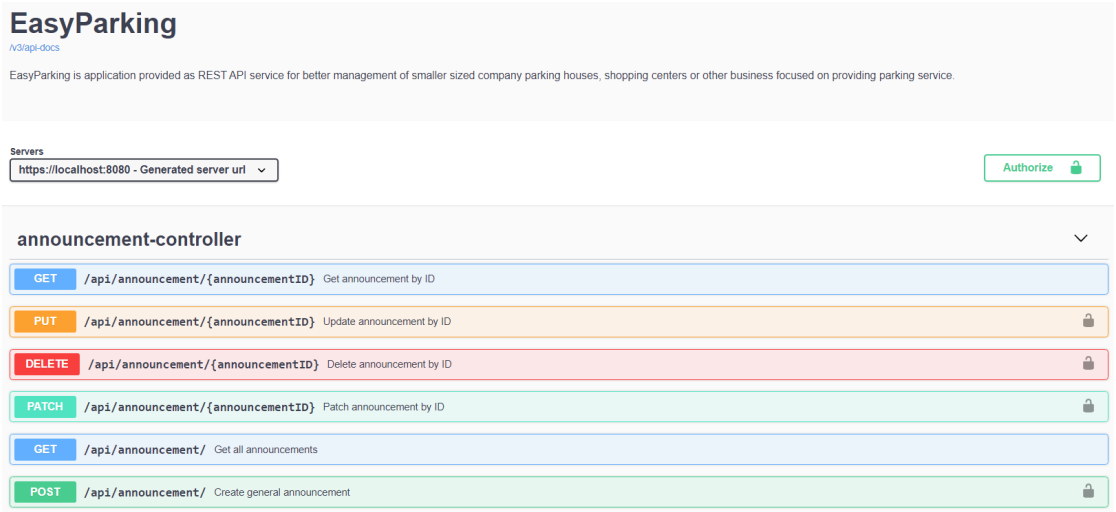
3.6 OpenAPI

Podle [11], OpenAPI definuje pro REST služby standardní informační rozhraní, které umožňuje porozumění dokumentované službě lidem i strojům bez konkrétní znalosti zdrojového kódu nebo sledování síťové aktivity. Hlavním cílem iniciativy (https://www.openapis.org/) je vývoj a správa unifikovaného formátu, přímo nezávislého na programovacích jazycích. Ačkoliv různé jazyky vyžadují mnohdy zcela různé implementace, výsledný standard zůstává pro všechny stejný.

3.6.1 Swagger UI

Swagger je knihovna implementující OpenAPI standard, umožňující vytváření interaktivní dokumentace REST rozhraní, která současně umožňuje udržovat odpovídající stav dokumentace s aktuálním stavem rozhraní [8].

Pro rozšíření základních funkcionalit poskytovaných knihovnou Swagger lze použít Swagger UI, které automaticky generuje uživatelsky přívětivé a intuitivní prostředí [8]. Klíčovým aspek-



Obrázek 3.1 Swagger UI

tem UI knihovny je zpřístupnění REST rozhraní pro manuální dotazování bez použití externích nástrojů nebo dodatečných znalostí.

Interaktivní rozhraní podporuje dotazování na zabezpečená REST rozhraní, kdy poskytuje implementaci různých zabezpečovacích standardů pro přihlašování (HTTP Basic, HTTP Bearer, OAuth2) a jejich intuitivní použití [12]. Kromě podpory dotazování na zabezpečená rozhraní lze i přístup k samotnému UI zabezpečit stejnými standardy.

Základní podporované formáty REST odpovědí pro Swagger UI jsou JSON a XML. Pomocí speciálních rozšíření knihovny lze dosáhnout podpory dalších formátů, např HAL.

3.7 UDP

UDP, celým názvem User Datagram Protocol, je internetový protokol nacházející se na 4. (transportní) vrstvě ISO-OSI modelu. Na rozdíl od protokolu TCP je UDP protokol orientován jako bezspojový. Ke komunikaci dochází pouhým zasíláním samotných zpráv, které netvoří žádné logické spojení a tudíž protokol nezaručuje pořadí ani kontrolu doručení [13].

Z výše zmíněného vyplývá že protokol UDP je výrazně jednodušší a tudíž méně náročný pro aplikace s výrazně limitovanými zdroji. Podle [13] je protokol UDP nejčastěji používán při posílání krátkých nebo často se opakujících zpráv, kdy ztráta části takto poslaných zpráv není kritická, ale důležitější je samotná rychlost. Díky své jednoduchosti se protokol často používá při přenosu zvuku nebo videa, kdy je doba zpoždění kritickým faktorem.

Řetězené spojení se stává nespolehlivým již při zapojení jediné části používající protokol UDP, tedy spojení UDP-TCP je teoreticky nespolehlivé. Další nevýhodou je potřeba mezičlenu, převádějící spojení mezi protokoly UDP a TCP. Naopak výhodou takto propojeného spojení je jednoduchost vstupního bodu.

Výpis kódu 3.5 UDP Server

```
@Autowired
private MessageChannel inputChannel;

@Bean
public UnicastReceivingChannelAdapter input() {
    UnicastReceivingChannelAdapter adapter
        = new UnicastReceivingChannelAdapter(
            parkingProperties.getUdp().getPort()
        );

    adapter.setOutputChannel(inputChannel);
    adapter.setSoTimeout(0);
    adapter.setTaskExecutor(threadPoolTaskExecutor);

    return adapter;
}
```

Kapitola 4

Aplikační technologie

V kapitole *Aplikační technologie* je věnována nástrojům a technologiím, které usnadňují vývoj, použití i nasazení samotného systému a dále zvyšují komfort zákazníků. Mezi nástroje, které podporují vytváření této parkovací aplikace, patří projekt Lombok a technologie objektového mapování. Podpora HTTP metody Patch dále usnadňuje použití implementovaného rozhraní. Zákaznický komfort je rovněž zajištěn použitím technologie strojově čitelných dat neboli čárových kódů.

4.1 Annotation processing

Zpracovat anotace, které představují metadata vložená ve zdrojovém kódu, je možné již při kompilování programu díky technologii *Annotation proccesor*, která slouží k jejich zpracování. Tato technologie, představena s verzí Java 5, umožňuje provádět libovolné akce na základě přítomnosti nebo obsahu zmíněných anotací [14]. Nejčastěji se tato pokročilá metoda používá při automatickém generování pomocných souborů, dokumentace nebo implementace tříd a rozhraní.

4.1.1 Lombok

Projekt Lombok [15] který obsahuje množství anotací a především samotný annotation processor, se používá pro automatické generování opakujících se částí zdrojového kódu. Tím se snižuje riziko implementačních chyb a zjednodušují případné změny v dotčených třídách. Mezi hlavní schopnosti této pokročilé knihovny patří generování konstruktorů, pomocných přístupových metod (`getter`, `setter`) a v neposlední řadě také metod pro instanční ekvivalenci (`equals`, `hashCode`), více v 4.1.

Použití této technologie nevylučuje manuální implementaci klíčových částí vyžadujících komplexnější funkčnost. V takové situaci je existující kód použit a k automatickému generování tak nedochází.

4.2 Metoda PATCH

Metoda PATCH, která je jednou z používaných metod HTTP protokolu, používaná pro aktualizaci zdrojů pomocí REST rozhraní umožňuje aktualizaci pouze části dotčeného zdroje. Na rozdíl od podobné metody PUT, která vždy aktualizuje celý stav zdroje přesně podle obsahu žádosti [8].

Výpis kódu 4.1 Třída obsahující Lombok anotace

```
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
public class Payment {

    @EqualsAndHashCode.Include
    private int id;

    private Bill owner;

    private LocalDateTime payed;

    private PaymentType type;

}
```

Výpis kódu 4.2 JSON Patch

```
PATCH /api/announcement/1 HTTP/1.1
Host: localhost
Content-Type: application/json-patch+json

[
  { "op": "replace", "path": "/title", "value": "New title" }
]
```

Zpracování metody PATCH se může lišit především díky způsobu popsaní požadovaných změn samotného zdroje do těla žádosti. Existují dva základní formáty PATCH žádostí a s tím lišící se struktury jejich obsahů - JSON Patch, JSON Merge Patch.

4.2.1 JSON Patch

Formát JSON Patch (application/json-patch+json) popisuje operace nad zdrojem, krok za krokem, nutné k dosažení požadovaného cílového stavu [16]. Tato struktura podporuje popis více dílčích kroků v těle jedné žádosti.

Při provádění aktualizace zdroje, dochází k ověřování stavu po každé operaci zvlášť. Pokud samotné provedení nebo ověření selže, je provádění další změn zastaveno, zdroj není aktualizován a je vrácen chybový výsledek. Konečná podoba zdroje, která je určena až po provedení všech změn, je následně použita jako nová platná hodnota.

Hlavní výhodou používání tohoto formátu PATCH žádosti je podpora `NULL` hodnot, která je výsledkem složitější a propracovanější struktury. Bohužel s komplikovanější strukturou souvisí i hlavní negativum formátu, netriviální použití rozhraní používající tento formát PATCH žádostí ze strany klienta.

4.2.2 JSON Merge Patch

Formát JSON Merge Patch (application/merge-patch+json) nepopisuje změny zdroje postupnými kroky, ale požadovanou výslednou hodnotou [16]. Validita REST zdroje je ověřována pouze

Výpis kódu 4.3 JSON Merge Patch

```
PATCH /api/announcement/1 HTTP/1.1
Host: localhost
Content-Type: application/merge-patch+json

{
  "title": "New title"
}
```

Výpis kódu 4.4 Mapování pomocí MapStruct

```
@Mapper(componentModel = "spring")
public interface ParkingMapper {

    Parking fromDTO(ParkingDTO target);

    ParkingDTO toDTO(Parking source);

}
```

jednou, po provedení všech žádaných změn, kdy žádost může obsahovat i několik změn současně. Mezi platné hodnoty formátu Merge Patch patří i prázdná žádost, je povolena nulová aktualizace nad zdrojem.

Výsledná podoba dotčeného zdroje je sjednocením existujících hodnot a hodnot obsažených v žádosti, kdy je vždy upřednostněna hodnota obsažená v přijatém požadavku. Jednoduchá tvorba a použití tohoto formátu patří mezi jeho hlavní výhody, kdežto nevýhodou je nemožnost pracovat s `NULL` hodnotou.

4.3 Object mapping

Vícevrstvé aplikace často obsahují větší množství různých modelů reprezentující stejná data pro různé části aplikace. A proto je třeba využít vhodných mechanismů pro efektivní převod mezi nimi [17]. Nejčastěji dochází k převodu mezi modely databázových entit a jejich DTO (Data Transfer Object). Manuální převádění mezi různými objekty zvyšuje riziko implementačních chyb a současně při změnách v modelech je třeba úprav v každém výskytu.

4.3.1 MapStruct

MapStruct je knihovnou implementující princip objektového mapování pomocí anotací a vlastního anotation procesoru, více v sekci 4.1. Ten při sestavování programu automaticky generuje zdrojové kódy s implementaci daných mapovacích rozhraní [18].

Mezi hlavní výhody této knihovny patří možnost vlastní implementace části nebo celého mapovacího rozhraní. Dalším pozitivem knihovny MapStruct je podpora specifických nastavení pouze pro konkrétní třídy a její automaticky generované implementace. Další nespornou výhodou MapStructu je podpora různých DI (Spring, Jakarta EE).

4.3.2 ModelMapper

ModelMapper je knihovnou implementující princip objektového mapování pomocí automatického a inteligentního mapování na základě shodnosti jmen a typů atributů jednotlivých tříd [19]. Této

Výpis kódu 4.5 Mapování pomocí ModelMapper

```
public class ParkingMapper{

    @Autowired
    private ModelMapper modelMapper;

    public ParkingDTO toDTO(Parking parking){
        return modelMapper.map(parking, ParkingDTO.class);
    }

    public ParkingDTO fromDTO(ParkingDTO parkingDTO){
        return modelMapper.map(parkingDTO, Parking.class);
    }
}
```

schopnosti je dosaženo použitím komplexní knihovny, zabudované v základech programovácího jazyku Java, `java.lang.reflection`, která umožňuje dynamickou inspekci vlastního kódu.

Jednoduché použití a jednotné rozhraní, dělají z této knihovny rychlý a snadný způsob pro základní potřeby mapování různých objektů. Netriviální instanční nastavení ji činí nevhodnou při potřebách složitějšího mapování. Podle výsledků srovnání [17] různých mapovacích knihoven ModelMapper, dosahuje výrazně pomalejších výsledků, což je dáno především použitou technologií, a proto se nehodí pro použití ve větších aplikacích.

4.4 MVC

Návrhový vzor MVC, který je často používán při vývoji podnikových aplikací, umožňuje přenesení zodpovědnosti a rozdělení činnosti mezi různé části aplikace [20]. Principem MVC podobající se návrhovému vzoru MVT používáném ve frameworku Django [21], je rozdělení aplikace do třech provázaných logických celků.

- Model — Vrstva, popisující veškeré aplikační zdroje, obsahuje jednoduchou aplikační logiku. Složitější logika, vyžadující provázání většího množství modelů je součástí Controller vrstvy.
- View — Vrstva, zajišťující prezentaci dat uložených v modelové vrstvě, nemodifikuje ani jinak nemění další zdroje aplikace.
- Controller — Vrstva, zodpovídající za zpracování požadavků, vykonává požadované akce s následnou úpravou dat uložených v modelové vrstvě.

Ačkoliv princip MVC rozděluje aplikaci do tří celků, lze celek *View* vynechat a jeho implementaci zajistit externí aplikací, využívající aplikační rozhraní implementované ve vrstvě *Controller*.

4.4.1 Template engine

Jedná se o klíčovou technologii logického celku *View* zajišťující komplexní zobrazení dat uložených v modelové vrstvě aplikace a především zobrazením ve webovém prohlížeči. Technologie pracuje s vlastními standardy šablon, často podobných XML nebo HTML formátům, více v ukázce A.3.

Výsledný stav zpracované šablony, která již obsahuje vložená data z vrstvy *Model*, lze prezentovat přímo nejen uživateli, ale i jako emailové zprávy, více v 4.6.

Výpis kódu 4.6 Templated email

```
@Autowired
private TemplateEngine templateEngine;

public MimeMessage getTemplatedEmail(Map<String, Object> properties){
    Context ctx = new Context();
    ctx.setVariables(properties);

    messageHelper = new MimeMessageHelper(message, true, "UTF-8");
    messageHelper.setText(templateEngine.process(
        (String) properties.get("template"), ctx),
        true
    );

    return messageHelper;
}
```

4.5 React

React je Javascript open-source knihovna, původně vyvinutá společností Facebook, umožňující vývoj komplexních uživatelských rozhraní, která používají rychle se měnící informace [22]. Framework může být dále rozšířen o pomocné knihovny usnadňující vývoj aplikací postavených na technologii SPA, více v sekci 4.5.2. Zvláštností odlišující knihovnu React od ostatních frontend frameworků (AngularJS) je využití technologie virtuálního DOM umožňující efektivní manipulaci s daty i samotným vzhledem webové stránky [23].

4.5.1 Komponenty

Klíčovým konceptem je návrh a tvorba opakovaně využitelných částí, komponent, které sdružují design, data a logiku jednotlivých částí aplikace. Jednoduché bezstavové komponenty jsou nejčastěji popsány pouhým JSX výrazem a komplexního chování je dosaženo jejich opakovaným skládáním. JSX výrazy jsou esenciální komponenty React frameworku popisující dostupné DOM elementy, včetně všech povolených HTML atributů.

Důležitou vlastností komplexních komponent založených na JS třídě `React.Component` je podpora vnitřních stavů, včetně schopnosti reakce na vnější změnu stavu. Při změně stavu komponenta automaticky zajišťuje překreslení zobrazovaných informací a následnou aktualizaci všech vnořených částí. Konkrétní stav, který je nejčastěji popsán objektem, je uložen v proměnné `state`. Aktualizaci stavu není možné provést přímou úpravou uloženého objektu, ale voláním speciální zděděné metody `setState(...)`, která zajistí zahájení potřebné aktualizace.

React systém dále umožňuje informovat komponentu o jejím životním cyklu pomocí speciálních řídících metod. Nejčastěji jsou používány tyto:

- `componentDidMount()` — Metoda informující o potřebě prvotního vykreslení komponenty
- `componentWillUnmount()` — Metoda informující o potřebě smazání, nejčastěji při použití Router DOM, více v sekci 4.5.2

4.5.2 Router DOM

Díky pomocné knihovně *Router DOM*, je možné rozdělit aplikaci do více různých logických částí, stránek, obsahující zcela jiné komponenty [23]. Použití knihovny neodporuje principu SPA,

Obrázek 4.1 2 of 5

Obrázek 4.2 ITF

nedochází k rozdělení aplikace do více různých fyzických stránek. Pouze pomocí DOM elementu dochází ke změně vykreslovaného obsahu a aktualizaci klientské adresace.

Router DOM při správné konfiguraci webového serveru umožňuje použití navigace v historii navštívených stránek, prostřednictvím různých webových adres pro odlišené stránky. Ve skutečnosti adresa popisuje pouze cestu v rámci aplikace na klientské straně. Vlastnosti adresace lze využít i při použití reverzního proxy serveru, více v sekci 6.3.1.1.

4.6 Čárový kód

Čárový kód umožňuje strojově čitelnou vizuální reprezentaci dat různých formátů a délek. Starší jednodušší kódy umí často pracovat pouze s čísly, která jsou současně značně limitována maximální délkou. Novější formáty umožňují kódování rozličných formátů dat. Nejmodernější umožňují kódování i skutečných binárních dat pomocí 2D rozměrných kódů (QR code, Data Matrix).

4.6.1 Code 128

Ačkoliv Code 128 patří mezi starší formáty kódování dat, byl vyvinut již v roce 1981, přesto umožňuje vizuální reprezentaci široké škály dat. Této schopnosti je dosaženo souběžnou existencí 3 podobných specifikací, které podporují různé, často neslučitelné, formáty dat [24].

- 128A — Kódování znaků A-Z, 0-9 a kontrolních ASCII znaků.
- 128B — Kódování znaků A-Z, a-z, 0-9 a kontrolních ASCII znaků.
- 128C — Kódování pouze matematických znaků 0-9.

Výhodou formátu 128 je podpora velké znakové sady a současně teoreticky neomezená maximální délka, kdy samotná délka je omezená pouze poměry stran skutečného kódu. Další výhodou je podpora kontrolního součtu pro ověření správnosti čtených dat. Nevýhodou formátu je vysoký poměr délky kódu k množství uložených dat, avšak výběrem správného podtypu lze dosáhnout výrazného zlepšení.

4.6.2 ITF

Formát ITF je vylepšenou verzí původního jednoduchého kódu *2 of 5*, nepodporující kontrolní součet a mající vysoký poměr délky ke kódovaným datům. Vylepšená verze ITF podporuje volitelný kontrolní součet a díky změně kódovacího algoritmu výrazně snižuje informační poměr. Této vlastnosti bylo dosaženo současným reprezentováním dat bílými mezerami a černými sloupci [24]. Za předpokladu dodržení výrazné odlišnosti mezi barvami sloupců a mezer povoluje standard ITF i jiné barevné kombinace.

Obrázek 4.3 Code128B

4.7 Automatické sestavovací nástroje

Sestavení aplikací využívajících větší množství dalších knihoven nebo závislostí, může být často obtížné a náchylné na chyby. Pro správné a rychlé sestavení lze proto použít specializované nástroje, které umožňují automatizaci celého procesu a současně nabízí podporu v dalších krocích při vývoji aplikací.

Základním konceptem těchto nástrojů je standardizace procesu výroby softwaru a správa potřebných závislostí. Pokročilejší nástroje nabízejí podporu různých verzí i jednotného úložiště pro často používané knihovny.

Dalším klíčovým aspektem zmíněných nástrojů je podpora a automatizace testů, včetně jejich následného zapojení do dalších kroků výrobního procesu. Nástroje mohou při správné konfiguraci pomoci s automatizací složitějších testů vyžadujících spolupráci více systémů.

4.7.1 Maven

Software Maven patří mezi komplexní nástroje umožňující automatické sestavení aplikace podporující především programovací jazyky Java, Ruby nebo Scala [25]. Podle [26] je klíčem pro funkčnost nástroje Maven soubor `pom.xml` ve formátu XML obsahující konfiguraci nutnou nejen pro úspěšné sestavení projektu. Díky technologii lokálních nebo vzdálených repozitářů, nástroj umožňuje snadné provázání menších projektů mezi sebou. Současně automaticky aktualizuje a spravuje knihovny nutné pro sestavení aplikace.

Akce nástroje Maven jsou logicky členěny do množství dílčích kroků vyžadujících bezchybné provedení všech předešlých. Mezi nejdůležitější patří:

- compile — Kompilování zdrojového kódu.
- test — Provedení a vyhodnocení automatických testů.
- package — Vytvoření souborů obsahujících zkompilovaný kód.
- deploy — Nahrání na vzdálený repozitář.

Kapitola 5
Návrh

5.1 Databázový model

O strukturu uložených aplikačních dat se stará 3. vrstva návrhového vzoru MVC, *Model*. Cílem vrstvy je jednotný popis struktur skutečných dat a vazeb mezi jednotlivými zdroji, včetně jejich násobností. Jednotné databázové rozhraní, které podporuje komunikaci přes sít, je typicky použito pro přístup ke skutečným datům nebo jejich následné modifikace.

Striktní rozdělení mezi logickou reprezentací dat a fyzickým uložením umožňující použití různých implementací pro ukládání dat, podporuje aplikační nezávislost na použité implementaci. Další členění logické reprezentace dat umožňuje nezávislé zapojení většího množství různých databázových implementací, včetně podpory duplikace, synchronizace a zálohy dat více databází.

Násobnost vazeb mezi zdroji, modelovými entitami, je vlastností, která má zásadní roli na výsledné logické i fyzické struktuře dat. Různé násobnosti vazeb se promítají do způsobu práce v ostatních vrstvách modelu *MVC*.

Aplikační data jsou rozdělena do těchto celků:

- Auth — Uživatel, uživatelská práva
- Audit — Záznamy přístupů přes webové API
- Parkování — Parkoviště, parkovací místa, rezervace
- Platba — Platby, účty

Ve zdrojích práce je přiložen úplný databázový model vytvořený pomocí nástroje IntelliJ IDEA 2020.2.4, graficky popisující strukturu modelové vrstvy.

5.1.1 Auth

Logický celek *Auth* obsahuje entity nutné pro základní fungování celé aplikace. Kromě popisu existujících účtů také obsahuje seznam bezkontextových práv, více v 5.3, spojených s existujícími účty a autorizačními klíčí nutných pro jejich případnou modifikaci.

5.1.1.1 User

Základem logického celku *Auth* je entita *User*, popisující všechny uživatelské účty, které se mohou v aplikaci autorizovat, včetně systémových a administrátorských účtů.

Tabulka 5.1 User entity

Aspekt	Popis
email	Unikátní email
password	Hash hesla
firstName	První jméno
lastName	Poslední jméno
enabled	Aktivace účtu
locked	Uzamčení účtu

Tabulka 5.2 Permission entity

Aspekt	Popis
type	Druh oprávnění
value	Skutečná hodnota oprávnění

Aplikace používá pro unikátní identifikaci účtů emailovou adresu, která je nutnou podmínkou pro jeho založení. Další nutnou podmínkou při založení účtu je existence hesla splňující následující podmínky:

- Délka — Rozmezí mezi 8 a 30 znaky
- Písmena — Obsahuje velká a malá písmena ASCII znakové sady
- Čísla — Obsahuje minimálně 1 číslo
- Mezery — Neobsahuje žádné bílé znaky

Heslo spojené s účtem není ukládáno v přímo čitelné podobně, ale je reprezentováno ve speciální formě HASH, která je uložena při vytvoření nového účtu nebo aktualizaci stávajícího hesla. Tato forma, ze které nelze běžnými prostředky zjistit původní hesla, zajišťuje bezpečné uložení těchto hesel pro případ neoprávněného přístupu do systému nebo samotné databáze. Při přihlašování dochází pouze k ověřování shodnosti uloženého HASH řetězce s nově vytvořeným, více v ukázce A.2.

Entita *User*, kromě základních informací, hesla a jména, obsahuje a popisuje i hodnoty dalších vlastností, viz tabulka 5.1.

5.1.1.2 Permission

Povolení přístupu k aplikačním zdrojům pomocí účtů je podmíněno úspěšnou autorizací a vlastnictvím potřebných práv. Aplikace používá pro přístup pouze práva bezkontextová, kdy potřeba kontextových práv je emulována pomocí ACL emulátoru 5.2, více v sekci 5.3.

5.1.1.3 Token

Údaje přímo související s autorizací pomocí uživatelského účtu, především jméno a heslo, podléhají přísnějším standardům v zabezpečení, včetně ochrany proti nežádoucím nebo neautorizovaným změnám. Nejčastěji je této vlastnosti dosaženo pomocí podmínění změn speciálním způsobem autorizace, například pomocí emailu nebo telefonu.

Entita *Token* obsahuje data všech vydaných autorizačních klíčů, platných pro změnu uživatelských dat. Důležitou součástí zabezpečení je omezení platnosti a automatické zneplatnění při prvním použitím. Automatické generování klíčů je zásadním aspekt zabezpečení, více v sekci 5.2.

Tabulka 5.3 Token entity

Aspekt	Popis
token	Autorizační klíč
status	Stav použití
creationDate	Datum a čas vytvoření
expirationDate	Datum a čas propadnutí
confirmationDate	Datum a čas potvrzení

Výpis kódu 5.1 Audit Entity pro Spring Actuator

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;

private String principal;
private LocalDateTime date;
private String type;

@ElementCollection(fetch = FetchType.EAGER)
@CollectionTable
private Map<String, String> data = new HashMap<>();
```

5.1.2 Audit

Záznamy provedených akcí jsou dalším prvkem zabezpečení používaných především při opravě vzniklých chyb nebo narušení systému, kdy je třeba zajistit uživatelský účet autorizující dané změny.

Sledování provedených změn lze uskutečnit jednoduchým monitorováním přijatých HTTP žádostí a jejich vyhodnocením nebo komplexním sledováním se záznamem změn všech modelových entit.

5.1.2.1 Spring Actuator

Podle [27] patří *Spring Actuator* mezi nástroje podporující monitorování a řízení produkčních verzí aplikací. Klíčem fungování je vytvoření webových rozhraní informujících o stavu a problémech běžící aplikace. Kromě webových rozhraní lze výstup monitorovacích systémů použít pro vytváření záznamů v podobě entit nebo zápisu do aplikačního logu.

Aplikace využívá podpůrnou knihovnu pro sledování HTTP žádostí o autorizaci a její výsledný stav včetně dalších pomocných dat (IP adresa) zapisuje do databáze i aplikačního logu.

Záznamy jsou uloženy ve dvou propojených tabulkách, kdy tabulka `audit` obsahuje primární informace o provedené akci (záznam—audit) a pomocná tabulka `audit_data` doplňující informace. Pomocná tabulka může obsahovat více záznamů přímo souvisejících s jedním auditem z hlavní tabulky.

5.1.2.2 Hibernate Envers

Nástroj *Hibernate Envers* umožňuje sledovat změny prováděné na entitách nebo pouze určitých atributech a následně tyto změny, včetně přihlášeného uživatele a časové známky, ukládat do speciálních tabulek. Každá změna je reprezentována jedním záznamem do příslušné tabulky.

Systém funguje na bázi speciální tabulky pro každou entitu podléhající monitorování, nejčastěji pojmenovanou příponou AUD. Tabulky mohou být nastaveny pro fungování bez DB pravidel, tj. SQL constraint.

Tabulka 5.4 Hours entity

Aspekt	Popis
day	Den v týdnu
type	Druh příjezdu/zákazníka
price	Cena za účtovací období
perUnit	Způsob účtování
since	Počáteční čas platnosti
till	Konečný čas platnosti

Technologie auditovaných entit nebyla použita v samotné aplikaci pro složité uživatelské použití a zbytečnou robustnost, plynoucí z potřeby speciálních záznamových tabulek.

5.1.3 Parkování

Logický celek *Parkování* sjednocuje entity potřebné k fungování aplikační logiky pro poskytování základních parkovacích služeb. Obsahuje jednoduché modely popisující dostupná parkoviště, parkovací místa, ceník a komplexní struktury zajišťující fungování rezervací a vjezdových záznamů.

5.1.3.1 Otevírací doba

Otevírací doba a s tím související zpoplatnění parkování patří mezi základní rozhodovací faktory zákazníků, zda parkoviště využít či nikoliv. Aplikace umožňuje zpoplatnění parkování pouze během otevírací doby, která není časově limitována a může být i celodenní. Účtování probíhá automaticky po hodinách nebo minutách podle typu zpoplatnění. Jiné typy nejsou podporovány.

Tabulka 5.4 popisuje způsob rozdělení kombinující otevírací dobu, den v týdnu a druh příjezdu. Tato vlastnost umožňuje otevření parkovišť pouze předem dané skupině zákazníků. Ostatním bude příjezd zamítnut.

5.1.3.2 Vjezd

Entita *Vjezd* obsahuje údaje o všech provedených nebo plánovaných vjezdech, kdy při vytvoření rezervace dochází automaticky k naplánování daného vjezdu. Entita si udržuje informace (SPZ) o vjezdu vozidel nacházejících se v danou dobu na parkovišti. Tyto informace lze použít pro bezkontaktní výjezd z parkoviště.

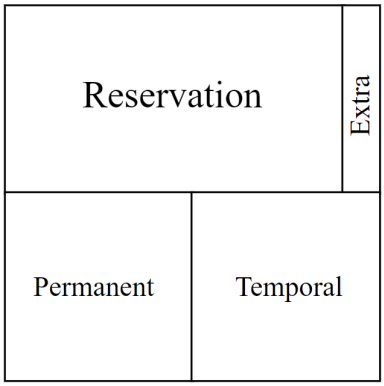
Mezi další údaje nutné pro fungování vjezdového systému patří unikátně generovaný klíč, více v 5.2. Systém nezávisle na autorizaci odjezdů pomocí tohoto klíče umožňuje výjezd na základě shodnosti SPZ. Aplikace podporuje více různých formátů SPZ. Informace o značce je limitována pouze datovým formátem použitým pro její reprezentaci v databázi (textový řetězec).

5.1.3.3 Rezervace

Entita *Rezervace* je komplexní datová struktura, která zaznamenává existující rezervace s podporou lineárního vyhledávání v čase pro konkrétní parkovací místa. Tento systém umožňuje, kromě uložení časových informací plánovaného příjezdu a odjezdu, také vzájemné propojení uživatele s konkrétním místem a platným záznamem vjezdu.

Existují dva různé druhy rezervace lišící se způsobem použití:

- Klasická rezervace — Každý přihlášený uživatel s vhodným oprávněním má právo na její vytvoření i smazání, kdy rezervace je omezena délkou trvání a maximální kapacitou parkoviště.



Obrázek 5.1 Distribuce míst

- Trvalá rezervace — Pouze administrátor, tedy uživatel s vyšším oprávněním, má právo na její vytvoření. Tato možnost je dále limitována maximální kapacitou parkoviště. Placení permanentkou není zahrnuto v platebním systému.

5.1.3.4 Parkovací místo

Entita, *Parkovací místo*, popisuje všechna dostupná parkovací místa, včetně jejich další vlastností. Přímé atributy parkovacího místa jsou: jméno, typ, stav obsazení a datum poslední změny. Informace o volných místech může být použita ve vizuálním směrovacím systému, který usnadňuje vyhledávání volného parkovacího místa.

Parkovací místa lze po vytvoření smazat nebo libovolně upravit, ale daný druh místa nelze dále měnit. Pro dosažení optimálního fungování celého systému se doporučuje vizuální oddělení míst a dodržení optimálních poměrů viz obrázek 5.1.

- Temporal — Určeno pro dočasné vjezdy bez použití uživatelského účtu. Systém automaticky hlídá stav zaplnění podle platných vjezdových oprávnění.
- Permanent — Určeno pouze pro rezidenty. Celkový počet platných rezidentů nemůže přesáhnout počet míst tohoto typu.
- Reservation — Určeno pouze pro rezervace. Místo nemůže být součástí dvou různých rezervací na stejný čas.
- Extra — Slouží jako trvalá rezerva pro pokrytí rezervovaných zákazníků, kteří ještě nemají uvolněné místo.

5.1.4 Platba

Logický celek *Platba* sjednocuje entity potřebné k vytváření účtů za parkování a provádění plateb v hotovosti nebo kartou.

5.1.4.1 Účet

Entitu *Účet* je možné vytvořit pouze pomocí platebních terminálů, které využívají speciální účty s vyšším oprávněním umožňující následné zaplacení. Pokud účet nebyl manuálně vytvořen vzniká automaticky při odjezdu. Platnost účtu je časově omezená a pokud nedojde k zaplacení v daném limitu je třeba vytvořit účet nový. Odjezd z parkoviště je povolen pouze s platným zaplaceným účtem. Při promeškání doby odjezdu je třeba se zkontaktovat s obsluhu parkoviště a postupovat dle jejich pokynů.

5.2 Generování klíčů

Účelem automaticky generovaných klíčů je identifikace aplikačních zdrojů a spojených přístupových práv plynoucích ze znalosti těchto klíčů. Ze způsobu použití často plyne potřeba zcela jednoznačné identifikace, které je dosaženo unikátním generování nebo jinými pomocnými mechanismy.

Samotné vytváření klíčů může probíhat podle předem jasného algoritmu nebo zcela náhodně, kdy lze tyto klíče použít i pro přístup k omezeným zdrojům. Znalost generovaných klíčů může značně ohrozit jejich bezpečnost, především znalost všech platných a použitých klíčů.

Ačkoliv samotný formát není ničím limitován, pro jednoduchou přenositelnost a použití je vhodné použít klíče složené pouze z určitých znaků, nejlépe ASCII. Klíče jsou generovány již s omezením konkrétních znaků nebo v binární podobě, která je pomocí kódování *base64* mohou být převedeny do formátu obsahující pouze ASCII znaky [28].

5.3 Oprávnění

Práce s aplikačními zdroji pomocí webového REST rozhraní může vyžadovat úspěšné přihlášení uživatele. Pokud autorizační podmínky pro práci se zdrojem nejsou splněny, server automaticky odmítá provedení žádosti a odpovídá *401 Unauthorized*.

Podpora kontextu je základní vlastnost oprávnění, podle kterého se liší způsob vytváření a dalšího použití.

5.3.1 Bezkontextová

Bezkontextová práva popisují povolené akce uživatele nad daným typem zdroje, bez dalších znalostí konkrétní instance. Z této vlastnosti plyne, že bezkontextová práva umožňují provedení konkrétní akce nad všemi zdroji stejného typu.

Bezkontextová oprávnění jsou dále rozdělena podle typu na dvě skupiny:

- Role — Popisují sadu oprávnění, obvykle napříč aplikačními zdroji i dostupnými metodami. Snižují nutnost duplikace oprávnění a zjednodušují počáteční nastavení systému.
- Autority — Popisují oprávnění pouze jedné akce nad konkrétním zdrojem.

Hlavní výhodou při použití bezkontextového systému oprávnění je jednoduchost implementace a způsob použití. Jelikož různá práva (role i autority) mohou nezávisle na sobě popisovat stejné akce, je důležitou vlastností pro zachování jednoduchosti jednotný jmenný systém oprávnění, více v tabulce 5.5.

Zásadní nevýhodou plynoucí z vlastností bezkontextového systému je nemožnost omezení přístupu uživatelů k určitým zdrojům. Řešením je souběžné použití bezkontextového a kontextového systému oprávnění.

5.3.2 Kontextová

Kontextová práva popisují povolené akce přímo mezi uživatelem a konkrétní instancí zdroje, čímž eliminují nevýhody jednoduchých bezkontextových systémů.

Mezi nevýhody kontextových systémů patří složitá správa, komplikované počáteční nastavení a vyšší aplikační náročnost plynoucí z potřeby uložení každého dílčího oprávnění.

Tabulka 5.5 Naming policy

Metoda	Oprávnění	Popis
GET	<x>.getAll	Zisk všech zdrojů daného typu
GET	<x>.getByID	Zisk zdroje pomocí přímého ID
GET	<x>.getBy<y>ID	Zisk zdroje pomocí ID
PUT	<x>.updateBy<y>ID	Aktualizace celého zdroje pomocí ID
POST	<x>.create	Vytvoření nového zdroje
POST	<x>.createBy<y>ID	Vytvoření nového zdroje pomocí ID
PATCH	<x>.patchBy<y>ID	Aktualizace části zdroje pomocí ID
DELETE	<x>.deleteBy<y>ID	Smazání zdroje pomocí ID

Výpis kódu 5.2 Rozhraní emulátoru ACL

```
public interface AccessService<E, K> {

    boolean checkByKey(K primaryKey, Authentication authentication);

    boolean checkByEntity(E entity, Authentication authentication);

}
```

5.3.2.1 Spring ACL

Podle [29] je Spring ACL komponenta, umožňující definovat oprávnění konkrétních uživatelů. Konceptem fungování komponenty je popis všech dostupných zdrojů pomocí unikátních identifikátorů a jejich následné propojení s obdobně popsanými vlastníky. Vlastníci mohou být konkrétními uživateli nebo jsou zastoupeny rolemi.

Klíčem fungování je existence čtyř tabulek, které popisují vztahy mezi entitami:

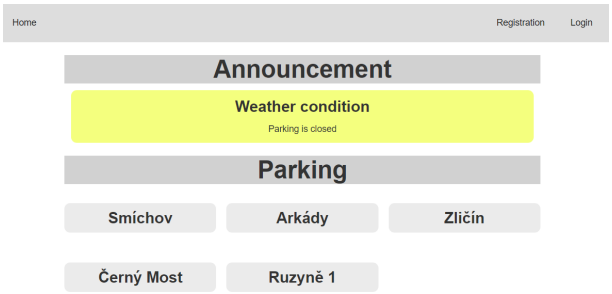
- ACL_CLASS — Typu entity popsané v tabulce ACL_OBJECT_IDENTITY
- ACL_SID — Univerzální popis autority nebo uživatele
- ACL_OBJECT_IDENTITY — Univerzální popis aplikační entity
- ACL_ENTRY — Spojovací tabulka, která popisuje přiřazení konkrétních práv

Mezi základní dostupná práva patří: READ, WRITE, CREATE, DELETE a ADMINISTRATION.

5.3.2.2 ACL Emulátor

Aplikace pro své fungování používá současně bezkontextový i kontextový systém. Bezkontextový systém oprávnění definuje práva podle tabulky, viz 5.5, a také tři pomocné role s ekvivalencí práv popsaných v sekci 6.1.2.

- USER — Uživatelské oprávnění umožňující základní přístup k hlavním zdrojům.
- ADMIN — Administrátorské oprávnění umožňující úplný přístup ke zdrojům.
- SYSTEM — Systémové oprávnění umožňující přístup technického vybavení parkoviště do aplikace.



Obrázek 5.2 Administrátorská sekce

Bezkontextový systém umožňuje administrátorům obejít emulovaný kontextový systém a získat plný přístup ke všem aplikačním zdrojům.

Tento systém, který je emulován rozhraním v 5.2, umožňuje přístup běžným uživatelům disponujících danými zdroji, neboť přímá úprava daných zdrojů může vyžadovat bezkontextové administrátorské oprávnění.

5.4 API

Aplikace se ovládá pomocí webového REST rozhraní a není rozděleno na sekce podle požadovaných oprávnění. Ta jsou zdokumentována v Swagger UI dokumentaci, více v 3.6.1.

Část rozhraní je veřejná, nevyžaduje žádnou doplňující autentizaci, ale zbytek potřebuje přihlášení pomocí metody HTTP Basic.

5.4.1 Autentizace

Základním vstupním bodem do neveřejné částí aplikace je rozhraní `GET <API>/api/user/self`, které zprostředkovává přihlášení do aplikace. Tím umožňuje získání informací o uživatelském účtu použitém při přihlášení.

5.5 UI

Kromě webového REST rozhraní přímo ovládající aplikaci, je možné k jednoduchému zákaznickému ovládání využít uživatelsky příjemného prostředí, dostupného pomocí moderního webového prohlížeče.

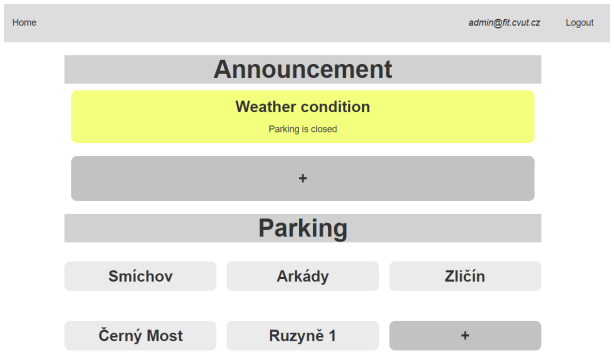
Pro snadné použití na mobilních zařízeních nebo zařízeních s nestandardními rozměry obrazovky, podporují všechny dostupné stránky responzivní design (qHD, HD, Full HD, QHD).

Webová stránka umožňuje bezpečné přihlášení pomocí REST API, včetně rozdělení zobrazeného obsahu na základě uživatelských práv. Dále podporuje dynamické načítání obsahu a v případě neúspěchu informuje uživatele.

5.5.1 Veřejná sekce

Veřejná sekce je nejdůležitější částí webové stránky. Slouží k prezentaci aktuálního stavu a dalších důležitých informací ohledně dostupných parkovišť a volných míst pro potenciální zákazníky.

Důraz při návrhu veřejné sekce webových stránek je kladen na jednoduchost a přehlednost, viz obrázek 5.2.



Obrázek 5.3 Veřejná sekce

5.5.2 Uživatelská sekce

Uživatelská sekce umožňuje provádět změny informací, které souvisí s účtem použitým při přihlášení. Tyto změny podléhají autentizaci pomocí odkazu doručeného emailem, *2FA*.

Sekce tím vytváří možnost rezervace pro specifické parkovací místo. Součástí uživatelské sekce je rovněž stránka s přehledem všech rezervací, která zobrazuje i stav konkrétní rezervace.

Uživatelská sekce je z části zakomponována do veřejné části, zbytek je umístěn na speciálních stránkách, `<UI>/auth/reservation` a `<UI>/auth/account`.

5.5.3 Administrátorská sekce

Pro zobrazení a přístup do administrátorské sekce stránek je vyžadováno přihlášení pomocí účtu se vyšším oprávněním, více v sekci 5.3 a 6.1.2.

Vzhled administrátorské sekce je založen na bázi sekce uživatelské podle obrázku 5.2 a navíc obsahuje ovládací prvky pro přidání nebo změny základních zdrojů aplikace, viz obrázek 5.3.

Sekce nepodporuje přímé ovládání všech zdrojů aplikace. Změny vyžadující hlubší znalost systému je nutné provádět pomocí přímých žádostí na REST API.

Kapitola 6

Implementace

Při tvorbě samotné aplikace jsou použity dvě různá vývojová prostředí, která nabízejí specifickou podporu pro použité technologie. Výrazným způsobem zjednodušují další tvorbu projektu. Každá z jeho částí je vyvíjena jako samostatný projekt.

Pro *Backend* části projektu psané v programovací jazyce Java (Core API, SpotProxy a GateProxy) je využito prostředí IntelliJ IDEA, které nabízí širokou podporu použitého frameworku Spring a Spring Boot. Zároveň umožňuje snadné použití nástroje Maven pro automatické sestavování projektu.

Pro *Frontend* část projektu je použit jednodušší nástroj Visual Studio Code, který se zaměřuje na podporu při práci s HTML a JS.

6.1 Core API

Projekt Core API, který nezávisí na funkčnosti ostatních vyvíjených částí aplikace, implementuje REST rozhraní a definuje entity pro modelovou vrstvu, je vytvářen jako první.

6.1.1 Konfigurace

Důležitou vlastností je možnost pokročilé konfigurace projektu dle požadavků a přání zákazníka. Konfigurační systém, který je přímo spravován použitou knihovnou Spring, umožňuje načtení projektového nastavení z různých typů konfiguračních souborů (XML, YAML, properties) nebo pomocí proměnných prostředí.

Seznam a detailní popis všech konfigurovaných vlastností projektu se nachází v dokumentační a instalační příručce, víc v souboru `./README.md`.

6.1.1.1 Aplikační nastavení

Mezi nejdůležitější konfigurace, které zásadně ovlivňují fungování aplikace, patří:

- `application.pm.user.registration-enabled` — Umožnění registrace novým uživatelům
- `application.pm.user.promotion-enabled` — Prvotní povýšení uživatele na administrátora
- `application.pm.email.web` — Veřejná adresa webové stránky, více v sekci 6.3

Obrázek 6.1 Swagger UI Security expression

6.1.1.2 Spring nastavení

Následující vlastnosti jsou nutné pro úspěšné spuštění frameworku Spring Boot a všech jeho částí:

- `spring.datasource.*` — Databázové nastavení
- `spring.mail.*` — Definice mailového serveru pro odchozí poštu
- `spring.ssl.*` — Aktivace šifrovaného spojení pomocí SSL certifikátu

6.1.2 Authorization expression

Pro snadné použití Swagger dokumentace v dalším vývoji jsou zveřejněny autentizační a autorizační požadavky všech implementovaných rozhraní, které popisují nutnost přihlášení a případná další oprávnění, viz obrázek 6.1. Bezpečnostní požadavky jsou popsány ve formě SSE umožňující použití ACL emulátoru, více v sekci 5.3.2.2.

6.1.3 Rezervace

Pro regulaci dopravních situací a obsazenosti parkovišť je použit systém umožňující rezervaci konkrétních míst pomocí aplikačního rozhraní nebo webové stránky. Funkčnost rezervačního systému je podmíněna existencí zvláštních parkovacích míst, *Reservation* a *Extra*, více v sekci 5.1.3.4.

6.1.3.1 Vytvoření rezervace

Vytvoření rezervace je komplexní proces, který vyžaduje úspěšné přihlášení standardního uživatele. Provádí kontrolu požadované rezervace, identifikuje parkovacího místo, následně vyhledá všechny ostatní platné rezervace. Na základě těchto informací porovná čas vyhledaných záznamů a současně vyhodnotí dostupnost daného místa, viz diagram na obrázku A.1.

Možnost tvorby rezervací je omezena pokročilou konfigurací systému, která umožňuje nastavení celkové doby platnosti rezervací, viz `application.pm.reservation.*`. Cena za parkování je za použití rezervace počítána podle jiného ceníku než dočasný vjezd, ačkoliv výsledná suma je vždy závislá pouze na skutečné době strávené v parkovacím zařízení.

6.1.4 Vjezdová brána

Vjezd a výjezd z parkoviště je realizován pomocí speciální služby Gate proxy, která zprostředkovává komunikaci mezi UDP klientem a HTTPS rozhraním vyžadujícím vyšší oprávnění, více v sekci 6.2.3.

6.1.4.1 Vjezd

Informační systém indikuje příjezd vozidla na parkoviště pomocí přijetí žádosti o vjezdové oprávnění. Na základě této žádosti je vytvořeno nové nebo aktualizováno stávající. Při nedostatečné kapacitě nebo neplatné otevírací době je příjezd automaticky zamítnut.

Žádost obsahuje informaci identifikující konkrétní parkoviště, která je poskytnuta vjezdovým zařízením. Další součást žádosti je informace umožňující identifikaci již vytvořeného oprávnění.

- Klíč — Vjezdový klíč, který unikátně identifikuje vjezdové oprávnění. Více v sekci 6.1.5.2.
- SPZ — Značka, která umožňuje snadnou identifikaci při odjezdu.

Přesný způsob fungování je znázorněn pomocí BPMN diagramu na obrázku A.2.

6.1.4.2 Výjezd

Odjezd z parkoviště, který není limitován otevírací dobou, je indikován žádostí na stejné rozhraní umožňující příjezd automobilu. Současně dochází ke kontrole stavu zaplacení a případně k vytvoření účtu nového. Žádost o výjezd s nezaplaceným účtem je odmítnuta. Parkovací permanentky jsou automaticky vynechány z platebního systému.

Obsah žádosti se liší v závislosti na povolení autentizace vjezdového oprávnění pomocí poznávací značky automobilu, konfigurace `application.pm.entry.plate-authentication`, více na obrázku A.3.

6.1.5 Generování klíčů

6.1.5.1 Uživatelské klíče

Uživatelské klíče jsou použity pro unikátní identifikaci všech žádostí o změnu účtů i včetně registrací, více v sekci 5.2.

Klíče, které jsou automaticky generovány pomocí Java implementace standardu UUID [30] jsou následně reprezentovány pomocí ASCII řetězce o fixní délce 32 znaků. Klíčovou vlastností použitého standardu je velice nízká pravděpodobnost vzniku stejného klíče, tzv. kolize, zabraňující nežádoucím modifikacím účtů.

Stejný typ klíče je použit pro povýšení běžného účtu, `application.pm.user.promotion-token`, více v 8.1

6.1.5.2 Parkovací klíče

Požadavku na unikátní identifikaci vjezdových oprávnění je dosaženo pomocí speciálně generovaných parkovacích klíčů. Jelikož je generování klíčů v různých parkovištích na sobě nezávislé, je nutné pro konkrétní určení vjezdu přidat identifikátor parkoviště.

Formát generovaných klíčů je neměnný s 10 numerickými znaky, které umožňují reprezentaci pomocí jednoduchých čárových kódů, více v sekci 4.6. Klíč se skládá ze tří různých částí, viz ukázka 6.1.

- `<\%y>`— Poslední číslo roku
- `<\%j>`— Pořadí dne v konkrétním roce
- `<[0-9]{6}>` — Náhodná kombinace čísel obsahující vždy 6 znaků

Celkový počet denních klíčů je omezen počtem náhodných kombinací na milion klíčů pro každé parkoviště. Problémy, které plynou při opětovném generování z relativně malého počtu všech dostupných klíčů, jsou řešeny opětovnou tvorbou.

Výpis kódu 6.1 Formát parkovacího klíče

```
<%y><%j><[0-9]{6}>
1   001 000000
```

Výpis kódu 6.2 Formát žádosti Spot Proxy

```
<spotID>:<true|false>
4352    :true
```

6.2 Proxy

Služba *Proxy* je specializovanou částí informačního systému, která zprostředkovává komunikaci mezi zařízením provozovatele a zabezpečeným HTTPS rozhraním, jež vyžaduje vyšší administrátorské oprávnění. Služba vytváří veřejně nebo interně dostupné rozhraní jiného protokolu, nejčastěji *TCP* a *UDP*.

6.2.1 Konfigurace

Přehled nejdůležitějších konfigurací:

- `application.ep.web.address`— Interní/veřejná adresa webového rozhraní
- `application.ep.web.username`— Uživatelské jméno použité v HTTPS žádosti
- `application.ep.ssl.trustStore` — Podpora šifrovaného spojení pomocí SSL

Podle [31] interní komunikace mezi jednotlivými Docker službami je běžně adresována názvem služby, z čehož plyne nutnost použít `org.apache.http.conn.ssl.NoopHostnameVerifier`, která je způsobená neschopností plné autentizace doménového jména.

6.2.2 Spot proxy

Službu *Spot proxy*, která je založena na automatické aktualizaci parkovacích míst, lze použít pro provoz systému zobrazující skutečnou dostupnost míst.

Parkovací čidla při změně stavu vysílají UDP žádost na proxy službu obsahující identifikátor a aktuální stav místa, formát v ukázce 6.2. služba naváže HTTPS spojení s aplikačním rozhraním a aktualizuje parkovací místo, avšak neinformuje o výsledku žádosti.

6.2.3 Gate proxy

Služba *Gate proxy*, která je nutnou součástí pro fungování informačního systému, zprostředkovává vytváření a aktualizaci vjezdových oprávnění, více v sekci 6.1.4.

Žádosti, které jsou odesílány externím zařízením provozovatele, obsahují informace ve formátu viz 6.3. Služba po zpracování a vyhodnocení žádosti odpovídá formátem 6.4, kdy veškeré odpovědi začínající znakem `0` jsou vždy považovány za úspěch.

Výpis kódu 6.3 Formát žádosti Gate Proxy

```
<IN|OUT>:<parkingID>:<park_token>:<plate>
IN      :2332       :1001000000  :XXX-XXXX
```

Výpis kódu 6.4 Formát odpovědi Gate Proxy

```
<0>:<park_token>
0  :1001000000
<0-4>
0
```

Výpis kódu 6.5 Konfigurace Nginx page discovery service

```
try_files $uri $uri/ /index.html?$args /;
```

- `1` — Serverové selhání
- `2` — Klientské selhání, chybné údaje
- `3` — Chybné formátování UDP žádosti
- `4` — Systémové selhání proxy služby

6.3 Frontend

Služba *Frontend* vytváří statické webové stránky založené na technologii Single-page application vytvořené pomocí Javascript frameworku React, více v sekci 3.3 a 4.5.

6.3.1 Konfigurace

Díky použité technologii statických stránek, které vyžadují před produkčním spuštěním úplné sestavení, není možná dynamická konfigurace pomocí proměnných prostředí. Pro změnu konfigurace je třeba opakovat znovu sestavení celé služby.

Konfigurace, která je možná pouze pomocí argumentů při sestavování projektu:

- `API_URL` — Webová adresa API
- `PUBLIC_URL` — Veřejná adresa statických stránek
- `PUBLIC_URL_SUB` — Podpora prefix proxy, více v sekci 8.2

6.3.1.1 Nginx

Webové stránky aplikace jsou přímo dostupné pomocí nástroje Nginx, který slouží jako server pro statické soubory. Použití technologie Single-page application s knihovnou React Router DOM vyžaduje doplňující konfiguraci, která umožňuje přístup k hlavní stránce aplikace při použití neznámé adresace, více v ukázce 6.5.

6.3.2 Login storage

Fungování uživatelské a administrátorské sekce stránek je omezeno úspěšným přihlášením pomocí jména a hesla. Jelikož používané REST rozhraní využívá autorizaci pomocí schématu HTTP Basic, více v sekci 3.2.1, je nutné bezpečné uložení autorizačního řetězce, který není časově limitován a obsahuje heslo v lehce čitelné podobě.

- Context — Data jsou uložena v dynamických proměnných daného programovacího jazyku, jež omezuje dostupnost pro určitou část aplikace s nemožností dlouhodobého uložení

Výpis kódu 6.6 FETCH Service

```
fetch: (method, url, body = undefined, authHeader = undefined) => {
    return fetch(
        url, {
            method: method,
            headers: {
                'Content-Type': 'application/json',
                'Authorization': authHeader },
            body: body ? JSON.stringify(body) : undefined
        }
    )
}
```

- Local Storage — Data jsou uložena v paměti prohlížeče s možností dlouhodobého přihlášení

Ačkoliv systém contextového ukládání neumožňuje dlouhodobé přihlášení uživatele, je přesto používán, aby snížil riziko a dopad bezpečnostního selhání.

6.3.3 Services

Služby oddělují způsob načítání a provádění uživatelských akcí od samotné potřeby vyskytujících se v dalších částech aplikace.

Podporují přímý přístup k datům pomocí souvisejících identifikátorů, `getByID(id)`, nebo pomocí odkazů generovaných technologií HATEOAS, `getByLink(link, auth)`, více v sekci 3.5. Při použití prefixových proxy serverů je třeba dodatečné konfigurace pro správnou funkčnost odkazů, více v sekci 8.2.

Načítání dat je prováděno pomocí Javascript async frameworku, který poskytuje jednoduché ovládání pro blokované I/O operace. Selhání žádosti na API, 4xx a 5xx, je uživateli automaticky zobrazeno speciální hláškou s popisem konkrétní chyby a důvodem selhání.

6.3.3.1 Fetch

Služba *Fetch* je klíčovou částí umožňující načítání dat ze serveru. Na ukázce 6.6 je zobrazena její část, která zajišťuje spojení ostatních služeb s vestavěnou JS funkcí pro dynamické načítání, FETCH.

Kód 6.6 podporuje použití autentizační hlavičky pro schéma HTTP Basic, změnu HTTP metody a dodatečné připojení těla žádosti ve formátu JS objektu.

6.3.4 Pages

Ačkoliv je aplikace postavena na technologii SPA, současně využívá knihovnu React Router DOM, která umožňuje oddělení různých částí aplikace pomocí URL adresace zobrazované klientovi [32], vice v 6.3.1.1.

Stránky jsou rozděleny do kategorií podle vyžadovaného oprávnění:

- `/auth/*` — Stránky nutně vyžadující přihlášení s rolí USER nebo ADMIN, více v 5.3.2.2
- `/*` — Veřejné stránky, které mohou obsahovat skryté části v závislosti na uživatelských oprávněních

Kapitola 7
Testování

Během vývoje každého softwaru vznikají malé i velké chyby nemající zásadní dopad na systém, ale na druhou straně mohou způsobit až kritické selhání nebo nefunkčnost celé aplikace. Právě testování si klade za cíl včasné odhalení chyb, čímž předchází potencionálním problémům [33]. Podnikové a vícevrstvé aplikace využívají speciálních návrhových vzorů (IoC a DI), aby usnadnily implementaci různých druhů testů. Ty lze primárně rozdělit podle použitého způsobu testování, cílů a samotných autorů.

Většina z nich může být prováděna manuálně nebo automaticky. Výhodou automatizovaných testů je jejich znovupoužitelnost, která urychluje testování samotné aplikace a pomáhá předcházet návratu již vyřešených problémů. Současně lze zapojit tyto testy do procesu sestavování aplikace, kdy přeruší proces nasazení při selhání implementovaných testů.

7.1 Unit testy

Podle [34] mají unit testy za cíl izolovaně ověřit základní funkčnosti jednotlivých aplikačních objektů. Samotné testy pracují na základě oddělení testovaného objektu od zbytku systému. Technologii Mock objektů simuluje chování ostatních netestovaných částí systému. Třebaže tyto testy pomáhají odhalit chyby v implementované logice dané třídy, nedokáží však odhalit komplexní chyby vznikající při komunikaci jednotlivých částí programu. Další nevýhodou použití unit testů je množství potřebných simulovaných závislostí při testech silně provázaných aplikačních tříd.

Z těchto důvodu tyto testy nebyly použity k prověření parkovací aplikace. Pro ověření funkčnosti systému se zde používají automaticky spouštěné systémové integrační testy přes webové API.

7.2 Integrační testy

Integrační testy se zaměřují na testování komunikace mezi jednotlivými částmi aplikace (databáze, webový server) [34]. Na rozdíl od unit testů se prověřování pomocí integračních testů provádí v prostředí, které je velmi podobné produkčnímu. Vzhledem k tomu že je systém připojen na všechny potřebné testovací služby, pak toto prostředí nemusí obsahovat žádné simulované objekty. Hlavním cílem implementovaných testů je ověření základních funkčností popsaných v dokumentaci, více v sekci 3.6.1. Nevýhodou integračních testů je jejich celková doba trvání, která zvláště při obnově do výchozího stavu po každém provedeném testu výrazně zpomaluje celkové sestavení systému.

Tabulka 7.1 Implementované testy

Třída	Počet	Stav	Veřejné	Neveřejné
`/auth/UserController`	11	40%	ANO	ANO
`/auth/token/TokenController`	10	100%	ANO	ANO
`/auth/permission/PermissionController`	22	100%	ANO	ANO
`/auth/permission/UserPermissionController`	9	100%	ANO	ANO

Aktuální stav automatických testů souvisejících s jednotlivými částmi webového rozhraní je popsán v tabulce 7.1, zbývající části rozhraní byly testovány ručně pomocí nástroje Postman.

Kapitola 8
Provoz

8.1 První spuštění

Přestože aplikace nabízí přednastavenou spustitelnou verzi, tak bezpečný produkční provoz by neměl v žádném případě nabízet možnost vytvářet další administrátorské účty ani veřejně dostupnou dokumentaci Swagger UI, více v sekci 3.6.1.

8.1.1 Konfigurace

- `application.pm.user.promotion-enabled` — Aktivace systému umožňující povýšení uživatelů
- `application.pm.user.promotion-token` — Klíč nutný k povýšení uživatele
- `springdoc.api-docs.enabled` — Aktivace veřejné dokumentace

8.2 Proxy

Potřeba vyšší bezpečnosti nebo podnikové standardy mohou vyžadovat umístění serverů za prefixový reverzní proxy server. Umístění aplikace za tento server vyžaduje změnu konfigurace některých částí pro správné fungování.

8.2.1 Backend

- `application.pm.email.web` — Veřejná adresa včetně případné cesty ke složce obsahující základní webovou stránku, více v sekci 6.3.1.1
- `server.forward-headers-strategy` — Aktivace podpory speciálních hlaviček automaticky popisující způsob přesměrování, `X-Forwarded-*`
- `server.servlet.context-path` — Podpora dostupnosti služeb na podadresáři

8.2.2 Frontend

- `API_URL` — Veřejná adresa Backend API
- `PUBLIC_URL` — Veřejná adresa webového serveru, základ dostupnosti statických souborů

Výpis kódu 8.1 Konfigurace Ofelia

```
labels:
    ofelia.enabled: "true"
    ofelia.job-exec.db_backup.schedule: "@daily"
    ofelia.job-exec.db_backup.command: "sh -c
        'pg_dumpall -U pm_database --column-inserts
            -f ./home/dump/$$(date +%F_%R).dump'"
```

- `PUBLIC_URL_SUB` — Podpora stránek běžící mimo kořenovou adresu, při použití prefixového proxy serveru

8.3 Zálohování

Softwarová nebo hardwarová selhání představují zásadní rizika ohrožující fungování celé aplikace. Mezi mechanismy, které se snaží předcházet vzniku takových situací, patří důkladné testování a pravidelná údržba systému.

Zálohování je jedním ze základních mechanismů, které snižují celkový dopad selhání na systém. Klíčovým aspektem celého mechanismu je pravidelná a častá tvorba záloh, které umožňují jednoduché a rychlé obnovení poškozených nebo ztracených dat.

8.3.1 Ofelia

Podle [35] je Ofelia moderní a jednoduchý plánovač úloh pro Docker prostředí implementovaný v jazyce Go. Nástroj, který podporuje přímé napojení pomocí soketů na Docker systém, umožňuje plánované a pravidelné spouštění úloh ve všech kontejnerech aplikace. Konfigurace úlohy a četnost jejího opakování je založena na metadatech připojených k jednotlivým službám, více v ukázce 8.1.

Nástroj je použit pro automatické generování souborů popisujících aktuální strukturu databázových entit, včetně všech dostupných dat. Obnovení databáze podle daného souboru je komplexní procedura vyžadující dočasnou deaktivaci kontroly cizích klíčů, jelikož entita *Parking* obsahuje přímou rekurzivní relaci, více v sekci 5.1.3.3.

Kapitola 9
Závěr

Výsledkem bakalářské práce je vytvoření plně funkčního informační systému, který umožňuje centrální správu parkovišť. Systém je rozdělen do několika různých částí, které nabízí veřejně dostupné aplikační rozhraní, zákaznické webové stránky a speciální služby umožňující připojení externích zařízení. Implementovaná aplikace umožňuje podporu různých druhů parkování, včetně flexibilního systému rezervace parkovacích míst. Součástí tohoto vytvořeného nástroje je rovněž zákaznické rozhraní podporující komplexní rozdělení uživatelských práv.

Práce se kromě tvorby samotného systému zabývá jeho analýzou, návrhem, testováním a provozem. Analýza se věnuje již existujícím nástrojům pro správu parkovišť a jejich schopnostem. Na základě této analýzy byl navržen speciální systém, který umožňuje podporu různých technických zařízení provozovatele. Současně je v návrhu blíže představena databázová struktura projektu, včetně vzhledu a funkcionalit zákaznických stránek. Cílem testování je představit mechanismy předcházení chyb a následné vytvoření vhodných automatizovaných testů. Část zabývající se provozem se věnuje předcházení rizik plynoucích z možného technického selhání a také bližší konfiguraci nutné pro bezpečný produkční provoz.

Jednou z nejnáročnější částí práce je vytvoření dostatečně kvalitního webového rozhraní, umožňujícího dosažení všech stanovených cílů. Ke konfigurace a nastavení samotných služeb je zapotřebí dodatečná modifikace Docker kontejnerů pro jejich vzájemnou komunikaci. V neposlední řadě je třeba přidání speciální externí služby, která slouží jako mailový server s webovým UI.

Ačkoliv vytvořený systém je připraven k okamžitému spuštění a použití, je možné, že používání daných zařízení provozovatelem bude vyžadovat dodatečný vývoj. V tomto směru je patrné, že práci je možné dále rozšiřovat a obohacovat, tj. umožnit vyšší konfiguraci a podporu externích služeb.

Příloha A

Přílohy

Výpis kódu A.1 UDP Outbound

```
@Bean
@ServiceActivator(inputChannel = "outputChannel")
public UnicastSendingMessageHandler unicastSendingMessageHandler() {
    UnicastSendingMessageHandler unicastSendingMessageHandler
        = new UnicastSendingMessageHandler(
            "headers['ip_packetAddress']"
        );

    /* Reusing receiver port */
    unicastSendingMessageHandler
        .setSocketExpressionString("@input.socket");
    return unicastSendingMessageHandler;
}
```

Výpis kódu A.2 Vytváření nového uživatele

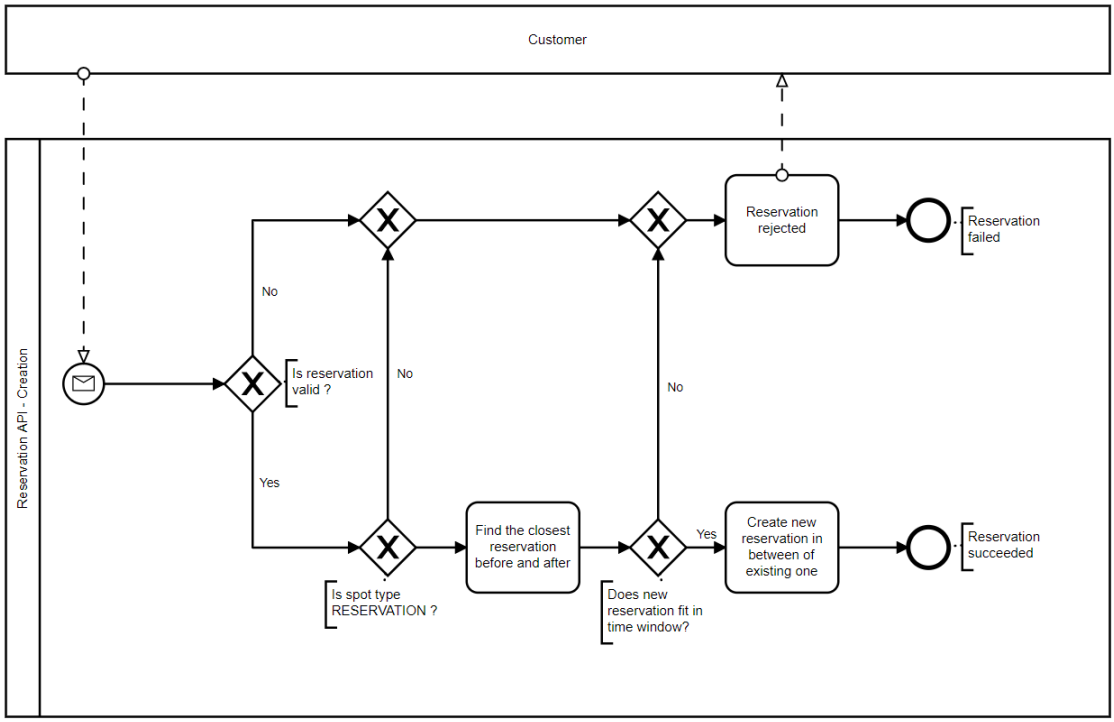
```
User user = userRegistrationMapper.toSource(userDTO);
user.setPassword(passwordEncoder.encode(userDTO.getPassword()));
user.setCreationDate(LocalDateTime.now());
user.setEnabled(false);
user.setLocked(false);
```

Výpis kódu A.3 Thymeleaf template

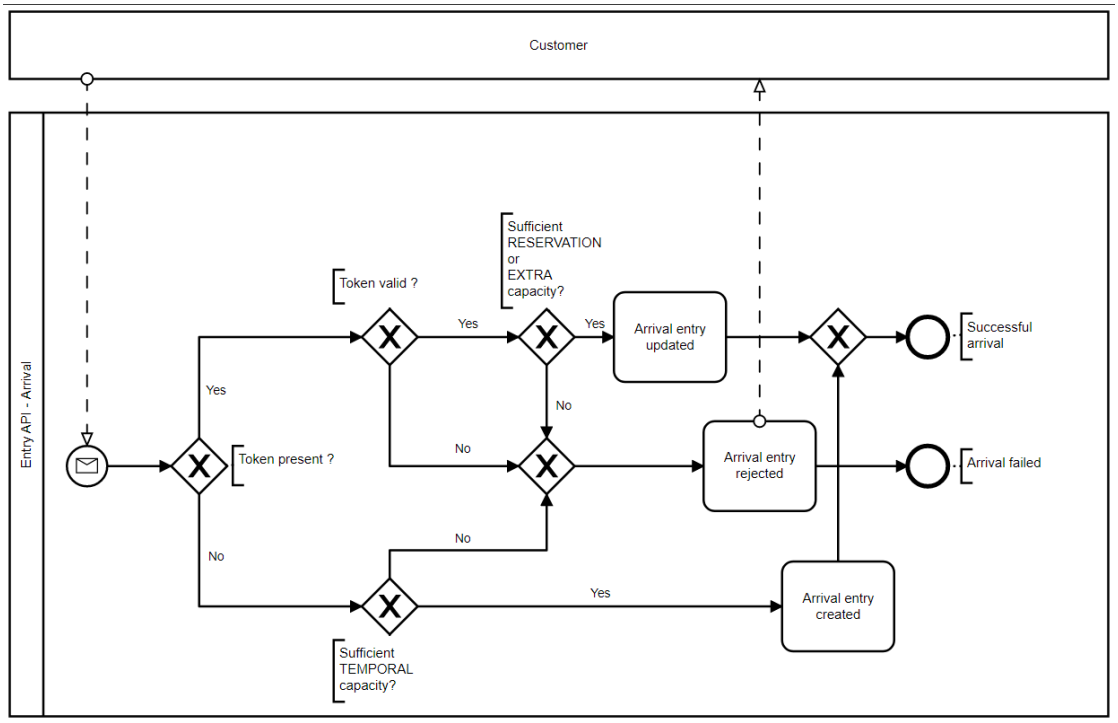
```
<body style="background-color: #f8f9fa;">
    <img src='cid:logo' width="180px" />
    <hr align="left" style="width: 200px;">
    <p>Dear <span th:text="${lastName}"></span>,</p>
    <p>
        <div>Reservation has been successfully created.</div>
        <div>
            <a th:href="${address} + 'reservation/' + ${reservationID}">
                Reservation link
            </a>
        </div>
    </p>
    <hr align="left" style="width: 200px;">
    <img src='cid:barcode' width="180px" />
</body>
```

Tabulka A.1 Platné PATCH operace

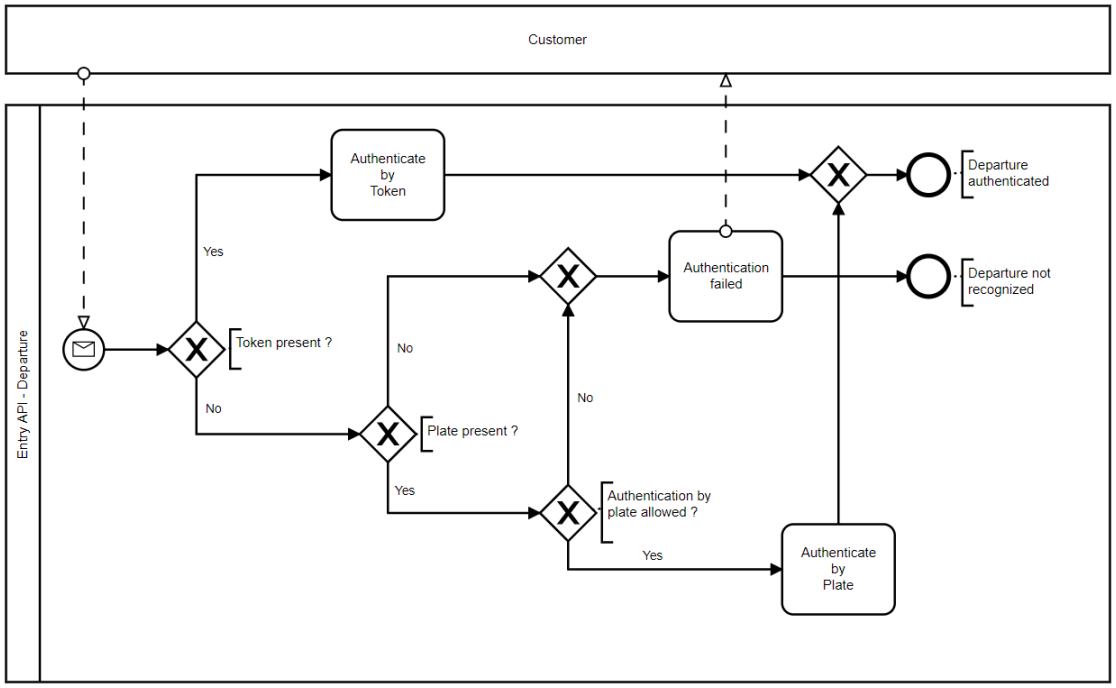
Operace	Popis
add	Přidání nebo nahrazení hodnoty
remove	odebrání hodnoty
replace	Nahrazení hodnoty
move	Přesun hodnoty
copy	Kopírování hodnoty
test	Test hodnoty oproti žádosti



Obrázek A.1 BPMN Vytvoření rezervace



Obrázek A.2 BPMN Příjezd automobilu



Obrázek A.3 BPMN Odjezd automobilu

Bibliografie

1. GREENCENTER. *O Společnosti GREEN Center* [online]. 2019 [cit. 2021-04-26]. Dostupné z: https://www.green.cz/clanek-o-nas-52-137.
2. PARKINGBOXX. *About US | Parking BOXX* [online]. 2020 [cit. 2021-04-26]. Dostupné z: https://parkingboxx.com/about-us.
3. PARKINGBOXX. *Parking system pricing* [online]. 2020 [cit. 2021-04-26]. Dostupné z: https://parkingboxx.com/parking-system-pricing.
4. LOXONE. *Automatizace a řízení pro komerční budovy* [online]. 2020 [cit. 2021-04-13]. Dostupné z: https://www.loxone.com/cscz/komercni-budovy/.
5. FIELDING, Roy et al. *Hypertext Transfer Protocol – HTTP/1.1* [online]. 1999 [cit. 2021-04-12]. Dostupné z: https://tools.ietf.org/html/rfc2616.
6. RESCHKE, Julian. *The 'Basic' HTTP Authentication Scheme* [online]. 2015 [cit. 2021-04-13]. Dostupné z: https://tools.ietf.org/html/rfc7617.
7. FINK, Gil et al. *Pro Single Page Application Development* [online]. Berkeley, CA: Apress, 2014 [cit. 2021-04-17]. ISBN 978-1-4302-6674-7. Dostupné z: https://doi.org/10.1007/978-1-4302-6674-7_1.
8. VARANASI, Balaji et al. *Spring REST* [online]. Berkeley, CA: Apress, 2015 [cit. 2021-04-11]. ISBN 978-1-4842-0823-6. Dostupné z: https://doi.org/10.1007/978-1-4842-0823-6_1.
9. BAELDUNG. *An Intro to Spring HATEOAS* [online]. 2021 [cit. 2021-04-17]. Dostupné z: https://www.baeldung.com/spring-hateoas-tutorial.
10. KELLY, Mike. *HAL - Hypertext Application Language* [online]. 2013 [cit. 2021-04-13]. Dostupné z: https://stateless.group/hal_specification.html.
11. EPSTEIN, Ted et al. *OpenAPI Specification* [online]. 2018 [cit. 2021-04-14]. Dostupné z: https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.2.md.
12. VARGA, Ervin. *Creating Maintainable APIs* [online]. Berkeley, CA: Apress, 2016 [cit. 2021-04-19]. ISBN 978-1-4842-2196-9. Dostupné z: https://doi.org/10.1007/978-1-4842-2196-9_9.
13. HERCOG, Drago. *Communication Protocols* [online]. Cham: Springer International Publishing, 2020 [cit. 2021-04-12]. ISBN 978-3-030-50405-2. Dostupné z: https://doi.org/10.1007/978-3-030-50405-2_22.
14. SHARAN, Kishori. *Java Language Features* [online]. Berkeley, CA: Apress, 2018 [cit. 2021-04-27]. ISBN 978-1-4842-3348-1. Dostupné z: https://doi.org/10.1007/978-1-4842-3348-1_1.

15. LOMBOK. *Java - Project Lombok* [online]. 2009 [cit. 2021-04-23]. Dostupné z: https://projectlombok.org/.
16. MOLIN, Cassio Mazzochi. *Using HTTP PATCH in Spring* [online]. 2019 [cit. 2021-04-16]. Dostupné z: https://cassiomolin.com/2019/06/10/using-http-patch-in-spring/.
17. BAELDUNG. *Performance of Java Mapping Frameworks* [online]. 2020 [cit. 2021-04-13]. Dostupné z: https://www.baeldung.com/java-performance-mapping-frameworks.
18. BAELDUNG. *Quick Guide to MapStruct* [online]. 2021 [cit. 2021-04-17]. Dostupné z: https://www.baeldung.com/mapstruct.
19. HALTERMAN, Jonathan. *Property Mapping* [online]. 2019 [cit. 2021-04-13]. Dostupné z: http://modelmapper.org/user-manual/property-mapping/.
20. SARCAR, Vaskaran. *Java Design Patterns* [online]. Berkeley, CA: Apress, 2019 [cit. 2021-04-24]. ISBN 978-1-4842-4078-6. Dostupné z: https://doi.org/10.1007/978-1-4842-4078-6_26.
21. KO, Timothy. *A Quick Glance of Django* [online]. 2017 [cit. 2021-04-18]. Dostupné z: https://medium.com/@timmykko/a-quick-glance-of-django-for-beginners-688bc6630fab.
22. GACKENHEIMER, Cory. *Introduction to React* [online]. Berkeley, CA: Apress, 2015 [cit. 2021-04-25]. ISBN 978-1-4842-1245-5. Dostupné z: https://doi.org/10.1007/978-1-4842-1245-5_1.
23. SO, Preston. *Decoupled Drupal in Practice* [online]. Berkeley, CA: Apress, 2018 [cit. 2021-04-16]. ISBN 978-1-4842-4072-4. Dostupné z: https://doi.org/10.1007/978-1-4842-4072-4_1.
24. JIA, Rachel. *The Comprehensive Guide to 1D and 2D Barcodes* [online]. 2020 [cit. 2021-04-16]. Dostupné z: https://www.dynamsoft.com/blog/insights/the-comprehensive-guide-to-1d-and-2d-barcodes/.
25. KULSHRESTHA, Saurabh. *Get Started With Maven For Building Java Applications* [online]. 2019 [cit. 2021-04-13]. Dostupné z: https://medium.com/edureka/maven-tutorial-2e87a4669faf.
26. VARANASI, Balaji et al. *Introducing Maven* [online]. Berkeley, CA: Apress, 2014 [cit. 2021-04-13]. ISBN 978-1-4842-0841-0. Dostupné z: https://doi.org/10.1007/978-1-4842-0841-0_1.
27. GUTIERREZ, Felipe. *Pro Spring Boot 2* [online]. Berkeley, CA: Apress, 2019 [cit. 2021-04-27]. ISBN 978-1-4842-3676-5. Dostupné z: https://doi.org/10.1007/978-1-4842-3676-5_10.
28. JOSEFSSON, Simon et al. *The Base16, Base32, and Base64 Data Encodings* [online]. 2006 [cit. 2021-04-21]. Dostupné z: https://tools.ietf.org/html/rfc4648.
29. ALEX, Ben et al. *Spring Security Reference* [online]. 2020 [cit. 2021-04-18]. Dostupné z: https://docs.spring.io/spring-security/site/docs/5.2.7.RELEASE/reference/html5/.
30. LEACH, Paul et al. *A Universally Unique IDentifier (UUID) URN Namespace* [online]. 2005 [cit. 2021-04-17]. Dostupné z: https://tools.ietf.org/html/rfc4122.
31. VOHRA, Deepak. *Docker Management Design Patterns* [online]. Berkeley, CA: Apress, 2017 [cit. 2021-04-16]. ISBN 978-1-4842-2973-6. Dostupné z: https://doi.org/10.1007/978-1-4842-2973-6_10.
32. SUBRAMANIAN, Vasan. *Pro MERN Stack* [online]. Berkeley, CA: Apress, 2019 [cit. 2021-04-25]. ISBN 978-1-4842-4391-6. Dostupné z: https://doi.org/10.1007/978-1-4842-4391-6_9.

33. DEINUM, Marten et al. *Spring 5 Recipes* [online]. Berkeley, CA: Apress, 2017 [cit. 2021-04-14]. ISBN 978-1-4842-2790-9. Dostupné z: https://doi.org/10.1007/978-1-4842-2790-9_16.

34. COSMINA, Iuliana et al. *Pro Spring 5* [online]. Berkeley, CA: Apress, 2017 [cit. 2021-04-17]. ISBN 978-1-4842-2808-1. Dostupné z: https://doi.org/10.1007/978-1-4842-2808-1_13.

35. CUADROS, Máximo et al. *mcuadros/ofelia* [online]. 2021 [cit. 2021-04-13]. Dostupné z: https://github.com/mcuadros/ofelia.

Obsah přiloženého média

- `.gitignore` — Nastavení GIT repozitáře
- `.gitlab-ci.yml` — Nastavení Gitlab CI
- `README.md` — Dokumentace a instalační příručka
- `backend`
  - `core` — zdrojové kódy v jazyce Java, služba Core API
  - `gateProxy` — zdrojové kódy v jazyce Java, služba Gate Proxy
  - `spotProxy` — zdrojové kódy v jazyce Java, služba Spot Proxy
- `frontend`
  - `core` — zdrojové kódy v jazyce JS, služba Web Core
- `scripts`
  - `local` — lokální vstupní bod
  - `local_sample` — lokální plně konfigurovaný vstupní bod
  - `remote` — vzdálený vstupní bod
  - `remote_sample` — vzdálený plně konfigurovaný vstupní bod
- `tests` — testy pro aplikaci Postman
- `thesis`
  - `database.pdf` — struktura databáze ve formátu PDF
  - `thesis.pdf` — text práce ve formátu PDF
  - `src` — zdrojový kód práce ve formátu LaTeX