Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Rezervační systém letenek

Bc. Matěj Černovický

Diplomová práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Matěj Černovický**
Osobní číslo:	**I16209**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Rezervační systém letenek**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem diplomové práce je vytvořit systém pro rezervaci letenek, který bude tvořen zvlášť webovou a zvlášť mobilní aplikací.
V teoretické části práce bude v úvodu provedena rešerše a porovnání dostupných webových technologií a frameworků s důrazem na využitelnost technologie v kombinaci s webovými službami (REST, SOAP, ...). V další části práce budou popsány použité technologie a realizován návrh vlastní webové a mobilní aplikace.
V praktické části bude analyzována a implementována aplikace, která bude umožňovat uživatelům si zarezervovat vybrané letenky. Webová aplikace bude realizována na platformě Java a frameworku Spring s použitím nejnovějších technologií. Mobilní aplikace bude napsána pro operační systém Android. Mezi základní funkcionality bude patřit evidence letenek a jejich správa (slevy, akce, statistiky, grafy). V aplikaci budou vytvořeny minimálně 3 role - administrátor, prodejce a zákazník.

Rozsah grafických prací:

Rozsah pracovní zprávy: **50-60 stran**

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

Seznam odborné literatury:

AMUTHAN, G. Spring MVC Beginner's Guide. Packt Publishing, 2014. ISBN 1-78328-488-9.
WALLS, Craig. Spring in action. Fourth Edition. Shelter Island, NY: Manning, 2015. ISBN 161729120x.
WALLS, Craig. Spring Boot in action. Shelter Island, NY: Manning Publications, 2016. ISBN 9781617292545.
LACKO, L'uboslav. Mistrovství - Android. Přeložil Martin HERODEK. Brno: Computer Press, 2017. Mistrovství. ISBN 9788025148754.

Vedoucí diplomové práce: **Ing. Roman Diviš**
Katedra softwarových technologií

Datum zadání diplomové práce: **30. října 2017**
Termín odevzdání diplomové práce: **18. května 2018**

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2017

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 23. 8. 2019

Matěj Černovický

PODĚKOVÁNÍ

Na tomto místě bych rád poděkoval vedoucímu mé diplomové práce Ing. Romanu Divišovi za poskytnutí cenných rad, připomínek a vedení při zpracování této diplomová práce. Dále bych chtěl poděkovat rodičům, kamarádům a své přítelkyni Aleně Štreblové za trpělivost a podporu během mého studia.

ANOTACE

Cílem diplomové práce je pomocí moderních technologií navrhnout a vytvořit jednoduchý systém, který umožňuje rezervovat letenky. V teoretické části jsou rozebrány principy webových služeb (SOAP, REST, GraphQL), provedena rešerše dostupných webových technologií a popsány použité technologie a nástroje. V praktické části je provedena analýza a implementace rezervačního systému.

KLÍČOVÁ SLOVA

Java, Spring, Angular7, Android, rezervační systém, webové služby, REST

TITLE

Flight booking system.

ANNOTATION

The aim of the master thesis is to design and create a simple system for booking air tickets using modern technologies. In the theoretical part are discussed principles of web services (SOAP, REST, GraphQL), then is done research of available web technologies and description of used technologies and tools. In the practical part is done an analysis and implementation of the reservation system.

KEYWORDS

Java, Spring, Angular7, Android, reservation system, web services, REST

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK A ZNAČEK

AOP	Aspect Oriented Programming
API	Application Programming Interface
DI	Dependency Injection
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JPA	Java Persistence API
JPQL	Java Persistence Query Language
JSX	JavaScript Syntax Extension
MVC	Model-View-Controller
MVVM	Model-View-ViewModel
MySQL	My Structured Query Language
NPM	Node Package Manager
REST	Representational State Transfer
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
SPA	Single Page Application
SQL	Structured Query Language
TCP	Transmission Control Protocol
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	Extensible Markup Language

ÚVOD

S pojmem rezervační systém se zajisté každý z nás již setkal. Často bývá spojen s realizací konkrétní služby. V dnešní době, kdy čas jsou peníze a málokdy je ho dostatek, slouží jako efektivní pomocník v běžném životě. Narazit na něj můžete téměř kdekoliv, od rezervace času objednání u lékaře po složitější systémy používané v hotelnictví, školství či dopravě.

Cílem diplomové práce je za pomoci moderních technologií navrhnout a implementovat systém pro rezervaci letenek. V rámci systému jsou vytvořeny dvě klientské aplikace, a to webová a mobilní. Obě dvě aplikace získávají svá data ze serveru, který je založen na technologii Spring, skrze webové rozhraní REST.

Práce je rozdělena na několik částí. V teoretické části je nejprve představen pojem rezervační systém spolu s krátkým úvodem do jeho historie. Druhá kapitola je věnována webovým službám a protokolu HTTP. Jsou zde popsány nejpoužívanější způsoby realizace webových služeb a jejich následné porovnání. Konkrétně se jedná o protokol SOAP, architekturu REST a dotazovací jazyk GraphQL. Ve třetí kapitole je provedena rešerše vybraných webových technologií a frameworků. V závěru kapitoly jsou tyto technologie porovnány v kombinaci s webovými službami uvedenými ve druhé kapitole. Ve čtvrté kapitole se nachází popis technologií, které byly použity při návrhu a vývoji rezervačního systému. U jejich výběru byl kladen důraz na případnou snadnou rozšiřitelnost systému a aby odpovídaly moderním trendům v dané oblasti.

Další část práce je zaměřena na samotnou realizaci rezervačního systému. Nejprve je provedena jeho analýza, která začíná definováním funkčních a nefunkčních požadavků. Dále jsou zde charakterizovány dostupné uživatelské role vyskytující se v systému. V rámci analýzy je samozřejmostí využití vybraných UML diagramů. Například pro lepší pochopení jednotlivých rolí a jejich práv je vytvořen UML Use Case diagram. Kromě toho se v této části nachází také návrh schématu databáze pomocí E-R diagramu spolu s výpisem nejdůležitějších tabulek. Závěr této kapitoly popisuje implementaci vyvíjeného systému. Větší důraz je kladen na server, protože se jedná o hlavní část systému. V poslední kapitole diplomové práce jsou podrobně popsány jednotlivé funkcionality webové a mobilní aplikace. Tato část se dá označit za uživatelskou příručku, po jejímž přečtení by měl být uživatel seznámen s obsluhou systému.

Od čtenáře této diplomové práce se očekávají alespoň základní znalosti z oblasti tvorby moderních webových a mobilních aplikací. Kromě toho je vhodná také znalost jazyka Java, TypeScript a standardu SQL.

1 REZERVAČNÍ SYSTÉM

Rezervační systém je specifickým případem informačního systému, který uživateli umožňuje vybrat ze širokého seznamu komodit. Komoditu je poté možné zarezervovat a případně ji i koupit, přičemž vše probíhá v reálném čase.

Rezervační systémy lze z hlediska plošného pokrytí rozdělit na lokální rezervační systémy, regionální a celostátní informačně-rezervační systémy, centrální rezervační systémy a globální distribuční systémy. Centrální rezervační systémy neboli CRS byly vyvinuty ve spolupráci letecké společnosti American Airlines se společností IBM na počátku 70. let minulého století. Na počátku byly provozovány pouze na intranetu, kde sloužily pro potřeby administrace letecké společnosti. S postupem času, jak sílila konkurence a požadavky, byl systém zpřístupněn partnerským subjektům. Partneři tak získali aktuální informace o leteckých spojeních, řádech či cenách letů. [1]

V 80. letech minulého století se kvůli neustále zvyšujícím požadavkům a nástupu globalizace z CRS systému vyvinuly tzv. globální distribuční systémy (GDS). Jejich hlavním úkolem bylo rozšířit a zkvalitnit funkce CRS. Jsou realizovány pomocí internetu a terminálů umístěných po celém světě. Přes terminály se jednotlivé subjekty propojují. Za nejznámější globální systémy současnosti jsou považovány Sabre, Amadeus, Galileo a Worldspan (viz obrázek 1). [1]

Obrázek 1 – Globální distribuční systémy [2-4]

Sabre je nejstarší globální distribuční systém. Byl založen v roce 1960 leteckou společností American Airlines. V současné době pokrývá všechny oblasti letectví po celém světě. Používá ho více než 435 tisíc cestovních kanceláří, ročně je vypraveno pomocí systému Sabre kolem 35 milionů letů a odbaveno přes 790 milionů cestujících. [1, 2]

Amadeus byl založen v roce 1987 společnostmi Lufthansa, Air France a Iberia. Jedná se o evropský globální systém, který je alternativou systému Sabre. Amadeus je největším GDS z hlediska podílu na trhu. Má zastoupení ve více než 190 zemích, a to hlavně v Evropě a Jižní Americe. Pobočku Amadeus lze nalézt i v České republice. [1, 3, 5]

Systém *Galileo* byl založen v roce 1971 skupinou evropských a amerických společností. Sídlo má ve Spojených státech, ale podobně jako Amadeus je významně zastoupen v Evropě, konkrétně i v České republice. Systém je navíc součástí společnosti Travelport. [1, 4, 5]

Worldspan je nejnovější systém z této vyjmenované čtveřice. Byl založen roku 1990 americkými společnostmi Delta Air Lines, Northwest Airlines a Trans World Airlines. Podobně jako systém Galileo je součástí společnosti Travelport. [1, 4]

Dále je třeba zmínit v krátkosti i informace o společnosti Travelport, která je považována za největšího provozovatele globálních distribučních systémů. Tato společnost byla založena ve Spojených státech, ale nyní sídlí ve Velké Británii. Od roku 2007 zahrnuje GDS Galileo a GDS Worldspan. Kombinuje jejich silné stránky a bezkonkurenční dosah, což je jeho velká výhoda, protože distribuuje služby a produkty po celém světě. To je rozdíl oproti GDS Amadeus, který je dominantní v Evropě, a GDS Sabre, který je dominantní v USA. Společnost má zastoupení přibližně ve 180 zemích. [4, 5]

Kromě již zmíněné letecké dopravy se dnes rezervační systémy používají snad ve všech odvětvích např. v hotelnictví, školství a železniční dopravě. Na obrázku 2 lze vidět veškeré služby, které poskytuje GDS Amadeus.

Obrázek 2 – Služby poskytované systémem Amadeus [6]

2 WEBOVÉ SLUŽBY

Veškerá komunikace a propojení rezervačních systémů je vedeno přes internet. K tomuto účelu se standardně používají webové služby. Velkou výhodou webových služeb je nezávislost na platformách, na kterých jsou komunikující systémy vytvořeny. Proto je možné pomocí tohoto rozhraní propojit i aplikace napsané v různých programovacích jazycích. Webové služby lze rozdělit do dvou skupin, a to na procedurální a datové. Mezi procedurální patří například SOAP a mezi datové REST nebo GraphQL. Ačkoliv jsou v lecčem odlišné, pro přenos dat používají primárně protokol HTTP. Představení těchto technologií je součástí následujících podkapitol.

2.1 Protokol HTTP

Protokol HTTP je bezestavový protokol aplikační vrstvy, jehož úkolem je přenos hypertextových dokumentů mezi klientem (prohlížeč nebo robot) a serverem. Jedná se tedy o architekturu klient-server, která funguje tak, že klient zašle požadavek na server, ten jej zpracuje a vrátí klientovi odpověď. Celá komunikace probíhá formou zpráv (viz kapitola 2.1.1). To, jaký má mít formát definuje specifikace HTTP, těch existuje hned několik.

Navzdory tomu, že je již standardizována verze HTTP/2, tak následující kapitoly čerpají ze specifikace starší verze, konkrétně HTTP/1.1 obsažené v dokumentu RFC 2616[1] (dostupné z [7]). Hlavním důvodem je nedostatečné rozšíření protokolu HTTP/2 napříč webovými stránkami. Podle W3Tech je zatím používán pouze u 34,6 % všech webových stránek. [8]

2.1.1 Zprávy HTTP

Jak již bylo řečeno výše, zprávy slouží ke komunikaci mezi klientem a serverem. Zpráva má následující strukturu:

```
<PRVNÍ ŘÁDEK>
<HLAVIČKY> <CRLF>
<CRLF>
<TĚLO POŽADAVKU>
```

- *První řádek* – přenáší se v textové podobě. Obsahuje dotazovací metodu/kód v závislosti na tom, zda klient adresuje zprávu serveru nebo opačně.
- *Hlavička zprávy* – obsahuje informace o zprávě nebo o datech obsažených v těle zprávy.
- *Tělo zprávy* – slouží k přenášení dat.

[1] Request for Comments (RFC) jsou dokumenty, které definují internetové standardy.

Struktura požadavku a odpovědi je téměř stejná, liší se pouze v prvním řádku a druhem použitých hlaviček. [7]

2.1.1.1 Požadavek HTTP

Na prvním řádku požadavku se nachází metoda (viz kapitola 2.1.2), která se má provést nad cílovým zdrojem, dále URI daného zdroje a verze použitého protokolu. Řádek je ukončen dvojicí znaků Cr (carriage return) a Lf (line feed). Ty byly původně používány jako řídící příkazy pro tiskárny. [7]

```
<METODA HTTP> <URI ZDROJE> <VERZE PROTOKOLU> <CRLF>
GET http//www.seznam.cz HTTP/1.1
```

2.1.1.2 Odpověď HTTP

První řádek odpovědi obsahuje údaj o verzi použitého protokolu, stavový kód (viz kapitola 2.1.3) a textový popis stavového kódu. Stejně jako u požadavku se na konci objevuje dvojice znaků Cr a Lf. [7]

```
<VERZE PROTOKOLU> <STAVOVÝ KÓD> <STAVOVÁ HLÁŠKA> <CRLF>
HTTP/1.1 201 CREATED
```

2.1.2 Metody HTTP

Standard HTTP definuje několik dotazovacích metod, které se mají provést nad uvedeným zdrojem. Nachází se v úvodní řádce zprávy společně s URI daného zdroje. [7]

- *GET* – jedná se o nejpoužívanější metodu. Používá se pro získání požadovaného zdroje identifikovaného pomocí URL adresy. Příkaz GET zpravidla nemá tělo.
- *POST* – slouží k zasílání informací, v těle požadavku, na server. Například při zasílání dat z formulářů.
- *OPTIONS* – slouží k získání povolených HTTP metod.
- *HEAD* – identická metoda jako GET s tím rozdílem, že odpověď nevrací žádné tělo. Většinou se používá pro zjištění, zda zdroj na dané adrese existuje, popřípadě k získání metadat (ta jsou stejná, jako kdyby klient zaslal příkaz GET).
- *PUT* – využívá se k upravení či nahrazení existujícího zdroje, nebo pro vytvoření nového zdroje na zadané adrese.
- *DELETE* – používá se k odstranění zdroje.
- *TRACE* – umožňuje klientům vidět cestu požadavku a použít tato data pro testovací, diagnostické či ladící účely.

- *PATCH* – upravuje zdroj zasláním pouze modifikovaných údajů. Byla přidána až později v rámci RFC 5789.
- *CONNECT* – používá se pro připojení s proxy servery, které podporují dynamické přepnutí na funkci tunelování (např. SSL tunelování).

2.1.3 Stavové kódy

Jedná se o třímístné číslo, jenž je součástí HTTP odpovědi. Umožňuje klientovi zjistit výsledek daného požadavku a patřičně na něj reagovat. První číslice kódy rozděluje do tříd a ostatní čísla ji blíže specifikují. Význam jednotlivých tříd kódů je následující:

- 1xx – jedná se o informační oznámení.
- 2xx – značí úspěšné provedení akce.
- 3xx – přesměrování (před vyřízením požadavku musí prohlížeč klienta provést další akci).
- 4xx – nastala chyba na straně klienta (např. požadavek k přístupu na stránku, která neexistuje).
- 5xx – indikují případy, kdy nastala chyba na straně serveru. Zaslaný validní požadavek nedokázal server zpracovat, protože na něm došlo k chybě.

Protokol HTTP je rozšiřitelný, takže lze ve specifických případech použít vlastní stavový kód, pokud bude zachována hlavní třída (tzn. 1xx – informační atd.). V následující tabulce (tabulka 1) jsou vybrány a popsány nejpoužívanější stavové kódy. [7]

Tabulka 1 – Přehled nejpoužívanějších stavových kódů [7]

Stavový kód	**Popis**
200 OK	Standardní oznámení o úspěšném zpracování požadavku. Obsah odpovědi je závislý na typu požadavku.
201 Created	Požadavkem byl vytvořen nový zdroj.
204 No Content	Požadavek byl úspěšně zpracován serverem. Na rozdíl od kódu 200 nevrací žádný obsah.
301 Moved Permanently	URI daného požadavku bylo změněno. Budoucí požadavky by měly být směřovány na nové URI.
400 Bad Request	Server nemůže zpracovat požadavek klienta (např. z důvodu špatné syntaxe).
401 Unauthorized	Požadavek nemůže být zpracován z důvodu nedostatečné autorizace.

Stavový kód	Popis
403 Forbidden	Server pochopil požadavek, ale odmítl na něj odpovědět.
404 Not Found	Server nenalezl požadovaný zdroj.
500 Internal Server Error	Na serveru nastala chyba, která brání dokončení požadavku.

2.1.4 HTTP/2

Od roku 2015 je dostupná zatím nejnovější verze s označením HTTP/2 (založena na protokolu SPDY od Googlu). Nová verze přináší oproti verzi HTTP/1.1 především vylepšení po stránce výkonnostní, konkrétně v rychlosti přenosu dat. Níže se nachází výčet nejdůležitějších změn [9-11]:

- *Multiplexing* – umožňuje po jednom TCP spojení přenášet více požadavků. To sice již existovalo ve verzi HTTP/1.1 pomocí pipeliningu, ale bylo nutné, aby požadavky dorazily ve stejném pořadí, jaké měly při odeslání. Ve verzi HTTP/2 na pořadí nezáleží.
- *Server Push* – server může prohlížeči zaslat objekty ještě předtím, než si o ně požádá. Typicky se jedná o styly, skripty nebo obrázky používané na požadované stránce. Server Push nefunguje automaticky, je nutné mu to oznámit pomocí hlavičky LINK.
- *Komprimování hlaviček* a *přenášených cookies* pomocí nového algoritmu HPACK.
- *Binární formát zpráv*.
- *Prioritizace operací*.

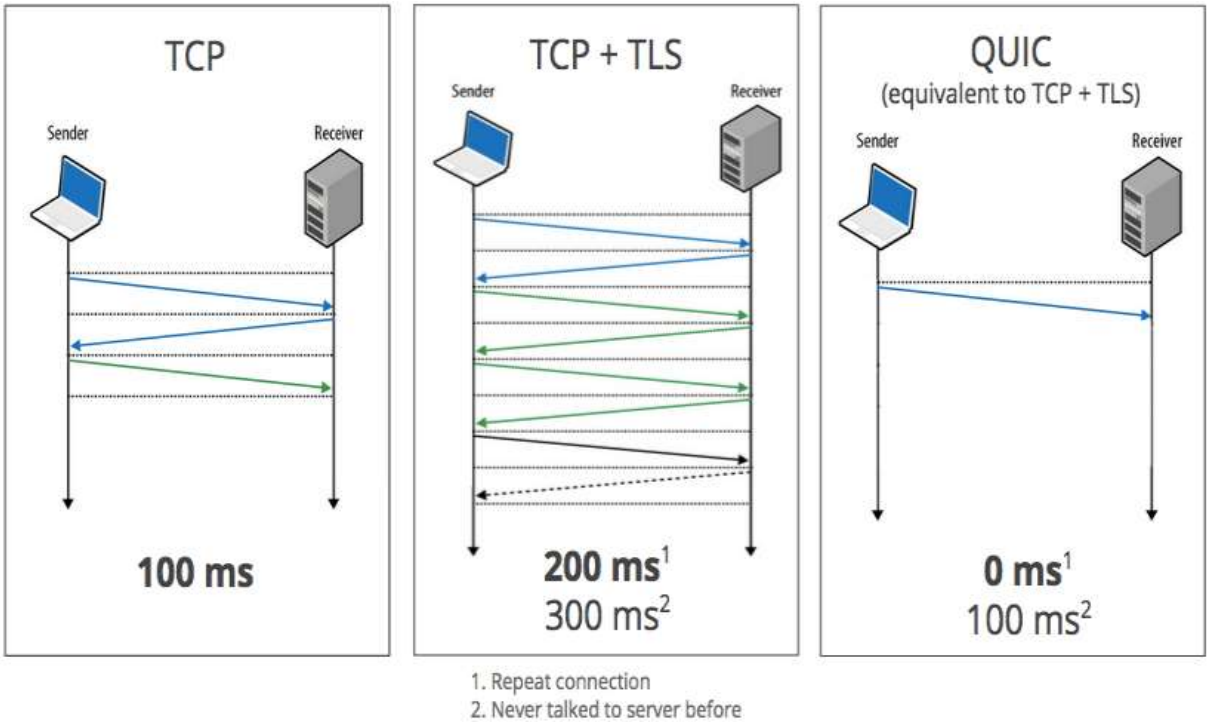
2.1.5 HTTP/3 (HTTP over QUIC)

Všechny současné verze protokolu HTTP (tedy 1.0, 1.1 a 2.0) používají transportní protokol TCP. Ten zaručuje, že se pakety dostanou do cíle bez chyb a ve správném pořadí. V případě problémů při přenosu (např. chybně přenesená data) je schopen sám chyby detekovat a zjednat nápravu. Vzhledem k tomu, že při každém navázání spojení je nutné vyměňovat velké množství paketů, může být použití protokolu TCP pro některé situace až příliš robustní. [12, 13]

V roce 2012 se společnost Google rozhodla vyvinout vlastní univerzální protokol, který by odstraňoval nedokonalosti protokolu TCP (značná komunikační zátěž). A tak, o rok později, vznikl QUIC neboli Quick UDP Internet Connection. Už podle názvu je zřejmé, že není postaven nad protokolem TCP, ale nad UDP (User Datagram Protocol). Protokol UDP se dá označit za protipól k TCP, na rozdíl od něj nezaručuje doručení paketů ani jejich pořadí. Na

druhou stranu je jednoduchý a odpovědnost za spolehlivost přenosu přesouvá na aplikační vrstvu. [12, 13]

Jednou z hlavních výhod technologie je snížení komunikační zátěže. Pokud se klient připojí k serveru, se kterým již komunikoval, může QUIC posílat požadavky bez jakéhokoliv sestavování spojení (handshake). V případě připojení na server nový umožňuje QUIC sestavit spojení za čas totožný jako u TCP. Porovnání komunikační zátěže je k dispozici na následujícím obrázku (obrázek 3). [12, 13]



Obrázek 3 – Režie TCP spojení a QUIC [13]

Od roku 2015 Internet Engineering Task Force (IETF) připravuje standardizovanou podobu protokolu QUIC. Ta se bude v určitých aspektech lišit od verze, kterou navrhl a používá Google. Zatímco původní verze počítala pouze s protokolem HTTP, verze vzniklá v IETF dovoluje využít jakýkoliv aplikační protokol. [13]

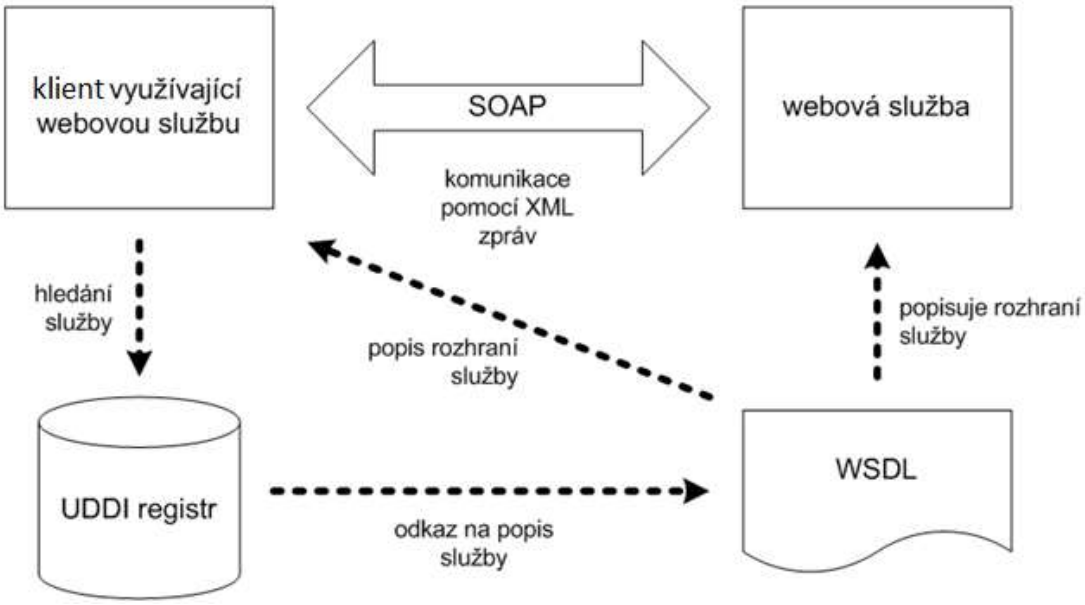
Vzhledem k tomu, že QUIC je úplně nová koncepce, kterou nemá cenu zpětně vsadit do HTTP/2, rozhodla komunita na podzim 2018 o vzniku nové verze protokolu HTTP. Nová verze dostala označení HTTP/3 a QUIC bude jejím základním prvkem. Datum uvedení protokolu HTTP/3 k 10. červnu 2019 nebylo oznámeno. [12, 13]

2.2 SOAP

Následující kapitola se věnuje komunikačnímu protokolu SOAP, což je zkratka pro označení Simple Object Access Protocol. SOAP vychází z principu vzdáleného volání procedur (RPC) a slouží pro zasílání XML zpráv po síti. Společně s technologiemi WSDL a UDDI se jedná o standard webových služeb. [14, 15]

2.2.1 Struktura zprávy

Zpráva v protokolu SOAP je tvořena kořenovým elementem tzv. obálkou (Envelope). Uvnitř obálky se vyskytuje nepovinný element hlavička (Header) a povinný element tělo (Body). Obálka definuje jmenné prostory a povinné atributy používané v rámci celé zprávy. V hlavičce se nachází pomocné informace pro zpracování zprávy např. identifikace nebo autentizace uživatele. Nejdůležitější částí zprávy je její tělo. To zahrnuje samotný obsah zprávy, tedy identifikaci volané služby, její parametry či návratové hodnoty služby. Na následujícím obrázku (obrázek 4) lze vidět jednoduché volání funkce pomocí SOAP služby. [14, 15]



Obrázek 4 – Vztah technologií SOAP, WSDL a UDDI [15]

2.2.2 WSDL

Jedná se o protokol, který slouží k popisu rozhraní webové služby. Vznikl jako společná iniciativa firem IBM a Microsoft. Jak již bylo zmíněno v předchozí kapitole, používá se nejčastěji ve spojení s protokolem SOAP. [14, 15]

WSDL dokument je psán v jazyce XML a jeho struktura je definována pomocí několika základních elementů [14, 15]:

- *Typy (types)* – slouží k popisu datových typů zpráv webové služby. Nejčastěji se využívá XSD (XML Schema Definition), ačkoliv je teoreticky možné použít jakýkoliv typový systém. Kromě základních typů lze definovat i složené typy, případně i rozšíření jiných typů.
- *Zprávy (messages)* – abstraktní definice struktur zpráv. Definují vstup nebo výstup operací.
- *Operace (operation)* – definuje podporované operace a zprávy, které jsou těmto operacím přiřazené.
- *Rozhraní (portType)* – zapouzdřuje dostupné operace.
- *Vazba (binding)* – určuje, jaký protokol a formát zprávy komunikuje s konkrétním rozhraním.
- *Brána (port)* – definuje koncový bod jako kombinaci síťové adresy a vazby.
- *Služba (service)* – zapouzdřuje porty do výsledné služby.

Na obrázku 5 se pro ilustraci nachází ukázka WSDL souboru získaného z modulu webových služeb IS/STAG.

```
▼<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:tns="http://stag-ws.zcu.cz/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:ns1="http://schemas.xmlsoap.org/soap/http" name="ZnamkyServiceImplService" targetNamespace="http://stag-ws.zcu.cz/">
  ▼<wsdl:types>
    ▼<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://stag-ws.zcu.cz/" elementFormDefault="unqualified"
     targetNamespace="http://stag-ws.zcu.cz/" version="1.0">
       <xs:element name="getZkouskovyKatalog" type="tns:getZkouskovyKatalog"/>
       <xs:element name="getZkouskovyKatalogResponse" type="tns:getZkouskovyKatalogResponse"/>
       <xs:element name="getZnamkyByPredmet" type="tns:getZnamkyByPredmet"/>
       <xs:element name="getZnamkyByPredmetResponse" type="tns:getZnamkyByPredmetResponse"/>
       <xs:element name="getZnamkyByRoakce" type="tns:getZnamkyByRoakce"/>
       <xs:element name="getZnamkyByRoakceResponse" type="tns:getZnamkyByRoakceResponse"/>
       <xs:element name="getZnamkyByStudent" type="tns:getZnamkyByStudent"/>
       <xs:element name="getZnamkyByStudentResponse" type="tns:getZnamkyByStudentResponse"/>
       <xs:element name="getZnamkyByTermin" type="tns:getZnamkyByTermin"/>
       <xs:element name="getZnamkyByTerminResponse" type="tns:getZnamkyByTerminResponse"/>
       <xs:element name="hromadne_znamky" type="tns:hromadne_znamkyType"/>
       <xs:element name="uploadZnamek" type="tns:uploadZnamek"/>
       <xs:element name="uploadZnamekResponse" type="tns:uploadZnamekResponse"/>
       <xs:element name="upload_result" type="tns:upload_resultType"/>
      ▼<xs:complexType name="getZkouskovyKatalog">
        ▼<xs:sequence>
           <xs:element minOccurs="0" name="stagUser" nillable="true" type="xs:string"/>
           <xs:element name="pracoviste" type="xs:string"/>
           <xs:element name="zkratka" type="xs:string"/>
           <xs:element minOccurs="0" name="rok" nillable="true" type="xs:string"/>
           <xs:element name="semestr" type="xs:string"/>
           <xs:element minOccurs="0" name="lang" nillable="true" type="xs:string"/>
         </xs:sequence>
       </xs:complexType>
      ▼<xs:complexType name="getZkouskovyKatalogResponse">
        ▼<xs:sequence>
           <xs:element name="katalog" type="xs:base64Binary"/>
         </xs:sequence>
       </xs:complexType>
      ▼<xs:complexType name="getZnamkyByRoakce">
        ▼<xs:sequence>
           <xs:element minOccurs="0" name="stagUser" nillable="true" type="xs:string"/>
           <xs:element minOccurs="0" name="roakIdno" nillable="true" type="xs:long"/>
         </xs:sequence>
       </xs:complexType>
```

Obrázek 5 – Ukázka WSDL souboru [16]

2.2.3 UDDI

UDDI (The Universal Description, Discovery and Integration Service) je mechanismus, který umožňuje registraci a vyhledávání webových služeb. Je možné si ho představit jako velký adresář (registr), kam mohou poskytovatelé webových služeb (firmy) vkládat informace o sobě a jimi poskytovaných službách. Klienti si poté mohou tyto informace vyhledat. Samotné UDDI funguje také jako webová služba, přičemž komunikace probíhá prostřednictvím protokolu SOAP. [14, 15]

2.3 REST

V této kapitole je představen a popsán architektonický styl Representation State Transfer (neboli REST), jenž od svého uvedení v roce 2000 prošel dost zásadními změnami a vývojem. Rozhraní REST se používá jak pro jednotný, tak i snadný přístup ke zdrojům (resources). Za zdroj lze považovat data a případně i stavy aplikace, pokud jsou popsány daty. Na rozdíl od SOAPu (popsán v předešlé kapitole 2.2) je orientován datově, nikoli procedurálně. [17]

REST byl poprvé definován v disertační práci Roye Fieldinga *Architectural Styles and the Design of Network-based Software architectures* publikované v roce 2000. Kromě práce na RESTu Roy Fielding v té době spolupracoval na vytvoření protokolu HTTP/1.1 a URI technologie, což logicky vyústilo v použití těchto technologií i u RESTu. V konečném důsledku je REST vlastně softwarovou architekturou pro hypermediové systémy (např. WWW), jenž komunikuje prostřednictvím propojené sítě za účelem popsat definice a adresace zdrojů. [17, 18]

2.3.1 Zdroje (resources)

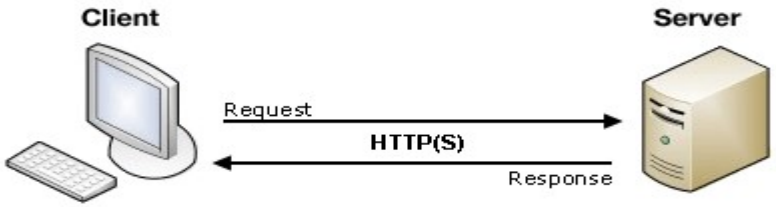
Jedná se o základní kámen architektury. Za zdroj se dá považovat prakticky cokoliv, od objektu v databázi po konkrétní webovou stránku. REST nemá pro zdroje téměř žádné omezení, jediné jeho pravidlo je, že zdroj musí mít přesně specifikované URL (viz identifikace zdroje v kapitole 2.3.2.4). Zdroje je možné seskupovat do kolekcí. Tyto kolekce jsou neuspořádané a homogenní, tedy mohou obsahovat pouze jeden druh zdroje. Samotná kolekce je také zdroj, z čehož vyplývá, že zdroj může obsahovat také další kolekce a zdroje. [18, 19]

2.3.2 Hlavní principy RESTu

Fielding ve své práci nejprve definoval výchozí styl tzv. „Null Style" a poté přidával jednotlivá omezení. Aby mohlo být API označováno jako RESTful, musí splňovat šest základních omezení, která jsou popsána v následujících odstavcích.

2.3.2.1 Klient-server

Hlavním důvodem zavedení tohoto omezení je princip Separation of Concerns (oddělení zodpovědnosti). Ten rozděluje systém na komponenty, kde každá komponenta má svůj účel a minimálně zasahuje do ostatních. V tomto případě se jedná o dvě komponenty: klient a server. Server má definovanou množinu služeb, na které klient zasílá požadavky. Ty server následně zpracuje a vrátí klientovi odpověď. Schéma komunikace je zobrazeno na následujícím obrázku (obrázek 6). [19]



Obrázek 6 – Komunikace klienta se serverem [20]

2.3.2.2 Bezstavovost

Další omezení udává, že komunikace mezi klientem a serverem musí být bezstavová. To znamená, že každý požadavek od klienta musí obsahovat veškeré potřebné informace, aby byl server schopný takový požadavek zpracovat. Všechny informace o stavu má na starosti klient. [19]

Toto omezení sebou přináší množství výhod, konkrétně se jedná o škálovatelnost, viditelnost a spolehlivost. Bezstavovost umožňuje rychle obsluhovat jednotlivé požadavky a uvolňovat zdroje, což vede ke zlepšení škálovatelnosti. Ke zlepšení viditelnosti dochází díky tomu, že monitorovací systém nemusí zkoumat celou historii požadavku, aby zjistil jeho účel. Protože není nutné ukládat stav, je možné v případě chyby aplikaci jednoduše obnovit. Tím se vyznačuje poslední výhoda – spolehlivost aplikace. [19]

Bezstavovost sebou nese i některé nevýhody. Hlavní nevýhodou je celkové snížení výkonnosti systému zasíláním stejných dat stále dokola. Přičemž by data mohla být, při ignorování tohoto omezení, uložena na serveru. [19]

2.3.2.3 Vyrovnávací paměť cache

Následující omezení bylo přidáno za účelem zvýšení výkonu a efektivity sítě. Je nutné u každého požadavku/odpovědi implicitně či explicitně definovat, zda může cache využít, nebo ne. Pokud je možné odpověď uložit do vyrovnávací paměti, tak klient může znovu použít odpověď později při zaslání identického požadavku. [19]

Správné využívání vyrovnávací paměti cache vede v některých případech k úplné, nebo alespoň částečné eliminaci komunikace se serverem. To má za následek zlepšení efektivity, škálovatelnosti a snížení odezvy systému. Na druhou stranu může vést využití cache ke snížení spolehlivosti. Tento problém nastává tehdy, pokud se data v paměti cache výrazně liší od dat, která by v případě přímého požadavku server skutečně vrátil. [19]

2.3.2.4 Jednotné rozhraní

Jednotné rozhraní je nejdůležitějším požadavkem na REST, kterým se odlišuje od ostatních síťových architektur. Rozhraní není závislé na typu aplikace nebo serveru. Je vždy stejné. Fielding ve své práci definoval čtyři základní omezení na rozhraní [18, 19]:

- *Identifikace zdroje* – každý zdroj by měl mít svůj vlastní identifikátor. V RESTové architektuře není nikde definováno, jak identifikovat zdroje, záleží to čistě na autorovi. Jednou z možností je použití URI. Ten jednoznačně určuje, o který zdroj se jedná.
- *Manipulace se zdroji skrze jejich reprezentace* – klient nikdy nevidí zdroj přímo, ale vidí jeho reprezentaci. Reprezentace je sekvence bajtů a metadat, jenž dané bajty popisují (např. JSON, HTML atd.). Jeden zdroj může mít více reprezentací. Pokud tomu tak je, musí mít klient možnost si reprezentaci zvolit.
- *Samopopisující zprávy* – zprávy by měly obsahovat veškeré informace potřebné pro úspěšné zpracování dotazu, resp. odpovědi, bez nutnosti cokoliv dohledávat. Koresponduje to s podmínkou bezstavovosti.
- *HATEOAS (hypermédia)* – je zkratkou pro „*Hypermedia As The Engine Of Application State"*. Hypermédia jsou rozšířením pro hypertext, tedy kromě textu a odkazů přinášejí navíc obrázky či videa. Toto omezení říká, že aktuálně přistupovaný zdroj by měl v odpovědi obsahovat odkazy na všechny související zdroje.

2.3.2.5 Vrstvený systém

Hlavním cílem tohoto omezení je snížit složitost systému jeho rozdělením na jednotlivé komponenty do hierarchických vrstev. Každá komponenta musí být nezávislá a může komunikovat pouze s nejbližší vrstvou. Jiná komunikace mezi vrstvami není dovolena. [19]

Kromě toho umožňuje mezi klienta a server vkládat prostředníky. Prostředník slouží například k vyvažování zátěže (load balancing), prosazování bezpečnostních zásad či zjednodušování komponent serveru tím, že přesune málo používané funkčnosti na sebe. Klient netuší, zda komunikuje se serverem, nebo nějakým prostředníkem. [19]

Vrstvený systém má jeden zásadní nedostatek. Snižuje propustnost v důsledku latence a zvýšené režie při komunikaci mezi vrstvami, což ale může být potlačeno správnou implementací cache. [19]

2.3.2.6 Code-On-Demand

Jediným volitelným omezením stylu REST je tzv. kód na vyžádání. Umožňuje funkcionalitu klienta rozšířit kódem například o JavaScript, Java aplety atd. Díky tomu se část operací, které by vykonával server, přenese na klienta, což má za následek vylepšení odezvy a zjednodušení rozhraní serveru. Na druhou stranu dochází k značnému snížení viditelnosti, a právě z toho důvodu je toto omezení označené jako volitelné. Mělo by se použít pouze tehdy, pokud je to výhodné nebo nutné. [19]

2.3.3 CRUD operace

REST implementuje čtyři základní operace pro manipulaci se zdroji, jinak také známé pod označením CRUD. Název CRUD je zkratkou pro operace Create, Read, Update a Delete. Následující tabulka (tabulka 2) porovnává vztah těchto operací a HTTP metod spolu s krátkým popisem. [17, 18]

Tabulka 2 – Mapování HTTP metod na základní CRUD operace

HTTP metoda	CRUD operace	Popis
POST	Create	Vytvoření nového zdroje
GET	Read	Získání zdroje
PUT	Update	Úprava existujícího zdroje
DELETE	Delete	Odstranění zdroje

2.4 GraphQL

Jedná se o deklarativní dotazovací jazyk vyvinutý Facebookem v roce 2012, ale oficiálně byl vydán až v roce 2015. Poskytuje alternativu k architektuře REST. Hlavním cílem GraphQL je zvýšit produktivitu vývojářů a minimalizovat množství datových přenosů. Aktivně používají tuto technologii již stovky společností např. Facebook, GitHub, Paypal a jiné. [21]

GraphQL není vázán na žádnou konkrétní technologii pro uložení dat a lze jej implementovat nad jakýmkoliv stávajícím systémem. Je podporován v mnoha různých programovacích

jazycích, např. se jedná o C#, Groovy, PHP, Python, Java, Scala, Elixir, Ruby, Erlang, JavaScript a Go. [21]

Stejně jako pro REST jsou základním kamenem zdroje (resources), pro GraphQL je to tzv. schéma. Schéma je založeno na SDL (Schema Definition Language). Základním prvkem schématu jsou typy a jejich vzájemné vztahy. Existuje pět základních skalárních typů, a to ID, Int, Float, String a Boolean. Případné další typy je možné si dodefinovat pomocí klíčového slova type. [21, 22]

V GraphQL lze s daty pracovat třemi různými způsoby [21]:

- dotazovat se na data – *query* (viz zdrojový kód 1),
- manipulavat s daty – *mutation*,
- reagovat na specifickou událost v reálném čase – *subscription*.

```
{                                    {
  user(id: 4) {                        "user": {
    name                                 "name": "Mark Zuckerberg"
  }                                    }
}                                    }
```

Zdrojový kód 1 – Příklad požadavku (vlevo) a odpovědi (vpravo) v GraphQL [23]

2.4.1 Principy designu

V následujících bodech se nachází popis základních principů GraphQL definovaných společností Facebook. Kompletní specifikace je dostupná v GitHub repozitáři. [23]

- *Hierarchický* – pro dosažení maximální shody se strukturou moderních aplikací je GraphQL dotaz hierarchický. Dotaz má stejný tvar jako odpověď, kterou vrací. To umožňuje klientům specifikovat podobu dat, se kterou budou pak dále pracovat.
- *Produktově orientovaný* – požadavky na data nejsou specifikovány globálně serverem, ale jsou řízeny potřebami klientů.
- *Silně typový* – GraphQL definuje typový systém specifický pro aplikaci. Dotazy jsou poté vykonávány v kontextu tohoto typového systému. Systém obsahuje nástroje pro ověření syntaktické správnosti a platnosti dotazu ještě před jeho spuštěním, čímž lze značně ušetřit výpočetní výkon.

- *Klientem specifikované dotazy* – GraphQL poskytuje klientům možnosti skrze typový server. Klient je zodpovědný za specifikaci toho, jak danou možnost zkonzumovat. Platí zde, že klient dostane přesně to, na co se dotazuje, nic víc, nic míň.
- *Introspektivní* – typový systém musí být dotazovatelný pomocí GraphQL dotazů. Díky tomu může vývojář prozkoumávat API, aniž by musel pročítat kód nebo dokumentaci.

2.5 Shrnutí

V předcházejících kapitolách byly představeny tři způsoby tvorby webových API. Jedná se o protokol SOAP, architekturu REST a dotazovací jazyk GraphQL. Již z jejich představení je patrné, že se mezi nimi objevují značné rozdíly. Na jejich základě je možné definovat výhody či nevýhody, což je obsahem následující kapitoly.

SOAP

- Výhody:
  - vysoká bezpečnost,
  - standardizovaný, rozšiřitelný,
  - nezávislý na komunikačním protokolu.
- Nevýhody:
  - nepodporuje HTTP caching,
  - podpora pouze formátu XML,
  - pomalejší.

REST

- Výhody:
  - podpora různých formátů dat např. XML či JSON,
  - podporuje HTTP caching – redukuje počet odeslaných a získaných dat,
  - rychlá křivka učení.
- Nevýhody:
  - overfetching – API vrací více informací, než klient chce,
  - underfetching – API nevrací všechny informace, které klient požaduje[2],
  - slabé typování dat, může způsobit komunikační problémy mezi aplikacemi.

[2] Pro získání dodatečných informací musí klient odeslat další separátní požadavky.

GraphQL

- Výhody:
  - získání všech potřebných dat v jednom dotaze, čímž ulehčuje práci klientovi,
  - silně typový – automatická validace a kontrola typu,
  - introspektivní – automatické generování API dokumentace.
- Nevýhody:
  - při odpovědi je možné vrátit pouze stavový kód 200, takže není možné se zpracováním chybových stavů spoléhat na stavové kódy,
  - nepodporuje HTTP caching,
  - nepodporuje verzování.

Následující tabulka (tabulka 3) obsahuje shrnutí informací o jednotlivých webových API probraných v předešlých podkapitolách.

Tabulka 3 – Shrnutí probraných webových služeb [24]

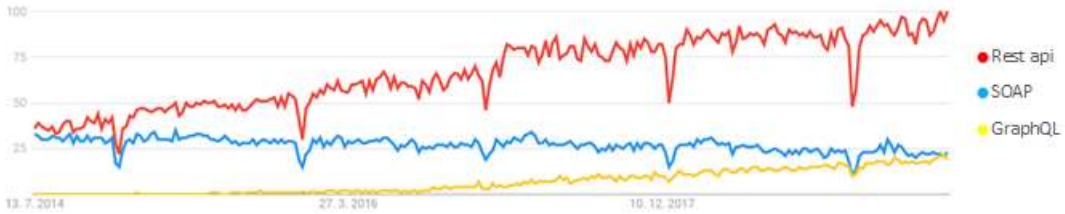
	SOAP	***REST***	***GraphQL***
Rok vytvoření	1998	2000	2015
Typ	Protokol	Architektonický styl	Dotazovací jazyk
Povolené formáty dat	XML	plain text, XML, HTML, JSON, …	JSON
Komunikační protokol	TCP, HTTP, SMTP	HTTP	HTTP
Typování dat	Silné	Slabé	Silné
Definice dat	Server	Server	Klient
Koncový bod (endpoint)[3]	Jeden	Několik	Jeden
Caching	Ne	Ano	Ne
Verzování	Ano	Ano	Ne

Který návrh pro tvorbu API je tedy nejlepší? Na tuhle otázku nelze najít jednoznačnou odpověď. Neexistuje žádné univerzální řešení, protože každá aplikace je unikátní a výběr by se měl odvíjet od toho, co je nejlepší pro daný projekt nebo co zákazník vyžaduje. Případně je

[3] Endpoint – jedná se o bod, skrze který se komunikuje s webovou službou.

možné použít jejich kombinaci. Existují však doporučení, kterých je dobré se držet. SOAP se používal, používá a bude používat ve státní sféře nebo pro firemní řešení, kde je důležitá bezpečnost (např. v bankách, při odesílání tržeb v rámci EET). GraphQL se hodně používá při komunikaci mezi frontendem a backendem díky tomu, že dokáže zredukovat počet potřebných dotazů na minimum. REST je potom využíván ve zbývajících případech například pro komunikaci mezi projekty. [24]

Na následujícím obrázku (obrázek 7) je k dispozici statistika vyhledávání vybraných termínů na portálu Google.com. Jak je z obrázku vidět, popularita GraphQL od jeho uvedení pozvolna stoupá, ale nejpopulárnější je stále REST. Naproti tomu popularita SOAPu postupně klesá.



Obrázek 7 – Graf trendů pro vyhledávání vybraných termínů na Google.com [25]

3 WEBOVÉ TECHNOLOGIE

V následující kapitole je uvedena rešerše čtyř vybraných technologií, které je možné používat v kombinaci s webovými službami popsanými v předcházející kapitole. Konkrétně byly vybrány tyto: Spring Framework, ASP.NET, React a Angular.

Technologie byly zvoleny na základě výzkumu provedeného v letech 2018 a 2019 webem Stack Overflow. Stack Overflow je jedním z nejpopulárnějších webů pro vývojáře, kteří zde sdílejí své problémy a jejich řešení v desítkách programovacích jazycích. [26]

Cílem této kapitoly není popsat přesné fungování vybraných technologií, ale jejich představení. Pro detailnější pochopení je doporučeno navštívit oficiální dokumentace.

3.1 Spring Framework

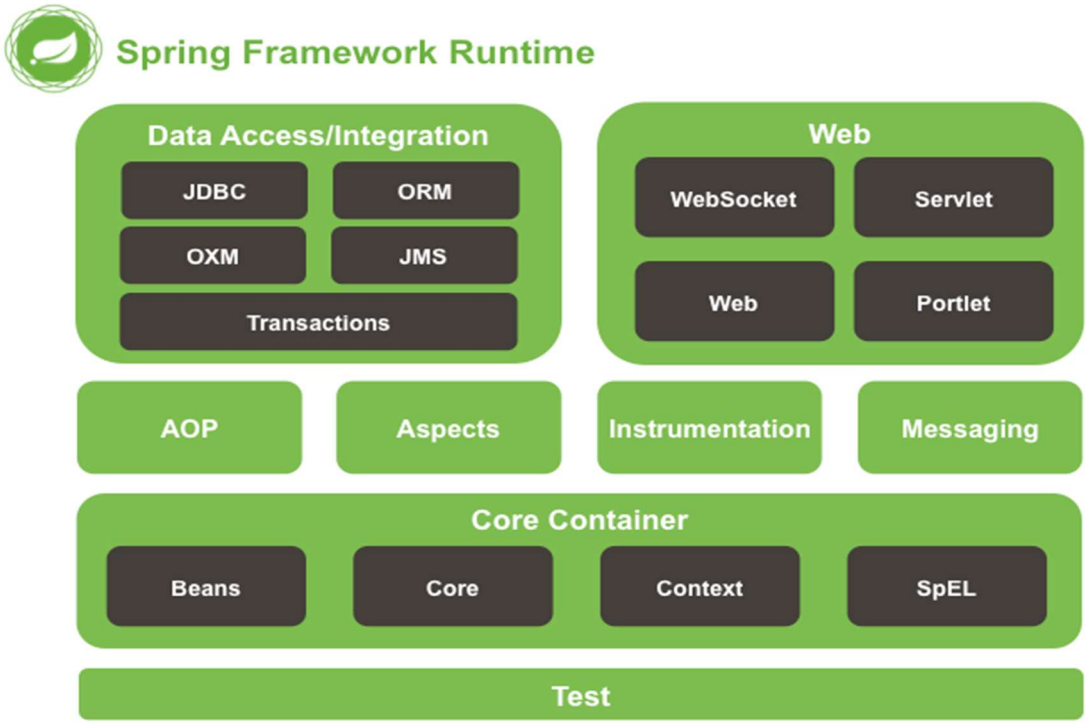
Jedná se o open-source framework pro vývoj Java EE (Enterprise Edition) aplikací. Jeho velkou předností je nízká provázanost jednotlivých částí a díky tomu zvýšená modulárnost, takže je možné v každé aplikaci využít jen tu část frameworku, kterou opravdu potřebuje. Vzhledem k tomu se jedná o řešení vhodné nejen pro rozsáhlé podnikové aplikace, ale i pro aplikace jednoduché. [27]

Spring vznikl na základě publikace *Expert One-on-One J2EE Design and Development* od Roda Johnsona. Napříč knihou je představen nový framework nazývaný Interface21, který měl usnadnit, zpřehlednit a také zjednodušit vývoj J2EE aplikací. Díky spolupráci se svými kolegy Juergenem Hoellerem a Yannem Caroffem později kód frameworku rozšířil a v červnu 2003 byl poprvé představen pod svým stávajícím názvem Spring Framework. [28]

Jak již bylo řečeno v úvodu kapitoly, Spring je modulární. Skládá se zhruba z dvaceti modulů, které jsou uspořádány do několika skupin, nejdůležitější jsou vypsány níže [27]:

- *Core Container* – obsahuje základní části frameworku jako IoC, DI nebo aplikační kontext.
- *Data Access/Integration* – moduly pro práci s daty.
- *Web* – obsahuje různé funkce pro tvorbu webových aplikací např. implementace návrhového vzoru MVC nebo REST služeb.
- *AOP a Aspects* – podpora aspektově orientovaného programování.
- *Test* – modul podporující testování jednotkovými a integračními testy za pomoci JUnit nebo TestNG.

Přehlednější znázornění jednotlivých skupin s jejich moduly je zobrazeno na obrázku 8.



Obrázek 8 – Architektura Spring Frameworku [27]

Spring IoC kontejner

Inversion of Control (IoC) kontejner je srdcem Spring Frameworku. Je zaváděn při spuštění aplikace a reprezentován třídou *ApplicationContext*. V celé aplikaci se nachází pouze jeden. IoC kontejner se stará o správu celého životního cyklu objektů, od vytvoření až po jejich zničení. Kromě toho řídí i jejich konfiguraci či provázanost. Tyto objekty se nazývají beany a tvoří funkční jádro aplikace. K jejich provázání se používá Dependency injection (DI). [29]

Dependency injection

Dependency injection je konkrétní implementací návrhového vzoru IoC[4]. Jedná se o mechanismus, který vkládá (injektuje) do třídy instanci jiné třídy. Vkládání má na starost přímo Spring Framework a řídí se podle vybrané konfigurace. [29]

3.2 ASP.NET

ASP.NET je webový framework vyvíjený společností Microsoft, který se používá k vytváření webových aplikací a služeb. Název ASP.NET je odvozen od starší technologie ASP (Active Server Pages) a platformy .NET. [30, 31]

[4] Inversion of Control – jedná se o návrhový vzor, ve kterém je řízení objektů z aplikace přesunuto na framework.

.NET je tedy vývojářská platforma, pomocí které lze vyvíjet nejen klasické aplikace pro Windows, webové aplikace a služby, ale i aplikace pro mobilní zařízení. Skládá se z nástrojů, knihoven a programovacích jazyků (C #, F # nebo Visual Basic). Kromě toho obsahuje i běhové prostředí CLR (Common Language Runtime), které zajišťuje provoz a kompilaci aplikací. V roce 2000 vyšla zkušební verze .NET Frameworku 1.0, která byla v roce 2002 i první plnou verzí. Přelomovou novinkou vydanou v roce 2016 byl .NET Core. [30-32]

ASP.NET Core je open-source webový framework. Používá se k vývoji moderních webových aplikací a služeb na platformě .NET Core. Vznikl kompletním přepsáním staršího ASP.NET Frameworku. Původně byl označován jako ASP.NET 5, ale aby to pro vývojáře nebylo matoucí, rozhodl se Microsoft pro přejmenování na ASP.NET Core. Na rozdíl od svého předchůdce je již od počátku multiplatformní a podporuje operační systémy Windows, macOS a Linux. ASP.NET Core aplikace mohou být spuštěny jak na .NET Core, tak i na starším .NET Frameworku. Práce s frameworkem ASP.NET Core se zakládá na využití návrhového vzoru MVC. Jednou z jeho klíčových vlastností je modulárnost, o kterou se stará mechanismus pro sdílení kódu NuGet. Pomocí NuGet balíčků dochází k distribuci po menších částech, které jsou zaměřené na jednotlivá rozšíření a funkce. Minimalizuje aplikace a umožňuje jejich optimalizaci tak, aby obsahovaly jen ty části, které jsou potřebné. [30]

Microsoft zatím stále pracuje jak na novém .NET Core, tak i na .NET Frameworku. Na podzim letošního roku vyjde verze .NET Core 3.0, která bude naposledy používat název .NET Core. Do budoucna totiž Microsoft plánuje sjednocení .NET Frameworku, .NET Core a Mono pod jednu platformu označenou .NET 5 (viz obrázek 9). Ta vyjde v roce 2020 a bude pokračovat ve směru udávaném .NET Core. [33, 34]

Obrázek 9 – Budoucnost .NET platformy [33]

3.3 React

React je JavaScriptová open-source knihovna, která se používá pro tvorbu uživatelského prostředí (UI), vyvíjena společností Facebook. Kromě Facebooku se na její správě podílí i komunita vývojářů. Využívá se primárně k tvorbě tzv. single-page aplikací (SPA), dále k vykreslování webových stránek na straně serveru (použití v kombinaci s jinou knihovnou např. Next.js v serverovém prostředí Node.js[5]) nebo k tvorbě mobilních aplikací pomocí React Native. V rámci MVC architektury bývá velmi často označován jako „V“ tedy View. [35-37]

React byl vytvořen roku 2011 softwarovým inženýrem Jordanem Walkem ve společnosti Facebook. Jordan Walk nejprve vydal prototyp Reactu nazvaný FaxJS, který byl silně ovlivněn technologií XHP[6]. Z počátku byl nasazen v kanále vybraných příspěvků („News Feed”) na Facebooku a poté byl použit i na Instagramu. V květnu roku 2013 byl prohlášen Facebookem, už s názvem React, za open-source na JSConf US[7]. [36, 38]

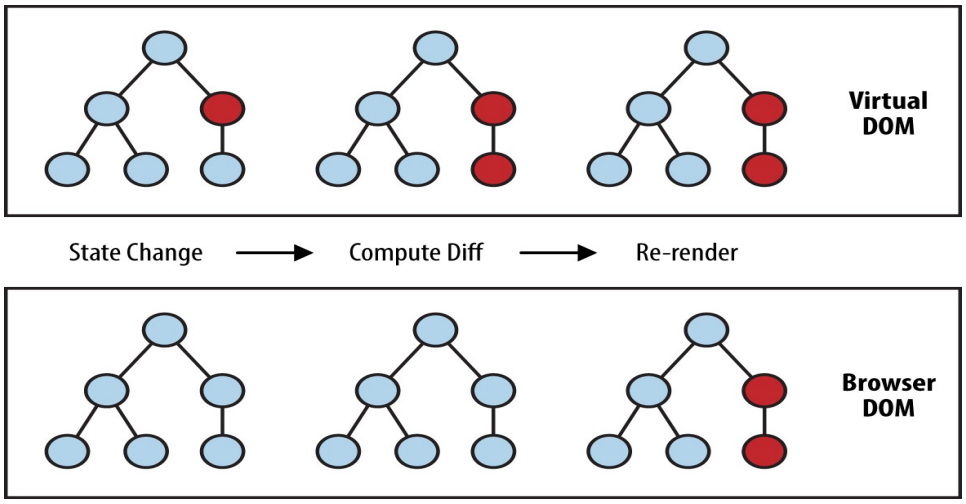
React je založen na komponentách, které umožňují rozdělit UI do samostatných, dobře udržitelných a znovupoužitelných částí. Komponenty jsou koncepčně jako funkce JavaScriptu, které přijímají libovolné vstupy (tzv. props) a vracejí prvky Reactu popisující, co se má zobrazit na výstupu (obrazovce). Celá aplikace je pak tvořena z těchto komponent poskládaných do stromové struktury. V případě, že dojde ke změně v komponentě, tak se zbytečně nepřekresluje celá stránka, ale pouze daná komponenta, případně její podstrom. [37]

Použití Reactu sebou přináší určitá vylepšení pro tvorbu klientských aplikací. Jedním z nich je rozšíření syntaxe JavaScriptu o tzv. JSX. JSX umožňuje využívat HTML značky uvnitř JavaScriptového kódu. React jeho použití nevyžaduje, je volitelné. Stále je možné používat standardní JavaScript, ale zápis pomocí JSX je mnohem jednodušší a přehlednější. [37]

Dříve většina JavaScriptových frameworků při změně elementu v DOMu[8] překreslila celou komponentu, ve které se vyskytovala. Například při změně jednoho elementu z listu byl překreslen celý list, což bylo značně neefektivní. React řeší tento problém využitím konceptu tzv. virtuálního DOMu (viz obrázek 10). Jedná se o virtuální kopii reálného DOMu uloženou v paměti. V případě změny v datech je vytvořen další virtuální dokument a proběhne porovnání

[5] Next.js je framework Reactu. Node.js je prostředí pro spouštění JavaScriptového kódu mimo webový prohlížeč.
[6] XHP – PHP rozšíření, jenž umožňuje zapisovat XML kód přímo do PHP.
[7] JSConf Us je zkratkou pro Conferences for the JavaScript community.
[8] Document Object Model – stromová struktura elementů, ze kterých je složena webová stránka.

s předešlým (před změnou). React poté zjistí na základě onoho porovnání, které změny musí promítnout do reálného DOMu. Překreslují se tedy jen ty nejnutnější změny. [39]



Obrázek 10 – Virtual DOM [39]

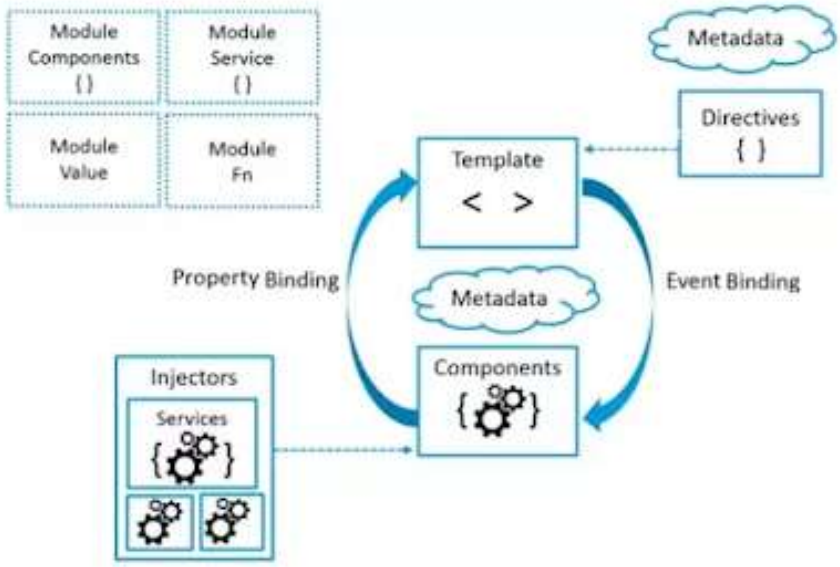
3.4 Angular

Jedná se o open-source platformu a framework pro tvorbu klientských single-page aplikací v HTML a TypeScriptu (samotný Angular je v něm také napsán). Angular udržuje a vyvíjí skupina vývojářů a společností v čele s firmou Google. Veškerá funkcionalita Angularu je implementována jako sada knihoven, které se importují do aplikací. [40]

Původně vznikl roku 2010 pod názvem Angular 1.x. Postupem času přestával Angular stačit novým trendům ve vývoji, což spolu s nástupem nových technologií, jako např. React, vedlo Google ke kompletnímu přepracování Angularu. Nově vzniklý framework dostal označení Angular 2 a není zpětně kompatibilní s původní verzí. [40]

Protože docházelo velmi často k záměně, byl původní Angular 1.x. přejmenován na AngularJS a nová verze si ponechala označení Angular. Nové velké verze jsou vydávány každých šest měsíců. Aktuální verze v době psaní této práce nese označení Angular 8. Pokud se v práci nachází označení Angular, tak je tím myšlena verze Angular 2+. [40]

Nejdůležitější části architektury Angularu jsou moduly, komponenty a služby, jejichž znázornění lze vidět na následující straně (viz obrázek 11). Kromě toho je pod obrázkem uveden i jejich stručný popis.



Obrázek 11 – Architektura Angular aplikací [41]

Moduly

Základním stavebním kamenem Angularu jsou moduly (*NgModules*), které poskytují kontext pro kompilaci komponent. Modul lze chápat jako kontejner pro seskupení souvisejících částí aplikace. Může obsahovat komponenty, služby nebo další soubory. V každé aplikaci se nachází alespoň jeden modul, tzv. kořenový modul, který se podle konvence nazývá *AppModule*. Ačkoliv pro jednoduché aplikace může stačit pouze jeden modul, tak většina aplikací má funkcionalitu rozdělenou do více modulů. Každý modul je označen dekorátorem *@NgModule()*. [40]

Služby (service)

Pro práci s daty a logikou aplikace, kterou je možné sdílet mezi více komponent a zároveň není asociována s konkrétní šablonou, se využívají služby. Jedná se o klasické třídy označené dekorátorem *@Injectable()*, které je možné poskytovat komponentám pomocí principu vkládání závislostí (Dependency Injection). Používají se například pro načítání dat skrze API. [40]

Komponenty

Stejně jako v případě Reactu je i Angular založen na stromové struktuře komponent. Každá aplikace obsahuje alespoň jednu komponentu. Ta zodpovídá za uživatelské rozhraní aplikace. Obsahuje aplikační logiku a data spolu s HTML šablonou a příslušnými CSS styly. Pro komponenty se používá dekorátor *@Component()*. [40]

3.5 Porovnání

Cílem této kapitoly bylo představit webové technologie, které je možné používat v kombinaci s webovými službami. Byly zvoleny následující technologie: Spring Framework, ASP.NET, React a Angular. V této části následuje jejich porovnání (viz tabulka 4) s ohledem na to, zda danou webovou službu podporují nativně, pomocí knihoven třetích stran nebo vůbec nepodporují. Informace pro porovnání byly získány z oficiálních dokumentací technologií.

Tabulka 4 – Porovnání vybraných technologií s využitelností webových API

Technologie	***SOAP***	***REST***	***GraphQL***
Angular	Ano – pomocí knihoven (např. ngx-soap)	Ano	Ano – pomocí klienta (Apollo)
React	Ano – pomocí knihoven (např. ngx-soap)	Ano (vlastnost jazyka)	Ano – pomocí klienta (Apollo) nebo frameworku Relay
ASP. NET	Ano	Ano	Ano – pomocí knihovny (graphql-dotnet)
Spring	Ano	Ano	Ano – pomocí knihovny (graphql-java)

Z výše uvedeného porovnání je vidět, že za určitých podmínek lze ve vybraných technologiích využít všechny uvedené způsoby pro tvorbu webových API. Jako nejlepší volba vychází použití technologie REST, protože je nativně podporována ve většině moderních frameworků. Kromě toho, jak již bylo zmíněno v minulé kapitole (viz kapitola 2.5), je REST zatím stále nejoblíbenější technologie pro tvorbu API.

4 POUŽITÉ TECHNOLOGIE PŘI VÝVOJI A NÁVRHU

Jelikož je cílem práce vytvořit komplexní systém, je nutné si na počátku vývoje vybrat vhodné technologie a nástroje k jeho implementaci. Následující kapitola slouží ke stručnému popisu těchto technologií. Pro podrobnější informace je doporučeno navštívit oficiální dokumentace nebo odbornou literaturu.

Spring Framework a Angular, tedy technologie, které jsou stěžejní pro vyvíjené aplikace, jsou popsané v předcházející části (viz kapitoly 3.1, 3.4).

4.1 Android

Android je open-source platforma určená primárně pro mobilní zařízení, jako jsou chytré telefony, tablety a navigace. Tím jeho možnosti však zdaleka nekončí a v současné době se s ním můžeme setkat i u řady dalších produktů (např. televize, hodinky, auta a jiné). [42, 43]

Android je postavený na linuxovém jádře a vyvíjí ho organizace Open Handset Alliance[9], pod kterou spadá např. Google, HTC, Intel, Samsung a mnohé další. Jde tedy o operační systém podporovaný více platformami, což nese i nevýhodu, a to, že chybí optimalizace systému na konkrétní platformu. Největší výhodou a zároveň i nevýhodou této platformy je otevřenost a možnost úprav ze strany výrobců, a dokonce i uživatelů. Na rozdíl od jiných platforem nejsou vydávány aktualizace operačního systému centrálně Googlem, ale výrobci konkrétních zařízení. To znamená, že se u různých zařízení můžete setkat s různými verzemi systému. [42, 43]

4.1.1 Historie

V roce 2003 vznikla společnost Android Inc., kterou založil Andy Rubin, Rich Miner, Nick Sears a Chris White v kalifornském Palo Alto. V roce 2005 tuto společnost odkoupil Google a v roce 2007 byla představena první verze Androidu spolu se založením organizace Open Handset Alliance, která jej vyvíjí dodnes. Zároveň byl vydán veřejnosti i první vývojářský kit (SDK – Software Development Kit). Na podzim v roce 2008 byl v USA vyvinut první chytrý telefon HTC Dream (G1) s operačním systémem Android 1.0. Již v únoru 2009 byla vydána aktualizace Android 1.1 pro G1, do které byla přidána podpora pro placené aplikace v obchodě Android Market nebo pokus o první hlasové vyhledávání. V dubnu 2009 vyšel první update s oficiálním názvem Android 1.5 Cupcake. [42-44]

[9] Open Handset Alliance – sdružení firem z oblasti hardwaru, softwaru a telekomunikace, jejichž cíl je zlepšovat technologie pro mobilní zařízení.

Každá jednotlivá verze operačního systému Android má číselné a kódové označení. Kódové označení je názvem zákusku s abecedním pořadím. Cílem práce není popsat jednotlivé verze Androidu, ale pro stručný přehled slouží následující obrázek (obrázek 12), který obsahuje všechny doposud vydané verze Androidu. [42, 43]

Obrázek 12 – Verze Androidu [45]

4.2 Spring Boot

Jedná se o řešení založené na Spring Frameworku určené pro tvorbu webových aplikací. Neduhem Spring Frameworku při vývoji aplikací menších rozsahů je velké množství konfigurace, které se musí před startem aplikace nastavit. Naproti tomu Spring Boot v této oblasti exceluje, protože má vše již předkonfigurováno. Nutné je pouze, před prvním spuštěním projektu, použít anotaci *@SpringBootApplication* nad hlavní spouštěcí třídou (viz zdrojový kód 2). V případě potřeby lze jakékoliv nastavení jednoduše upravit či přenastavit. [46]

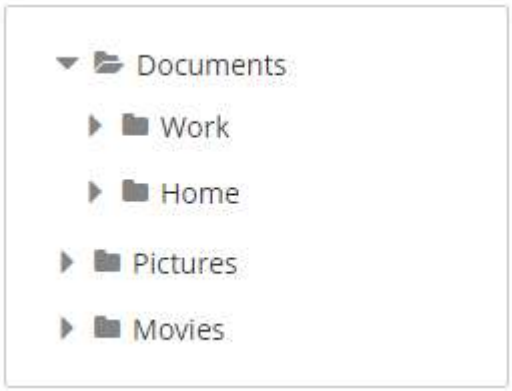
```
@SpringBootApplication
public class AirticketsServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(AirticketsServerApplication.class, args);
    }
}
```

Zdrojový kód 2 – Spuštění Spring Boot aplikace

Ve výchozím nastavení používá Spring Boot jako aplikační server Tomcat. Ten je možné případně nahradit serverem Jetty nebo Undertow. [47]

4.3 PrimeNG

PrimeNG je bohatá kolekce UI komponent pro jazyk Angular 2+ vyvíjená společností PrimeTek Informatics. Celkově se v kolekci nyní nachází již přes 80 komponent, mimo jiné např. tabulky, modální dialogy, kalendáře či různé grafy. Pro vývojáře se jedná o výrazné usnadnění práce, protože si nemusí takové komponenty psát sám, pouze naimportuje kýžený modul a začne ho používat. Dalším plusem je responzivita komponent. Na obrázku 13 je ukázána komponenta Tree, která slouží k zobrazení hierarchických dat. [48, 49]



Obrázek 13 – Komponenta Tree z PrimeNG knihovny [49]

Veškeré zdrojové kódy ke komponentám jsou dostupné z GitHub repozitáře projektu. Kromě toho má PrimeNG na svých stránkách i přehlednou dokumentaci s ukázkami, jak dané komponenty použít. [48]

4.4 MySQL

Jako datové úložiště byla zvolena relační databáze MySQL, původně vytvořena švédskou firmou MySQL AB, nyní vlastněna společností Oracle Corporation. MySQL je multiplatformní systém napsaný v programovacích jazycích C a C++. V dnešní době patří mezi nejpopulárnější relační databáze, což dokládá i fakt, že je používán velkými firmami jako např. Facebook, Twitter nebo YouTube. [50]

Pro komunikaci s databází se stejně jako u většiny ostatních relačních databází používá dialekt jazyka SQL. Případně lze využít i některý z nástrojů pro grafickou nadstavbu z důvodu větší přehlednosti a jednodušší správy systému pro jejího administrátora. Takových nástrojů existuje hned několik např. DataGrip, phpMyAdmin, MySQL Workbench atd. [50]

MySQL poskytuje dva typy licencování. Bezplatnou licenci GPL (General Public License) nebo placenou komerční licenci. Aktuální podporovaná verze 8.0.11 byla vydaná 19. října 2018. Tato verze je spolu s nástrojem MySQL Workbench použita i ve vyvíjené aplikaci. [50]

4.5 Hibernate

Jedná se o volně dostupný framework, který slouží pro objektově-relační mapování (ORM) tříd jazyka Java na tabulky v relační databázi. Tím značně usnadňuje práci programátorům, protože nemusí transformovat objekty do relací ručně. Mapovat objekty je možné dvěma způsoby, a to pomocí anotací ve třídách objektů nebo speciálních mapovacích souborů (jde o XML soubory s koncovkou hbm.xml např. Student.hbm.xml). Hibernate je jednou z možných implementací Java Persistence API (JPA). Lze ho používat přímo voláním jeho metod, nebo obecně přes rozhraní definované pomocí JPA. [51, 52]

Součástí frameworku je i objektově orientovaný jazyk Hibernate Query Language (HQL). Ten je inspirován dotazovacím jazykem SQL, ale místo tabulek a sloupců pracuje s perzistentními objekty a jejich atributy. To je výhodou pro programátory, kteří díky tomu nepotřebují znát přesnou syntaxi používaného databázového systému. HQL je tedy databázově nezávislý. [51, 52]

4.6 JSON

JSON (JavaScript Object Notation) je jazykově nezávislý, jednoduchý a snadno čitelný textový formát pro výměnu dat. Je založený na dvou univerzálních strukturách, které je možné převézt téměř do všech programovacích jazyků. Jedná se o [53, 54]:

- neuspořádanou množinu párů klíč-hodnota (objekt),
    - uvozena složenými závorkami,
    - jednotlivé páry jsou oddělené čárkami,
- seřazenou kolekci hodnot (pole),
    - uvozena hranatými závorkami,
    - jednotlivé hodnoty jsou oddělené čárkami.

V dnešní době se hojně používá i jako formát pro konfigurační soubory. Alternativou je například XML, nicméně JSON je úspornější (velkou část XML zabírají tagy a jejich atributy) a lépe čitelný. Na následujícím zdrojovém kódu (zdrojový kód 3) je k vidění rozdíl v zápise jednoduchého objektu v JSONu a v XML. [53, 54]

```
{"uzivatele": [
    {"jmeno":"John","prijmeni":"Doe"},
    {"jmeno":"Bryany","prijmeni":"Sharman"},
    {"jmeno":"Ann","prijmeni":"Warner"}
]}
```

```
<uzivatele>
    <uzivatel>
        <jmeno>John</jmeno>
        <prijmeni>Doe</prijmeni>
    </uzivatel>
    <uzivatel>
        <jmeno>Bryany</jmeno>
        <prijmeni>Sharman</prijmeni>
    </uzivatel>
    <uzivatel>
        <jmeno>Ann</jmeno>
        <prijmeni>Warner</prijmeni>
    </uzivatel>
</uzivatele>
```

Zdrojový kód 3 – Zápis jednoduchého objektu pomocí JSONu (vlevo) a XML (vpravo)

4.7 Softwarové nástroje

4.7.1 Maven

Apache Maven slouží jako nástroj pro správu závislostí a sestavení (build) projektů postavených především na jazyce Java. Hlavním důvodem vytvoření Mavenu byla snaha o zkrácení doby potřebné k vývoji aplikace, přičemž k jejímu naplnění si tvůrci definovali několik dílčích cílů [55]:

- zjednodušit proces sestavení aplikace,
- stanovení jednotného sestavujícího systému,
- poskytovat dostatečně kvalitní informace o projektu,
- poskytnout tzv. *best practices* (dobré, osvědčené postupy vývoje softwaru),
- povolit transparentní přidávání nových rozšíření a funkcí.

Maven využívá konceptu výchozích nastavení, která není třeba dále konfigurovat, také známé jako konvence místo konfigurace. V případě potřeby je samozřejmě možné konfiguraci změnit.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
    <version>2.1.6.RELEASE</version>
</dependency>
```

Zdrojový kód 4 – Ukázka závislosti v souboru pom.xml

Veškeré informace o závislostech (viz zdrojový kód 4) se nachází v souboru pom.xml (POM). Jedná se o XML soubor, který definuje strukturu a vlastnosti vytvářeného projektu. Pokud není

explicitně definováno jinak, dědí každý POM z tzv. SuperPOMu. Ten obsahuje výše zmíněné výchozí nastavení projektu, jako je např. adresářová struktura. [55]

4.7.2 NPM

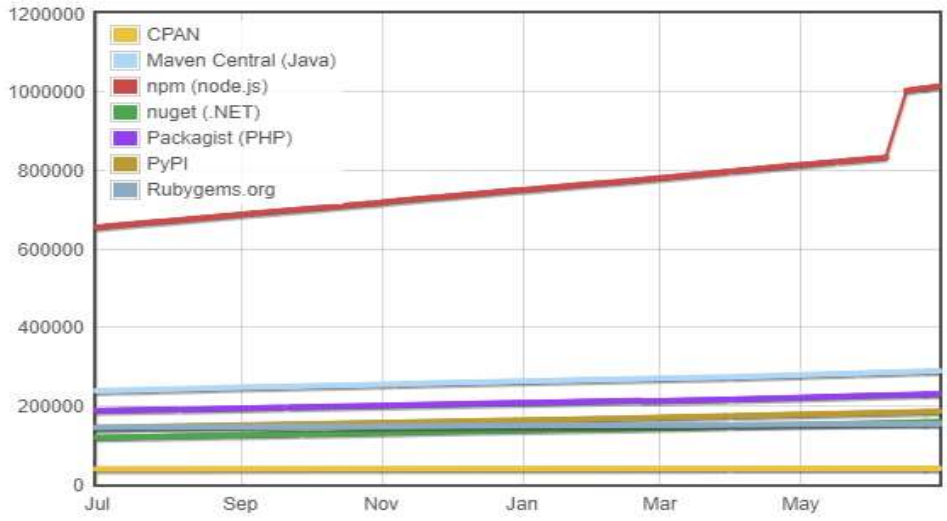
Jedná se o správce JavaScriptových balíčků v aplikaci. Slouží k jejich instalaci a následné údržbě. Kromě toho je možné pomocí NPM (Node package manager) sdílet a distribuovat i kód vlastní. V podstatě se jedná o obdobu Mavenu (popsaný v předcházející kapitole 4.7.1) či Composeru (používaný v PHP aplikacích). [56, 57]

NPM se instaluje společně s Node.js, jinak ho nelze samostatně používat. Node.js je běhové prostředí pro vykonání skriptů, jehož jádro tvoří otevřený JavaScriptový compiler V8 od společnosti Google. [56, 57]

Struktura všech projektů, které používají NPM, musí obsahovat minimálně následující části [56, 57]:

- *node_modules* – složka, kam se instalují balíčky. Jako jediná se z důvodu šetření místa nenahrává do verzovacího nástroje Git.
- *package.json* – hlavní konfigurační soubor uložený ve formátu JSON. Obsahuje seznam potřebných balíčků a meta informace o projektu. Kromě názvu a verze není žádná položka povinná.
- *package-lock.json* – obsahuje informace o instalovaných verzích balíčků.

Jak je vidět na následujícím obrázku (obrázek 14), NPM má k dispozici největší kolekci balíčků – zhruba milion.



Obrázek 14 – Počet balíčků pro vybrané balíčkovací systémy v posledním roce [58]

4.7.3 Gradle

Standardně se pro sestavování (build) mobilních aplikací na platformě Android nevyužívá Maven, ale podobný nástroj, jenž se nazývá Gradle. Je to modernější varianta nástrojů Ant a Maven, přičemž kombinuje nejlepší z obou plus přidává něco vlastního. [59]

4.7.4 Git

Git je volně dostupný verzovací systém používaný pro přehlednou a jednoduchou správu projektů všech rozsahů, od malých spravovaných jednotlivci až po velké komerční aplikace. Byl vyvinut v roce 2005 Linusem Torvaldsem, který je známý také tím, že vyvinul jádro pro operační systém Linux. Mezi hlavní funkce Gitu patří možnost sledovat celou historii projektu, tedy sledovat všechny prováděné změny v souborech. Díky tomu lze zjistit, kdo a kdy udělal jakou úpravu v projektu. Kromě toho je Git používán kvůli dobrému výkonu, bezpečnosti a flexibilitě. Výhodou používání tohoto systému je i propojení s mnoha softwarovými nástroji a službami jako např. s IDE (vývojové prostředí) nebo s GitHubem. [60-62]

Veškeré změny v projektu ukládá Git pouze lokálně na počítač uživatele, což není dostačující při práci z více zařízení nebo pokud se na projektu podílí více programátorů. Je nutné zajistit synchronizaci a sdílení daného projektu. K tomu účelu existuje hned několik nástrojů jako např. GitHub, GitLab či Bitbucket. V rámci tohoto projektu byl použitý GitHub, kde jsou vytvořeny tři soukromé repozitáře obsahující zdrojové kódy k aplikacím. [60, 62]

4.7.5 Enterprise Architect

Enterprise Architect (EA) je softwarový nástroj vyvinutý firmou Sparx Systems, který se používá k systémové analýze a návrhu systému. Jde o placený produkt, ale lze jej vyzkoušet po dobu třiceti dní zdarma. Používá se po celý životní cyklus vývoje systému, tedy od analýzy přes návrh, implementaci, testování až po údržbu modelů. Velkou výhodou EA je obrovský repozitář modelů a rychlost, se kterou dokáže tyto mnohdy velice rozsáhlé modely načíst. Je vhodný nejen pro práci jednotlivců, ale i celého týmu na sdílených projektech díky integrovaným funkcím pro správu verzí a cloudovým serverům. [63]

4.8 Vývojové prostředí – IDE

4.8.1 IntelliJ IDEA a Webstorm

Obě vývojová prostředí jsou produktem společnosti JetBrains, která poskytuje nástroje a IDE pro spoustu programovacích jazyků. V tomto případě se jedná o Javu (IDEA) a JavaScript (Webstorm). Důvodem pro jejich zvolení bylo velké množství nabízených funkcí a nástrojů, jenž výrazně zvyšují programátorovu produktivitu. Například stojí za zmínku vydatná podpora

klávesových zkratek, detekce duplicitního kódu, rychlý a kvalitní našeptávač nebo také integrovaná podpora nástrojů pro verzování projektů. O tom, jak jsou tyto nástroje oblíbené a kvalitní, vypovídá i řada obdržených certifikátů a cen. [64-66]

IntelliJ IDEA (viz obrázek 15) je dostupná ve dvou verzích, a to Community a Ultimate. Community Edition je volně dostupná verze vhodná především pro vytváření Java SE aplikací. Tvorba webových aplikací není v této verzi podporována, a tudíž se k tomuto účelu používá zpoplatněná Ultimate Edition. Druhé zvolené IDE Webstorm nabízí pouze placenou verzi, kterou je možné si vyzkoušet po dobu třiceti dní zdarma. Pro studenty a akademický personál jsou JetBrains produkty zdarma. [64-66]

Obrázek 15 – IntelliJ IDEA IDE

4.8.2 Android studio

Pro vývoj mobilní aplikace na platformě Android bylo zvoleno volně dostupné, oficiální IDE, které vyvíjí Google a nazývá se Android studio. Je založeno na vývojovém prostředí IntelliJ IDEA, díky čemuž obsahuje i řadu jeho funkcí. Kromě toho však nabízí i funkce vlastní specifické pro vývoj mobilních aplikací. Řadí se mezi ně například rozsáhlé testovací nástroje, funkce Instant Run (okamžitý projev změny kódu ve spuštěné aplikaci bez vytvoření nového APK[10]), emulátor s rychlým a bohatým nastavením nebo podpora pro Google Cloud Platform[11]. V neposlední řadě má také integrovaný nástroj Gradle. [67]

S pomocí Android studia lze vyvíjet aplikace pro všechna zařízení, která používají operační systém Android. Tedy nejen pro mobilní telefony, ale i pro chytré hodinky, televizory s Android TV nebo automobily se systémem Android Auto. [67]

[10] APK (Android application package file) – instalační balíček aplikací pro operační systém Android.
[11] Google Cloud Platform – sada nástrojů a služeb používaná pro cloudový hosting webových aplikací.

5 REZERVAČNÍ SYSTÉM LETENEK

Cílem praktické části této diplomové práce je vytvořit komplexní systém pro rezervaci letenek. Systém se skládá ze serveru, mobilní a webové aplikace. Výčet použitých technologií se nachází v předchozí kapitole. V této části je popsána analýza, návrh databáze a implementace vyvíjeného systému.

5.1 Analýza požadavků

Základním kamenem a první fází při vývoji systému je analýza požadavků. Jedná se o proces, jehož cílem je definovat všechny požadavky na systém, jeho funkčnost a případná omezení, která by měl splňovat. Zachytit všechny požadavky před začátkem vývoje je téměř nemožné, protože se během vývojového cyklu často mění. Požadavky se dělí na funkční a nefunkční.

5.1.1 Funkční požadavky

Funkční požadavky popisují chování a požadované funkce vyvíjeného systému.

- R01 – Rezervační systém bude umožňovat přihlášení pomocí jména a hesla.
- R02 – Rezervační systém bude ověřovat oprávnění uživatelů podle definovaných rolí.
- R03 – Rezervační systém bude obsahovat tři role. Jedná se o tyto role: administrátor, zástupce letecké společnosti a zákazník.
- R04 – Rezervační systém bude umožňovat přidávat slevy k jednotlivým letům.
- R05 – Rezervační systém bude zobrazovat grafy a statistické informace.
- R06 – Rezervační systém bude evidovat a spravovat různé letecké společnosti.
- R07 – Rezervační systém bude evidovat a spravovat různé typy letadel.
- R08 – Rezervační systém bude umožňovat evidenci a správu informací o letištích a jejich regionálním umístění.
- R09 – Rezervační systém bude umožňovat vytvářet a upravovat let z letiště *A* do letiště *B*.
- R10 – Rezervační systém bude umožňovat rezervovat letenky na vybrané lety.

5.1.2 Nefunkční požadavky

Nefunkční požadavky slouží k definování omezení daného systému či specifikaci jeho vlastností. Do této kategorie patří např. požadavky na vzhled, výkonnost či údržbu. V rámci vyvíjeného systému se jedná o následující požadavky:

- R11 – Rezervační systém bude tvořen serverem, mobilní a webovou aplikací.
- R12 – Rezervační systém bude využívat jednotnou databázi – MySQL verze 8.0.11.

- R13 – Rezervační systém bude lokalizován do češtiny.
- R14 – Serverová část bude vytvořena pomocí frameworku Spring.
- R15 – Mobilní aplikace bude fungovat na zařízeních s OS Android 7.0 a vyšší.
- R16 – Webová aplikace bude vytvořena pomocí frameworku Angular 7.
- R17 – Aplikace budou mít jednoduché a intuitivní ovládání.
- R18 – Rezervační systém bude podporovat základní prvky bezpečnosti (autorizace, autentizace).
- R19 – Rezervační systém bude snadno rozšiřitelný.

5.2 Uživatelské role

Při návrhu systému je nutné specifikovat uživatelské role a oprávnění, která daná role bude mít. V systému existují následující role:

- nepřihlášený uživatel,
- přihlášený uživatel,
    - administrátor,
    - zástupce letecké společnosti (prodejce),
    - zákazník.

5.2.1 Nepřihlášený uživatel

Nepřihlášený uživatel bude mít oprávnění k zobrazení jen některých letů. Jedná se o lety, kterým zbývají pouze poslední místa k rezervaci nebo lety vyhledané podle vyplněných kritérií. V případě, že uživatele let zaujme, bude si moct zobrazit jeho plný detail. Samozřejmostí je možnost registrace či přihlášení.

5.2.2 Zákazník

Náleží mu veškerá práva nepřihlášeného uživatele. Navíc bude mít možnost zobrazení kompletního seznamu aktuálních letů či zarezervování letenek. Také oproti nepřihlášenému uživateli bude moci editovat svůj profil, zobrazit si seznam rezervací včetně jejich detailu nebo rušit ještě neproběhlé rezervace.

5.2.3 Zástupce letecké společnosti

Účelem této role je reprezentovat vybranou leteckou společnost a jejím jménem vytvářet nové lety z destinace *A* do destinace *B*. Nedílnou součástí tohoto procesu bude i definování počtu a cen letenek či určení případných slev. Kromě toho bude mít přístup k nejrůznějším statistikám

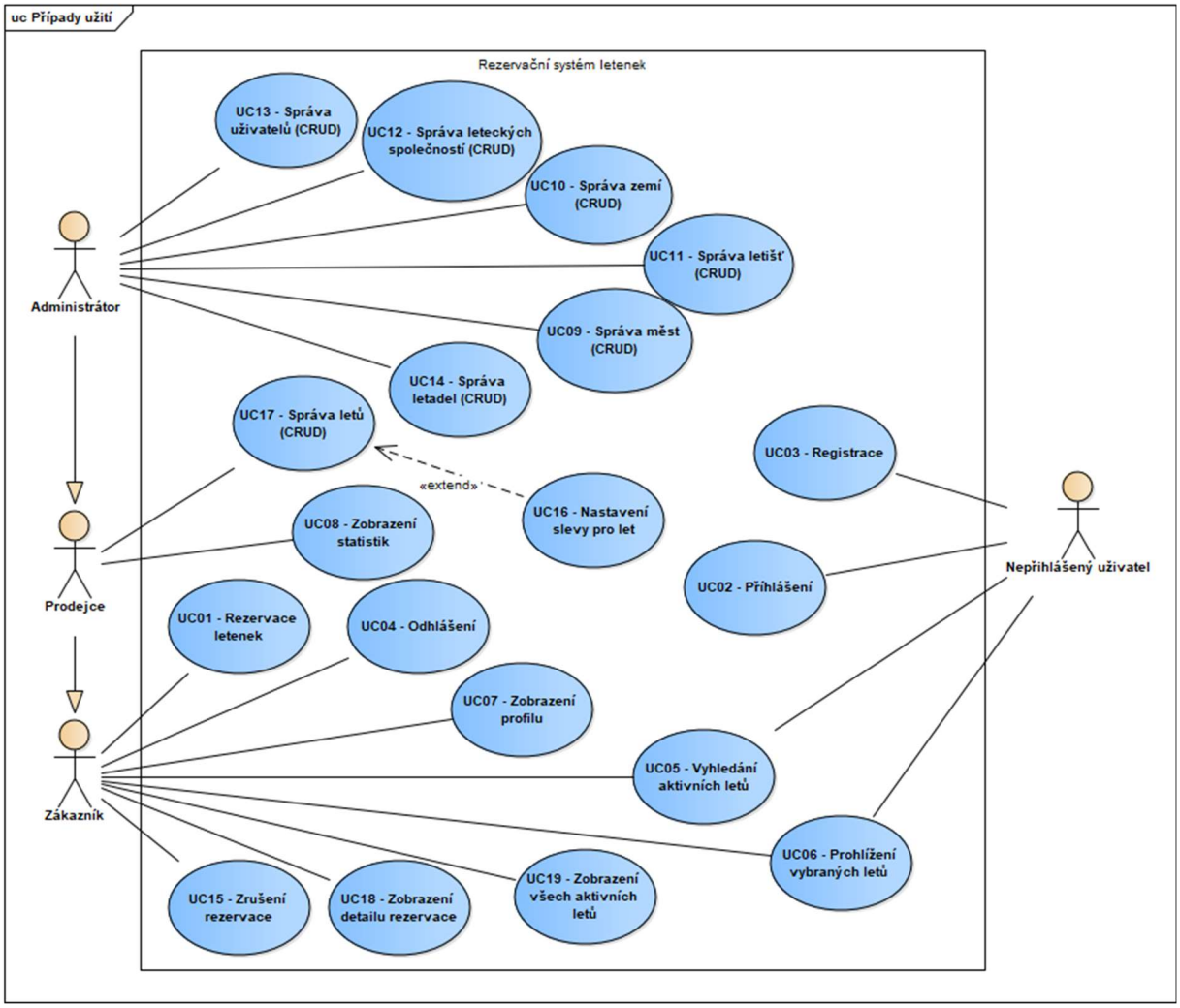
o vytvořených rezervacích u letů vlastní společnosti. Dále mu náleží stejná oprávnění jako zákazníkovi.

5.2.4 Administrátor

Administrátor může využívat veškeré funkce, které již byly popsány výše. Kromě toho je zodpovědný za celkovou správu aplikace. Konkrétně se jedná o správu a evidenci následujících informací: uživatelů, leteckých společností, typů letadel a regionálních informací o letištích.

5.3 UML Use Case diagram

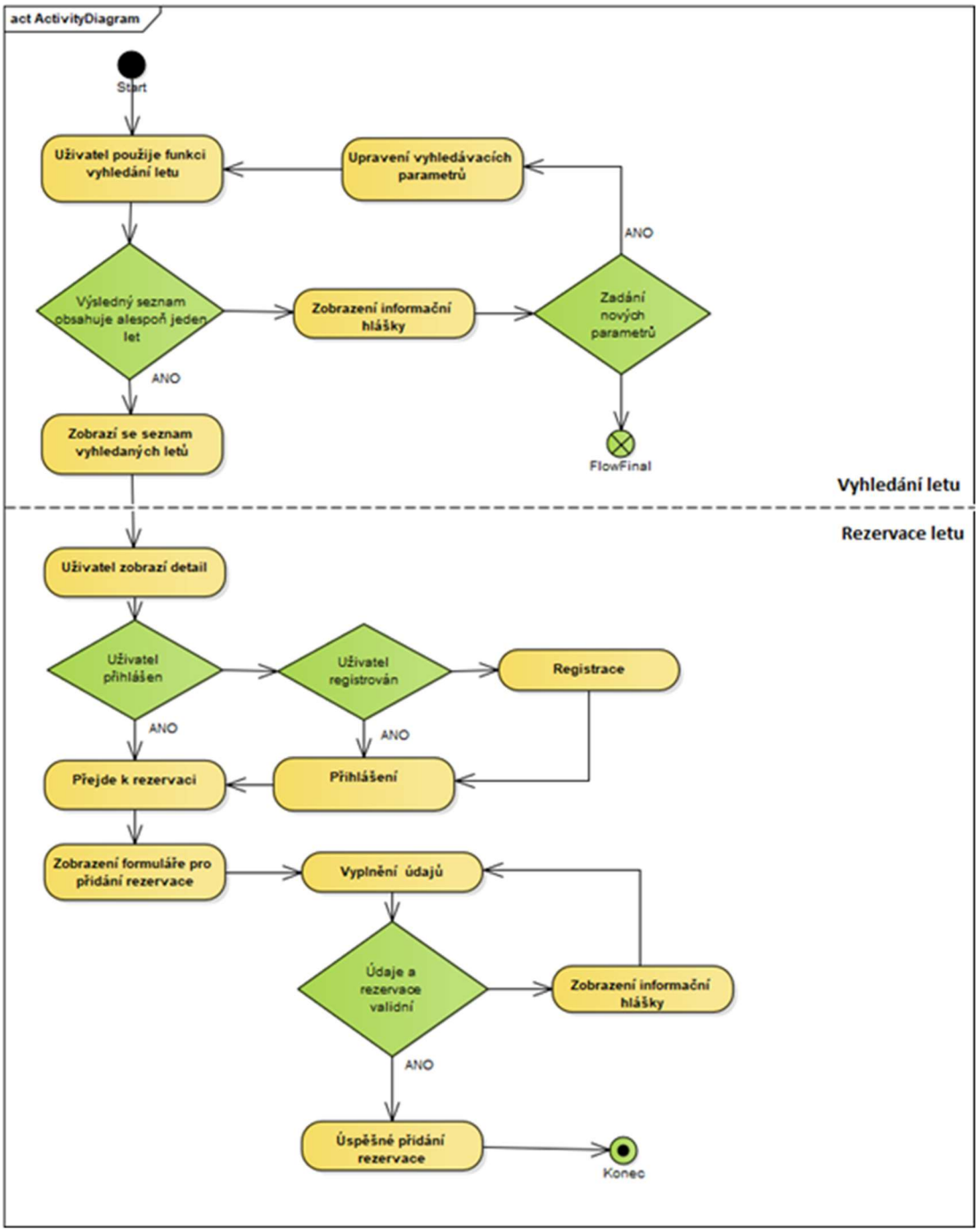
Zobrazuje chování systému z pohledu rolí definovaných v předešlé podkapitole. Jeho účelem je popsat funkcionalitu systému, tedy co má systém umět, ale ne jak to má udělat. Use Case diagram (viz obrázek 16) se skládá z případů užití (use case), aktérů a jejich vzájemných vztahů.



Obrázek 16 – UML Use Case diagram systému

5.4 UML Activity diagram

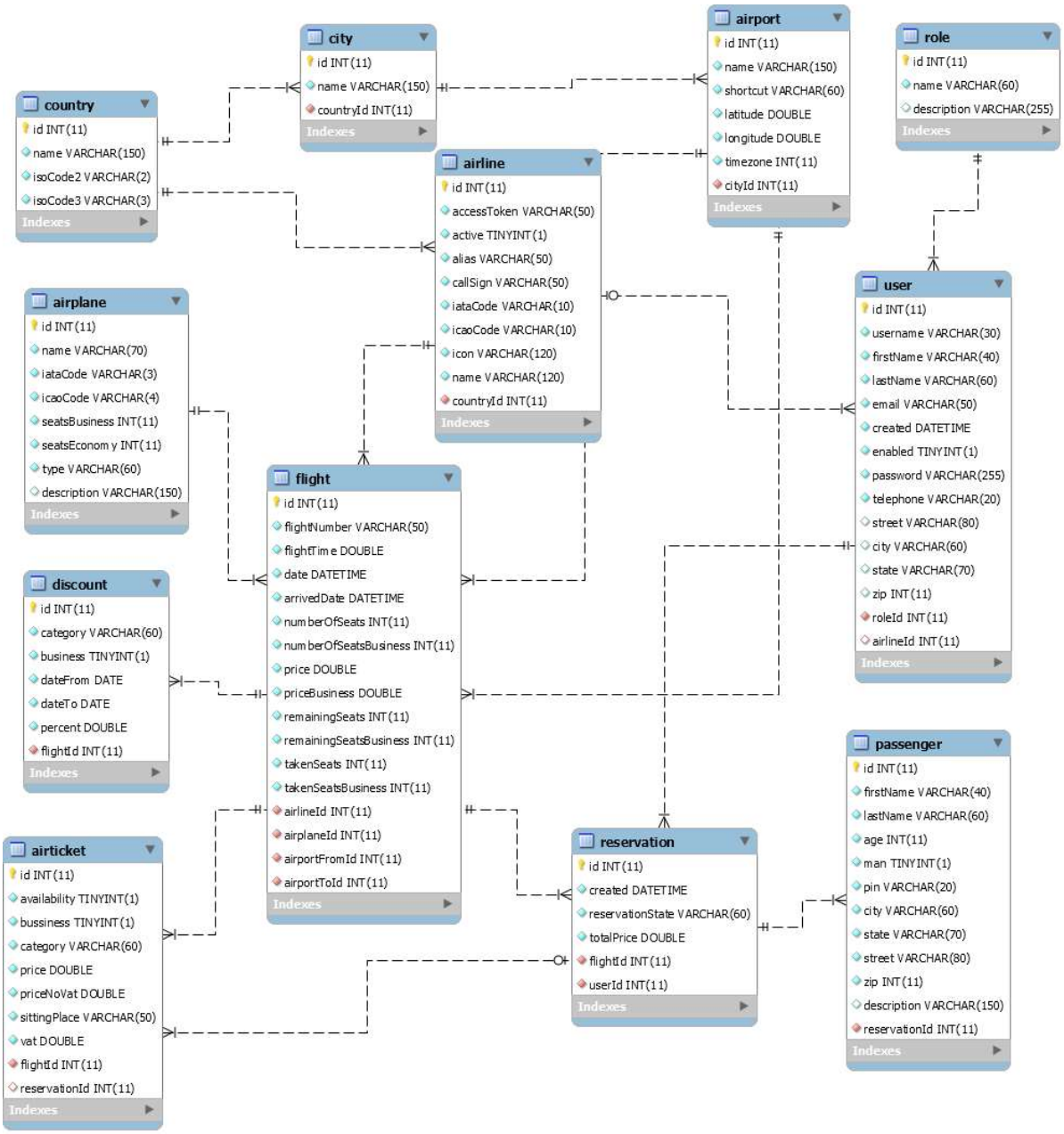
Jedná se o druh UML diagramu popisující chování akcí, které probíhají v aplikaci. Následující obrázek (obrázek 17) zobrazuje Activity diagram, který se skládá ze dvou na sebe navazujících částí: vyhledání a rezervace letu.



Obrázek 17 – UML Activity diagram

5.5 Návrh databáze

Jako datové úložiště byla vybrána databáze MySQL, jejíž představení se nachází v kapitole 4.4. Návrh schématu byl zhotoven v nástroji MySQL Workbench a je zobrazen na následujícím obrázku (obrázek 18). Schéma obsahuje celkem třináct tabulek. Nejdůležitější tabulky jsou *reservation, flight, user, airticket* a *passenger*. V jejich rámci se odvíjí hlavní funkcionalita aplikace – rezervace letenek. Naproti tomu tabulky *airplane, airline, city, country* a *airport* slouží hlavně pro administrátora aplikace.



Obrázek 18 – E-R diagram

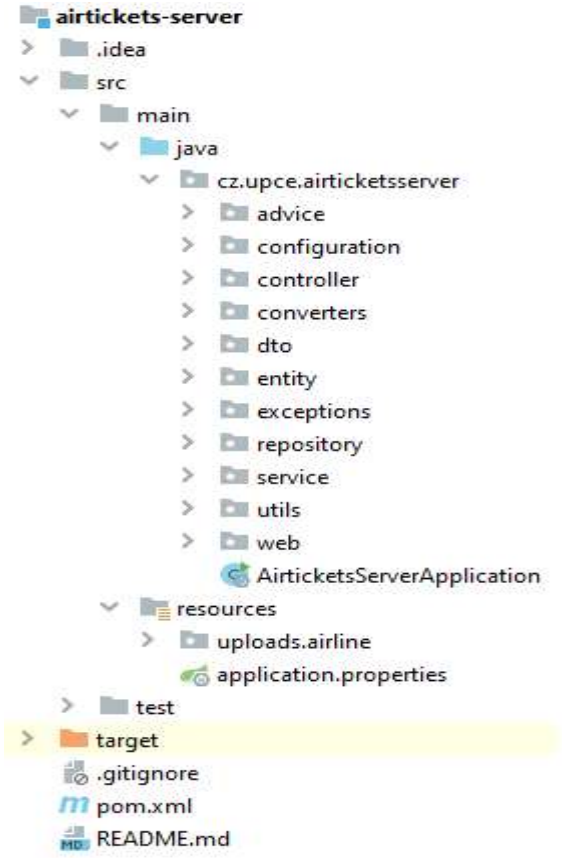
5.6 Popis implementace

Implementovat takto komplexní systém je možné několika způsoby. Na základě autorových zkušeností s vývojem tzv. backend aplikací byl zvolen způsob, kdy bylo nejdříve vytvořeno REST API založené na vrstvené architektuře frameworku Spring a až následně došlo k implementaci webové SPA Angular aplikace a mobilní aplikace pomocí architektury MVVM. Cílem této kapitoly je seznámit čtenáře s vybranými částmi implementovaného systému.

5.6.1 Serverová část

Tato část je věnována popisu nejdůležitější části systému – serveru. Ten za pomoci architektonického stylu REST poskytuje data pro obě klientské aplikace.

Adresářová struktura



Obrázek 19 – Struktura Spring Boot projektu

Serverová část diplomové práce je postavena na technologii Spring Boot s využitím Mavenu. Jak již bylo řečeno v kapitole 4.7.1, Maven vytváří výchozí strukturu projektu (viz obrázek 19). Zdrojové kódy se nachází v adresáři *src/main/java/cz/upce/airticketsserver*. Adresář *src/main/resources* obsahuje složku pro ukládání ikon leteckých společností a konfigurační

soubor *application.properties*. Ten obsahuje konfiguraci připojení k databázi (viz zdrojový kód 5), dobu platnosti JWT tokenů nebo nastavení maximální velikosti přenášeného souboru.

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/diplomova_prace?useLegacyDatetimeCode=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
```

Zdrojový kód 5 – Konfigurační soubor application.properties

Server dodržuje vrstvenou architekturu frameworku Spring. Vybrané části budou popsány dále.

Datová vrstva

Základem datové vrstvy jsou tzv. entitní třídy, ze kterých se díky použití JPA anotací (z balíčku *javax.persistence*) a frameworku Hibernate generují databázové tabulky. Každá entita obsahuje svůj jednoznačný unikátní identifikátor typu *Integer*, který je generován automaticky na základě zvolené strategie definované pomocí anotace *@GeneratedValue*. Pro databázový systém MySQL se doporučuje používat strategii *IDENTITY*, jenž využívá pro generování sloupec s označením *auto_increment*. Tento sloupec je zodpovědný za hodnotu identifikátoru a v případě vložení nového záznamu dojde k automatickému zvýšení jeho hodnoty. V každé tabulce se nachází vždy pouze jeden takový sloupec.

```
@Entity
@Getter
@Setter
public class City {
    /**
     * Unikátní identifikátor města
     */
    @Id
    @Column(name = "id", nullable = false)
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    /**
     * Jméno města
     */
    @Column(name = "name", nullable = false, length = 150)
    private String name;
    /**
     * Země ve které se město nachází
     */
    @ManyToOne
    @JoinColumn(referencedColumnName = "id", nullable = false)
    private Country country;
}
```

Zdrojový kód 6 – Entitní třída City

Pro většinu aplikací v programovacím jazyce Java je typické, že jsou plné tzv. *boilerplate*[12] kódu. Pro jeho redukci bylo využito knihovny s názvem Lombok. Ta za pomoci anotací vygeneruje vybraný kód do tříd až při kompilaci aplikace, čímž se zdrojový kód výrazně zkrátí a zpřehlední. Pro použití v IntelliJ IDEA IDE je nutné doinstalovat plugin. Ve zdrojovém kódu 6 je zobrazena entitní třída *City* i s použitou *Lombok* anotací *@Getter* a *@Setter*.

Rozhraní označené anotací *@Repository* slouží pro manipulaci s databází skrze vytvořené entity. Jedná se o tzv. repozitáře, přičemž každé entitě odpovídá právě jeden. Pro zjednodušení práce s repozitáři poskytuje *Spring Data JPA* několik generických rozhraní. V aplikaci bylo použito rozhraní *JpaRepository*, které obsahuje dva parametry, a to entitu a datový typ jejího identifikátoru. Jeho hlavní výhodou je poskytnutí základních CRUD operací bez nutnosti jejich implementace.

Následující zdrojový kód (zdrojový kód 7) obsahuje dva příklady, jak lze toto rozhraní rozšiřovat o další metody. V prvním případě Spring vygeneruje potřebný SQL dotaz na základě názvu metody. Spring poskytuje velké množství klíčových slov a modifikátorů, skrze které je možné dotaz specifikovat např. *NotContains, IsStartingWith, Between* atd. Pro jejich plný výčet je doporučeno navštívit oficiální dokumentaci. Druhý způsob je využití anotace *@Query*, kam se výsledná podoba dotazu zapisuje ručně za pomoci SQL nebo JPQL. Tato možnost se používá pro konstrukci složitějších dotazů.

```
@Repository
public interface FlightRepository extends JpaRepository<Flight, Integer> {
    Boolean existsByFlightNumber(String flightNumber);

    @Query("SELECT f FROM Flight f WHERE f.airportFrom.id = :airport OR f.airportTo.id = :airport")
    Collection<Flight> getFlightsByAirport(@Param("airport") int airport);
    // another repository methods
}
```

Zdrojový kód 7 – Ukázka repozitáře

Servisní vrstva

Jedná se o řadu servisních rozhraní a jejich implementací. Servisní třídy jsou označené anotací *@Service*. V této vrstvě se nachází veškerá aplikační logika včetně využití transakcí při komunikaci s perzistentní vrstvou. Kromě toho slouží také k překladu DTO[13] objektů na entitní objekty. DTO je označení pro jednoduché objekty určené k přenosu dat, které se používají na

[12]Boilerplate – často se opakující kód.
[13] DTO – Data Transfer Object

komunikaci mezi serverem a klienty. Jejich použití je vhodné především, když je potřeba v rámci jednoho dotazu na server poskytnout klientovi data z různých entit. Bez využití DTO by bylo nutné odeslat více samostatných dotazů, což není příliš žádoucí pro mobilní aplikace. Mimo to je samozřejmě možné některé atributy skrýt, například při odeslání entity User není vhodné zároveň s ní odesílat i její heslo.

V rámci aplikace je vytvořena obalová třída *Message*, která je používána jako návratová hodnota většiny metod. Hlavním důvodem jejího zavedení je vytvoření unifikované struktury, která bude jednoduše zpracovatelná v rámci klientských aplikací.

```
@Override
public Message getById(int id) {
    Message m = new Message();
    Country country = countryRepository.findById(id).orElseThrow(()
            -> new NotFoundException("Země s tímto id neexistuje!"));
    m.setCode(ApplicationConstants.STATUS_OK);
    m.setPayload(countryConverter.createDTO(country));
    return m;
}
```

Zdrojový kód 8 – Ukázka metody ze servisní třídy

Ve zdrojovém kódu (zdrojový kód 8) je zobrazena jednoduchá metoda servisní třídy, která slouží k získání země podle jejího ID. V případě, že pro toto ID neexistuje záznam, tak nastane výjimka. Pro zpracování výjimek je v aplikaci použita speciální třída s anotací *@ControllerAdvice*. Jednotlivé typy výjimek, které chceme zpracovávat, musí obsahovat veřejnou metodu s anotací *@ExceptionHandler (typ výjimky)*. V těle metody je pak výjimka přeložena na objekt *Message*, který je odeslán v rámci HTTP odpovědi. Kromě toho je pomocí anotace *@ResponseStatus* definován i správný HTTP status odpovědi. V následujícím kódu (zdrojový kód 9) se nachází ukázka zpracování výjimky *NotFoundException*.

```
@ResponseBody
@ExceptionHandler(NotFoundException.class)
@ResponseStatus(HttpStatus.NOT_FOUND)
public Message handleNotFound(final NotFoundException ex) {
    return new Message(ex.getMessage(), ApplicationConstants.STATUS_NOT_FOUND);
}
```

Zdrojový kód 9 – Ukázka třídy pro zachytávání výjimek

Vrstva REST rozhraní (controller)

Ke komunikaci s klientskou aplikací je zapotřebí, aby server zpracovával její HTTP požadavky. Tyto požadavky jsou vedeny na URL adresu, kterou je nutné namapovat k příslušným metodám REST rozhraní pomocí anotace *@RequestMapping*. Za tento proces jsou zodpovědné tzv. controllery. Nachází se ve stejnojmenném balíčku controller a jsou označovány anotací *@RestController*.

Spring ve verzi 4.3 představil nové způsoby zápisu anotace *@RequestMapping*. Jde o anotace ve tvaru *<HTTP metoda>Mapping,* tedy např. *@GetMapping, @PostMapping, @PutMapping* a *@DeleteMapping*. Jejich používání má za následek zlepšení přehlednosti kódu.

Pro možnost autorizace na úrovni jednotlivých metod je v aplikaci použita anotace *@PreAuthorize*. Ta určuje, zda může být metoda uživatelem s danou rolí použita, nebo ne. Metodu pro úpravu letiště z kódu 10 může použít pouze uživatel s rolí Admin, v ostatních případech bude vracen stavový kód 401 Unauthorized.

```
@RestController
@RequestMapping("/api/airport")
public class AirportController {
    @Autowired
    IAirportService airportService;
    @Autowired
    ICityService cityService;

    @PutMapping("/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    public Message updateAirport(@PathVariable("id") Integer airportID, @RequestBody @Valid AirportDTO airport,
                                 BindingResult result) throws BindException {
        if (result.hasErrors()) {
            throw new BindException(result);
        } else {
            return airportService.updateAirport(airportID, airport);
        }
    }
    // another methods
}
```

Zdrojový kód 10 – Ukázka REST controlleru

V předcházející ukázce kódu (zdrojový kód 10) se nachází metoda, která slouží k upravení letiště. Pomocí anotace *@PathVariable* je z URL adresy získán parametr ID upravovaného letiště. Data pro upravení získává controller ve formátu JSON. Pokud mají správnou strukturu a odpovídají validacím, jsou pomocí knihovny Jackson automaticky namapovány na příslušné DTO objekty. Poté jsou skrze metodu servisní vrstvy dále zpracovávány. V ideálním případě by controller neměl obsahovat vůbec žádnou aplikační logiku, ale měl by pouze všechny

požadavky směrovat do servisní vrstvy. Pokud vše proběhlo úspěšně, tedy nenastala žádná výjimka, je vracen stavový kód OK a zpráva o provedení úpravy. Zachytávání výjimek bylo popsáno v předcházející části. Odpověď, ať chyba či úspěch, je poté pomocí Jacksonu serializována[14] a ve formátu JSON odeslána zpět klientovi.

Zabezpečení

Pro zabezpečení serveru je použit jeden z modulů Spring Frameworku, konkrétně knihovna *Spring security*. Ta se zaměřuje na poskytnutí autentizace a autorizace pro Java aplikace. Její výhodou je poměrně rychlé a snadné nastavení. Konfigurace se nachází v třídě *SecurityConfiguration* v balíčku configuration. Lze zde najít například informace o použitých filtrech nebo o tom, které URL adresy jsou zabezpečené (včetně role nutné pro přístup).

Na ověření identity uživatele je používán JSON Web Token, zkráceně JWT. JWT je otevřený standard, který slouží pro bezpečný přenos informací mezi dvěma stranami. Skládá se ze tří částí: *header*, *payload* a *signature*. *Header* obsahuje informace o tom, že jde o JWT token a jaký způsob šifrování je použit (např. HS512). *Payload* obsahuje tzv. *claims*, což jsou například informace o uživateli nebo čas vypršení tokenu. *Signature* slouží jako digitální podpis pro ověření integrity tokenu.

Token je vygenerován v případě úspěšného přihlášení a v odpovědi odeslán klientovi. Následně je při každém HTTP požadavku, který vyžaduje oprávnění, zasílán na server v hlavičce *Authorization* pomocí *Bearer* schématu.

Protože *Spring security* neobsahuje žádné výchozí autentizační filtry pro práci s JWT, je vytvořen vlastní filtr, který se nachází v třídě *JWTAuthenticationFilter*. Ta obsahuje metodu *doFilterInternal*, která při každém HTTP požadavku na zabezpečené URL získá z hlavičky token, který následně validuje. Pokud nenastane žádná chyba, tak dojde k uložení údajů o uživateli do bezpečnostního kontextu *SecurityContext*. Prostřednictvím pomocné třídy *SecurityContextHolder* lze získat informace o aktuálně přihlášeném uživateli. V případě, že je token nevalidní nebo uživatel přistupuje k zabezpečenému koncovému bodu, na který nemá práva, je vracena chybová zpráva s HTTP kódem 401 Unauthorized.

Validace

V každé aplikaci je dobrým zvykem validovat zadávané vstupní hodnoty. Je důležité ověřovat správnost vstupů jak na straně klienta, tak hlavně na straně serveru, protože požadavky na server

[14] Jedná se o proces, při kterém je objekt převeden do proudu bytů a následně uložen.

jdou zasílat i bez pomoci klientských aplikací (např. skrze aplikaci Postman). V případě, že by server neobsahoval validátory, tak by mohlo dojít ke vzniku nekonzistence dat nebo jejich úplné ztrátě.

5.6.2 Klientské aplikace

V rámci tohoto systému byly vytvořeny dvě klientské aplikace. Jedna určená pro mobilní platformu Android a druhá webová, která je vytvořena jako klasická jednostránková aplikace pomocí frameworku Angular. Minimální podporovaná verze u mobilní aplikace je 7.0 Nougat (api 24). V této podkapitole se nachází popis vybraných částí obou aplikací.

Webová aplikace

Jak již bylo řečeno v předcházející kapitole, komunikace mezi klientskými aplikacemi a serverem probíhá pomocí architektonického stylu REST. Moderní prohlížeče podporují dva základní způsoby pro odesílání požadavku na server: *XMLHttpRequest* a *fetch() API*. Angular pro tento účel využívá službu *HttpClient* z balíčku *HttpClientModule,* která poskytuje jednoduché API pro aplikace založené na *XMLHttpRequest*. Na následující ukázce zdrojového kódu (zdrojový kód 11) je zobrazena servisní třída *AirplaneService*, která zmíněnou službu *HttpClient* využívá.

```
@Injectable({
  providedIn: 'root'
})
export class AirplaneService extends BaseHttpService {
  private relativeUrl = `${this.baseUrl}airplane`;
  constructor(http: HttpClient) {
    super(http);
  }
  getAllAirplanes() {
    return this.http.get<Airplane[]>(this.relativeUrl);
  }
  deleteAirplane(id: number): Observable<Message> {
    return this.http.delete<Message>(`${this.relativeUrl}/${id}`);
  }
  addAirplane(airplane: Airplane): Observable<Message> {
    return this.http.post<Message>(this.relativeUrl, airplane);
  }
  updateAirplane(airplane: Airplane): Observable<Message> {
    return this.http.put<Message>(`${this.relativeUrl}/${airplane.id}`, airplane);
  }
  // another methods
}
```

Zdrojový kód 11 – Servisní třída Angular aplikace

Po získání dat v servisní třídě dojde v rámci komponent k jejich zpracování a případnému vykreslení v prohlížeči. Ve zdrojovém kódu 12 se nachází ukázka metody z komponenty *AirplaneListComponent*, která slouží pro smazaní vybraného letadla ze seznamu. Uvnitř metody dojde k použití servisní třídy, která požadavek odešle. V případě úspěchu je entita odebrána ze seznamu zobrazených letadel. Pokud nastane chyba, například letadlo nelze smazat z důvodu integritního omezení, tak server odešle chybovou zprávu a ta je následně zobrazena.

```
deleteAirplane() {
  const index = this.airplaneList.indexOf(this.selectedAirplane);
  this.airplaneService.deleteAirplane(this.selectedAirplane.id).subscribe(res => {
      this.airplaneList = this.airplaneList.filter((val, i) => i !== index);
      this.messageService.add({severity: 'success', summary: 'Úspěch', detail: res.message[0]});
    }, (err) => {
      this.messageService.add({severity: 'error', summary: 'Chyba', detail: err});
    }
  );
  this.displayDialog = false;
  this.selectedAirplane = null;
}
```

Zdrojový kód 12 – Ukázka metody v rámci komponenty

Pro tvorbu uživatelského rozhraní jsou použity komponenty z knihovny PrimeNG, jedná se např. o *p-table*, *p-dialog* či *p-toast*. *P-table* slouží pro zobrazení dat tabulky. Umožňuje získaná data různě filtrovat, řadit, stránkovat, exportovat do csv atd. *P-dialog* vytváří jednoduché dialogové okno, které je používané pro upravení nebo přidání entit. *P-toast* je používán napříč celou aplikací pro zobrazení informačních, chybových či varovných zpráv.

V následující ukázce zdrojového kódu (zdrojový kód 13) je vyobrazen krátký příklad použití výše zmíněné komponenty *p-table* pro zobrazení seznamu letadel. Ve vrchní části se definují jednotlivé parametry tabulky, mezi které patří například parametr *columns* (definuje zobrazované sloupce) nebo parametr *value* (nastavuje data). Detailní popis všech dalších parametrů se nachází v dokumentaci PrimeNG.

Tělo elementu *p-table* je rozděleno do čtyř částí – *caption, header, body* a *summary*. První část obsahuje globální filtr tabulky a tlačítka pro otevření dialogu na přidání letadla a exportu do csv. V hlavičce se definuje řazení a typ filtru pro každý sloupec tabulky. Třetí část slouží k samotnému vykreslení dat. Data lze různě formátovat pomocí tzv. *pipes*. Angular obsahuje velké množství již předpřipravených *pipe* (např. *titlecase* – první znak výstupu bude velkým

písmenem), ale samozřejmě je možné si vytvořit i vlastní. Na závěr je zobrazen aktuální počet entit tabulky, v tomto případě letadel.

```
<p-table #dt [columns]="cols" [value]="airplaneList" [responsive]="true" selectionMode="single"
          [(selection)]="selectedAirplane"  [resizableColumns]="true" columnResizeMode="expand"
          exportFilename="airplanes"  dataKey="id" [paginator]="true" [rows]="15"
          (onRowSelect)="onRowSelect($event)">
    <ng-template pTemplate="caption">
        <div style="float:left">
          <button type="button" pButton icon="fa fa fa-plus" iconPos="left" style="float:left"
                  label="Přidat letadlo" (click)="showDialogToAddAirplane()"></button>
          <button type="button" pButton icon="fa fa-file-o" label="Exportovat do csv"
                  class="ui-button-success" iconPos="left" (click)="dt.exportCSV()"></button>
        </div>
        <div style="text-align: right"...> //global filter
    </ng-template>
    <ng-template pTemplate="header" let-columns>
      <tr>
        <th *ngFor="let col of columns" [pSortableColumn]="col.field">
          {{col.header}}
          <p-sortIcon [field]="col.field" ariaLabel="Aktivovat řazení"...> // sort icon
        </th>
      </tr>
      <tr>
        <th *ngFor="let col of columns" [ngSwitch]="col.field">
          <input *ngSwitchCase="'type'" pInputText type="text" style="width: 100%"
                 (input)="dt.filter($event.target.value, col.field, 'contains')">
          // another filters
        </th>
      </tr>
    </ng-template>
    <ng-template pTemplate="body" let-rowData let-airplane>
      <tr [pSelectableRow]="rowData">
        <td>{{airplane.type | lowercase | titlecase}}</td>
        // another values
      </tr>
    </ng-template>
    <ng-template pTemplate="summary" let-rowData>
      <div style="text-align:left">
        <label>Počet letadel:  {{getTableSize()}}</label>
      </div>
    </ng-template>
  </p-table>
```

Zdrojový kód 13 – Tabulka seznamu letadel

Mobilní aplikace

Mobilní aplikace ke komunikaci se serverem využívá populární knihovnu Retrofit. Jedná se o HTTP klienta pro platformu Android, který umožňuje odesílat jak asynchronní, tak synchronní požadavky na vzdálený server. Požadavky jsou specifikovány skrze jednotlivé

metody rozhraní, které jsou doplněné o anotace obsahující příslušnou HTTP metodu a relativní URL cestu na server. Návratovou hodnotou je objekt typu *Call,* jenž obsahuje druh očekávaného objektu. V parametrech metod lze také používat různé anotace pro specifikaci požadavku jako např. *@Query* nebo *@Path*. Anotace *@Query* označuje parametry dotazu, které se pak automaticky přidávají na konec URL adresy. Pomocí anotace *@Path* je možné specifikovat dynamickou část URL adresy. Vzhledem k menšímu počtu požadavků jsou metody seskupeny pouze do jednoho rozhraní.

V rámci aplikace jsou vytvořené modelové POJO[15] třídy, které mají stejnou strukturu jako přenášené JSON objekty. Pro jejich převod existují konvertory. Retrofit podporuje množství konvertorů, přičemž v aplikaci je využita knihovna Gson.

Zdrojový kód 14 ukazuje vytvoření instance API, skrze kterou se v aplikaci volají jednotlivé funkce rozhraní. Je nutné definovat URL adresu serveru, výše zmíněný konvertor a metodu, která přidá JWT token do každého odeslaného požadavku.

```
public static Retrofit getRetrofitInstance(final Context context) {
    if (retrofit == null) {
        retrofit = new retrofit2.Retrofit.Builder()
                .baseUrl(BASE_URL)
                .addConverterFactory(GsonConverterFactory.create())
                .client(getUnsafeOkHttpClient(context))
                .build();
    }
    return retrofit ;
}
```

Zdrojový kód 14 – Inicializace instance Retrofit

Vzhledem k tomu, že aplikace komunikuje s REST API přes internet, je nutné tuto funkci v kódu povolit. Aplikace totiž přístup k internetu umožňuje pouze pokud se v souboru *AndroidManifest.xml* nachází následující řádka kódu:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Zdrojový kód 15 – Povolení přístupu k internetu v Android aplikaci

Kromě toho se v tomto souboru nachází i některé další důležité informace o aplikaci jako je její název, ikona, nastavení úvodní aktivity, různá další povolení (např. fotoaparát, senzory) a jiné.

[15] POJO – Plain old Java object

Po vytvoření instance Retrofit již nic nebrání odesílání požadavku na server. Následující ukázka kódu (zdrojový kód 16) zobrazuje použití neblokujícího asynchronního volání pro zisk seznamu letů pro úvodní stránku. Pro komunikaci skrze Retrofit je nutné využít rozhraní *Callback*, které definuje dvě metody. Metoda *onResponse()* se provede při úspěšné komunikaci se serverem, tedy pokud je přijata HTTP odpověď. Pokud komunikace neproběhne, tak nastane druhá metoda nazývaná *onFailure()*. K tomu dojde například tehdy, když není zařízení připojené k internetu.

```
private void loadCarouselData() {
    ApiInterface service = RetrofitInstance.getRetrofitInstance(getContext())
            .create(ApiInterface.class);
    service.getCarouselData().enqueue(new Callback<List<FlightSearchResponse>>() {
        @Override
        public void onResponse(Call<List<FlightSearchResponse>> call,
                               Response<List<FlightSearchResponse>> response) {
            dismissLoadingDialog();
            if (response.isSuccessful()) {
                mViewModel.setData(response.body());
                setUpAdapter(response.body());
            } else {
                Toast.makeText(getContext(), getString(R.string.no_result),
                        Toast.LENGTH_LONG).show();
            }
        }

        @Override
        public void onFailure(Call<List<FlightSearchResponse>> call, Throwable t) {
            dismissLoadingDialog();
            Toast.makeText(getContext(), getString(R.string.internet_error),
                    Toast.LENGTH_LONG).show();
        }
    });
}
```

Zdrojový kód 16 – Ukázka neblokujícího asynchronního volání API

Aplikace je ovládána skrze uživatelské obrazovky, které reprezentují aktivity a fragmenty. V rámci aplikace byla vytvořena pouze jedna hlavní aktivita – *MainActivity*, která zastřešuje všechny fragmenty. Každá aktivita nebo fragment se skládá z jednotlivých elementů, které jsou seskupovány v rámci speciálních kontejnerů. Ty určují, jak budou prvky na obrazovce rozloženy. V aplikaci je hodně využíván *LinearLayout* pro vertikální či horizontální rozložení. Z elementů jsou to zase třeba *TextView* pro textové nápisy, *EditText* pro vstupní text nebo *Button* pro tlačítka. V následujícím kódu (zdrojový kód 17) je zobrazena ukázka registračního formuláře ze souboru *registration_fragment.xml*.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".RegistrationFragment">
    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
            xmlns:tools="http://schemas.android.com/tools"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical"
            tools:context=".RegistrationFragment">
            <com.google.android.material.textfield.TextInputLayout
                android:id="@+id/firstNameError"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                app:errorEnabled="true">
                <EditText
                    android:id="@+id/etFirstName"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:hint="@string/reg_first_name"
                    ... />
            </com.google.android.material.textfield.TextInputLayout>
            ...
        </LinearLayout>
    </ScrollView>
</LinearLayout>
```

Zdrojový kód 17 – Příklad XML layoutu mobilní aplikace

V dnešní době využívají moderní Android aplikace hojně komponenty z Android Architecture Components, což je kolekce knihoven umožňující tvorbu robustních aplikací. Vyvíjená aplikace není výjimkou a používá např. *ViewModel* nebo *LiveData*. *LiveData* je třída založená na návrhovém vzoru Observer. Slouží ke sledování změn v datech, přičemž na příslušnou změnu reaguje okamžitě, aby byla data stále aktuální. *ViewModel* je třída, která má za úkol ukládat a spravovat data související s životním cyklem komponent aplikace. Díky použití této třídy jsou zachována konzistentní data i při probíhajících konfiguračních změnách, jako je rotace displeje.

V rámci mobilní aplikace není využívána žádná interní databáze, protože všechna potřebná data jsou získávána ze serveru. Zároveň vzhledem k charakteru systému nebylo žádoucí, aby aplikace fungovala i v off-line režimu. V rámci budoucího rozšíření je možné tuto funkcionalitu přidat například pro zobrazení detailu již proběhlých rezervací i bez připojení k internetu.

6 FUNKCIONALITY SYSTÉMU

V následující kapitole se nachází popis funkcí obou vytvořených aplikací. Vše je popsáno tak, aby byl uživatel schopný s vybranou aplikací bez problému pracovat.

6.1 Webová aplikace

6.1.1 Přihlášení do systému

Přihlášení do systému je nutné pro využití veškerých funkcí aplikace. Pro přihlášení musí uživatel kliknout v pravém horním rohu na tlačítko *Přihlášení*. Následně se mu zobrazí formulář (viz obrázek 20), kam musí zadat své uživatelské jméno a heslo. Po přihlášení dojde k přesměrování na hlavní stránku a uživateli se zpřístupní veškeré funkce dostupné pro jeho roli. Pokud nejsou přihlašovací údaje vyplněny korektně nebo je účet zakázán, k přihlášení do aplikace nedojde a uživatel bude upozorněn na příslušnou chybu pomocí vyskakovací zprávy.

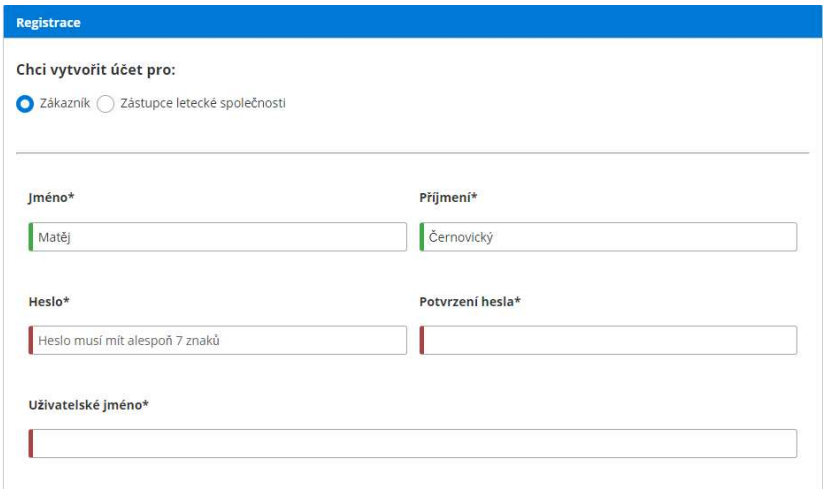
Obrázek 20 – Formulář pro přihlášení

Po úspěšném přihlášení je původní tlačítko *Přihlášení* nahrazeno tlačítkem *Odhlásit se*. Při odhlášení z aplikace dojde k přesměrování zpět na formulář pro přihlášení.

6.1.2 Registrace

K registraci uživatele je potřeba kliknout na tlačítko *Registrace* v pravém horním rohu. Zobrazí se registrační formulář (viz obrázek 21), kde je možné provést registraci pro zákazníka nebo pro zástupce letecké společnosti. Oba formuláře mají stejné náležitosti, pouze u zástupce letecké společnosti je nutné navíc zadat token. Každá společnost má svůj vlastní token, který byl vytvořen při jejím založení administrátorem systému.

Povinné položky pro vyplnění jsou označené hvězdičkou a jedná se o jméno, příjmení, heslo (potvrzení hesla), uživatelské jméno, emailovou adresu a telefon. Dále lze zadat adresu, tedy ulici, směrovací číslo, město a stát. Nakonec je třeba potvrdit souhlas se zpracováním osobních údajů. Po úspěšné registraci se může uživatel ihned přihlásit do aplikace.

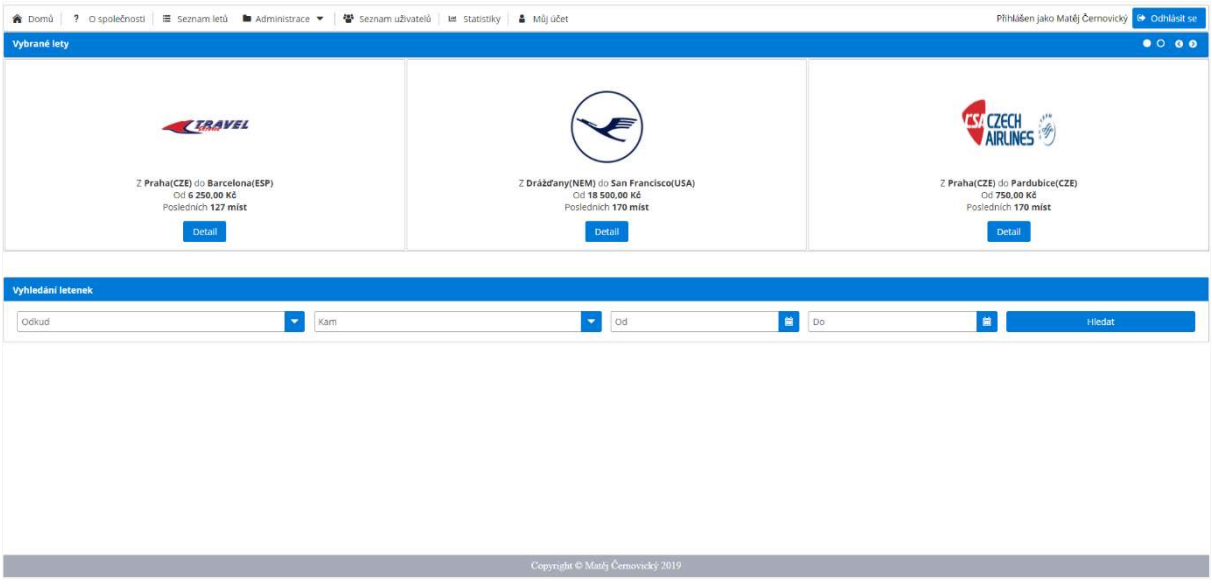


Obrázek 21 – Ukázka registračního formuláře

6.1.3 Vyhledávání letů

Dostupné pro role: všechny

Funkce vyhledávání letů se nachází na domovské obrazovce aplikace (viz obrázek 22). Kromě toho zde uživatel také nalezne seznam vybraných letů, u kterých brzy dojde k naplnění kapacity. Každý let ze seznamu obsahuje informace o ceně za nejlevnější letenku a o tom kdy, odkud a kam se uskuteční. Pro více podrobností je možné kliknout na tlačítko *Detail*.



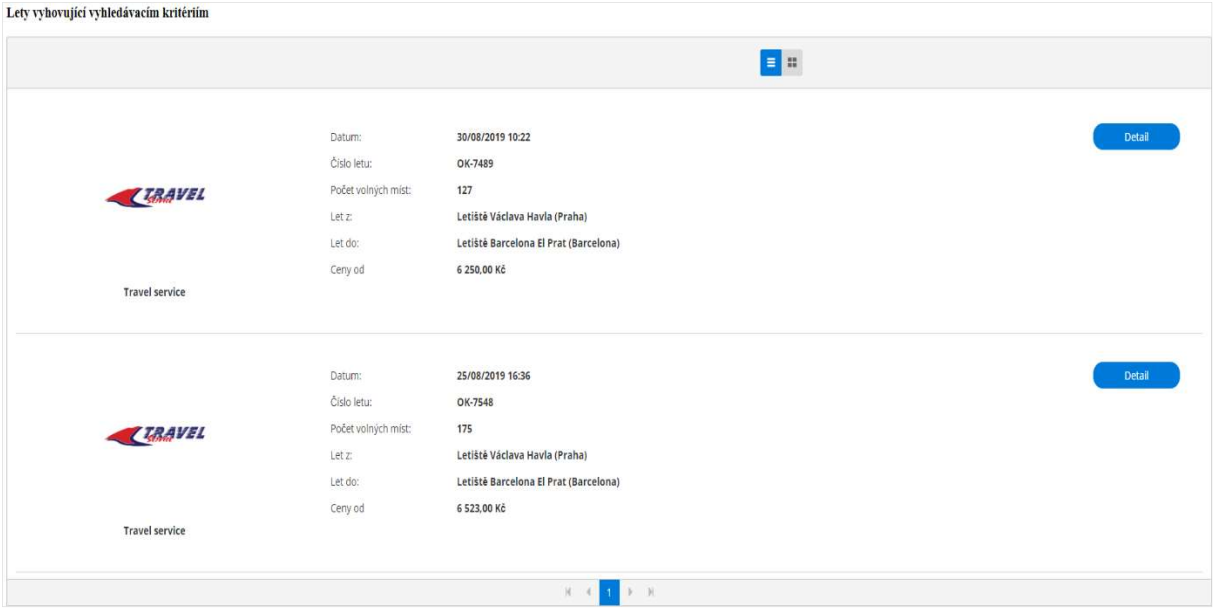
Obrázek 22 – Domovská stránka

Vyhledávání letů slouží uživatelům k nalezení letu podle zadaných parametrů. Mezi tyto parametry patří odkud a kam se má let uskutečnit a datumové rozmezí, ve kterém se mají lety vyhledat. Po vyplnění údajů už stačí jen kliknout na tlačítko *Hledat* a proběhne samotné

vyhledávání. Pokud nejsou vyplněné všechny parametry nebo je některý z parametrů vyplněn chybně, tak k vyhledávání nedojde a uživatel je informován o konkrétní chybě.

Cílem práce není vytvořit komplexní vyhledávač letenek, ale rezervační systém. Vyhledávání tedy funguje pouze z bodu *A* do bodu *B*, ale díky použitým technologiím a architektuře je možné systém v budoucnu rozšířit například o datovou strukturu graf a aplikovat na ni algoritmy pro hledání nejkratší cesty.

Pokud zadaným kritériím nevyhovuje žádný let, zobrazí se informace o tom, že zadání neodpovídá žádný výsledek a je potřeba parametry změnit. V opačném případě dojde k přesměrování na další stránku s výpisem vyhledaných letů (viz obrázek 23). Pro každý let se evidují následující informace: datum odletu, číslo letu, celkový počet volných míst, místo odletu, místo příletu a cena za nejlevnější letenku.

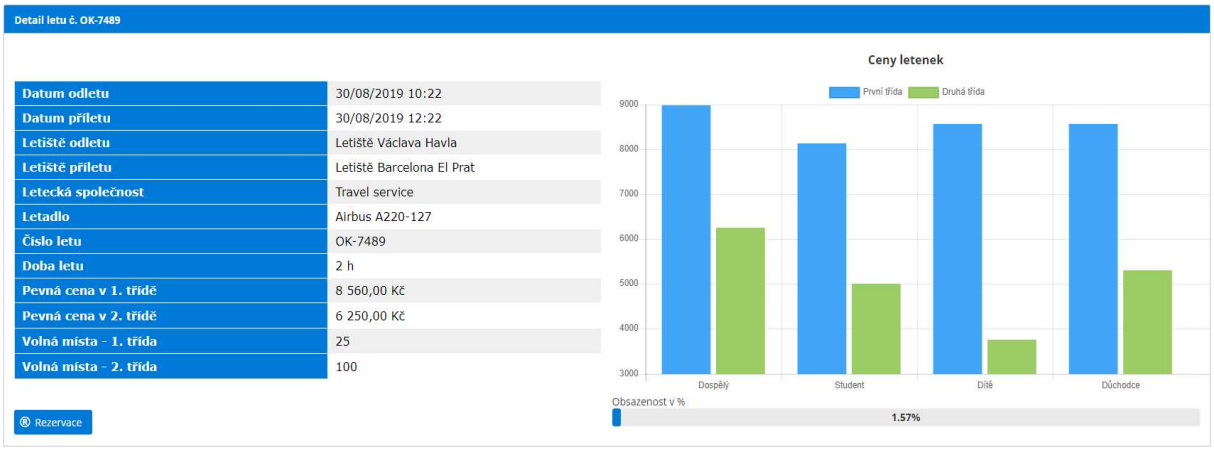


Obrázek 23 – Seznam vyhledaných letů

V případě, že je uživatel přihlášen a nechce let vyhledávat podle parametrů, může kliknout v hlavní liště na záložku *Seznam letů*. Zde jsou zobrazeny veškeré dostupné aktivní lety. Každý let obsahuje úplně stejné informace jako při využití funkce vyhledávání letů.

Pokud let uživatele zaujme, může si pomocí tlačítka *Detail* zobrazit podrobnější informace (viz obrázek 24), ze kterých se navíc dozví následující údaje: název letecké společnosti uskutečňující let, typ letadla, dobu letu, počet volných míst pro 1. a 2. třídu a cenu letenek v 1. a 2. třídě pro všechny dostupné kategorie (dítě, dospělý, student a senior). Přihlášený

uživatel může dále pokračovat v rezervaci (viz kapitola 6.1.4) pomocí stejnojmenného tlačítka umístěného v levém dolním rohu.

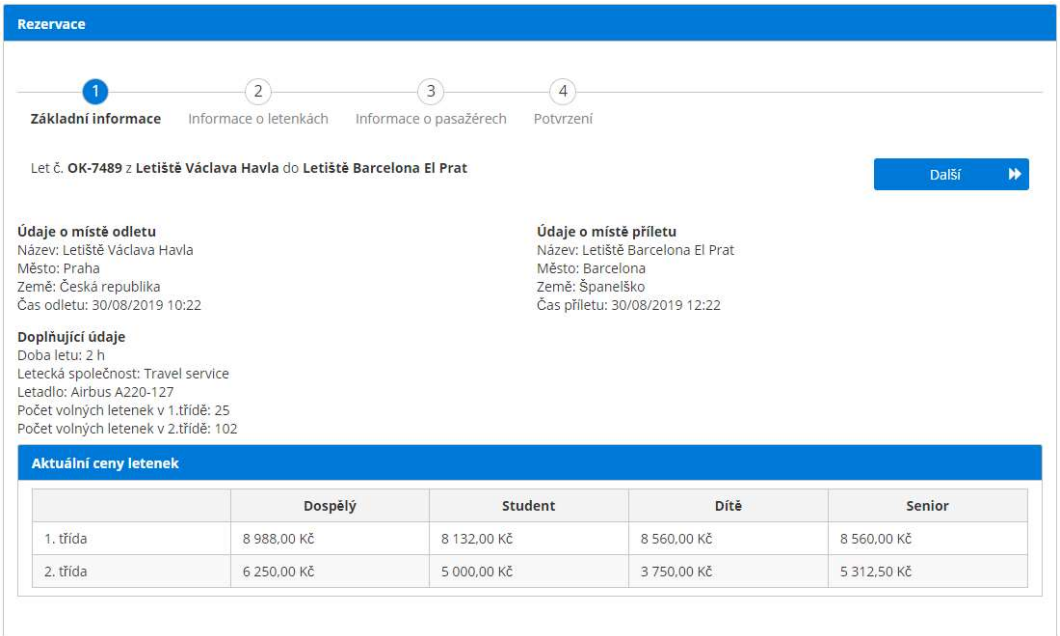


Obrázek 24 – Detail letu

6.1.4 Rezervace letenek

Dostupné pro role: zákazník, zástupce let. společnosti, administrátor

Po kliknutí na tlačítko *Rezervace* je uživatel přesměrován na stránku, kde probíhá nejdůležitější funkce aplikace – rezervace letenek. Ta je rozdělena do čtyř dílčích kroků, přičemž jednotlivé kroky jsou popsány dále.

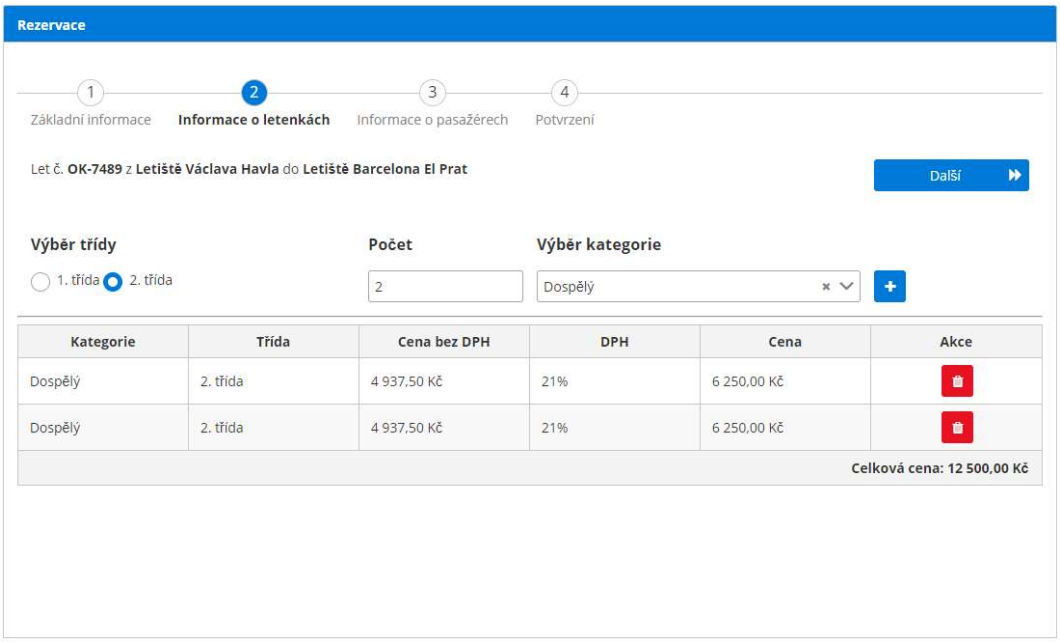


Obrázek 25 – Rezervace 1. krok

V prvním kroku (viz obrázek 25) se nachází shrnutí informací o daném letu. Konkrétně jsou to údaje o místě odletu/příletu (název letiště, město, země a čas odletu/příletu), počtu volných míst

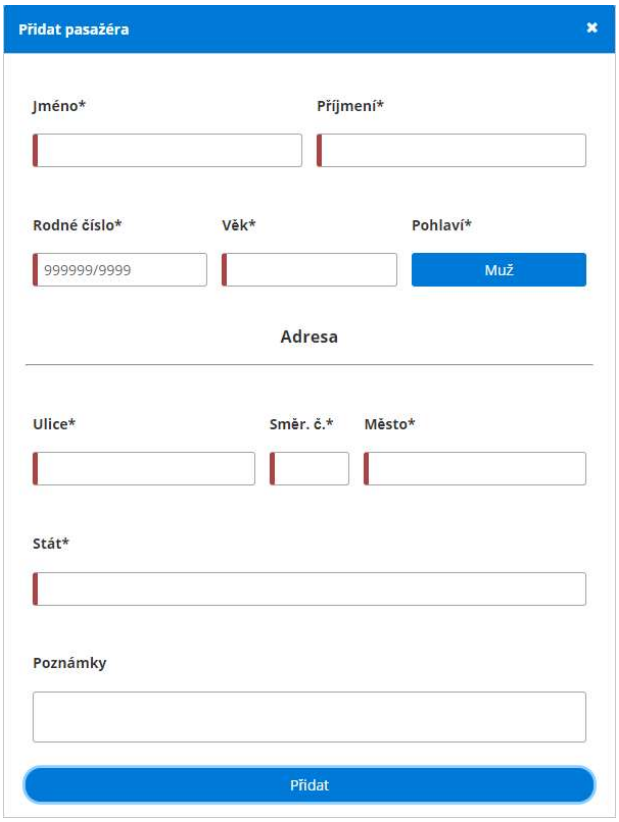
pro 1. a 2.třídu a ceně letenek pro všechny kategorie. V případě, že chce uživatel pokračovat v rezervaci, stiskne tlačítko *Další*.

Druhý krok slouží k výběru letenek (viz obrázek 26). Pro vybrání letenky musí uživatel zvolit třídu, počet letenek a z rozbalovacího menu vybrat kategorii. Následně po kliknutí na tlačítko přidat „+“ dojde ke vzniku seznamu s letenkami a s informacemi o jejich kategorii, třídě, ceně bez DPH, DPH a celkové ceně. Takto lze vybrat nejvýše pět letenek. V případě, že se uživatel splete, je možné letenku odstranit ve sloupci *Akce*. Pokud má již vybraný dostatečný počet letenek, může pokračovat na další krok.



Obrázek 26 – Rezervace 2. krok

Ve třetím kroku dochází k vytvoření seznamu cestujících. K tomu opět slouží tlačítko přidat „+“, po jehož stisknutí se zobrazí dialog pro přidání pasažéra (viz obrázek 27). Mezi povinné pole k vyplnění patří jméno, příjmení, rodné číslo, věk, pohlaví a adresa (ulice, poštovní směrovací číslo, město a stát). V případě potřeby je možné využít pole s poznámkami, kam lze dopsat doplňující informace o cestujícím. Veškerá pole jsou validována proti zadání neplatných vstupů stejně jako u ostatních formulářů. Pokud je formulář validní, dojde obdobně jako u letenek k vytvoření seznamu, ale tentokrát s pasažéry. Po přidání všech zamýšlených cestujících může uživatel pokračovat k závěrečnému kroku.



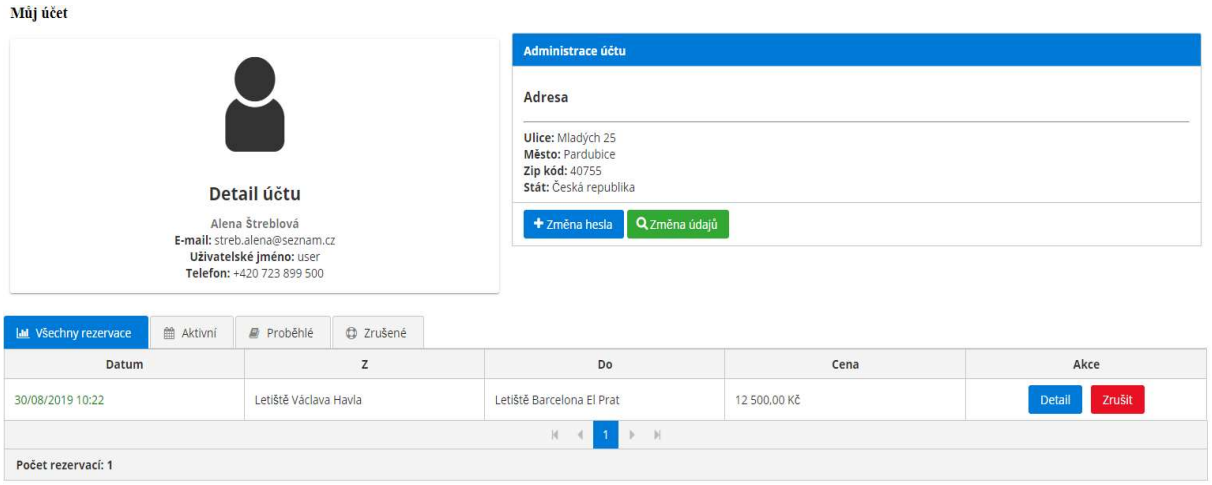
Obrázek 27 – Formulář pro přidání pasažéra

V posledním kroku nalezne uživatel shrnutí celého průběhu rezervace. Dozví se zde informace o daném letu, celkové ceně za letenky, počtu letenek a pasažérů. Počet pasažérů a letenek musí být shodný, jinak rezervace nemůže proběhnout. Po kontrole údajů a kliknutí na tlačítko *Rezervace* dojde k samotné rezervaci letenek. Všechny své rezervace nalezne uživatel v záložce *Můj účet* (viz následující kapitola 6.1.5), na který je v případě úspěchu přesměrován.

6.1.5 Můj účet

Dostupné pro role: zákazník, zástupce let. společnosti, administrátor

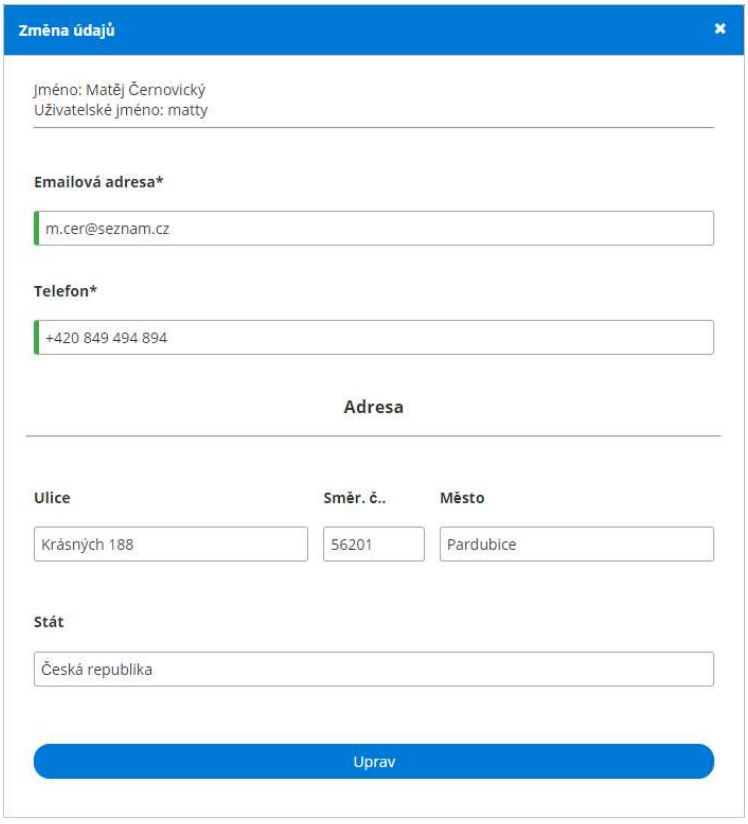
Každý přihlášený uživatel, ať už zákazník, zástupce letecké společnosti nebo administrátor, má k dispozici záložku *Můj účet* v hlavní liště aplikace. Všechny vyjmenované role vidí v této záložce své osobní informace (viz obrázek 28). Jedná se o jméno, email, uživatelské jméno, telefon a adresu (ulice, město, směrovací číslo a stát). Kromě toho jsou pro uživatele k dispozici funkce pro změnu hesla a změnu údajů.



Obrázek 28 – Můj účet

Při kliknutí na tlačítko *Změna hesla* dojde k zobrazení formuláře, do kterého je zapotřebí zadat své staré a nové heslo. Poté už jen stačí potvrdit požadavek kliknutím na tlačítko *Ano*.

Pokud bude chtít uživatel změnit své údaje, tak musí kliknout na tlačítko *Změna údajů*. Po kliknutí se zobrazí modální okno (viz obrázek 29), kde je možné změnit email, telefon a adresu (ulice, směrovací číslo, město a stát). Změnu údajů stačí následně potvrdit stisknutím tlačítka *Upravit*.

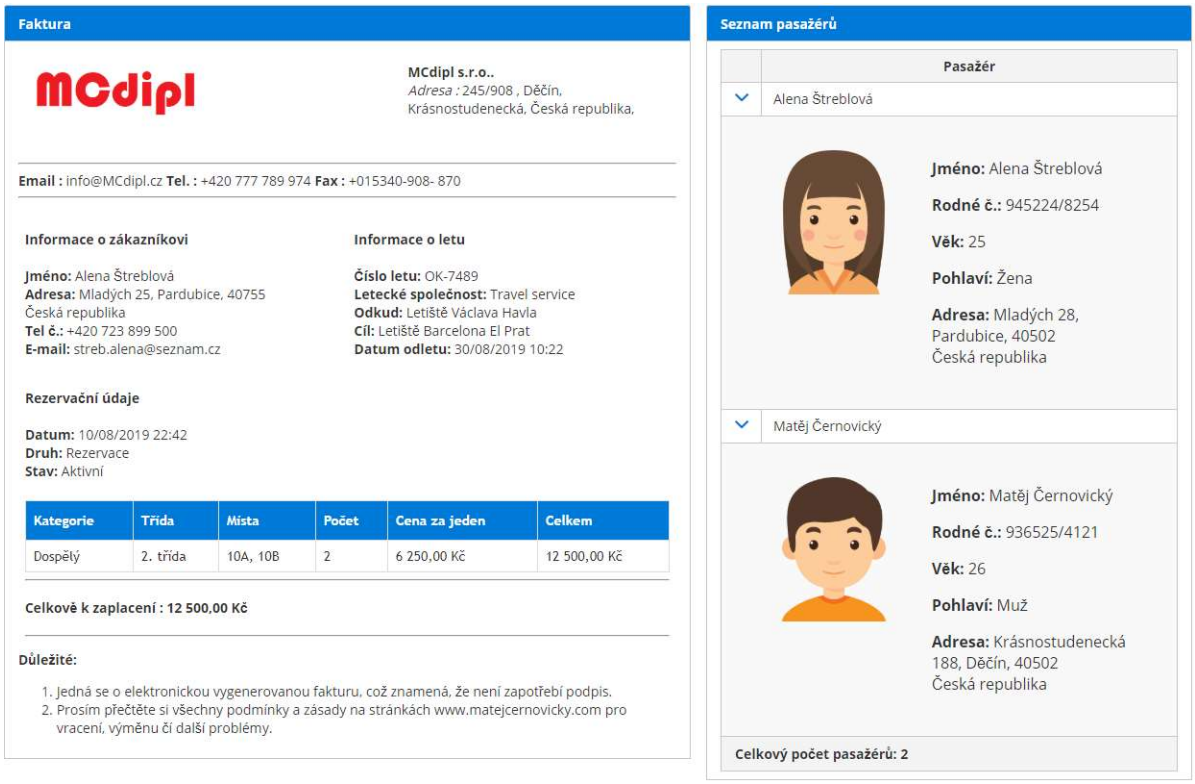


Obrázek 29 – Formulář pro změnu údajů

Nastane-li situace, kdy bude uživatel nespokojený se svým jménem, například pokud by se při registraci omylem přepsal nebo by slečna po vstupu do manželství změnila své příjmení, může kontaktovat administrátora aplikace, který je schopný v administraci jméno a příjmení upravit.

Dále je v záložce *Můj účet* zobrazen seznam všech rezervací uživatele (viz obrázek 28). Rezervace jsou rozděleny do záložek: *Všechny rezervace, Aktivní, Proběhlé* a *Zrušené*. V záložce *Všechny rezervace*, jak název napovídá, jsou zobrazeny veškeré rezervace, které jsou od sebe snadno rozpoznatelné barvou datumu (zelené – aktivní rezervace, modré – proběhlé rezervace, červené – zrušené rezervace).

Každá jednotlivá rezervace obsahuje informace o datumu letu, odkud a kam je let uskutečněn a ceně letu. Pokud by tyto údaje nebyly pro uživatele dostačující, může si pro každou rezervaci zobrazit její detail (viz obrázek 30). V rámci detailu nalezne uživatel kontaktní údaje na firmu, která je za rezervace zodpovědná, dále informace o zákazníkovi, informace o letu, rezervační údaje, celkovou částku k zaplacení, důležité poznámky a seznam pasažérů, které uživatel přidal při rezervaci. Pokud se jedná o aktivní rezervaci, je možné navíc danou rezervaci zrušit. Administrátor může zrušené rezervace ze systému kompletně odstranit.

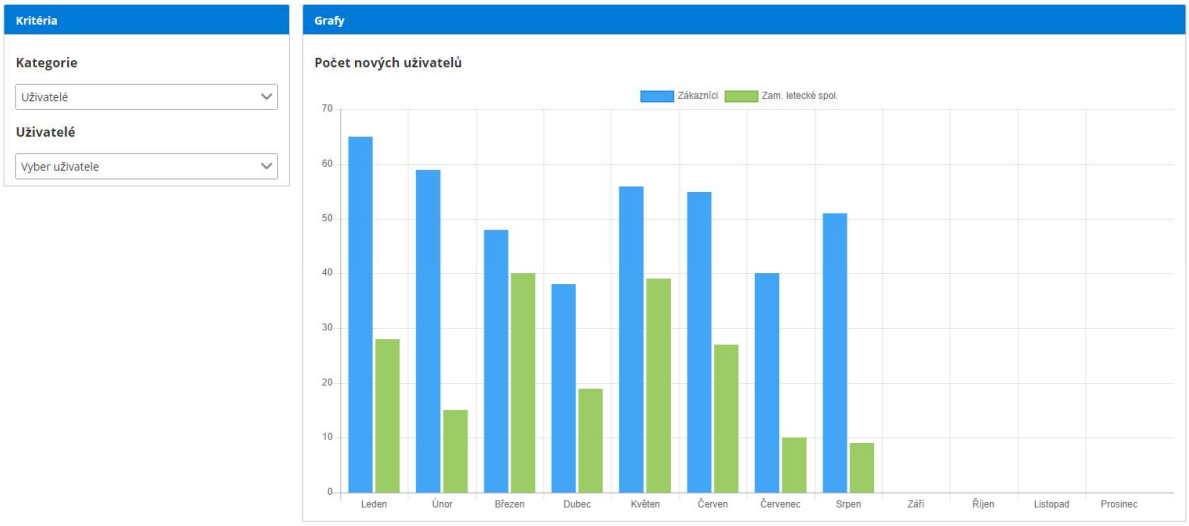


Obrázek 30 – Detail zarezervovaného letu

6.1.6 Statistiky

Dostupné pro role: zástupce let. společnosti, administrátor

Statistiky lze nalézt v hlavní liště aplikace. Po přesměrování na tuto stránku jsou uživateli zpřístupněny různé statistické informace zpracované ve formě grafů (obrázek 31). Jedná se např. o zobrazení celkové rezervovanosti letů, pro roli administrátora je uvedena i statistika jednotlivých uživatelů (kolik daný uživatel zarezervoval letů) atd.



Obrázek 31 – Statistika počtu nových uživatelů za jednotlivé měsíce

6.1.7 Administrace aplikace

Kromě správy letů se o veškerou administraci stará administrátor aplikace.

6.1.7.1 Správa letů

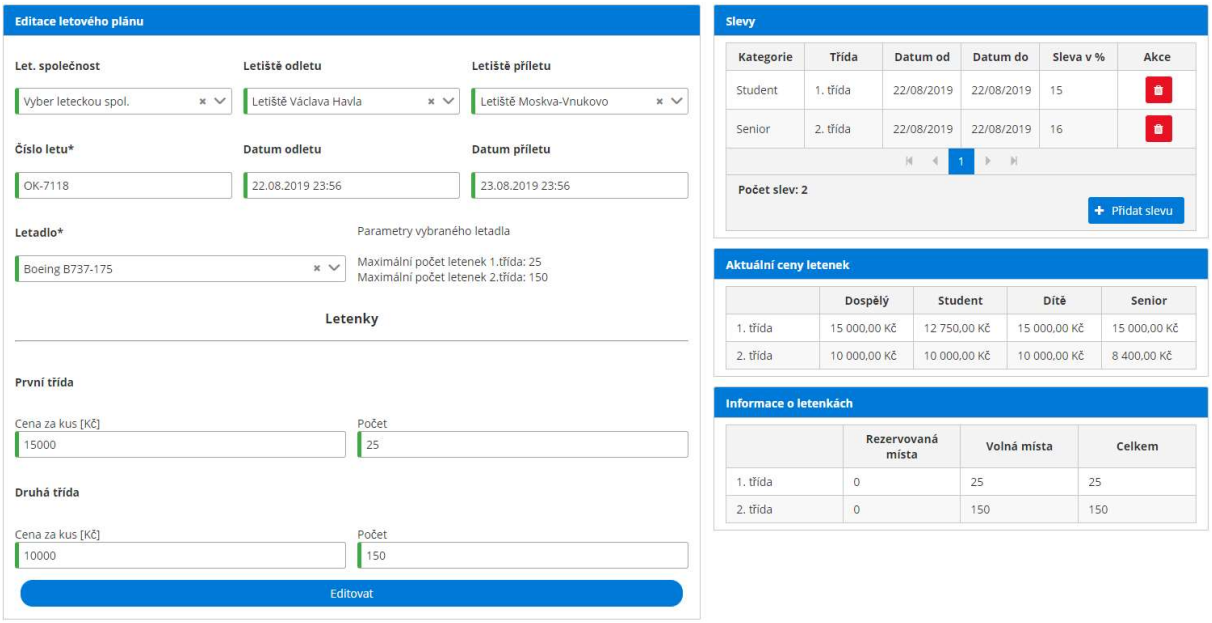
Dostupné pro role: zástupce let. společnosti, administrátor

Pod správu letů spadá vytváření nových letových plánů, editování již vytvořených a případně také jejich odstranění. Tuto funkci nalezne každá z uvedených rolí na odlišném místě v aplikaci. Zástupce letecké společnosti ji má dostupnou v hlavní liště přes záložku *Firemní lety*. Administrátor se k této funkci dostane skrze záložku *Administrace*, která obsahuje rozbalovací seznam a v ní položku *Všechny lety*. Zástupce letecké společnosti vidí pouze lety společnosti, za kterou je zavedený, kdežto administrátor vidí všechny.

Seznam vytvořených letů obsahuje záložky *Všechny lety, Aktivní a Proběhlé*. Rozřazení letů do těchto záložek probíhá samozřejmě podle datumu letu. Každý let obsahuje informace jako letiště odletu, letiště příletu, typ letadla, číslo letu, datum letu a dobu letu. Dále obsahuje informace o 1. a 2. třídě jako je cena, počet rezervovaných a volných letenek.

Editovat či smazat jdou pouze aktivní lety. Lze to provést tak, že uživatel klikne na konkrétní let, čímž dojde k zobrazení dialogového okna s volbou, zda chce let upravit nebo smazat. Po kliknutí na tlačítko *Smazat* dojde ke zrušení letu, ale pouze v případě, že na tento let zatím nedošlo k zarezervování letenky. Pokud už jsou nějaké letenky zarezervované, objeví se vyskakovací okno s informací, že daný let nelze smazat.

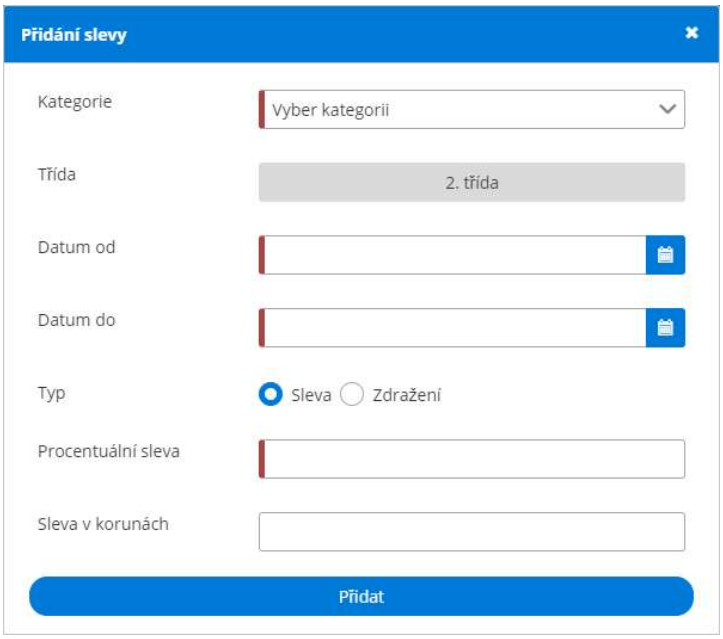
V případě kliknutí na tlačítko *Upravit* dojde k zobrazení formuláře pro editaci letového plánu, případně slev (viz obrázek 32). Po kliknutí na tlačítko *Editovat* dojde k aktualizaci dat letu. Při chybném nebo nedostatečném vyplnění údajů k úpravě letu nedojde a uživatel bude upozorněn na vyskytující se problém. Jestliže už je zarezervována i jediná letenka, položky pro editaci letového plánu nelze přepsat.



Obrázek 32 – Úprava vytvořeného letu

Dále je možné v úpravě aktivních letů přidávat slevy. Na rozdíl od editace letu lze slevu vytvořit i pro let, ve kterém už jsou zarezervované letenky. Kliknutím na tlačítko *Přidat slevu* v pravém dolním rohu tabulky *Slevy* dojde k zobrazení dialogu s formulářem (viz obrázek 33). Tento formulář obsahuje kategorii s rozbalovacím seznamem (dítě, dospělý, student, senior), kalendář pro výběr datumu od a datumu do, volbu třídy a výběr, zda se jedná o slevu nebo zdražení. Slevu, resp. zdražení, je možné přidat buď procentuálně, nebo přímo v korunách. Pokud jsou veškeré údaje vyplněny správně, dojde k vytvoření slevy a přidání do tabulky slev. Je-li aktuální den v datumovém rozmezí přidané slevy, tak se změna obratem promítne do ceny letenek. V případě špatného nebo nedostatečného vyplnění polí bude uživatel informován o chybě. Chyba také nastane, pokud je přidána sleva, která svým datumovým rozpětím, kategorií a třídou

zasahuje do některé již dříve vytvořené. Pokud už sleva z nějakého důvodu uživateli nevyhovuje, je možné ji odstranit pomocí tlačítka ve sloupci *Akce*.



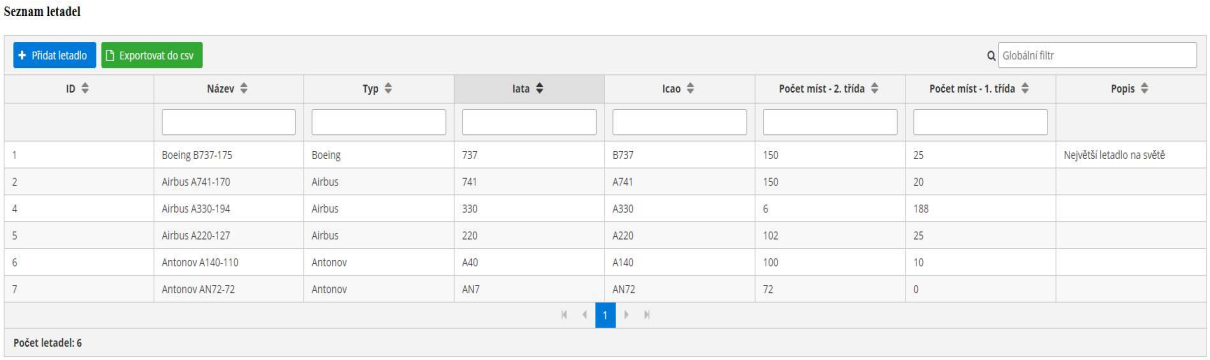
Obrázek 33 – Formulář pro přidání slevy

Pomocí tlačítka *Přidat let*, které je umístěno vlevo pod záložkou *Všechny lety*, dojde k přesměrování na stránku s formulářem pro vytvoření letu. Struktura formuláře je stejná jako v případě výše popsané editace (viz obrázek 32), tudíž je povinné vyplnit všechny položky. Po správném zadání všech údajů dojde k vytvoření letu, který se obratem zobrazí uživatelům v seznamu.

6.1.7.2 Správa uživatelů, regionálních informací, letadel a leteckých společností

Dostupné pro role: administrátor

Administrátor nalezne všechny tyto vyjmenované položky v hlavní liště aplikace v záložkách *Administrace* a *Správa uživatelů*. Po kliknutí na jakoukoliv položku dojde k přesměrování na novou stránku, kde se nachází tabulka s příslušným seznamem. Každá tabulka obsahuje v hlavičce tlačítka pro přidání a export do csv (viz obrázek 34). Po stisknutí tlačítka *Přidat* dojde k zobrazení dialogového okna s formulářem (např. formulář pro vytvoření nového letadla). Veškeré vstupy jsou validovány. Pro rychlejší vyhledávání informací v tabulkách lze využít pomocné funkce. Patří mezi ně řazení sloupců, stránkování či filtrování dat. Filtry jsou umístěny nad jednotlivými sloupci nebo každá tabulka obsahuje i globální filtr.



Obrázek 34 – Seznam letadel

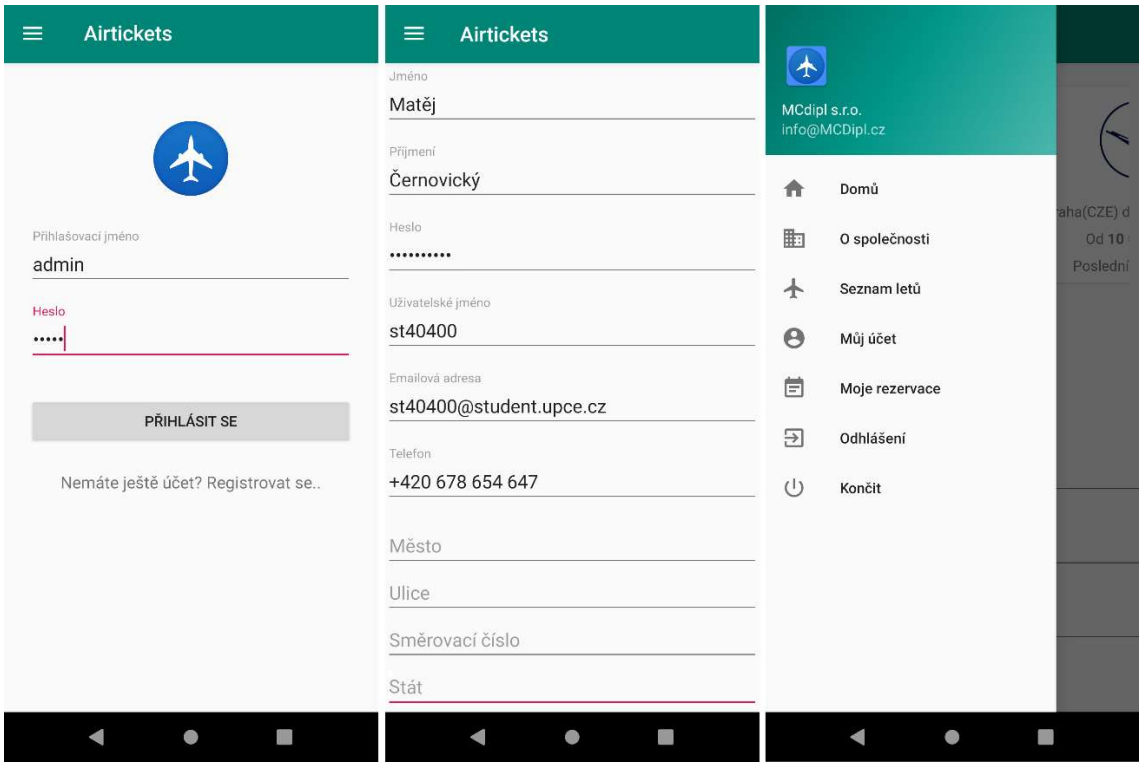
Dále může administrátor všechna zobrazená data upravit nebo odstranit. Tyto funkce lze nalézt při kliknutí na konkrétní řádek, přičemž následně dojde k otevření dialogového okna. V dialogu se nachází detail vybraného řádku s možností jeho editace. Pro úpravu dat stačí kliknout na tlačítko *Upravit* (například pro seznam měst lze editovat název a stát). Pokud chce administrátor daný řádek pouze odstranit, stiskne tlačítko *Smazat*. V případě chyby, např. integritní omezení, bude upozorněn.

6.2 Mobilní aplikace

Mobilní aplikace má k dispozici, kromě administrace a statistik, úplně stejné funkce jako aplikace webová. Z tohoto důvodu zde nebudou popsány úplně dopodrobna, ale spíše bude znázorněno v čem se liší jejich obsluha.

Při spuštění aplikace dojde k zobrazení domovské stránky, na které lze najít menu, vybrané lety a funkci vyhledávání letů (viz obrázek 36). Vybrané lety a vyhledávání letů byly popsány už ve webové aplikaci (viz kapitola 6.1.3) a jejich funkcionalita je stejná. Po rozkliknutí menu se zobrazí navigace s položkami (viz obrázek 35), které se liší pro nepřihlášeného a přihlášeného uživatele. Nepřihlášený uživatel má k dispozici pouze položky *Domů, O společnosti, Přihlášení* a *Končit*. Přihlášený uživatel v menu nalezne navíc *Seznam letů, Můj účet, Moje rezervace* a místo *Přihlášení* položku *Odhlášení*.

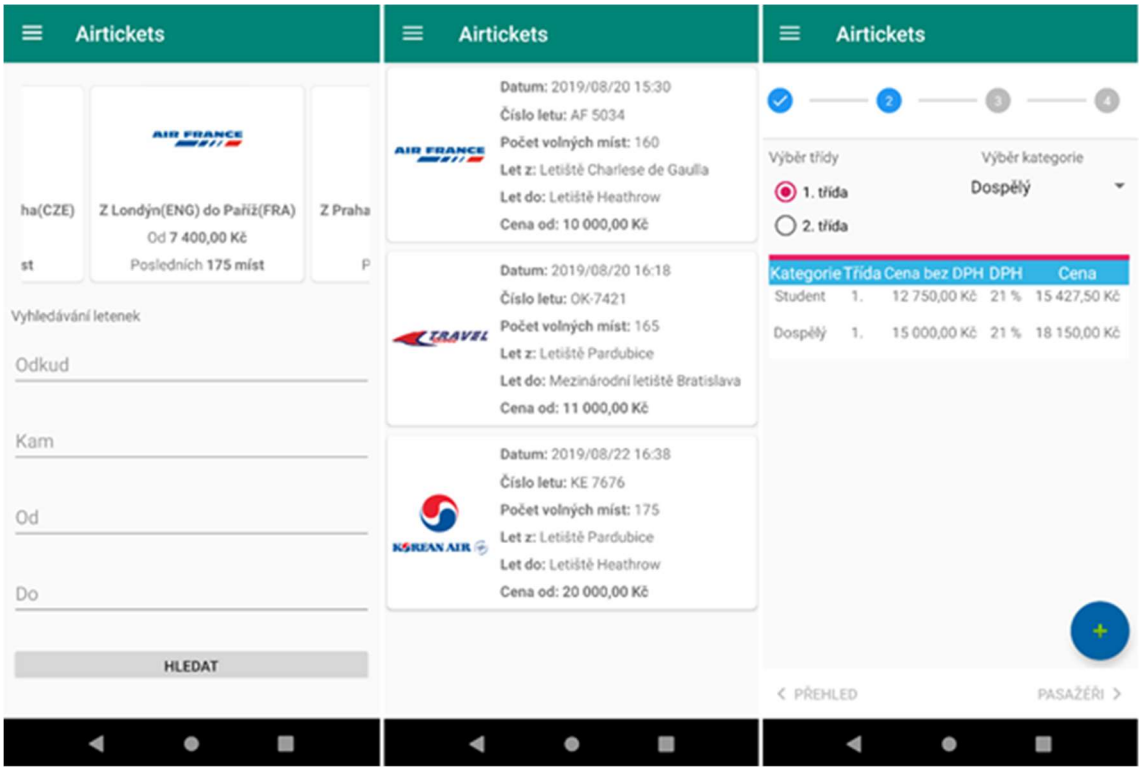
Formulář pro přihlášení (viz obrázek 35) nalezne uživatel v menu po kliknutí na položku *Přihlášení*. Pokud uživatel nemá ještě vytvořený účet, může se zaregistrovat. Odkaz na registrační formulář je umístěn pod přihlašovacími údaji. Nový účet lze vytvořit, stejně jako u webové aplikace (viz kapitola 6.1.2), pro zákazníka nebo zástupce letecké společnosti (viz obrázek 35). K odhlášení musí uživatel kliknout v menu na *Odhlášení* a po odsouhlasení dojde k přesměrování zpět na domovskou stránku.



Obrázek 35 – Přihlášení, registrace a navigační menu mobilní aplikace

Pro zobrazení detailu konkrétního letu postačí kliknout přímo na daný let, čímž dojde k přesměrování na novou stranu s podrobnějšími informacemi, kde je možné let i zarezervovat. Rezervace probíhá stejně jako u webové aplikace ve čtyřech krocích (viz kapitola 6.1.4). V prvním kroku nalezne uživatel souhrn informací o daném letu a na další krok se dostane zmáčknutím tlačítka *Letenky.* Ve druhém kroku (viz obrázek 36) může uživatel přidávat letenky stejně jako u webové aplikace, jen odstranění letenky je odlišné. Pro odstranění neexistuje tlačítko, ale stačí daný řádek s letenkou posunout do strany, čímž dojde k odstranění. V případě, že si uživatel nevybere žádnou letenku, tak nemůže přejít na třetí krok rezervace. Ten slouží k přidávání pasažérů. Pokud není vybrán ani jeden pasažér, nemůže uživatel pokračovat k poslednímu kroku, ve kterém stačí rezervaci už jen potvrdit. Po potvrzení rezervace je uživatel přesměrován zpět na domovskou stránku aplikace.

Na vytvořené rezervace se může uživatel dostat dvěma způsoby. Buď v menu přes položku *Moje rezervace,* nebo ve správě svého účtu přes tlačítko *Rezervace*. Kliknutím na danou rezervaci je možné zobrazení detailu, který informuje zákazníka o rezervačních údajích (datum letu, druh a stav rezervace), seznamu pasažérů, seznamu letenek a celkové ceně k zaplacení. Aktivní rezervaci je možné zrušit posunutím dané rezervace do strany.



Obrázek 36 – Domovská stránka, seznam letů a rezervace 2. krok

Položka *Domů* v menu slouží, jak už název napovídá, k navrácení na domovskou stránku. Pro ukončení aplikace slouží položka *Končit*.

ZÁVĚR

Cíle teoretické i praktické části diplomové práce se podařilo splnit v plném rozsahu. První část práce byla věnována úvodu do rezervačních systémů. Dále byl naznačen princip fungování přenosového protokolu HTTP. Protože novější verze protokolu nejsou zatím tak rozšířené, byla popsána verze HTTP/1.1. V návaznosti na tuto kapitolu byly charakterizovány nejpoužívanější webové služby současnosti včetně jejich závěrečného porovnání. V další části práce byla provedena rešerše webových technologií a frameworků. Konkrétně byly vybrány čtyři technologie (Spring Framework, ASP.NET, React a Angular) na základě průzkumu provedeného webem Stack Overflow. Následně byly porovnány s ohledem na jejich využitelnost v kombinaci s webovými službami. Kapitola byla věnována také popisu technologií a nástrojů použitých při návrhu a vývoji rezervačního systému. V jejich výčtu nejsou obsažené úplně všechny technologie, což z důvodu jejich počtu nebylo možné učinit, ale ty nejdůležitější ano.

V praktické části diplomové práce byl analyzován a implementován systém pro rezervaci letenek. Nejprve byla provedena analýza funkčních a nefunkčních požadavků. Dále byly vytvořeny UML diagramy pro lepší pochopení a orientaci v systému. Kromě jiného bylo znázorněno i databázové schéma pomocí E-R diagramu. Poté následovala část o vlastní implementaci systému, kde byly uvedeny a rozebrány jeho klíčové části. V závěru diplomové práce byly podrobně popsány funkcionality obou klientských aplikací.

Aplikace splňují definované požadavky a umožňují rezervaci letenek na vybrané lety. Jelikož se jedná o funkční prototyp, nemusí být vše zcela odladěné, ale nemělo by se jednat o klíčové části systému. Během realizace systému byly nalezeny nové funkcionality, které by se do budoucna mohly přidat. To by vzhledem k použitým technologiím neměl být problém. Případným vylepšením by mohlo být například vytvoření komplexního vyhledávače letenek, který by se opíral o datovou strukturu graf a využíval algoritmy pro hledání nejkratší cesty. Kromě toho je možné přidat i rezervace zpátečních letů nebo letů s přestupy. Zdokonalení by určitě sneslo i grafické pojetí aplikací.

Práce pro mě byla obrovským přínosem. Teprve při jejím vytváření jsem si uvědomil, co vše se skrývá pod tvorbou rozsáhlejších systémů. Získané zkušenosti a vědomosti určitě využiji ve svém dosavadním zaměstnání.

Diplomová práce může sloužit též jako návod (podklad) pro vývoj komplexního systému založeného na nejnovějších technologiích.

POUŽITÁ LITERATURA

[1] VYSTOUPIL, Jiří, ŠAUER, Martin, HOLEŠINSKÁ, Andrea, METELKOVÁ, Petra. Informační a rezervační systémy. *Informační a rezervační technologie v cestovním ruchu* [online]. Brno: Ekonomicko-správní fakulta MU, 2005 [cit. 25.5.2019]. Dostupné z: http://cgi.math.muni.cz/kriz/prevod/info5.html.

[2] *Sabre* [online]. SABRE GLBL INC., © 2003-2019 [cit. 25.5.2019]. Dostupné z: https://www.sabre.com/.

[3] *Amadeus.com* [online]. Amadeus IT Group SA, [cit. 25.5.2019]. Dostupné z: https://amadeus.com/en.

[4] *Travelport* [online]. Travelport, © 2019 [cit. 25.5.2019]. Dostupné z: https://www.travelport.com/.

[5] BARTEN, Martijn. *Revfine.com* [online]. Revfine.com, © 2019 [cit. 25.5.2019]. Dostupné z: https://www.revfine.com/.

[6] Amadeus – A Central Reservation Systém. *Complete Travel Technology Solutions* [online]. travelOTAs, ©2019, 25.2.2015 [cit. 25.5.2019]. Dostupné z: http://www.travelotas.com/amadeus-central-reservation-system/.

[7] FIELDING, Roy T. a kolektiv. Hypertext Transfer Protocol -- HTTP/1.1. *IETF Tools* [online]. 1999 [cit. 29.5.2019]. Dostupné z: https://tools.ietf.org/html/rfc2616.

[8] Usage statistics of HTTP/2 for websites. *W3Techs* [online]. Q-Success, © 2009-2019, 29.5.2019 [cit. 29.5.2019]. Dostupné z: https://w3techs.com/technologies/details/ce-http2/all/all.

[9] MICHÁLEK, Martin. Rychlý protokol HTTP/2: S nasazením na weby na nic nečekejte. *Vzhůru dolů* [online]. 1.1.2019 [cit. 10.6.2019]. Dostupné z: https://www.vzhurudolu.cz/prirucka/http-2.

[10] JANOVSKÝ, Dušan. Přednačítání přes HTTP 2 server push. *Jak psát web* [online]. [cit. 17.7.2019]. ISSN 1801-0458. Dostupné z: https://www.jakpsatweb.cz/server/http-2-server-push.html.

[11] SATRAPA, Pavel. Jak funguje nový protokol HTTP/2. *Root.cz* [online]. Internet Info, s.r.o., © 1998-2019, 4.3.2015 [cit. 10.6.2019]. ISSN 1212-8309. Dostupné z: https://www.root.cz/clanky/jak-funguje-novy-protokol-http-2.

[12] ŠURKALA, Milan. HTTP/3 nebude používat TCP, přejde na protokol QUIC. *Svět hardware* [online]. oXy Online s.r.o., © 1998-2019, 14.11.2018 [cit. 10.6.2019]. ISSN 1213-0818. Dostupné z: https://www.svethardware.cz/http-3-nebude-pouzivat-tcp-prejde-na-protokol-quic/47992.

[13] KRČMÁŘ, Petr. HTTP/3 nebude postavené na TCP, základem bude QUIC používající UDP. *Root.cz* [online]. Internet Info, s.r.o., © 1998-2019, 14.11.2018 [cit. 10.6.2019]. ISSN 1212-8309. Dostupné z: https://www.root.cz/clanky/http-3-nebude-postavene-na-tcp-zakladem-bude-quic-pouzivajici-udp/.

[14] KUBA, Martin. Webové služby [přednáška]. Brno: Fakulta Informatiky MU, 6.12.2004. In: *Informační systém MU* [online]. [cit. 28.6.2019]. Dostupné z: https://is.muni.cz/el/1433/podzim2004/PA165/um/prednaska11/index.html.

[15] KOSEK, Jiří. Využití webových služeb a protokolu SOAP při komunikaci. *kosek.cz* [online]. Jiří Kosek, © 1999-2018 [cit. 28.6.2019]. Dostupné z: https://www.kosek.cz/diplomka/html/websluzby.html.

[16] *Webové služby nad IS/STAG* [online]. Oddělení podpory IS, Fakulta elektrotechniky a informatiky UPa. ©2013-2019 [cit. 28.6.2019]. Dostupné z: https://stag-ws.upce.cz/ws/services/soap/znamky?wsdl.

[17] MALÝ, Martin. REST: architektura pro webové API. *Zdroják* [online]. Devel.cz Lab s.r.o, 3.8.2009 [cit. 17.7.2019]. ISSN 1803-5620. Dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api.

[18] HORDĚJČUK, Vojtěch. REST. v*oho* [online]. Vojtěch Hordějčuk, © 2008-2019 [cit. 17.7.2019]. Dostupné z: http://voho.eu/wiki/rest.

[19] FIELDING, Roy T. *Architectural Styles and the Design of Network-based Software Architectures*. Irvine, 2000. Disertační práce. University of California, Irvine.

[20] Sending form data. *MDN Web Docs* [online]. Mozilla and individual contributors, © 2005-2019 [cit 17.7.2019]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Sending_and_retrieving_form_data.

[21] *GraphQL* [online]. The GraphQL Foundation, © 2019 [cit. 17.7.2019]. Dostupné z: https://graphql.org/.

[22] BURK, Nikolas. GraphQL SDL– Schema Definition Language. *Prisma* [online]. Prisma, © 2019 [cit. 17.7.2019]. Dostupné z: https://www.prisma.io/blog/graphql-sdl-schema-definition-language-6755bcb9ce51.

[23] GraphQL API v4. *GitHub Developer* [online]. GitHub Inc., © 2019 [cit. 17.7.2019]. Dostupné z: https://developer.github.com/v4.

[24] The Ultimate Guide to API Architecture: REST, SOAP or GraphQL?. *DA-14 – Web and Mobile App Development Company* [online]. DA-14, © 2019, 7.2.2018 [cit. 17.7.2019]. Dostupné z: https://da-14.com/blog/ultimate-guide-api-architecture-rest-soap-or-graphql/.

[25] *Google Trends* [online]. Google, © 2019 [cit. 17.7.2019]. Dostupné z: https://trends.google.cz/trends/explore?cat=5&date=2014-07-13%202019-07-13&q=%2Fm%2F077dn,rest%20API,%2Fg%2F11cn3w0w9t.

[26] Stack Overflow Annual Developer Survey. *Stack Overflow* [online]. Stack Exchange Inc, © 2019 [cit. 17.7.2019]. Dostupné z: https://insights.stackoverflow.com/survey.

[27] Overview of Spring Framework. *Spring* [online]. Pivotal Software, Inc., © 2019, 14.6.2017 [cit. 17.7.2019]. Dostupné z: https://docs.spring.io/spring/docs/5.0.0.RC2/spring-framework-reference/overview.html.

[28] JOHNSON, Rod. Spring Framework: The Origins of a Project and a Name. *Spring* [online]. Pivotal Software, Inc., © 2019, 9.11.2006 [cit. 17.7.2019] Dostupné z: https://spring.io/blog/2006/11/09/spring-framework-the-origins-of-a-project-and-a-name.

[29] KUNČAR, Petr. Spring - IoC Kontejner. *ITnetwork.cz* [online]. itnetwork.cz, © 2019 [cit. 17.7.2019]. Dostupné z: https://www.itnetwork.cz/java/pokrocile/spring-ioc-kontejner.

[30] Learn ASP.NET. *Microsoft* [online]. Microsoft, © 2019 [cit. 17.7.2019]. Dostupné z: https://dotnet.microsoft.com/learn/web.

[31] ČÁPKA, David. Lekce 1 - Úvod do ASP.NET. *ITnetwork.cz* [online]. itnetwork.cz, © 2019 [cit. 17.7.2019]. Dostupné z: https://www.itnetwork.cz/csharp/asp-net/tutorial-uvod-do-asp-dot-net.

[32] HERCEG, Tomáš. Úvod do .NET frameworku. *dotNETportal.cz* [online]. Tomáš Herceg, Tomáš Jecha, © 2019, 3.4.2009 [cit. 17.7.2019]. Dostupné z: https://www.dotnetportal.cz/clanek/125/Uvod-do-NET-Frameworku.

[33] LANDER, Richard. Introducing .NET 5. *Microsoft* [online]. Microsoft, © 2019, 6.5.2019 [cit. 17.7.2019]. Dostupné z: https://devblogs.microsoft.com/dotnet/introducing-net-5/.

[34] HERCEG, Tomáš. Build, .NET core a na co se můžeme těšit. *dotNETportal.cz* [online]. Tomáš Herceg, Tomáš Jecha, © 2019, 7.5.2019 [cit. 17.7.2019]. Dostupné z: https://www.dotnetportal.cz/blogy/3/Tomas-Herceg/8571/Build-NET-Core-a-na-co-se-muzeme-tesit.

[35] Getting Started. *React – A JavaScript library for building user interfaces* [online]. Facebook Inc., © 2019 [cit. 17.7.2019]. Dostupné z: https://reactjs.org/docs/getting-started.html.

[36] MIKŠŮ, Vojtěch. React - Úvod. *DžejEs - JavaScript pro web* [online]. 2016 [cit. 17.7.2019]. Dostupné z: https://www.dzejes.cz/react-uvod.html.

[37] MÁCA, Jindřich. Lekce 1 - Úvod do React. *ITnetwork.cz* [online]. itnetwork.cz, © 2019 [cit. 17.7.2019]. Dostupné z: https://www.itnetwork.cz/javascript/react/uvod-do-react.

[38] React.js History. *Education Ecosystem* [online]. Education Ecosystem, © 2019 [cit. 17.7.2019]. Dostupné z: https://www.education-ecosystem.com/guides/programming/react-js/history.

[39] HAMEDANI, Mosh. React Virtual DOM Explained in Simple English. *Programming with Mosh* [online]. © 2015, 3.12.2018 [cit. 17.7.2019]. Dostupné z: https://programmingwithmosh.com/react/react-virtual-dom-explained/.

[40] Angular - Architecture overview. *Angular* [online]. Google, © 2010-2019 [cit. 17.7.2019]. Dostupné z: https://angular.io/guide/architecture.

[41] SINGH, Anil. What Is Architecture Overview of Angular?. *Code sample* [online]. © 2017 [cit. 17.7.2019]. Dostupné z: https://www.code-sample.com/2018/01/angular-4-and-5-architecture-overview.html.

[42] LACKO, Ľuboslav. *Vývoj aplikací pro Android.* Brno: Computer Press, 2015, 472 s. ISBN: 978-80-251-4347-6.

[43] JUNEK, Pavel. Operační systém Android. *ITnetwork.cz* [online]. itnetwork.cz, © 2019 [cit. 17.7.2019]. Dostupné z: https://www.itnetwork.cz/mobilni-zarizeni/android-operacni-system-google.

[44] Historie mobilního operačního systému Android. *oTechnice.cz* [online]. oTechnice.cz, © 2019, 24.8.2017 [cit. 17.7.2019]. Dostupné z: https://otechnice.cz/historie-mobilniho-operacniho-systemu-android/.

[45] The Evolution of Android Over Years. *Mindster* [online]. Mindster, © 2018 [cit. 17.7.2019]. Dostupné z: https://mindster.in/blog/evolution-android.

[46] 18. Using the @SpringBootApplication Annotation. *Spring Boot Reference Guide* [online]. © 2012-2018 [cit. 17.7.2019]. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-using-springbootapplication-annotation.html.

[47] 78. Embedded Web Servers. *Spring Boot Reference Guide* [online]. © 2012-2018 [cit. 17.7.2019]. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/html/howto-embedded-web-servers.html.

[48] PrimeNG. *GitHub* [online]. GitHub, Inc., © 2019 [cit. 17.7.2019]. Dostupné z: https://github.com/primefaces/primeng.

[49] PrimeNG. P*rimeFaces* [online]. PrimeTek, ©2009-2019 [cit. 17.7.2019]. Dostupné z: https://www.primefaces.org/primeng/#/.

[50] MySQL Documentation. *MySQL* [online]. Oracle Corporation, © 2019 [cit. 17.7.2019]. Dostupné z: https://dev.mysql.com/doc/.

[51] PETEROVÁ, Alena. Seznámení s Hibernate ORM. *BCVlog* [online]. BCV solutions s.r.o., 16.4.2014 [cit.17.7.2019]. Dostupné z: https://blog.bcvsolutions.eu/seznameni-s-hibernate-orm/.

[52] OTTINGER, Joseph B., LINWOOD, Jeff, MINTER, Dave. *Beginning Hibernate: For Hibernate 5*. New York: Apress Media, 2016, 223 s. ISBN: 978-1-4842-2318-5.

[53] *Úvod do JSON* [online]. [cit. 17.7.2019]. Dostupné z: https://www.json.org/json-cz.html.

[54] HASSMAN, Martin. JSON: jednotný formát pro výměnu dat. *Zdroják* [online]. Zdroják.cz, 29.9.2008 [cit. 17.7.2019]. ISSN 1803-5620. Dostupné z: https://www.zdrojak.cz/clanky/json-jednotny-format-pro-vymenu-dat/.

[55] Welcome to Apache Maven. *Apache Maven Project* [online]. The Apache Software Foundation, ©2002-2019, 15.7.2019 [cit. 17.7.2019]. Dostupné z: http://maven.apache.org/index.html.

[56] *NPM* [online]. npm, Inc., [cit. 17.7.2019]. Dostupné z: https://www.npmjs.com/.

[57] MICHÁLEK, Martin. NPM: Průvodce začátky a základními příkazy. *Vzhůru dolů* [online]. 19.11.2018 [cit. 17.7.2019]. Dostupné z: https://www.vzhurudolu.cz/prirucka/npm.

[58] DEBILL, Erik. *Module Counts* [online]. [cit. 17.7.2019]. Dostupné z: http://www.modulecounts.com/.

[59] *Gradle Build Tool* [online]. Gradle Inc., © 2019 [cit. 17.7.2019]. Dostupné z: https://gradle.org/.

[60] KANAPARTHI, Vineeth. Git and GitHub in a NutShell. *Codeburst.io* [online]. 14.2.2018 [cit. 17.7.2019]. Dostupné z: https://codeburst.io/git-and-github-in-a-nutshell-b0a3cc06458f.

[61] What's a version control system?. *GitHub Guides* [online]. GitHub, Inc., © 2019, 25.9.2017 [cit. 17.7.2019]. Dostupné z: https://guides.github.com/introduction/git-handbook/.

[62] VALKOVIČ, Patrik. Lekce 1 - Git - Historie a principy. *ITnetwork.cz* [online]. itnetwork.cz, © 2019 [cit. 17.7.2019]. Dostupné z: https://www.itnetwork.cz/software/git/git-tutorial-historie-a-principy.

[63] Enterprise Architect. *Sparx Systems* [online]. Sparx Systems Pty Ltd., © 2000-2019 [cit.17.7.2019]. Dostupné z: https://sparxsystems.com/products/ea/index.html.

[64] IntelliJ IDEA. *Jet Brains* [online]. JetBrains s.r.o., © 2000-2019 [cit. 17.7.2019]. Dostupné z: https://www.jetbrains.com/idea/.

[65] Webstorm. *Jet Brains* [online]. JetBrains s.r.o., © 2000-2019 [cit. 17.7.2019]. Dostupné z: https://www.jetbrains.com/webstorm/.

[66] Awards. *Jet Brains* [online]. JetBrains s.r.o., © 2000-2019 [cit. 17.7.2019]. Dostupné z: https://www.jetbrains.com/company/customers/awards/.

[67] Meet Android Studio. *Android Developers* [online]. Google, [cit. 17.7.2019]. Dostupné z: https://developer.android.com/studio/intro.