Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Vývoj mobilní aplikace pro záznam letů všeobecného letectví

Matěj Jelínek

Bakalářská práce
2022

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Matěj Jelínek**
Osobní číslo:	**I19235**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Vývoj mobilní aplikace pro záznam letů všeobecného letectví**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem práce je implementace aplikace, která bude umožňovat uživateli zaznamenávat všechna potřebná data, týkající se letů na aeroklubových letištích. S těmito daty bude možno provádět všechny základní operace vytvořit, upravit, odstranit a následně tato data bude odesílat na SQL server, kde je bude možné dále zpracovávat a exportovat ve formát CSV, který lze importovat do účetního programu Flight Office.

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

SPÄTH, Peter. Pro Android with Kotlin: Developing Modern Mobile Apps. Berlin: Springer, 2018.
LACKO. Ľuboslav. Mistrovství – Android. Brno: Computer Press, 2017.
Riordan, R.M. Vytváříme relační databázové aplikace. Brno: Computer Press, 2001.

Vedoucí bakalářské práce: **Ing. Jan Panuš, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **17. prosince 2021**
Termín odevzdání bakalářské práce: **13. května 2022**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2022

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 24. 1. 2022

Matěj Jelínek v. r.

ANOTACE

Tento aplikační systém se zaměřuje na zjednodušení procesu evidence letů ve všeobecném letectví. Tento proces začíná vytvořením papírového záznamu. Záznam obsahuje informaci o letadle, pilotovi, času startu, času přistání, počtu přistání a druhu letu. Následně se tyto záznamy zpravidla přepisují do účetního softwaru.

Pomocí tohoto systému je možné vytvářet záznamy o letu elektronicky a prostřednictvím souboru CSV exportovat záznamy pro účetní software k dalšímu zpracování. Vzhledem k nutnosti vytváření záznamů přímo na letištní ploše, kde je potřeba věnovat hodně pozornosti bezpečnosti provozu, tak vytváření záznamů v aplikaci je zaměřeno na minimalizaci textového vstupu uživatele.

KLÍČOVÁ SLOVA

Kotlin, SQL, PHP, mobilní aplikace, chytrý formulář, export dat, letectví, časomíra

TITLE

Mobile application development for recording general aviation flights.

ANNOTATION

This system aims to simplify the process of registration of flights in general aviation. This process begins with the creation of a paper record. The record contains information about the aircraft, pilot, take-off time, landing time, number of landings and type of flight. Subsequently, these records are usually transcribed into the accounting software.

With this system it is possible to create flight records electronically and export records for accounting software via CSV file for further processing. Due to the need of creating records directly at the airport, where it is necessary to pay a lot of attention to traffic safety, creating records in the application is focused on minimizing the user's text input.

KEYWORDS

Kotlin, SQL, PHP, mobile application, smart form, export data, general aviation, timekeeper

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK

PDF	Portable Document Format
ART	Android Runtime
PHP	Hypertext Preprocessor (původně Personal Home Page)
RDBMS	Relational Database Management System
GPL	General Public License
OODBMS	Object-Oriented Database Management Systém
ÚCL	Ústav Civilního Letectví
PIC	Pilot In Command
CSV	Comma Separated Values
P2P	Peer to Peer
UID	Unikátní identifikátor
ICAO	International Civil Aviation Organization
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
SSL	Secure Socket Layer
VPD	Vzletová a přistávací dráha
OGN	Open Glider Network
GPS	Global Positioning System
API	Application Programming Interface
REST	Representational state transfer

ÚVOD

Hlavním cílem bakalářské práce je vytvoření části softwarového systému pro správu záznamů letů ve všeobecném letectví. Jedná se o mobilní aplikaci navrženou v programovacím jazyce Kotlin pro operační systém Android. Tento systém má za cíl zjednodušit proces vedení záznamů letů, který je aktuálně realizován evidováním letů do bloku časoměřiče. Lety jsou vytvářeny osobou, která je v daný den zvolena do role časoměřiče. Časoměřič se může během letového dne volně pohybovat po celé letištní ploše včetně vzletové a přistávací dráhy, kde musí mimo vytváření záznamů letů dbát na bezpečnost provozu. Celý den tak časoměřič nosí u sebe knihu (blok časoměřiče), do které zapisuje prováděné lety. Tyto papírové záznamy je následně nutné převést do elektronické podoby pro účetní program Flight Office. Převod těchto záznamů je zdlouhavý manuální proces, který vyžaduje kvalifikovanou a důvěryhodnou osobu k jeho provedení. Eliminováním papírového bloku časoměřiče z tohoto procesu dojde ke zjednodušení práce s daty a umožní aeroklubům šetřit náklady na provoz tohoto procesu, protože v některých aeroklubech jsou tyto pozice i finančně ohodnocené.

Teoretická část práce se věnuje vysvětlení pojmů týkajících se jazyku Kotlin, SQL a mobilní aplikace s relační databází. V této části je také představen návrh celého softwarového systému včetně částí, které tato práce neobsahuje. Tímto čtenář získá lepší představu o funkci aplikace a její roli v systému. V rámci návrhu jsou popsány důvody volby jednotlivých technologií k řešení dílčích softwarových problémů včetně jejich alternativ.

Praktická část se zaměřuje na konkrétní části vývoje tohoto aplikačního systému. V této části je popsán způsob autentizace a autorizace uživatelů v systému. To zahrnuje způsob registrace a přihlášení aeroklubů i jednotlivých uživatelů. Dále je zde uveden návrh relační databáze k uchovávání dat mobilní aplikace i celého systému. Následně je popsán způsob řešení jednotlivých funkcí serverové i mobilní aplikace. Základem je formulář pro vytvoření záznamu o letu. V této části práce je popsáno, jakým způsobem systém umožňuje tyto záznamy zobrazovat, upravovat a mazat. Obě aplikace následně obsahují další funkce pro jednodušší používání. Okrajově práce zmiňuje grafické provedení nabízených funkcí systému. Nakonec je rozebráno řešení komunikace serveru s mobilní aplikací a nastínění části problémů týkající se synchronizace dat mezi serverem a mobilní aplikací.

Závěr práce popisuje výslednou aplikaci, možnosti úprav existujících částí a vývoj dalších funkcí k dalšímu zvýšení efektivity procesu vytváření záznamů.

1 POUŽITÉ TECHNOLOGIE A JEJICH ALTERNATIVY

Tato kapitola představuje jednotlivé programovací jazyky a technologie použité v celém systému.

1.1 Mobilní aplikace

Mezi dva hlavní operační systémy současnosti pro mobilní zařízení patří Android a iOS. Operační systém Android od společnosti Google je celosvětově nejrozšířenější operační systém pro mobilní zařízení. Operační systém iOS od společnosti Apple je exkluzivně dostupný pouze na jejich vlastních zařízeních. To je jich velkou výhodou, ale zároveň i nevýhodou. Na základě dostupnosti se autor rozhodl pro vývoj aplikace na platformě operačního systému Android. Aplikace pro Android lze implementovat v řadě jazyků, mezi které patří například Java, Kotlin, Python, C++ a C#.

Java byla vydána společností Sun Microsystems roku 1995 a od roku 2008 je oficiálním jazykem pro vývoj mobilních aplikací pro Android. Je jedním z nejpoužívanějších programovacích jazyků na světě. Java je považována za nevhodný jazyk pro začátečníky, protože dokáže být v mnoha ohledech velmi komplikovaná. Java kód nekompiluje pro samotný procesor, ale pro virtuálního hosta. Spouštění kódu v Javě je tedy na tomto hostu závislé, ale na druhou stranu je možné psát kód pro více platforem najednou. Originální virtuální host v Androidu pro kompilaci kódu Javy je Dalvik, ale dnes začíná být oblíbený jeho nástupce ART.[1]

Kotlin je moderním staticky typovaným jazykem založeným na Javě. Snaží se převzít výhody Javy a naopak vyřešit její nedostatky. Kotlin například řeší problém prázdných odkazů, nebo zkracuje předpisy kódu pro kratší a rychlejší vyjádření smyslu kódu. Jinými slovy je méně „upovídaný“ oproti Javě. Kotlin byl vyvinut programátory společnosti JetBrains v Petrohradě roku 2016. Název Kotlin pochází ze jména ostrova nacházejícího se poblíž Petrohradu. V roce 2018 je Kotlin prohlášen oficiálním jazykem pro vývoj aplikací pro platformu Android.[2]

1.1.1 Kotlin vs Java

Rozdílů mezi těmito jazyky je více, ale většina z nich měla minimální, nebo žádný dopad na volbu jazyka pro vývoj aplikace. Kotlin oproti Javě umožňuje rozšiřovat funkcionality tříd bez nutnosti použití dědičnosti. Tato vlastnost zjednodušuje řadu problémů týkajících se vývoje pro platformu Android. Kotlin dále nabízí „líné načítání“, které inicializuje objekty až ve chvíli, kdy jsou zapotřebí. Java na druhou stranu je velmi populární a je široce využívána v mnoha

odvětvích vývoje softwaru. S tím se pojí velikost komunity, která může být vývojáři užitečná při hledání řešení problémů tykajících se vývoje.

Volba jazyka tedy není jednoduchá. Oba jazyky mají své výhody i nevýhody. Java nabízí stabilitu v létech praktického použití pro vývoj a velkou komunitu vývojářů. Kotlin na druhou stranu je více odolný proti chybám v kódu a předchází programovacím chybám z návrhu aplikace. Autor se nakonec rozhodl pro použití jazyka Kotlin, protože se zdá být více perspektivní do budoucna.[3]

1.2 Programovací jazyk serveru

Pro zpracování dat na serveru je zapotřebí zvolit vhodný jazyk. Na trhu existuje řada jazyků a ještě více aplikačních rámců. Tento výběr je zúžen na dva velmi rozšířené skriptovací jazyky PHP a Python.

PHP, nebo také Hypertextový preprocesor byl poprvé vydán už roku 1995. Jedná se o objektově orientovaný skriptovací jazyk se slabou typovou kontrolou. Převážně se používá pro tvorbu dynamického obsahu na webových stránkách a zároveň je v tomto odvětví mezi ostatními jazyky nejrozšířenější. Tento jazyk je implementován v jazyce C a také částečně přejímá jeho syntaktickou strukturu.[4]

Python byl vyvinut již v roce 1989, ale rozsáhlejšího použití se dočkal až v roce 2000 po vydání druhé verze. Tento jazyk je především známý a oblíbený pro svoji jednoduchou syntaxi. Stejně jako PHP je implementován v jazyce C. Podle webu pypl.github.io se jedná o nejpopulárnější jazyk s podílem přes 27 % a to díky obecnosti použití. Standardní knihovny v Pythonu nabízí prostředky pro grafické aplikace, strojové učení, webové aplikační rámce, testování, zpracování obrázků a další.[5]

1.2.1 PHP vs Python

PHP ve srovnání s Python má nestandardní syntaxi, která může být pro vývojáře obtížná na pochopení. V PHP je zároveň obtížnější vytvářet přehledný a udržitelný kód. Na druhou stranu PHP je specialistou ve svém oboru a to je jeho nespornou výhodou, ale v některých případech se může jednat spíše o nevýhodou. Stále má vysokou podporu z komunity vývojářů a tudíž je velmi jednoduché hledat řešení problémů, s kterými se vývojář potká. Aktuálním trendem je postupný ústup PHP a je nahrazován jazykem Python a dalšími.

Do budoucna je jednoznačně více perspektivní Python. Toto rozhodnutí se později bude moci zdát být jako špatné ale autor pro vývoj zvolil PHP z důvodů široké podpory komunity a možnosti si vybrat z širší škály aplikačních rámců. [6]

1.3 Relační databáze

Sbíraná data je zapotřebí ukládat pro budoucí použití. K tomu je zapotřebí vybrat vhodný databázový systém. Vzhledem k povaze a množství sbíraných dat je ideální zvolit strukturovanou relační databázi.

MariaDB je open-source RDBMS pod GPL. Společně s MySQL to jsou jedny z nejrozšířenějších databázových systémů. Vyznačují se širokou podporou jak od vývojářů, tak i od komunity. Nabízí velkou škálovatelnost a to od malých systémů až po velké systémy obsahující desetitisíce tabulek a miliardy záznamů. MariaDB také podporuje transakce.[7]

Stejně jako MariaDB, tak i PostgreSQL je open-source projekt. Kořeny tohoto databázového systému sahají, až do roku 1986 na univerzitu v Berkeley. Tento projekt byl následně vydán roku 1995 pod názvem PostgreSQL95 a o rok později by přejmenován na PostgreSQL. Jedná se o objektově orientovaný databázový systém (OODBMS). Vzhledem k objektovému přístupu tento systém nemá téměř žádné limity, kromě vlastní kapacity diskového uložiště hostitelského systému.[8]

2 NÁVRH SOFTWAROVÉHO SYSTÉMU

Tato kapitola se zaměřuje na analýzu procesu evidence provedených letů ve všeobecném letectví v České republice. Tento proces se může na jednotlivých letištích mírně lišit, ale dle základní struktury jej lze rozdělit do dvou kategorií. První kategorii zastupují komerční letecké školy, které mají již pevně zavedené postupy a pravidla pro tyto účely. Z tohoto důvodu tento aplikační systém není přímo určen pro tyto letecké školy. Cílovou skupinou pro tento aplikační systém jsou letiště z druhé kategorie. Na těchto letištích sídlí aerokluby, které mají zavedená vlastní pravidla a postupy vhodná pro jejich vlastní podmínky.

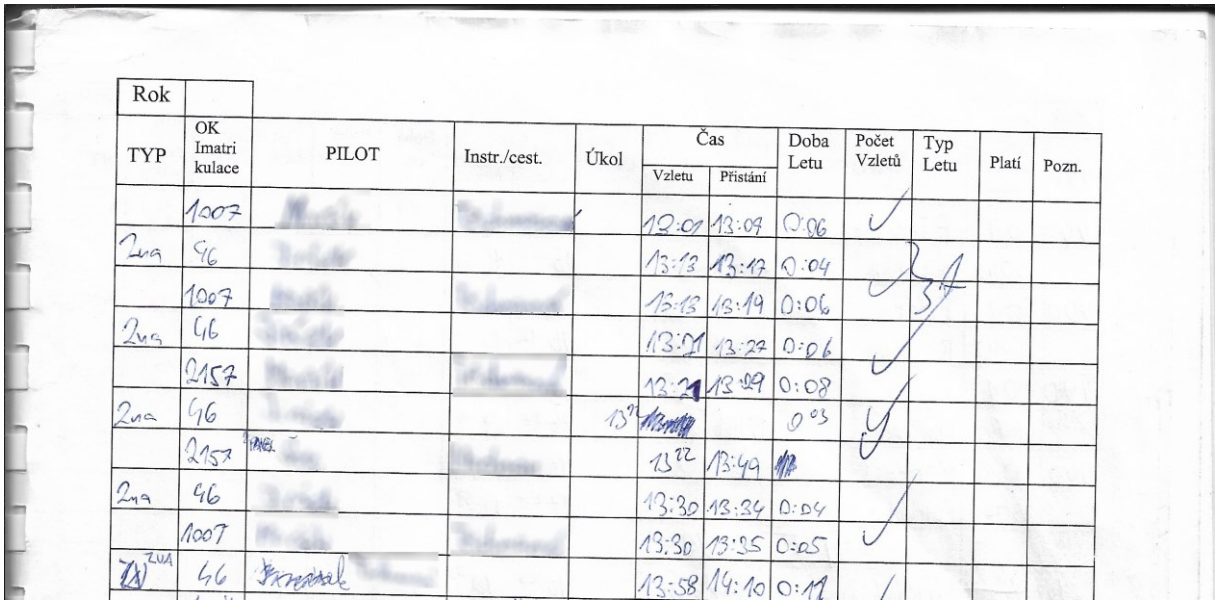
2.1 Analýza aktuálního stavu

Aktuální legislativa, kterou uvádí ÚCL, udává povinnost pilotovi udržovat platný a aktuální vlastní letový zápisník a zároveň zápisník letadla, které pilotuje. Oba tyto zápisníky sdílí velmi podobnou strukturu evidovaných dat, a proto budeme dále hovořit pouze o zápisníku pilota. Tento zápisník musí obsahovat záznam o každém provedeném letu. Záznam o letu musí obsahovat jméno PIC, datum letu, místo a čas odletu a příletu, poznávací značku letadla, celkovou dobu letu, součet letové doby, způsob vzletu a detaily funkce pilota.

Aerokluby vyžadují vedení účetnictví pro evidenci pronajímání letadel, jejich údržby, jednotlivých letů a dalších položek. Pro tyto účely slouží účetní program Flight Office, který mimo jiné zprostředkovává data pro vedení letových zápisníků pilotů i letadel. Tato část procesu evidence o provedení letů je plně funkční a nevyžaduje úpravy. Problém nastává při vytváření jednotlivých záznamů o letu.

Záznamy do programu Flight Office může provádět pouze pověřená a důvěryhodná osoba. Z tohoto důvodu aerokluby zavedly blok časoměřiče (papírový dokument pro evidenci provedených letů). Pro každý letový den, kdy bude proveden libovolný nenulový počet letů, je vypsána funkce časoměřiče, který po celý den ručně eviduje všechny provedené lety do papírového bloku časoměřiče. Tato funkce může během dne být přenesena z jedné osoby na druhou a v praxi ji provádí osoba, která má zrovna čas a přístup k bloku časoměřiče. Evidované lety v tomto dokumentu je pak nutné ručně přepsat do účetního programu Flight Office. Tento proces přepisování záznamů je časově náročný a měsíčně spotřebuje řádově jednotky až desítky člověkohodin. Pověřená a důvěryhodná osoba, která tento proces provádí, často není ani finančně ohodnocena. Cílem je tedy minimalizovat, nebo lépe kompletně eliminovat tento proces přepisování záznamů o letu.[9]

Na následujícím obrázku je možné vidět záznam několika letů v bloku časoměřiče. Na první pohled je možné vidět, že v tomto dokumentu často dochází ke škrtání a přepisování chybně zapsaných údajů.



Rok											
TYP	OK Imatri kulace	PILOT	Instr./cest.	Úkol	Čas		Doba Letu	Počet Vzletů	Typ Letu	Platí	Pozn.
					Vzletu	Přistání					
	1007				12:01	13:09	0:86	✓			
2ua	46				13:13	13:17	0:04				
	1007				13:13	13:19	0:06				
2ua	46				13:21	13:27	0:06				
	0157				13:21	13:29	0:08				
2ua	46				13:34		0:03				
	2157				13:22	13:49					
2ua	46				13:30	13:34	0:04				
	1007				13:30	13:35	0:05				
ZUA	46				13:58	14:10	0:11				

Obrázek 1 *– Část stránky bloku časoměřiče (zdroj vlastní)*

2.2 Specifikace systému

Cílem tohoto aplikačního systému je minimalizace náročnosti procesu přepisování záznamů o letu. Nejdříve je zapotřebí charakterizovat jaké bude mít systém vstupy a jaké výstupy.

Vstupy systému jsou velmi komplikované. Nejzákladnějším vstupem je samotné provedení letu. Následně uživatel systému, který je pověřen funkcí časoměřiče musí být schopen evidovat data o letu. Mezi tato data patří datum letu, jméno pilota, jméno instruktora, poznávací značku letadla, funkce pilota v letadle, účel letu, čas vzletu, čas přistání, dobu letu, počet přistání, způsob vzletu, letiště vzletu, letiště přistání a dodatečně umožňuje přidat poznámku časoměřiče. Tato data musí být schopen evidovat i při nedostupnosti internetového připojení. Dále je kladen důraz na pokud možno nejjednodušší způsob zadávání, protože časoměřič zároveň musí věnovat pozornost udržování bezpečnosti provozu na letištní ploše. Uživatelů provádějících evidenci letů může být více a zároveň potřebují mít současný přístup k této evidenci.

Všechny záznamy je zapotřebí shromažďovat do jednoho uzlu pro jejich centralizovanou správu. Je požadováno, aby jednotlivé lety bylo možné dále upravovat a případně i smazat. Po shromáždění dat je nutné mít možnost data v kterýkoliv okamžik exportovat ze systému ve formátu CSV.

2.3 Funkční požadavky

1. Autentizace uživatelů, kdykoliv se připojí k systému
2. Registrace do společnosti musí být ověřena již ověřeným uživatelem u této společnosti.
3. Systém musí podporovat role uživatelů.
4. Systém umožňuje zpracovávat lety více společností, které jsou navzájem oddělené.
5. Systém je možné používat i bez připojení k internetu.
6. Formuláře systému minimalizují textový vstup uživatele.
7. Každý ověřený uživatel smí vytvářet, modifikovat a mazat záznamy letů.
8. Každý ověřený uživatel smí vytvářet, modifikovat a mazat, všechny číselníky společnosti.
9. Systém musí být optimalizován pro použití na mobilních zařízeních.

2.4 Nefunkční požadavky

1. Systém musí být schopni používat čeští uživatelé od patnácti let.
2. Systém umožňuje připojení minimálně dvou set uživatelů v jednom okamžiku.
3. Systém by měl odpovídat na všechny dotazy do deseti sekund.

2.5 Architektura klient-server

Systém má dva klíčové požadavky. V první řadě je zapotřebí data shromažďovat do jednoho uzlu a za druhé je zapotřebí umožnit řadě uživatelů vytvářet data a to i za nedostupnosti připojení k internetové síti. Jedním z možných řešení je použití P2P sítě. Jednotlivá zařízení by obsahovala stejnou aplikaci, která by umožňovala vytvářet záznamy letů a zároveň by umožňovala tyto záznamy exportovat. Problém nastává při synchronizaci a udržování konzistence dat napříč jednotlivými zařízeními. Už tak složitou synchronizaci nijak nezjednodušuje ani požadavek na možnost evidovat data bez připojení k internetové síti. Použití P2P sítě pro tento systém by přineslo více nevýhod nežli výhod.[10]

Další z možností je architektura Klient-Server. Jedná se o komunikační schéma, kde uprostřed všeho je server. Ten poskytuje svoje zdroje a služby všem klientům pro jejich vlastní potřebu. Pro náš systém je velmi výhodná jednosměrná závislost mezi klientem a serverem.

Server nevyžaduje přítomnost jednotlivých klientů, takže odpojení klienta od sítě nepředstavuje žádné důležité problémy, které by mohly systém ohrozit. Při použití tohoto schématu je poměrně jednoduché udržovat data konzistentní a synchronizovaná. Server bude vždy poskytovat aktuální data všem klientům, kteří s nimi budou moci nakládat dle svých potřeb. Klienti tak budou schopni na základě aktuálních dat ze serveru vytvářet nová data, které následně mohou zpět na server ukládat.[11]

Komunikaci mezi klientem a serverem lze dále charakterizovat tak, že server nepřetržitě čeká na požadavky klientů, a ve chvíli jejich obdržení provede jejich zpracování a následně klientovi zašle odpověď na daný požadavek. Pod odpovědí serveru si lze představit jednoduché zprávy specifikující výsledek zpracování požadavku, nebo se může jednat klidně o složitá data, která si klient v požadavku vyžádá.

3 NÁVRH DATABÁZE

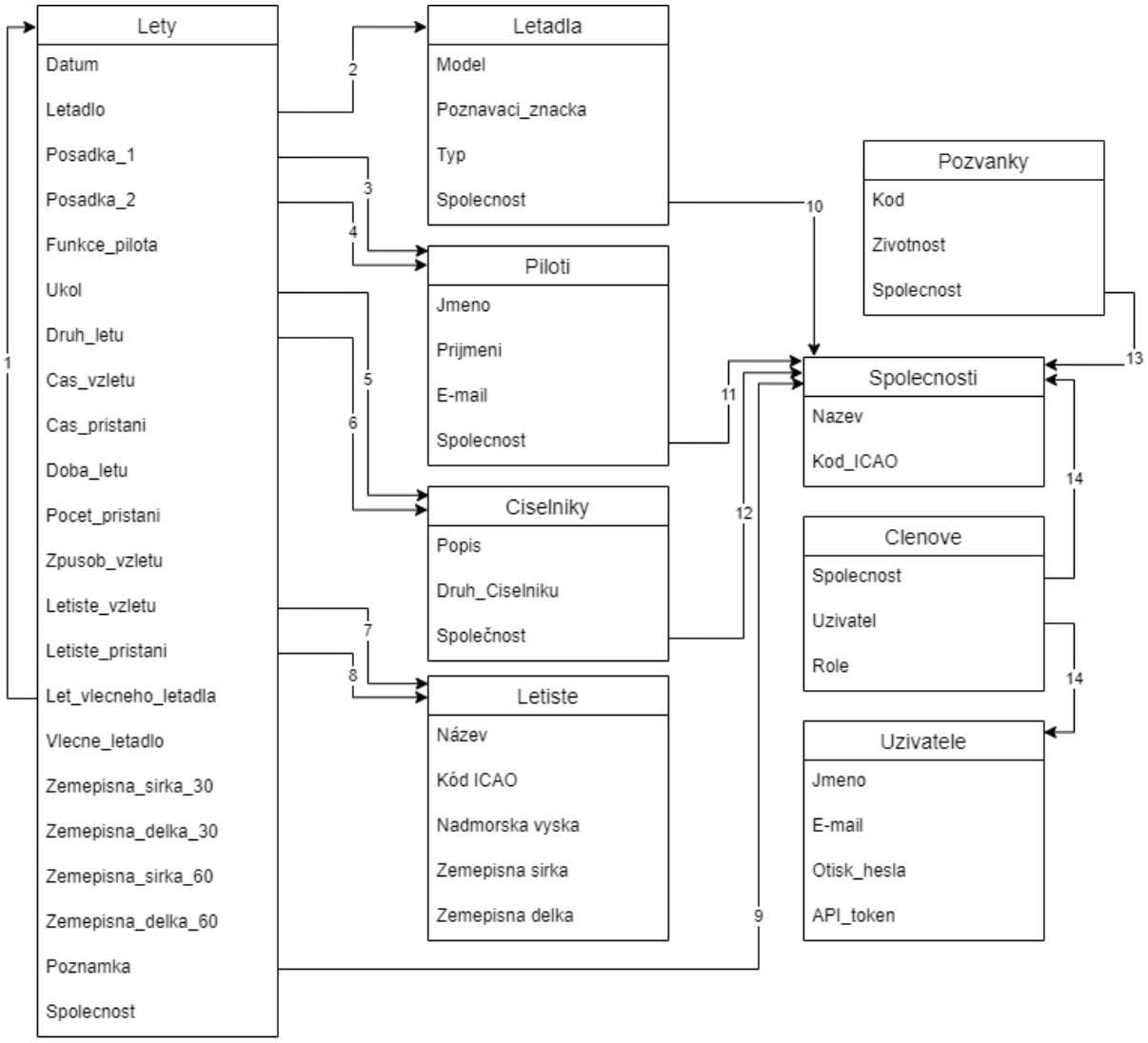
Sbíraná data je zapotřebí nějakým způsobem ukládat pro možnost jejich budoucího exportování. Data mají pevně danou strukturu a pro tento druh dat je vhodné použít databázi umožňující ukládat strukturovaná dat. Námi zvolenou databází pro shromažďování dat na serveru je MySQL. Nesmíme zapomenout na požadavek umožňovat evidenci letů i za nedostupného internetového připojení. Z toho vyplývá, že klient bude muset umět data ukládat, než je bude moci odeslat na server. Klient na rozdíl od serveru bude používat jednodušší databázový systém SQLite.

3.1 Server

První důležitou tabulkou je tabulka společností, nebo můžeme říct tabulka aeroklubů. Tato tabulka obsahuje sloupce s názvem společnosti a ICAO kódem letiště, kde společnost sídlí. Ačkoli databáze obsahuje číselník letišť s jejich ICAO kódy, tak není vhodné je navzájem propojovat, protože číselník letišť je určen pro identifikaci místa vzletu a přistání. Zatímco kód ICAO u společnosti slouží jako ekvivalent názvu společnosti. Tato tabulka slouží k oddělení dat v systému na základě společnosti, které patří. Dalšími dvěma tabulkami jsou členové a uživatelé. K jednotlivým uživatelům systém vede záznam o jejich jménu, mailové adrese, otisku jejich hesla a bezpečnostní token pro komunikaci s aplikací uživatele. Tento token je volitelný a generuje se až ve chvíli, kdy je zapotřebí. Tabulka členů je vazební tabulkou mezi uživatelem a společností, která navíc obsahuje záznam o roli, kterou má uživatel v dané společnosti přiřazenou. Z hlediska požadavků systému tato tabulka není nutná, ale jedná se o návrh, který umožňuje propojení uživatele s více společnostmi. Tato vlastnost může být velmi přínosná, neboť pilot může mít současně zavedený účet na více letištích, neboli ve více společnostech. Další tabulkou jsou pozvánky. Tato tabulka obsahuje kód pozvánky a její životnost. Pozvánky se využívají k registraci uživatelů, neboť se jedná o uzavřený systém.

Všechny následující tabulky se již přímo týkají konkrétního záznamu letu. Tabulka letadel obsahuje jejich model, poznávací značku a typ letadla. Typ letadla se ukládá jako jednoduché číslo a následně v aplikační logice se mu přiřazuje jedna z hodnot: kluzák, motorový kluzák, motorové letadlo, proudové letadlo. Tabulka pilotů vede záznamy o jméně a příjmení pilota a jeho mailové adrese. Tato adresa je zapotřebí k identifikaci pilota v programu Flight Office pro, kterým se data exportují. Další tabulka slouží jako číselník letišť. Ta obsahuje název letiště, ICAO kód, nadmořskou výšku, zeměpisnou šířku a výšku daného letiště.

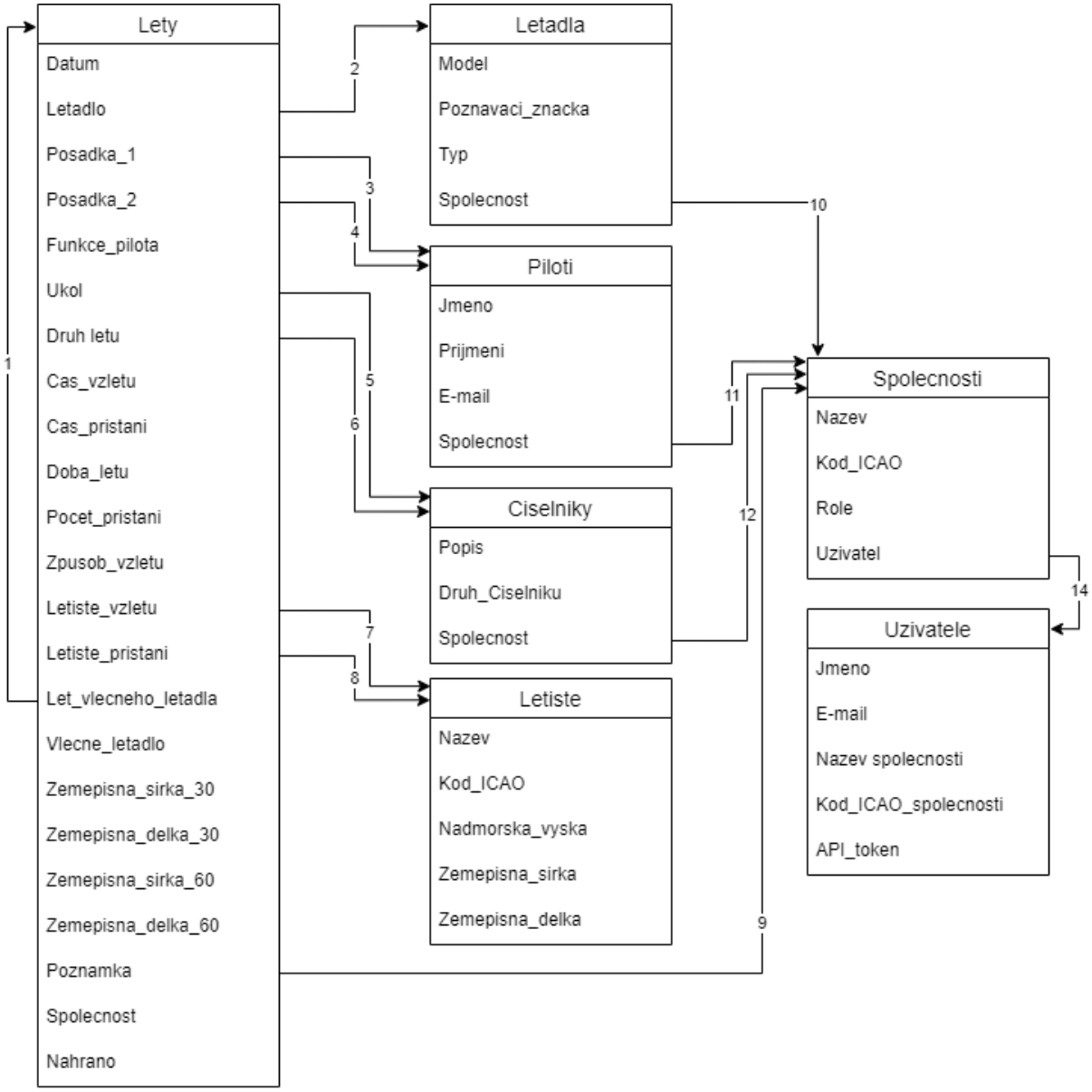
Poslední a také největší tabulkou je tabulka letů. V první řadě musíme znát, kdy letadlo odstartovalo a kdy přistálo. K tomu by stačila dvě časová razítka, která by zároveň obsahovala i datum provedení. V takovém případě by nebyl problém evidovat lety delší než 24 hodin. Bohužel program Flight Office ukládá datum zvlášť od časů vzletů a přistání. Tyto časy následně ukládá jako číslo reprezentující počet minut od půlnoci. Autor se rozhodl použít variantu přímo kompatibilní s programem Flight Office, ačkoli to databázi znemožní evidovat lety zasahující do dvou kalendářních dnů. Dále let vede informaci o letadle a až dvoučlenné posádce tohoto letadla. Sloupcem „funkce pilota“ se rozumí číselník implementovaný až v aplikační logice, který je v databázi ukládán jako jednoduché číslo. Funkce pilota specifikuje vztah pilota k danému letu. Může se tedy jednat o let sólo, s instruktorem, více pilotní, sólo dvojí, s examinátorem, nebo s examinátorem dvojí. „Úkol“ a „druh letu“ jsou sloupce, které společnostem umožňují kategorizovat jednotlivé lety. Vzhledem k jejich jednotné struktuře, a to pouze jejich textovému popisu, jsou tato data spojena do jedné tabulky nazvané číselníky. Dobou letu se obecně rozumí rozdíl času vzletu a přistání. Ovšem v některých případech tomu tak být nemusí a je potřeba tuto hodnotu též ukládat. Sloupec způsob vzletu je stejně jako sloupec funkce pilota implementován až v aplikační logice a v databází je též reprezentován jednoduchým číslem. Tímto sloupcem je možné letu specifikovat způsob vzletu, který může nabývat následujících hodnot naviják, aerovlek, samostatně, motorový kluzák, za autem a na gumě. Tyto hodnoty jsou převzaty z programu Flight Office pro vzájemnou kompatibilitu. Následuje sloupce počtu přistání, který aktuálně není exportován. V budoucnu tento sloupec může sloužit k zapisování více letů stejné povahy do jednoho záznamu, kde tento sloupec specifikuje počet přistání. V neposlední řadě let ukládá místo vzletu a přistání. Sloupce „let vlečného letadla“ a „vlečné letadlo“ slouží k ukládání vztahu dvou letů na základě způsobu vzletu aerovlek, protože tohoto způsobu vzletu se účastní dvě letadla. Každý let ještě může obsahovat textovou poznámku od uživatele, který daný let vytváří. Poslední čtyři dosud nezmíněné sloupce týkající se zeměpisných souřadnic se aktuálně nevyužívají. Jedná se o zeměpisné souřadnice popisující polohu letadla třicet a následně šedesát sekund po vzletu. Pomocí těchto informací bude možné automaticky vzájemně párovat lety se způsobem vzletu aerovlek.



Obrázek 2 *– Diagram tabulek v databázi serveru (zdroj vlastní)*

3.2 Klient

Schéma databáze klientské aplikace je velmi podobné schématu serverové databáze. Z pohledu tabulek zajišťujících ukládání dat o letu jsou tyto databáze stejné. Jediným rozdílem je nový sloupec v tabulce letů na straně klienta. Sloupec „nahráno“ není ničím jiným než přidáním příznaku letu, díky kterému lze ze strany klienta určit, zdali se konkrétní záznam již nachází v centrální serverové databázi. Tato informace umožňuje provádět synchronizaci mezi klientem a serverem. Z ostatních tabulek zde kompletně mizí tabulka pozvánek a vazební tabulka uživatelů a společností. Vazba M:N mezi uživatelem a společností již u klienta nemá žádný význam. Klientskou aplikaci používá zpravidla jeden konkrétní uživatel, který může být zapsán ve více společnostech. Pokud se ke klientské aplikaci připojí jiný uživatel, tak dochází k přepsání těchto údajů.



Obrázek 3 *– Diagram tabulek v databázi klienta (zdroj vlastní)*

3.3 Vazby tabulek

1. Tabulka letů má hierarchickou vazbu 1:1, k provázání letu s letem vlečného letadla.
2. Tabulka letů má vazbu s tabulkou letadel 1:N, kde letadlo letí v daném letu.
3. Tabulka letů má vazbu s tabulkou pilotů 1:N, kde piloti pilotují daný let.
4. Tabulka letů má vazbu s tabulkou pilotů 1:N, kde piloti jsou instruktory nebo pasažéry letu.
5. Tabulka letů má vazbu s tabulkou číselníků 1:N, k určení úkolu letu.
6. Tabulka letů má vazbu s tabulkou číselníků 1:N, k určení druhu letu.

7. Tabulka letů má vazbu s tabulkou letišť 1:N, k určení letiště vzletu.
8. Tabulka letů má vazbu s tabulkou letišť 1:N, k určení letiště přistání.
9. Tabulka letů má vazbu s tabulkou společností 1:N, k určení vlastnictví letu společností.
10. Tabulka letadel má vazbu s tabulkou společností 1:N, k určení vlastnictví letadel společností.
11. Tabulka pilotů má vazbu s tabulkou společností 1:N, specifikující piloty, které u společnosti létají.
12. Tabulka číselníků má vazbu s tabulkou společností 1:N, k určení vlastnictví číselníků společností.
13. Tabulka pozvánek má vazbu s tabulkou společností 1:N, k specifikaci pozvánek patřící společnosti.
14. Tabulka členů je vazební tabulkou mezi tabulkami společností a uživatelů M:N, kde uživatel pracuje pro N společností a společnost má M uživatelů.

4 REALIZACE APLIKACÍ

Následující kapitola se zabývá praktickým provedením aplikace pro zpracování dat na serveru a klientskou mobilní aplikací pro sbírání dat. Aplikace serveru je realizována v jazyce PHP a konkrétně se jedná o framework Laravel. Podrobněji se zabývá autentizací a autorizací uživatelů v systému. Dále popisuje způsob provedení formulářů, tabulek a komunikační rozhraní s databází.

4.1 Laravel

Jedná se o robustní aplikační Framework pro webové aplikace. Nabízí vysokou míru abstrakce, tak aby se vývojář mohl soustředit na důležité aspekty aplikace, zatímco framework obstará detaily implementace. Mezi důležité vlastnosti patří abstraktní vrstva nad databází, integrované testování, nebo vkládání závislostí. Zároveň nabízí i bezpečnostní vrstvu. Framework například nabízí zabezpečení formulářů pomocí CSRF tokenů.[12]

4.1.1 Architektura

Všechny požadavky na aplikaci jsou směrovány do souboru *index.php,* který je vstupním bodem do Laravelu. Další důležitý soubor je *web.php*, který je rozcestníkem pro všechny požadavky. Každý požadavek, který žádá neznámou cestu v rozcestníku, je zahozen. Zde se také rozhoduje, jestli před obsloužením samotného požadavku má dojít k předání požadavku k nějakému middleware pro jeho podrobnější kontrolu.

Service Provider je druh třídy, který slouží k zapouzdření jednotlivých komponent, které framework nabízí. Tito poskytovatelé služeb musí být nejdříve registrovány v konfiguračním souboru. Laravel poté při spuštění inicializuje všechny registrované poskytovatele a spustí je. Používají se například k zapouzdření směrovacích komponent, nebo také databáze.[12]

4.1.2 Middleware

Laravel nabízí nástroj middleware, který je v serverové aplikaci široce používán. Tento nástroj umožňuje sledovat a filtrovat požadavky vstupující do aplikace. Stává se tak ideálním nástrojem pro autorizaci různorodých požadavků. Lze jej však použít i pro jiné účely, jako například přesměrování požadavku při dočasném vyřazení některých zdrojů aplikace.[12]

4.2 Registrace a přihlášení

Vzhledem k citlivosti dat, která mohou jednotlivé letiště do systému vkládat, je nutné regulovat, který uživatel má přístup ke kterým datům. Pro tyto účely je nutné do systému zavést správu uživatelů a s tím spojenou registraci a přihlášení uživatelů. Registraci do systému lze provést pouze přímo v serverové aplikaci. Systém nabízí dva druhy registrace, registraci letiště a registraci uživatele.

Registrace letiště slouží k registraci jednoho konkrétního letiště společně s prvním uživatelem, který bude pod tímto letištěm registrován a kterému automaticky může být přiřazena konkrétní role (například administrátora). Registrační formulář letiště tak mimo identifikační údaje uživatele vyžaduje název zakládaného letiště a značku ICAO daného letiště. V praxi se může vyskytovat více společností na jednom letišti, proto je vyžadován název společnosti (letiště a společnost lze v tomto kontextu považovat za synonyma), který je využíván pro jednoznačnou identifikaci společnosti v systému. Ukládání značky ICAO pak slouží hlavně pro samotné uživatele, neboť se společnosti navzájem mezi sebou referují pomocí těchto značek, tak to umožňuje uživateli jednoduše určit, kterou společnost aktuálně spravuje. Pro zachování konzistence dat v systému jsou položky název letiště a e-mail uživatele kontrolovány jestli se jedná o unikátní hodnoty. V opačném případě je uživateli zabráněno v registraci. Všechny položky registračního formuláře letiště jsou patřičně chráněny proti útokům, mezi které patří například SQLInjection. Samotné heslo uživatele je serverovou aplikací šifrováno algoritmem SHA-256 a výsledný otisk hesla je ukládán do datové vrstvy.

Registrace uživatele slouží k registraci dalších uživatelů pod již existující společnost. Tento způsob registrace mimo identifikační údaje uživatele vyžaduje zadání pozvánkového kódu. Tento kód slouží ke dvěma účelům. V první řadě slouží k jednoznačnému určení, ke které společnosti bude uživatel registrován a následně slouží jako bezpečnostní prvek, který zamezuje registracím nežádoucích uživatelů k již existujícím společnostem. Stejně jako registrace letiště, tak i tento formulář náležitě chrání vstupní pole včetně vytváření otisku hesla.

Pozvánkový kód letiště může být vytvořen pouze již registrovaným uživatelem daného letiště. Pro tyto účely slouží v systému funkce pro generování pozvánky. Uživatel může zvolit časové období, ve kterém bude pozvánka platná, a samotný pozvánkový kód je generován systémem jako pseudonáhodný řetězec desíti znaků. Pozvánkový kód a životnost tohoto kódu jsou následně ukládány do datové vrstvy. Tímto způsobem by se postupem času datová vrstva

zahltila zbytečnými již neplatnými pozvánkami. Z tohoto důvodu na datové vrstvě je naplánovaná pravidelná událost, která jednou měsíčně odstraní všechny exspirované pozvánky.

Přihlášení uživatele probíhá standardním způsobem. Je vyžadováno zadání e-mailu pro jednoznačnou identifikaci uživatele a zadání hesla, které je po přijetí převeden na otisk hesla a následuje provedení autentizace uživatele, kde se kontroluje správnost e-mailu a správnost otisku hesla, který je k tomuto uživateli na datové vrstvě uložen. Po úspěšném přihlášení jsou identifikační údaje uživatele údaje uloženy do dočasného souboru aktuální relace na straně serveru. Následně je uživatel přesměrován na interní stránku letiště.

4.3 Správa přístupu do systému

Každý dotaz směřující na server je nejdříve zpracován pomocí middleware, který bude dále referován už jen jako prostředník. Systém tyto prostředníky využívá k autorizaci uživatelů pro přístup k jednotlivým zdrojům serveru a zároveň v případě neúspěšné autorizace zajišťují přesměrování uživatele na bezpečnou stránku s chybovou hláškou. Každý prostředník nejdříve kontroluje, zdali je uživatel přihlášen. V případě neúspěchu je uživatel ihned přesměrován k přihlášení a v případě úspěchu se provádí kontrola role, kde prostředník má vnitřně určenou minimální roli, kterou uživatel musí mít pro přístup na zdroje serveru, kterým bude tento prostředník přidělen.

```
if(session()->has('LoggedUser') && session()->has('ActiveCompany')){
    $userCompany = UserCompany
        ::where('user', '=', session('LoggedUser'))
        ->where('company', '=', session('ActiveCompany'))
        ->first();
    if($userCompany['role'] >= 1){
        return $next($request);
    }else {
        return redirect(route('notVeritified'));
    }
}else{
    return redirect(route('login'))->with('fail', 'Musíš být přihlášen.');
}
```

Zdrojový kód 1 *– Middleware pro kontrolu uživatele (zdroj vlastní)*

Systém rozlišuje čtyři základní role: „nový uživatel“, „uživatel“, „administrátor“ a „vlastník“. Při registraci uživatele je mu přiřazena role „nový uživatel“, který nemá přístup k žádným zdrojům serveru. Každý uživatel s rolí „uživatel“ a vyšší pak může přistoupit ke správě uživatelů v systému, kde může modifikovat roli uživatele, který má hierarchicky nižší

roli než on samotný. Tím je zajištěno ověření nově registrovaného uživatele uživatelem jiným, který je již ověřen. Zároveň tak zamezuje uživateli měnit roli uživatelům, kteří ji mají stejnou nebo vyšší než on samotný. V případě výskytu nežádoucích uživatelů systém, vedle modifikace rolí, nabízí možnost nežádoucího uživatele odstranit.

4.4 Správa záznamů letů na serveru

Hlavní funkcí celého systému je správa záznamů letů. V následujících kapitolách je podrobně popsána struktura a vlastnosti správy záznamů letů v serverové aplikaci.

4.4.1 Číselníky

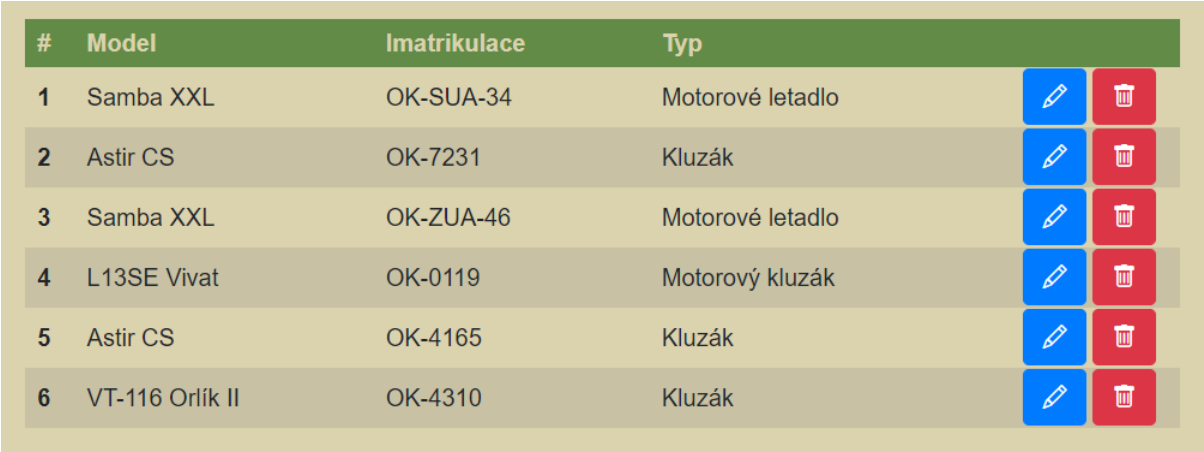
Pro zjednodušení manipulace se záznamy letů systém nabízí řadu číselníků. Záznam letu obsahuje velké množství důležitých dat a právě číselníky minimalizují textový vstup uživatele při manipulaci s těmto záznamy. Tímto způsobem systém minimalizuje počet míst, kterými by mohl útočník napadnout systém a zároveň minimalizuje možnost zadání chybových dat neopatrným uživatelem.

Systém nabízí statické a dynamické číselníky. Statické číselníky mají systémem pevně přiřazenou hodnotu ve výčtovém datovém typu a samotný let pak nese pouze číselný index této hodnoty. Mezi statické číselníky patří „typ letadla“, „funkce pilota“ a „způsob vzletu“. Tato sledovaná data jsou svojí povahou neměnná a systém je nemusí nijak modifikovat.

Dynamické číselníky se naopak nemají pevně vázané hodnoty, ale mohou se v čase měnit. Tyto číselníky se zároveň mohou lišit mezi jednotlivými společnostmi registrovanými v systému. Mezi dynamické číselníky patří letadla, piloti, úkol letu a druh letu. Každý z těchto číselníků lze prostřednictvím serverové aplikace modifikovat podle potřeb společnosti.

Číselník letadel má za úkol udržovat informace týkající se kompletní flotily letadel, kterými společnost disponuje. Letadla létající u dané společnosti mohou být vlastněna přímo společností, nebo můžou být vlastněna soukromou osobou, která u této společnosti létá. Z pohledu systému není nutné tuto vlastnost o letadle znát, neboť se jedná o záležitost a zodpovědnost navazujícího systému Flight Office. O každém letadle systém vede data týkající se jeho modelového typu, státní poznávací značky a typu letadla. Modelový typ letadla slouží pro jednoduchou identifikaci letadla z pohledu uživatele. Na druhou stranu státní poznávací značka slouží k jednoznačné identifikaci letadla a nakonec typ letadla, jak už bylo zmíněno, je statický číselník, který umožňuje systému rozlišovat jednotlivá letadla mezi sebou a tím

provádět kontroly letů uskutečněných s tímto letadlem. Například letadlo typu kluzák nemůže odstartovat samostatně.



#	Model	Imatrikulace	Typ
1	Samba XXL	OK-SUA-34	Motorové letadlo
2	Astir CS	OK-7231	Kluzák
3	Samba XXL	OK-ZUA-46	Motorové letadlo
4	L13SE Vivat	OK-0119	Motorový kluzák
5	Astir CS	OK-4165	Kluzák
6	VT-116 Orlík II	OK-4310	Kluzák

Obrázek 4 *– Dynamický číselník letadel (zdroj vlastní)*

Dynamický číselník pilotů slouží k udržování informací týkajících se všech pilotů, kteří létají u dané společnosti. Systém o každém pilotovi vede záznam o jeho jménu a příjmení, které slouží pro jeho uživatelsky příjemnou identifikaci a následně vede záznam o e-mailu pilota, který se používá pro jednoznačnou identifikaci pilota z pohledu systému. E-mail pilota se zároveň používá pro jeho identifikaci v externím systému Flight Office. Toto systém nemá možnost kontrolovat, takže celková zodpovědnost za správnost tohoto údaje zodpovídá uživatel vytvářející záznam o pilotu.

Posledními dvěma dynamickými číselníky jsou úkoly a druhy letu. Tyto slouží ke kategorizaci letu podle vnitřních potřeb letiště. Tyto číselníky obsahují pouze jednoduchý textový popis letu, který se přenáší do externí aplikace Flight Office pro umožnění vytváření statistik a dalších věcí souvisejících s účetnictvím.

Systém obsahuje ještě jeden číselník a tím jsou letiště pro záznam letiště vzletu a přistání. K těmto datům systém přistupuje jako k dynamickým, ale zároveň neumožňuje uživatelům tento číselník modifikovat. Vzhledem k povaze dat tento číselník není vázaný na společnost, ale je pro všechny společnosti stejný jako statické číselníky. V případě změny stávajících letišť je úkolem administrátora systému tento číselník aktualizovat.

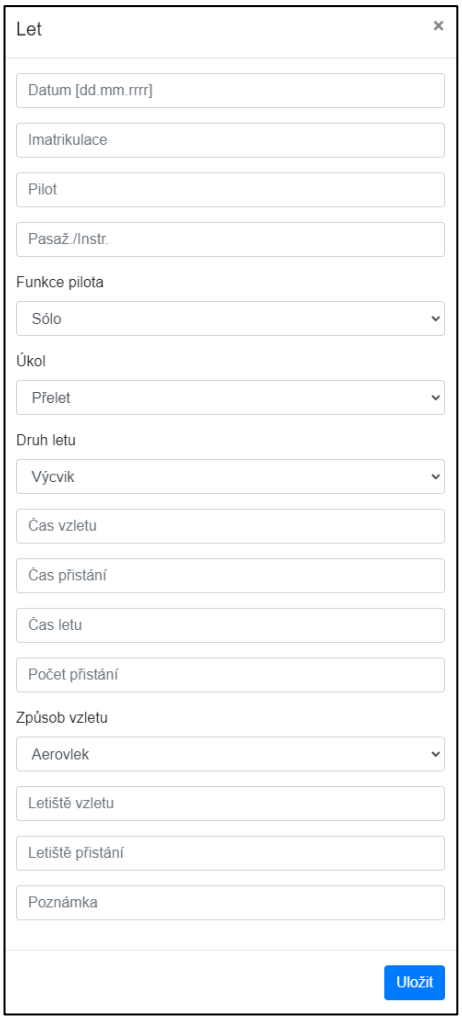
4.4.2 Správa letů

V serverové aplikaci k vytváření a úpravě záznamů letů slouží jeden formulář. Formulář provádí nezbytné ochranné kroky k zabezpečení všech vstupních dat, kde všechny číselníkové

položky kontroluje, zdali opravdu v systému existují a jestli nebyly podvrženy. Formulář dále obsahuje položky pro zadání data provedení letu, času vzletu, času přistání, době letu a poznámku. Tyto položky jsou kontrolovány regulárními výrazy na povolené hodnoty.

V případě nevyplnění položek data letu a času vzletu není vyvolávána chyba, ale je automaticky doplněno o aktuální datum a čas, ve kterém byl let vytvořen. Položka času přistání není povinná, neboť systém umožňuje zadávat i právě probíhající lety. Doba letu je v případě dostupnosti času vzletu i přistání dopočítána jejich rozdílem, ale to pouze v případě pokud uživatelem není tato položka přímo specifikována. Položky poznávací značka letadla, první člen posádky, funkce pilota, úkol letu, druh letu, způsob vzletu a letiště vzletu jsou povinné položky a při jejich nevyplnění je uživatel patřičně upozorněn na jejich absenci. Nakonec jsou zde dvě nepovinné položky, mezi které patří druhý člen posádky a letiště přistání. Ovšem dodatečně je povinné tyto položky doplnit.

Mazání jednotlivých letů je velmi jednoduchá operace, kterou může provádět, kterýkoliv uživatel s rolí stejnou nebo vyšší než je role „uživatel“. Při pokusu o smazání záznamu je uživatel nejdříve vyzván o potvrzení, protože se jedná o nebezpečnou operaci. Teprve po úspěšném potvrzení je záznam opravdu smazán.



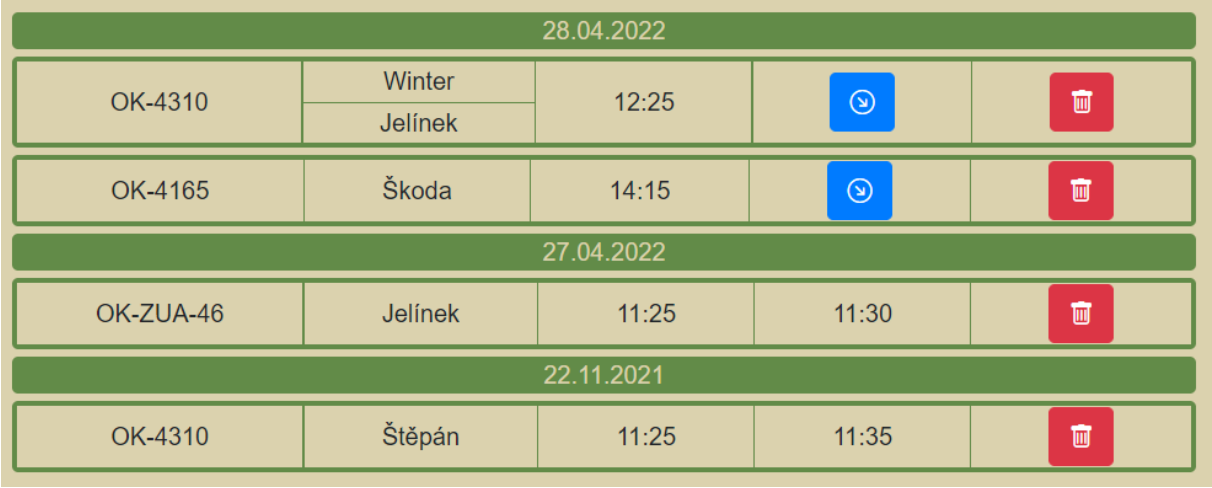
Obrázek 5 *– Formulář letu (zdroj vlastní)*

4.4.3 Exportování letů

Tato funkce je nabízena pouze v serverové aplikaci, neboť v mobilní aplikaci jej není zapotřebí. Systém aktuálně nabízí pouze základní možnosti exportování letů do souboru CSV. Jednotlivé lety jsou nejdříve kontrolovány, jestli obsahují všechny povinné položky, protože se v systému se mohou nacházet i záznamy, které nejsou dokončené, a které je nevhodné exportovat. Dokončené lety jsou následně formátovány do textových řádků, kde jsou jednotlivé položky letu od sebe odděleny znakem čárky. Po vytvoření CSV souboru jsou sestaveny příslušné hlavičky pro přenos souboru prostřednictvím protokolu http a následně je soubor odeslán uživateli. Soubor je sestavován pouze dočasně a tak po odeslání není zatěžován souborový systém serveru.

4.4.4 Grafické zobrazení letů

Grafické zobrazení seznamu záznamů letů je opravdovou výzvou. Každý záznam obsahuje patnáct pro uživatele důležitých položek. Ačkoli systém obsahuje dedikovanou mobilní aplikaci, tak i serverová aplikace musí být optimalizována pro zobrazení na mobilních zařízeních. Toho je docíleno vybráním opravdu nejnutnějších položek pro reprezentování záznamu letu v seznamu. Vybranými položkami jsou datum letu, první člen posádky, druhý člen posádky, čas vzletu a čas přistání. Dále se předpokládá, že řada letů bude provedena ve stejný den. Z tohoto důvodu je seznam letů seřazen podle jejich data a času vzletu s tím, že na začátku seznamu bude vždy nejmladší záznam. Toto řazení následně umožňuje odebrat zobrazení data letu od každého záznamu a jednotlivé záznamy jsou rozděleny do skupin podle tohoto data letu. Následně stačí zobrazit tento datum pouze jednou pro celou skupinu letů a ušetří se tak cenné místo na obrazovce uživatele. Dále se seznam musí vypořádat se zobrazením volitelné položky druhého člena posádky. Z tohoto důvodu seznam není zobrazen pomocí běžné tabulky, ale používají se dva návrhy pomocí CSS stylu Grid. Tento přístup k zobrazení seznamu dále v budoucnu jednoduše umožní zobrazovat i dva párované záznamy letů. Následují položky časů vzletu a času přistání, kde položka času přistání je nepovinná a v případě její absence u záznamu je tato položka nahrazena tlačítkem pro rychlé zaznamenání času přistání bez nutnosti celý let upravovat.



Obrázek 6 *– Zobrazení seznau záznamů letů (zdroj vlastní)*

4.5 Mobilní aplikace

Mobilní aplikace na první pohled nabízí stejné možnosti jako serverová aplikace. Hlavním důvodem této aplikace je umožnit uživateli vytvářet lety i za nedostupnosti internetového

připojení. Tento požadavek přináší řadu výzev, jejichž řešení je velmi komplikované. V rámci této práce je vyřešen základní koncept pro komunikaci aplikace se serverem, ale nelze očekávat ideální řešení pokrývající všechny specifické situace, ve kterých se systém může vyskytnout.

Architekturu mobilní aplikace lze rozdělit do pěti základních komponent. Vzhledem k požadavkům mobilní aplikace nemůže být závislá na datech obdržených ze serveru. K tomuto účelu si aplikace udržuje vlastní kopii serverové databáze. První komponentou je rozhraní, které zprostředkovává komunikaci s lokální SQLite databází. Pro komunikaci se serverem slouží druhá komponenta, která zprostředkovává komunikaci se serverovým restAPI modulem. Třetí kategorií komponent jsou „aktivity“, které zajišťují logické ovládání grafických prvků aplikace a předává je podle potřeby lokální databázi, nebo serveru přes rozhraní. Další kategorií komponent jsou adaptéry, které ovládají komplexní grafické prvky aplikace a tím odlehčují zátěž jednotlivých „aktivit“. Poslední kategorií komponent jsou grafické návrhy aplikace pomocí jazyka XML.

4.5.1 Coroutines

Coroutines jsou nástroj, který byl v jazyku Kotlin představen ve verzi 1.1. Tento nástroj slouží k paralelnímu spouštění kódu. Jedná se o odlehčenou alternativu k běžným vláknům. Coroutines jsou implementovány na nejnižší možné úrovni, a tím je umožněno ostatním knihovnám využívat API, které tento nástroj nabízí.

Na rozdíl od vláken, tak Coroutines jsou nezávislé na operačním systému. Každá blokující funkce je kompilátorem převedena do série blokujících volání. Před každým blokujícím voláním je následující stav uložen ve třídě generované kompilátorem. Při návratu z blokujícího volání je tento stav obnoven. Tímto způsobem je umožněno v rámci programu spouštět desetitisíce těchto Coroutines, zatímco vláknům by při tomto počtu velmi rychle došla operační paměť.[15]

```
lifecycleScope.launch(Dispatchers.IO) {
    db.logoutUser()
    val responseResult = ApiHandler.login(email, password, db)
    withContext(Dispatchers.Main) {
        val resultView = findViewById<TextView>(R.id.loginResponseView)
        val progressBar = findViewById<ProgressBar>(R.id.loginProgressBar)
        resultView.text = responseResult
        progressBar.visibility = View.INVISIBLE
        loginBtn.isEnabled = true
        loginBtn.isClickable = true
        if(responseResult == "Úspěch")
```

```
            startActivity(Intent(this@LoginActivity,
FlightListActivity::class.java))
    }
}
```

Zdrojový kód 2 *– Použití Coroutines v mobilní aplikaci (zdroj vlastní)*

4.5.2 SQLite

Je odlehčený databázový systém, který nevyžaduje konfigurace ani server. Zároveň se jedná o open-source řešení a je zdarma i pro komerční použití. SQLite zapisuje a čte přímo do běžných souborů. Tento databázový systém mimo běžné tabulky s cizími klíči dále podporuje spouštěče a pohledy. Soubory databáze lze bez problému přenášet mezi systémy různých architektur.

Datové typy v SQLite jsou slabě typované, ale autoři jej raději označují jako flexibilně typované. Při pokusu vložení čísla v podobě textového řetězce do sloupce datového typu integer, tak se databáze pokusí převést textový řetězec na integer stejně jako běžné databázové systémy, ale pokud textový řetězec bude obsahovat běžný text, který je nepřeveditelný na integer, tak místo navrácení chyby dojde k uložení řetězcové hodnoty nehledě na datový typ sloupce. V SQLite také nenalezneme specifikace maximálních délek řetězců a zároveň nenabízí oddělené datové typy boolean a datetime. Boolean je v databázi jednoduše reprezentován jako čísla 0 a 1. Datetime pak lze zapsat jako text, nebo jako číslo reprezentující počet sekund od roku 1970. Vzhledem k takovémuto přístupu k datům pak dokonce není nutné datový typ sloupce ani specifikovat. Během vývoje SQLite byla do systému implementována chyba, která umožňovala, aby primární klíč obsahoval hodnotu null. Při odhalení této chyby však existovalo již několik databází, které byly kriticky závislé na této chybě a od té doby se z této chyby stala vlastnost tohoto systému.[13]

Tabulka 1 – Datové typy SQLite (zdroj: [13])

Datový typ sloupce	Možné hodnoty sloupce
Integer	Integer, Real, Text, Blob
Real	Real, Text, Blob
Text	Text, Blob
Blob	Integer, Real, Text, Blob

4.5.3 Rozhraní lokální databáze

Hlavním úkolem tohoto rozhraní je správa dat v lokální databázi. Zde se nachází všechny metody vytváření, získávání, upravování a mazání dat. Toto rozhraní zároveň obsahuje kompletní předpis pro vytvoření samotné databáze.

Při vkládání dat dochází nejdříve ke kontrole, jestli se data v databázi již nenachází. Tím se zabraňuje vkládání duplicitních dat. Následně pomocí třídy ContentValues se přiřazují hodnoty dat jednotlivým sloupcům v databázi a následně se tyto hodnoty předávají samotné databázi k vložení. Nakonec se vrací zpráva o úspěšném nebo neúspěšném vložení dat. Vzhledem k podobnosti operací vkládání a upravování dat v databázi, tak některé z metod implementují obě tyto operace a zároveň tak zabraňují psaní duplicitního kódu.

```
fun addAircraft(aircraft: Aircraft): Boolean {
    val db = this.writableDatabase
    if(getAircraft(aircraft.regist(), aircraft.company(), null) == null) {
        val companyId = getCompanyId(aircraft.company(), true)
        val values = ContentValues().apply{
            put("model", aircraft.model())
            put("registration", aircraft.regist())
            put("type", EnumAircraftType.valueOf(aircraft.type()))
            put("company", companyId)
        }
        db.insert("Aircrafts", null, values)
        return true
    }
    return false
}
```

Zdrojový kód 3 *– Metoda rozhraní s lokální databází (zdroj vlastní)*

Získávání dat z databáze je komplexnější operace, kde metody rozhraní nabízí možnost získat záznam z databáze jak kombinací dat jednoznačně identifikující jeden záznam, nebo na základě unikátního identifikátoru záznamu. Zároveň v rozhraní nalezneme metody pro získání všech dat z konkrétních tabulek a v případě komplexních dat jakými jsou například samotné záznamy letů, tak umožňuje i získávání dat podle zadaných podmínek například získávání záznamů v konkrétním časovém intervalu.

4.5.4 Rozhraní komunikace s restAPI

Komunikaci mobilní aplikace se serverem zajišťuje rozhraní implementující metody pro realizaci autentizace uživatele společně se získáváním a nahráváním dat. K navázání komunikace se serverem pomocí protokolu http aplikace využívá knihovnu *khttp-android.* Rozhraní nejdříve zasílá příslušný dotaz na server pomocí této knihovny a po obdržení odpovědi v datovém formátu JSON, tato data deserializuje do objektů aplikace. Provádění všech těchto dotazů je pomocí Coroutines spouštěno mimo hlavní vlákno aplikace k zamezení zamrznutí grafického rozhraní při čekání na odpověď serveru. Každý dotaz zároveň má pevně

stanovený čas, při kterém přestane očekávat odpověď od serveru a prohlásí jej za nedostupný. V případě neúspěšného navázání spojení metody tohoto rozhraní vrací stavový kód odpověď a tomu příslušný textový popis.

```
val response = khttp.post(
    url = "https://SERVER/api/Auth/login",
    headers = mapOf(
        "Content-Type" to "application/json"
    ),
    data = loginParametersJSON,
    timeout = 1.0
)
val contentType : String? = response.headers["Content-Type"]
if(response.statusCode != 200 || contentType != "application/json"){
    Log.d("login", response.text)
    return response.text
}
```

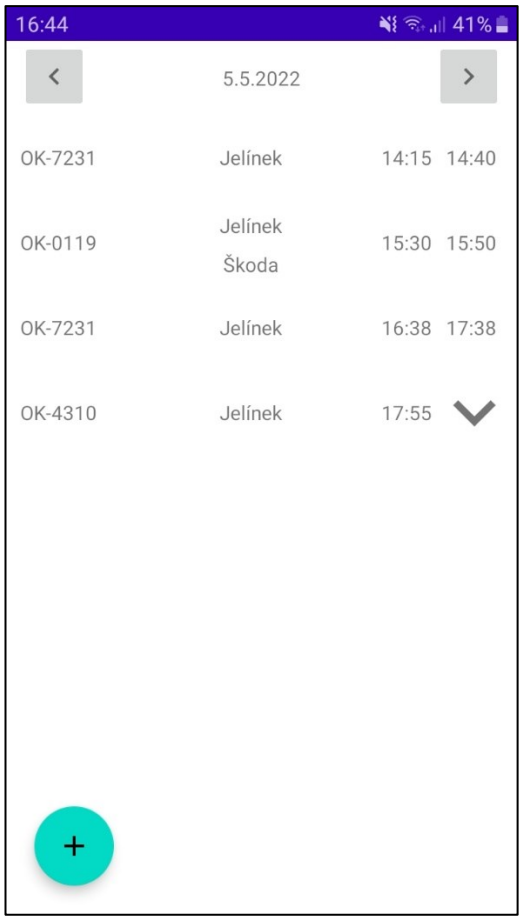
Zdrojový kód 4 *– Dotaz v rozhraní mobilní aplikace na server (zdroj vlastní)*

4.5.5 Aktivity mobilní aplikace

Zde jsou popsány všechny „aktivity“, které aplikace nabízí. Každá z těchto „aktivit“ obsluhuje jednu stránku grafického rozhraní.

Přihlášení uživatele je první „aktivitou“, která je uživateli zobrazena. Před vstupem do aplikace je zapotřebí uživatele autentizovat na serveru. Po zadání přihlašovacích údajů se tyto údaje předávají komunikačnímu rozhraní a následně se asynchronně čeká na odpověď. Při neúspěšné autentizaci je uživateli předána chybová zpráva a v případě úspěchu tato „aktivita“ pomocí komunikačního rozhraní zasílá dotaz pro vytvoření obrazu části serverové databáze. Pokud i tento dotaz se uskuteční úspěšně, tak je uživatel přesměrován na „aktivitu“ seznamu letů.

Seznam letů má za úkol zobrazovat všechny zaznamenané lety nacházející se v lokální databázi. To je ovšem vzhledem k omezené velikosti obrazovky uživatele nemožné, proto jsou zobrazovány pouze lety jednoho konkrétního dne. K navigaci mezi jednotlivými dny může uživatel využít orientovaných tlačítek, které zobrazovaný den posouvají o den vpřed, nebo vzad. Uživatel zároveň může vybrat konkrétní den pomocí grafické komponenty DatePicker z jazyka Kotlin. K zobrazení seznamu letů je využíváno další grafické komponenty z jazyka Kotlin, a to komponenty RecyclerView. Ovládání této komponenty je komplexní a z těchto důvodů je její ovládání odděleno do vlastního adaptéru.



***Obrázek 7** – Seznam letů mobilní aplikace (zdroj vlastní)*

Vytvoření nového záznamu letu a úprava existujícího záznamu letu zajišťují dvě velmi podobné „aktivity“. Jejich úkolem je zobrazení formuláře odpovídající záznamu letu. K zadávání dat a časů je využíváno příslušných komponent DatePicker a TimePicker, který nejen zpříjemňují způsob zadávání informací, ale zároveň zajišťují ochranu před nežádoucím vstupům do aplikace. Mimo textovou položku „poznámka“ jsou všechny ostatní položky zadávány prostřednictvím listů. Jednotlivé listy jsou zobrazovány v dialogových oknech společně s vyhledávacím polem, které dále zjednodušuje zadávání dat. V případě, že se jedná o volitelnou položku, tak je v dialogovém okně také nabízena možnost vložená data smazat. Formulář provádí jak dynamickou, tak statickou kontrolu zadaných dat. Při dynamické kontrole jsou sledované položky kontrolované tak, aby bylo možné upozornit uživatele ihned po zadání špatných dat. Tyto kontroly jsou zpravidla informační pro uživatele na zadávání nekonzistentních dat, avšak z pohledu systému se jedná o validní data a při odeslání formuláře jsou náležitě uložena. Na druhou stranu statické kontroly se provádí až při odeslání formuláře, kde tyto kontroly hlídají validnost dat z pohledu systém. Díky statickým kontrolám, tak lze zamezit vstupu škodlivým datům do systému. Následující kód ukazuje dynamickou kontrolu,

jestli let má zadán druhého člena posádky, pokud by tam být měl na základě funkce pilota, který toto specifikuje.

```
when (pilotFunction) {
    SOLO, SOLODVOJ, SEXAMSOLO -> {
        nfCrewTwoErrTV.text =
            SpannableStringBuilder("Let by měl mít druhého člena posádky.")
        nfCrewTwoErrTV.visibility = View.VISIBLE
    }
    SINSTR, VICPIC, SEXAMDVOJ -> {
        nfCrewTwoErrTV.visibility = View.GONE
    }
}
```

Zdrojový kód 5 *– Dynamická kontrola pole formuláře (zdroj vlastní)*

4.5.6 Grafické komponenty mobilní aplikace

Tyto komponenty slouží ke specifikaci grafického zobrazení jednotlivých stránek aplikace, které jsou na rozdíl od předchozích komponent implementované v jazyku XML. Pro základní rozložení stránky je využíváno komponenty ConstraintLayout, která umožňuje responzivní vkládání prvků. Umisťování prvků je řešeno pomocí vazeb k okrajům zobrazovacího okna, nebo je možné vytvářet mezi jednotlivými prvky. Tímto způsobem jsou navrženy stránky pro všechny „aktivity“, které je následně ovládají. Vzhledem ke stádiu vývoje, ve kterém se systém nachází, je dokonce nevhodné vytvářet komplexní a na oko příjemné UI, protože aktuální návrh aplikace nelze považovat za dokončený a může se v budoucnu významně změnit.

4.6 Synchronizace mobilní aplikace se serverovou aplikací

Serverová aplikace implementuje rozhraní restAPI pro komunikaci s klientskými aplikacemi. Vzhledem k tomu, že toto rozhraní nabízí neveřejná data, je zapotřebí toto rozhraní zabezpečit. Předpokládá se, že služby nabízené serverovou aplikací budou zabezpečené bezpečnostní vrstvou SSL k zabránění odposlechu a dalších hrozeb spojených s komunikací prostřednictvím veřejného internetu. Dále je zapotřebí tato data zabezpečit proti zneužití neautorizovaným uživatelem. Toto rozhraní veřejně nabízí možnost autentizace, kde po zaslání uživatelských údajů server provede autentizaci a po jejím úspěšném provedení zašle klientovi pseudonáhodně vygenerovaných šedesát znaků dlouhý token. Následně si tento token server uloží ve vlastní databázi k danému uživateli a klient již může pomocí tohoto tokenu zasílat dotazy na celé API.

Mezi nabízená data rozhraním patří získávání obsahu jednotlivých tabulek serverové databáze a navíc umožňuje klientovi upravovat a mazat lety. Dodatečně jsou nabízena všechna data pod jedním dotazem pro účely optimalizace, když si klient po úspěšné autentizaci vytváří lokální obraz serverové databáze. Bez této možnosti by klient musel zasílat dotaz na každou tabulku zvlášť a to vzhledem vysoké režii komunikace je nežádoucí. Tímto způsobem se podařilo ušetřit zhruba pět sekund z doby vytváření lokální kopie serverové databáze. Dosud bylo řečeno, že klient vytváří kompletní obraz serverové databáze, ale to není tak úplně pravda. Server zasílá klientovi pouze část obsahu své databáze, která je pro klienta relevantní. Relevantními daty se myslí záznamy letů a číselníky, které jsou přímo spojeny se společností, u které je klient registrován.

```
Route::group(['middleware'=>['ApiUserCheck']], function(){
    Route::post('/dbImage/get', [Api::class, 'getDbImage']);
    Route::post('/towFlights/get', [Api::class, 'getTowFlights']);
    Route::post('/flights/get', [Api::class, 'getFlights']);
    Route::post('/flights/addUpdate', [Api::class, 'addUpdateFlights']);
    Route::post('/flights/delete', [Api::class, 'delFlight']);
    Route::post('/aircrafts/get', [Api::class, 'getAircrafts']);
    Route::post('/members/get', [Api::class, 'getMembers']);
    Route::post('/dialItems/get', [Api::class, 'getDialItems']);
    Route::post('/airports/get', [Api::class, 'getAirports']);
});
```

Zdrojový kód 6 *– Nabízené metody rozhraním restAPI (zdroj vlastní)*

Řešení samotné komunikace klienta se serverem je ta jednodušší část problému synchronizace. K zajištění plnohodnotné synchronizace a pro spolehlivé udržení konzistence dat je zapotřebí vyřešit řadu vzestupujících problémů spojených s druhem dat, s kterými systém pracuje. Záznam o letu není vždy vytvořen v jednom okamžiku a také jej nemusí vytvářet pouze jeden uživatel. Celý tento problém je ještě složitější o podporu odpojení klienta od centrální databáze na dobu neurčitou. Tento systém implementuje pouze základní komunikační rozhraní pro synchronizaci, ale nezohledňuje všechny tyto komplexní situace. Následující scénáře slouží k nastínění komplexního problému synchronizace.

V rámci tohoto scénáře vystupuje jeden pilot s letadlem společně s dvěma uživateli, kteří se snaží vytvořit záznam o letu pilota. První z uživatelů, který bude nazýván jako Jakub, sedí na kontrolní věži, kde zapisuje probíhající lety do serverové aplikace. Druhý uživatel, který bude nazýván jako Matěj, se nachází na okraji VPD, kde pomáhá pilotu s přípravami ke vzletu. V okamžiku vzletu Jakub zapíše do serverové aplikace záznam o letu konkrétního letadla s konkrétním pilotem a vzletem v konkrétním čase. Totéž provede Matěj, ale vzhledem k tomu,

že na okraji VPD nemá připojení k internetu, tak daný záznam vytvoří v mobilní aplikaci do lokální databáze. Následuje nepředvídatelně dlouhá doba, než Matěj získá připojení a bude moci nahrát záznam o letu na server. Zároveň se u obou záznamů toho samého letu jedná o nedokončený záznam, protože obsahují pouze čas vzletu, ale už ne čas přistání, neboť let stále probíhá. Při pokusu o nahrání záznamu o letu Matějem, musí být systémem spolehlivě rozpoznáno, že se jedná o duplicitní záznam, a nabídnuto tak uživateli řešit tento konflikt. Následně při dokončení letu musí být zaznamenán čas přistání. Nejen, že se celý dosud popsaný scénář opakuje, ale zároveň se nemusí jednat ani o dvojici uživatelů Jakuba a Matěje. Tudíž pokusit se zapsat čas přistání tohoto letu můžou i například další dva různí uživatelé, protože Jakub s Matějem za doby průběhu letu museli odjet například do práce. Po zaznamenání času a místě přistání je již let dokončený a připravený k exportování.

4.6.1 JSON

JavaScript Object Notation je jednoduchá datová struktura pro přenos dat mezi systémy. Tato struktura se vyznačuje jednoduchou čitelností a zároveň je jednoduše zpracovatelná počítačem. JSON je kompletně nezávislí na programovacím jazyku. Používané konvence jsou velmi podobné rodině jazyků C.

Základní součástí datové struktury JSON je objekt, který může nést jméno a hodnotu, která je ohraničena složenými závorkami. K objektu tak lze přistupovat jako k páru klíče a hodnoty. Objekt může obsahovat další objekty, pole a samotné hodnoty. Pole jsou ohraničena hranatými závorkami a jednotlivé položky se oddělují čárkou. Tyto položky mohou také obsahovat další objekty, pole a hodnoty. Samotnými hodnotami pak jsou páry textového klíče a hodnoty.[14]

ZÁVĚR

Cílem této bakalářské práce bylo navrhnout a implementovat mobilní aplikaci pro záznam letů všeobecného letectví s možností exportovat tyto záznamy na centrální databáze. Všechny tyto požadavky, které na práci byly kladeny, jsou úspěšně splněny. Autor této práce je zároveň i jejím budoucím uživatelem. Díky spojení role návrháře se zadavatelem v jedné osobě je vyřešen problém stanovení špatných a nedostatečných požadavků na systém.

Ovšem i za těchto podmínek se vývoj systému setkal s radikálními změnami v jeho specifikaci a z toho vyplývajícího návrhu. Nejvíce viditelnou změnou, bylo upuštění od myšlenky, kde v první řadě vystupovala mobilní aplikace, která pouze odesílala data na centrální SQL server. Aktuální řešení je založené na centrální serverové aplikaci spojené s hlavní SQL databází a následně mobilní aplikace, která zastupuje minoritní roli doplňování funkčních nedostatků serverové aplikace. Díky novému pojetí celého systému se podařilo odstranit vážný nedostatek systému, kterým by byla závislost uživatele vlastnit mobilní zařízení s operačním systémem Android. Aktuálně se tak každý uživatel může bez problému interagovat se systémem pomocí webového prohlížeče a použití rozšířených funkcí mobilní aplikace se tak stává pouze volitelnou možností.

Tato práce byla pro autora velmi přínosná, neboť zahrnuje používání řady technologií mezi, které patří i technologie, se kterými se autor setkal při řešení této práce poprvé v životě. Celkově se jednalo o velkou výzvu, kde autor samostatně navrhl serverovou aplikaci pomocí jazyka PHP, mobilní aplikaci v jazyce Kotlin, společně s návrhem uživatelského rozhraní obou těchto aplikací. Autor dále navrhl datovou vrstvu ve dvou různých databázových systémech a nakonec bylo zapotřebí všechny tyto komponenty propojit.

Systém v aktuálním stavu nelze považovat za dokončený a připravený pro produkční nasazení. Chybí zde několik vlastností, které jsou velmi důležité pro produkční nasazení. Mezi tyto vlastnosti patří možnost obnovy zapomenutého hesla uživatele společně s možností modifikace uživatelského účtu. K tomu bude zapotřebí implementovat automatické zasílání zpráv na uživatelský e-mail. Dále bude zapotřebí vypracovat návody s názorným použitím aplikací, protože věková kategorie cílových uživatelů sahá od náctiletých až po seniory, pro které může být používání aplikací velmi zmatečné. V neposlední řadě bude nutné korektně vyřešit celkový problém týkající se synchronizace serveru a klientů, kde bude vhodné nabídnout uživateli nástroje pro správu synchronizace. Další důležitou funkcí bude umožnit uživatelům odstraňovat položky číselníků, na kterých jsou již závislé některé záznamy letů tak, že se

položky odstraní pouze z uživatelského rozhraní. Nakonec bude nutné zajistit produkční server společně s konfiguracemi nabízených služeb a s možností celý systém zálohovat.

Autor má dále v plánu implementovat i další funkce, které už nejsou pro správnou funkčnost systém nutné. Do tohoto plán spadá implementace plnohodnotné podpory pro zaznamenávání aerovleků, kde je zapotřebí provázat dva záznamy letů mezi sebou. Další vhodnou funkcí je propojení systému s externí aplikací OGN logbook, která v reálném čase nabízí data obsahující časy vzletu a přistání právě prováděných letů na území několika zemí Evropy. Poslední plánované rozšíření systému je optimalizace mobilní aplikace tak, aby mohl jednoduše vytvořit záznam o letu i pilot samotný. K tomu bude zapotřebí využívat systému GPS pro automatické vyhodnocení časů vzletu a přistání, kde pilot před samotným letem vyplní nezbytné informace o letu, jako jsou informace o letadle, funkci pilota, způsobu vzletu, druhu letu a úkolu letu. Nadále ostatní položky, jako letiště vzletu a přistání společně se všemi časy bude možné určit pomocí systému GPS. Pilot se tak bude mít možnost plnohodnotně věnovat řízení letadla, zatímco aplikace za něj automaticky zaznamená informace vznikající v průběhu letu.

POUŽITÁ LITERATURA

[1] SIMS, Gary. *Android authority* [online]. 10. 8. 2019 [cit. 2022-07-05]. Dostupné z: https://www.androidauthority.com/develop-android-apps-languages-learn-391008/

[2] HARKIRAN, Kaur. *Top Programming Languages for Android App Development* [online]. 26. 4. 2022 [cit. 2022-07-05]. Dostupné z: https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/

[3] BOSE, S. – KUNDU, A. – MUKHERJEE, M. – BENERJEE, M. A comparative study: Java vs Kotlin programming in android application development. *International Journal of advanced research in computer science,* červen 2018, Volume 9, No. 3, ISSN 0976-5697

[4] WILSON, Ella. *Medium* [online]. 23. 6. 2021 [cit. 2022-07-05]. Dostupné z: https://medium.com/quick-code/php-vs-python-which-is-best-for-web-applications-in-2021-b7ad3fe0743a

[5] *Python Programming Language* [online]. 3. 4. 2022 [cit. 2022-07-05]. Dostupné z: https://www.geeksforgeeks.org/python-programming-language/

[6] KOTHARI, Josh. *Pyrhon vs PHP* [online]. 9. 8. 2021 [cit. 2022-07-05]. Dostupné z: https://www.geeksforgeeks.org/python-vs-php/

[7] JANKOV, Tonino. *MariaDB vs MySQL: A Database Technologies Rundown* [online]. 3. 4. 2022 [cit. 2022-07-05]. Dostupné z: https://kinsta.com/blog/mariadb-vs-mysql/

[8] MOMJIAN, Bruce. *PostgreSQL intrudoction and concepts*. 1. vyd. One Lake Street: Pearson Education Corporate Sales Division, 2000. 490 s. ISBN 0-201-70331-9

[9] *Zápisníky letů pro piloty kluzáků a motorových kluzáků* [online]. 28. 11. 2019 [cit. 2022-07-05]. Dostupné z: https://www.caa.cz/news/zapisniky-letu-pro-piloty-kluzaku-a-motorovych-kluzaku/

[10] Cope, James. *What's a Peer-to-Peer (P2P) Network?* [online]. 8. 4. 2002 [cit. 2022-07-05] Dostupné z: https://www.computerworld.com/article/2588287/networking-peer-to-peer-network.html

[11] *Client Server Architecture* [online]. 13. 12. 2018 [cit. 2022-07-05]. Dostupné z: https://cio-wiki.org/wiki/Client_Server_Architecture

[12] *Laravel dokumentace* [online]. [2013?] [cit. 2022-07-05]. Dostupné z: https://laravel.com/docs/9.x

[13] *SQLite dokumentace* [online]. 9. 5. 2000 [cit. 2022-07-05]. Dostupné z: https://www.sqlite.org/docs.html

[14] *Introducing JSON* [online]. [cit. 2022-07-05]. Dostupné z: https://www.json.org/json-en.html

[15] BÁRTA, Milan. Kotlin Coroutines in Android[online]. 16. 4. 2018 [cit. 2022-01-26]. Dostupné z: https://sourcediving.com/kotlin-coroutines-in-android-e2d5bb02c275

PŘÍLOHY

PŘÍLOHA A – WEBOVÁ APLIKACE

Obsahuje kompletní zdrojové kódy k webové aplikaci serveru společně s migračním souborem pro vytvoření databáze. K spuštění je zapotřebí zajistit připojení k databázi a nástroj Composer. Alternativně lze fungující aplikaci najít na https://www.x.jhp.cz:6676/.

PŘÍLOHA B – MOBILNÍ APLIKACE

Obsahuje kompletní zdrojové kódy mobilní aplikace připravené ke spuštění vývojovým prostředím Android Studio. Tato aplikace má v sobě přímo vepsanou komunikaci se serverem https://www.x.jhp.cz:6676/. Tudíž bez úpravy komunikačního rozhraní nelze systém zprovoznit lokálně.