Elektronická evidence dokladů v prostředí leteckého provozovatele

Vojtěch Zapletal

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Vojtěch Zapletal**
Osobní číslo:	**A21296**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Elektronická evidence dokladů v prostředí leteckého provozovatele**
Téma práce anglicky:	**Electronic Document Records in the Environment of an Aviation Operator**

Zásady pro vypracování

1. Vypracujte literární rešerši na zadané téma.
2. Navrhněte systém pro elektronickou evidenci dokladů nezbytných v leteckém provozu.
3. Vytvořte navržený systém dle návrhu.
4. Věnujte pozornost zabezpečení systému.
5. Otestujte funkčnost systému.
6. Implementaci a testování vhodně popište.

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

Seznam doporučené literatury:

1. GRINBERG, Miguel. *Flask web development*. Sebastopol, CA: O'Reilly, 2014. ISBN 1449372627.
2. BANKS, Alex a PORCELLO, Eve. *Learning React: modern patterns for developing React apps*. Second edition. Sebastopol, CA: O'Reilly Media, 2020. ISBN 1492051721.
3. NELSON, Brett. *Getting to Know Vue.js: Learn to Build Single Page Applications in Vue from Scratch*. Imprint: Apress, 2018. ISBN 978-1484237809.
4. *Vue.js – Documentation* [online]. 2023. Dostupné také z: https://vuejs.org/guide/.
5. *Ionic Framework – Documentation* [online]. 2023. Dostupné také z: https://ionicframework.com/docs.
6. *React – Documentation* [online]. 2023. Dostupné také z: https://react.dev.
7. *Flask – Documentation* [online]. 2023. Dostupné také z: https://flask.palletsprojects.com/.

Vedoucí bakalářské práce: **Ing. Tomáš Vogeltanz, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

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

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

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

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.
- Prohlašuji, že při tvorbě této práce jsem použil/a nástroj generativního modelu AI ChatGPT; www.openai.com za účelem gramatické korektury textu. Po použití tohoto nástroje jsem provedl/a kontrolu obsahu a přebírám za něj plnou zodpovědnost.

Ve Zlíně, dne 10.05.2024

Vojtěch Zapletal v. r.
……
podpis studenta

ABSTRAKT

Tato bakalářská práce se zabývá, tvorbou elektronické evidence dokladů pro letecké provozovatele. Cílem tohoto projektu je vytvořit nástroj, díky kterému je komunikace (předávání dokladů) mezi posádkami a ekonomickým oddělením společnosti provozovatele rychlejší a efektivnější. Doposud proces probíhá tak, že se doklady sbírají a následně osobně předávají. Výstupem této bakalářské práce je aplikace, která umožní leteckému provozovateli snazší správu dokladů – výdajů posádek. Vytvořený software si klade za cíl tento proces zrychlit, zjednodušit a zamezit komplikacím ve formě prodlení s předáním a následné fakturaci s tím spojené a případné ztrátě dokladů. Předpokládá se, že uživatelsky přívětivý program bude sloužit svému účelu a zefektivní část práce koncovému zákazníkovi.

Klíčová slova: Python, JavaScript, FastAPI, Vue.js, API

ABSTRACT

This bachelor thesis focuses on the development of an electronic document records in the environment of an aviation. The aim of this project is to create a tool that facilitates faster and more efficient communication (document transfer) between crew members and the accounting department of the operator company. Until now, the process involved collecting documents and then handing them over in person. The output of this bachelor thesis is an application that allows the airline operator to manage crew expenses more easily. The created software aims to speed up this process, simplify it, and prevent complications such as delays in handing over and the subsequent billing associated with potential loss of documents. It is anticipated that this user friendly program will serve its purpose and make the work of the end customer more efficient.

Keywords: Python, JavaScript, FastAPI, Vue.js, API

Rád bych poděkoval vedoucímu této práce panu Ing. Tomáši Vogeltanzovi Ph.D. za cenné rady a vedení v průběhu psaní této práce. Obzvláště bych pak chtěl poděkovat mé manželce a dětem za trpělivost a podporu, kterou mi během psaní této práce a i celého studia na univerzitě projevili.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Tématem této bakalářské práce je tvorba systému fullstack aplikace, kde budou využity a uplatněny znalosti získané během studia na Fakultě aplikované informatiky UTB. Nápad a návrh na tuto aplikaci vychází z mé praxe a práce pro letecké provozovatele v oblasti soukromé dopravy. Je běžné, že posádky soukromých letadel hradí velkou část nákladu na let, jako je palivo, letištní poplatky, hotely nebo transport na místě. Z každé této transakce jsou sbírány účtenky, faktury, které jsou následně vloženy do obálky a předány účetní v sídle provozovatele. Odevzdané doklady jsou následně zaúčtovány a přefakturovány cestujícím nebo majiteli letadla.

Během studia na UTB jsme se zabývali tématy jako je tvorba webových aplikací, webových stránek, mobilních aplikaci, databází a dalších, proto jsem se rozhodl pro tento účel spojit a prohloubit znalosti získané během studia a praxí získanou v letectví. Mým cílem je touto prací vyřešit výše zmíněné zastaralé řešení předávání dokladů. Pro svou bakalářskou práci jsem si vybral technologie, které jsou v dnešním světě velmi rozšířené jako Python, Vue 3 a další, protože jsem přesvědčen, že nabízejí profesionální řešení, širokou škálu zdrojů a díky tomu i robustní bezpečné řešení pro tuto konkrétní aplikaci.

První, teoretická část práce, je rozdělena na dvě kapitoly. První z nich se bude zabývat rešerší existujících řešení a druhá pak popisem jednotlivých technologií, které budou pro implementaci backendu a frontendu použity.

Praktická část se bude zabývat návrhem systému, kde budou popsány požadavky na aplikaci, strukturu databáze, rozdělení jednotlivých endpointů a další. Dále bude v této části popis implementace backendu, využití frameworku FastAPI a také implementace frontendu, zabezpečením systému pomocí autorizace a autentizace, nahrání aplikace do cloudového prostředí a jejím následným otestováním, včetně testů provedených nezávislým uživatelem. Závěr praktické části nastíní návrhy do budoucna, jako je rozšíření aplikace o další platformy, prvky umělé inteligence a další možnosti rozvoje.

I. TEORETICKÁ ČÁST

1 REŠERŠE EXISTUJÍCÍCH ŘEŠENÍ

1.1 Potřeba digitální evidence dokladů

V současné době je běžnou praxí, že se doklady a faktury shromažďují v papírové podobě, což je proces, který může být nejen neefektivní a pomalý, ale také nákladný. Tato tradiční metoda evidence dokladů vede ke značným prodlevám v účetním procesu, což může mít negativní dopad na finanční toky firmy. Kromě toho ruční manipulace s papírovými dokumenty přináší riziko ztráty nebo poškození důležitých dokladů, což může vést k dalším zdržením a potenciálním finančním ztrátám.

Dalším významným aspektem je časová prodleva mezi obdržením dokladů a jejich přefakturováním konečnému klientovi. V tradičním systému může tento proces trvat několik dní až týdnů, což vede k pomalejšímu oběhu peněz. Ve specifickém prostředí leteckého provozovatele, který provozuje a pronajímá soukromá letadla, je toto zpoždění komplikací a může společnost dostat do komplikované pozice vůči majitelům letadel či zákazníkům, kteří si letadla pronajímají.

Digitalizace procesu evidence dokladů nabízí řešení těchto problémů tím, že značně zrychluje celý proces, minimalizuje riziko chyb, ztráty nebo poškození dokumentů a zjednodušuje přístup k účetním záznamům. Tento přístup nejen snižuje administrativní zátěž a náklady spojené s papírovými dokumenty, ale také zvyšuje transparentnost a usnadňuje rychlé přefakturování a zvyšuje úroveň poskytnutých služeb zákazníkům. Vzhledem k uvedeným výhodám je evidentní, že digitalizace dokladů je nejen žádoucí, ale v mnoha případech nevyhnutelným krokem k dosažení vyšší efektivity.

1.2 Srovnání s podobnými aplikacemi na trhu

Na trhu existuje několik aplikací určených pro digitální evidenci dokladů, nicméně žádná z nich nevyhovuje plně specifickým potřebám leteckého provozovatele. Při bližším zkoumání alternativ byla identifikována například aplikace Moss. Avšak i aplikace Moss je spojena s určitými omezeními, která značně snižují její přínos pro projekt našeho charakteru.

V následujících bodech budou podrobněji rozebrány také další dostupné aplikace na trhu a bude vysvětleno proč tyto alternativy nejsou ideálním řešením pro naše specifické potřeby.

1.2.1 Expensify

V rámci průzkumu existujících řešení na trhu byl analyzován systém Expensify, který se řadí mezi nejrozšířenější nástroje pro správu firemních výdajů. Expensify nabízí možnost napojení na různé účetní softwary, což je výhoda pro společnosti operující zejména v západní Evropě, jako je Francie, kde jsou jeho služby široce využívány. Přestože tento systém poskytuje širokou škálu řešení pro správu výdajů, má určitá omezení, což vedlo k rozhodnutí vyvinout vlastní řešení. [1]

Hlavním nedostatkem systému Expensify pro využití u leteckých provozovatelů je jeho nadměrná komplexnost a nedostatečná možnost přizpůsobení konkrétním požadavkům. Expensify je sice velmi komplexní nástroj pro správu výdajů, ale jeho rozsáhlé funkcionality jsou pro účely leteckých provozovatelů zbytečně komplikované a neumožňují nastavit kategorizaci výdajů dle daných specifikací. Například nelze jednoduše upravit systém tak, aby rozlišoval výdaje dle přiřazení k jednotlivým letadlům nebo tripům, což je pro operace leteckých provozovatelů klíčové.

Dále je třeba zmínit, že i přes široké možnosti integrace s účetními systémy, které Expensify nabízí, by vyžadovalo další značné investice do nových účetních systémů, které ani nejsou plně využitelné v prostředí slovenského či českého účetnictví. Plná integrace by významně zvýšila provozní náklady. Konkrétně různé tarify Expensify mohou dosahovat až 36 amerických dolarů za uživatele měsíčně, což je pro rozpočet neúměrně nákladné. [1]

Z těchto důvodů nebyl využit Expensify jako systém na správu dokladů a raději byl vyvinut vlastní systém, který bude přesně odpovídat potřebám provozovatele a bude mít lepší integraci s jeho interními procesy, zatímco bude stále udržovat nízké provozní náklady.

1.2.2 Evernote Scannable

Evernote Scannable představuje lehké řešení pro digitalizaci dokumentů, které na první pohled může vypadat jako vhodná alternativa pro uvedené potřeby. Tento software sice podporuje uchovávání různých typů dat, včetně účetních dokladů, vizitek a dalších, avšak jeho základní funkcionalita a omezené možnosti kategorizace neodpovídají specifickým požadavkům leteckého provozovatele. Hlavní problém spočívá v jeho nevhodnosti pro detailní kategorizaci a nemožnosti integrace s dalšími systémy, jako jsou technické deníky a spárování s konkrétními tripy, což je provozovatele klíčové. Ačkoli je Evernote Scannable cenově dostupnější než některé robustnější systémy, stále představuje finančně náročnější

řešení v porovnání s potenciálem vlastního vývoje aplikace, která by přesně odpovídala potřebám provozovatelů. Tyto nedostatky také vedly k rozhodnutí vyvinout vlastní řešení, které by bylo plně na míru zákazníka. [2]

1.2.3 Moss

Software Moss byl navržen pro efektivní sledování nákladů a využívá pokročilé technologie jako umělou inteligenci pro třídění dokladů a spárování plateb s výdaji na kreditní kartě, ale pro naše potřeby je výrazně nevhodný. Hlavním omezením je nutnost použití kreditní karty vydávané k účtu Moss, což v podstatě znemožňuje využití v námi nadefinovaném případě. Tento systém je navíc optimalizován primárně pro trhy západní Evropy, a jeho provoz je spojen s vysokými náklady. Pro provozovatele, kteří potřebují systém adaptovatelný na jejich specifické požadavky, a nezávislý na externích platebních řešeních, je Moss příliš restriktivní a nákladný. Jeho implementace by navíc nezaručila kompatibilitu se stávajícími procesy a nástroji, což by mohlo vést k dalším komplikacím. [3]

1.3 Shrnutí rešerše

Shrnutí vyhodnocení existujících řešení na trhu, jako jsou Expensify, Moss a Evernote Scannable, ukazuje, že žádná z těchto aplikací není plně vyhovující pro specifické potřeby tohoto projektu. Hlavními důvody, proč tyto systémy nejsou vhodné pro letecké provozovatele, jsou jejich vysoké náklady a nedostatečná flexibilita v kategorizaci dat.

Cena: Všechny zmíněné systémy přicházejí s podstatnými měsíčními nebo ročními náklady, které se mohou v dlouhodobém horizontu výrazně promítnout do celkových provozních výdajů firmy. Vývoj vlastního řešení by umožnil lepší kontrolu nad rozpočtem a optimalizaci nákladů.

Nemožnost plné kategorizace: Žádný z analyzovaných systémů nenabízí dostatečnou možnost adaptace kategorizačních pravidel tak, aby odpovídaly správnému zařazení konkrétní firmy. Je zásadní, aby systém umožňoval přesné kategorizování výdajů dle požadovaných parametrů a dokázal je přiřadit k jednotlivým tripům nebo letům.

Nedostatečná integrace s technickým deníkem letadla: Řešení, které je požadováno, musí být schopno přiřazovat jednotlivé položky k datům v technickém deníku, k tomu slouží položka trip, žádná z výše uvedených aplikací tuto možnost nenabízela.

Vzhledem k těmto nedostatkům je pro nás nejlepší volbou vyvinout vlastní aplikaci, která bude přesně odpovídat našim potřebám. Tento přístup nejen že sníží dlouhodobé náklady spojené s licenčními poplatky, ale také zabezpečí, že systém bude mít všechny potřebné funkce pro efektivní správu a analýzu firemních výdajů, a bude možno jej kombinovat se systémem evidence letů.

2 POUŽITÉ TECHNOLOGIE

2.1 Backend

V rámci vývoje backendu této bakalářské práce byl využit programovací jazyk Python. Tento jazyk je známý svou všestranností, snadnou čitelností a dynamickou typovou kontrolou, což jej činí ideálním nástrojem pro rychlý vývoj aplikací. Python je také jazykem vyšší úrovně, což znamená, že poskytuje abstrakci od nízkoúrovňových hardwarových systémů, což umožňuje soustředit se více na logiku aplikace, než na její technickou implementaci. [4]

Dalším důvodem, proč byl vybrán Python pro backend, je jeho rozsáhlá standardní knihovna. Tyto knihovny nabízejí již vytvořené moduly pro širokou škálu účelů, což umožňuje implementovat složité funkce s minimálním úsilím a zkrátit tak celkový čas potřebný k vývoji. Konkrétně byl využit framework FastAPI pro tvorbu API a SQLAlchemy pro komunikaci s databází, což jsou obě volby dobře podporované v Python komunitě. [5][6]

Python má také velmi dobrou podporu pro práci s cloudovými službami, což je v souladu s rozhodnutím hostovat aplikaci v cloudu. Tento přístup umožňuje využít výhod škálovatelnosti a flexibility cloudového hostingu.

Vzhledem k těmto faktorům lze konstatovat, že Python je vhodnou volbou pro vývoj backendu této aplikace. Jeho všestranost, společně s bohatým výběrem knihoven a frameworků, dokáže poskytnout spolehlivé zázemí pro rychlý vývoj softwaru.

2.1.1 FastAPI

FastAPI byl vybrán jako základní framework pro vývoj backendu tohoto projektu. Jak již název pro tento open-source framework napovídá („Fast“) je opravdu rychlý, a to jak ve zpracování dat, tak v tvorbě kódu samotného. [7]

Zde je několik klíčových důvodů, které jej odlišují od jiných populárních frameworků jako je Flask, jehož využití v projektu, bylo původně plánováno. Na rozdíl od Flasku poskytuje FastAPI již zabudovanou podporu pro asynchronní zpracování požadavků, což je benefitem pro hladký běh aplikace při obsluze souběžných požadavků. Tato vlastnost činí FastAPI ideální pro moderní webové aplikace, které vyžadují rychlé a efektivní zpracování dat.

Další z hlavních výhod FastAPI je jeho integrovaná podpora pro validaci a serializaci dat s využitím knihovny Pydantic. Tato integrace umožňuje snadnou validaci vstupních dat s

možností využití pokročilých typových kontrol, jako jsou validace e-mailových adres nebo URL. Tato funkcionalita zvyšuje bezpečnost aplikace tím, že zajistí, že všechna data přijímaná od klientů jsou správně typována a validována předtím, než jsou zpracována nebo uložena do databáze. [5]

Další předností FastAPI je snadná integrace s API dokumentačními nástroji jako Swagger UI a ReDoc, dostupnými přímo přes endpointy *„/docs“* a *„/redoc“*. Tyto nástroje usnadňují vývoj a testování API tím, že poskytují interaktivní uživatelské rozhraní pro prohlížení a interakci s API endpointy. Vývojáři mohou snadno a rychle ověřovat funkcionalitu endpointů a zajišťovat jejich správnou implementaci. [6]

FastAPI také nabízí vynikající podporu pro moderní autentizační schémata, což umožňuje snadnou implementaci bezpečnostních mechanismů, jako jsou tokeny JWT pro ověřování a autorizaci uživatelů. Tato vlastnost je klíčová pro vývoj bezpečných aplikací, které chrání citlivá uživatelská data a zajišťují integritu transakcí. [6]

Výběr FastAPI pro tento projekt byl též motivován snahou vytvořit aplikaci, která může efektivně sloužit různým typům zařízení, od desktopů po mobilní telefony. S FastAPI můžeme vyvíjet backend, který je nejen rychlý a flexibilní, ale také připravený na budoucí rozšíření a integraci s dalšími systémy a technologiemi.

2.1.2 MySQL

Jako databázový systém pro správu databáze našeho projektu byl zvolen MySQL, oblíbený relační databázový systém známý pro svou rychlost, snadnou instalaci, používání a správu. MySQL nám také umožňuje ukládat širokou škálu datových typů, včetně souborů, které jsou součástí našeho systému do databáze. Tento systém je kompatibilní s mnoha operačními systémy, včetně různých verzí Linuxu a Windows, což umožňuje jeho široké použití napříč platformami. Komunikace s databází je realizována prostřednictvím ORM knihovny SQLAlchemy, což umožňuje vytvářet SQL dotazy do Python kódu a zjednodušuje tak práci s databází. [8]

Pro přístup a používání SQL jazyka, je v rámci distribucí MySQL zahrnutý klientský program mysql. [8]

Integrace MySQL s Pythonem a FastAPI v Linuxovém cloudovém prostředí poskytuje flexibilní řešení pro správu databází. Kombinace SQL s Pythonem nám dává mnoho možností pro vydávání dotazů a zpracování jejich výsledků. Tato konfigurace zvyšuje

schopnost provádět i složité databázové operace a je ideálním řešením pro efektivní správu relačních databází v moderních cloudových aplikacích.

Díky této konfiguraci je vytvořená aplikace dobře připravena pro efektivní správu dat. Použití MySQL ve spojení s těmito technologiemi nám umožňuje vytvářet aplikace, které jsou nejen rychlé a spolehlivé, ale také se snadno přizpůsobí dle potřeb projektu.

2.1.3 Pyndantic

V naší aplikaci byla využita knihovna Pydantic pro validaci a serializaci dat, což je zásadní pro zajištění integrity a bezpečnosti dat při komunikaci s API a ukládání dat do databáze. Pydantic je silný nástroj, který pomáhá zamezit poslání a ukládání neplatných dat do databáze, což je klíčové pro prevenci bezpečnostních rizik, jako například SQL injection.

Hlavní výhody Pydantic spočívají v jeho schopnosti využívat takzvané type hints, která umožňují explicitně specifikovat očekávaný datový typ pro kontrolu a serializaci dat, což znamená méně kódu, jednodušší integraci a lepší spolupráci s nástroji IDE. Jádro validace v Pydantic je napsáno v Rustu, což z něj dělá jednu z nejrychlejších knihoven pro validaci dat v Pythonu. [9]

Pydantic také umožňuje generování JSON schémat z modelů, což usnadňuje integraci s dalšími nástroji a systémy. Tato knihovna podporuje mnoho typů z Python standardní knihovny včetně dataclass a TypedDict, a nabízí mnoho možností pro přizpůsobení, včetně vlastních validátorů a serializátorů. [9]

Dále Pydantic podporuje režimy strict True a False, což umožňuje buď striktní dodržování typů bez konverze, nebo naopak pokus o převod dat na správný typ, když to je vhodné. Tento mechanismus je výhodný pro zachování flexibility při zpracování dat. Pokud není uvedeno jinak strict mód je defaultně nastavený na False. [9]

Pydantic je široce používán ve více než 8,000 balíčcích na PyPI a je součástí několika velmi populárních knihoven jako FastAPI, huggingface, a další. Díky jeho rozsáhlému použití v praxi a ve velkých společnostech, jako jsou FAANG a 20 z 25 největších společností na NASDAQ, je Pydantic ověřeným a spolehlivým řešením pro každého, kdo potřebuje efektivně a bezpečně pracovat s daty v Pythonu. [9]

2.1.4 SQLAlchemy

V rámci backendové části naší aplikace byla zvolena knihovna SQLAlchemy, která zajistí vysokou úroveň abstrakce našeho kódu za využití ORM. ORM je programová vrstva, která umožňuje snadnější a efektivnější práci s relačními databázemi pomocí objektově orientovaného přístupu. Tato knihovna propojuje relační databáze s objektovými typy v Pythonu, což umožňuje programátorům manipulovat s databázovými daty bez přímého psaní SQL dotazů, ale za pomocí vytváření Pythonových tříd.

SQLAlchemy je známá pro svou flexibilitu a sílu, což dává vývojářům plnou kontrolu nad SQL operacemi a zachovává výhody vyšší úrovně abstrakce, kterou poskytuje ORM. Tato knihovna nabízí širokou škálu podpory a je tak vhodná pro použití v tomto projektu, protože nám zajistí snadnější rozšíření aplikace v budoucnu. [10]

Díky SQLAlchemy je kód, který komunikuje s databází, mnohem přehlednější a snadněji spravovatelný. Knihovna zjednodušuje práci s databází tím, že automatizuje mnoho rutinních úloh spojených s datovými operacemi a transakcemi, což zvyšuje produktivitu vývoje. Navíc umožňuje lepší ochranu proti běžným bezpečnostním hrozbám, jako je SQL injection, protože vývojáři nejsou přímo vystaveni surovým SQL příkazům. [10]

Využití SQLAlchemy v tomto projektu tedy přináší výhody v podobě zjednodušení práce s databází, zvýšení bezpečnosti a zlepšení organizace a čitelnosti kódu. Tímto způsobem může SQLAlchemy efektivně podporovat rychlý vývoj a škálovatelnost aplikace, což je zásadní pro splnění požadavků moderních aplikací.

2.1.5 JWT

Pro autentizaci uživatelů ve vyvíjené aplikaci byl zvolen mechanismus JSON Web Token (JWT), což je otevřený standard (RFC 7519) definující způsob, jak bezpečně přenášet informace mezi stranami ve formě kompaktního a samostatného JSON objektu. Tento formát je ideální pro naše účely z několika důvodů. [11]

JWT je způsob, jakým lze bezpečně přenášet informace mezi dvěma stranami v podobě JSON objektu. Tyto informace jsou ověřitelné a důvěryhodné, protože jsou digitálně podepsané. Podepisování může být provedeno pomocí tajného klíče s algoritmy jako je HMAC, nebo s veřejným/soukromým klíčem pomocí RSA nebo ECDSA. [11]

Ačkoliv je možné JWT také šifrovat pro zajištění utajení mezi stranami, v našem případě se zaměříme na podepsané tokeny, které umožňují ověřit integritu tvrzení obsažených v token, zatímco šifrované tokeny tyto informace skrývají před ostatními.

V našem případě je token používán jak k autentizaci, tak autorizaci. Toto je nejběžnější scénář pro použití JWT. Jakmile je uživatel přihlášen, každý následný požadavek obsahuje JWT, který umožňuje uživateli přístup k trasám, službám a prostředkům, které jsou s tímto tokenem povolené.

Využití JWT v naší aplikaci zabezpečuje, že každá komunikace mezi klientem a serverem je chráněna, což minimalizuje riziko neoprávněného přístupu a zaručuje, že uživatelé mají přístup pouze k datům, která mají oprávnění vidět. Tento způsob autentizace proto hraje klíčovou roli v zabezpečení celého systému.

2.2 Frontend

Při vývoji frontendové části naší aplikace byl použit JavaScript, což je jazyk, který dominuje webovému vývoji a je v podstatě nezbytný pro vytváření interaktivních a dynamických webových stránek. Vzhledem k jeho univerzálnímu přijetí a široké podpoře mezi vývojovými nástroji a prohlížeči je JavaScript přirozenou volbou pro vývoj moderního frontendu.

Z dostupných JavaScriptových frameworků jako React, Angular, Svelte či Vue byl využit framework Vue.js, framework i důvody jeho použití budou popsány níže v bodu 2.2.1.

Výběr JavaScriptu spolu s frameworkem Vue.js umožňuje vytvářet uživatelské rozhraní, které je nejen vizuálně přitažlivé a snadno použitelné, ale také rychlé a adaptabilní na různé potřeby uživatelů. Tyto technologie poskytují vynikající základ pro vývoj moderního a efektivního frontendu, který splňuje očekávání a požadavky současných webových aplikací.

2.2.1 Vue.js

Jak již bylo výše zmíněno, hlavní součástí celého frontendu tvoří framework Vue, a to především kvůli jeho jednoduchosti, modularitě a snadné rozšiřitelnosti, což umožňuje efektivní práci s komponentami a jednoduchou integraci s externími knihovnami a API. Tento framework podporuje deklarativní renderování a automatickou reaktivitu, která zajišťuje, že jakékoli změny ve stavech aplikace jsou okamžitě reflektovány v uživatelském rozhraní bez nutnosti manuálního zásahu. [12]

Vue také umožňuje snadné začlenění dalších modulů, jako je Bootstrap, který byl využit pro zjednodušení a zrychlení vývoje uživatelského rozhraní. Dalším důvodem pro výběr Vue byl záměr prohloubit dovednosti v práci s jedním z výše zmíněnými javascriptovými frameworky, a také dále využít tyto dovednosti a části kódu při vývoji mobilních aplikací pomocí Ionic frameworku pro iOS.

Vue umožňuje efektivní správu komponent prostřednictvím Single File Components (SFC), kde každá komponenta obsahuje svůj vlastní HTML, JavaScript a CSS v jediném souboru. Tento přístup podporuje udržitelnost a modulárnost kódu a zjednodušuje přehled o jednotlivých komponentách. Výběr Composition API pro implementaci logiky komponent byl motivován jeho schopností poskytnout větší flexibilitu a lepší organizaci kódu ve srovnání s Options API. Composition API umožňuje definovat logiku komponenty mnohem strukturovaněji a čitelněji, což je zásadní pro udržení čistoty kódu a efektivity ve větších projektech. [13][14]

Tato kombinace technologií a přístupů umožňuje, že Vue není jen framework pro jednoduché projekty, ale skvěle se hodí i pro složitější aplikace. S využitím Vue je možné rychle reagovat na požadavky trhu, iterovat a adaptovat funkce bez velkých kompromisů na výkon a udržitelnost aplikace, čímž se Vue stává vhodným frameworkem, který může růst s potřebami projektu a jeho uživatelů.

2.2.2 Bootstrap

Ve vývoji této aplikace pro frontend hraje klíčovou roli framework Bootstrap, který umožňuje vytvářet esteticky příjemné a intuitivní uživatelské rozhraní. Díky jeho široké škále předdefinovaných tříd je možné rychle a efektivně stylizovat elementy webové stránky od formulářů a tlačítek až po navigační prvky.

Bootstrap je oblíbený zejména pro svou responzivní architekturu. Vzhledem k rostoucí popularitě mobilních a tabletových zařízení je dnes nezbytné, aby se webové aplikace automaticky přizpůsobily různým velikostem obrazovek. Bootstrap toto zajišťuje, díky rozdělení celé stránky do dvanácti sloupců, které se potom mění dle velikosti obrazovky. Tato vlastnost Bootstrapu zajišťuje, že naše aplikace poskytne kvalitní uživatelskou zkušenost napříč všemi platformami. [15]

Jeho význam se dále projevuje ve snadné integraci s moderními JavaScriptovými frameworky, jako je Vue.js, který v našem projektu používáme. Díky kompatibilitě s Vue.js

jsme schopni rychle implementovat dynamické a interaktivní prvky, aniž bychom museli znovu vymýšlet základní estetické stavební bloky.

Dalším důvodem, proč byl Bootstrap vybrán, je jeho velká komunita a množství dostupných zdrojů a rozšíření. To umožňuje nejen snadněji řešit potenciální problémy, ale také rychle najít inspiraci a šablony pro různé uživatelské scénáře.

Nakonec Bootstrap přispívá k udržitelnosti kódu a jeho dlouhodobé udržovatelnosti, protože uplatňuje konzistentní styl a pravidla pro UI prvky. Toto vede k jednotnějšímu vzhledu aplikace a usnadňuje další rozvoj a úpravy, což je vzhledem k dynamice projektu a neustálé potřebě inovací neocenitelná vlastnost.

Z těchto důvodů je Bootstrap neodmyslitelnou součástí tohoto vývojářského stacku a významně přispívá k rychlosti, flexibilitě a uživatelské spokojenosti při tvorbě aplikace.

2.2.3 Axios

V rámci tohoto projektu hrají důležitou roli HTTP požadavky, které umožňují interakci mezi klientskou aplikací a serverem. Pro zpracování těchto požadavků je využíván Axios, což je klient založený na promise, vhodný jak pro serverovou stranu, tak pro prohlížeče (klienta). Jeho velkou výhodou je, že lze stejný kód použít jak na serveru, tak na klientské straně bez potřeby jakýchkoliv změn, v případě že na serverové straně je používán Node.js. [16]

Díky Axiosu je možné řešit HTTP/HTTPS požadavky s využitím moderních JavaScriptových konstrukcí, jako jsou asynchronní funkce a await, což výrazně zjednodušuje kód a činí ho čitelnějším. Navíc Axios podporuje ochranu proti cross-site request forgery (CSRF), což přispívá k bezpečnosti systému při práci s citlivými daty.[16]

Jedním z dalších důvodů, proč byl vybrán právě Axios, je jeho schopnost transformovat JSON data a nabízí široké možnosti konfigurace pro předvolby požadavků, což vede k efektivnější komunikaci s backendem a k větší flexibilitě při implementaci komplexních požadavků.

Axios poskytuje spolehlivé řešení pro HTTP/HTTPS komunikaci, která stojí na pevných základech moderních webových technologií a ve vytvořené aplikaci plní klíčovou roli v oblasti výměny dat a komunikace mezi klientskou a serverovou částí.

2.2.4 Vue-router

Dalším prvkem projektu je správa odkazů na klientské straně, pro kterou byl využit Vue Router. Tento oficiální router pro Vue.js je zásadní pro správné fungování jednostránkových aplikací (SPA), kde je nutné, aby URL adresa v prohlížeči korespondovala s obsahem, který uživatel vidí. Pokud uživatelé klikají na různé odkazy v rámci aplikace, URL se aktualizuje, ale stránka se nemusí znovu načítat ze serveru. [17]

Vue Router je plně integrován do systému komponent (stránek) Vue a umožňuje definovat trasy, které specifikují, které komponenty mají být zobrazeny v závislosti na cestě URL. To značně zvyšuje uživatelský komfort, jelikož interakce s aplikací jsou plynulejší a rychlejší, bez nutnosti neustálého přenačítání stránek.

Důležitou součástí Vue Routeru v této aplikaci je ochrana tras. Před každým přechodem na jinou stránku dojde k ověření, zda má uživatel v prohlížeči uložený platný JWT token. Pokud je token ověřený a je platný, uživatel může pokračovat v práci. V opačném případě je uživatel přesměrován na přihlašovací stránku. Toto opatření zajišťuje, že uživatelé jsou neustále autentizováni a mají přístup pouze k informacím, na které mají oprávnění.

Tento mechanismus nejenže zvyšuje bezpečnost aplikace tím, že omezuje přístup k citlivým datům, ale také vylepšuje uživatelskou zkušenost tím, že umožňuje uživatelům, aby byli přihlášení po celou dobu platnosti jejich JWT tokenu. Použití Vue Routeru tedy přináší efektivní a bezpečný způsob, jak spravovat navigaci a přístupy v moderní webové aplikaci.

2.2.5 Pinia

Další technologií pro vývoj aplikace je Pinia Store jako klíčová knihovna pro správu stavu v aplikacích Vue, která byla původně navržena jako experiment s cílem přepracovat koncept store v Vue s využitím Composition API kolem listopadu 2019. Od té doby zůstaly základní principy stejné, ale Pinia je nyní kompatibilní jak s Vue 2, tak s Vue 3 a nevyžaduje použití Composition API, což z ní dělá flexibilní řešení pro různé verze Vue. [18]

Pinia umožňuje sdílet stav mezi komponentami nebo stránkami aplikace, což je zásadní pro správu dat a interakcí v aplikaci. V případě, že potřebujeme upravit nějakou funkci, která je využívána na více místech, změna se provádí pouze v jednom místě, což značně zjednodušuje údržbu kódu.

Integrace s Vue Devtools podporuje sledování akcí a mutací a umožňuje ladění aplikace metodou time travel. Funkce Hot Module Replacement usnadňuje vývoj tím, že umožňuje

úpravy store bez nutnosti obnovení stránky. Dále Pinia podporuje Server Side Rendering, což zvyšuje bezpečnost a výkon aplikací, které vyžadují serverové předrenderování. [18]

Tyto vlastnosti z Pinia činí dobrou volbu pro správu stavu ve Vue aplikacích, kde je kladen důraz na efektivitu, bezpečnost a snadnou údržbu kódu.

2.3 Bezpečnostní mechanismy

Bezpečnost je v digitálním světě nezbytností a díky rostoucí závislosti na technologických systémech se bezpečnost stává neodmyslitelnou součástí vývoje softwarových aplikací. Tato kapitola je zaměřena na základní stavební kameny, které formují obranu vytvořeného systému proti potenciálním bezpečnostním hrozbám.

2.3.1 HTTPS

Hypertext Transfer Protocol Secure (HTTPS) je rozšířením základního komunikačního protokolu HTTP, který je využíván pro přenos webových stránek. Přidává důležitou vrstvu zabezpečení pomocí protokolů SSL/TLS, které šifrují data odesílaná mezi uživatelovým prohlížečem a webovým serverem. Tato šifrovací vrstva zajišťuje, že všechny informace, včetně citlivých dat jako jsou hesla a platební informace, jsou chráněny před možným odposlechem třetími stranami.

Význam HTTPS roste, a to hlavně z důvodu, že stávající kybernetické hrozby jsou stále sofistikovanější a útočníci vynalézavější. S využitím HTTPS se veškerá komunikace stává nečitelnou pro každého, kdo se pokusí data zachytit. To je nejen zásadní pro ochranu uživatelů, ale stává se i jedním z kritérií hodnocení důvěryhodnosti a profesionality webové služby.

Implementace HTTPS na serveru zahrnuje získání SSL/TLS certifikátu od certifikační autority, například Let’s Encrypt, což potvrzuje identitu serveru a umožňuje šifrovanou komunikaci. V dnešní době je HTTPS standardem pro webové aplikace, a proto je klíčovým prvkem v návrhu vytvářeného systému, což zajišťuje bezpečný a soukromý přenos dat.

Pro naši aplikaci byl v cloudu implementován webový server Caddy, který automaticky implementuje certifikát Let’s Encrypt a zajišťuje tak zabezpečené spojení se serverem.

2.3.2 Autentizace

Autentizace je klíčovým aspektem zabezpečení informačních systémů. Jejím hlavním cílem je ověřit identitu uživatele, který se snaží získat přístup k systému, obvykle pomocí kombinace uživatelského jména a hesla. Tento tradiční přístup k autentizaci je preferován pro jeho jednoduchost a účinnost, přičemž se však musí věnovat zvýšená pozornost bezpečnosti hesel.

Síla hesla je rozhodujícím faktorem v ochraně před neoprávněným přístupem. Proto v našem systému požadujeme hesla, která mají minimálně osm znaků a kombinují písmena, číslice a speciální znaky, což významně ztíží jejich prolomení prostřednictvím útoků hrubou silou nebo slovníkovými útoky.

Pro zajištění ochrany hesel v tomto systému jsou použity sofistikované hashovací metody z knihovny Passlib. Hesla jsou transformována do hashované formy před uložením do databáze, což znamená, že původní hesla nejsou nikdy uložena nebo přenášena v čitelné podobě. V našem backendu jsou tyto hashovací funkce implementovány v souboru *auth.py*, kde jsou využity bezpečné algoritmy poskytované knihovnou Passlib, čímž je zajištěno, že i v případě nechtěného úniku dat, hashovaná hesla nedají útočníkovi možnost přístupu k účtům.

2.3.3 Autorizace

Autorizace je proces, při kterém se ověřuje, zda má daný uživatel oprávnění k provedení určité operace nebo přístupu k určitým datům. Na rozdíl od autentizace, která ověřuje identitu uživatele, autorizace se zaměřuje na jeho práva a oprávnění v rámci systému. V tomto systému jsou rozlišovány tři základní role: uživatel, manažer a administrátor, přičemž každá z nich má specifickou sadu oprávnění, která odráží jejich rozdílné role a odpovědnosti.

Administrátoři mají nejvyšší úroveň přístupových práv, což jim umožňuje přístup ke všem datům v systému bez jakýchkoliv omezení. Jsou schopni spravovat uživatelské účty, prohlížet všechny logy a provádět další administrativní úkony potřebné pro udržení chodu systému.

Manažeři mají rozšířený přístup k logům, což jim umožňuje monitorovat a spravovat všechny logy v systému, avšak už nemají přístup ke všem uživatelským datům. Typickým uživatelem typu manažer je účetní, která spravuje účetní doklady.

Běžní uživatelé mají přístup pouze ke svým vlastním logům a uživatelským datům. Tato omezení jsou navržena tak, aby chránila informace před neoprávněným přístupem a zajišťovala, že uživatelé mohou bez obav spravovat svá data.

Tato práva a oprávnění jsou pečlivě nastavena a vynucována na úrovni každého endpointu v našem backendu v souboru „*main.py*“. Takovýto přístup zaručuje, že pokud dojde k pokusu o manipulaci dat na straně klienta, nedojde k neoprávněnému úniku informací. Tímto způsobem dosahujeme vysoké úrovně bezpečnosti, když každý uživatel může přistupovat pouze k informacím, ke kterým má oprávnění, a nemůže být neúmyslně nebo úmyslně zneužit.

II. PRAKTICKÁ ČÁST

3 NÁVRH SYSTÉMU

Při návrhu systému pro digitální evidenci dokladů bylo nutné zvážit řadu důležitých aspektů, aby bylo možné vytvořit robustní, efektivní a uživatelsky příjemné řešení.

Prvním krokem bylo definování a specifikace funkčních a nefunkčních požadavků systému. Tento proces zahrnoval identifikaci hlavních funkcionalit, které systém musí nabídnout, stejně jako požadavky na výkon, bezpečnost a uživatelskou přívětivost.

Dalším zásadním krokem byl návrh databáze. Byla zvolena taková struktura databáze, která umožňuje efektivní ukládání a vyhledávání informací o dokladech, zahrnující metadata jako datum vytvoření, kategorii dokladu, částku a příslušného uživatele. Důraz byl kladen na optimalizaci pro snadné vyhledávání, škálovatelnost a ukládání pouze nezbytně nutných dat.

Kromě toho byly vytvořeny základní modely případů užití, které pak pomohly lépe pochopit interakce uživatelů se systémem a zajistit, aby design reflektoval reálné pracovní procesy. Tyto modely sloužily jako základ pro vývoj uživatelského rozhraní a určení priorit jednotlivých funkcí systému.

Další částí návrhu systému bylo také určení cílových uživatelů systému. Byly identifikován\ různé typy uživatelů, jako účetní, administrátor a zaměstnanců (piloti), kteří budou systém využívat pro zadávání a správu dokladů. Pro každou z těchto skupin bylo navrženo uživatelské rozhraní a funkcionalita tak, aby co nejlépe vyhovovali jejich potřebám a pracovním procesům.

Výsledkem návrhového procesu je systém, který je jednoduchý, snadno použitelný a schopný efektivně zpracovávat a archivovat doklady v digitální formě. Díky tomuto řešení je možné výrazně zefektivnit pracovní procesy související s evidencí dokladů a poskytnout uživatelům rychlý a bezpečný přístup k potřebným informacím.

3.1 Funkční a nefunkční požadavky

3.1.1 Funkční požadavky

Funkční požadavky určují konkrétní operace nebo funkce, které musí systém vykonávat. Pro tuto aplikaci byly definovány následující klíčové funkční požadavky:

- **Přihlášení uživatele do systému:** Uživatelé musí být schopni přihlásit se do systému, pokud již nejsou přihlášeni.

- **Zadávání dokladů:** Uživatelé musí být schopni snadno zadávat nové doklady do systému, včetně možnosti nahrávat digitální kopie těchto dokladů.
- **Správa a archivace dokladů:** Systém musí umožňovat efektivní správu a vyhledávání uložených dokladů, včetně možnosti jejich kategorizace a filtrování.
- **Účetní přístup pro správu dokladů:** Uživatelé by měli mít možnost zapracovat doklady do svých účetních systémů a vytvořit tak fakturaci pro konečného zákazníka.
- **Ověřování a autorizace uživatelů**: Systém musí zahrnovat robustní mechanismy pro ověřování uživatelů a správu jejich oprávnění k přístupu k různým částem aplikace.
- **Tvorba a správa uživatelů**: Uživatelé s oprávněním administrátora by měli mít přístup ke správě uživatelů, to včetně tvorby nových uživatelů, mazání a úprav již existujících uživatelů.

3.1.2 Nefunkční požadavky

Nefunkční požadavky se zaměřují na kvalitu a omezení systému, včetně výkonnostních specifikací, bezpečnostních standardů a uživatelského zážitku. Mezi klíčové nefunkční požadavky naší aplikace patří:

- **Výkon a škálovatelnost:** Systém musí být schopen zpracovávat doklady ve formátech převážně PDF o velikosti jednotek až nižších desítek megabajtů a musí být navržen tak, aby byl snadno škálovatelný podle rostoucích potřeb uživatelů.
- **Bezpečnost a ochrana dat:** Veškerá uživatelská data, včetně digitálních kopii dokladů, musí být chráněna před neoprávněným přístupem a musí být zajištěna jejich důvěrnost a integrita.
- **Uživatelská přívětivost:** Rozhraní aplikace musí být intuitivní a snadno použitelné pro všechny typy uživatelů, aby bylo možné minimalizovat potřebu školení a podpory.
- **Kompatibilita a integrace:** Systém by měl být kompatibilní s běžnými operačními systémy a zařízeními a umožňovat snadnou integraci s externími systémy nebo aplikacemi.

3.2 Uživatelé a jejich role

V našem systému rozlišujeme tři základní uživatelské role, které odrážejí hierarchii a pracovní rozdělení v rámci společnosti: uživatele, manažery a administrátory. Každá role je vybavena specifickou sadou oprávnění, která jsou navržena tak, aby splňovala rozdílné potřeby a odpovědnosti jednotlivých uživatelů.

Uživatel: Tato základní úroveň přístupu je přidělena všem, kteří se mohou přihlásit do aplikace. Uživatelé mají možnost vytvářet nové záznamy, tzv. logy, a také vytvářet jednotlivé tripy, do kterých tyto logy spadají (každý log musí být součástí nějakého tripu). Mají přístup pouze k informacím, které sami vytvořili, což zajišťuje důvěrnost a správu vlastních dat. Dále mohou změnit své osobní heslo, čímž se podporuje bezpečnostní povědomí a samostatnost uživatelů.

Manažer: Manažeři mají rozšířené pravomoci, které jim umožňují dohled nad logy vytvořenými všemi uživateli napříč celou společností. Tato role je typicky spojená s účetní nebo vedoucím jednotlivé letky.

Administrátor: Jako nejvyšší úroveň uživatelských práv, administrátoři mají možnost nejen vytvářet a spravovat uživatelské účty, ale i přidávat či upravovat uživatelská práva, a to až po úroveň smazání účtu. Administrátoři mohou také přidávat nebo modifikovat nové letouny ve společnosti, což je důležité pro správné přiřazení a evidenci logů. Právě tato role zastřešuje a udržuje celý systém v chodu vhodnou správou dat v systému.

Toto rozdělení je navrženo tak, aby odráželo reálné potřeby společnosti, a zároveň chránilo citlivá data. V našem systému je každá role důkladně promyšlena, aby poskytla nezbytnou flexibilitu a zabezpečení, a zároveň udržovala přehlednost a kontrolu nad důležitými operacemi.

3.3 Struktura databáze

Struktura databáze byla navržena s využitím nástroje ERDPlus, což je online nástroj pro návrh entitně-relačních diagramů, který umožňuje efektivně vizualizovat vztahy mezi tabulkami a entitami v databázi. Při návrhu byl kladen důraz na minimalismus a efektivitu, aby jednotlivé tabulky obsahovaly pouze nezbytné informace pro fungování systému. Tento přístup zajišťuje, že databázový systém zůstává lehký a jednoduchý na správu, přičemž zároveň poskytuje dostatečný základ pro ukládání všech klíčových dat. [8]

Důraz byl kladen na to, aby tabulky nebyly přeplněné nadbytečnými daty, což přispívá k rychlejšímu zpracování dotazů a celkovému výkonu aplikace. Navzdory své jednoduchosti je databázový model navržen tak, aby byl snadno rozšířitelný. To umožňuje budoucí rozvoj a integraci nových funkcí bez potřeby komplexních přestaveb nebo zásahů do stávající struktury. Tento přístup zajišťuje, že aplikace může růst spolu s požadavky uživatelů a technologickými změnami, což je klíčové pro udržení konkurenceschopnosti a adaptability na měnícím se trhu.

Zde je uveden popis jednotlivých tabulek databáze a jejich vzájemné vztahy, které podporují škálovatelnost a rozšiřitelnost systému.

Tabulka Company

Tato tabulka obsahuje základní informace o firmách, které využívají aplikaci. Každá firma je identifikována unikátním *„company_id“*. Dále tabulka uchovává název firmy, zemi a město působení, daňové identifikační číslo (DIČ), adresu a kontaktní email. Tato tabulka slouží jako základ pro přiřazení uživatelů a letadel k firmě.

Tabulka Airplanes

V tabulce Airplanes jsou evidovány všechny letouny, které jsou spravovány prostřednictvím aplikace. Každý letoun má přiděleno unikátní *„airplane_id“*, typ letounu, registrační číslo a je svázán s konkrétní firmou pomocí *„company_id“*. Tato tabulka umožňuje sledovat letouny v kontextu jednotlivých firem.

Tabulka Category

Obsahuje kategorie, které mohou být přiřazeny k různým logům (například: letištní poplatky, palivo, catering a další). Každá kategorie má své *„category_id“* a název. Každý název kategorie je unikátní. Inicializace této tabulky probíhá při instalaci systému a data v ní je možné měnit pouze přímým přístupem do databáze, kategorie jsou tak stále flexibilní, ale není možná přímá změna uživatelem, která by mohla být nežádoucí.

Tabulka Method

Tato tabulka definuje metody, které mohou být použity pro záznam logů, například různé způsoby platby (hotovost, faktura či platba kartou). Každá metoda je identifikována pomocí *„method_id“*. Platí zde stejný princip změny jako u tabulky Category.

Tabulka Trip

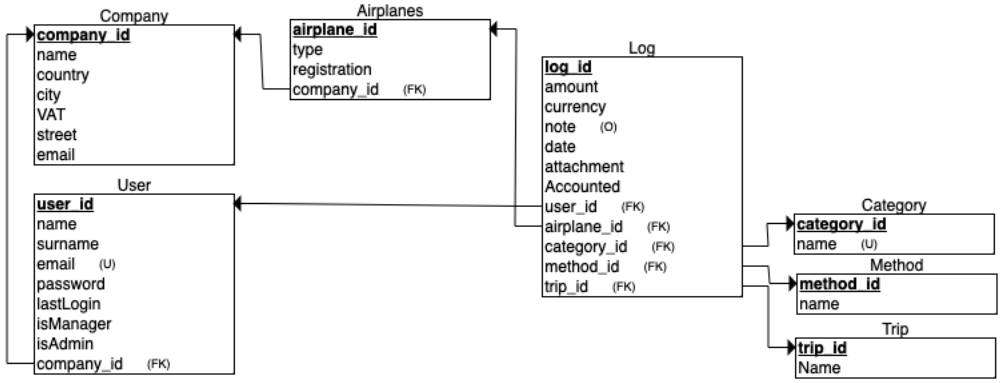
Slouží pro evidenci jednotlivých cest nebo úkolů, které mohou být přiřazeny k logům. Každý trip má své unikátní *„trip_id“* a název. Zde je prostor pro přiřazení jednotlivých stránek technického deníku letadla k jednotlivým tripům, které se zpravidla shodují.

Tabulka User

Uchovává informace o uživatelích aplikace. Zahrnuje *„user_id“*, jméno, příjmení, email, heslo, datum posledního přihlášení, informace zda je uživatel manažerem nebo administrátorem a *„company_id“* pro spojení s konkrétní firmou.

Tabulka Log

Tato tabulka je klíčová pro evidenci všech transakcí nebo záznamů, které uživatelé aplikace vytvářejí. Obsahuje údaje jako množství, měnu, poznámky, datum, přílohy a zda byly účetně zaúčtovány. Logy jsou svázány s uživatelem, letounem, kategorií, metodou a tripy prostřednictvím cizích klíčů.



Obrázek 1. Schéma databáze

3.4 API endpointy

V rámci návrhu systému bylo zásadním úkolem definovat a implementovat API endpointy, které zajišťují komunikaci mezi frontendem a backendem aplikace. Celkově bylo navrženo dvacet pět API endpointů, které pokrývají širokou škálu operací potřebných pro správné fungování databáze a uživatelské interakce.

Struktura Endpointů

Endpointy byly navrženy s ohledem na potřeby jednotlivých tabulek v databázi. Pro tabulky, kde je to relevantní, byly implementovány všechny základní HTTP metody – GET, POST, PUT a DELETE. Toto umožňuje plnou manipulaci s daty (vytváření, čtení, aktualizace, mazání). Například, pro tabulku Logs, existují endpointy, které umožňují přidávat nové záznamy, získávat detaily o existujících záznamech, aktualizovat je nebo je odstraňovat.

Některé tabulky, jako například Company, nevyžadují plnou sadu HTTP operací kvůli specifické povaze dat, která obsahují. Pro tyto tabulky nebyly implementovány endpointy pro POST a DELETE, aby se předešlo nechtěnému přidávání nebo mazání kritických informací o společnostech.

Přístupová práva

Každý endpoint má specifická přístupová práva, která určují, které uživatelské role mají k daným datům přístup. To zajišťuje, že uživatelé mohou pracovat pouze s daty, která jsou pro jejich roli relevantní nebo jsou jejich vlastní. Například, zatímco administrátoři mají přístup ke všem operacím a datům, manažeři mohou mít omezený přístup jen k některým specifickým datům a uživatelé zpravidla pouze ke svým datům.

Testování

Pro každý endpoint byly implementovány testy, které ověřují jak správné fungování (pozitivní testy), tak reakci systému na neplatné nebo škodlivé požadavky (negativní testy). Toto testování je nezbytné pro zajištění bezpečnosti systému, kde je zvláště důležité ověřit chování systému v reakci na potenciálně škodlivé nebo nevalidní vstupy. Tyto testy jsou určeny ke spuštění vždy, když dojde ke změnám v aplikaci a mají za úkol pokrýt jen testování základních funkcí API endpointů.

Dokumentace

Veškeré API endpointy jsou zdokumentovány, včetně informací o tom, jaké parametry očekávají, jaký je očekávaný výstup a jaké chyby mohou během jejich volání nastat. Dokumentace je přístupná prostřednictvím swagger rozhraní dostupného na *„/docs“* a reDoc dostupného na *„/redoc“*, což usnadňuje orientaci ve funkcionalitách API jak vývojářům, tak uživatelům.

Tato struktura a organizace API endpointů jsou klíčové pro efektivní a bezpečnou operaci celého systému a zajišťují, že aplikace bude schopna růst a adaptovat se na budoucí požadavky uživatelů a implementaci nových funkcionalit.

API Endpoints								
Table	URL	Method	All Users	Logged In	Manager	Admin	Test P	Test N
Companies	/company	GET	X	X	X	X	X	0
Companies	/company	PUT	0	0	0	X	X	X
User	/users	GET	0	0	0	X	X	X
User	/users/{my_id}	GET	0	X	0	X	X	0
User	/users	POST	0	0	0	X	X	X
User	/users/{user_id}	PUT	0	0	0	X	X	0
User	/users/{user_id}	DELETE	0	0	0	X	X	0
Airplanes	/airplanes	GET	X	X	X	X	X	0
Airplanes	/airplanes/{a_id}	GET	X	X	X	X	X	0
Airplanes	/airplanes	POST	0	0	0	X	X	0
Airplanes	/airplanes/{a_id}	PUT	0	0	0	X	X	0
Airplanes	/airplanes/{a_id}	DELETE	0	0	0	X	X	0
Log	/logs	GET	0	X (self)	X	X	X	X
Log	/logs/{log_id}	GET	0	X (self)	X	X	X	X
Log	/logs	POST	0	X	0	X	X	X
Log	/logs/{log_id}	PUT	0	X	0	X	X	X
Log	/logs/{log_id}	DELETE	0	X	0	X	X	X
Trip	/trips	GET	0	X (self)	X	X	X	0
Trip	/trips/{trip_id}	GET	0	X (self)	X	X	X	0
Trip	/trips	POST	0	X	0	X	X	0
Trip	/trips/{trip_id}	PUT	0	X (self)	0	X	X	0
Trip	/trips/{trip_id}	DELETE	0	X (self)	0	X	X	0
Categories	/categories	GET	X	X	X	X	X	0
User	/change-password	POST	0	X (self)	X	X	0	0
Methods	/methods	GET	X	X	X	X	X	0

Obrázek 2. Přehled API endpointů

3.5 Případy užití

V sekci případů užití je popsáno, jak jednotlivé funkční požadavky systému přechází do konkrétních scénářů použití. Tyto případy užití poskytují jasný pohled na to, jak uživatelé interagují se systémem a jaké akce mohou provádět. Každý případ užití odpovídá na základní funkční požadavky, které byly dříve definovány. Pro ukázku jsou zde představeny dva scénáře případů užití podrobněji a to Přihlášení uživatele do systému a Zadávání dokladů.

Tabulka 1. Přihlášení uživatele do systému (hlavní tok)

Název: **Přihlášení uživatele do systému**
ID: **UC001**
Charakteristika: Zachycení případu užití pro přihlášení uživatele
Primární aktér: Nepřihlášený uživatel

<table>
<tr><td colspan="3">Vedlejší aktéři:

Nejsou</td></tr>
<tr><td colspan="3">Vstupní podmínky:

1. Uživatel aktuálně není přihlášen
2. Uživatel má vytvořený účet v databázi
3. Uživatel není přihlášen</td></tr>
<tr><td colspan="3">Výstupní podmínky:

1. Systém přihlásí uživatele
2. Systém vytvoří a uloží autentizační token
3. Systém přesměruje uživatele na domovskou obrazovku</td></tr>
<tr><td colspan="3">Hlavní scénář:</td></tr>
<tr><td>Krok</td><td>Aktér/Systém</td><td>Popis</td></tr>
<tr><td>1</td><td>A</td><td>Tento případ začíná na úvodní stránce, kde nepřihlášený uživatel vyplní uživatelské jméno (emailová adresa) a heslo.</td></tr>
<tr><td>2</td><td>S</td><td>Systém ověří zadané údaje

Alternate: UC001a – Přihlášení se nezdařilo</td></tr>
<tr><td>3</td><td>S</td><td>Systém přesměruje uživatele na domovskou stránku.</td></tr>
<tr><td>4</td><td>A</td><td>Případ užití končí.</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td colspan="3">Alternativní scénáře:

UC001a – Přihlášení uživatele se nezdařilo.</td></tr>
</table>

Tabulka 2. Přihlášení uživatele se nezdařilo (alternativní scénář)

Název – Alternativní scénář: **Přihlášení uživatele se nezdařilo**
ID: **UC001a**
Charakteristika: Zachycení alternativního toku případu užití, kdy se přihlášení uživatele nezdařilo.
Alternativní scénář:

Krok	Aktér/Systém	Popis
1	S	Ověření přihlašovacích údajů se nezdařilo.
2	S	Systém zobrazí varovnou hlášku o tom, že přihlášení do systému selhalo a zůstane na přihlašovací obrazovce.
3	A	Nepřihlášený uživatel může znovu zadat přihlašovací údaje.

Tabulka 3. Zadávání dokladů (hlavní tok)

Název: **Zadávání dokladů**		
ID: **UC002**		
Charakteristika: Zachycení případu užití pro zadání nového dokladu do systému		
Primární aktér: Přihlášený uživatel		
Vedlejší aktéři: Nejsou		
Vstupní podmínky: 1. Uživatel je aktuálně přihlášen. 2. Uživatel je oprávněn provádět zadávání dokladů do systému.		
Výstupní podmínky: 1. Systém uloží nový doklad 2. Nový doklad se zobrazí v seznamu uložených dokladů		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	A	Tento případ začíná, když přihlášený uživatel v menu zvolí položku Logs.

2	S	Systém přesměruje uživatele na stránku, kde je možno spravovat záznamy.
3	A	Uživatel zvolí možnost Add log.
4	S	Systém zobrazí modální okno s formulářem pro vložení záznamu.
5	A	Uživatel vyplní formulář a zvolí Add log. Alternate: UC002a – Uživatel zavře okno.
6	S	Systém ověří vstupní data. Alternate: UC002b – Vstupní data nejsou validní
7	S	Systém zobrazí nový seznam záznamů, který obsahuje nově zadaná data.
8	A	Případ užití končí.
Alternativní scénáře: UC002a – Uživatel zavře okno UC002b – Vstupní data nejsou validní		

Tabulka 4. Uživatel zavře okno (alternativní scénář)

Název – Alternativní scénář: **Uživatel zavře okno**		
ID: **UC002a**		
Charakteristika: Zachycení alternativního toku případu užití, kdy uživatel před odesláním formuláře předčasně ukončí zadávání dat.		
Alternativní scénář:		
Krok	Aktér/Systém	Popis

1	A	Uživatel zvolí možnost pro zavření formulářového okna, bez odeslání dat.
2	S	Systém přesměruje uživatele zpět na stránku se záznamy.
3	S	Systém zachová již zadaná data do formuláře ve stavu pro případné znovu otevření formuláře.
4	A	Uživatel může znovu otevřít formulář a pokračovat (již zadaná data zůstanou v paměti).

Tabulka 5. Vstupní data nejsou validní (alternativní scénář)

Název – Alternativní scénář: **Vstupní data nejsou validní**		
ID: **UC002b**		
Charakteristika: Zachycení alternativního toku případu užití, kdy uživatelem vložená data nebyla validní.		
Alternativní scénář:		
Krok	Aktér/Systém	Popis
1	S	Ověření zadaných údajů se nezdařilo.
2	S	Systém zobrazí varovnou hlášku o tom, že zadávání dat nebylo úspěšné a formulář pro zadání dat zůstane otevřený.
3	A	Uživatel opakovaně vyplní požadovaná data.

3.6 Návrh uživatelského rozhraní

Uživatelské rozhraní (UI) tohoto systému bylo navrženo s důrazem na uživatelskou přívětivost a efektivitu práce. Pro návrh UI byl zvolen nástroj Figma, ve kterém byl kompletně navržen jeho design. V tomto nástroji byla vytvořena uživatelská rozhraní jak pro desktop, tak i pro mobilní zařízení. [19]

Postup vývoje:

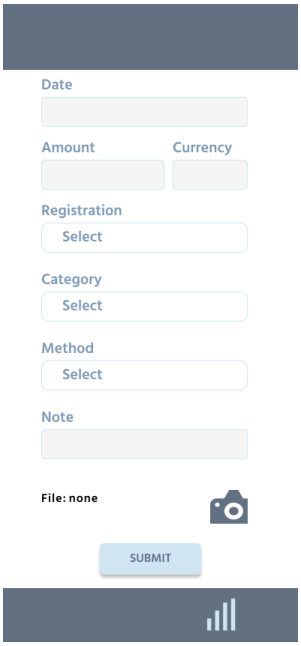
Definice barevného schématu a stylů: V prvním kroku bylo důležité stanovit základní vizuální jazyk aplikace, což zahrnovalo výběr barev, typů písma a dalších vizuálních prvků tak, aby byly konzistentní v celé aplikaci a vytvořily jednotný vizuální styl.

Komponenty a jejich opětovné využití: Následně byly ve Figma nástroji vytvořeny komponenty, jako jsou tlačítka, formuláře, navigační prvky a další, které byly v aplikaci použity. Tyto komponenty byly navrženy tak, aby byly snadno použitelné a utvářely jednotný vizuální styl aplikace.

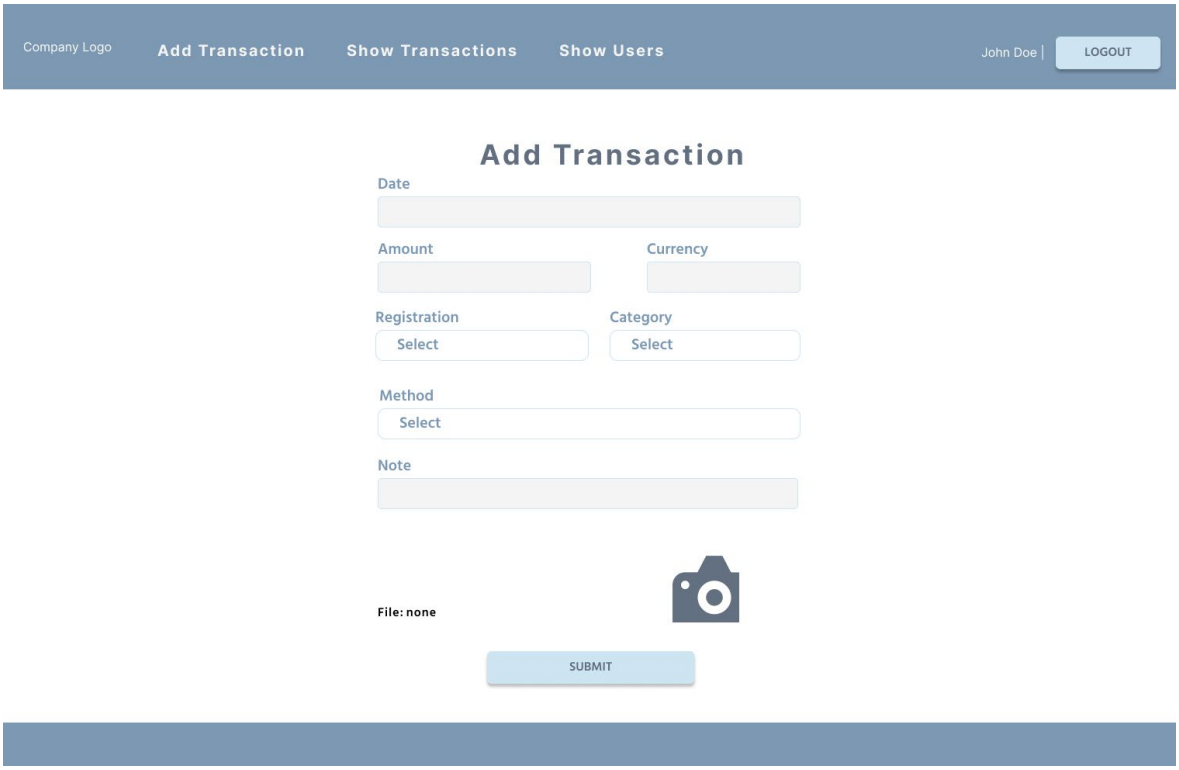
Sestavení stránek: Po vytvoření základních komponent byly sestavovány jednotlivé stránky aplikace. Stránky byly navrženy jak pro desktopovou tak pro mobilní verzi aplikace. Každá stránka byla navržena tak, aby vyhovovala svému účelu, přehledně zobrazovala požadované informace a zároveň poskytovala intuitivní a dobrý uživatelský zážitek.

Prototypování a interaktivita: Díky Figma bylo možné vytvořit interaktivní klikací prototyp, který umožňuje simulovat používání aplikace ještě před jejím skutečným kódováním.

Už od začátku bylo zřejmé, že aplikace musí být plně funkční na různých zařízeních, od stolních počítačů až po mobilní telefony. Proto byl návrh vytvořen s důrazem na responzivní design.



Obrázek 3. Ukázka návrhu mobilní verze formuláře (vytvořeno v: Figma)



Obrázek 4. Ukázka desktopového návrhu formuláře (vytvořeno v: Figma)

4 BACKEND

V této části jsou uvedeny základní informace o implementaci backendu tohoto systému, včetně příkladů kódu, který byl v aplikaci použit. Tato kapitola nabízí přehled technologických rozhodnutí, procesů vývoje a konkrétních implementačních kroků, které byly podniknuty k zajištění backendu, založeného na API.

4.1 Implementace backendu

Zahájení projektu

Na počátku vývoje byl založen repozitář na GitHubu, který slouží jako centrální úložiště pro veškerý kód aplikace, zajišťuje verzování a umožňuje spolupráci mezi vývojáři.

K izolaci závislostí a jejich správě bylo vytvořeno virtuální prostředí pomocí příkazu *„python3 -m venv env“*. Toto prostředí umožňuje lokalizovat knihovny potřebné pro aplikaci a její spouštění, což zvyšuje kontrolu nad vývojovým prostředím a minimalizuje riziko konfliktů knihoven.

Dále byly instalovány knihovny FastAPI a Uvicorn jakožto základní nástroje pro vytváření a spouštění asynchronních API. SQLAlchemy a PyMySQL byly instalovány pro práci s databází MySQL, kde SQLAlchemy slouží jako ORM nástroj pro mapování databázových objektů do Python kódů a PyMySQL jako databázový konektor.

Modely databáze

V souboru *„models.py“* byly definovány třídy, které odpovídají struktuře databázových tabulek. Tento návrh využívá výše zmíněného konceptu ORM k abstrahování a manipulaci s databázovými daty v objektově orientovaném stylu. Každá třída v *„models.py“* reprezentuje jednu tabulku v databázi a je navržena tak, aby odrážela vztahy a omezení definované v ERD diagramu, což usnadňuje správu databáze.

Autentizace a správa uživatelů

Autentizační mechanismy byly implementovány v souboru *„auth.py“*, kde byly vytvořeny funkce pro registraci, přihlášení a ověření uživatelů. Tyto funkce využívají technologii JWT pro generování a ověřování tokenů, což zabezpečuje, že uživatelé jsou autentizováni a autorizováni k provádění operací v rámci aplikace. Opatření proti běžným bezpečnostním hrozbám, jako je SQL injection, byla zavedena pomocí bezpečné manipulace s databází a

validace vstupních dat skrze Pydantic, který zajišťuje, že všechna data jsou správně formátovaná a validní před jejich zpracováním nebo uložením do databáze.

Hlavní soubor aplikace – *„main.py“*

Soubor *„main.py“* slouží jako hlavní vstupní bod pro backendovou část aplikace. V tomto souboru jsou definovány všechny API endpointy, které aplikace nabízí. Využívá jednoduchou a strukturovanou definici každého endpointu, včetně HTTP metod, které podporují (GET, POST, PUT, DELETE) a očekávaných vstupních a výstupních dat.

Endpointy v *„main.py“* jsou navrženy s ohledem na různé funkcionální požadavky aplikace, od správy uživatelů, přes manipulaci s daty o letadlech, až po evidenci a správu logů a dokladů. Každý endpoint má přiřazené autorizační požadavky, které určují, jaké oprávnění je nutné pro přístup a manipulaci s daty. Tento přístup zvyšuje bezpečnost aplikace tím, že omezuje přístup k citlivým operacím pouze pro autorizované uživatele.

Endpointy jsou navrženy tak, aby byly snadno testovatelné a rozšiřitelné, což umožňuje přidávání nových funkcí a úpravy stávajících funkcí s minimálním dopadem na stávající systém. Kód je strukturován do logických bloků, což usnadňuje údržbu a rozvoj aplikace.

Testování API endpointů

Další částí tvorby systému bylo testování jednotlivých API endpointů, aby bylo zajištěno, že backend naší aplikace funguje správně a je bezpečný. Pro tento účel byl vytvořen soubor *„test_endpoints.py“*, ve kterém jsou definovány automatizované testy pro všechny endpointy. Testy pokrývají široké spektrum scénářů od úspěšného vytváření a získávání záznamů po ošetření chyb, jako jsou neoprávněné přístupy nebo manipulace s nevalidními daty, nepokrývají ovšem všechny případy a slouží hlavně jako kontrola, zda během změn v kódu nedošlo k narušení některých základních funkcionalit programu.

V tabulce, která zobrazuje seznam endpointů (obrázek č. 2), je naznačeno, které endpointy mají testy, který mají pozitivní scénáře, jako je úspěšné vytvoření a načtení záznamů, stejně jako negativní scénáře, kde se ověřuje zacházení s neoprávněnými požadavky nebo s nevalidními daty. Cílem těchto testů je ověřit správnost, bezpečnost a spolehlivost funkcionalit souvisejících se správou logikou v aplikaci.

Tento přístup k testování umožňuje identifikovat a opravit potenciální problémy dříve, než bude aplikace nasazena do produkčního prostředí, a tím zvyšuje kvalitu celého systému.

Testy jsou implementovány s využitím knihovny pytest a FastAPI testovacího klienta, což zajišťuje, že jsou naše testy efektivní a snadno rozšiřitelné pro budoucí vývoj aplikace.

```
def test_read_log(client, auth_token):
    headers =  "Authorization": f"Bearer {auth_token}"}
    response ₹ client.get("/logs/1", headers=headers)
    assert response.status_code == 200
    assert "amount" in response.json()
```

Obrázek 5. Test logu s ID.

4.2 Konfigurace databáze

Konfigurace databáze pro tento systém začíná inicializací prázdných tabulek, což je první krok pro zajištění, že má aplikace při spuštění k dispozici všechny potřebné databázové struktury. Tento proces je řízen skriptem *„db_init.py“*, který obsahuje všechny potřebné instrukce pro vytvoření tabulek definovaných v *„models.py“*. Tento skript také naplňuje některé tabulky základními daty, které jsou potřebné pro běh aplikace, jako jsou metody platby, kategorie výdajů a údaje o společnosti.

Inicializace zahrnuje vytváření relací mezi tabulkami pomocí cizích klíčů, což zajišťuje integritu a konzistenci dat v rámci databáze a také zajišťuje, že jsou tyto vztahy vždy správně udržovány.

4.3 Autentizace a autorizace

Ověřování uživatelů

V této aplikaci je ověřování uživatelů základním faktorem pro zajištění bezpečného a efektivního provozu. Implementace autentizace je založena na použití JSON Web Tokenů (JWT), které poskytují spolehlivý způsob ověřování uživatelských práv a ověřování identity uživatelů.

Když uživatel provede přihlášení, systém ověří jeho přihlašovací údaje proti údajům uloženým v databázi. Po úspěšném ověření systém vygeneruje token, který obsahuje identifikátor uživatele a další relevantní informace. Tento token je následně odeslán klientovi, který ho používá pro autentizaci následujících požadavků odesílaných na server. JWT je konfigurován tak, aby vypršel po definované době, což vyžaduje periodické obnovování tokenů pro pokračující interakce s aplikací. V tomto případě byla expirace nastavena na 300 minut.

Ověřování úrovně oprávnění

V tomto systému je přístup k různým funkcím a datům založený na uživatelských rolích, úrovních oprávnění. Každý endpoint, který to vyžaduje, je konfigurován tak, aby ověřoval uživatelskou roli před zpracováním jakéhokoli požadavku. To zajišťuje, že uživatelé mohou přistupovat pouze k operacím, pro které mají odpovídající oprávnění.

Pro implementaci tohoto zabezpečení používá každý API endpoint middleware pro autentizaci, který rozpozná JWT token poslaný uživatelem. Token je validován a zkontrolován proti pravidlům nastaveným pro každý endpoint.

Při přístupu k endpointu systém zkontroluje, zda role uživatele odpovídá požadované roli pro daný endpoint. Pokud uživatel nemá potřebné oprávnění, požadavek je zamítnut s odpovídajícím HTTP stavovým kódem *„401 Unauthorized“*.

Endpointy určené pro administrátory jsou přístupné pouze uživatelům s rolí administrátora. Toto zahrnuje operace jako je správa uživatelských účtů a změny v nastavení systému.

Endpointy pro správu dat, jako jsou logy, mohou vyžadovat oprávnění manažera, který může zobrazit logy z různých účtů bez omezení.

Běžní uživatelé mají přístup pouze k endpointům, které jsou nezbytné pro základní funkce, jako je zadávání a správa vlastních logů a úprava osobních informací.

Tento přístup nejenže zvyšuje bezpečnost našeho systému tím, že provádí validaci uživatelů na straně serveru a omezuje přístup na základě uživatelské role, ale také zjednodušuje správu oprávnění, protože každá role má jasně definovaný obor povolených akcí.

4.4 API

Přehled API endpointů

Do souboru *„main.py“* byly implementovány všechny endpointy, pomocí FastAPI frameworku, kde každá funkce odpovídá specifickému API volání. Každému endpointu je specificky přiřazena HTTP metoda a v případě potřeby jsou omezeny operace, které mohou provádět, tak aby bylo riziko pro bezpečnost dat nebo integrity systému minimální.

Pro lepší pochopení je zde příklad implementace endpointu pro vytváření a čtení logů. Zdrojový kód ukazuje, jak je využitý framework FastAPI společně s SQLAlchemy pro manipulaci s databází a Pydantic model tříd pro validaci dat, kde validaci provádí třída *„AirplaneCreate“*, určená k ověření vstupních dat o letadle.

```
@app.post("/airplanes", status_code=status.HTTP_201_CREATED)
async def create_airplane(airplane: AirplaneCreate, db: DbDependency,
                          current_user: UserBase = Security(get_current_user)):
    '''
    Creates a new airplane with the provided detail
s.  Only accessible by administrators. Assigns the airplane to the current
    user's company and saves it to the database. Returns a success message
    upon creation.

''' if not current_user.isAdmin:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
                            detail="You are not authorized to perform this action.")

    airplane.company_id = current_user.company_id
    db_airplane = models.Airplanes(**airplane.model_dump())
    db.add(db_airplane)
    db.commit()
    return  "message": "Airplane created successfully."}
          {
```

Obrázek 6. Ukázka endpointu pro vložení nového letadla

Oprávnění jednotlivých endpointů

Všechny endpointy v systému jsou opatřeny mechanismy pro kontrolu přístupu, které zajišťují, že uživatelé mohou interagovat pouze s daty, ke kterým mají oprávnění. To je implementováno pomocí kombinace JWT autentizace a role-based access control (RBAC), které jsou integrovány přímo do logiky každého endpointu. [11]

V praxi to znamená, že při volání jakéhokoli endpointu, který to vyžaduje, systém nejprve ověří identitu uživatele a poté zkontroluje, zda má uživatel dostatečná oprávnění k vykonání požadované operace. Tento přístup minimalizuje riziko neoprávněného přístupu a zvyšuje celkovou bezpečnost aplikace.

Zde je příklad, jak je zabezpečení implementováno v rámci endpointu. V tomto případě je přístup k endpointu pro čtení logu omezen pouze na uživatele s administrátorskými či manažerskými právy nebo pro své vlastní logy.

```python
@app.get("/logs", status_code=status.HTTP_200_OK)
async def get_logs(db: DbDependency, current_user: UserBase = Security(get_current_user)):
    '''
    Retrieves a list of all logs. Administrators and managers get access to all logs,
    while other users can only see their own. Raises a 404 error if no logs are found.

''' if  current_user.isAdmin or current_user.isManager):
      ( logs = db.query(models.Log).all()
        if logs is None:
            raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Log not found")
        return logs

    logs = db.query(models.Log).filter(models.Log.user_id == current_user.user_id).all()
    if logs is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Log not found")
    return logs
```

Obrázek 7. Ukázka zabezpečeného endpointu

Tento kód ukazuje, jak „*Depends*“ slouží k ověření, že uživatel má potřebné oprávnění předtím, než je mu umožněno provést operaci. Funkce „*get_logs*“ ověří, zda je uživatel oprávněn logy získat a vrátí pouze logy, které uživateli náleží.

Tento přístup, kdy ověřování práv uživatelů probíhá vždy na straně serveru, zajišťuje ochranu pro případnou manipulaci dat na straně klienta.

5 FRONTEND

Kapitola frontend se zaměřuje na praktickou stránku implementace. Je zde podrobně popsáno, jak byly využity knihovny a techniky pro vývoj uživatelského rozhraní. Cílem bylo vytvořit přehlednou a modulárně strukturovanou frontendovou architekturu, která podporuje rychlé iterace a udržitelný vývoj, a zároveň poskytuje dobrou uživatelskou zkušenost na jakémkoliv zařízení. Také je rozebráno využití Vue-Router, Pinia store, Axios a dalších nástrojů, které v aplikaci využíváme.

5.1 Implementace frontendu

Tvorby frontendové části této aplikace začíná instalací Vue.js pomocí příkazu *„npm create vue@latest“*, což zajišťuje základní kostru pro tuto aplikaci. Kromě Vue byla nainstalována i podpora pro JSX Pinia pro efektivní správu stavu aplikace a Vue Router pro navigaci mezi jednotlivými stránkami. Součástí přípravy bylo také pročištění aplikace, aby obsahovala pouze nezbytné složky a soubory. [13][14]

Pro zajištění jednotného a uživatelsky příjemného vizuálního stylu byl do aplikace integrován Bootstrap 5. Tento framework byl importován do souboru *„main.js“*, díky čemu jsou stylistické prvky dostupné napříč celou aplikací. Tyto vizuální styly byly doplněny o soubor *„style.css“*, ve kterém jsou definovány globální, námi upravené styly, včetně písem a základních barev aplikace. Toto vše podporuje jednotný vizuální jazyk a usnadňuje další stylizaci.

Pro spouštění aplikace v developerském režimu a přípravu buildů pro produkční prostředí byl využit modul Vite, který tyto funkce zajišťuje. Komunikace s backendem je zajišťována pomocí knihovny Axios, která umožňuje snadné odesílání HTTP požadavků a správu odpovědí z API. [13]

5.2 Struktura projektu a komponent

V aplikaci byl kladen důraz na čistou a modulární strukturu složek a souborů, která zajišťuje snadnou orientaci a udržitelnost kódu. Kořenová struktura adresáře byla rozdělena do několika podsložek. Složka *„assets“* je určena pro statické zdroje, jako jsou obrázky a globální styly. Složka *„views“* obsahuje jednotlivé stránky aplikace, které obsahují HTML kód, místní styly a logiku jednotlivých komponent.

Pro správu stavu aplikace včetně sdílených metod slouží složka *„services“*, kde jsou uloženy různé Pinia stores podle jejich použití ve specifických oblastech aplikace, například pro autentizaci uživatelů nebo správu dat logů.

Složka *„components“* je rozdělena na podsložky podle funkčnosti komponent, jako jsou *„airplanes“*, *„logs“*, *„users“* a další. Tento přístup udržuje kód přehledný. Každá podsložka obsahuje komponenty, které jsou specifické pro danou doménu, což usnadňuje další rozvoj a údržbu aplikace. V kořenovém adresáři jsou umístěny komponenty pro tlačítko, záhlaví a zápatí stránky, které jsou vždy viditelné a pouze se lehce přizpůsobují dle změny stavu aplikace.

Jednotlivé stránky do sebe implementují jednu či více komponent. Komponenty byly vytvořeny dle Vue standardů, kde každá komponenta obsahuje tři sekce (HTML kód, javascriptovou logiku dané komponenty a lokální CSS styly). [13]

5.3 Důležité stavební prvky frontendu

Vue-router

Pro navigaci a správu směrování v aplikaci byl použit Vue-router, který umožňuje mapování URL adres na specifické komponenty nebo pohledy (views). Tento router je konfigurován v souboru *„/router/index.js“*, kde jsou definovány cesty a přiřazeny odpovídající komponenty.

Vue-router byl nastaven tak, aby reagoval na neexistující cesty pomocí speciálně určené komponenty pro stránku *„not found"*. Díky tomu je uživatel vždy informován, pokud se pokusí přistoupit na stránku, která neexistuje, což zabraňuje zobrazování nežádoucích chybových hlášek.

Duležitým prvkem Vue-routeru je také zabezpečení; router obsahuje ochranné mechanismy, které za pomoci route guards chrání před přístupem k aplikaci a ověřují platnost JWT tokenu. Pokud token není platný nebo je expirován, uživatel je automaticky přesměrován na stránku pro přihlášení (*„/login“*). Tento postup zajišťuje, že k chráněným zdrojům mají přístup pouze ověření uživatelé. [17]

Pinia-store

Pinia Store v naší aplikaci byl implementován pro správu stavu a komunikaci s backendem. V rámci projektu bylo vytvořeno několik store modulů umístěných v adresáři *„stores“*, kdy každý z nich zodpovídá za určitou oblast aplikace. [18]

Struktura a funkce Pinia Store:

LogStore: Tento store spravuje logy - získává, ukládá a aktualizuje logy prostřednictvím API volání. Funkce tohoto storu zahrnují načtení všech logů, přidání nového logu, úpravy existujících a odstranění logu. Store využívá *„http.js“* pro nastavení základní konfigurace axios volání, včetně přidání JWT tokenu do hlaviček požadavků.

Constant Store: Uchovává obecné hodnoty jako seznam platebních metod, kategorie výdajů a seznam letadel. Tyto informace jsou klíčové pro různé části aplikace, jako jsou formuláře pro přidání nebo úpravu logů.

UserStore: Spravuje uživatelská data a operace související s uživateli, jako jsou přidání nového uživatele, aktualizace údajů a mazání uživatelů.

Tento přístup s centralizovaným managementem stavu a dat umožňuje efektivní správu a opětovnou použitelnost kódu. V případě, že potřebujeme udělat změnu v API volání, uděláme ji pouze na jednom místě. [18]

Axios

Axios je v aplikaci využit jako nástroj pro HTTP komunikaci mezi frontendem a backendem. Veškerá konfigurace Axiosu je nastavená v souboru *„http.js“*. Zde je nastavena základní URL pro všechny API volání, což zjednodušuje údržbu a změny v případě potřeby aktualizace cesty k serveru. Dále *„http.js“* přidává JWT token do hlaviček HTTP požadavků, což zajišťuje, že uživatelská relace zůstává chráněná a validní pro každý požadavek. Tento přístup umožňuje udržovat čistotu a modularitu kódu, protože veškerá logika pro nastavení HTTP požadavků je na jednom místě a může být snadno znovu použita napříč různými částmi aplikace. [16]

5.4 Implemantace Composition API

Ve vývoji frontendu bylo Composition API zvoleno pro jeho schopnost zefektivnit a zpřehlednit strukturu komponent. Tento přístup poskytuje výrazné výhody v lepší správě stavu a logiky aplikace, což je zásadní pro efektivní vývoj a snadnou údržbu kódu.

V implementaci je Composition API používáno v rámci *„<script setup>“* tagu, který umožňuje psát méně kódu a přehledněji. Díky reaktivním funkcím, jako jsou *„ref“* a *„onMounted“*, lze reaktivně reagovat na změny stavu a provádět akce při inicializaci komponenty, aniž by bylo nutné stránku znovu načítat. [13]

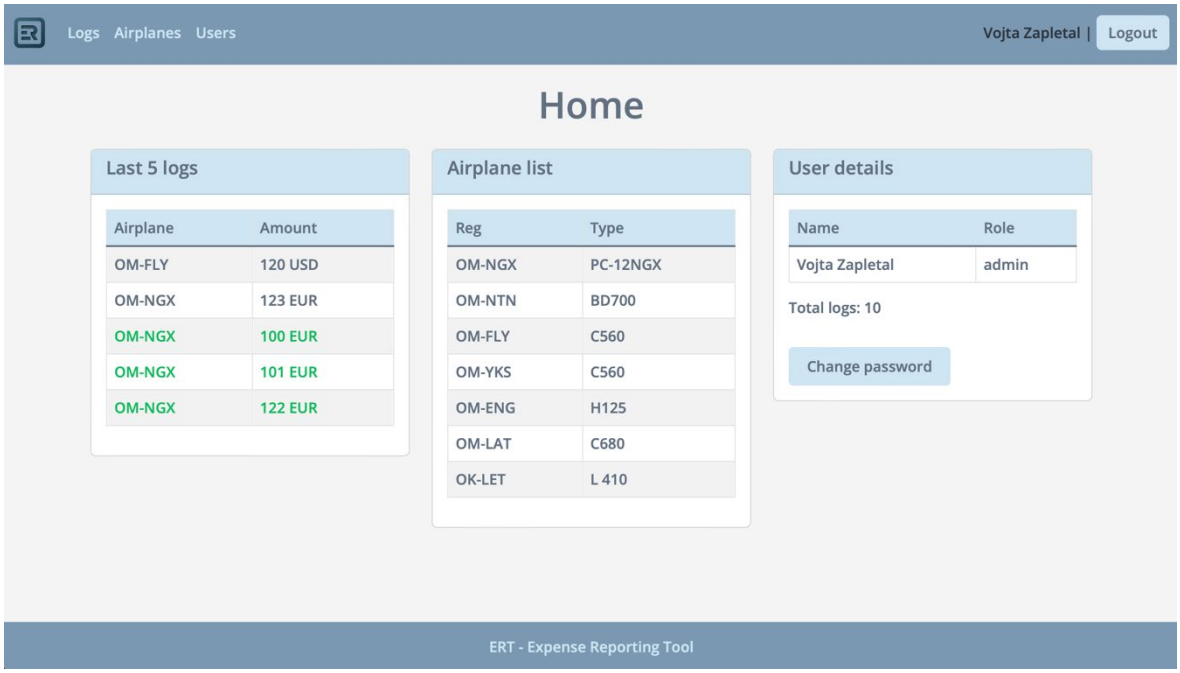
Struktura kódu v Composition API v komponentě začíná importy potřebných knihoven a funkcí, následované deklarací proměnných a definicí metod, které tato komponenta využívá. Tato organizace zvyšuje čitelnost a udržitelnost kódu a umožňuje lepší orientaci v něm.

```
<script setup>
import { ref } from 'vue'
import Button from '@/components/Button.vue'
import { useUserStore } from '@/stores/userStore.js'

const userStore = useUserStore()

const passwords =
    oldPassword: {',
    newPassword: '',
}

const oldPassword = ref('')
const newPassword = ref('')
const confirmPassword = ref('')
const changePassword = () =>
    userStore.changePassword{passwords)
}
</script>
```

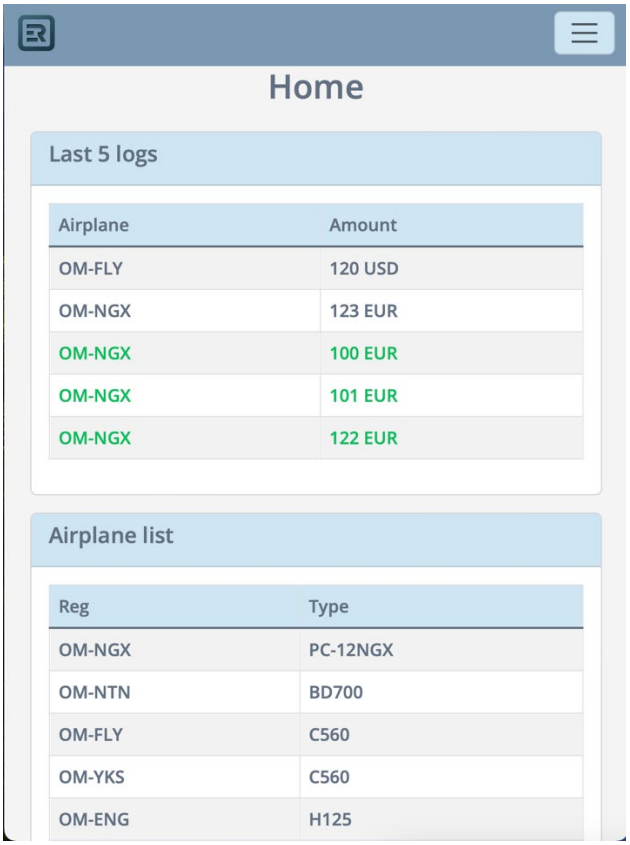
Obrázek 8. Ukázka využití Composition API v komponentě

5.5 Responzivní design a adaptace pro mobilní zařízení

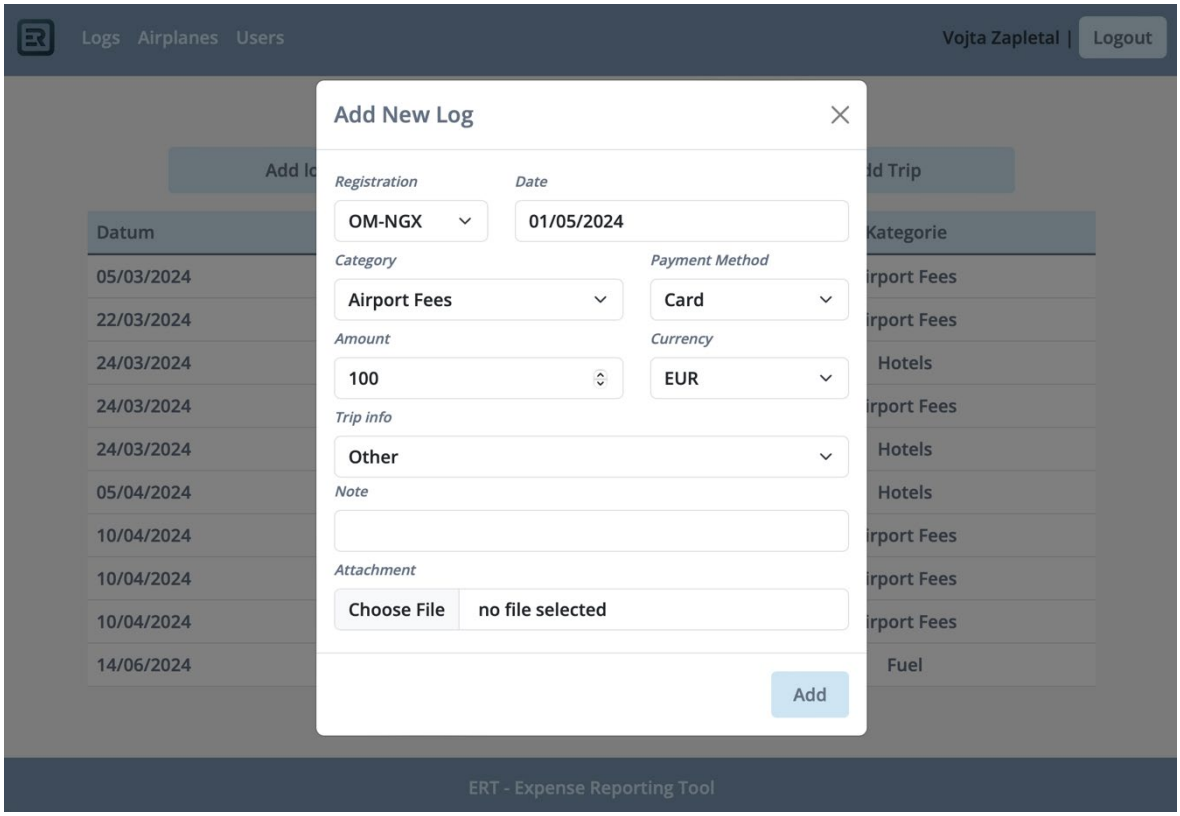
V aplikaci bylo důležité aby byla responzivní a plně se přizpůsobovala na různé velikosti obrazovek. Díky použití Bootstrapu, který má pečlivě navržené CSS styly, bylo zabezpečeno, že aplikace funguje bez problémů na široké škále zařízení, od velkých monitorů po mobilní telefony. Tato univerzálnost zajišťuje, že uživatelé mohou přistupovat k aplikaci bez ohledu na to, kde se nacházejí. Jako příklad byly v aplikaci použity bootstrapové třídy jako *„row“*, *„col-6“*, *„col-md-3“* a byly využity pro dynamické rozložení komponent. Tyto třídy umožňují, aby prvky na širších obrazovkách byly zobrazeny vedle sebe a na úzkých zařízeních se uspořádají pod sebou, což optimalizuje uživatelské prostředí pro každé zařízení a zlepšuje uživatelský zážitek. [15][20]



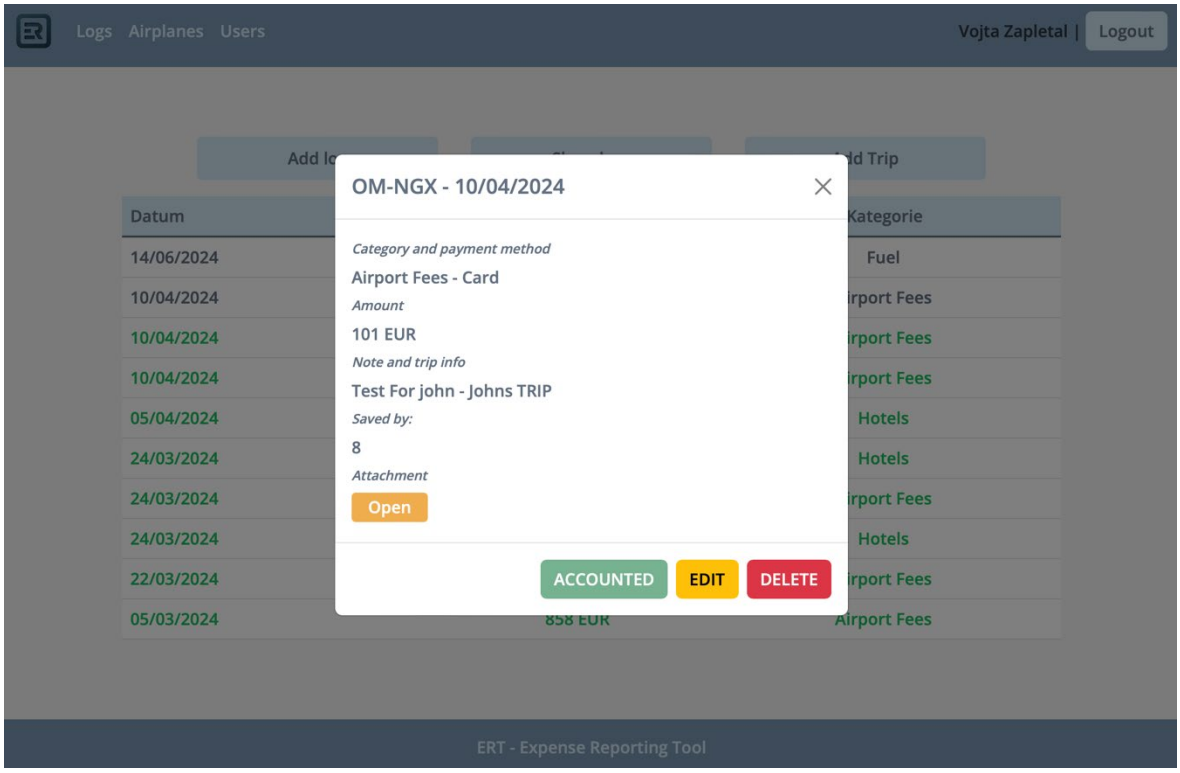
Obrázek 9. Ukázka Home Page



Obrázek 10. Ukázka Home Page – mobilní zařízení



Obrázek 11. Ukázka Add Log



Obrázek 12. Ukázka Edit Log

6 CLOUD DEPLOYMENT

V této kapitole si popíšeme, jak byl systém implementován a nahrán na cloudovou platformu. Tato sekce poskytne pohled na kroky a technologie použité k zajištění hladkého a bezpečného provozu této aplikace v cloudovém prostředí.

6.1 Volba cloudové platformy

Pro hostování této aplikace byl vybrán provozovatel cloudových úložišť Linode. Zde byl vytvořen server s operačním systémem Ubuntu, což je linuxová distribuci založená na Debian Linux. Linode bylo zvoleno kvůli jeho flexibilitě, výkonnosti, snadné instalaci a dobrému poměru cena/výkon, což je ideální pro potřeby tohoto projektu. Platforma nabízí stabilní prostředí pro provoz jak backendu, tak i frontendu aplikace. [21]

6.2 Nastavení a deployment aplikace

Implementace systému v cloudovém prostředí zahrnovala několik klíčových kroků:

Docker a Docker Compose: Pro snadný deployment a zajištění konzistence mezi vývojovým a produkčním prostředím byl použit Docker. Byl vytvořen Dockerfile, který definuje všechny potřebné závislosti a spustí backend na portu 80. S pomocí Docker Compose byly nastaveny služby pro backend a databázi, které běží v izolovaných kontejnerech a jsou navzájem propojené. Pro aktualizaci tak stačí stáhnout poslední verzi aplikace z verzovacího systému, v tomto případě GitHubu a restartovat Docker Compose. [22]

Webový server Caddy: Před backendové a databázové kontejnery byl umístěn webový server Caddy, který řeší HTTPS spojení. Caddy automaticky spravuje SSL/TLS certifikáty prostřednictvím Let's Encrypt, což zjednodušuje správu bezpečnostních certifikátů. A automaticky všechny požadavky na server směřuje na zabezpečené HTTPS spojení.

DNS a směrování: DNS záznamy pro použitou doménu byly nastaveny tak, aby směrovaly na domény *„api.gluti.cz“* pro backend aplikace a *„ert.gluti.cz“* pro frontend aplikace.

6.3 Zabezpečení a monitorování cloudového prostředí

Pro zabezpečení serveru a aplikace byla přijata řada opatření:

Firewall (UFW): Na serveru byl nainstalován a nakonfigurován firewall UFW (Uncomplicated Firewwall), který slouží jako první obranná linie proti nežádoucím přístupům. UFW byl nastaven tak, aby povoloval pouze žádoucí síťový provoz a blokoval nežádoucí spojení na portech, které nejsou používány.

HTTPS a SSL/TLS: Jak již bylo zmíněno, využití Caddy serveru umožňuje automatické řízení SSL/TLS certifikátů, což zaručuje bezpečné šifrované spojení mezi klientem a serverem.

Monitorování a logování: Systémové a aplikační logy jsou monitorovány a analyzovány, aby bylo možné rychle reagovat na jakékoli chybové hlášky bezpečnostní incidenty nebo technické problémy. Logy jsou kontrolovány pomocí příkazu *„docker-compose logs“*, který vypíše celou historii příkazů.

Tato kapitola tedy poskytla náhled na to, jak bylo nastaveno zabezpečení provozu aplikace v cloudovém prostředí, což je důležité pro bezproblémový běh aplikace.

7 TESTOVÁNÍ SYSTÉMU

Pro ověřování funkčnosti našeho systému byly použity tři klíčové metody, které zajišťují, že aplikace splňuje všechny požadavky a je použitelná před jejím finálním nasazením. První metodou je kontrola vybraných endpointů, systematické jednotkové testování, které se automaticky spouští s každým novým nahráním verze systému do Git repozitáře. Tento přístup umožňuje identifikovat a opravit problémy dříve, než se dostanou do produkčního prostředí.

Druhá metoda využívá nástroj Swagger, který vývojářům umožňuje interaktivně testovat různé kombinace vstupů pro endpointy přímo ve webovém prohlížeči. Tato metoda je obzvláště užitečná pro simulaci složitějších scénářů a ověření, že se API chová správně při různorodých vstupních datech.

Poslední fází ověřování je praktické testování systému nezávislým uživatelem, který dostane za úkol provést řadu operací a ověřit, zda dokáže s aplikací efektivně pracovat. Toto testování zahrnuje interakci s reálnými uživatelskými scénáři a poskytuje cennou zpětnou vazbu na uživatelskou přívětivost a celkovou funkčnost systému.

Každá z těchto metod přispívá k důkladnému ověření systému z různých úhlů pohledu a zajišťuje, že finální produkt je funkční, uživatelsky přívětivý.

7.1 Ověření funkčnosti uživatelem

Testování systému nezávislým uživatelem je další součástí ověření uživatelské přívětivosti a funkčnosti. Pro tento účel byl vybrán uživatel, který nebyl předtím se systémem seznámen. Uživateli byly poskytnuty přihlašovací údaje a zadány tři konkrétní úkoly, které měly simulovat běžné činnosti typického uživatele systému.

Prvním úkolem bylo přihlášení do systému a vytvoření nového záznamu s přílohou. Uživatel byl schopen se úspěšně přihlásit a bez problémů navigoval v uživatelském rozhraní k vytvoření záznamu, což zahrnovalo nahrání přílohy. Tento krok dokončil správně a bez jakýchkoliv komplikací.

Druhý úkol spočíval v zobrazení tohoto nově vytvořeného záznamu, změně poznámky u záznamu a označení záznamu jako zaúčtovaného. Uživatel i tento úkol zvládl bez obtíží, což svědčí o tom, že editace dat je jednoduchá a systém správně reflektuje provedené změny.

Třetí úkol testoval správu uživatelů, kde bylo požadováno smazání nežádoucího uživatele. Uživatel byl schopen lokalizovat správnou sekci pro správu uživatelů, vyhledat uživatele a úspěšně smazat zvolený účet.

Úspěšné splnění všech tří úkolů bez nápomoci a váhání potvrzuje, že systém je intuitivní a snadno použitelný i pro osoby, které s ním nemají předchozí zkušenost. Tato fáze testování poskytla cenné potvrzení, že uživatelské rozhraní je logicky organizované, systém funguje jak je zamýšleno a že systém může být využíván v reálných provozních podmínkách.

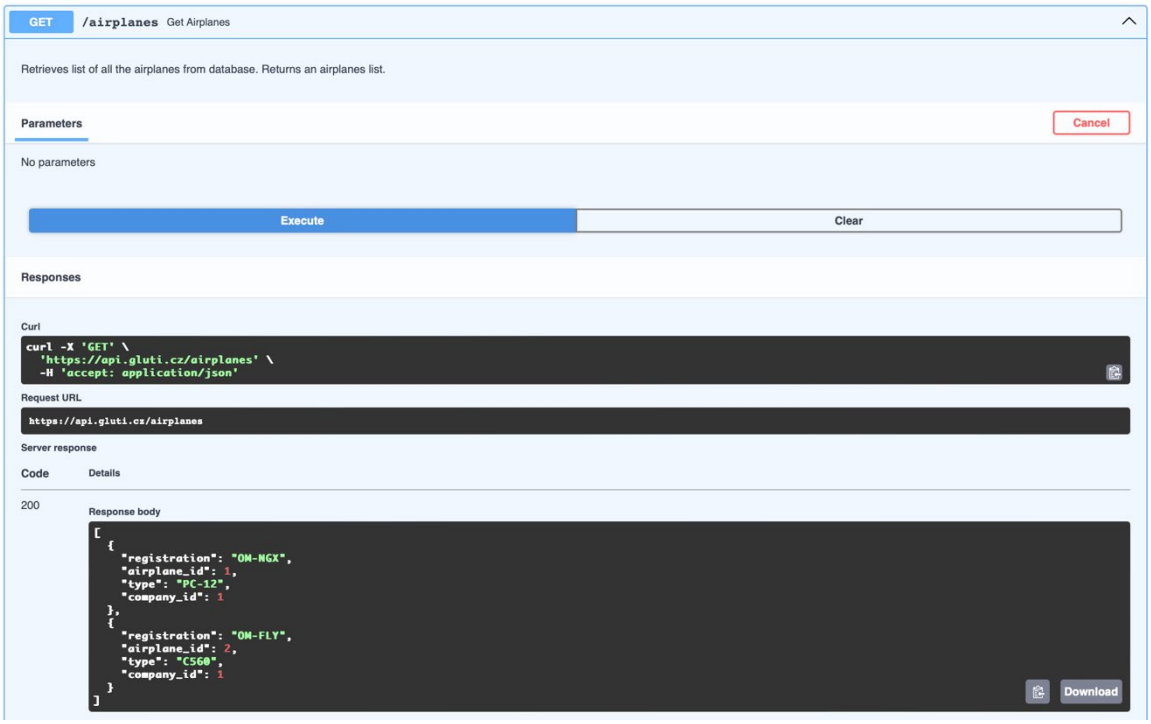
7.2 Ověření pomocí FastAPI nástrojů

Testování API pomocí Swaggeru a Redoc je velmi nápomocným krokem ve vývoji našeho systému. Tyto nástroje poskytují interaktivní rozhraní, které vývojářům umožňuje provádět API volání přímo z webového prohlížeče, což značně usnadňuje testování a validaci chování API. Díky Swaggeru (dostupnému na endpointu *„/docs“*) a Redocu (na *„/redoc“*) mohou vývojáři snadno procházet dostupné API endpointy, vidět očekávané parametry volání, možné odpovědi a statusové kódy, které API vrací.

Během vývoje aplikace byly pomocí těchto nástrojů provedeny stovky testovacích volání, což umožnilo iterativně ověřovat a upravovat chování API. Každý endpoint byl pečlivě testován s různými kombinacemi vstupních parametrů a kontextů, aby bylo zajištěno, že systém správně reaguje na všechny očekávané i neočekávané vstupy.

V průběhu těchto testů bylo možné identifikovat a opravit chyby v logice a vylepšit zabezpečení. Tyto nástroje také poskytly cennou zpětnou vazbu pro další rozvoj API.

Díky tomuto přístupu k testování, kde byl každý endpoint prověřen na různé kombinace parametrů, bylo dosaženo toho, že všechny API volání v poslední fázi vývoje odpovídaly dle očekávání. Tento postup zajistil, že finální produkt je stabilní a připravený k nasazení a užívání v reálném prostředí.



Obrázek 13. Swagger nástroj

8 MOŽNOSTI DALŠÍHO ROZVOJE

V této kapitole jsou nastíněny budoucí směry, kterými by se mohla aplikace dále vyvíjet. Hlavním cílem dalšího rozvoje je rozšíření aplikace o mobilní verzi pro iOS zařízení, zejména pro iPhone a iPad. Tyto zařízení jsou často využívány jako EFB (Electronic Flight Bag) v letectví, kde nahrazují tradiční papírové dokumenty jako mapy, tabulky pro výpočty výkonů a letové příručky. Implementace aplikace do EFB by umožnila posádkám nahrávat a spravovat doklady ze zařízení, které na palubě vždy používají.

8.1 Vývoj aplikace pro iOS

V rámci dalšího rozvoje aplikace je možnost vytvoření dedikované aplikace pro iOS, která by se stala součástí EFB, které mají posádky vždy na palubě letadla.

Výběr technologie pro iOS aplikaci:

Pro vývoj iOS aplikace je možné zvolit několik technologických řešení. Jednou z možností je využití PWA (Progressive Web Apps), což umožňuje vytvářet aplikace fungující podobně jako nativní aplikace, ale jsou přístupné přímo z webového prohlížeče. Druhou alternativou je využití Ionic frameworku, který umožňuje vývoj aplikací s využitím stejných technologií jako webovou aplikaci jako je HTML, CSS a JavaScript, a to s přímým využitím Vue.js nebo React.js, který byl zpočátku uvažován pro vývoj. Ionic framework nabízí výhody v podobě široké kompatibility s různými platformami a efektivního přístupu k vývoji mobilních aplikací s bohatými uživatelskými rozhraními.[23]

Výzvy a omezení:

S nedávnými aktualizacemi operačního systému iOS, kdy společnost Apple spekulovala o omezení možnosti používání PWA, by byl vývoj spíše orientován framework Ionic, který poskytuje lepší možnosti integrace a lepší kontrolu nad bezpečnostními aspekty aplikace. Ionic umožňuje efektivně vytvářet aplikace, které mohou bez problémů fungovat na různých zařízeních Apple, což je požadované pro integraci s EFB systémy v letadlech a to i v případě, že Apple se znovu rozhodne omezit používání PWA na jejich zařízeních. [24]

Vývoj iOS aplikace je vnímán jako logický krok v evoluci našeho systému, který nejen zlepší uživatelský zážitek, ale také zvýší jeho praktičnost a efektivitu v každodenním používání v letovém provozu.

8.2 Umělá inteligence

Dalším krokem pro rozvoj aplikace je integrace umělé inteligence pro automatizaci a zefektivnění procesů, jako je rozpoznávání a validace údajů z nahraných dokladů. Implementace umělé inteligence by mohla automaticky extrahovat data z dokladů, ověřovat jejich správnost a před vyplňovat relevantní informace, což by značně snížilo čas potřebný pro ruční zpracování a zvýšilo přesnost dat.

8.3 Integrace s dalšími systémy

Rozvoj aplikace by také mohl směřovat k integraci s dalšími systémy používanými v letectví a podnikové sféře:

Technické dokumentace letadel: Integrace s technickým deníkem letadel by umožnila lépe přiřazovat náklady ke konkrétnímu letu.

Účetní systémy: Automatizovaný export dat do účetních systémů by usnadnil účetním práci s předem uživatelem ověřenými a konzistentními daty. Tato integrace by byla závislá na API rozhraní využívaného účetního systému.

Bankovní integrace: Spárování platebních transakcí z banky s údaji v aplikaci by zefektivnilo správu finančních toků a zvýšilo by přehled o platbách. Aplikace by mohla rozpoznat chybějící doklady v databázi a naopak by mohla upozornit pokud není možné vložený doklad spárovat s platbou.

Tyto možnosti rozvoje naznačují, jak by se aplikace mohla v budoucnu stát ještě užitečnější a integrovanější, což by přispělo k lepší efektivitě a zjednodušení firemních a finančních operací ve společnosti.

ZÁVĚR

Tato bakalářská práce se zabývala vytvořením aplikace pro evidenci dokladů v prostředí leteckého provozovatele.

Cílem této práce bylo vytvořit funkční aplikaci, která bude sloužit k elektronické evidenci dokladů, čímž přispěje k digitalizaci systému. Toto bude mít pozitivní vliv na efektivitu chodu firem leteckých provozovatelů.

V teoretické části práce bylo nejprve popsáno, proč bylo žádoucí takovou aplikaci vyvinout, k čemu bude leteckým provozovatelům sloužit a jaká úskalí by měla vyřešit. Dále byla provedena analýza existujících řešení, které jsou na trhu dostupné. Následovalo pak jejich srovnání a shrnutí výsledků tohoto zkoumání.

Nakonec byly popsány technologie, které byly v této bakalářské práci při vývoji aplikace použity, a to jak v backendu, tak ve frontendu včetně popisu bezpečnostních mechanismů, které byly v aplikaci použity.

Praktická část práce začíná návrhem celého systému, kde byly shrnuty základní požadavky na aplikaci a ukázány některé případy užití podrobněji. Byly zde popsány i jednotlivé endpointy a jejich vlastnosti.

Dále zde byla popsána tvorba backendu. Jak byl backend implementován. Jak funguje připojení k databázi a také jak je implementováno zabezpečení systému, tak aby probíhala autorizace i autentizace vždy kdy je to potřebné.

V podobném duchu byla popsána také tvorba frontendu pro webovou aplikaci a to jak byly jednotlivé části frontendu implementovány a konfigurovány. Jak byl systém, zabezpečen implementován do cloudu a jak bylo uživatelem ověřeno jeho chování.

Závěrečná kapitola nastiňuje, jaké jsou další možnosti rozvoje aplikace do budoucna.

Návrh systému byl úspěšně realizován a byly splněny všechny stanovené cíle. Přestože nebylo možné implementovat všechny plánované funkce, základní funkcionalita je zajištěna a systém poskytuje solidní základ pro další rozvoj.

Díky práci na tomto projektu jsem si prohloubil znalosti v oblasti fullstack vývoje a splnil tak i hlavní cíl, kterého jsem během tohoto studia chtěl dosáhnout. Věřím, že poskytne pevný základ pro mé budoucí kariérní směřování i další akademické studium.

SEZNAM POUŽITÉ LITERATURY

[1] *Expense management: unlimited receipt tracking and management for all* [online]. 2024 [cit. 2024-04-30]. Dostupné z: https://use.expensify.com/expense-management

[2] *Why choose Evernote?* [online]. 2024 [cit. 2024-04-28]. Dostupné z: https://evernote.com/why-evernote

[3] *Invoice managment* [online]. 2024 [cit. 2024-05-01]. Dostupné z: https://www.getmoss.com/en-gb/invoice-management

[4] GRINBERG, Miguel. *Flask web development: developing web applications with Python*. Second edition. Beijing: O'Reilly, 2018. ISBN 978-1491991732.

[5] *Python FastAPI vs Flask: A Detailed Comparison* [online]. 2024 [cit. 2024-04-13]. Dostupné z: https://www.turing.com/kb/fastapi-vs-flask-a-detailed-comparison

[6] *FastAPI - Documentation* [online]. 2024 [cit. 2024-04-15]. Dostupné z: https://fastapi.tiangolo.com

[7] *Úvod do FastAPI* [online]. 2024 [cit. 2024-04-28]. Dostupné z: https://www.itnetwork.cz/python/fastapi/uvod-do-fastapi-frameworku-a-webovych-aplikaci-v-pythonu

[8] *MySQL Cookbook*. 4th edition. O'Reilly Media, 2022. ISBN 978-1492093169.

[9] *Pydantic* [online]. 2024 [cit. 2024-04-30]. Dostupné z: https://docs.pydantic.dev/latest/why/

[10] *SQLAlchemy* [online]. 2024 [cit. 2024-04-30]. Dostupné z: https://www.sqlalchemy.org

[11] *Introduction to JSON Web Tokens* [online]. 2024 [cit. 2024-04-30]. Dostupné z: https://jwt.io/introduction

[12] *Vue Declarative rendering* [online]. 2024 [cit. 2024-05-02]. Dostupné z: https://www.geeksforgeeks.org/vue-js-declarative-rendering/

[13] NELSON, Brett. Getting to Know Vue.js: Learn to Build Single Page Applications in Vue from Scratch. Apress, 2018. ISBN 978-1484237809.

[14] *Vue.js documentation* [online]. 2024 [cit. 2024-04-13]. Dostupné z: https://vuejs.org/guide/introduction.html

[15] *Get started with Bootstrap* [online]. 2024 [cit. 2024-04-30]. Dostupné z: https://getbootstrap.com/docs/5.3/getting-started/introduction/

[16] *Axis documentation* [online]. 2024 [cit. 2024-05-01]. Dostupné z: https://axios-http.com/docs/intro

[17] *Vue-router docs* [online]. 2023 [cit. 2024-05-01]. Dostupné z: https://router.vuejs.org/guide/

[18] *Pinia documentation* [online]. 2023 [cit. 2024-05-02]. Dostupné z: https://pinia.vuejs.org/introduction.html

[19] *Figma* [online]. 2024 [cit. 2024-05-03]. Dostupné z: https://www.figma.com

[20] SCHAUMBURG, Jensen. The Missing Bootstrap 5 Guide: Customize and extend Bootstrap 5 with Sass and JavaScript to create unique website designs. Packt Publishing, 2022. ISBN 978-1801076432.

[21] AKAMAI. *Linode* [online]. 2023 [cit. 2024-05-03]. Dostupné z: https://www.linode.com/choosing-linode/

[22] *Docker compose* [online]. 2024 [cit. 2024-05-03]. Dostupné z: https://docs.docker.com/compose/intro/features-uses/

[23] *Ionic docs* [online]. 2024 [cit. 2024-05-02]. Dostupné z: https://ionicframework.com/docs

[24] APPLE INC. *Update on apps distributed in the European Union* [online]. 2024 [cit. 2024-05-05]. Dostupné z: https://developer.apple.com/support/dma-and-apps-in-the-eu/#dev-qa

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface
CSS	Cascading Style Sheets
DIČ	Daňové Identifikační Číslo
ECDSA	Eliptic Curve Digital Signature Algorithm
EFB	Electronic Flight Bag
ERD	Entity Relationship Diagram
FAANG	Facebook, Amazon, Apple, Netflix, Google
HTML	Hypertext Markup Language
HMAC	Hash-based Message Authentication Code
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
JWT	JSON Web Token
ORM	Object Relation Mapping
PDF	Portable Document Format
PWA	Progressive Web Apps
RBAC	Role-based Access Control
RFC	Request For Comments
RSA	Rivet-Shamir-Adleman
SFC	Single File Components
SPA	Single Page Application
SQL	Structured Query Language
SSL	Secure Socket Layer
TLS	Transport Layer Security

UFW	Uncomplicated Firewwall
UI	User Interface
URL	Uniform Resource Locator
UTB	Univerzita Tomáše Bati

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

P1 USB disk se zdrojovým kódem a bakalářskou prací