VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

WEBOVÁ APLIKACE PRO SPRÁVU VETERINÁRNÍ ORDINACE

WEB APPLICATION FOR THE ADMINISTRATION OF A VETERINARY PRACTICE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Jakub Foltán

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Jan Roupec, Ph.D.

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Jakub Foltán**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	bez specializace
Vedoucí práce:	**doc. Ing. Jan Roupec, Ph.D.**
Akademický rok:	2022/23

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Webová aplikace pro správu veterinární ordinace

Stručná charakteristika problematiky úkolu:

Problematika počítačové podpory veterinárních ordinací je podobná ordinacím humánní medicíny, má však některé významné odlišnosti. Záměrem práce je
je analyzovat potřeby veterinárních ordinací a prozkoumat existující systémy pro podporu jejich provozu. Na základě této analýzy by měla vzniknout aplikace, která bude potřeby veterinárních ordinací co nejlépe řešit.

Cíle diplomové práce:

V textové části práce bude provedena analýza potřeb počítačové podpory provozu veterinárních ordinací a přehled případných existujících softwarových systémů. Z výsledků analýz vyplynou požadavky na softwarový systém pro provoz veterinární ordinace. Dále se práce bude zabývat technologiemi pro vytváření webových aplikací. Po výběru vhodné technologie bude realizována webová aplikace splňující požadavky definované v analytické části práce.

Seznam doporučené literatury:

LAURENČÍK, M.: Tvorba www stránek v HTML a CSS. Grada, 2019. ISBN 978-80-271-2241-7.

LAURENČÍK, M.: SQL. Grada, 2018, ISBN 978-80-271-0774-2.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2022/23

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRAKT

Tato diplomová práce se zaměřuje na vývoj webové aplikace pro správu veterinární ordinace s cílem poskytnout inovativní a efektivní řešení pro moderní softwarové potřeby. Práce začíná analýzou současného stavu veterinární ordinace a identifikací nedostatků stávajícího softwaru. Na základě požadavků veterinární ordinace byla vyvinuta webová aplikace, která umožňuje evidenci zvířat a klientů, správu skladu, fakturaci a veterinární úkony. Při vývoji aplikace byly použity moderní technologie včetně Django frameworku pro backend, HTML, CSS a JavaScriptu pro frontend a PostgreSQL databáze pro správu dat. Testování potvrdilo, že výsledná aplikace efektivně řeší potřeby veterinární ordinace. Tato práce ukazuje, že vývoj vlastní webové aplikace může být účinným řešením pro specifické potřeby klienta a přináší možnosti pro budoucí vývoj a vylepšení.

ABSTRACT

This thesis focuses on the development of a web-based veterinary practice management application to provide an innovative and effective solution for modern software needs. The thesis starts by analyzing the current state of the veterinary practice and identifying the shortcomings of the existing software. Based on the requirements of the veterinary practice, a web application was developed to enable animal and client registration, warehouse management, billing and veterinary operations. Modern technologies were used in the development of the application including Django framework for the backend, HTML, CSS and JavaScript for the frontend and PostgreSQL database for data management. Testing confirmed that the resulting application effectively addresses the needs of the veterinary practice. This work demonstrates that developing a custom web application can be an effective solution to a client's specific needs and provides opportunities for future development and enhancements.

KLÍČOVÁ SLOVA

webová aplikace, správa veterinární ordinace, Python, Django, PostgreSQL

KEYWORDS

web application, veterinary practice management, Python, Django, PostgreSQL

2023

BIBLIOGRAFICKÁ CITACE

FOLTÁN, Jakub. *Webová aplikace pro správu veterinární ordinace*. Brno, 2023. Dostupné také z: `https://www.vut.cz/studenti/zav-prace/detail/149672`. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, Vedoucí práce: doc. Ing. Jan Roupec, Ph.D.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků.

V Brně dne 21. 5. 2023 .....

Jakub FOLTÁN

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Janu Roupcovi, Ph.D. za jeho ochotu a poskytnutí možnosti psát práci na toto téma. Dále chci poděkovat Veterině Mája Oslavany za spolupráci a konzultace v rámci vývoje aplikace. Především však poděkování patří mé rodině a přítelkyni za jejich neustálou podporu a trpělivost během celého studia.

OBSAH

1 ÚVOD

V dnešní digitální éře, kde se technologie stávají nezbytnou součástí každodenního života, je efektivní správa podniků a institucí nezbytnou součástí jejich úspěchu. Veterinární ordinace nejsou výjimkou. S nárůstem poptávky po kvalitní veterinární péči se zvyšuje i potřeba moderního softwaru, který usnadní a zefektivní jejich provoz. Tato diplomová práce se zaměřuje právě na vývoj webové aplikace pro správu veterinární ordinace, která přináší inovativní řešení a přináší efektivitu, spolehlivost a uživatelskou přívětivost.

Cílem této práce bylo analyzovat současný stav veterinární ordinace a identifikovat nedostatky stávajícího softwaru, který je v ordinaci využíván. Ve spolupráci s veterinární ordinací Mája Oslavany byla provedena důkladná analýza požadavků a byly stanoveny přednosti a nedostatky současného softwaru. Byl také proveden průzkum dostupných řešení na místním trhu, ale žádné z nich nebylo dostačující pro potřeby klienta. Proto bylo rozhodnuto vyvinout vlastní webovou aplikaci, která nejlépe odpovídá požadavkům veterinární ordinace a poskytuje přístup z libovolného zařízení a možnost provozu aplikace z více míst současně.

V rámci této práce dojde k seznámení s technologiemi pro vývoj webových aplikací. Jsou zde popsány základní koncepty webových aplikací a technologií. Dále jsou představeny frontend technologie a backend technologie včetně jazyků a frameworků pro backend vývoj a dostupné možnosti pro databáze.

V rámci výsledné části práce byl detailně popsán proces vytvoření webové aplikace pro správu veterinární ordinace. Byly uvedeny hlavní funkce aplikace, použité technologie pro backend a frontend, návrh databáze a nastavení vývojového prostředí. Dále byla provedena implementace jednotlivých aplikací, které odpovídají potřebám veterinární ordinace. Tyto aplikace zahrnují kartotéku zvířat a klientů, aplikaci pro sklad, faktury a úkony. Při vývoji aplikace byl využit framework Django pro backend, který poskytuje robustní nástroje a je populární mezi vývojáři. Pro frontend byly použity základní webové technologie - HTML, CSS a JavaScript. Pro správu dat byla zvolena databáze PostgreSQL.

Tato diplomová práce představuje webovou aplikaci pro správu veterinární ordinace, která usnadňuje efektivní provoz a poskytuje uživatelsky přívětivé rozhraní. Aplikace umožňuje evidenci zvířat a klientů v kartotéce, správu skladových zásob, seznamu úkonů a také generování a úschovu faktur. S ohledem na stále rostoucí poptávku po kvalitní veterinární péči, je tato webová aplikace odpovědí na potřebu moderního softwaru pro veterinární ordinace.

2 PROBLEMATIKA POČÍTAČOVÉ PODPORY VETERINÁRNÍ ORDINACE

Veterinární ordinace jsou zdravotnická zařízení zaměřená na poskytování profesionální péče, diagnostiky a léčby zvířat. Provoz takovýchto zařízení je náročný a komplexní, jelikož zahrnuje řadu specifických činností a úkolů spojených s organizací práce, komunikací s majiteli zvířat a správou informací o pacientech a jejich léčbě. Počítačová podpora v tomto kontextu hraje klíčovou roli, neboť může významně přispět k efektivitě, zjednodušení a zlepšení kvality služeb poskytovaných veterinárními odborníky. Správně navržený a implementovaný softwarový systém může usnadnit řadu procesů, což umožňuje zaměstnancům veterinárních ordinací soustředit se na péči o zvířata. V následujících podkapitolách budou upřesněny hlavní potřeby a specifika na software pro správu veterinární ordinace.

Je nutné zmínit, že níže uvedené potřeby a specifika jsou obecné a nemusí být vhodné pro všechny ordinace. Zároveň jde o výčet pouze základních potřeb a je možné takový software rozšířit o nespočetné množství dalších prvků, závisejících na konkrétních potřebách jednotlivých ordinací. Příkladem může být potřeba zpracovávat a uchovávat laboratorní výsledky, pokud je daná ordinace takovým pracovištěm vybavena.

2.1 Kartotéka

Jednou z nejdůležitějších částí softwaru pro správu veterinární ordinace je kartotéka, která umožňuje vést veškeré potřebné záznamy v digitální podobě. Oproti humánní medicíně, kde je jako pacient brán člověk, je v oblasti veterinárního lékařství pacientem zvíře. Zvíře ovšem nelze evidovat samostatně bez jeho majitele. Tudíž oproti humánním softwarovým systémům je nutné zahrnout vztah majitel - pacient (zvíře). K majiteli i k jeho zvířatům jsou pak evidovány základní údaje pro potřeby identifikace, či další údaje pro usnadnění péče o zvířata např. zda zvíře již bylo kastrované). Ke zvířatům jsou následně evidovány jeho návštěvy spolu se záznamem z návštěvy.

2.2 Skladová evidence

Nezbytnou součástí softwaru pro správu veterinární ordinace je rovněž vedení skladové evidence pro snadný přehled o léčivech, zdravotnickém materiálu, krmivech a dalších skladových položek potřebných pro provoz veterinární ordinace. Kromě evidování množství dané skladové položky je nutné evidovat také datum expirace, pro případnou likvidaci prošlých léčiv, krmiv a dalších položek podléhající expirační

lhůtě. Dalším údajem, které je zejména v případě léčiv a krmiv velmi důležitý je výrobní šarže. Ta slouží v případě nějaké chyby ve výrobě ke snadné identifikaci vadných výrobků a stažení z prodeje. Následně se pak standardně evidují údaje o nákupní a prodejní ceně, DPH, marži apod.

2.3 Účetnictví

Neméně důležitou součástí systému pro správu veterinární ordinace je také vedení účetnictví. Není zde na mysli kompletní správa účetnictví pro celou ordinaci, k tomu zpravidla slouží specializovaná softwarová řešení, ale je vhodné, aby systém poskytoval možnost vytvořit a uchovávat faktury spojené s návštěvou pacienta. Užitečná pak může být možnost exportu všech faktur za zvolené časové období.

2.4 Seznamy

Užitečnou součástí softwaru pro správu veterinární ordinace mohou být seznamy. Ty slouží ke snadné kategorizaci a uspořádání dat, která se často opakují. Takové seznamy lze například použít pro úkony, diagnózy, druhy zvířat, plemena apod. Samozřejmě by mělo být možné takovéto seznamy rozšiřovat na základě aktuálních potřeb.

3 ANALÝZA SOUČASNÉHO STAVU KLIENTA

Jak již bylo zmíněno v úvodu, modelovým klientem pro potřeby analýzy potřeb a vývoje softwaru, byla Veterina Mája Oslavany s.r.o. Jedná se o menší pracoviště s jednou ordinací, kde ordinují celkem 4 veterináři. Návštěvnost se pohybuje okolo 600 - 700 návštěv měsíčně. V současnosti používají desktopovou aplikaci Vetis office, která nesplňuje některé požadavky klienta, a proto bylo přistoupeno k hledání jiného řešení.

Na českém trhu existuje více dostupných řešení, které řeší problematiku podpory provozu veterinární ordinace, ne všechny však musí být vhodné. Samozřejmě existují i řešení zahraniční, ty však ve většině případů nejsou pro český trh vhodně uzpůsobená vzhledem k lišící se legislativě mezi různými státy, proto se jimi dále nebude práce zabývat. Pro nalezení nejlepšího řešení je zapotřebí nejdříve stanovit funkce, které v původním softwaru klientovi vyhovují, a naopak také definovat nedostatky, které by nová aplikace měla řešit.

3.1 Popis současného softwaru

Vetis office je prezentován jako veterinární informační systém, který nabízí komplexní řešení lékařské a účetní agendy se specifiky veterinární praxe. Je neustále vyvíjen dle požadavků klienta a legislativních změn. Program je koncipován jako modulární, tedy je možné si dokoupit moduly, které uživatel skutečně potřebuje a nemusí platit za funkce, které nevyužije. Nabízí i možnost provozovat systém v síti nebo v režimu klient-server. Součástí je spolupráce s diagnostickými přístroji nebo s RFID čtečkami. Pro usnadnění práci se skladem lze importovat skladové položky automaticky pomocí načtení souboru dodávky. [1]

3.1.1 Přednosti současného softwaru

Současný software má některé přednosti, které je vhodné při výběru nového řešení pokud možno zachovat:

- **Přehledná kartotéka** - Pro vysokou efektivitu při práci se systémem je podstatná snadná orientace v záznamech. Program Vetis office má kartotéku k dispozici hned při spuštění a nabízí rychlé vyhledávání mezi majiteli a přehledné zobrazení jejich zvířat a případů k danému zvířeti. Zároveň jsou všechny důležité informace zobrazeny přehledně v tabulce.
- **Rozdělení návštěv na případy** - Velmi ceněnou vlastností současného programu je rozdělení návštěv na případy. To významným způsobem napomáhá přehlednosti. Pro snadnější pochopení této funkce je možné si představit následující příklad: Majitelka zvířete, paní Nováková, přivede svého psa Rexe

k veterináři. Rex má několik zdravotních problémů, která je potřeba vyřešit během několika návštěv:

1. První návštěva: Rex trpí kožním zánětem, veterinář předepíše vhodnou medikaci a péči.
2. Druhá návštěva: Rex se zotavuje s kožního zánětu, ale paní Nováková si všimla, že má problémy s chůzí. Veterinář diagnostikuje artritidu a předepisuje léky.
3. Třetí návštěva: Rex je na kontrolní návštěvě kvůli kožnímu zánětu a artritidě. Veterinář kontroluje zlepšení a přizpůsobuje léčbu.

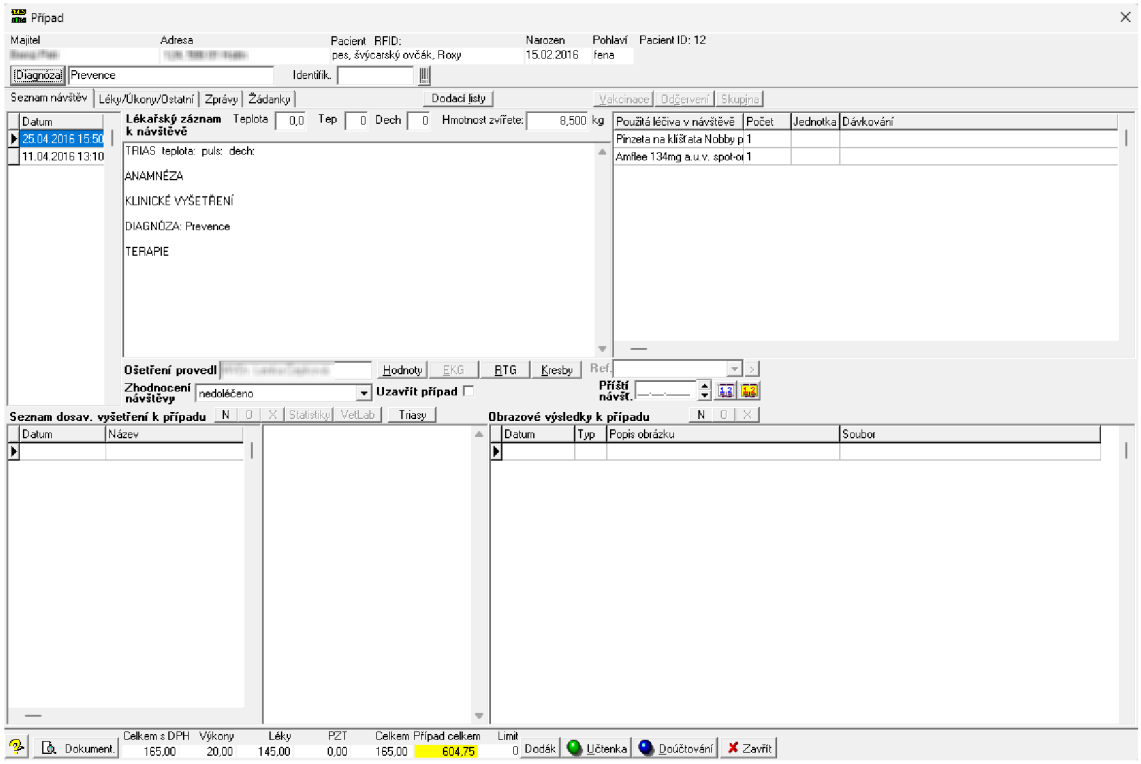
Na tomto modelovém příkladu je vidět, že je velmi užitečné členit jednotlivé návštěvy na případy, protože každá návštěva se týká jiného zdravotního problému. Rozdělení návštěv na případy umožňuje lépe sledovat zdravotní stav zvířete a sledovat reakci na danou léčbu. Zároveň je možné případ označit jako uzavřený čímž se dosáhne snadného rozlišení mezi aktivními a již vyřešenými případy a současně je možné kdykoliv k datům přistupovat pro budoucí léčbu podobných případů.

- **Více skladů** - Možnost mít více skladů je užitečná hned z několika důvodů. Jedním je zřízení více skladů v případě rozšíření o další pracoviště na jiném místě. V takovém případě by bylo značně problematické udržet přehled s jedním skladem pro obě pracoviště. Druhým případem pro rozdělení skladů může být z důvodu nastavení rozdílných marží pro určité typy položek.
- **Účetnictví** - Současný program nabízí bohaté možnosti pro zjednodušení účetnictví. Program umožňuje generovat a uschovávat vydané faktury. Kromě toho nabízí i možnost exportu vydaných faktur za určité časové období.
- **Klient-server verze** - Pro budoucí rozšíření o další pracoviště nebo práci v terénu je výhodou verze programu v režimu klient-server, která umožňuje přístup ke všem potřebným datům z více míst.

3.1.2 Nedostatky současného softwaru

Současný program má několik nedostatků, které by nové řešení mělo vyřešit:

- **Zastaralé a nepřehledné uživatelské rozhraní** - Ačkoliv sekce kartotéky je po stránce uživatelského rozhraní přehledná, většina ostatních části působí nepřehledně a zastarale. Pro nové pracovníky tak bývá problém s orientací v programu. To vede k delšímu zaškolování personálu a nižší efektivitě zejména v počátku, než se pracovník s program dostatečně seznámí. Jako příklad může sloužit obrázek č. 1 zobrazující detail případu.



Obr. 1: Uživatelské rozhraní Vetis office - detail případů

- **Pouze pro operační systém Windows** - Program Vetis office je momentálně dostupný pouze pro operační systém Windows. To v případě provozu na stolním počítači nebo notebooku není velký problém vzhledem k jeho rozšířenosti. Pro práci v terénu však není příliš praktické nosit s sebou notebook, zvláště pokud uvážíme, že veterinář sebou musí brát i značné množství jiného vybavení pro jeho práci. Praktičtější by byla možnost mít přístup do systému například ze smartphonu nebo tabletu.
- **Nevýhody provozu klient-server verze** - V případě, že by klient potřeboval rozšířit pracoviště, nabízí se použití klient-server verze, kde databáze běží na samostatném PC (serveru) a klienti se k němu připojují. Takové řešení má vedle svých výhod i řadu nevýhod, a to hlavně potřeba pořízení dalšího počítače, který bude sloužit jako server, což zvyšuje pořizovací náklady. Další nevýhodou může být stabilita serveru, v případě jeho výpadku nebo ztrátě dat, nelze z klientských počítačů k datům přistupovat a obnovení systému může trvat delší dobu.

3.2 Dostupná řešení

Jak již bylo zmíněno, na českém trhu existuje více dostupných řešení, které se zabývají problematikou počítačové podpory veterinární ordinace. V následujících podka-

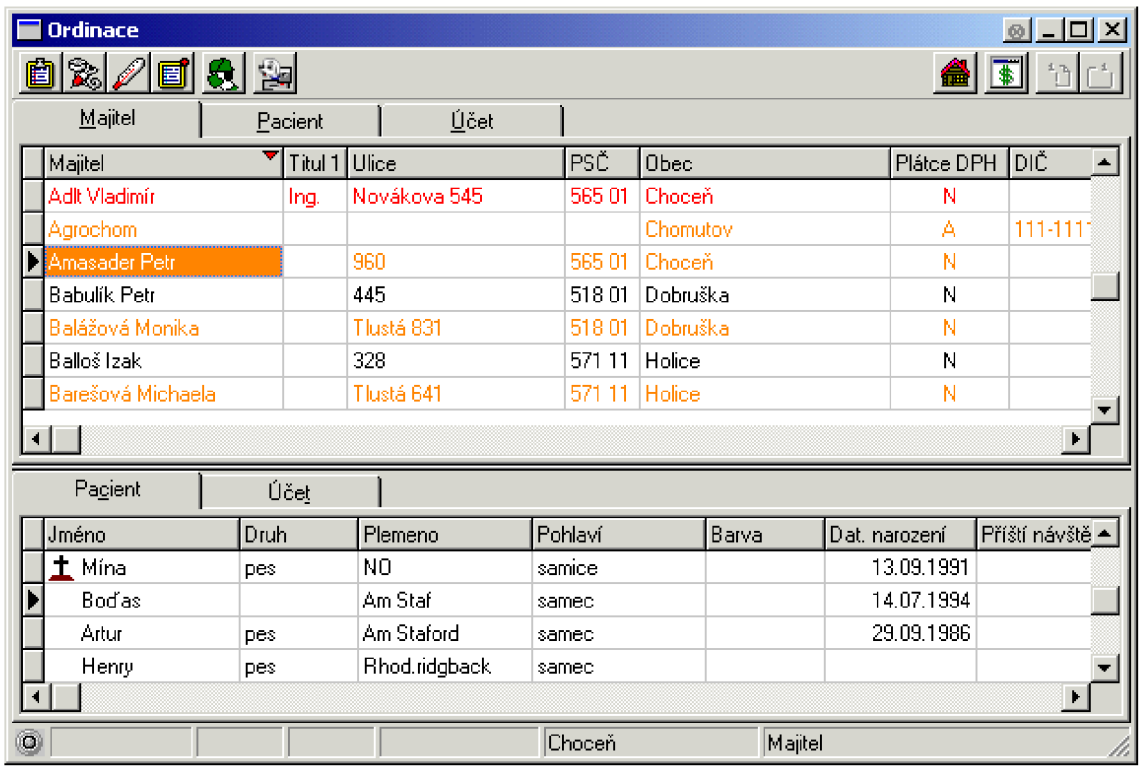
pitolách dojde k představení dostupných řešení a analýze jejich vhodnosti použití pro klienta tak, aby došlo k zachování výhod současného softwaru a odstranění daných nedostatků. U jednotlivých řešení nebudou zmíněny všechny funkce nebo vlastnosti, a to z toho důvodu, že často nabízí velké množství funkcí, které jsou samozřejmě užitečné, ale pro klienta nejsou tolik podstatné.

3.2.1 WinVet

WinVet je program vyvíjený společností Noviko s.r.o., která je největším distributorem produktů pro provoz veterinárních ordinací. Jedná se desktopovou aplikaci pro operační systém Windows, která je neustále vyvíjena. Aplikace je určena jak pro samostatné veterinární lékaře, tak i pro rozsáhlé veterinární kliniky nebo ordinace, díky možnosti provozovat verzi s vyhrazeným serverem. [2]

Aplikace nabízí přehledné vedení kartotéky majitelů a jejich zvířat. Pro jednotlivé návštěvy zvířete je možné přiřadit případ nebo ponechat návštěvu nezařazenou. Součástí programu jsou číselníky pro snadné zadávání některých informací jako jsou plemena, diagnózy apod. Pro skladové hospodářství je dostupná volba více skladů. Ke skladovým položkám jsou evidovány všechny potřebné informace. Lze nastavit sledování zásob na skladě. Rovněž nabízí import skladových položek ze souboru. Pro potřeby účetnictví je v aplikaci implementováno generování a uchovávání faktur, pokladní doklady a vratky. K dispozici jsou i pokročilé funkce jako jsou různé statistiky, propojení s diagnostickými přístroji, SMS brána a další. Z hlediska uživatelského rozhraní je aplikace poměrně nepřívětivá, jednotlivé moduly se otevírají v nových oknech což může postupně vést k nepřehlednosti. [3]

Cenová politika programu umožňuje buď pronájem licence na 1 rok s cenou za jednu licenci 6 888 Kč bez DPH, nebo prodej licence, kde je možné program používat neomezeně dlouho. V takovém případě činí cena za jednu licenci 24 400 Kč bez DPH. Nutno podotknout, že v případě pronájmu, je cena včetně systémové podpory, která zahrnuje nové aktualizace programu, servis a poradenství. V případě zakoupení licence je ovšem program bez této podpory a pro aktualizace je nutné platit roční poplatek pro systémovou podporu, který činí 4 728 Kč bez DPH za jednu licenci. Pokud by klient potřeboval program využít pro více pracovišť, bylo by nutné zakoupit další licence (další licence jsou za výhodnější cenu) a navíc i modul pro spojení databází. V takovém případě může cena velice rychle vystoupat na hodnotu, která nemusí být pro některé ordinace výhodná. [4]



Obr. 2: Uživatelské rozhraní Winvet - kartotéka, převzato z [2]

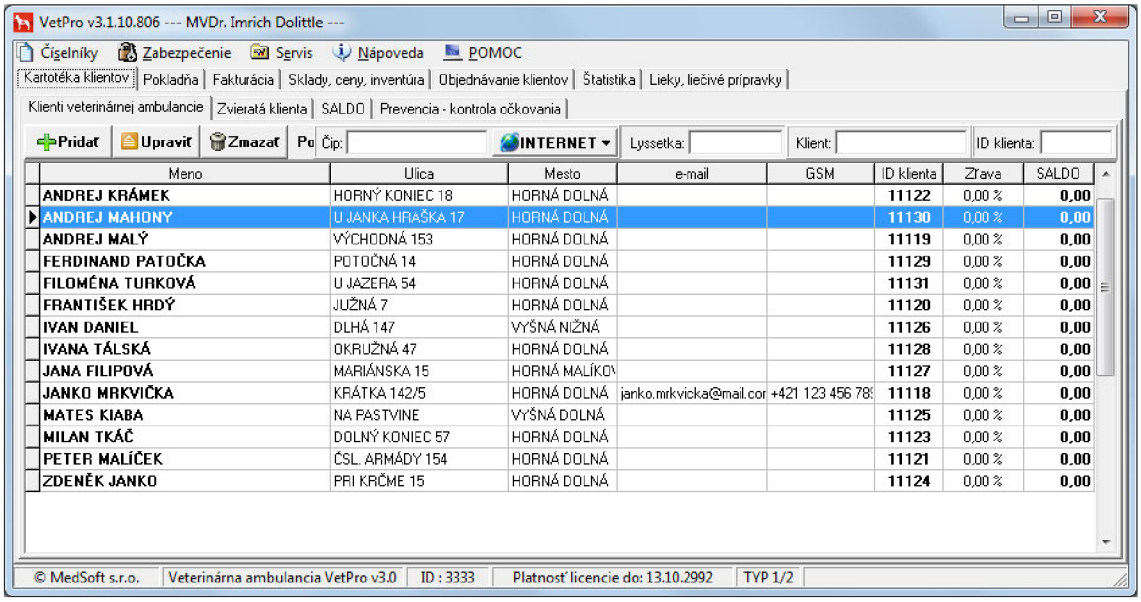
3.2.2 VetPro

VetPro je desktopovou aplikací vyvíjenou společností MedSoft s.r.o. Firma vznikla v roce 2011 na Slovensku a v roce 2015 se rozšířila do České republiky. Poslední aktualizace programu proběhla v roce 2020, těžko tedy říct, zda je program stále ve vývoji. Program je určen pro samostatné veterinární lékaře nebo malé ordinace bez více pracovišť, jelikož nenabízí žádnou možnost sdílení dat mezi více počítači. [5]

Z funkčního hlediska aplikace poskytuje kartotéku majitelů a zvířat. Navíc umožňuje klientům přiřadit věrnostní slevu nebo zobrazit dlužnou částku přímo v tabulce majitelů. Ke zvířeti je možné přidat denní záznam (návštěvu), ke kterému lze přiřadit diagnózu z číselníku. Nelze však tyto záznamy rozdělit na případy. Skladové hospodářství je přehledné, umožňuje rozdělení na více skladů, import skladových karet ze souboru nebo hlídání skladových zásob. Program rovněž nabízí velmi přehlednou správu účetních dokladů. Barevně jsou rozlišeny příjmové a výdajové doklady. Rovněž faktury jsou barevně rozlišeny na uhrazené a neuhrazené, současně jsou výrazně označeny faktury po splatnosti. Program nabízí i podporu pro odesílání SMS zpráv majitelům. Uživatelské rozhraní je jednoduché a lze se v něm rychle orientovat i bez předchozí znalosti. [6]

Program je nabízen formou pronájmu nebo plné licence. Pronájmem získá klient licenci na 1 rok, během platnosti této licence jsou aktualizace programu zdarma. Cena roční licence je 1 400 Kč. V případě zakoupení plné licence nejsou vyžadovány

žádné pravidelné nebo dodatečné poplatky. Licence je platná navždy a v ceně jsou i budoucí aktualizace. Cena je v tomto případě 4 850 Kč[1]. [7]



Obr. 3: Uživatelské rozhraní VetPro - kartotéka, převzato z [5]

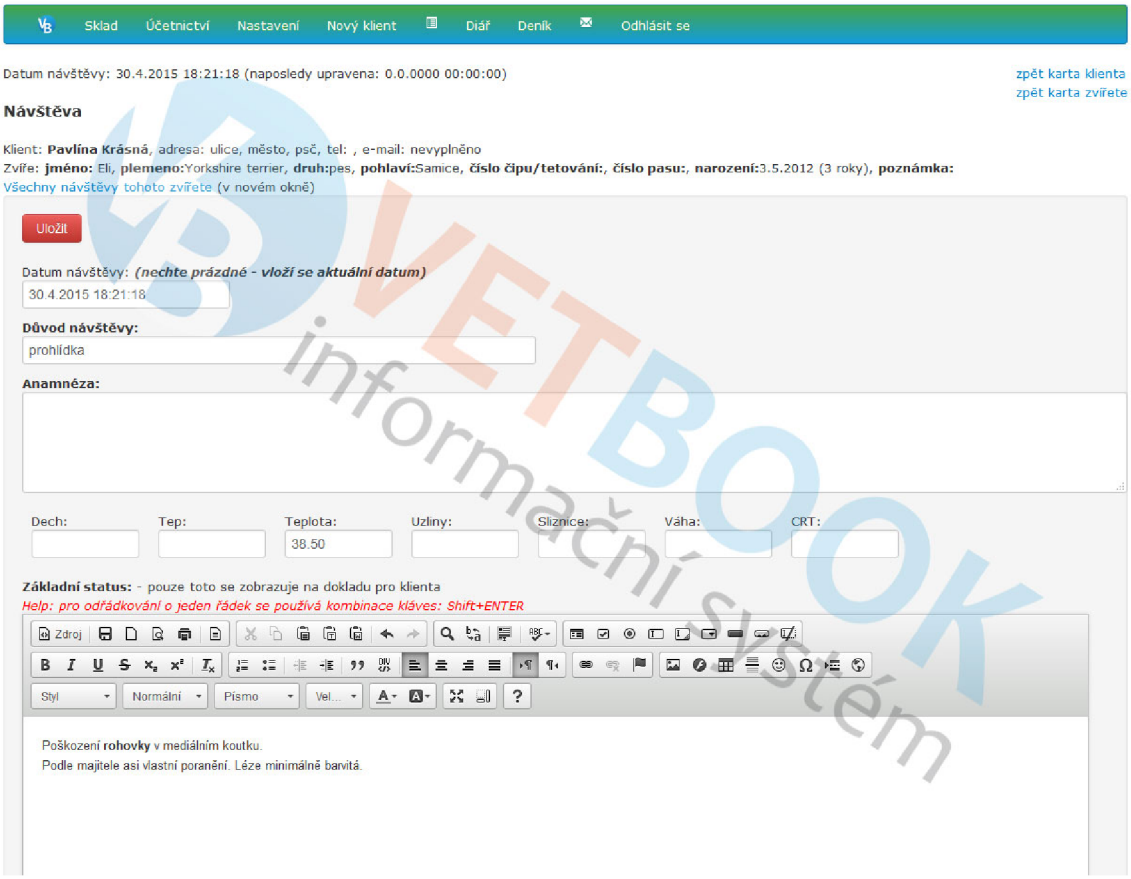
3.2.3 Vetbook

Informační systém Vetbook je webová aplikace, za kterou stojí Ing. Kamil Mrázek, Ph.D. Verze 1.0 byla vydána v roce 2012 a od vzniku je pravidelně aktualizována. Jelikož se jedná o webovou aplikaci, není závislá na hardwarové nebo systémové konfiguraci koncového zařízení. Je možné ji tedy obsluhovat například z tabletu nebo smartphonu. Aplikace je určená jak pro nezávislé veterináře nebo menší ordinace, tak i pro větší ambulance nebo kliniky. Vzhledem k tomu, že jde o webovou aplikaci, k datům se lze dostat z jakéhokoliv místa a zařízení, není tedy třeba řešit žádný vlastní server. [8]

Aplikace nabízí vzhledem k ostatním řešením na trhu nejvíce funkcí. Samozřejmostí je evidence majitelů, jejich zvířat a správa návštěv, ovšem bez možnosti rozdělení na případy (lze přidat důvod návštěvy). Rovněž nabízí i možnost rozdělení skladů nebo automatický import skladových položek ze souboru. V oblasti účetnictví zvládá vytváření dokladů a jejich evidenci, evidenci dlužníků nebo možnost slevy pro věrné zákazníky. Pro upozornění klienta na očkování lze využít informační SMS zprávy. Systém se snaží vyjít vstříc nejen veterinářům, ale i zákazníkům, proto nabízí objednávací kalendář, skrze nějž si klienti mohou sami rezervovat termín návštěvy. Další užitečnou funkcí pro urychlení odbavení nového zákazníka je možnost vyplnit osobní údaje klientem při čekání na vyšetření. Veterinář pak jen vyplněné údaje potvrdí a uloží. [8]

[1] Na oficiálních stránkách programu není uvedeno, zda jsou ceny s nebo bez DPH.

Vetbook nabízí pro jeho používání jediný tarif za cenu 500 Kč/měsíčně, respektive 6 000 Kč/ročně (ceny bez DPH). V ceně tarifu je 1 GB úložiště, neomezený počet uživatelů, aktualizace, pravidelné zálohy databáze, servis a nonstop uživatelská a technická podpora. V případě, že dojde k překročení úložiště nad 1 GB, je nutné za každý další 1 GB zaplatit 50 Kč bez DPH/měsíčně. V případě zájmu o rezervační systém je pak přidán poplatek 100 Kč/měsíčně bez DPH. [9]



Obr. 4: Uživatelské rozhraní Vetbook - detail návštěvy, převzato z [10]

3.2.4 Vetfox

Webová aplikace Vetfox je vyvíjená už dříve zmíněnou firmou Noviko s.r.o. a slouží jako modernější alternativa k desktopové aplikaci WinVet. Aplikace je na trhu od roku 2014 a je stále vyvíjena a vylepšována. Vzhledem k jejím funkcím je vhodná jak pro jednotlivé veterináře nebo malé ordinace tak i pro větší kliniky nebo ambulance. [11]

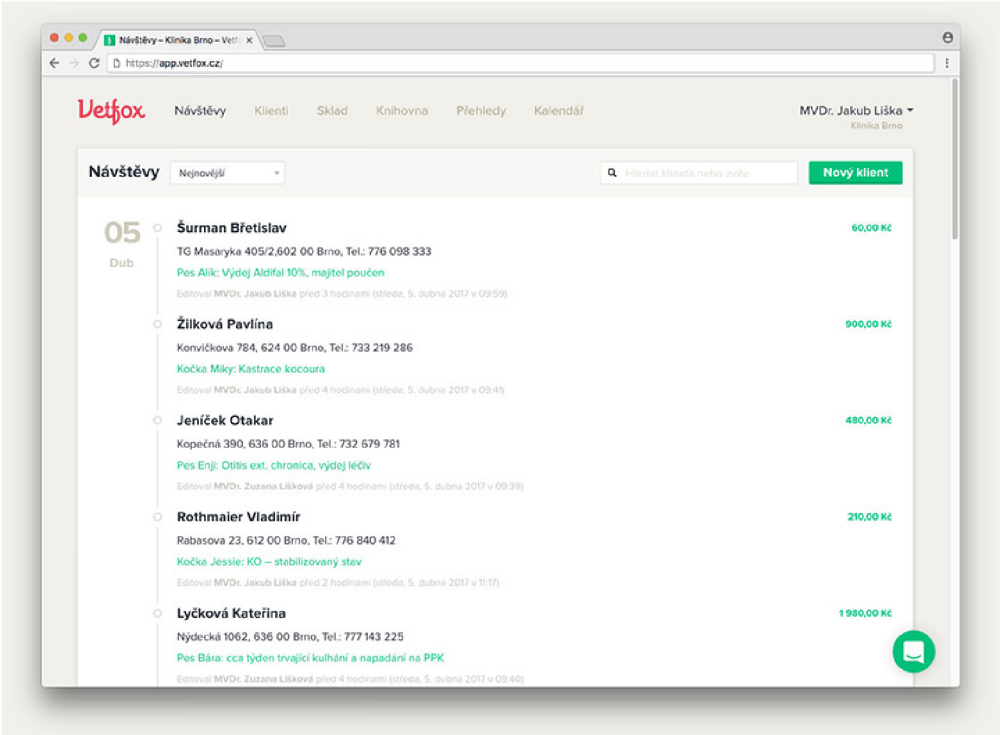
Aplikace je navržena s důrazem na co největší jednoduchost při zachování všech potřebných funkcí. Kartotéka je přehledně členěna. Návštěvy ovšem není možné rozdělovat na případy. Ve výpisu se však přehledně zobrazuje název návštěvy s použitými léky a úkony. Sklad lze rozdělit na libovolný počet oddělených skladů. Bohužel jsou součástí skladu i úkony. Číselníky nebo seznamy zcela v aplikaci chybí.

Je zde však možnost do skladu automaticky importovat položky ze souboru od dodavatele. Dále aplikace nabízí možnost ke každé návštěvě generovat fakturu a také možnost faktury hromadně exportovat. Součástí jsou také různé přehledy a statistiky, modul pro laboratorní výsledky nebo modul pro tvorbu marketingových kampaní. Stejně jako aplikace Vetbook tak i Vetfox nabízí rezervační systém pro snadnou rezervaci návštěvy. [11]

Vetfox nabízí pro provoz 4 různé tarify, které jsou funkčně nebo kapacitně omezeny. Jejich porovnání lze vidět v tabulce č. 1. V případě překročení hodnoty úložiště je nutné zakoupit extra úložiště o velikosti 10 GB za cenu 200 Kč bez DPH měsíčně. [12]

Tab. 1: Vetfox - porovnání tarifů [12]

	LITE	**PRO**	**PRO PLUS**	**PREMIUM**
Cena/rok bez DPH	3 480 Kč	5 880 Kč	10 680 Kč	15 480 Kč
Počet klientů	150	neomezeně	neomezeně	neomezeně
Počet návštěv/měsíc	200	750	750	neomezeně
Velikost úložiště	1 GB	1 GB	5 GB	10 GB
Kalendář	ne	ne	ano	ano
Laboratoř	ne	ne	ano	ano
Marketingové nástroje	ne	ne	ano	ano
Rozšířené přehledy	ne	ne	ne	ano



Obr. 5: Uživatelské rozhraní Vetfox - kartotéka, převzato z [11]

3.3 Vyhodnocení analýzy

Z popisu předností a nedostatků současného softwaru pro správu veterinární ordinace a z konzultace s klientem plynou pro nové řešení tyto požadavky:

- **Přehledné a moderní uživatelské rozhraní**
- **Jednoduchost obsluhy**
- **Možnost rozdělovat návštěvy na případy**
- **Připravenost pro provoz více pracovišť**
- **Generování faktur, jejich ukládání a export**
- **Přístup z mobilního telefonu nebo tabletu**

Co se týče přehlednosti a moderního uživatelského rozhraní, tak jde z velké části o subjektivní hodnocení. Přesto však lze říci, že z nabízených řešení je na tom nejlépe webová aplikace Vetfox, případně za ní v závěsu aplikace Vetbook a VetPro. Tento bod zároveň částečně souvisí i s jednoduchostí obsluhy, kde opět nejlépe tomuto požadavku vyhovuje Vetfox spolu s Vetbookem. VetPro ač je relativně přehledný, tak má velké množství voleb a nastavení, ve kterých se lze snadno ztratit. Co s týče těchto dvou bodů tak ani jednomu nevyhovuje aplikace WinVet, která je dle názoru klienta velmi nepřehledná a špatně se s ní pracuje.

Funkce rozdělování návštěv na případy je pro klienta velmi důležitá. I když některá řešení místo rozdělení nabízí alespoň nějaké označení návštěvy, není to plnohodnotné řešení, které klient požaduje. Tomuto požadavku odpovídá pouze program WinVet. Ostatní řešení mají pouze možnost přidat buď diagnózu pro odlišení, nebo návštěvu vhodně pojmenovat.

Jelikož klient nevylučuje budoucí rozšíření o další pracoviště, je nutné zahrnout do požadavků i připravenost řešení na tuto situaci. K tomu je potřeba mít možnost oddělených skladů, což nabízí všechna zmíněná řešení. Dále je také potřeba mít přístup ke společným datům z obou pracovišť. V případě programu WinVet je možné využít jeden PC jako samostatný server s databází. Pro klienta to ovšem není příliš vhodné řešení ať už z hlediska složitější konfigurace, zabezpečení stále dostupnosti, zálohám atd. V případě webových aplikací Vetbook a Vetfox již toto není potřeba řešit, protože jsou data dostupná odkudkoliv a obě platformy se zároveň starají o zálohy a nepřetržitý provoz.

Klient kromě práce v ordinaci zajišťuje i výjezdy za pacienty, proto dalším důležitým požadavkem je možnost přistupovat k systému přes mobilní telefon nebo tablet. Klient tak nemusí sebou za pacienty nosit notebook. Toto řešení bez problému splňují obě webové aplikace, jelikož k jejich obsluze stačí jakékoliv zařízení s internetovým prohlížečem.

Posledním požadavkem je schopnost systému generovat faktury z návštěv. Klient tak nemusí používat další software a fakturu může vytvořit ihned po vyplnění

návštěvy. Pro potřeby účetnictví je pak požadováno, aby faktury byly uschovány a systém umožňoval jejich export nebo stažení. Tento požadavek splňují všechny aplikace.

Ačkoliv pro klienta není cena za vhodné řešení tak důležitá, je užitečné porovnat i cenovou politiku jednotlivých řešení. Cenově nejlevněji vychází program VetPro s jeho neomezenou doživotní licencí a cenou 4 850 Kč (není specifikováno, zda cena zahrnuje DPH). Ovšem není zde jasný další vývoj aplikace. Velmi podobně jsou na tom program WinVet a webová aplikace Vetbook. Roční licence programu Winvet vyjde na 6 888 Kč, v případě Vetbooku na 6 000 Kč (obě ceny bez DPH). Vetbook však z pohledu klienta nabízí více užitečných funkcí a zároveň je možné k němu přistupovat odkudkoliv. U aplikace Vetfox je na výběr z více tarifů. Pro případy klienta by byla relevantní pouze nejvyšší verze tarifu, jelikož nižší tarify jsou velmi blízko hranici pro měsíční návštěvnost klienta. Nejvyšší tarif je ovšem nabízen za docela vysokou cenu 15 840 Kč bez DPH.

Ze srovnání požadavků a dostupných řešení vyplývá, že na tuzemském trhu není žádné řešení, které by splňovalo všechny požadavky klienta, a proto bylo přistoupeno k vytvoření řešení vlastního. A to konkrétně webové aplikace, která nejlépe splňuje požadavek na provoz více pracovišť současně a přístupu z mobilních zařízení.

4 TECHNOLOGIE PRO VÝVOJ WEBOVÝCH APLIKACÍ

4.1 Úvod

V současném rychle se měnícím digitálním světě se webové aplikace stávají zásadním nástrojem pro jakékoli podnikání i osobní využití. Od jednoduchých webových stránek a blogů až po komplexní obchodní aplikace a sociální sítě - webové aplikace nyní hrají klíčovou roli ve všech aspektech našeho života. Tato kapitola bude zaměřena na technologie, které umožňují vývoj těchto webových aplikací, jejich význam v současném digitálním světě a výhody, které přinášejí.

4.1.1 Definice webové aplikace

Webové aplikace představují zásadní a dynamickou součást internetového prostředí. Tyto aplikace jsou uloženy na serveru a přístupné prostřednictvím internetu. Uživatelé k nim přistupují pomocí webových prohlížečů a pro jejich tvorbu se používají stejné technologie jako pro webové stránky. Rozdíl mezi webovými aplikacemi a webovými stránkami spočívá v jejich funkci a úrovni interaktivity. Zatímco hlavním účelem webových stránek je poskytnout uživatelům informace, webové aplikace jsou navrženy tak, aby poskytovaly funkčnosti, které uživatelům umožňují dosahovat určitých cílů. Díky tomu se webové aplikace vyznačují vyšší úrovní interaktivity než tradiční webové stránky. [13]

Webové aplikace fungují na principu klient-server modelu. To znamená, že se skládají z klientské komponenty a alespoň jedné serverové komponenty. Serverová komponenta obvykle implementuje aplikační logiku, která využívá jednu nebo více databází pro načítání a ukládání dat. Aplikační logika může také komunikovat s dalšími serverovými komponentami. [14]

Vzhledem k modelu klient-server vyžadují webové aplikace skriptování na straně klienta (frontend) i na straně serveru (backend). Skriptování na straně klienta se zabývá prezentací informací a uživatelským rozhraním. Pro tuto část se využívají technologie založené na prohlížeči, jako je JavaScript, CSS a HTML. [15]

Na druhé straně, skriptování na straně serveru se zabývá výběrem obsahu, který má být vrácen jako odpověď na požadavek. Obvykle zahrnuje validaci požadavků nebo interakci s databází. Komunikace mezi komponentami webové aplikace probíhá prostřednictvím standardizovaných protokolů, jako je HTTP. [15]

4.1.2 Důležitost a výhody webových aplikací

Webové aplikace nabízí mnoho výhod oproti tradičním desktopovým aplikacím. Hlavní výhodou je, že je lze využívat z jakéhokoliv zařízení s připojením k inter-

netu, což eliminuje nutnost individuální instalace na každém zařízení. Tento přístup je často označován jako „software jako služba“ (SaaS). Příkladem je přechod na cloudové e-mailové platformy, které umožnili uživatelům přístup k e-mailům odkudkoliv, aniž by museli instalovat specifický desktopový e-mailový klient. [16]

V době, kdy se čím dál více pracovních procesů, obchodních operací a sociálních interakcí přesouvá do online prostoru, webové aplikace umožňují podnikům i jednotlivcům zůstat konkurenceschopnými a přizpůsobit se rychle se měnícímu digitálnímu prostředí. [16]

4.2 Historie vývoje webových aplikací

V současné době mají vývojáři webových aplikací k dispozici široký arzenál nástrojů, frameworků a knihoven, které výrazným způsobem zjednodušují vývoj. Je však důležité si uvědomit, že tomu tak nebylo vždy. Proto je dobré seznámit se s historií vývoje webových aplikací a webu pro lepší pochopení současného stavu a získat tak představu, jak se technologie v průběhu let inovovala.

Jako první milník můžeme označit rok 1989, kdy britský vědec Tim Berners-Lee během práce pro CERN vynalezl systém World Wide Web (WWW). Web byl původně koncipován a vyvinut pro automatizované sdílení informací mezi vědci na univerzitách a vědeckých ústavech po celém světě. [17]

Tento počáteční koncept se významně rozvinul a evoluce webových technologií může být v zásadě rozdělen do tří hlavních etap.

4.2.1 Web 1.0

Jde o první generaci World Wide Webu, o které se často hovoří jako o „webu jen pro čtení“ kvůli jeho omezené interaktivitě a funkčnosti. Web 1.0 představoval především jednosměrný přenos informací profesionálními autory a novináři na statických webových stránkách. Činnost uživatelů v té době spočívala především v prohlížení webových stránek za účelem získání informací. Jednalo se o pasivní konzumaci obsahu bez možnosti poskytovat zpětnou vazbu v podobě komentářů nebo jiného uživatelsky vytvářeného obsahu. Přes omezení této první generace, hrála významnou roli pro budoucí růst internetu zejména z hlediska rozšíření služeb pro poskytování internetového připojení nebo pro služby zabývající se hostováním webových stránek. Tato éra se také vyznačovala modelem zpoplatnění, kdy uživatelé museli platit za každou zobrazenou stránku. [18]

4.2.2 Web 2.0

První zmínka o Webu 2.0 byla zaznamenána v článku Fragmented future [19]. Termín se ovšem stal široce používány až po konferenci O'Reilly Media Web 2.0 v roce

2004. Na konferenci nebyl definován termín Web 2.0 jako takový, ale spíše se hovořilo o o změně přístupu k internetu a možnostem které nabízí. Přesto lze poměrně přesně vidět hlavní rozdíly mezi předchozí generací. [20]

„Web 1.0“ vs. „Web 2.0“

	WEB 1.0	WEB 2.0
OBSAH	obsah webu je vytváren převážne jeho vlastníkem	návštevníci se aktivne podílejí na tvorbe obsahu – vlastník je v roli moderátora
INTERAKCE	interakce vytvárí nároky na vlastníka, proto jen v nezbytné míre	interakce je vítána, má formu diskusí, chatu, propojení s messengery, sociálních profilu
AKTUALIZACE	odpovídá možnostem vlastníka	web je živý organismus – tvurcu obsahu mohou být miliony
KOMUNITA	neexistuje, návštevník je pasivní príjemce informací bez interakcí	návštevník je soucasne ten, „o kom web píše“, jednotlivec je soucástí rozsáhlé komunity
PERSONALIZACE	weby neumožnují implicitní personalizaci	umožnují vytváret a využívat sociální profily ctenáru

Obr. 6: Porovnání mezi Web 1.0 a Web 2.0, převzato z [21]

Web 2.0 tak měl dodat „druhý dech“ internetu a přinést revoluci pomocí chápání webu jako platformy. Hlavním záměrem bylo vytvoření aplikací, které budou s přibývajícím počtem uživatelů stále lepší. Často měli být vytvářené samotnými uživateli. Vzniklo tak mnoho sociálních platforem jako jsou Facebook, Twitter nebo YouTube, které umožňují uživatelům nahrávat obsah a získávat zpětnou vazbu. K popularitě přispělo i rozšíření mobilních zařízení. [18]

4.2.3 Web 3.0

Web 3.0, často označovaný jako „sémantický web“ nebo „decentralizovaný web“, je konceptem, který se zabývá posunem od současného stavu internetu, charakterizovaného dominantními korporacemi a centralizovanými platformami, k více demokratickému a rovnoměrně rozdělenému prostoru. [22]

Tato vize Webu 3.0 je založena na myšlence, že uživatelé budou moci plně vlastnit a kontrolovat svá data. To by umožnilo vytvořit prostředí, ve kterém by data byla chráněna a kde by uživatelé mohli mít větší kontrolu nad tím, jak jsou jejich data používána. [22]

Jedním z klíčových technologických prvků, které by mohly umožnit realizaci Webu 3.0, je blockchain. Blockchain technologie, známá hlavně díky kryptoměnám jako je Bitcoin a Ethereum, nabízí decentralizované a transparentní řešení pro uklá-

dání a přenos dat. Kromě blockchainu se také zkoumají další technologie jako jsou umělá inteligence, pokročilé algoritmy pro zpracování dat, a vývoj jazyků pro sémantický web, které by mohly přispět k realizaci této vize. [22]

4.3 Základní koncepty webových aplikací

4.3.1 Klient-server model

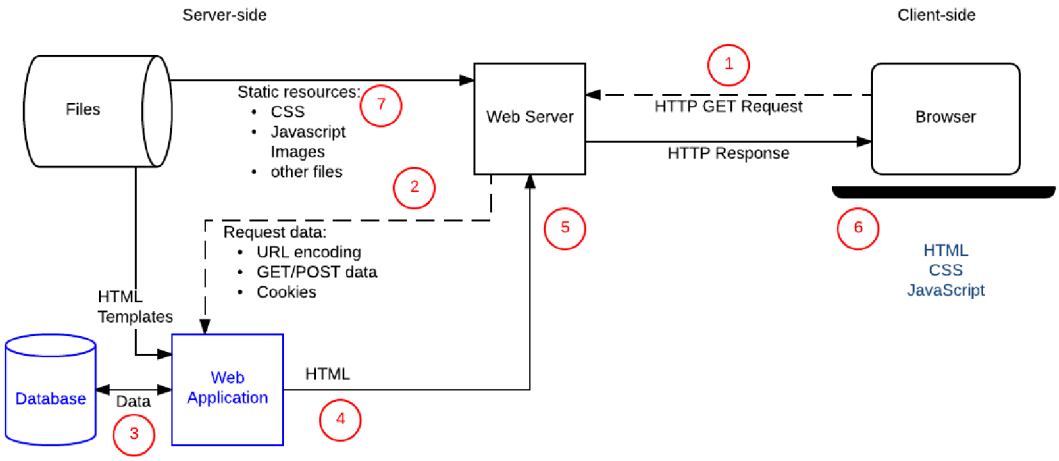
Klient-server model je základním paradigmatem používaným ve webových aplikacích. Tento model definuje komunikaci mezi dvěma počítačovými programy - klientem a serverem. [23]

Klient je program, který zasílá požadavek na server. V kontextu webových aplikací je klientem obvykle webový prohlížeč, který požaduje určitou webovou stránku nebo zdroj z webového serveru. Klient také může posílat požadavky na manipulaci s daty, například odeslání formuláře, nahrání souboru, nebo příspěvek na sociální síti. [23]

Server je program, který přijímá požadavky od klienta a zpracovává je. Server může zpracovat požadavek tím, že vrátí požadovaný zdroj (jako je webová stránka), provede požadovanou akci (jako je uložení dat do databáze) nebo vrátí odpověď oznamující výsledek akce. Server je obvykle hostován na fyzickém nebo virtuálním stroji, který je připojen k internetu a je schopen komunikovat s klienty. [23]

Komunikace mezi klientem a serverem probíhá pomocí protokolů. Nejběžnější protokol používaný v webových aplikacích je Hypertext Transfer Protocol (HTTP). Tento protokol definuje, jak se formátují a přenášejí požadavky a odpovědi mezi klientem a serverem. [24]

V rámci tohoto modelu může jeden server obsluhovat mnoho klientů a jeden klient může komunikovat s mnoha servery. To umožňuje komplexní interakce a funkcionalitu, kterou vidíme v moderních webových aplikacích. [24]



Obr. 7: Diagram modelu klient - server, převzato z [14]

Výše uvedený diagram zobrazuje základní strukturu webové aplikace s modelem klient-server. Klient je v tomto případě webový prohlížeč, který zobrazuje uživateli webovou stránku. Server je počítač, který poskytuje data pro webovou stránku. Klient a server spolu komunikují pomocí HTTP. Dle [14] lze celý proces popsat následovně:

1. Webový prohlížeč vytvoří požadavek HTTP GET na server pomocí URL adresy, kde jsou zakódovány parametry.
2. Webový server zjistí, že se jedná o dynamický požadavek, a předá jej webové aplikaci ke zpracování.
3. Webová aplikace zjistí z URL adresy požadované parametry a získá je z databáze.
4. Webová aplikace dynamicky vytvoří HTML stránku a vloží do ní získaná data z databáze.
5. Webová aplikace vrátí vygenerovanou HTML stránku webovému prohlížeči spolu se stavovým kódem HTTP 200 (úspěšný požadavek).
6. Webový prohlížeč začne zpracovávat vrácenou HTML stránku a odešle požadavky na získání dalších souborů (CSS, JavaScript).
7. Webový server načte statické soubory a vrací je přímo prohlížeči.

4.3.2 Protokoly HTTP a HTTPS

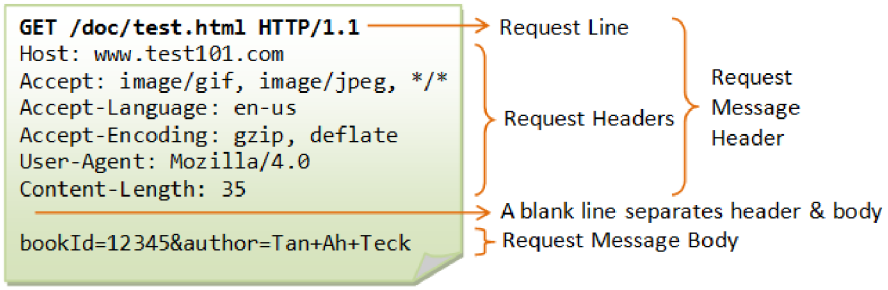
Hypertext Transfer Protocol (HTTP) je protokol aplikační vrstvy, který používají webové prohlížeče a webové servery k přenosu souborů, například textových a grafických. Jedná se o protokol klient-server. Klient (obvykle webový prohlížeč) si vyžádá zdroj (webovou stránku) od webového serveru. Webový server odpoví požadovanou

webovou stránkou. Ke komunikaci mezi klientem a serverem se standardně používá spojení TCP (Transmission Control Protocol). [25]

Dle [26], protokol HTTP používá k provádění různých úkolů specifické metody požadavků:

- **GET** - Metoda GET slouží k získání informací z daného serveru pomocí zadané URL adresy. Požadavky používající metodu GET by měly pouze načítat data a neměly by mít žádný jiný vliv na data.
- **HEAD** - Metoda HEAD požaduje odpověď stejnou jako u požadavku GET, ale bez těla odpovědi.
- **POST** - Požadavek POST se používá k odeslání dat na server, například informací o zákazníkovi, nahrání souboru atd. pomocí formulářů HTML.
- **PUT** - Nahradí všechny aktuální reprezentace cílového prostředku nahraným obsahem.
- **DELETE** - Metoda DELETE odstraní zadaný prostředek.
- **CONNECT** - Vytvoří tunel k serveru identifikovanému danou URL adresou.
- **OPTIONS** - Metoda OPTIONS popisuje možnosti komunikace pro cílový prostředek.
- **TRACE** - Metoda TRACE provede test smyčky zpráv podél cesty k cílovému prostředku.
- **PATCH** - Metoda PATCH aplikuje na prostředek částečné změny.

Součástí HTTP požadavku bývá stavový řádek, hlavička, případně tělo. Stavový řádek obsahuje metodu požadavku a verzi HTTP protokolu. Hlavička obsahuje dodatečné informace o klientovi a v těle jsou pak umístěna data odesílaná na server. [27]

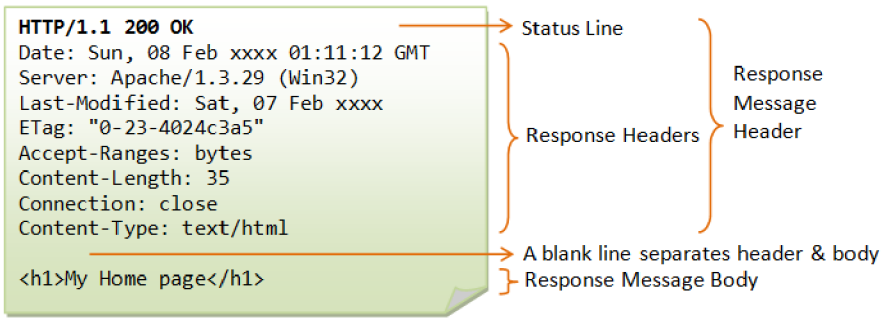


Obr. 8: Ukázka HTTP požadavku, převzato z [28]

Jako reakce serveru na požadavek klienta, server odesílá odpověď, která se skládá ze stavového řádku, hlavičky a těla. Stavový řádek obsahuje verzi protokolu a stavový kód označující výsledek požadavku. V hlavičce jsou další informace o odpovědi, například typ obsahu nebo datum a čas odeslání odpovědi. Tělo obsahuje data odpovědi, například HTML stránku. [27]

Některé nejdůležitější stavové kódy jsou vypsány zde:

- **100 Continue** - Klient může pokračovat v požadavku nebo ho ignorovat, pokud je již hotov. [29]
- **101 Switching Protocols** - Označuje protokol, na který server přepíná. [30]
- **200 OK** - Požadavek byl úspěšný. Odpověď 200 je ve výchozím nastavení uložena v mezipaměti. [31]
- **201 Created** - Požadavek byl úspěšný a byl vytvořen nový prostředek. [32]
- **202 Accepted** - Požadavek byl přijat ke zpracování, ale zpracování ještě nebylo dokončeno. [33]
- **203 Non-Authoritative Information** - Požadavek byl úspěšný, ale vrácené informace mohou pocházet z jiného zdroje. [34]
- **301 Moved Permanently** - Požadovaný zdroj byl trvale přesunut na novou URL. [35]
- **404 Not Found** - Server nemohl najít požadovaný zdroj. [36]
- **500 Internal Server Error** - Na serveru došlo k neočekávané chybě a požadavek nemohl být dokončen. [37]



Obr. 9: Ukázka HTTP odpovědi, převzato z [28]

HTTPS (Hypertext Transfer Protocol Secure) je rozšířením HTTP, které zajišťuje bezpečný přenos dat mezi klientem a serverem. HTTPS využívá šifrování dat pomocí SSL (Secure Sockets Layer) nebo TLS (Transport Layer Security) protokolů, které zajišťují, že data přenášená mezi klientem a serverem jsou chráněna před odposlechem, modifikací nebo falšováním. V praxi to znamená, že když webový prohlížeč (klient) pošle požadavek na webový server přes HTTPS, požadavek a odpověď jsou zašifrovány. To zvyšuje bezpečnost, protože i kdyby byla data při přenosu odchycena, útočník by je nemohl přečíst ani změnit. [38]

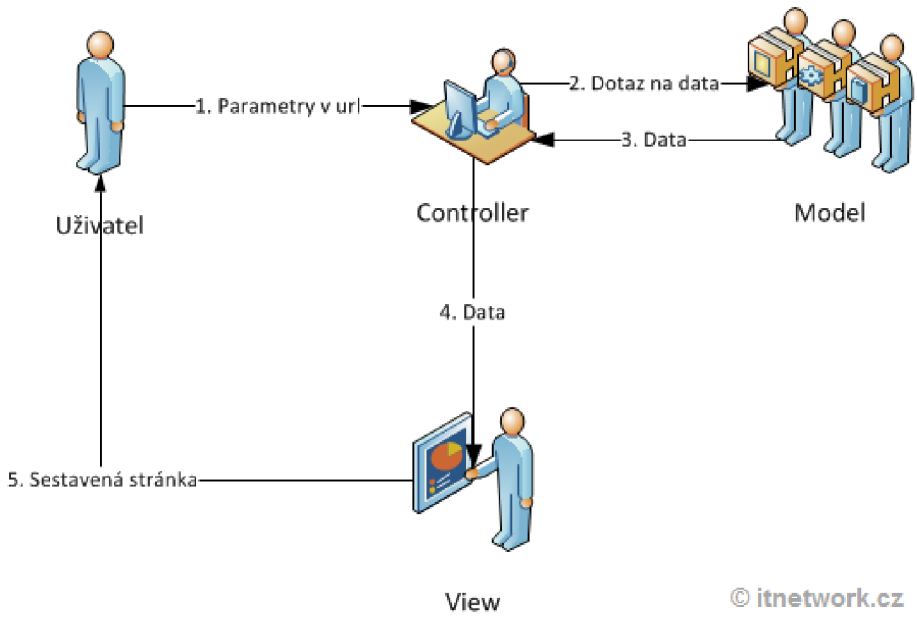
4.3.3 MVC architektura

MVC (Model-View-Controller) je architektonický vzor, který se prosadil zejména v oblasti webových aplikací. Původně vznikl pro desktopové aplikace, ale nyní je integrální součástí populárních webových frameworků, jako jsou Zend, Nette, Django,

Ruby On Rails nebo ASP.NET MVC. Hlavní motivací pro využití MVC je oddělení logiky od výstupu, tedy řešení problému „špagetového kódu“, kdy se logické operace a renderování výstupu mísí dohromady. Aplikace využívající MVC je rozdělena na tři komponenty: Modely, View (pohledy) a Controllery (kontrolery). [39]

Model obsahuje logiku aplikace, včetně výpočtů, databázových dotazů a validací. Model neví nic o výstupu, jeho úkolem je pouze přijmout parametry a vrátit data. View (pohledy) se starají o zobrazení výstupu uživateli. Pohledy jsou často implementovány pomocí šablon, které obsahují HTML a značkovací jazyk pro vkládání proměnných a provádění iterací a podmínek. Controller (kontroler) je prostředník, který komunikuje s uživatelem, modelem i pohledem. Drží celý systém pohromadě a propojuje jednotlivé komponenty. [39]

V praxi se životní cyklus stránky začíná zadáním URL adresy uživatelem. Controller na základě parametrů URL zjistí, co má udělat, zavolá odpovídající model, aby získal potřebná data, a poté předá tato data do pohledu, který je zobrazí uživatelem. [39]



Obr. 10: Diagram životního cyklu stránky, převzato z [39]

4.4 Frontend technologie

Frontend technologie tvoří základ toho, co vidíme a s čím interagujeme na internetu. Jde o kombinaci technologických nástrojů a programovacích jazyků, jako jsou HTML, CSS a JavaScript, které společně vytvářejí vizuální a funkční aspekty webových a mobilních aplikací. Těmito technologiemi se vytváří uživatelské rozhraní, které nám umožňuje prohlížet obsah, vyplňovat formuláře, hrát hry nebo nakupo-

vat online. Bez frontend technologií by internetové stránky a aplikace byly statické, neinteraktivní a obtížně použitelné. Frontend technologie tedy představují zásadní součást moderního digitálního světa, umožňující plynulou a uživatelsky přívětivou interakci s digitálními produkty a službami.

4.4.1 HTML

HTML, což je zkratka pro HyperText Markup Language, je základním stavebním kamenem všech webových stránek. Jde o standardní značkovací jazyk používaný pro tvorbu webových stránek. HTML definuje strukturu a layout webových stránek pomocí různých prvků a atributů. HTML dokument je tvořen řadou takzvaných „elementů“ nebo „tagů“, které určují, jak se má obsah webové stránky interpretovat a zobrazovat ve webovém prohlížeči. Tyto tagy jsou ve většině případů párové a obsah se vkládá mezi ně. Tyto elementy mohou zahrnovat nadpisy, odstavce, seznamy, obrázky, odkazy, tlačítka a mnoho dalšího. [40]

Obr. 11 zobrazuje základní strukturu HTML dokumentu. Význam jednotlivých elementů je následující:

- **<!DOCTYPE html>** - Tento tag oznamuje prohlížeči verzi HTML (v tomto případě HTML5.
- **<html>** - Tento tag označuje začátek a konec HTML dokumentu. Všechny ostatní HTML tagy musí být umístěny mezi otevíracím a uzavíracím tagem <html>.
- **<head>** - Tento tag obsahuje metadatové informace o webové stránce, které nejsou zobrazeny na samotné stránce. Může obsahovat například odkazy na CSS styly, definice znakové sady, meta informace pro vyhledávače a další.
- **<meta>** - Meta tagy poskytují další metadatové informace o HTML dokumentu, jako je kódování, popis stránky, klíčová slova pro vyhledávače, autor a další.
- **<title>** - Tento tag definuje název webové stránky. Název se zobrazuje v záhlaví prohlížeče a je také používán vyhledávači při indexaci stránky.
- **<body>** - Tento tag obklopuje hlavní obsah webové stránky, který je viditelný pro uživatele. Může obsahovat text, obrázky, odkazy, tabulky, formuláře a další.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta ... >
        <title>Titulek stranky</title>
    </head>
    <body>
        Obsah stranky
    </body>
</html>
```

Obr. 11: Struktura HTML dokumentu

Jedním z klíčových aspektů HTML je jeho schopnost vytvářet odkazy pomocí takzvaných „hypertextových odkazů“. Tyto odkazy mohou odkazovat na jiné části stejné stránky, na jiné stránky na stejném webu, nebo rovnou na jiné weby. HTML jazyk obsahuje velké množství dalších tagů, které slouží k definování různých typů obsahu na webové stránce. [41]

4.4.2 CSS

Cascading Style Sheets (CSS) je jazyk pro popis vzhledu a formátování dokumentu napsaného v HTML nebo XML (včetně různých XML jazyků jako SVG, MathML nebo XHTML). CSS popisuje, jak by měly být elementy zobrazeny na obrazovce, na papíře, v mluveném slově nebo na jiných médiích. CSS je klíčovým nástrojem pro vytváření webových stránek a aplikací. Jeho hlavní výhodou je schopnost oddělit prezentaci webové stránky od jejího obsahu. To znamená, že můžete měnit vzhled stránky bez zásahu do samotného HTML kódu, což vede k efektivnější správě a údržbě stránky. Zároveň se díky případnému problému s CSS nezmění význam obsahu, pouze jeho vzhled. Díky CSS je možné optimalizovat vzhled stránky pro různá zařízení. [40]

Příklad zápisu v CSS lze vidět na obr. 12, kde se mění barva textu elementu *h1*, který reprezentuje nadpis a barva textu elementu *p*, který reprezentuje odstavec.

```css
h1 {
    color: blue;
}

p {
    color: red;
}
```

Obr. 12: Příklad zápis v CSS

CSS Frameworky

CSS (Cascading Style Sheets) frameworky jsou knihovny předdefinovaných stylů, které mohou vývojáři webových stránek využít k rychlému a efektivnímu designu. Tyto frameworky často obsahují širokou škálu komponent, jako jsou tlačítka, formuláře, navigační prvky, typografie atd., které lze snadno implementovat do projektu. Frameworky mohou také zahrnovat responzivní styly, které automaticky upravují rozložení stránky na základě velikosti obrazovky uživatele, což usnadňuje vývoj na mobilní zařízení. [42]

Existuje mnoho CSS frameworků, z nichž každý má své specifické výhody. Pro příklad jsou níže vypsány některé z nejpoužívanějších:

- **Bootstrap** - Bootstrap je nejpoužívanější CSS framework, který byl původně vyvinut týmem v Twitteru. Bootstrap je komplexní sada nástrojů pro tvorbu webových stránek a webových aplikací. Nabízí širokou škálu předem definovaných CSS tříd, komponent a JavaScriptu, které usnadňují rychlý vývoj responsivních webových stránek. [43]
- **Foundation** - Foundation, vytvořený společností Zurb, je flexibilní a pokročilý framework, který je široce používán pro tvorbu responsivních webových stránek. Nabízí mnoho nástrojů a komponent pro designéry a vývojáře. Foundation je také známý svou podporou pro přístupnost a sémantiku. [44]
- **Bulma** - Bulma je relativně nový, ale rychle se rozvíjející CSS framework založený na Flexboxu. Je velmi oblíbený pro svou jednoduchost, čitelnost a přizpůsobitelnost. Bulma je také 100% responsivní, což znamená, že design vašeho webu se automaticky přizpůsobí jakékoliv velikosti obrazovky. [45]
- **Tailwind CSS** - Tailwind je framework nízké úrovně, který umožňuje tvořit jedinečné uživatelské rozhraní „od základu“. Místo předdefinovaných komponent poskytuje Tailwind sadu nástrojů pro vytváření vlastních designů pomocí utility-first přístupu. To znamená, že místo toho, abyste definovali globální styly a pak je upravovali pro jednotlivé komponenty, definujete styly přímo pro danou komponentu. [46]
- **Flowbite** - Flowbite je moderní CSS a JavaScript framework, který je postaven na základech utility-first frameworku Tailwind CSS. Využívá flexibilitu a přizpůsobitelnost Tailwindu a rozšiřuje jej o předem definované komponenty, které usnadňují a zrychlují proces vývoje. Flowbite nabízí sadu nástrojů pro vytváření responzivních, přizpůsobitelných a přístupných webových stránek a aplikací. Jeho hlavní předností je kombinace flexibility utility-first přístupu a pohodlí předdefinovaných komponent. [47]

4.4.3 JavaScript

JavaScript je skriptovací jazyk vytvořený v roce 1995 firmou Netscape Communications jako součást jejich webového prohlížeče Netscape Navigator. I když sdílí název a některé syntaxe s programovacím jazykem Java, JavaScript je zcela nezávislý jazyk s vlastní specifikací. JavaScript je známý především pro své použití na webových stránkách. [48]

Všechny moderní webové prohlížeče podporují JavaScript bez potřeby jakýchkoli pluginů. JavaScript je zodpovědný za mnoho interaktivních prvků na webových stránkách, jako jsou tlačítka, která reagují na kliknutí, automaticky se aktualizující obsah, animace a mnoho dalších. [49]

```
<script>
    document.write('Hello, World!');
</script>
```

Obr. 13: Příklad kódu v JavaScriptu

Navzdory svému názvu a původnímu zaměření na webové prohlížeče se JavaScript stal univerzálním jazykem, který je možné použít v mnoha různých kontextech. S vývojem technologií jako Node.js je JavaScript nyní možné používat také na straně serveru, což z něj činí plně funkční backendový jazyk. [50]

JavaScript frontend frameworky

JavaScript frameworky jsou nástroje, které vývojářům poskytují strukturovaný způsob vytváření aplikací. Tyto frameworky poskytují šablony a knihovny kódu, které usnadňují a zrychlují vývoj tím, že poskytují již vytvořené komponenty pro běžné úkoly. Frameworky také často poskytují způsob, jak organizovat a strukturovat kód, což vede k čistším a snadněji udržitelným aplikacím. [51]

Pro usnadnění vývoje JavaScriptu existuje celá řada frameworků, mezi nejpopulárnější patří následující:

- **React.js** - React.js je JavaScriptový framework vytvořený a udržovaný společností Facebook. Je to jedno z nejpoužívanějších řešení pro vývoj uživatelských rozhraní, zvláště pro jednostránkové aplikace. React se zaměřuje na vytváření komponent, které mohou být znovu použity a kombinovány pro vytvoření komplexních uživatelských rozhraní. [52]
- **Angular.js** - Angular.js je open-source framework pro vývoj webových aplikací, který byl vytvořený společností Google. Angular je plně vybavený framework, který zahrnuje nástroje pro vše od vytváření uživatelských rozhraní až po práci s daty na straně klienta a serveru. Angular používá TypeScript, což

je nadstavba nad JavaScriptem přidávající statické typování a další pokročilé funkce. [53]

- **Vue.js** - Vue.js je další populární JavaScriptový framework pro vývoj uživatelských rozhraní. Vue je známý svou jednoduchostí a flexibilitou, což ho činí přístupným pro nové vývojáře, zatímco je stále dostatečně silný pro vývoj složitých aplikací. Vue také podporuje systém komponent podobný tomu v Reactu. [54]

4.5 Backend technologie

Backend technologie jsou další podstatnou částí webových aplikací. Představují soubor serverových technologií, databází a programovacích jazyků, které zpracovávají požadavky od frontendu, ukládají data a zajišťují jejich správnou distribuci. Při interakci s aplikací, například při vyplnění formuláře, backend přijme data, uloží je do databáze a provede požadované akce, jako je například odeslání potvrzovacího e-mailu. Tedy zatímco frontend se stará o uživatelské rozhraní a interakci, backend zajišťuje práci s daty, která je před uživateli skryta. Společně tvoří základ moderních webových aplikací. Mnoho backend frameworků jsou však tzv. „full-stack“ frameworky. To znamená, že kromě backend části implementují i frontend část.

4.5.1 Jazyky pro backend vývoj a jejich frameworky

Pro vývoj backendu existuje celé řada jazyků a frameworků pro práci s nimi. Volba správného jazyka a frameworku pro konkrétní projekt může být náročná. Neexistuje univerzální odpověď, protože rozhodnutí závisí na řadě aspektů: typu a velikosti projektu, požadavků na výkon, zkušenosti a preference vývojářů, komunitě a podpoře, dokumentaci atd. V následující části jsou vyjmenovány některé z nejpoužívanějších jazyků a jejich frameworků.

Python

Python je jedním z nejpopulárnějších programovacích jazyků díky své jednoduchosti, čitelnosti a flexibilitě. Tento interpretovaný jazyk je ideální pro rychlý vývoj a prototypování, ale je také dostatečně výkonný pro náročné aplikace, což ho činí oblíbeným jak mezi začátečníky, tak mezi zkušenými programátory. [55]

Jedním z klíčových rysů Pythonu je jeho rozsáhlá standardní knihovna a bohatý ekosystém balíčků třetích stran, které poskytují funkce pro řadu úkolů - od webového vývoje po strojové učení. Jeho podpora více programovacích paradigmat také umožňuje programátorům volit nejvhodnější přístup k řešení specifických problémů. [56]

Mezi nejznámější a nejpoužívanější frameworky pro vývoj webových aplikací v Pythonu patří tyto:

- **Django** - Django je vysoce robustní a komplexní webový framework pro Python. Jeho filozofií je „bateriemi vybavený“ přístup, což znamená, že přichází s širokou škálou funkcí předinstalovaných a připravených přímo k použití. Django podporuje MVC (Model-View-Controller) architekturu[1] a je navržen tak, aby podporoval rychlý vývoj a čistý, opakovatelný kód. Je ideální jak pro menší aplikace s potřebou rychlého vývoje, tak i pro velké a složité webové aplikace, které vyžadují hlubokou funkcionalitu a robustnost. [58]
- **Flask** - Na druhé straně spektra je Flask, což je mikroframework, který se zaměřuje na minimalismus a jednoduchost. Flask neobsahuje mnoho funkcí „out of the box“ jako Django, ale jeho lehkost a flexibilita umožňuje vývojářům mít více kontroly nad jejich aplikací a její funkcionalitou. Flask je ideální pro menší aplikace nebo projekty, kde je potřeba větší míra přizpůsobení. [59]
- **FastAPI** - FastAPI je relativně nový webový framework, který kombinuje nejlepší aspekty Django a Flasku. Je to vysoký výkon, snadno použitelný, rychlý k nasazení a napsaný s ohledem na moderní standardy a techniky. FastAPI je založený na standardních Python type hints[2], podporuje asynchronní request handling a automatické vytváření API dokumentace. Je ideální pro vývoj moderních webových API. [61]

Java

Java je objektově orientovaný programovací jazyk, který je ceněn pro svou přenositelnost a univerzálnost. Tento kompilovaný jazyk je základem pro mnoho korporátních a mobilních aplikací, zejména pro Android. Java je navržena s principem „napsáno jednou, spuštěno kdekoli“, což znamená, že její kód může být spuštěn na jakémkoli zařízení s Java Virtual Machine (JVM). To je výhodné pro vývoj cross-platform aplikací. [62]

Java také podporuje silné typování a objektově orientované programování, což napomáhá vytváření čistého a udržitelného kódu. Její rozsáhlá standardní knihovna a široké spektrum nástrojů třetích stran usnadňují vývoj složitých aplikací. [62]

Populární frameworky pro vývoj webových aplikací v jazyku Java jsou následující:

- **Spring Boot** - Spring Boot je členem široké rodiny Spring Framework projektů. Je to open-source Java framework, který se zaměřuje na zjednodušení

[1] Ve skutečnosti jde o MVT (Model-View-Template) architekturu. Rozdíl je v pojmenování jednotlivých komponent. View v MVT = Controller v MVC, Template v MVT = View v MVC. Funkce modelu je stejná. [57]

[2] Type hints (typové anotace) jsou vlastností jazyka Python, která umožňuje explicitně specifikovat očekávaný typ proměnné, funkčního parametru nebo návratové hodnoty funkce. FastAPI využívá tyto typové anotace k automatické validaci dat, konverzi dat (např. do JSON) a vytváření dokumentace API. [60]

vývoje stand-alone, produkčně připravených Spring aplikací. Poskytuje předdefinovanou šablonu pro vývoj webových aplikací, která eliminuje potřebu mnoha konfiguračních kroků typicky spojených s tradičním Springem. Také automaticky spravuje všechny závislosti a poskytuje zabudované funkce pro rychlý vývoj, jako je vložený server, bezpečnost, metriky a jiné. [63]

- **Struts** - Struts je robustní open-source framework pro vývoj webových aplikací v Javě. Je založen na model-view-controller (MVC) architektuře, která odděluje logiku aplikace od prezentační vrstvy, což usnadňuje správu a údržbu kódu. Struts poskytuje bohatou sadu tagů, které mohou být použity v JSP (JavaServer Pages), což zjednodušuje vývoj uživatelských rozhraní. Tento framework také podporuje interní validaci formulářů a mezinárodní podporu pro vícejazyčné aplikace. [64]

JavaScript

JavaScript, původně navržený pro webové prohlížeče, se díky technologii Node.js stal důležitým jazykem i pro backend. Node.js je běhové prostředí, které umožňuje spouštění JavaScriptu na serveru. [50]

JavaScript je dynamicky typovaný jazyk, což znamená, že proměnné mohou měnit svůj datový typ během provádění kódu. Tato flexibilita může zvýšit produktivitu, ale také zvyšuje riziko chyb. [65]

Díky své non-blocking I/O architektuře je Node.js vhodný pro aplikace s vysokou úrovní konkurencí, jako jsou real-time aplikace nebo chaty. Navíc, použitím stejného jazyka na frontendu i backendu můžete snížit složitost vývoje a zlepšit soudržnost kódu. [66]

V současné době je JavaScript jako jazyk pro backend velmi populární zejména pro full-stack vývojáře, kteří tak mohou používat stejný jazyk jak pro frontend tak pro backend. Mezi nejoblíbenější frameworky patří tyto:

- **Express.js** - Express.js je minimalistický a flexibilní webový framework pro Node.js, který je oblíbený pro svou jednoduchost a rychlost. Express.js zjednodušuje vývoj serverové části aplikace tím, že nabízí sadu middleware funkcí pro zpracování HTTP požadavků a snadnou definici tras (routes). I přes svou minimalistickou povahu je Express.js velmi rozšiřitelný a lze ho snadno doplnit o další knihovny a middleware pro rozšíření funkčnosti. [67]
- **Nest.js** - Jde o výkonný a progresivní framework pro vytváření serverových aplikací, který se opírá o moderní JavaScript, konkrétně TypeScript. Nest.js nabízí soudržnou a extenzibilní architekturu pro efektivní vytváření škálovatelných a udržitelných aplikací. Nest.js je framework postavený na Express.js HTTP serveru, ale navíc přidává další vrstvu abstrakce a řadu dalších funkcí, které umožňují vývojářům vytvářet komplexnější a dobře strukturované aplikace. [68]

PHP

PHP je otevřený skriptovací jazyk, který je široce používán pro vývoj webových aplikací. Od svého vzniku v roce 1994 se stal jedním z nejpopulárnějších jazyků pro webový vývoj. PHP skripty jsou obvykle vložené do HTML kódu a interpretovány na serveru, což umožňuje dynamické generování webových stránek. [69]

PHP je známý svou jednoduchostí a širokou podporou. Mnoho webových hostingů poskytuje vestavěnou podporu pro PHP, což činí jeho použití snadným pro začínající vývojáře. Zároveň PHP poskytuje robustní sadu funkcí pro pokročilé vývojáře, včetně podpory pro mnoho databázových systémů a protokolů. [69]

Pro PHP existuje celá řada velmi populárních frameworků, mezi nejznámější patří:

- **Laravel** - Laravel je jeden z nejpopulárnějších PHP frameworků, který značně zjednodušuje vývoj webových aplikací díky široké škále nástrojů a funkcí. Laravel přichází s vlastním ORM (Object-Relational Mapping) nástrojem nazvaným Eloquent, který zjednodušuje práci s databázemi. Laravel je také známý pro svůj systém routování, šablonovací jazyk Blade a robustní ekosystém balíčků pro rozšíření jeho funkcionalit. Laravel je ideální pro vývoj moderních, škálovatelných webových aplikací. [70]
- **Symfony** - Symfony je silný a flexibilní PHP framework, který se hodí pro vývoj široké škály webových aplikací, od jednoduchých webů po složité webové služby. Symfony je modulární v designu, což umožňuje vývojářům používat jen ty komponenty, které potřebují pro svůj projekt. Symfony je také základem pro mnoho dalších PHP frameworků, včetně Laravelu. Je známý svou robustností, škálovatelností a dlouhodobou podporou. [71]
- **Nette** - Nette je výkonný PHP framework, který klade velký důraz na bezpečnost a čistotu kódu. Nette představuje řadu inovativních funkcí, jako je například systém pro bezpečné vytváření webových stránek zvaný Latte. Nette také poskytuje pokročilé nástroje pro práci s formuláři a jeho modulární architektura umožňuje snadné rozšiřování. Nette je oblíbený zejména mezi vývojáři v střední Evropě. [72]

Ruby

Ruby je dynamický, interpretovaný programovací jazyk, který klade důraz na jednoduchost a produktivitu. Jeho design je založen na filozofii, že programování by mělo být zábavou a že efektivní kód by měl být nejen funkční, ale také elegantní. Ruby podporuje několik paradigmat programování, včetně objektově orientovaného, procedurálního a funkcionálního stylu. Syntaxe Ruby je čistá a přímočará, s pravidly, která umožňují vývojářům psát kód, který je snadno čitelný a srozumitelný. Tento jazyk také obsahuje dynamické typování a garbage collector. Ruby má silnou

standardní knihovnu a bohatý ekosystém open-source knihoven, které zvyšují jeho flexibilitu a použitelnost v široké škále aplikací. [73]

Pro jazyk Ruby existuje několik frameworků, které se zabývají backend vývojem, nejpopulárnější jsou:

- **Ruby on Rails** - Ruby on Rails, často zkracovaný jako Rails, je jeden z nejznámějších webových frameworků napsaných v Ruby. Rails je postaven na filozofii „Convention over Configuration“ (konvence nad konfigurací), což znamená, že vývojáři mohou snadno vytvářet aplikace bez nutnosti psaní zbytečného kódu. Rails také využívá architekturu Model-View-Controller (MVC), která pomáhá udržovat kód organizovaný a snadno udržitelný. Rails je ideální pro rychlý vývoj komplexních webových aplikací. [74]
- **Sinatra** - Sinatra je lehký a flexibilní webový framework pro Ruby, který se hodí pro vývoj jednoduchých webových aplikací nebo API. Místo toho, aby se snažil poskytnout kompletní řešení jako Rails, Sinatra dává vývojářům více kontroly nad tím, jak je aplikace postavena a jak funguje. V Sinatra je routování velmi jednoduché a přímočaré, což umožňuje rychlý vývoj a nasazení webových aplikací. [75]
- **Hanami** - Hanami je moderní, lehký webový framework pro Ruby, který klade důraz na čistý kód, výkon a udržitelnost. Hanami je postaven s architekturou, která se snaží minimalizovat závislosti a zaručit oddělení starostí, což vede k vytvoření robustních a efektivních webových aplikací. Hanami také obsahuje řadu nástrojů a knihoven pro podporu komplexních aplikací, včetně ORM, routování a šablonování. [76]

C#

C# je moderní, objektově orientovaný programovací jazyk vyvinutý společností Microsoft. Jazyk C# se stal klíčovou součástí .NET frameworku a je často používán pro vývoj široké škály aplikací, včetně desktopových, webových, mobilních a herních aplikací. C# je známý svou jednoduchostí, čitelností a vysokou výkonností. Nabízí silný systém typů, automatickou správu paměti a bohatou sadu knihoven a nástrojů, které usnadňují vývoj. C# podporuje asynchronní programování a mnoho moderních funkcí, jako jsou LINQ, delegáty, události a vzory. Díky své blízké integraci s platformou .NET a širokou podporou vývojových nástrojů, C# se stal jedním z preferovaných jazyků pro vývoj aplikací na platformě Microsoft. [77]

V případě jazyka C# je hlavním frameworkem pro vývoj backendu webových aplikací ASP.NET Core. I když existují i alternativy, například Nancy [78], jsou velmi málo používané, a proto nebudou dále rozebírány.

ASP.NET Core je moderní a výkonný open-source webový framework pro vývoj backendu v jazyce C#. Je to evoluce původního ASP.NET frameworku, která nabízí lehký a škálovatelný přístup. ASP.NET Core poskytuje mnoho funkcí, jako

je routování, middleware, autentizace a autorizace. Zároveň podporuje vývoj webových aplikací pomocí architektury Model-View-Controller (MVC). Jeho modulární design umožňuje vývojářům vybrat a použít pouze potřebné komponenty, což vede k optimalizovanému výkonu. ASP.NET Core je také kompatibilní s různými operačními systémy, což umožňuje nasazení aplikací na platformách jako Windows, Linux nebo macOS. Díky integraci s platformou .NET, nabízí ASP.NET Core širokou škálu nástrojů a knihoven pro vývoj a správu aplikací. Je to silný a oblíbený framework, který je široce využíván pro vytváření rychlých, škálovatelných, moderních webových aplikací a API. [79]

4.5.2 Databáze

Tato kapitola se zabývá databázemi pro webové aplikace a jejich významem v procesu vývoje. Databáze jsou nezbytnou součástí webových aplikací, kde jsou využívány k ukládání, správě a získávání dat. Při vývoji webových aplikací je klíčové vybrat vhodný databázový systém, který odpovídá specifickým požadavkům projektu a zohledňuje faktory jako výkon, škálovatelnost a bezpečnost. Existují různé typy databázových systémů, které se často dělí na relační (SQL) a nestrukturované (NoSQL) databáze.

Relační databáze (SQL) jsou ideální pro aplikace, které vyžadují pevně definované a strukturované vztahy mezi daty. Pokud je potřeba ukládat a manipulovat s daty, která mají přesně definovanou strukturu a logiku, relační databáze jsou vhodnou volbou. Příklady takových aplikací zahrnují systémy pro správu zákazníků (CRM), systémy pro správu inventáře nebo finanční systémy. [80]

Nestrukturované databáze (NoSQL) jsou vhodné pro aplikace, které pracují s velkými objemy dat nebo vyžadují rychlou škálovatelnost. NoSQL databáze poskytují flexibilitu a dynamický přístup k datům, což je ideální pro aplikace s proměnlivou a nestrukturovanou povahou dat. Tyto databáze jsou často využívány v aplikacích jako jsou sociální sítě, analytické nástroje, personalizované doporučovací systémy nebo systémy sběru a analýzy dat. [81]

SQL databáze

Existuje celá řada SQL databází a jejich výběr záleží na konkrétních potřebách projektu, mezi nejznámější patří:

- **MySQL** - MySQL je jedním z nejpopulárnějších open-source relačních databázových systémů. Je snadno použitelný, škálovatelný a nabízí širokou škálu funkcí. MySQL je vysoce spolehlivý, rychlý a podporuje transakce, což z něj činí ideální volbu pro webové aplikace a podnikové systémy. Je kompatibilní s různými operačními systémy a poskytuje rozsáhlou podporu pro standardní SQL dotazy. Je vhodný spíše pro menší projekty. [82]

- **PostgreSQL** - PostgreSQL je výkonný a rozšiřitelný open-source relační databázový systém s důrazem na přesnost a bezpečnost dat. Podporuje širokou škálu pokročilých funkcí, jako jsou transakce, pohledy, procedury a integrovaný plnohodnotný programovací jazyk. PostgreSQL je oblíbený pro svou flexibilitu a schopnost zpracovat rozsáhlá datová schémata a složité dotazy. Oproti MySQL nabízí pokročilejší funkce a je vhodnější pro složitější projekty. [83]
- **Microsoft SQL Server** - Microsoft SQL Server je relační databázový systém vyvinutý společností Microsoft. Je vysoce výkonný, spolehlivý a nabízí širokou škálu funkcí pro správu dat, zabezpečení a výkonové optimalizace. SQL Server je navržen pro integraci s dalšími produkty společnosti Microsoft, jako je .NET Framework a Azure, a je často používán pro vývoj podnikových aplikací. [84]

NoSQL databáze

Následující seznam zahrnuje některé z nejpopulárnějších NoSQL databází:

- **MongoDB** - MongoDB je oblíbená open-source dokumentová NoSQL databáze. Ukládá data ve formátu podobném dokumentům JSON, což umožňuje flexibilitu a snadnou škálovatelnost. MongoDB je výkonná, rychlá a nabízí možnost dotazování pomocí jazyka dotazování nad dokumenty (Query Language). Je vhodná pro aplikace s proměnlivou a nestrukturovanou povahou dat, jako jsou sociální sítě, personalizované doporučovací systémy a analýza velkých dat. [85]
- **Redis** - Redis je vysoce výkonná open-source klíč-hodnota NoSQL databáze. Umožňuje ukládání dat ve formě klíč-hodnota a nabízí rychlý přístup k datům. Redis je často používán pro cacheování, správu sezení, zpracování front úloh (job queueing) a publikování-předplatné systémy. Jeho výkonnost a jednoduchost použití ho činí oblíbenou volbou pro aplikace, které vyžadují rychlý přístup k datům v reálném čase. [86]
- **Neo4j** - Neo4j je populární open-source grafová databáze, která se specializuje na ukládání a manipulaci s grafovými daty. Je optimalizována pro efektivní zpracování vztahů mezi daty a poskytuje výkonné dotazovací možnosti pomocí jazyka Cypher. Neo4j je často využívána v sociálních sítích, doporučovacích systémech, sítích dopravních spojení a analýze sítě vztahů. [87]

4.6 Verzování

Verzování je kritickým aspektem vývoje webových aplikací, umožňujícím sledování a řízení změn v zdrojovém kódu. Umožňuje vývojářům vrátit se ke starším verzím kódu, sledovat kdo a kdy udělal určité změny a řešit konflikty mezi různými změnami provedenými různými vývojáři. Navíc verzování může podporovat různé

vývojové metodiky, jako je například agilní vývoj nebo kontinuální integrace a nasazení (CI/CD). [88]

Ačkoliv existuje více verzovacích systému a platforem pro hosting repozitářů. V drtivé většině případů se používá Git a Github.

4.6.1 Git

Git je nejčastěji používaný systém pro správu verzí, který byl původně vytvořen Linusem Torvaldsem pro správu vývoje jádra Linuxu. Jde o distribuovaný systém, což znamená, že každý vývojář má vlastní lokální kopii celého repozitáře. To umožňuje práci i mimo připojení k internetu a zvyšuje rychlost mnoha operací. Git podporuje snadné sloučení změn z různých zdrojů, což je klíčové pro týmový vývoj. Git je navržen tak, aby byl flexibilní a mohl podporovat různé vývojové workflow. Může být použit pro lineární vývoj s jednou hlavní větví, pro paralelní vývoj s mnoha větvemi pro jednotlivé funkce nebo pro kombinaci těchto přístupů. [89]

4.6.2 GitHub

GitHub je webová platforma, která hostuje repozitáře Git a nabízí široké spektrum nástrojů pro týmovou spolupráci a správu projektů při vývoji softwaru. Jednou z klíčových funkcí je sledování problémů (issue tracking), které umožňuje týmům sledovat a řešit problémy a požadavky na nové funkce na jednom místě. Další zásadní funkcí je code review, proces, který umožňuje vývojářům prohlížet a komentovat změny v kódu předtím, než jsou začleněny do hlavní větve. Tento proces může výrazně zlepšit kvalitu kódu a podporovat sdílení znalostí v rámci týmu. GitHub také podporuje integraci s mnoha dalšími nástroji pro vývoj software, včetně systémů pro kontinuální integraci a nasazení (CI/CD), což umožňuje automatické testování a nasazení kódu. [90]

5 Vytvoření aplikace

Tato kapitola se zaměřuje na proces vytvoření webové aplikace dle požadavků definovaných v analytické části práce. Na začátku budou popsány funkce aplikace, které budou splňovat dané požadavky. Následně proběhne výběr vhodných technologií pro tvorbu aplikace. Zbytek kapitoly bude zaměřen na samotnou tvorbu aplikace včetně návrhu databáze, nastavení vývojového prostředí, popis struktury projektu, rozdělení aplikace do logických celků a následným popisem jednotlivých částí programu. Pro tvorbu aplikace bylo převážně čerpáno z [91], kde je na praktických příkladech ukázán vývoj webových aplikací v Djangu.

5.1 Funkce aplikace

Funkce aplikace vychází jak ze všeobecné problematiky provozu veterinární ordinace tak z požadavků plynoucích z analytické části. Nová webová aplikace by tedy měla obsahovat následující funkce:

- **Kartotéka** - Kartotéka je nejdůležitější částí aplikace. Je potřeba aby bylo možné evidovat majitele, jejich zvířata a především evidovat ke zvířatům případy a k nim pak návštěvy. Pro všechny tyto entity bude možnost provádět tzv. CRUD operace, tedy vytvoření (Create), čtení (Read), aktualizace (Update) a odstranění (Delete).
- **Sklad** - V rámci vedení skladu je potřeba mít možnost vedení více skladů s definováním marže skladu, a samozřejmě vedení skladových položek, včetně všech potřebných údajů. Opět bude možné pro všechny entity provádět CRUD operace. Položky ze skladu musí být možné přidat k návštěvě pro potřeby vytvoření faktury.
- **Úkony** - Jde o seznam, který obsahuje veterinární úkony prováděné v rámci návštěvy nebo operace. Oproti skladovým položkám se u nich nevede množství, proto není možné mít úkony v rámci skladu jako položky. Opět pro úkony budou k dispozici všechny CRUD operace.
- **Účetnictví** - Pro potřeby účetnictví je potřeba možnost k návštěvě vytvořit fakturu obsahující informace o majiteli, skladových položkách a úkonech přiřazených k návštěvě. Faktury budou shromážděny na jedné stránce a bude možné je exportovat.
- **Autentizace** - V rámci zabezpečení je potřeba se do aplikace registrovat, nebo v případě již vytvořeného účtu, přihlásit.

Pro splnění zbytku požadavků bude aplikace vyvíjena s ohledem na responzivitu na mobilních zařízeních, přehlednost a jednoduchost.

5.2 Použité technologie

Po definování požadavků na funkčnost je potřeba zvolit technologie pro vývoj webové aplikace. Jelikož je nabídka velmi široká, může být výběr obtížný. Je proto vhodné seznámit se se všemi možnostmi a následně zvolit vhodné technologie, jak z hlediska požadovaných funkcí, tak i z hlediska osobních preferencí.

5.2.1 Backend

V případě výběru vhodné technologie pro backend část je jako první na řadě vybrat vhodný jazyk. V případě volby jazyka jde ve velké částí o subjektivní výběr dle zkušeností vývojáře nebo o možnosti, které jazyk a jeho ekosystém nabízí. Pro backend byl tedy zvolen jazyk Python, z důvodu jednoduchosti, dobrého výkonu a bohaté knihovně balíčků. Co se týče frameworků pro vývoj webových aplikací v jazyku Python, tak se nabízí několik možností, které již byly zmíněny v části o dostupných technologiích. Pro tvorbu aplikace byl zvolen framework Django a to hlavně díky jeho výborně zpracované dokumentaci, zabudovaným systémem autentizace, administračním rozhraním, MVT architektuře a velkým množstvím dostupných balíčků pro rozšíření jeho funkcionality.

5.2.2 Databáze

Pro uchování dat byla použita databáze PostgreSQL. Ačkoliv výchozí databází pro Django je SQLite, není tato databáze příliš vhodná kvůli omezenému sdílenému přístupu a nedostatečné škálovatelnosti. PostgreSQL byla zvolena zejména kvůli nejlepší podpoře ve frameworku Django, kde dosahuje s touto databází nejlepšího výkonu.

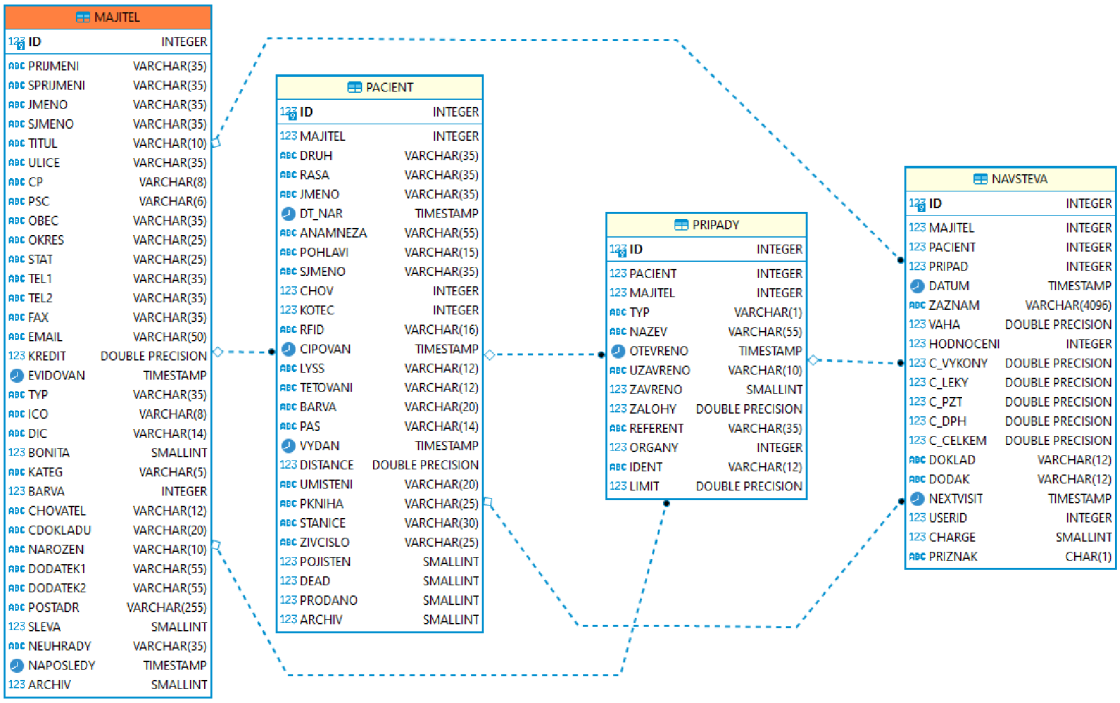
5.2.3 Frontend

V rámci fronendu se v případě webových aplikací nelze obejít bez HTML, CSS a JavaScriptu, které tvoří základ moderních webových aplikací. Pro zjednodušení práce se stylováním pomocí CSS je na výběr z několika možných frameworků. Pro rychlý vývoj a originální vzhled se nabízí použití kombinace Tailwind CSS a Flowbite, kdy Tailwind CSS nabízí jednoduché individuální úpravy přímo v HTML kódu pomocí tříd a Flowbite nabízí předpřipravené komponenty, které lze rychle použít a pomocí Tailwind CSS upravit vzhled dle vlastních preferencí. Jelikož aplikace nebude využívat nějaké větší možnosti reaktivních prvků, bude použit čistý JavaScript bez frameworků.

5.3 Návrh databáze

Při návrhu databáze bylo částečně vycházeno z původní databáze programu Vetis office. Vetis office používá pro ukládání dat databázi Firebird, která běží na lokálním serveru, respektive na počítači, kde je program nainstalovaný. Z původní databáze bylo vycházeno hlavně kvůli vytvoření představy o ukládaných datech a vztazích mezi tabulkami. Díky tomu je možné vytvořit novou databázi tak, aby bylo možné při nasazení nové aplikace jednodušeji migrovat data ze staré databáze do nové.

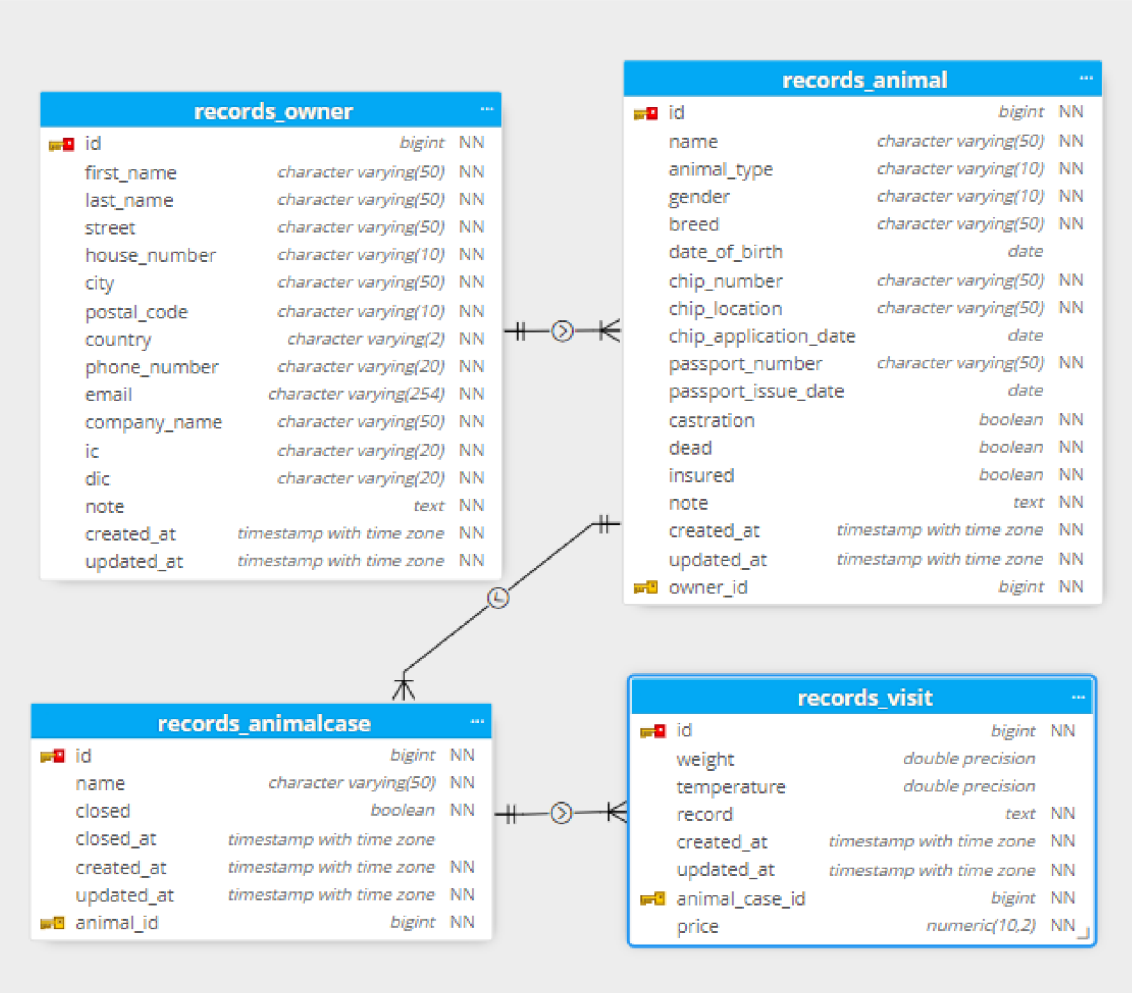
V případě kartotéky, tedy evidenci majitelů, zvířat, případů a návštěv, je struktura původní databáze popsána ER diagram zobrazeným na obr. 14.



Obr. 14: ER diagram kartotéky programu Vetis office

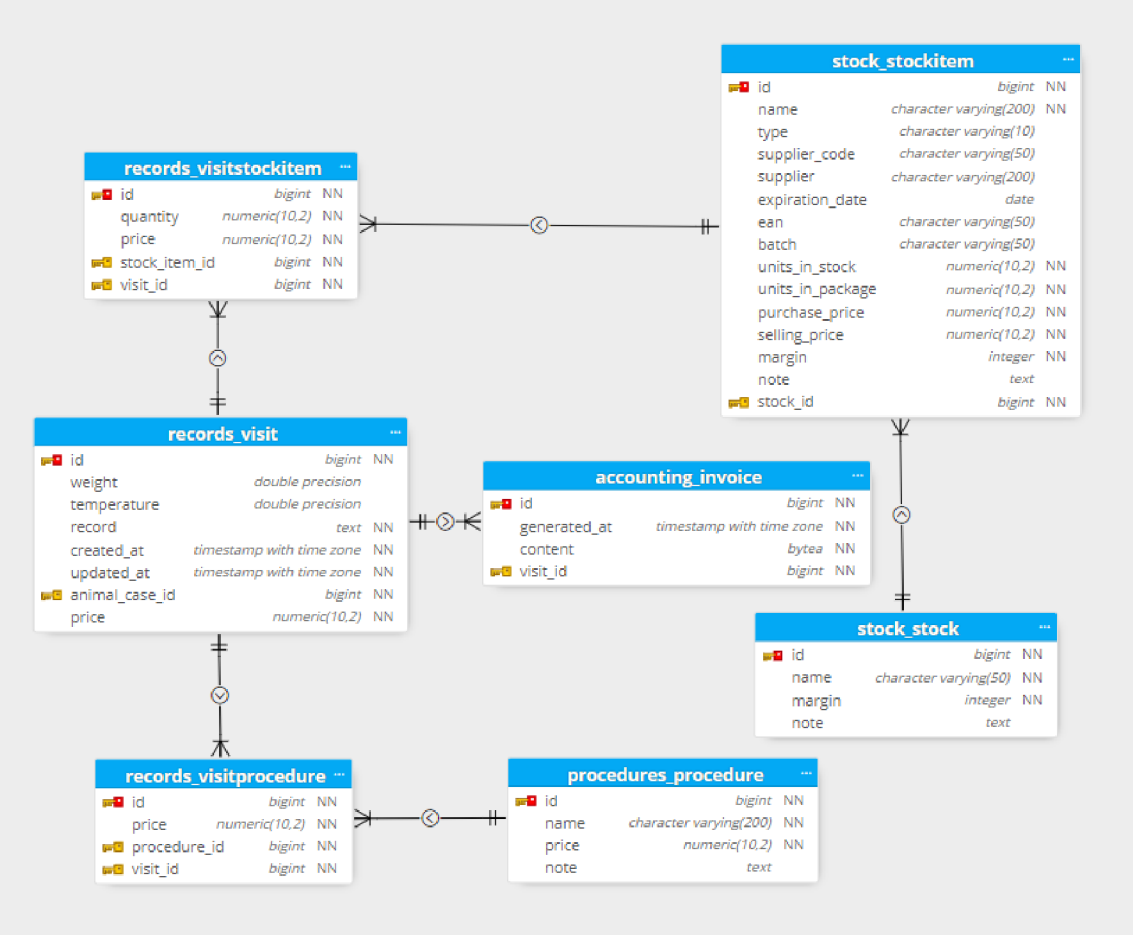
Ve výše uvedené ER diagramu lze vidět, že pro tabulku *NAVSTEVA* je využito více cizích klíčů. Kromě propojení s tabulkou *PRIPADY* jsou zde i cizí klíče pro spojení s tabulkou *PACIENT* a *MAJITEL*. To je užitečné v případě, že se využívá přímé spojení mezi těmito tabulkami a snižuje se tak počet potřebných spojení tabulek. Toto řešení má ovšem i nevýhody. Především se jedná o porušení 3. normální formy, která vyžaduje, aby byly eliminovány všechny nesouvislé závislosti. Tím pádem by mohlo dojít k problému s konzistencí dat, například pokud by byla změněna vazba mezi majitelem a zvířetem. Proto tyto případy může být výhodnější tyto cizí klíče odstranit. Tím je sice potřeba spojit více tabulek pro přístup k některým datům, ale nedojde pak problému s konzistencí dat. Vzhledem k tomu, že aplikace nebude tak velkého rozsahu, nemělo by dojít k výraznému zhoršení výkonu.

Další změnou oproti stávající struktuře databáze je odstranění některých atributů, které nejsou pro novou aplikaci potřeba. Zároveň se přistoupilo k pojmenování atributů v anglickém jazyce, který je považován za univerzální a standardní pro vývoj. Na obr. 15 je zobrazen ER diagram kartotéky pro novou aplikaci.



Obr. 15: ER diagram kartotéky webové aplikace

Dále je potřeba rozšířit databázovou strukturu o evidenci skladových položek a úkonů k návštěvě. S tím souvisí i potřeba vytvoření tabulek pro sklad a skladové položky v rámci skladu. Zde se již struktura nové aplikace bude od původní lišit. Je to zejména z toho důvodu, že původní aplikace má například oddělené léky od zbytku skladových položek nebo kvůli jinému způsobu evidenci faktur. Vytvořená struktura zbytku webové aplikace je reprezentována ER diagramem na obr. 16.



Obr. 16: ER diagram zbytku webové aplikace

5.4 Nastavení vývojového prostředí

Před vývojem samotné aplikace je potřeba nastavit vývojové prostředí. Vývoj probíhal na systému Windows 11. Aplikace je napsána v jazyce Python, který je nejdříve potřeba nainstalovat z oficiálních stránek[1]. Pro zajištění správné funkčnosti je potřeba stáhnout správnou verzi Pythonu, kterou podporuje aktuální verze Django frameworku. V době psaní práce je nejnovější verze Django frameworku 4.2, která podporuje rovněž nejnovější verzi Pythonu, tedy 3.11.3 [92].

Dále je potřeba editor pro psaní kódu aplikace. K tomu bude sloužit Visual Studio Code (zkráceně VSCode), který nabízí spoustu nástrojů usnadňující vývoj. Pro vývoj aplikace byly použiti doplňky Python, Django, djLint a Tailwind CSS IntelliSense. Doplněk Python zajišťuje podporu pro jazyk Python ve VSCode [93]. Django doplněk poskytuje syntaxi a barevné zvýrazňování v Django šablonách [94]. DjLint zajišťuje analýzu chyb a formátování Django šablon [95]. Tailwind CSS Intel-

[1] https://www.python.org/

liSense pak poskytuje automatické doplňování názvu tříd pro stylování HTML kódu pomocí Tailwind CSS frameworku [96].

Pro izolaci projektových závislostí je běžnou praktikou vytvoření virtuálního prostředí, které umožňuje oddělit závislosti od jiných projektů. To je užitečné v případě vývoje několik projektů, kdy každý vyžaduje jinou verzi Pythonu nebo jeho knihoven. Pro vytvoření virtuálního prostředí ve složce s projektem lze použít příkaz **`python -m venv myvenv`**, který vytvoří nové virtuální prostředí s názvem *myvenv*. Aby bylo možné instalovat knihovny do nově vytvořeného virtuálního prostředí, je nejdříve nutné ho aktivovat. To lze udělat příkazem **`myvenv\Scripts\activate`**.

Při použití databáze PostgreSQL je potřeba mít tuto databázi nainstalovanou. To lze opět udělat z oficiálních stránek[2]. Po stažení a instalaci je potřeba vytvořit prázdnou databázi, do které se budou data ukládat.

5.5 Django projekt

Jakmile je vývojové prostředí aktivováno je možné nainstalovat Django framework pomocí příkazu **`pip install django`**. Pro ověření, jestli instalace proběhla správně je možné použít příkaz **`django-admin version`**, který vypíše nainstalovanou verzi Django frameworku. Pak je možné vytvořit projekt pomocí příkazu **`django-admin startproject vetsimplify .`**, kde tečka na konci příkazu značí, že se projekt vytvoří do aktuálně vybrané složky.

Po vytvoření projektu je dobré zajistit jeho verzování, tedy uchovávání historie veškerých provedených změn. K tomu bude použit nástroj Git. Vytvořený repozitář pak bude nahráván na GitHub. Git je potřeba opět nainstalovat z oficiálních stránek[3]. Po instalaci je potřeba Git nakonfigurovat. To lze dosáhnout pomocí příkazu pro nastavení jména: **`git config --global user.name "Vase skutecne jmeno"`** a e-mailové adresy: **`git config --global user.email "vas@email.com"`**. U e-mailové adresy je vhodné použít adresu, která je použita pro GitHub účet pro nahrání do GitHub repozitáře. Pro nastavení přístupu pro GitHub lze postupovat podle návodu na webových stránkách GitHubu [97]. Předtím než se vytvoří Git repozitář je vhodné vytvořit konfigurační soubor .gitignore, který specifikuje soubory nebo adresáře, které by měly být ignorovány verzovacím systémem. To mohou být například konfigurační soubory editoru nebo IDE, soubory virtuálního prostředí nebo mediální soubory uživatelů. Následně je možné vytvořit repozitář pomocí příkazu **`git init`**, přidat všechny adresáře a soubory (mimo těch, které jsou specifikovány v .gitignore souboru) pomocí **`git add .`** a vytvořit první commit pomocí **`git commit -m "Initial commit"`**.

[2] https://www.postgresql.org/
[3] https://git-scm.com/

5.6 Rozdělení projektu na aplikace

Ve frameworku Django se rozlišuje mezi projektem a aplikacemi. Projekt lze pochopit jako rámec pro celou webovou stránku, zatímco aplikace jsou jednotlivé funkční části této webové stránky. Toto pojetí usnadňuje práci na větších projektech a zároveň poskytuje lepší přehlednost. Jednotlivé aplikace lze snadno vytvořit pomocí příkazu `python manage.py startapp nazev_aplikace`.

Pro tvorbu webové aplikace došlo k rozdělení projektu do několika aplikací:

- **accounts** - Aplikace, která se stará o uživatelské účty, tedy registraci a přihlášení uživatelů.
- **records** - Tato aplikace má na starosti celou kartotéku, tedy evidenci majitelů, zvířat, případů a návštěv
- **stock** - Jde o aplikaci, která se zabývá skladovým systémem.
- **accounting** - V této aplikaci je řešeno vytváření, uchovávání a export faktur.
- **procedures** - Jedná se o aplikaci, která zajišťuje seznam úkonů.
- **frontend_pages** - Tato aplikaci slouží pouze pro zobrazování statických stránek, tedy například domovská stránka, která nabízí přihlášení nebo registraci.

5.7 Adresářová struktura projektu

Při vytvoření nového projektu se automaticky vytvoří adresář s názvem projektu, který obsahuje několik souborů. Jejich účel je následující:

- **___init___.py** - Prázdný soubor, který Pythonu říká, že tento adresář má být považován za Python balíček.
- **settings.py** - Obsahuje všechna nastavení pro Django projekt, včetně konfigurace databáze, nastavení časové zóny, jazyka a seznamu instalovaných aplikací.
- **urls.py** - Tento soubor obsahuje definice URL adres pro projekt.
- **asgi.py a wsgi.py** - Tyto soubory jsou vstupními body pro webové servery. WSGI (Web Server Gateway Interface) je standard pro Python webové aplikace a servery, zatímco ASGI (Asynchronous Server Gateway Interface) je novější standard, který podporuje asynchronní funkce.

Dále se vytvoří v kořenovém adresáři projektu soubor **manage.py**. Jedná se o příkazový řádek, který poskytuje řadu příkazů, které usnadňují práci s Django projektem.

Po vytvoření nové aplikace pomocí příkazu `python manage.py startapp nazev_aplikace` se v rámci adresáře aplikace automaticky vytvoří následující soubory:

- **___init___.py** - Stejně jako v projektu, i zde je tento soubor prázdný a slouží k označení adresáře jako Python balíčku.

- **admin.py** - Tento soubor slouží pro definování, jaké modely aplikace budou zobrazeny v Django administračním rozhraní.
- **apps.py** - Obsahuje konfiguraci aplikace.
- **migrations/** - Adresář, který obsahuje všechny databázové migrace aplikace. Databázové migrace jsou způsob, jakým Django sleduje změny v modelech, takže může správně vytvořit databázové schéma.
- **models.py** - Tento soubor obsahuje všechny modely databáze aplikace. Modely jsou reprezentací databázových tabulek.
- **tests.py** - Tento soubor slouží pro psaní testů pro aplikaci.
- **views.py** - V tomto souboru jsou definovány všechny pohledy aplikace. Pohledy jsou funkce nebo třídy, které přijímají web request a vracejí web response.

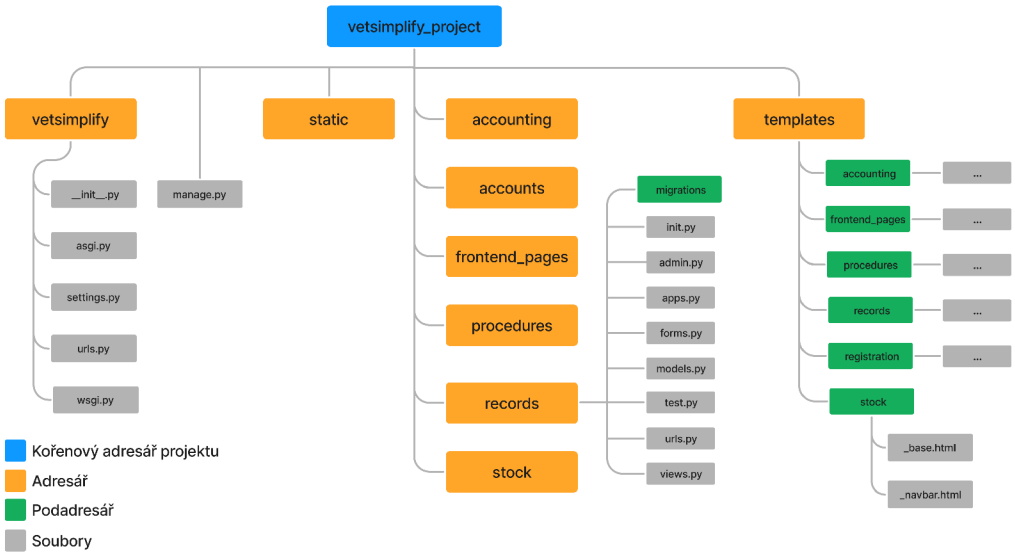
Přestože Django automaticky vytvoří základní soubory potřebné pro funkčnost aplikace, často je užitečné a nutné přidat další soubory, které rozšiřují funkčnost a organizaci kódu. Dva takovéto soubory, které jsou často přidávány do Django aplikací, jsou forms.py a urls.py:

- **forms.py** - Místo, kde se definují všechny formuláře související s danou aplikací.
- **urls.py** - Aplikace často obsahují svůj vlastní soubor, ve kterém se definují URL vzory, které jsou specifické pro danou aplikaci.

Dalšími důležitými komponentami adresářové struktury Django aplikace jsou adresáře *static* a *templates*. Tyto adresáře nejsou automaticky vytvořeny při vytvoření nového projektu nebo aplikace, ale jsou přidány ručně.

- **static** - Tento adresář obsahuje všechny statické soubory, které se používají v rámci aplikace. Jedná se o všechny soubory, které nejsou dynamicky generovány. To zahrnuje CSS soubory, JavaScript soubory nebo obrázky.
- **templates** - Adresář templates je místo, kde jsou všechny HTML šablony pro projekt. Pro usnadnění orientace je tento adresář rozdělen na podadresáře s názvy jednotlivých aplikací. V nich jsou pak šablony, které jsou specifické pro jednotlivé aplikace.

Výsledná struktura Django projektu vytvořené webové aplikace je znázorněna na obr. 17.



Obr. 17: Adresářová struktura projektu

5.8 Nastavení projektu - settings.py

V souboru *settings.py* se provádí veškeré nastavení projektu. V následujících částech budou ukázány některá z klíčových nastavení, které bylo potřeba provést.

5.8.1 Konfigurace databáze

Tato sekce konfiguruje databázový backend, který Django používá. Vzhledem k tomu, že byla použita databáze PostgreSQL je potřeba změnit toto nastavení způsobem zobrazeným na obr. 18.

```
85  DATABASES = {
86      "default": {
87          "ENGINE": "django.db.backends.postgresql",
88          "NAME": "vetsimplify_dev",
89          "USER": [redacted],
90          "PASSWORD": [redacted],
91          "HOST": "localhost",
92          "PORT": "5432",
93      }
94  }
```

Obr. 18: Nastavení databáze pro Django projekt

Ve výchozím stavu ovšem Django nezahrnuje databázový engine pro práci s PostgreSQL databází a je proto potřeba ho doinstalovat. To lze pomocí příkazu **`pip install psycopg`**.

5.8.2 Konfigurace umístění šablon

Ve výchozím stavu Django hledá šablony pouze v každé nainstalované aplikaci uvnitř příslušného podadresáře *templates*. K tomu aby bylo možné mít v kořenovém adresáři projektu adresář templates se všemi šablonami, je potřeba upravit *settings.py* dle obr. 19.

```
63 TEMPLATES = [
64     {
65         "BACKEND": "django.template.backends.django.DjangoTemplates",
66         "DIRS": [BASE_DIR / "templates"],
67         "APP_DIRS": True,
68         "OPTIONS": {
69             "context_processors": [
70                 "django.template.context_processors.debug",
71                 "django.template.context_processors.request",
72                 "django.contrib.auth.context_processors.auth",
73                 "django.contrib.messages.context_processors.messages",
74             ],
75         },
76     },
77 ]
```

Obr. 19: Úprava nastavení umístění šablon

5.8.3 Lokalizace a časová zóna

Pro změnu výchozí lokalizace pro překlad výchozích textů a zpráv generovaných Django frameworkem a úpravu výchozí časové zóny je nutné upravit *settings.py* způsobem zobrazeným na obr. 20.

```
119 LANGUAGE_CODE = "cs-cz"
120
121 TIME_ZONE = "Europe/Prague"
122
```

Obr. 20: Úprava nastavení lokalizace a časové zóny

5.8.4 Nainstalované aplikace

Při vytvoření nové aplikace je potřeba ji zaregistrovat v souboru *settings.py*. Stejně tak je zde potřeba zaregistrovat dodatečně nainstalované balíčky. Výsledný kód je zobrazen na obr. 21.

```
33  INSTALLED_APPS = [
34      # Django built-in apps
35      "django.contrib.admin",
36      "django.contrib.auth",
37      "django.contrib.contenttypes",
38      "django.contrib.sessions",
39      "django.contrib.messages",
40      "django.contrib.staticfiles",
41      # My custom apps
42      "frontend_pages.apps.FrontendPagesConfig",
43      "accounts.apps.AccountsConfig",
44      "records.apps.RecordsConfig",
45      "stock.apps.StockConfig",
46      "procedures.apps.ProceduresConfig",
47      "accounting.apps.AccountingConfig",
48      # Third party apps
49      "crispy_forms",
50      "crispy_tailwind",
51      "widget_tweaks",
52  ]
```

Obr. 21: Registrace aplikací

5.8.5 Konfigurace statických souborů

Pro statické soubory jako jsou CSS, JavaScript, obrázky a další je potřeba nastavit adresáře a URL adresy, přes které se bude k těmto souborů přistupovat. Pro nastavení lze použít kód z obr. 22.

```
131  # Static files (CSS, JavaScript, Images)
132  # https://docs.djangoproject.com/en/4.2/howto/static-files/
133
134  STATIC_URL = "static/"
135
136  STATICFILES_DIRS = [BASE_DIR / "static"]
137
138  STATIC_ROOT = BASE_DIR / "staticfiles"
```

Obr. 22: Konfigurace statických souborů

Pomocí **`STATIC_URL`** se definuje URL adresa **`static/`**, přes kterou bude Django přistupovat ke statickým souborům. **`STATICFILES_DIRS`** definuje seznam adresářů, kde Django hledá statické soubory. Nakonec **`STATIC_ROOT`** definuje místo, kam Django uloží statické soubory při použití příkazu **`collectstatic`**, který slouží ke shromáždění všech statických souborů a uložení na jedno místo, což usnadňuje nasazení na reálný server.

5.9 Vlastní implementace aplikace

V rámci této kapitoly bude popsán vývoj aplikace a vysvětlení některých částí aplikace. Vysvětlení vývoje ve frameworku Django bude vysvětleno především na implementaci kartotéky. U zbytku aplikace budou popsány jen zajímavě částí, které

se nevyskytovaly u implementace kartotéky. Toto rozhodnutí bylo učiněno zejména kvůli tomu, že většina principů se v rámci vývoje aplikace opakuje a není nutné je tak znovu opakovat.

5.9.1 Kartotéka

Kartotéka je klíčovou součástí webové aplikace pro správu veterinární kliniky. Umožňuje evidovat všechny potřebné informace o majitelích, jejich zvířatech, případech a návštěvách.

Modely

Při vytvoření aplikace, která se bude starat o evidenci všech potřebných údajů, je nejprve nutné vytvořit modely pro všechny potřebné objekty jako jsou majitelé, zvířata, případy, návštěvy a následně k návštěvám skladové položky a úkony. V Djagnu model popisuje strukturu a chování daného objektu a také slouží jako abstrakce databáze. Django dle definice modelu vytváří databázové migrace a dle nich jsou pak aplikovány všechny změny do databáze. Jako příklad je uveden na obr. 23 model pro objekt *Owner*.

```
8   class Owner(models.Model):
9       first_name = models.CharField(max_length=50)
10      last_name = models.CharField(max_length=50)
11      street = models.CharField(max_length=50)
12      house_number = models.CharField(max_length=10)
13      city = models.CharField(max_length=50)
14      postal_code = models.CharField(max_length=10)
15      country = CountryField(blank_label="(vyberte zemi)")
16      phone_number = models.CharField(max_length=20, blank=True)
17      email = models.EmailField(blank=True)
18      company_name = models.CharField(max_length=50, blank=True)
19      ic = models.CharField(max_length=20, blank=True)
20      dic = models.CharField(max_length=20, blank=True)
21      note = models.TextField(blank=True)
22      created_at = models.DateTimeField(auto_now_add=True)
23      updated_at = models.DateTimeField(auto_now=True)
24
25      def __str__(self):
26          return f"{self.first_name} {self.last_name}"
27
```

Obr. 23: Kód pro model objektu *Owner*

Kromě definice všech potřebných polí je na konci i metoda `__str__()`, která je využita pro převod objektu na řetězec. V tomto případě je funkce definována tak, aby vrátila řetězec sestávající z *first_name* a *last_name* dané instance *Owner*. Tato metoda je použita pro přehledné zobrazení v administrátorském rozhraní.

Pro vytvoření potřebných tabulek v databázi je nejdříve nutné vytvořit migraci pomocí příkazu **`python manage.py makemigrations`** a následně tuto migraci aplikovat na databází pomocí příkazu **`python manage.py migrate`**. Podobně jsou vytvořeny i ostatní modely, které odpovídají návrhu databáze.

Views

V Djangu je každá webová stránka reprezentována pomocí tzv. view. View je Python funkce nebo třída, která zpracovává HTTP požadavky a vrací HTTP odpověď. Tato odpověď může být různých forem: HTML stránka, přesměrování na jinou stránku, atd.

V Djangu jsou používány dva základní typy views: založené na funkcích (Function-Based-Views - FBV) nebo založené na třídách (Class-Based-Views - CBV). Zatímco FBV využívají Python funkcí, CBV využívají třídy a dědičnost pro obecnější a znovupoužitelný kód. I přesto, že CBV jsou často efektivnější, FBV nabízejí větší kontrolu a přehlednost. V aplikaci jsou CBV využívány především pro CRUD operace, které jsou již v Djangu definovány. Pro pokročilejší funkce, například vytváření a úpravu návštěvy jsou využity FBV.

Základem aplikace pro kartotéku jsou modely *Owner*, *Animal*, *AnimalCase* a *Visit*, pro které jsou vytvořeny views pro CRUD operace. Pro každý model jsou rovněž vytvořeny i třídy mixin (např. *OwnerMixin*, *AnimalMixin*, atd.), které obsahují metody pro získání instance modelu z URL parametrů. Tyto mixin třídy umožňují snadno získat přístup k relevantním instancím model ve views.

```python
class OwnerListView(LoginRequiredMixin, ListView):
    model = Owner
    paginate_by = 10

class OwnerDetailView(LoginRequiredMixin, OwnerMixin, DetailView):
    model = Owner

    def get_object(self):
        return self.get_owner()

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context["animal_list"] = Animal.objects.filter(owner=self.object)
        return context


class OwnerCreateView(LoginRequiredMixin, CreateView):
    model = Owner
    form_class = OwnerForm
    template_name = "records/owner_form.html"

    def get_success_url(self):
        return reverse("records:owner_detail", kwargs={"owner_id": self.object.pk})

class OwnerUpdateView(LoginRequiredMixin, OwnerMixin, UpdateView):
    model = Owner
    form_class = OwnerForm
    template_name = "records/owner_update_form.html"

    def get_object(self):
        return self.get_owner()

    def get_success_url(self):
        return reverse("records:owner_detail", kwargs={"owner_id": self.object.pk})


class OwnerDeleteView(LoginRequiredMixin, OwnerMixin, DeleteView):
    model = Owner

    def get_object(self):
        return self.get_owner()

    def get_success_url(self):
        return reverse("records:owner_list")
```

Obr. 24: Ukázka části kódu pro CRUD operace objektu *Owner*

Pro vytvoření a úpravu návštěvy byly použity FBV. V rámci vytvoření návštěv je potřeba zahrnout více funkcionalit. K návštěvě je potřeba evidovat kromě běžných údajů (text záznamu, hmotnost a teplota) i skladové položky a úkony. Klíčovou části funkce pro vytváření návštěvy je logika zpracování dat odeslaných uživatelem. Pokud uživatel odešle formulář, dojde k vytvoření instance tří formulářů, které zpracovávají různé části dat: data o záznamu z návštěvy, data o skladových položkách a data o úkonech. Pokud jsou všechny formuláře platné, vytvoří se nová návštěva a uloží se do databáze a data o skladových položkách a úkonech budou přidružena k návštěvě. Kód také aktualizuje zásoby ve skladu dle množství, které bylo ve formuláři zadáno. Pokud byl požadavek na vytvoření faktury, dojde k jejímu vygenerování a dojde k přesměrování na šablonu pro editaci návštěvy. Pro aktualizaci údajů lze použít obdobný kód s mírnými úpravami.

Formuláře

Pro přidání těchto objektů, nebo pro editaci jejich údajů jsou potřeba formuláře. Django sice nabízí vestavěné formuláře, ale práce s nimi není příliš efektivní. Pro zjednodušení práce s formuláři a snadné stylování byl použit balíček *Django Crispy Forms*. Lze ho nainstalovat pomocí příkazu `pip install django-crispy-forms` a vložit do instalovaných aplikací v souboru *settings.py* řádek `crispy_forms`. Django Crispy Forms využívá pro stylování formulářů balíčky šablon, v případě použití Tailwind CSS lze použít balíček *crispy-tailwind*, který lze nainstalovat příkazem `pip install crispy-tailwind`. Poté je ještě třeba nastavit, jaký balíček šablon má Django Crispy Forms použít. To se provede přidáním dvou řádku do *settings.py*: `CRISPY_ALLOWED_TEMPLATE_PACKS = "tailwind"`, který registruje balíček šablon a `CRISPY_TEMPLATE_PACK = "tailwind"`, který ho nastaví jako výchozí.

V šabloně lze pak jednoduše vložit formulář pomocí tagu `{% crispy forms %}`. Tento tag vytvoří formulář s metodou POST a přidá CSRF token, který zabezpečuje formuláře před CSRF útokem.

Pro návštěvy, jsou kromě standardního formuláře, navíc použity tzv. Formsety. Pomocí nich lze jednoduše vytvořit více instancí stejného formuláře. To je využito pro přidávání skladových položek a úkonů, jelikož je nutné mít možnost přidávat více různých položek nebo úkonů. Příklad užití Formsetů lze vidět na obr. 26.

```python
class OwnerForm(forms.ModelForm):
    You, 4 weeks ago | 1 author (You)
    class Meta:
        model = Owner
        fields = [
            "first_name",
            "last_name",
            "street",
            "house_number",
            "city",
            "postal_code",
            "country",
            "phone_number",
            "email",
            "company_name",
            "ic",
            "dic",
            "note",
        ]
        labels = {
            "first_name": "Jméno",
            "last_name": "Příjmení",
            "street": "Ulice",
            "house_number": "Č. p.",
            "city": "Město",
            "postal_code": "PSČ",
            "country": "Země",
            "phone_number": "Telefon",
            "email": "E-mail",
            "company_name": "Název firmy",
            "ic": "IČ",
            "dic": "DIČ",
            "note": "Poznámka",
        }

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.helper = FormHelper()
        self.helper.layout = Layout(
            Div(
                Div("first_name", css_class="col-span-3"),
                Div("last_name", css_class="col-span-3"),
                Div("street", css_class="col-span-2"),
                Div("house_number", css_class="col-span-1"),
                Div("city", css_class="col-span-2"),
                Div("postal_code", css_class="col-span-1"),
                Div("country", css_class="col-span-2"),
                Div("phone_number", css_class="col-span-2"),
                Div("email", css_class="col-span-2"),
                Div("company_name", css_class="col-span-2"),
                Div("ic", css_class="col-span-1"),
                Div("dic", css_class="col-span-1"),
                Div("note", css_class="col-span-6"),
                css_class="grid gap-4 grid-cols-6",
            ),
            Submit(
                "submit",
                "Uložit",
                css_class="cursor-pointer text-white bg-blue-700 hover:bg-blue-800
            ),
        )
```

Obr. 25: Ukázka formuláře pro objekt *Owner*

```python
class VisitStockItemForm(forms.ModelForm):
    You, 3 weeks ago | 1 author (You)
    class Meta:
        model = VisitStockItem
        fields = [
            "stock_item",
            "quantity",
            "price",
        ]
        labels = {
            "stock_item": "Položka skladu",
            "quantity": "Množství",
            "price": "Cena",
        }


VisitFormSet = inlineformset_factory(
    Visit,
    VisitStockItem,
    form=VisitStockItemForm,
    extra=0,
    can_delete=True,
    fields=("stock_item", "quantity", "price"),
)
```

Obr. 26: Použití Formsetů

URL adresy

URL adresy jsou definované v souboru *urls.py*. Každá URL adresa je přiřazena jednomu view, který je zodpovědný za zpracování požadavku a odeslání odpovědi. Pro příklad, zápis URL adres pro majitele a zvířata lze vidět na obr. 28.

Aby byly tyto URL adresy registrované v rámci projektu, je třeba je zahrnout do *urls.py* projektu. Na obr. 27 je zobrazen kód pro registraci URL adres pro všechny aplikace.

```python
urlpatterns = [
    path("admin/", admin.site.urls),
    path("", include("django.contrib.auth.urls")),
    path("", include("frontend_pages.urls")),
    path("", include("accounts.urls")),
    path("records/", include("records.urls")),
    path("stock/", include("stock.urls")),
    path("procedures/", include("procedures.urls")),
    path("accounting/", include("accounting.urls")),
]
```

Obr. 27: Registrace URL adres aplikací do projektu

```python
app_name = "records"

urlpatterns = [
    # Owners URLs
    path("owners", OwnerListView.as_view(), name="owner_list"),
    path("owners/create", OwnerCreateView.as_view(), name="owner_create"),
    path("owners/<int:owner_id>", OwnerDetailView.as_view(), name="owner_detail"),
    path(
        "owners/<int:owner_id>/update", OwnerUpdateView.as_view(), name="owner_update"
    ),
    path(
        "owners/<int:owner_id>/delete", OwnerDeleteView.as_view(), name="owner_delete"
    ),
    # Animals URLs
    path(
        "owners/<int:owner_id>/animals/create",
        AnimalCreateView.as_view(),
        name="animal_create",
    ),
    path(
        "owners/<int:owner_id>/animals/<int:animal_id>",
        AnimalDetailView.as_view(),
        name="animal_detail",
    ),
    path(
        "owners/<int:owner_id>/animals/<int:animal_id>/update",
        AnimalUpdateView.as_view(),
        name="animal_update",
    ),
    path(
        "owners/<int:owner_id>/animals/<int:animal_id>/delete",
        AnimalDeleteView.as_view(),
        name="animal_delete",
    ),
```

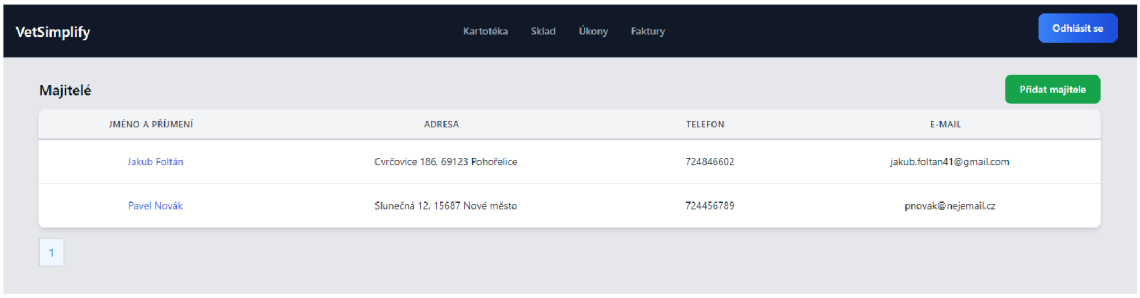
Obr. 28: Ukázka části URL adres pro kartotéku

Šablony a rozhraní aplikace

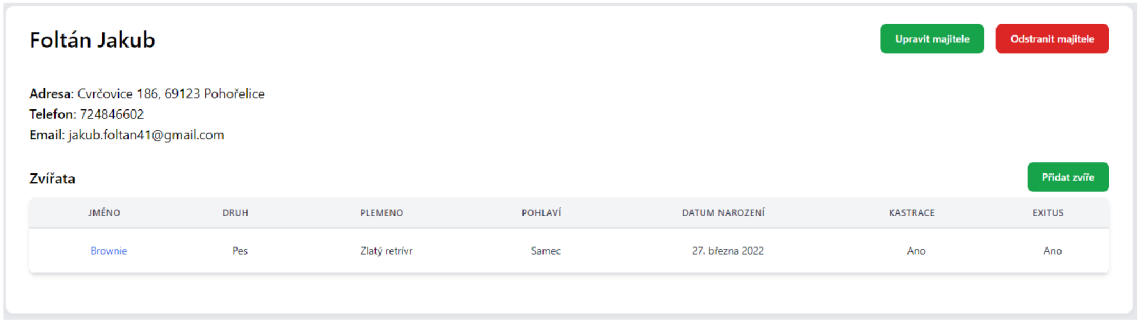
Šablony v Djangu jsou napsané pomocí Django Template Language (DTL), což je jednoduchý jazyk pro definování struktury HTML souborů. Umožňuje vkládat proměnné, používat logické konstrukce, jako jsou podmínky a cykly. Například pro procházení seznamu objektů lze použít cyklus `{% for %}`.

Kromě toho lze využívat dědičnosti šablon, což umožňuje vytvářet znovupoužitelné šablony například pro navigační menu. V rámci projektu byly použity dvě obecné šablony: *__base.html*, ve které je definovaná hlavička, která bude použitá pro všechny stránky a *__navbar.html* pro navigační menu.

Následně byly vytvořeny šablony pro zobrazení informací o majitelích, zvířatech, případech a návštěvách. Každý z těchto typů záznamů má vlastní šablonu pro zobrazení seznamu záznamů a detail záznamu. Kromě toho byly vytvořeny také šablony pro vytváření a úpravu těchto záznamů. V rámci šablon byl pro některé interaktivní prvky použit JavaScript, například pro interaktivní přidávání a odstranění skladových položek a úkonů.

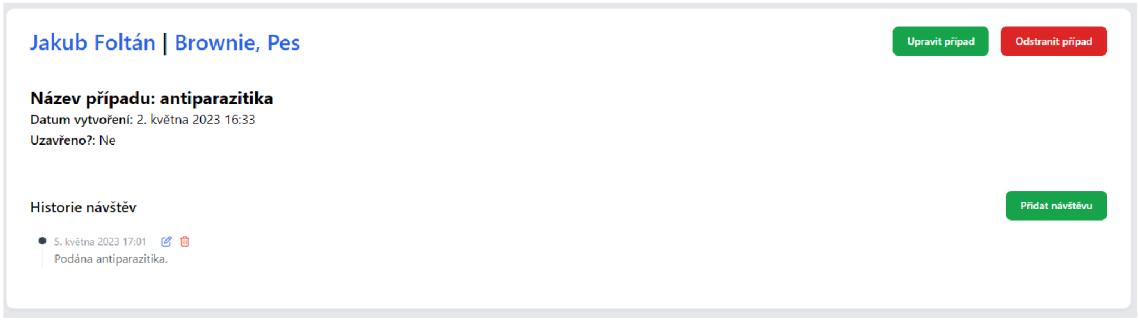


Obr. 29: Rozhraní pro seznam majitelů

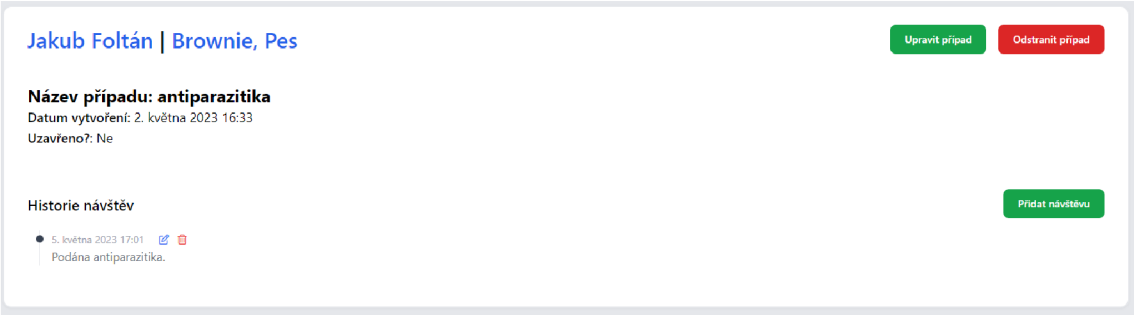


Obr. 30: Rozhraní pro detail majitele

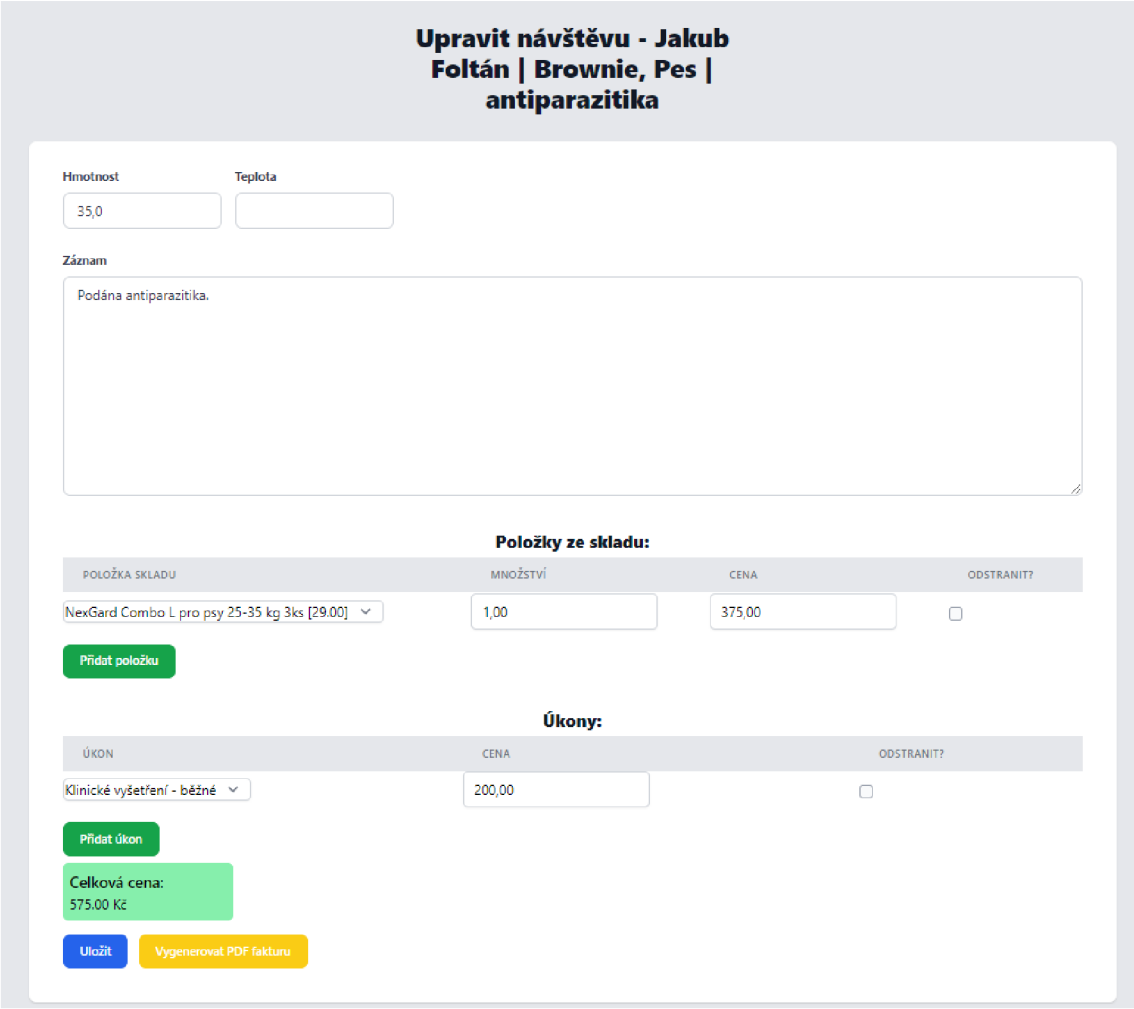
Obr. 31: Rozhraní pro detail zvířete



Obr. 32: Rozhraní pro detail případu



Obr. 33: Rozhraní pro detail případu



Obr. 34: Rozhraní pro formulář návštěvy

5.9.2 Aplikace pro sklad

Aplikace sklad je další zásadní částí. Tato aplikace umožňuje správu skladů a skladových položek.

Modely

V rámci modelů pro aplikaci skladu jsou vytvořeny dva objekty. Prvním je objekt *Stock*, který reprezentuje jednotlivé sklady. Je u nich možné evidovat název, jednotnou marži pro sklad a poznámku. V případě objektu *StockItem* jde o reprezentaci skladových položek. U nich jsou evidovány všechny potřebné údaje pro evidenci, účtování, atd. Výsledný kód pro modely lze vidět na obr. 35.

```
7   class Stock(models.Model):
8       name = models.CharField(max_length=50)
9       margin = models.IntegerField(validators=[MinValueValidator(0)])
10      note = models.TextField(blank=True, null=True)
11
12      def __str__(self):
13          return self.name
14          You, 3 weeks ago • add stockitems_list, create stockitem
15
    You, 3 weeks ago | 1 author (You)
16  class StockItem(models.Model):
17      TYPE_CHOICES = [
18          ("CURE", "Lék"),
19          ("MATERIAL", "Materiál"),
20          ("OTHER", "Ostatní"),
21      ]
22
23      stock = models.ForeignKey(Stock, on_delete=models.CASCADE, related_name="items")
24      name = models.CharField(max_length=200)
25      type = models.CharField(max_length=10, choices=TYPE_CHOICES, blank=True, null=True)
26      supplier_code = models.CharField(max_length=50, blank=True, null=True)
27      supplier = models.CharField(max_length=200, blank=True, null=True)
28      expiration_date = models.DateField(blank=True, null=True)
29      ean = models.CharField(max_length=50, blank=True, null=True)
30      batch = models.CharField(max_length=50, blank=True, null=True)
31      units_in_stock = models.DecimalField(max_digits=10, decimal_places=2)
32      units_in_package = models.DecimalField(max_digits=10, decimal_places=2)
33      purchase_price = models.DecimalField(max_digits=10, decimal_places=2)
34      selling_price = models.DecimalField(max_digits=10, decimal_places=2)
35      margin = models.IntegerField(validators=[MinValueValidator(0)])
36      note = models.TextField(blank=True, null=True)
37
38      def __str__(self):
39          return self.name + " [" + str(self.units_in_stock) + "]"
40
```

Obr. 35: Kód reprezentující modely pro sklad a skladové položky

Views

V rámci views je možné provádět běžné CRUD operace podobně jako v případě aplikace pro kartotéku. Pro zobrazení skladových položek je v aplikaci možné filtrovat položky dle skladu nebo typu položky. K tomu je použito filtrování, jehož kód lze vidět na obr. 36.

```python
@login_required
def stock(request, stock_id=None):
    items = StockItem.objects.all()

    # Filtrace podle skladu, pokud je zvolen
    if stock_id is not None:
        items = items.filter(stock_id=stock_id)

    # Filtrace podle typu, pokud je zvolen
    type = request.GET.get("type")
    if type:
        items = items.filter(type=type)      You, 3 weeks ago • add stockitems_list,

    stocks = Stock.objects.all()

    unique_types = set(item.type for item in StockItem.objects.all())

    # Převod kódů typů na lidsky čitelné hodnoty
    type_choices_dict = dict(StockItem.TYPE_CHOICES)
    types = {
        unique_type: type_choices_dict[unique_type] for unique_type in unique_types
    }

    context = {
        "items": items,
        "stocks": stocks,
        "choosen_stock": stock_id,
        "types": types,
        "choosen_type": type,
    }

    return render(request, "stock/stock_items.html", context)
```

Obr. 36: Kód pro zobrazení skladových položek a jejich filtraci

Šablony a rozhraní aplikace

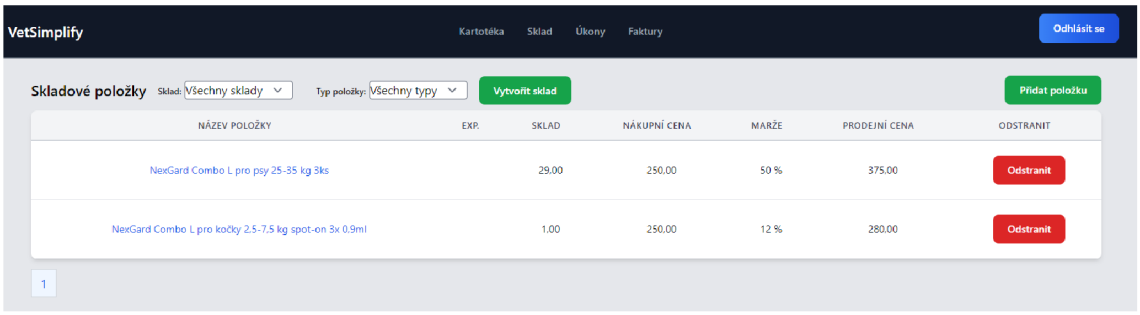
Pro aplikaci skladu byly vytvořeny všechny potřebné šablony pro zobrazení seznamu skladových položek s filtrací, vytvoření a úpravu skladu a také skladových položek. Pro filtrování skladových položek podle toho, v jakém skladu jsou nebo o jaký typ položky se jedná bylo použito JavaScript kódu (obr. 37).

```javascript
<script>
        document.addEventListener('DOMContentLoaded', function() {
          const stockSelect = document.getElementById('stock');
          const typeSelect = document.getElementById('type');

          function updateUrl() {
            let url = '{% url "stock:stock" %}';
            if (stockSelect.value) {
              url = '{% url "stock:stock_items_by_stock" stock_id=0 %}'.replace('0', stockSelect.value);
            }
            if (typeSelect.value) {
              url += '?type=' + encodeURIComponent(typeSelect.value);
            }
            window.location.href = url;
          }

          stockSelect.addEventListener('change', updateUrl);
          typeSelect.addEventListener('change', updateUrl);
        });
</script>
```

Obr. 37: JavaScript kód pro filtraci skladových položek

Výsledné rozhraní pro zobrazení seznamu skladových položek spolu s filtrací je možné vidět na obr. 38. Dále byl využit JavaScript při vytváření nebo editaci

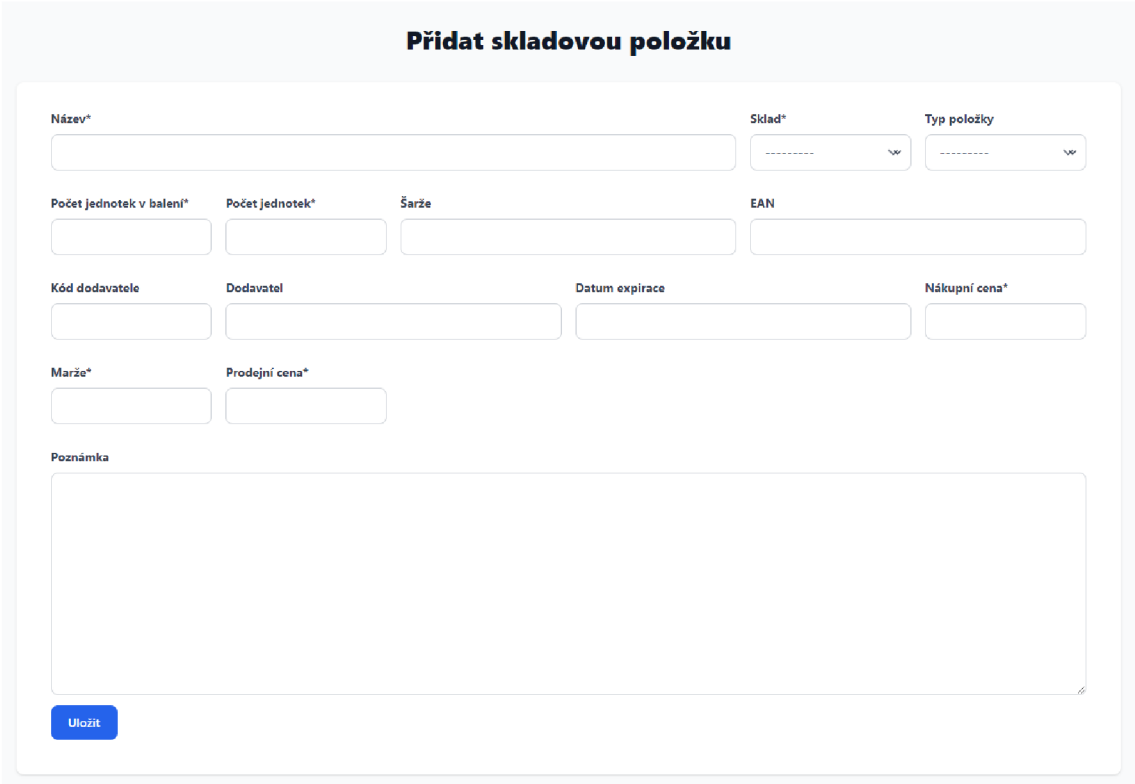
skladových položek, a to konkrétně pro výpočet prodejní ceny na základě zadané nákupní ceny a marže (obr. 39). Rozhraní, které slouží pro vytváření skladových položek lze vidět na obr. 40.



Obr. 38: Rozhraní pro skladové položky

```
<script>
    function updateSellingPrice() {
        const purchasePriceField = document.getElementById("id_purchase_price");
        const marginField = document.getElementById("id_margin");
        const sellingPriceField = document.getElementById("id_selling_price");

        if (purchasePriceField.value && marginField.value) {
            const purchasePrice = parseFloat(purchasePriceField.value);
            const margin = parseFloat(marginField.value);
            const sellingPrice = purchasePrice * (1 + (margin / 100));
            sellingPriceField.value = sellingPrice.toFixed(2);
        } else {
            sellingPriceField.value = "";
        }
    }

    document.getElementById("id_purchase_price").addEventListener("input", updateSellingPrice);
    document.getElementById("id_margin").addEventListener("input", updateSellingPrice);
</script>
```

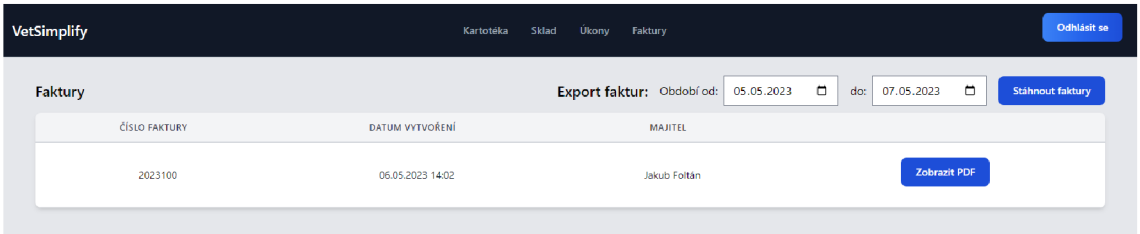
Obr. 39: JavaScript kód pro výpočet prodejní ceny



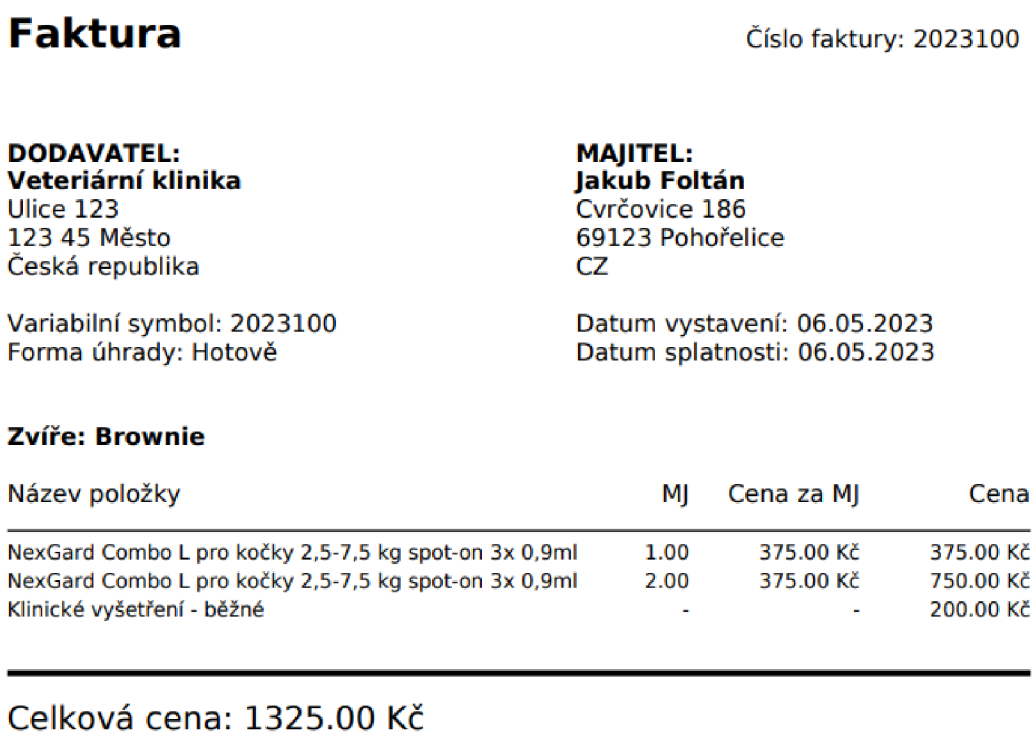
Obr. 40: Rozhraní pro vytvoření skladové položky

5.9.3 Aplikace pro faktury

Pro potřeby účetnictví je potřeba, aby program poskytoval možnost vytvořit fakturu k návštěvě, uložit ji a exportovat všechny faktury za určité časové období. Pro generování faktur byla použita knihovna Reportlab. Faktura je pomocí ReportLab vygenerována a následně uložena do databáze jako binární soubor.



Obr. 41: Rozhraní pro zobrazení faktur



Faktura

Číslo faktury: 2023100

DODAVATEL:
Veteriární klinika
Ulice 123
123 45 Město
Česká republika

MAJITEL:
Jakub Foltán
Cvrčovice 186
69123 Pohořelice
CZ

Variabilní symbol: 2023100
Forma úhrady: Hotově

Datum vystavení: 06.05.2023
Datum splatnosti: 06.05.2023

Zvíře: Brownie

Název položky	MJ	Cena za MJ	Cena
NexGard Combo L pro kočky 2,5-7,5 kg spot-on 3x 0,9ml	1.00	375.00 Kč	375.00 Kč
NexGard Combo L pro kočky 2,5-7,5 kg spot-on 3x 0,9ml	2.00	375.00 Kč	750.00 Kč
Klinické vyšetření - běžné	-	-	200.00 Kč

Celková cena: 1325.00 Kč

Obr. 42: Ukázka vygenerované faktury

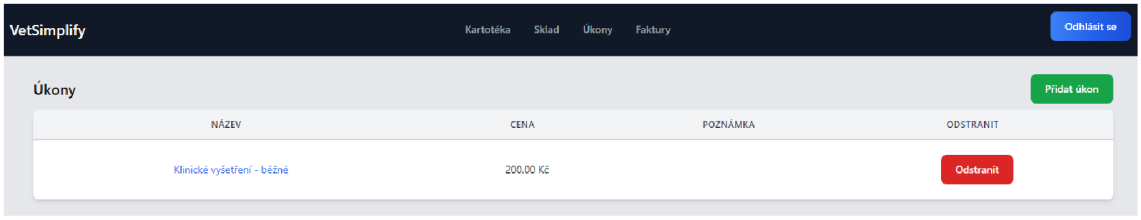
Aplikace dále umožňuje export faktur za určité časové období. Tato funkce umožňuje projít databázi všech faktur a podle zvoleného časového období zabalí všechny požadované PDF soubory do jednoho zip archivu. Kód pro export faktur lze vidět na obr. 43.

```python
@login_required
def download_invoices(request):
    start_date = request.GET.get("start_date")
    end_date = request.GET.get("end_date")

    if start_date is None or end_date is None:
        return HttpResponse("Missing start_date or end_date parameter.", status=400)

    try:
        start_date = datetime.strptime(start_date, "%Y-%m-%d")
        end_date = datetime.strptime(end_date, "%Y-%m-%d")
    except ValueError:
        return HttpResponse("Incorrect data format, should be YYYY-MM-DD", status=400)

    invoices = Invoice.objects.filter(
        Q(generated_at__gte=start_date) & Q(generated_at__lte=end_date)
    )

    zip_filename = (
        f"faktury_{start_date.strftime('%Y-%m-%d')}_{end_date.strftime('%Y-%m-%d')}.zip"
    )
    zip_path = os.path.join(settings.MEDIA_ROOT, zip_filename)

    # Vytvoření složky MEDIA_ROOT, pokud neexistuje
    os.makedirs(os.path.dirname(zip_path), exist_ok=True)

    with zipfile.ZipFile(zip_path, "w") as zip_file:
        for invoice in invoices:
            pdf_content = invoice.content      You, 9 minutes ago • Uncommitted changes
            pdf_filename = f"faktura_{invoice.visit.pk}_{invoice.generated_at.strftime('%Y-%m-%d')}.pdf"
            zip_file.writestr(pdf_filename, pdf_content)

    zip_file = open(zip_path, "rb")

    response = FileResponse(zip_file)
    response["Content-Type"] = "application/zip"
    response["Content-Disposition"] = f"attachment; filename={zip_filename}"

    return response
```

Obr. 43: Kód pro export faktur

5.9.4 Aplikace úkony

Poslední částí aplikace je seznam úkonů. Tato aplikace umožňuje evidovat úkony, které jsou potom přiřazovány k návštěvám. Nad těmito úkony lze provádět běžné CRUD operace. Jelikož je tato aplikace velmi jednoduchá a nezahrnuje žádné specifické části, které by již nebyly popsány dříve, nebude dále rozebírána.



Obr. 44: Rozhraní pro úkony

5.10 Autentizace

Django poskytuje robustní a bezpečný systém pro autentizaci uživatelů. Základem systému autentizace je model *User*, který obsahuje políčka pro uživatelské jméno, heslo, e-mail a další informace. Tento model může být v případě potřeby rozšířen. Navíc Django poskytuje funkce *login* a *logout*. Funkce *login* vezme požadavek a instanci modelu *User* a přihlásí uživatele. Po úspěšném přihlášení je uživatelova session uložena do databáze a je mu nastaven cookie, aby byl uživatel při dalších návštěvách rozpoznán. Na druhé straně, funkce *logout* odstraní uživatelovu session z databáze a odstraní jeho cookie, čímž uživatele odhlásí.

Zároveň Django poskytuje formulář *UserCreationForm* určený pro vytváření nových uživatelů. Tento formulář automaticky zajišťuje základní validaci, jako je ověření, zda hesla v obou polích jsou stejná a uživatelské jméno je jedinečné.

Zatímco funkce pro přihlášení je již vestavěná, pro registraci je potřeba vytvořit vlastní funkci, která využívá formulář *UserCreationForm*. Kód funkce, která vytváří uživatele je zobrazena na obr. 45.

accounts > views.py > ...

You, 4 weeks ago | 1 author (You)

```python
from django.shortcuts import render, redirect
from django.contrib.auth import login
from .forms import RegistrationForm

# Create your views here.


def sign_up(request):
    if request.method == "POST":
        form = RegistrationForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            return redirect("home")
    else:
        form = RegistrationForm()

    return render(request, "registration/sign_up.html", {"form": form})
```

Obr. 45: Konfigurace statických souborů

Pro zajištění přístupu jen pro přihlášené uživatele k jednotlivým částem aplikace lze použít dva přístupy. V případě CBV lze použít mixin **`LoginRequiredMixin`**, který zajistí, že pohled je přístupný pouze pro přihlášené uživatele. V případě FBV lze použít dekorátor **`login_required`**. Tento postup zajistí, že nepřihlášený uživatel, který se bude snažit přistoupit k dané stránce bude přesměrován na stránku pro přihlášení. Použití autentizace pro views lze vidět na obr. 46.

```
64  class OwnerListView(LoginRequiredMixin, ListView):
65      model = Owner
66      paginate_by = 10

245 @login_required
246 def visit_create_view(request, owner_id, animal_id, animalcase_id):
247     animal_case = get_animalcase(owner_id, animal_id, animalcase_id)
```

Obr. 46: Autentizace v CBV a FBV

6 ZÁVĚR

V rámci této práce bylo úkolem analyzovat požadavky na provoz veterinární ordinace. Tato část byla konzultována s Veterinární ordinací Mája Oslavany. Tato analýza se zabývala hledáním řešení pro nevyhovující stav současného softwaru, který ordinace aktuálně používá. Byly tedy stanoveny přednosti i nedostatky současného softwaru pro snadnější definici požadavků na nový software. Zároveň došlo k průzkumu místního trhu pro již existující řešení, ale žádné z nich nebylo plně vyhovující pro potřeby klienta. Proto bylo přistoupeno ke zhotovení vlastního řešení a to konkrétně webové aplikace, která nejlépe splňuje požadavky klienta na přístup z jakéhokoliv zařízení a možnost provozu aplikace z více míst současně.

V části týkající se technologií pro vývoj webových aplikací byl proveden detailní průzkum historie, základních konceptů a technologií webových aplikací. Byla definována podstata webových aplikací a zdůrazněna jejich důležitost a výhody v dnešní digitalizované společnosti. Byly také vysvětleny základní koncepty webových aplikací, včetně klient-server modelu, protokolů HTTP a HTTPS a architektury MVC. Tato část slouží jako základ pro pochopení, jak webové aplikace fungují a jak se data přenášejí mezi klientem a serverem. V kapitolách týkajících se frontend a backend technologií byly popsány klíčové technologie používané v moderním vývoji webových aplikací. Bylo diskutováno HTML, CSS a JavaScript na straně frontendu a různé jazyky a frameworky na straně backendu, včetně databází.

V části týkající se implementace webové aplikace pro správu veterinární ordinace byla představena funkcionalita aplikace, použité technologie, návrh databáze, nastavení vývojového prostředí, nastavení projektu a samotná implementace aplikace. Hlavními funkcemi aplikace je správa kartotéky, skladu, veterinárních úkonů a fakturace. Zároveň je zajištěno řízení přístupu a autentizace uživatelů.

Aplikace byla vyvinuta s využitím Django frameworku pro backend, PostgreSQL pro databázovou vrstvu a pro frontend bylo využito Django šablon spolu s JavaScriptem a CSS frameworkem Tailwind CSS. Návrh databáze byl pečlivě promyšlen, aby reflektoval potřeby aplikace a umožňoval efektivní manipulaci s daty. Návrh databáze zohledňoval požadavky na efektivitu, škálovatelnost a bezpečnost. Bylo provedeno nastavení vývojového prostředí a byl vytvořen Django projekt s důrazem na optimální rozdělení na jednotlivé aplikace. Adresářová struktura projektu byla navržena tak, aby byla co nejvíce přehledná a logická. V souboru settings.py byla provedena konfigurace projektu včetně nastavení databáze, umístění šablon, lokalizace a časové zóny, nainstalovaných aplikací a konfigurace statických souborů. Implementace aplikace zahrnovala několik klíčových komponent, včetně kartotéky, aplikace pro správu skladu, aplikace pro faktury a aplikace pro veterinární úkony.

Každá z těchto částí byla pečlivě navržena a implementována, aby splňovala požadavky na funkčnost, uživatelskou přívětivost a bezpečnost.

Po zhotovení aplikace byla otestována v rámci provozu Veterinární ordinace Mája Oslavany, kde bylo potvrzeno, že aplikace splňuje požadavky a očekávání klienta. Aplikace byla schopná účinně řešit nedostatky původního softwaru a zároveň poskytovat nové funkce a možnosti, které výrazně zlepšují provoz a efektivitu ordinace. Přestože je výsledná aplikace plně funkční a splňuje všechny požadavky stanovené v počáteční fázi projektu, je zde několik možností pro další vývoj a vylepšení. V budoucnu by aplikace mohla zahrnovat další funkce, jako je automatický import skladových položek, přidání možnosti nastavení pro odlišnosti dané veterinární ordinace, nebo o přidání uživatelských rolí.

Tato práce ukázala, že vývoj vlastní webové aplikace může být účinným řešením pro specifické potřeby klienta, které nemohou být plně pokryty existujícími produkty na trhu. Výsledná aplikace nejenže zlepšuje každodenní provoz veterinární ordinace, ale také poskytuje platformu pro budoucí vývoj a vylepšení, která mohou ještě více zvýšit efektivitu a kvalitu služeb poskytovaných klientům.

SEZNAM POUŽITÉ LITERATURY

[1] *Vetis office*. Dostupné z: https://vetis.cz/.

[2] *WinVet*. Dostupné z: https://www.winvet.cz/.

[3] *WinVet - Popis programu*. Dostupné z: https://www.winvet.cz/index.php/winvet.

[4] *Winvet - Ceník*. Dostupné z: https://www.winvet.cz/index.php/cenik.

[5] *VetPro v6.x – Veterinární informační systém | MedSoft s.r.o.* Dostupné z: https://medsoftsro.cz/vetpro-veterinarni-informacni-system/.

[6] *VetPro – veterinární informační systém | MedSoft s.r.o.* Dostupné z: https://medsoftsro.cz/vetpro-veterinarni-informacni-system/vetpro-veterinarni-software/.

[7] *Ceník – VetPro | MedSoft s.r.o.* Dostupné z: https://medsoftsro.cz/vetpro-veterinarni-informacni-system/cenik-vetpro/.

[8] WWW.OBSAH.INFO, V. *Vetbook*. Dostupné z: https://www.vetbook.cz/about.php.

[9] WWW.OBSAH.INFO, V. *Vetbook - Ceník*. Dostupné z: https://www.vetbook.cz/cenik.php.

[10] WWW.OBSAH.INFO, V. *Vetbook*. Dostupné z: https://www.vetbook.cz/demo.php.

[11] *Vetfox – Jednoduchá aplikace pro veterinární lékaře*. Dostupné z: http://www.vetfox.cz/.

[12] *Ceník | Vetfox – Jednoduchá aplikace pro veterinární lékaře*. Dostupné z: http://www.vetfox.cz/cenik.

[13] *What is the Difference Between a Website and a Web Application?* Březen 2021. Dostupné z: https://www.freecodecamp.org/news/difference-between-a-website-and-a-web-application/.

[14] *Client-Server Overview - Learn web development | MDN*. únor 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview.

[15] *Difference between Server-side Scripting and Client-side Scripting - javatpoint*. Dostupné z: https://www.javatpoint.com/server-side-scripting-vs-client-side-scripting.

[16] KHAMOOSHI, P. *The benefits of using web-based applications* [online]. 2019 [cit. 2023-05-14]. Dostupné z: https://www.geeks.ltd.uk/insights/blog/the-benefits-of-using-web-based-applications.

[17] *A short history of the Web*. Březen 2023. Dostupné z: https://home.cern/science/computing/birth-web/short-history-web.

[18] ZHENG, J. a LEE, D. K. C. *Understanding the Evolution of the Internet: Web1.0 to Web3.0, Web3 and Web 3*. Rochester, NY: [b.n.], duben 2023. DOI: 10.2139/ssrn.4431284. Dostupné z: https://papers.ssrn.com/abstract=4431284.

[19] DINUCCI, D. Fragmented future. *Print*. RC Publications Inc. 1999, sv. 53, č. 4, s. 32. Dostupné z: http://darcyd.com/fragmented_future.pdf.

[20] HLOUŠEK, M. *Web 2,0 - nové možnosti internetu*. 2009. Thesis. Technická Univerzita v Liberci. Accepted: 2013-12-27 Journal Abbreviation: Web 2,0 - new opportunities of internet. Dostupné z: https://dspace.tul.cz/handle/15240/5252.

[21] AMBROŽ, J. *Web 2.0: bublina, nebo nový směr webu?* ISSN: 1213-0702. Dostupné z: https://www.lupa.cz/clanky/web-2-0-bublina-nebo-novy-smer-webu/.

[22] GOEL, A. K., BAKSHI, R. a AGRAWAL, K. K. Web 3.0 and Decentralized Applications. *Materials Proceedings*. 2022, sv. 10, č. 1, s. 8. DOI: 10.3390/materproc2022010008. ISSN 2673-4605. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute. Dostupné z: https://www.mdpi.com/2673-4605/10/1/8.

[23] *Client-Server Model*. říjen 2019. Section: Web Technologies. Dostupné z: https://www.geeksforgeeks.org/client-server-model/.

[24] *HTTP (Hypertext Transfer Protocol): An Overview of the Internet's Most Widely Used Protocol*. Dostupné z: https://piembsystech.com/http-hypertext-transfer-protocol-an-overview-of-the-internets-most-widely-used-protocol/.

[25] UNIVERSITY, G. *Hypertext Transfer Protocol (HTTP) | CCNA#*. Dostupné z: https://geek-university.com/hypertext-transfer-protocol-http/,https://geek-university.com/hypertext-transfer-protocol-http/.

[26] *HTTP request methods - HTTP | MDN*. Duben 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.

[27] *What is HTTP? Protocol Overview for Beginners*. Duben 2023. Dostupné z: https://www.freecodecamp.org/news/what-is-http/.

[28] *In Introduction to HTTP Basics*. Dostupné z: https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/http_basics.html.

[29] *100 Continue - HTTP | MDN*. Duben 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/100.

[30] *101 Switching Protocols - HTTP | MDN*. Duben 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/101.

[31] *200 OK - HTTP | MDN*. Duben 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200.

[32] *201 Created - HTTP | MDN*. Duben 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/201.

[33] *202 Accepted - HTTP | MDN*. Duben 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/202.

[34] *203 Non-Authoritative Information - HTTP | MDN*. Duben 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/203.

[35] *301 Moved Permanently - HTTP | MDN*. Duben 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/301.

[36] *404 Not Found - HTTP | MDN*. Duben 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404.

[37] *500 Internal Server Error - HTTP | MDN*. Duben 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/500.

[38] *Why is HTTP not secure? | HTTP vs. HTTPS*. Dostupné z: https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/.

[39] Čápka, D. *MVC architektura*. Dostupné z: https://www.itnetwork.cz/mvc-architektura-navrhovy-vzor.

[40] Marek, L. *Tvorba www stránek v HTML a CSS*. Grada Publishing, srpen 2019.

[41] *Overview of HTML*. Dostupné z: https://web.dev/learn/html/overview/.

[42] LAWRENCE, M. *What is a CSS Framework?* Květen 2019. Dostupné z: https://medium.com/html-all-the-things/what-is-a-css-framework-f758ef0b1a11.

[43] CONTRIBUTORS, a. B. M. O. *Bootstrap*. Dostupné z: https://getbootstrap.com/.

[44] *The most advanced responsive front-end framework in the world. | Foundation*. Dostupné z: https://get.foundation/.

[45] *Bulma: Free, open source, and modern CSS framework based on Flexbox*. Dostupné z: https://bulma.io.

[46] *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML*. Listopad 2020. Dostupné z: https://tailwindcss.com/.

[47] *Flowbite - Build websites even faster with components on top of Tailwind CSS*. Dostupné z: https://flowbite.com.

[48] *JavaScript | MDN*. Květen 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[49] *Learn JavaScript Tutorial - javatpoint*. Dostupné z: https://www.javatpoint.com/javascript-tutorial.

[50] MÁCA, J. *Lekce 1 - Úvod do Node.js*. Dostupné z: https://www.itnetwork.cz/uvod-do-nodejs.

[51] *10 Best JavaScript Frameworks to Use in 2023 [Recommended]*. Dostupné z: https://hackr.io/blog/best-javascript-frameworks.

[52] *React*. Dostupné z: https://react.dev/.

[53] *Angular*. Dostupné z: https://angular.io/.

[54] *Vue.js - The Progressive JavaScript Framework | Vue.js*. Dostupné z: https://vuejs.org/.

[55] *The Python Tutorial*. Dostupné z: https://docs.python.org/3/tutorial/index.html.

[56] PAYNE, J. *Benefits of Python Programming Language*. Duben 2022. Dostupné z: https://www.developer.com/languages/python/python-benefits/.

[57] ČÁPKA, D. *Lekce 3 - Představení MVC a MVT architektury v Django*. Dostupné z: https://www.itnetwork.cz/predstaveni-mvc-a-mvt-architektury-v-django.

[58] *Django*. Dostupné z: https://www.djangoproject.com/.

[59] *Welcome to Flask — Flask Documentation (2.3.x)*. Dostupné z: https://flask.palletsprojects.com/en/2.3.x/.

[60] *Python Types Intro - FastAPI*. Dostupné z: https://fastapi.tiangolo.com/python-types/.

[61] *FastAPI*. Dostupné z: https://fastapi.tiangolo.com/.

[62] *Co je Java? – Příručka pro začátečníky v jazyce Java | Microsoft Azure*. Dostupné z: https://azure.microsoft.com/cs-cz/resources/cloud-computing-dictionary/what-is-java-programming-language/.

[63] *Home*. Dostupné z: https://spring.io/.

[64] *Welcome to the Apache Struts project*. Dostupné z: https://struts.apache.org/.

[65] *Dynamic typing - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. únor 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Dynamic_typing.

[66] *Node Js: Non-blocking or asynchronous | Blocking or synchronous*. Květen 2019. Dostupné z: https://www.cronj.com/blog/node-js-non-blocking-asynchronous-blocking-synchronous/.

[67] *Express - Node.js web application framework*. Dostupné z: https://expressjs.com/.

[68] *NestJS - A progressive Node.js framework*. Dostupné z: https://nestjs.com.

[69] *PHP | Introduction*. únor 2018. Section: PHP. Dostupné z: https://www.geeksforgeeks.org/php-introduction/.

[70] *Laravel - The PHP Framework For Web Artisans*. Dostupné z: https://laravel.com/.

[71] SYMFONY. *Symfony, High Performance PHP Framework for Web Development*. Dostupné z: https://symfony.com/.

[72] HTTPS://DAVIDGRUDL.COM, D. G. *Nette – Pohodlný a bezpečný vývoj webových aplikací v PHP*. Dostupné z: https://nette.org/cs/.

[73] *About Ruby*. Dostupné z: https://www.ruby-lang.org/en/about/.

[74] *Ruby on Rails*. Dostupné z: https://rubyonrails.org/.

[75] Łata, P. *What is Sinatra Ruby? How to build applications using Sinatra?* Srpen 2022. Dostupné z: https://blog.railwaymen.org/what-is-sinatra-ruby.

[76] *Hanami | The web, with simplicity.* Dostupné z: https://hanamirb.org/.

[77] BillWagner. *A tour of C# - Overview.* Květen 2023. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

[78] *Nancy - Lightweight Web Framework for .net.* Dostupné z: https://nancyfx.org/.

[79] tdykstra. *Overview of ASP.NET Core.* Listopad 2022. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core.

[80] Marek, L. *SQL: Podrobný průvodce uživatele.* Grada Publishing, říjen 2018.

[81] Smallcombe, M. *SQL vs NoSQL: 5 Critical Differences.* Dostupné z: https://www.integrate.io/blog/the-sql-vs-nosql-difference/.

[82] *What is MySQL?* Dostupné z: https://www.oracle.com/mysql/what-is-mysql/.

[83] *PostgreSQL: About.* Dostupné z: https://www.postgresql.org/about/.

[84] *Microsoft SQL Server.* Leden 2023. Page Version ID: 22288506. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Microsoft_SQL_Server&oldid=22288506.

[85] *MongoDB: The Developer Data Platform.* Dostupné z: https://www.mongodb.com.

[86] *Redis.* Dostupné z: https://redis.io/.

[87] *Neo4j Graph Database & Analytics – The Leader in Graph Databases.* Dostupné z: https://neo4j.com/.

[88] Atlassian. *What is version control | Atlassian Git Tutorial.* Dostupné z: https://www.atlassian.com/git/tutorials/what-is-version-control.

[89] Atlassian. *What is Git | Atlassian Git Tutorial.* Dostupné z: https://www.atlassian.com/git/tutorials/what-is-git.

[90] *GitHub.* Květen 2023. Page Version ID: 1155858740. Dostupné z: https://en.wikipedia.org/w/index.php?title=GitHub&oldid=1155858740.

[91] Mele, A. a Belderbos, B. *Django 4 By Example*. 4. vyd. Birmingham, England: Packt Publishing, září 2022.

[92] *Django*. Dostupné z: https://docs.djangoproject.com/en/4.2/faq/install/.

[93] *Python - Visual Studio Marketplace*. Dostupné z: https://marketplace.visualstudio.com/items?itemName=ms-python.python.

[94] *Django - Visual Studio Marketplace*. Dostupné z: https://marketplace.visualstudio.com/items?itemName=batisteo.vscode-django.

[95] *djLint - Visual Studio Marketplace*. Dostupné z: https://marketplace.visualstudio.com/items?itemName=monosans.djlint.

[96] *Tailwind CSS IntelliSense - Visual Studio Marketplace*. Dostupné z: https://marketplace.visualstudio.com/items?itemName=bradlc.vscode-tailwindcss.

[97] *Set up Git*. Dostupné z: https://ghdocs-prod.azurewebsites.net/en/get-started/quickstart/set-up-git.

[98] *What is Web Application (Web Apps) and its Benefits*. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app.

[99] *Web Applications*. Dostupné z: https://cs.lmu.edu/~ray/notes/webapps/.

[100] *What is CSS? - Learn web development | MDN*. únor 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

- Archiv VetSimplify.zip obsahující Django projekt pro vytvořenou webovou aplikaci.

Aplikace je také dostupná na GitHubu: https://github.com/jfoltan/VetSimplify