Webová aplikace pro efektivní kontrolu příjmů a výdajů

Leon Holub

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Leon Holub**
Osobní číslo:	**A21328**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Webová aplikace pro efektivní kontrolu příjmů a výdajů**
Téma práce anglicky:	**Web Application for Efficient Control of Income and Expenses**

Zásady pro vypracování

1. Rozepište terminologii v kontextu tématu práce.
2. Rozveďte aktuálně existující řešení.
3. Navrhněte vlastní řešení a zvolte vhodné technologie pro implementaci.
4. Implementujte vlastní řešení.
5. Vhodně výslednou aplikaci otestujte.
6. Popište dosažené výsledky a porovnejte s existujícími řešeními.

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

Seznam doporučené literatury:

1. GRUDL, David. Pohodlný a bezpečný vývoj webových aplikací v PHP. Nette [online]. 2008, 2023 [cit. 2023-11-03]. Dostupné z: https://nette.org/ – Get started with Bootstrap · Bootstrap v5.3.
2. OTTO, Mark a Jacob THORNTON. Bootstrap [online]. 2023. Dostupné z: https://getbootstrap.com/docs/5.3/getting-started/introduction/
3. BHAGAT, Varun. PHP Vs ASP.NET: How to Choose the Right One? In: PixelCrayons [online]. Dostupné z: https://www.pixelcrayons.com/blog/php-vs-asp-net-how-to-choose-the-right-one/
4. SCHWABISH, Jonathan. Better Data Visualizations: A Guide for Scholars, Researchers, and Wonks. Columbia University Press, 2021. ISBN 9780231193115.
5. MERCHANT, Amit D. PHP 8 in a Nutshell: Your no-fluff guide to all the new things in PHP 8, 8.1, and 8.2. Amit D. Merchant, 2022.

Vedoucí bakalářské práce: **Ing. Petr Žáček, Ph.D.**
Ústav informatiky a umělé inteligence

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

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

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

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

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

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

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.
- Prohlašuji, že při tvorbě této práce jsem použil/a nástroj generativního modelu AI ChatGPT od společnosti OpenAI, dostupné na https://www.openai.com/chatgpt, za účelem úpravy textů, asistence s citacemi a s návrhem názvu aplikace. Po použití tohoto nástroje jsem provedl kontrolu obsahu a přebírám za něj plnou zodpovědnost.

Ve Zlíně, dne

Leon Holub, v. r.
…….
podpis studenta

ABSTRAKT

Tato bakalářská práce se zaměřuje na návrh a implementaci webové aplikace ExpenseInsight. Aplikace umožňuje uživatelům efektivně spravovat příjmy a výdaje bez nutnosti platit za základní funkce. Cílem bylo vytvořit intuitivní, bezpečnou a škálovatelnou platformu využívající technologie Nette frameworku a JavaScriptové knihovny Naja. Práce obsahuje průzkum stávajících řešení, návrh, implementaci a testování aplikace, stejně jako hodnocení výsledků s důrazem na technickou funkčnost a uživatelskou přívětivost. ExpenseInsight přináší pokročilé funkce jako opakované transakce, sdílení účtů a další, které budou v práci popsány.

Klíčová slova:

správa financí, PHP, Nette framework, Nextras, webová aplikace, javascript

ABSTRACT

This bachelor's thesis focuses on designing and implementing the web application ExpenseInsight. The application enables users to efficiently manage their income and expenses without paying for basic features. The aim was to develop an intuitive, secure, and scalable platform utilizing Nette framework technologies and the Naja JavaScript library. The thesis includes a survey of existing solutions, design, implementation, and testing of the application, as well as evaluation of the outcomes emphasizing technical functionality and user friendliness. ExpenseInsight introduces advanced features such as recurring transactions and account sharing, which are further described in the work.

Keywords:

financial management, PHP, Nette framework, Nextras, web application, JavaScript

Na tomto místě bych rád vyjádřil svou upřímnou vděčnost a poděkování všem, kteří mi poskytli podporu během mého studia a práce na bakalářské práci.

Velké poděkování patří mé rodině, jejíž neustálá podpora pro mě byla klíčová nejen během psaní této práce, ale i po celou dobu studia.

Velké díky patří panu Ing. Petru Žáčkovi, Ph.D., jehož vedení a připomínky mi byly užitečné pro zdokonalení této práce.

Neméně důležitá byla podpora mé přítelkyně a přátel, kteří mě inspirovali svými nápady a připomínkami k tomu, co by si přáli vidět ve výsledném produktu, a pomohli tak lépe vystihnout potřeby uživatelů.

Zvláštní díky patří všem, kteří se zapojili do testování mé práce. Vaše zpětná vazba byla neocenitelná pro další vývoj a zlepšování.

OBSAH

ÚVOD

Před čtyřmi lety jsem jako první krok vytvořil jednoduchou mobilní aplikaci pro Android, která mi umožnila sledovat zůstatek v hotovosti. Ačkoliv tato aplikace byla funkční a užitečná, byla limitovaná svými možnostmi a nedostatkem funkcí, jako je kategorizace výdajů či příjmů. Tato zkušenost mne inspirovala k hlubšímu zkoumání oblasti osobních financí a vedla k rozhodnutí vypracovat bakalářskou práci zaměřenou na vytvoření sofistikovanější webové aplikace pro efektivní správu financí.

Cílem této práce je navrhnout a implementovat webovou aplikaci, která umožní uživatelům efektivně spravovat jejich příjmy a výdaje, aniž by byli omezováni nutností platit za základní funkcionality. Aplikace bude navržena s důrazem na uživatelskou přívětivost, bezpečnost a škálovatelnost, s použitím moderních technologií a metodik, včetně frameworku Nette a JavaScriptové knihovny pro interaktivní prvky.

Tato bakalářská práce obsahuje teoretickou část, která se věnuje webovým aplikacím a PHP frameworkům jako jsou Laravel a Symfony, s hlavním důrazem na framework Nette. V této části je Nette podrobně rozebrán, aby bylo možné lépe pochopit jeho vlastnosti.

Praktická část této bakalářské práce zahrnuje průzkum aktuálně existujících řešení, návrh vlastního řešení, jeho implementaci a následné otestování. Hlavní inspirací pro mé řešení byla aplikace Wallet, jejíž pokročilé funkce jsou často přístupné pouze v placené verzi. Proto bylo klíčové začlenit do aplikace ExpenseInsight pokročilé funkce, jako je opakované provádění transakcí, sdílení účtů mezi více uživateli a možnost tvorby a používání šablon pro transakce, a to vše bezplatně.

Výsledkem této bakalářské práce je plně funkční prototyp webové aplikace ExpenseInsight, dostupný na https://expenseinsight.leonholub.cz. Tento prototyp poskytuje solidní základ pro budoucí rozvoj a možné rozšíření funkcionalit.

I. TEORETICKÁ ČÁST

1 ZÁKLADNÍ TERMINOLOGIE WEBOVÝCH APLIKACÍ

Webová aplikace představuje software, který je dostupný přes internetový prohlížeč, což znamená, že není nutná jeho instalace na pevný disk počítače. Stačí zadat adresu v prohlížeči, jako je Opera, Google Chrome nebo Safari, a můžete ji okamžitě používat.

V poslední době se vývoj webových aplikací stává stále populárnějším, nejen kvůli snadné dostupnosti z jakéhokoli zařízení s internetovým prohlížečem, ale i díky schopnosti fungovat napříč různými platformami. Tyto aplikace nejsou omezeny na jediný operační systém nebo hardware, což z nich činí univerzální řešení. [1]

Příklady webových aplikací zahrnují online formuláře, nákupní košíky, streamování videí, sociální sítě, hry a e-mailové klienty. K jejich vytvoření se používají různé programovací jazyky pro klientskou (např. HTML, CSS, JavaScript) a serverovou část (např. PHP, ASP.NET) [2]

1.1 Výhody a nevýhody webových aplikací

Hlavní přínosy webových aplikací spočívají v jejich schopnosti fungovat na jakémkoliv zařízení s přístupem k internetu a webovým prohlížečem. Tato všestrannost eliminuje potřebu pro více verzí pro různé operační systémy, což vede k snížení nákladů na vývoj a údržbu. Navíc, distribuce těchto aplikací je značně usnadněná, jelikož uživatelé nepotřebují provádět žádné instalace, a mohou tak mít neustálý přístup k nejaktuálnější verzi aplikace.

Další klíčovou výhodou je možnost rychlého a snadného rozšíření aplikace o nové funkce a moduly. To dává podnikům flexibilitu v rychlém reagování na tržní vývoj nebo změny v požadavcích uživatelů. Webové aplikace také usnadňují digitalizaci a digitální transformaci firem, jelikož umožňují efektivnější správu zákaznických vztahů, správu obsahu či interní komunikaci a sdílení dokumentů v rámci organizace.

Z tohoto pohledu jsou webové aplikace klíčovým prvkem pro podniky hledající efektivní, flexibilní a cenově dostupná řešení, která jim umožní zůstat konkurenceschopnými v digitálně propojeném světě.

Nevýhody webových aplikací zahrnují hlavně závislost na připojení k internetu a jeho kvalitě. Další potenciální omezení může být v přístupu k hardwaru zařízení, který může být omezenější ve srovnání s desktopovými aplikacemi. [3][4]

1.2 Architektura webových aplikací

Architektura webové aplikace obvykle sleduje model rozdělený do několika vrstev, které spolu komunikují a spolupracují na zpracování a prezentaci dat uživateli. Nejběžnější architektura je třívrstvá. [5]

1.2.1 Prezentační vrstva (prezentace)

Tato vrstva je odpovědná za zobrazení uživatelského rozhraní a interakci s uživatelem. Nachází se přímo na straně klienta (uživatele) a zpracovává veškeré vstupy uživatele, jakož i prezentaci dat získaných z aplikace. Využívá technologie jako HTML, CSS a JavaScript. [5]

1.2.2 Aplikační vrstva (server)

Aplikační vrstva zpracovává logiku aplikace, provádí operace na základě požadavků z prezentační vrstvy, a může zahrnovat autentizaci uživatelů a interakci s databázemi. Programovací jazyky používané na této vrstvě zahrnují PHP, Ruby, Python a jiné. [5]

1.2.3 Datová vrstva (uložiště)

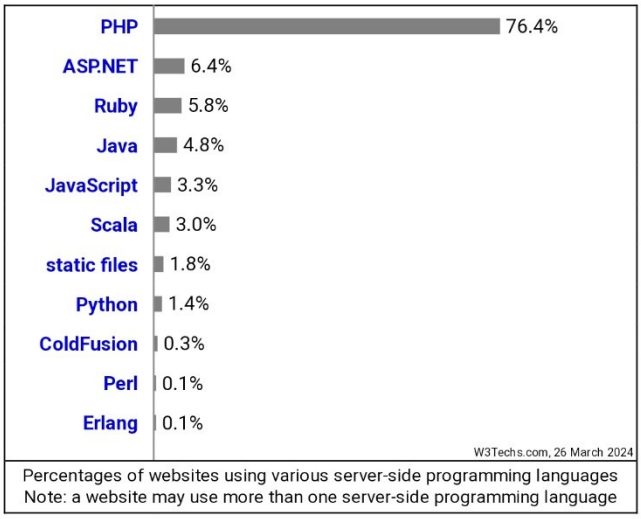
Tato vrstva zahrnuje databáze a další zdroje dat, s nimiž aplikace pracuje. Je zodpovědná za ukládání, načítání a správu dat, jako jsou uživatelské profily, transakční data a další. Používají se zde systémy řízení databází jako MySQL, PostgreSQL, MongoDB a další. [5]

1.3 Framework

Framework představuje sofistikovanou softwarovou architekturu, která usnadňuje a zefektivňuje vývoj aplikací tím, že nabízí předpřipravené moduly a komponenty. Jeho hlavní úlohou je poskytnout programátorům robustní základ, díky němuž mohou přeskočit rutinní úkoly, jako je manipulace s databázemi nebo zpracování uživatelského vstupu, a místo toho se zaměřit na specifické požadavky a inovace svých projektů. Frameworky, vybaveny širokým spektrem nástrojů a standardizovaných praktik, tím pádem výrazně zkracují dobu vývoje a zvyšují kvalitu výsledného softwaru. [6]

2 PHP FRAMEWORKY

PHP, globálně nejrozšířenější serverový programovací jazyk (viz Obrázek 1), je základem pro mnoho frameworků navržených ke zjednodušení a zefektivnění vývoje webových aplikací. Mezi nejpopulárnější frameworky patří Laravel a Symfony. Tyto frameworky poskytují vývojářům sady nástrojů a knihoven k rychlejšímu a pohodlnějšímu vytváření robustních a bezpečných aplikací. [7]



Obrázek 1. Statistiky využití programovacích jazyků na straně serveru pro webové stránky [8]

2.1 Laravel

Laravel se profiluje jako přední PHP framework využívající MVC architekturu pro efektivní tvorbu webových aplikací. Jeho unikátnost tkví v integraci komponent z různých existujících frameworků jako například CodeIgnite a Symfony. Nabízí rozsáhlou sadu funkcí – od autentizace a správy sezení po routování a cachování. Díky adopci MVC architektury Laravel podporuje organizované a logicky strukturované kódování, vedoucí ke kvalitnějším a udržitelnějším aplikacím. [9]

2.1.1 Šablonovací systém Blade

Laravel se pyšní vlastním šablonovacím systémem s názvem Blade, který kombinuje jednoduchost s výkonem. Blade umožňuje vývojářům vkládat PHP kód přímo do šablon, což podporuje rychlý vývoj aplikací. Šablony Blade jsou převedeny na čistý PHP kód a jsou uloženy v cache paměti až do chvíle, kdy dojde k jejich změně, což zvyšuje efektivitu aplikace. Standardně jsou tyto soubory umístěny v adresáři *resources/views* a mají příponu *.blade.php*. [10]

V šablonách se data z proměnných zobrazují pomocí dvojice složených závorek, přičemž Blade automaticky escapuje speciální znaky. Tímto Blade předchází takzvaným XSS útokům, které se zakládají na vložení potencionálně nebezpečného kódu do dynamické webové stránky, která jej následně provede a může tím vytvořit zranitelnost. [10] [11]

Příklad použití proměnné v Blade šabloně může vypadat takto:

```
Hello, {{ $name }}.
```

kde *$name* je proměnná obsahující jméno, které má být zobrazeno uživateli. Tento přístup umožňuje dynamické generování obsahu webových stránek s důrazem na čistotu kódu a bezpečnost. Pokud ale nastane situace, kdy chceme vypsat obsah bez escapování znaku vyměníme jedny složené závorky za dva vykřičníky.

```
Hello, {!! $name !!}.
```

Pro iterování prvků se v Blade šabloně píše *@foreach(...)* a musí končit *@endforeach*, kterým můžeme vypsat například jména všech uživatelů.

```
@foreach ($users as $user)
  {{ $user->name }}
@endforeach
```

Blade tak představuje klíčovou součást ekosystému Laravel, která značně usnadňuje a zefektivňuje vývoj webových aplikací. [10]

2.2 Symfony

Symfony je open-source PHP framework podporovaný společností SensioLabs, navržený pro vývoj komplexních webových aplikací. Je známý svou flexibilitou a schopností zvládat aplikace s velkým počtem připojení, včetně mikrostránek. Jeho výhody zahrnují flexibilitu, silnou ochranu proti webovým útokům a SQL injekcím, snadnou údržbu a znuvupoužitelnost kódu. Na druhou stranu, Symfony vyžaduje větší úsilí k osvojení. [7]

2.2.1 Šablonovací systém Twig

Symfony se spoléhá na Twig jako na svůj šablonovací systém, jehož přístup je podobný jako Blade u Laravelu, avšak s odlišnou syntaxí, která odstraňuje potřebu značky dolaru při referenci na proměnné. [12]

```
Hello, {{ name }}.
```

Pro výpis hodnot bez escapování Twig využívá filtr *raw*, který se píše za "|".

```
Hello, {{ name | raw }}.
```

Twig upřednostňuje tečku pro přístup k atributům objektů. Pro iterace využívá *for* s ukončovací značkou *endfor*, což představuje odchylku od tradičního PHP syntaxe. [12]

```
{% for user in users %}
  {{ user.username }}
{% endfor %}
```

Tyto změny v syntaxi Twig mohou představovat výzvy pro vývojáře zvyklé na standardní PHP.

3 FRAMEWORK NETTE

Nette je vysoce hodnocený český PHP framework s důrazem na produktivitu, osvědčené postupy a bezpečnost. Jeho filozofie klade velký důraz na bezpečnost, eliminuje běžné bezpečnostní hrozby a je považován za jeden z nejbezpečnějších PHP frameworků. Vyznačuje se čistým, objektově orientovaným designem a rychlostí. Nette se skládá z mnoha samostatných komponent a je oblíbený pro svou modularitu a rozšiřitelnost. [13]

3.1 Historie Nette

Historie frameworku Nette začala v roce 2004, kdy David Grudl hledal vhodný nástroj pro vývoj aplikací, protože se cítil omezen pouhým PHP. Protože žádný z existujících frameworků mu nevyhovoval, rozhodl se vytvořit vlastní, který později pojmenoval Nette. Tento krok předcházel vzniku dnes populárních frameworků jako Symfony nebo Laravel. Grudl se inspiroval znovupoužitelnými UI komponentami, které znal z Delphi, a snažil se vytvořit systém, který by byl kompatibilní s bezstavovým protokolem HTTP a přívětivý k uživatelům i vývojářům. Po tisících hodinách vývoje v garáži za Brnem se zrodila první verze Nette.

Framework zaujal architekturou založenou na MVC vzoru a brzy si získal pozornost komunity. Jeho představení v roce 2007 na konferenci v Praze předznamenalo významný krok v jeho rozvoji, a vedlo k vytvoření silné komunity okolo Nette. Důležitým milníkem bylo také začlenění Dependency Injection Container (DIC), což zásadně změnilo přístup k návrhu aplikací.

Moderní éra Nette začala vydáním verze 2.0 v roce 2012, která přinesla mnoho nových funkcí, včetně Nette Database a Exploreru, nástroje pro práci s databázemi. Vývoj frameworku pokračoval dalšími vydáními, které rozdělily jeho části do samostatných balíčků, čímž usnadnily jeho použití a integraci s nástrojem Composer. [14]

3.2 Tým Nette

Za vývojem frameworku Nette stojí čtveřice hlavních programátorů doplněná o širokou komunitu, která do projektu přináší své vlastní vylepšení. David Grudl, zakladatel Nette, v poslední době rozšiřuje své aktivity do oblasti umělé inteligence a vede na toto téma semináře. Od roku 2022 také pravidelně přispívá do videopořadu TECH GUYS, kde spolu s kolegy diskutuje o novinkách ve světě technologií. [15][16]

Jan Černý, známý jako Honza Chemix, se věnuje moderování a organizaci setkání pro programátory známých jako Poslední soboty a organizuje také Nette Campy.

Miloslav Hůla, přezdívaný Milo, se soustředí na vývoj a zdokonalování Nette Testeru a zajišťuje, že webhosting pro nette.org je na vysoké úrovni.

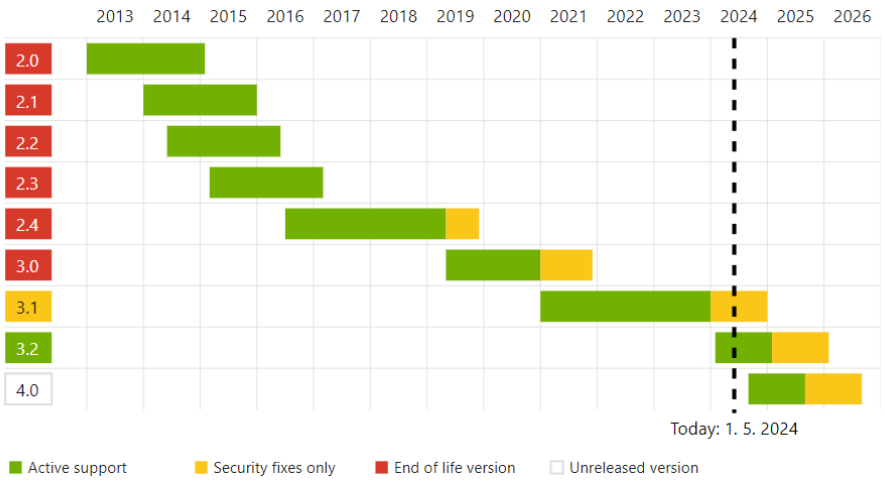
Milan Felix Šulc stojí za vytvořením a rozvojem Componette.org, což je platforma shromažďující doplňky pro Nette, a také spravuje knihovnu doplňků Contributte.

Dále je třeba zmínit Filipa Procházku, který společně s Davidem Grudlem a Tomášem Votrubou přispívá k tvorbě dokumentace frameworku Nette. Tento tým, obohacený o mnoho dalších přispěvatelů, se stará o to, aby dokumentace udržovala vysokou kvalitu a poskytovala vývojářům ucelený zdroj informací. [15]

3.3 Údržba Nette

Nette Framework se vyznačuje výjimečně dlouhodobou podporou pro každou z jeho verzí. Všechny hlavní vydání jsou kategorizovány jako LTS (Long-Term Support Release), což znamená, že každá větev je podporována alespoň dva roky.

Aktivní údržba každé verze trvá minimálně jeden rok od momentu, kdy je verze stabilně vydána. Opravy kritických chyb a záplaty pro zajištění bezpečnosti jsou zaručeny po dobu dvou let. Dále, jak je možné vyčíst z obrázku 2, existuje předpoklad, že Nette verze 4 by mohla být vydána ke konci roku 2024, což naznačuje plánovanou kontinuitu a rozvoj frameworku i do budoucna. [17]



Obrázek 2. Kalendář vydávání Nette [17]

3.4 Adresářová struktura Nette

Adresářová struktura frameworku Nette je předem definovaná, avšak nabízí vysokou míru flexibility. Umožňuje uživatelům strukturu přizpůsobit podle vlastních potřeb, což zahrnuje přejmenování nebo přesun složek. Aby tyto změny byly aplikovány, je nutné aktualizovat cesty k logům a dočasným souborům v souboru *Bootstrap.php* a také cestu k tomuto souboru v *composer.json* v části *autoload*. Tento postup je jednoduchý a nevyžaduje žádnou složitou rekonfiguraci ani úpravy konstant, díky chytrému mechanismu autodetekce, který Nette využívá.

```
web-project/
├── app/                    ← adresář s aplikací
│   ├── Presenters/         ← presentery a šablony
│   │   ├── HomePresenter.php ← třída presenteru Home
│   │   └── templates/      ← adresář se šablonami
│   │       ├── @layout.latte ← šablona layoutu
│   │       └── Home/       ← šablony presenteru Home
│   │           └── default.latte  ← šablona akce 'default'
│   ├── Router/             ← konfigurace URL adres
│   └── Bootstrap.php       ← zaváděcí třída Bootstrap
├── bin/                    ← skripty spouštěné z příkazové řádky
├── config/                 ← konfigurační soubory
│   ├── common.neon
│   └── local.neon
├── log/                    ← logované chyby
├── temp/                   ← dočasné soubory, cache, …
├── vendor/                 ← knihovny instalované Composerem
│   ├── ...
│   └── autoload.php        ← autoloading všech nainstalovaných balíčků
├── www/                    ← veřejný adresář neboli document-root projektu
│   ├── .htaccess           ← pravidla mod_rewrite
│   └── index.php           ← prvotní soubor, kterým se aplikace spouští
└── .htaccess               ← zakazuje přístup do všech adresářů krom www
```

Obrázek 3. Adresářová struktura Nette [18]

Pro rozsáhlejší aplikace je možné dále strukturu rozčlenit pomocí modulů, což umožňuje efektivní organizaci kódu do logických celků, jako jsou například různé uživatelské role nebo části aplikace určené pro administraci a běžné uživatele. Tyto moduly mohou být organizovány do podadresářů a třídy mohou být zařazeny do jmenných prostorů.

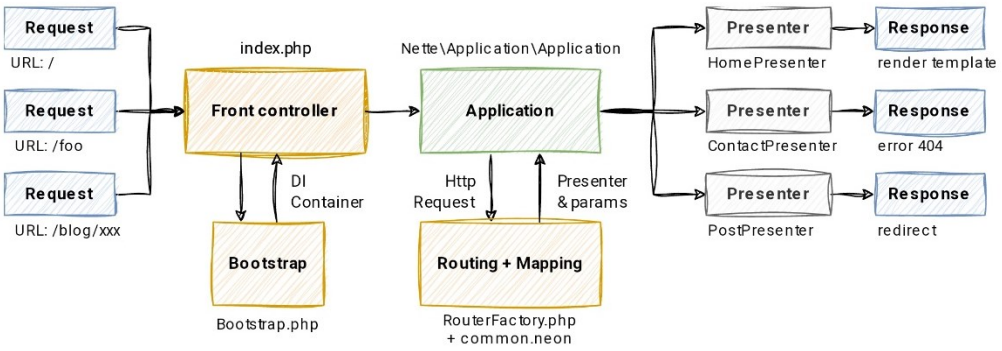
Veřejný adresář, označovaný jako *www/*, slouží jako kořenový dokument (*document-root*) projektu a lze jej přejmenovat bez nutnosti dalšího nastavování aplikace. Pro správné fungování na serveru je pak potřeba zajistit, aby hosting správně odkazoval *document-root* do tohoto adresáře. [18]

3.5 HTTP požadavek

Proces zpracování webového požadavku v rámci Nette frameworku začíná v okamžiku, kdy uživatel ve svém webovém prohlížeči navštíví určitou stránku. Tento krok aktivuje HTTP požadavek, který je odeslán na server. Klíčovým bodem pro zpracování tohoto požadavku je PHP soubor umístěný ve veřejném adresáři (*www/*), specificky soubor *index.php*. Například, pokud uživatel požaduje stránku s URL adresou *https://example.com/product/123*, správná konfigurace serveru zajistí, že tento požadavek bude převeden na spuštěném souboru *index.php*. [18]

Hlavním úkolem je:

1. inicializovat prostředí
2. získat továrnu
3. spustit Nette aplikaci, která vyřídí požadavek



Obrázek 4. Tok HTTP požadavku

3.6 Nette Application

Nette Application, tedy aplikační vrstva frameworku Nette, má za primární cíl zpracovat a odpovědět na HTTP požadavky. V Nette se upřednostňuje pojem presenter místo tradičního controlleru z důvodu, aby se vyhnulo záměně s ostatními termíny začínajícími na con. Aplikace v Nette jsou tedy strukturovány do mnoha presenterů, kde každý reprezentuje specifickou stránku nebo funkcionalitu webové aplikace, jako je domovská stránka, produktová stránka e-shopu, přihlašovací/registrační formulář či sitemap.

Práce s HTTP požadavky začíná u tzv. routeru, který rozhoduje, kterému presenteru bude požadavek přiřazen. Rozhodování závisí na URL adrese požadavku, přičemž router identifikuje odpovídající presenter a akci, kterou má vykonat. Toto přiřazení se obvykle

zaznamenává ve formátu *Presenter:akce + parametry*, například *Product:show + id 123* pro zobrazení produktu s identifikátorem 123.

Jakmile je určen správný presenter, Application se obrátí na DI kontejner, aby vytvořil instanci třídy presenteru, v tomto případě *ProductPresenter*. Presenter poté převezme kontrolu a jeho úkolem je provést požadovanou akci, v tomto příkladě metodou *renderShow($id),* kde *$id* obsahuje hodnotu 123.

Odpověď presenteru může být různorodá – od HTML stránky, přes obrázek, XML dokument, JSON, odesílání souborů až po přesměrování na jinou stránku. Většina akcí však končí ve vykreslení HTML šablony, což je výchozí chování, protože to odpovídá běžné potřebě vykreslovat webové stránky. Nette aplikace automaticky vyhledá odpovídající šablonu podle konvence pojmenování souborů, například ve formátech *templates/Product/show.latte* nebo *templates/Product.show.latte*, a pokud je dostupný, použije i společný layout definovaný v souboru *@layout.latte*.

Tímto způsobem Nette Application dokončí svůj úkol – efektivně, flexibilně a s výchozím nastavením, které usnadňuje práci vývojářům. V případě, že by odpovídající šablona nebyla nalezena, je uživateli zobrazena chyba 404. [18]

3.7 DI kontejner

DI kontejner neboli továrna na objekty, představuje jádro každé aplikace postavené na Nette frameworku. Jedná se o PHP třídu, kterou Nette dynamicky generuje a ukládá do cache adresáře. Tato třída obsahuje množství metod, jako například *createServiceTranslation()*, kde každá metoda je schopna vytvořit a vrátit specifický objekt. Mezi tyto metody patří i *createServiceApplication()*, která je zásadní pro vytvoření instance *Nette\Application\Application* potřebné pro spuštění celé aplikace v souboru *index.php*. Kontejner také obsahuje metody pro vytváření presenterů, které jsou klíčové pro správnou funkci webové aplikace.

Objekty vytvořené pomocí DI kontejneru jsou označovány jako služby, což reflektuje jejich význam pro celou aplikaci. Unikátnost této třídy spočívá v tom, že není tvořena programátorem, ale je generována samotným frameworkem. Programátor tedy nepíše přímo PHP kód třídy, ale poskytuje frameworku instrukce skrze konfigurační soubory ve formátu NEON, které určují, jaké objekty má kontejner vytvořit a jakým způsobem.

Tyto konfigurační soubory slouží k definování chování DI kontejneru. Například, pokud v konfiguraci u sekce *session* nastavím *expiration: 14 days*, DI kontejner při generování objektu *Nette\Http\Session* automaticky zavolá jeho metodu *setExpiration('14 days')*, čímž se tato konfigurační volba promítne do reálného chování session. Tímto způsobem framework Nette umožňuje vývojářům flexibilně a efektivně spravovat a konfigurovat služby své aplikace. [18]

3.8 Bootstrap

Bootstrap v kontextu Nette frameworku označuje klíčový zaváděcí kód, který zajišťuje správné nastartování celé aplikace. Tento proces zahrnuje inicializaci prostředí, tvorbu DI kontejneru, který spravuje závislosti mezi objekty aplikace, a nakonec samotné spuštění aplikace.

V minulosti se pro inicializaci často používaly soubory jako *include.inc.php*, které byly inkludovány na začátku každého skriptu. Dnes tuto úlohu přebírá v Nette aplikacích třída Bootstrap. Její zdrojový kód lze obvykle najít v souboru *app/Bootstrap.php* a slouží jako vstupní bod pro všechny typy aplikací – ať už se jedná o webové aplikace nebo skripty spouštěné z příkazové řádky.

Struktura třídy Bootstrap je navržena tak, aby poskytovala vývojářům čisté a organizované rozhraní pro inicializaci aplikace. Umožňuje snadno konfigurovat a rozšiřovat aplikaci pomocí konfiguračních souborů a zároveň zachovává přehlednost a udržitelnost kódu. [19]

```php
use Nette\Bootstrap\Configurator;

class Bootstrap
{
    public static function boot(): Configurator
    {
        $appDir = dirname(__DIR__);
        $configurator = new Configurator;
        //$configurator->setDebugMode('secret@23.75.345.200');
        $configurator->enableTracy($appDir . '/log');
        $configurator->setTempDirectory($appDir . '/temp');
        $configurator->createRobotLoader()
            ->addDirectory(__DIR__)
            ->register();
        $configurator->addConfig($appDir . '/config/common.neon');
        return $configurator;
    }
}
```

Obrázek 5. Příklad třídy Bootstrap [19]

Nette framework nabízí dva primární provozní režimy pro zpracování požadavků: vývojářský (development) a produkční (production) režim. Tyto režimy jsou uzpůsobeny pro různé potřeby a fáze vývoje aplikace.

Přepínání mezi těmito režimy je řízeno autodetekcí, díky čemuž ve většině případů není nutné, jakkoliv ručně konfigurovat, který režim má být použit. Tato inteligentní funkcionalita umožňuje snadný přechod od vývoje k ostrému nasazení bez nutnosti zásadních změn v nastavení aplikace. [19]

3.8.1 Vývojářský režim

Vývojářský režim je designován tak, aby co nejvíce vyhovoval potřebám programátora během vývoje aplikace. Nabízí různé nástroje a funkce usnadňující ladění a testování, jako je například zobrazování chybových hlášení prostřednictvím Tracy, automatická aktualizace cache při změnách v šablonách nebo konfiguraci DI kontejneru, a další vývojářské pohodlí. Tento režim je aktivován, pokud je aplikace spuštěna na adrese localhostu, což je indikováno IP adresou 127.0.0.1 nebo ::1, a zároveň není detekována žádná proxy na základě HTTP hlavičky. [19]

3.8.2 Produkční režim

Produkční režim je naopak optimalizován pro maximální výkon a je určen pro ostré nasazení aplikace. V tomto režimu Tracy pouze loguje výskyt chybových stavů a systém neprovádí automatické testování změn v šablonách nebo konfiguračních souborech, aby se zvýšila celková efektivita a rychlost aplikace. [19]

3.8.3 Debugovací nástroj Tracy

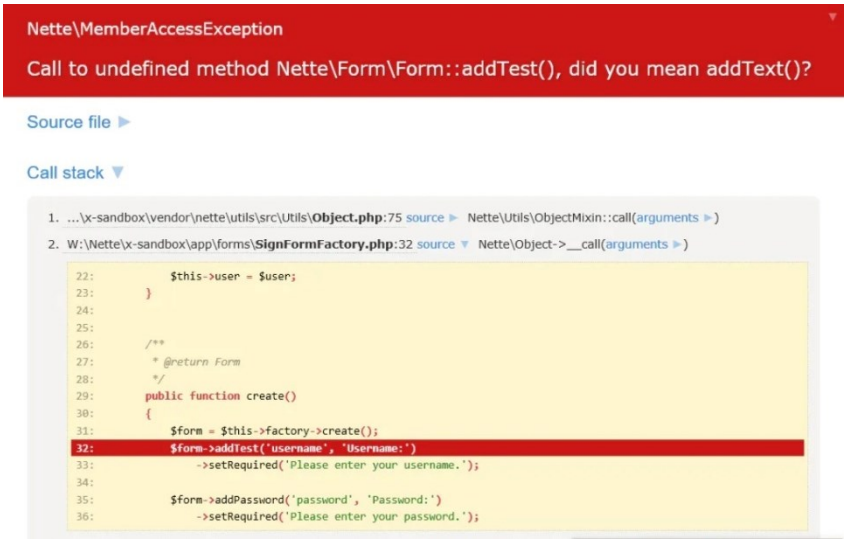
Knihovna Tracy, známá také jako Laděnka, se stala neocenitelným nástrojem pro každodenní práci PHP programátorů. Její hlavní přednosti zahrnují:

- Rychlou identifikaci a opravu chyb: Tracy umožňuje efektivně odhalovat chyby v kódu a přispívá k jejich rychlému řešení.
- Logování chyb: Vedle odhalování chyb knihovna poskytuje možnosti jejich zaznamenávání pro další analýzu.
- Výpis proměnných: Tracy usnadňuje ladění tím, že zobrazuje hodnoty proměnných bez nutnosti psaní dodatečného kódu.

- Měření času skriptů a databázových dotazů: Díky tomu můžete optimalizovat výkon aplikace.
- Sledování paměťových nároků: Tracy pomůže identifikovat, které části aplikace vyžadují nejvíce systémových zdrojů.

Tracy nabízí Tracy Bar, plovoucí panel, který v reálném čase poskytuje základní informace relevantní pro vývoj a ladění aplikace. Panel zahrnuje detaily o použitém presenteru a akci, přítomné parametry akce jako identifikátor (id), informace o době zpracování požadavku, podrobnosti o databázových dotazech včetně jejich délky a typů, údaje o přihlášeném uživateli, a mnoho dalších důležitých metrik. [20]

Když je Tracy aktivována, chybové hlášení se nezobrazí jako obyčejný text vložený přímo do zdrojového kódu stránky, ale jako detailně zpracovaný výpis s možnými návrhy na opravu, včetně upozornění na překlepy ve jménech funkcí, které se snažíme volat. [20]



Obrázek 6. Ukázka výpisu chyby pomocí Tracy [20]

3.9 Presentery

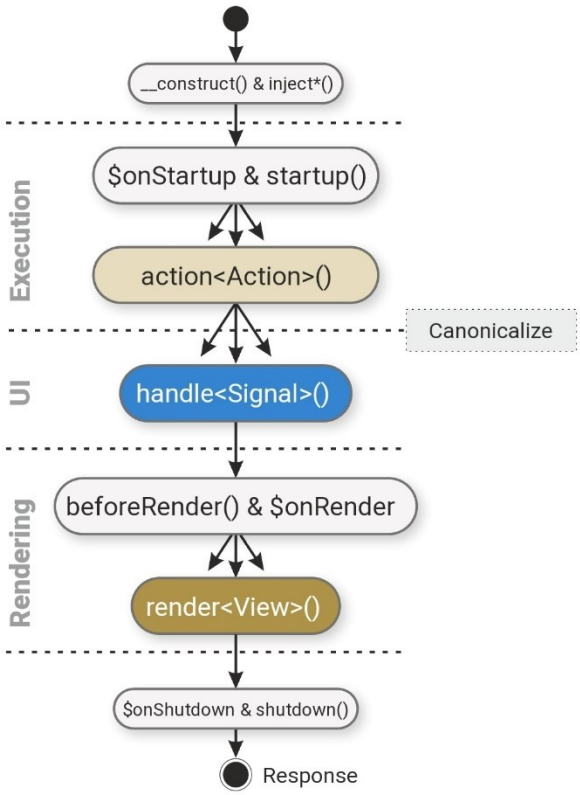
V kontextu Nette frameworku je presenter pojmem pro třídu, která reprezentuje specifickou stránku nebo funkční část webové aplikace, jako je například úvodní stránka (homepage), produktová stránka v e-shopu a další. Aplikace může obsahovat široký rozsah presenterů, od jednoho až po tisíce, v závislosti na komplexitě a struktuře dané aplikace. V jiných frameworcích se pro podobné komponenty používá termín controllery.

Typicky je presenter v Nette odvozen od třídy *Nette\Application\UI\Presenter*, která je specificky navržena pro tvorbu webových rozhraní. V širším smyslu může být za presenter považován jakýkoliv objekt, který implementuje rozhraní *Nette\Application\IPresenter*.

3.10 Životní cyklus presenteru

Hlavní úlohou presenteru je zpracovat přijatý požadavek a vygenerovat odpovídající odpověď, která může mít podobu HTML stránky, obrázku, přesměrování, a podobně. Základem je objekt *Nette\Application\Request*, do kterého byl původní HTTP požadavek přetransformován pomocí routeru. Ačkoliv se s tímto objektem přímo často nesetkáme, presenter umožňuje jeho zpracování přesunout do různých metod, které se volají v určitém pořadí a zajišťují vyřízení požadavku.

Obrázek 7 ukazuje metody, které se volají postupně od shora dolů. Tyto metody ale existovat nemusí, pokud se například jedná o jednoduchý statický web. [21]



Obrázek 7. Životní cyklus presenteru [22]

3.10.1 __construct()

Konstruktor se nezapočítává přímo do životního cyklu presenteru, protože je aktivován v momentě, kdy dochází k tvorbě instance objektu. Účelem konstruktoru, spolu s metodou inject, je umožnění předání závislostí do presenteru.

Presenter by se neměl podílet na byznys logice aplikace, čtení či zápisu do databáze, nebo vykonávání výpočtů – toto je úloha pro třídy modelové vrstvy. Jako příklad, třída jako *ArticleRepository* by se mohla starat o načítání a ukládání článků. Presenter s takovouto třídou komunikuje skrze mechanismus dependency injection, který umožňuje efektivní a přehlednou správu závislostí v aplikaci. [21]

3.10.2 startup()

Hned po přijetí požadavku je vyvolána metoda *startup()*. Tato metoda poskytuje ideální místo pro různé inicializační úkony, jako je nastavení proměnných, ověřování uživatelských práv a podobně. [21]

3.10.3 action<Action>(args...)

Metoda *action<Action>()* je určena k zpracování požadavků nezávisle na vykreslování šablon. Na rozdíl od metod *render<View>()*, které se zaměřují na přípravu dat pro vykreslení konkrétní šablony, metody akcí se věnují logice aplikace, jako je zpracování dat, přihlašování nebo odhlášení uživatele, následované případným přesměrováním na jinou stránku či akci.

Akce je volána před renderem, což dává možnost ovlivnit, která šablona nebo render metoda bude následně použita. To lze provést pomocí volání *setView('jineView')*, čímž lze dynamicky měnit, která šablona se má zobrazit.

Metodám akcí mohou být předány parametry z HTTP požadavku. Je vhodné a doporučené specifikovat typy těchto parametrů, například:

actionShow(int $id, string $slug = null)

V případě, že potřebný parametr chybí nebo neodpovídá deklarovanému typu (např. pokud id není celé číslo), presenter automaticky vygeneruje chybu 404 a ukončí zpracování požadavku. [21]

3.10.4 handle<Signal>(args...)

Metoda *handle<Signal>(args...)* je v Nette presenteru zodpovědná za zpracování signálů, což jsou speciální typy požadavků, nejčastěji používané pro zpracování AJAXových požadavků. Signály umožňují efektivní komunikaci mezi klientem a serverem bez potřeby obnovování celé stránky, což vede k dynamickým a interaktivním webovým aplikacím.

Podobně jako u metod akcí i u handle mohou být metodě předány parametry z HTTP požadavku. Tyto parametry mohou a měly by být typově specifikované, aby se zajistila správná funkčnost a bezpečnost zpracování požadavku. [21]

3.10.5 beforeRender()

Metoda *beforeRender()* je, jak název napovídá, volána před vyvoláním kterékoliv render metody. Jejím hlavním účelem je umožnit společnou konfiguraci a přípravu šablony, ještě předtím, než dojde k samotnému vykreslení. To zahrnuje například předání proměnných, které budou využity v layoutu, nebo provedení jiných nastavení, která mají být společná pro všechny šablony daného presenteru. [21]

3.10.6 render<View>(args...)

Metoda *render<View>(args...)* v Nette presenteru je klíčovým bodem, kde se připravuje šablona pro vykreslení, a to tím, že jí jsou předávána relevantní data a další parametry potřebné pro zobrazení. Tato metoda umožňuje flexibilní manipulaci s obsahem, který má být uživatelům prezentován, a zajišťuje, že šablona bude mít přístup ke všem informacím, které potřebuje. Stejně jako metody akcí a renderu může přijímat parametry předané z HTTP požadavku. [21]

3.10.7 afterRender()

Metoda *afterRender()* je v Nette presenteru volána po každé render metodě. Tato metoda se obvykle využívá méně často a slouží pro provedení akcí, které mají nastat až po vykreslení šablony, jako je například logování nebo jiné post-renderingové operace. [21]

3.10.8 shutdown()

Metoda *shutdown()* je v Nette presenteru vyvolána na samém konci životního cyklu presenteru. Její hlavní úlohou je poskytnout místo pro vykonání závěrečných akcí předtím, než je životní cyklus presenteru kompletně ukončen. To může zahrnovat uvolnění zdrojů, zápis logů, dokončení transakcí nebo jakékoli jiné čistící operace. [21]

3.11 Odeslání odpovědi

Odpovědí presenteru může být obvykle vykreslení šablony s HTML stránkou, ale existují i další možnosti, jak reagovat na požadavek. Patří sem například odeslání souboru, výstup ve formátu JSON, nebo přesměrování na jinou stránku.

Během celého životního cyklu presenteru lze použít některou z následujících metod, které nejenže odešlou specifickou odpověď, ale také ukončí běh presenteru:

- *redirect()*, *redirectPermanent()*, *redirectUrl()*, *forward()* - tyto metody zajistí přesměrování na jinou adresu nebo akci. Rozdíl mezi nimi spočívá v typu přesměrování (dočasné, trvalé) nebo v tom, že *forward* předá zpracování jiné akci/presenteru bez změny URL v prohlížeči.
- *error()* - tato metoda ukončí běh presenteru s chybou, což může být užitečné například při pokusu zobrazení neexistující položky v databázi.
- *sendJson($data)* - odešle data ve formátu JSON a ukončí presenter. Ideální pro AJAXové požadavky nebo REST API.
- *sendTemplate()* - vykreslí šablonu a ukončí presenter.
- *sendResponse($response)* - umožňuje odeslat vlastní odpověď, která byla připravena mimo standardní vykreslovací proces Nette.
- *terminate()* - ukončí presenter bez vyslání jakékoliv odpovědi.

Pokud během životního cyklu presenteru není explicitně volána žádná z metod pro odeslání odpovědi, presenter přejde k automatickému vykreslení šablony. [21]

3.12 Šablonovací systém Latte

Nette využívá pro práci se šablonami systém Latte, který je oceňován především pro svou vysokou úroveň zabezpečení a intuitivnost. Latte používá pro šablonování značení velmi podobné PHP, což usnadňuje jeho ovládání. Narozdíl od systémů jako Twig, kde je potřeba se naučit úplně novou syntaxi, umožňuje Latte vývojářům snadno přejít k práci se šablonami bez nutnosti osvojovat si nový způsob zápisu.

V Latte je běžnou praxí, že stránka je kombinací layoutové šablony a šablony specifické pro danou akci. Tady je příklad, jak by mohla vypadat layoutová šablona:

```
<!DOCTYPE html>
<html>
<head>
	<title>{block title}My App{/block}</title>
</head>
<body>
	{include content}
</body>
</html>
```

A zde je příklad, jak by mohla vypadat šablona akce:

```
{block title}Homepage{/block}
{block content}
<h1>Homepage</h1>
...
{/block}
```

Použití značky *{include}* signalizuje, že šablona specifická pro danou akci by měla obsahovat blok se shodným názvem jako je ten, který je uveden v *{include}* v layoutové šabloně. Pokud takový blok není v akční šabloně nalezen, dojde k vyhození výjimky a zobrazení chybového hlášení.

Blok *{block}* je definován jako párový, což znamená, že má otevírací a zavírací tag. Obsah umístěný mezi těmito tagy v layoutové šabloně slouží jako výchozí hodnota. V našem případě, pokud by šablona akce neobsahovala blok s názvem title, automaticky by se použil obsah "My App" definovaný v layoutové šabloně. [23]

3.12.1 Předávání proměnných do šablony

Proměnné jsou do šablony v Nette předávány pomocí přiřazení hodnot k vlastnosti *template* objektu presenteru. Tento postup umožňuje, aby byly proměnné v šabloně dostupné jako lokální proměnné, což zjednodušuje jejich použití. Například:

```
$this->template->article = $this->articles->getById($id);
```

Tento kód předá šabloně článek, který byl získán pomocí metody *getById($id)* z repozitáře *articles*. V šabloně je poté možné k této proměnné přistupovat přímo pomocí názvu proměnné, v tomto případě *article*, a manipulovat s ní nebo zobrazovat její obsah. [23]

3.12.2 Výchozí proměnné v šabloně

V Nette frameworku jsou do šablon automaticky předávány některé užitečné proměnné, které usnadňují běžné úkoly a poskytují snadný přístup k důležitým informacím. Mezi tyto výchozí proměnné spadají:

- *$basePath*: Obsahuje absolutní URL cestu ke kořenovému adresáři aplikace, což umožňuje snadné odkazování na statické soubory, jako jsou obrázky, CSS a JavaScript.
- *$baseUrl*: Poskytuje absolutní URL ke kořenovému adresáři aplikace včetně schématu a domény, například *http://localhost/eshop*.
- *$user*: Je objekt reprezentující aktuálního uživatele a poskytuje metody pro zjištění, zda je uživatel přihlášen, jeho role a další.
- *$presenter*: Umožňuje přístup k aktuálnímu presenteru, což je užitečné například pro získání názvu aktuální akce.
- *$control*: Odkazuje na aktuální komponentu nebo presenter.
- *$flashes*: Obsahuje pole zpráv, které byly zaslány funkcí *flashMessage()*. Tyto zprávy jsou typicky používány pro zobrazení informačních, varovných či chybových hlášení uživateli. [23]

3.12.3 Bezpečnost Latte

Latte se odlišuje od ostatních šablonovacích systémů pro PHP především díky své efektivní ochraně proti kritické zranitelnosti Cross-site Scripting (XSS). Tuto ochranu zajišťuje skrze kontextově citlivé escapování, což znamená, že Latte rozumí kontextu, v němž jsou data vypisována, a upravuje své escapovací chování podle toho.

Kontext v tomto případě odkazuje na místo v dokumentu, které má specifická pravidla pro zpracování a vypisování dat. Tato pravidla se liší v závislosti na typu dokumentu (např. HTML, XML, CSS, JavaScript, obyčejný text atd.) a mohou se měnit i v rámci různých částí jednoho dokumentu. V HTML dokumentu existuje mnoho takových kontextů, kde každý vyžaduje odlišný přístup k ošetření vypisovaných dat – ať už se jedná o vkládání textu do těla dokumentu, do atributů elementů, jako jsou řetězce ve *value* atributu input elementu, do komentářů, do CSS nebo do JavaScriptu, jako je například obsah *onClick* atributu.

Díky schopnosti rozpoznat a správně zpracovat tyto různé kontexty, Latte nabízí výjimečně silnou ochranu proti XSS útokům. Na rozdíl od jiných šablonovacích systémů, jako jsou

Twig nebo Blade, které používají jednodušší escapovací mechanismus s nízkým nebo žádným ohledem na kontext, Latte poskytuje sofistikovanější a bezpečnější řešení. Toto činí Latte unikátním nástrojem zvláště v případě, že je prioritou zabezpečení aplikace. [24]

3.12.4 Výpis v Latte

V šablonách Latte se používají tagy uzavřené ve složených závorkách, jak již bylo zmíněno v kapitole o odstavci 3.12 „Šablonovací systém Latte“. Tyto tagy mohou být buď párové, nebo nepárové.

Pro výpis automaticky escapované proměnné se používá syntaxe:

```
Hello, {$name}
```

Pokud je však potřeba vypsat text bez escapování, použijeme filtr noescape.

```
Hello, {$name|noescape}
```

Na rozdíl od jiných šablonovacích systémů, kde iterace vyžadují specifické značky pro ukončení cyklu (například přípony typu "end"), v Latte takové speciální ukončovací značky pro iterace nejsou potřeba. Latte navíc obsahuje v iterátoru zabudovanou proměnnou *$iterator*, která poskytuje užitečné funkce pro zjištění, zda se jedná o první či poslední iteraci, aktuální pořadí iterace, nebo zda je iterace lichá či sudá.

Příklad iterace, kde se vypisují všichni uživatelé s pořadovým číslem začínajícím od 1:

```
{foreach $users as $user}
  {$iterator->counter}. {$user->name}
{/foreach}
```

Druhou možností iterace v Latte je použití atributu *n:foreach* přímo v HTML tagu, což zjednodušuje čitelnost kódu:

```
<ul>
  <li n:foreach="$users as $user"> {$iterator->counter}. {$user->name}</li>
</ul>
```

Latte dále nabízí řadu dalších užitečných *n:* atributů, jako je *n:if* pro zobrazení obsahu na základě splnění podmínky, a od verze 3.0.11 nově i *n:else*, který umožňuje definovat obsah pro případ, kdy podmínka není splněna. Tento atribut lze využít i v kontextu iterace, kde pokud není co iterovat, zobrazí se obsah určený *n:else* tagem.

Atribut *n:class* umožňuje dynamické přiřazování CSS tříd na základě podmínek, což je užitečné například pro zvýraznění aktuálně zobrazené stránky v navigaci. [25]

3.12.5 Překládání textů

Pro účely překladů využívá Nette framework rozhraní *Nette\Localization\Translator*, které definuje metodu *translate()*. Tato metoda je určena k překladu zpráv a umožňuje vkládání proměnných do překládaného textu. Příkladem použití je pozdrav typu "Ahoj {jmeno}", kde *{jmeno}* je proměnná, která bude v kontextu překladu nahrazena skutečným jménem z dodaného pole proměnných.

Nette sám o sobě neposkytuje žádnou předdefinovanou implementaci pro překladový systém. Proto je potřeba pro překlady v aplikaci využít externí řešení. V kontextu bakalářské práce bylo vybráno řešení *contributte / translation*, které představuje stále aktivně vyvíjenou knihovnu, nabízející aktuální a efektivní nástroje pro práci s překlady.

Při použití překladů v šablonách Latte existuje několik způsobů, jak tyto překlady realizovat:

- Překlad pomocí prefixu spodníku: Zápis *{_nav.settings}* používá spodník jako prefix, což signalizuje volání překladové funkce pro řetězec *nav.settings*. Tento způsob je rychlý a přehledný pro jednoduché překlady.
- Překlad pomocí filtru: Syntaxe *{nav.setting|translate}* využívá filtr *translate*, který aplikuje překlad na zadaný řetězec *nav.setting*. Tento způsob je flexibilní a může být snadno kombinován s dalšími filtry.
- Překlad v bloku: Zápis *{translate}nav.setting{/translate}* umožňuje definovat blok textu, který má být přeložen.

Každý z těchto způsobů má své využití v závislosti na konkrétní situaci a potřebách vývojáře. Díky této flexibilitě mohou být překlady šablonách snadno a efektivně implementovány tak, aby vyhovovaly požadavkům vícejazyčné aplikace. [23]

3.13 Routování

Routování v Nette umožňuje tvorbu tzv. lidštějších, nebo také "cool" či "pretty" URL, které jsou nejen lehce zapamatovatelné a použitelné, ale také přispívají k lepšímu SEO. Tyto URL umožňují navrhovat adresy stránek přesně podle potřeb projektu, a to nezávisle na tom, kdy je struktura URL navržena – zda na začátku vytváření projektu, během přidávání nových presenterů či Latte šablon, nebo dokonce až po dokončení aplikace. Všechna pravidla pro routování se definují na jednom místě, a to v routeru.

Unikátnost routeru v Nette spočívá v jeho obousměrné funkčnosti. Nejenže dokáže dekódovat příchozí URL do konkrétních požadavků na presentery a akce, ale zároveň umožňuje

generovat URL zpětně z těchto požadavků. To z routeru činí klíčový prvek celého *Nette Application*, protože nejen rozhoduje o tom, jaký presenter a jaká akce bude reagovat na daný HTTP požadavek, ale také poskytuje nástroje pro generování URL adres v šablonách a dalších částech aplikace. Díky tomu může být navigace a struktura webu konsistentní a přátelská jak pro uživatele, tak pro vyhledávače. [26]

3.14 Interaktivní komponenty

Komponenty v Nette frameworku představují samostatné, znovupoužitelné objekty, které lze vkládat do webových stránek. Tyto komponenty mohou zahrnovat širokou škálu prvků, jako jsou formuláře, datagridy, ankety, menu, grafy a mnoho dalšího, což z nich dělá flexibilní nástroje pro opakované použití v různých částech aplikace. Nette disponuje vestavěným komponentovým systémem, který je v PHP světě poměrně unikátní, ačkoliv podobné koncepty lze nalézt v technologiích jako Delphi, ASP.NET Web Forms, nebo ve front-endových frameworkách jako React či Vue.js.

Do jednoho presenteru je možné začlenit libovolný počet komponent, a tyto komponenty mohou obsahovat další komponenty, čímž vzniká hierarchická struktura zvaná komponentový strom, v jehož čele stojí presenter.

Komponenty jsou typicky do presenteru vkládány pomocí továrních metod, což je elegantní způsob, jak je vytvářet "na požádání" (lazy loading). Tovární metoda, nazvaná *createComponent<Name>()*, kde *<Name>* odpovídá názvu komponenty, je odpovědná za její vytvoření a inicializaci.

V kontextu Nette jsou komponenty obvykle odvozeny od třídy *Nette\Application\UI\Control*. Zatímco termín „controls“ by byl možná přesnější, v češtině se více ujaly „komponenty“.

Pro vykreslení komponenty se v šabloně používá značka *{control componentName}*, která volá metodu *render()* dané komponenty. V této metodě se zajišťuje vykreslení komponenty podobně jako v presenteru, s tím rozdílem, že je třeba explicitně specifikovat šablonu a vykreslit ji. Značka *{control}* navíc umožňuje předávat parametry do metody render komponenty, což rozšiřuje možnosti jejího využití a konfigurace. [27]

3.15 Signály

Signály v Nette frameworku fungují podobně jako akce, ale na rozdíl od nich jsou určeny pro práci s komponentami a volají metody *handle<Signal>()*. Zatímco pojem akce (nebo zobrazení/view) je přidružen výhradně k presenterům, signály mohou být použity u jakékoli komponenty v rámci aplikace.

Při tvorbě odkazu na metodu pro zpracování signálu (handle) v Latte šablonách je za názvem signálu umístěn vykřičník, což označuje, že jde o signál. Například, pro vytvoření odkazu, který by při aktivaci vyvolal metodu pro vytvoření modálního okna ke smazání uživatele, by bylo možné použít následující zápis:

```
<a n:href="createDeleteUserModal! $user->id">Delete user</a>
```

V presenteru by pak odpovídající metoda pro zpracování tohoto signálu vypadala takto:

```
public function handleCreateDeleteUserModal(int $userId): void
{
  // Zpracování signálu
}
```

Signál je vždy volán v kontextu aktuálního presenteru a zobrazení (view), což znamená, že nelze signál vyvolat na jiném presenteru nebo v jiném zobrazení. Toto omezení zajišťuje, že interakce mezi uživatelským rozhraním a aplikační logikou je dobře strukturovaná a snadno sledovatelná. [27]

3.16 AJAXový požadavek

AJAXový požadavek v podstatě funguje stejně jako běžný HTTP požadavek, s tím rozdílem, že je obvykle iniciálně vyvolán uživatelskou interakcí na straně klienta bez potřeby znovu načítání celé stránky. Presenter v Nette frameworku je pak zavolán s určitými parametry, podobně jako u tradičního požadavku, a je na něm, aby rozhodl, jak bude na tento požadavek reagovat.

Na serverové straně lze v presenteru rozpoznat, zda byl požadavek vyvolán prostřednictvím AJAXu, pomocí metody v presenteru *$this->isAjax()*. Tato metoda vrací *true* v případě, že aktuální požadavek je AJAXový, což umožňuje v kódu jednoduše rozhodnout o dalším postupu. Například může být užitečné vrátit pouze data potřebná pro aktualizaci určité části stránky, aniž by bylo nutné posílat kompletní HTML strukturu. [28]

3.17 Snippety

Snippety představují klíčový prvek Nette frameworku pro vytváření dynamických a interaktivních webových aplikací s minimálním úsilím.

Funkčnost snippetů spočívá v možnosti aktualizovat pouze specifické části stránky bez nutnosti jejího celkového znovu načítání. To nejen zvyšuje rychlost a efektivitu načítání stránek, ale také zlepšuje plynulost a komfort užívání aplikace.

Princip fungování snippetů je následující:

- Při prvním načtení stránky (tj. ne-AJAXovém požadavku) se načte celá stránka včetně všech definovaných snippetů.
- Pokud uživatel interaguje se stránkou (například kliknutím na tlačítko nebo odesláním formuláře) a vyvolá tím AJAXový požadavek, kód v presenteru provede požadovanou akci a určí, které snippety je potřeba aktualizovat.
- Nette následně vykreslí tyto snippety a odešle je ve formě JSON pole zpět klientovi.
- Klientský JavaScript pak zpracuje odpověď od serveru a aktualizované snippety dynamicky vloží do stránky.

Definice snippetů v kódu může probíhat dvěma způsoby:

1. Pomocí párových tagů
   ```
   {snippet snippetName}
   <!-- Obsah snippetu -->
   {/snippet}
   ```
2. Přímo v HTML tagu
   ```
   <div n:snippet="snippetName"></div>
   ```

Každý objekt odvozený od třídy Control (což zahrnuje i Presenter) v Nette frameworku si udržuje informaci o tom, zda došlo k jakýmkoliv změnám, které vyžadují jeho překreslení. Pro řízení tohoto procesu slouží metoda *redrawControl()*, která umožňuje znovu vykreslit všechny související komponenty, a tím i snippety.

V případě, že je potřeba překreslit pouze určité specifické snippety nebo komponenty, lze do metody *redrawControl()* předat parametr specifikující název snippetu, který chceme překreslit. [28]

4 JAVASCRIPTOVÁ KNIHOVNA NAJA

Pro manipulaci se snippety a obecně pro AJAXovou funkcionalitu na straně klienta v Nette aplikacích slouží JavaScriptová knihovna Naja. Tato knihovna zjednodušuje vytváření dynamických webových aplikací tím, že automaticky transformuje standardní odkazy a formuláře na AJAXové požadavky, aniž by bylo nutné psát vlastní JavaScriptový kód.

Pro použití Naja pro převod obyčejných odkazů nebo formulářů na AJAXové požadavky stačí do atributu *class přidat hodnotu ajax*. Příklad pro vytvoření AJAXového odkazu, který by po aktivaci uživatelem vyvolal signál pro vygenerování modálního okna, ve kterém by byla možnost smazání uživatele

```
<a n:href="createDeleteUserModal! $user->id" class="ajax">Delete user</a>
```

Tímto způsobem je možné velmi snadno převést jakýkoliv standardní odkaz nebo formulář na AJAXový požadavek bez nutnosti psát další kód pro jeho zpracování na straně klienta. Naja se postará o vše potřebné – od odeslání požadavku přes zpracování odpovědi až po aktualizaci relevantních částí stránky pomocí snippetů, které byly na serveru označeny k překreslení. [28]

4.1 Inicializace Naja

Naja je samostatná JavaScriptová knihovna určená pro snadnou integraci AJAXu do webových aplikací. Instalace a zprovoznění Naja je velmi přímočaré – stačí knihovnu stáhnout a poté ji v JavaScriptovém kódu inicializovat. Od té chvíle se Naja postará o základní manipulaci s AJAXovými požadavky a nabídne podporu pro různé funkce. [29]

4.2 Naja historie

Jednou z významných vlastností Naja je automatická podpora pro práci s historií prohlížeče. V základním nastavení, když Naja zpracovává AJAXový požadavek, dochází k aktualizaci URL v adresním řádku prohlížeče, což umožňuje uživatelům používat tlačítko zpět a vpřed ve svém prohlížeči a zároveň udržuje aplikaci přístupnou pro sdílení odkazů.

Pokud je potřeba aby se URL neaktualizovala je možnost historii vypnout. Toho se docílí za použití atributu *data-naja-history* s hodnotou *off* přímo v elementu, který AJAXový požadavek spouští. [30]

```
<a href="deleteUser! $user->id" class="ajax" data-naja-history="off">Delete user</a>
```

5 PHP KINOVNA NEXTRAS

Nextras představuje sadu rozšiřujících balíčků pro framework Nette, které jsou navrženy tak, aby výrazně usnadnily a zefektivnily práci s databázemi, formuláři a dalšími běžnými úkony při vývoji webových aplikací.

Hlavní oblasti, na které se Nextras zaměřuje, zahrnují:

- Orm (Object-Relational Mapping): Knihovna poskytující pokročilé možnosti pro mapování databázových tabulek na objekty v PHP.
- Dbal (Database Abstraction Layer): Abstraktní vrstva pro databáze, která umožňuje jednodušší a bezpečnější práci s databázovými dotazy.
- Migrace: Nástroj pro správu a automatizaci migrací databáze, což zahrnuje vytváření, modifikaci a mazání tabulek nebo sloupců.

[31]

5.1 Nextras Dbal

Nextras Dbal je nástroj poskytující stručné a zároveň bezpečné API pro práci s databázemi. Jeho hlavním cílem je umožnit vytváření databázových dotazů a načítání dat z různých typů úložišť deklarativním a intuitivním způsobem, přičemž je nezávislý na konkrétním typu použité databáze. Dbal podporuje populární databázové systémy jako MySQL, PostgreSQL a MS SQL Server.

Centrálním bodem pro přístup k databázi v Nextras Dbal je instance třídy *Connection*, která slouží jako hlavní interface pro všechny databázové operace. Tato instance je vytvářena pomocí konstruktoru, do kterého se předává konfigurační pole s detaily pro připojení k databázi.

Integrace Nextras Dbal do Nette aplikací je zjednodušena díky Nextras DI Extension, která umožňuje rychlé a snadné nastavení prostřednictvím konfiguračního souboru Neon. [32]

Pro nastavení Dbal v Nette aplikaci stačí do souboru neon doplnit následující konfiguraci:

```
extensions:
  nextras.dbal: Nextras\Dbal\Bridges\NetteDI\DbalExtension

nextras.dbal:
  driver: mysqli
  host: localhost # Uveďte hostitele
  database: nazev_databaze # Uveďte název databáze
  username: uzivatelske_jmeno # Uveďte přihlašovací jméno
  password: heslo # Uveďte přihlašovací heslo
  connectionTz: # Nastavte časovou zónu připojení,
```

Obrázek 8. Konfigurace nextras dbal [32]

5.2 Nextras Orm

Nextras Orm je moderní knihovna pro objektově-relační mapování (ORM), která je navržena s důrazem na vysokou efektivitu a snadnost použití. Tato knihovna je částí ekosystému Nextras a je speciálně vytvořena pro práci s Nette frameworkem, i když může být použita i mimo něj. Díky své flexibilitě a výkonnosti je vhodná pro širokou škálu aplikací, od jednoduchých projektů až po rozsáhlé webové aplikace. [33]

Všechny tyto vrstvy jsou propojeny ve středové třídě *Model*. Každá entita v rámci Orm musí mít definovaný svůj vlastní repositář a mapper. Toto uspořádání umožňuje srozumitelné rozdělení zodpovědností a podporuje snadné rozšíření a údržbu aplikace. [34]

5.2.1 Architektura

Nextras Orm je strukturován do tří hlavních vrstev, aby efektivně oddělil entity od implementace databáze. Tento přístup umožňuje čistou a modulární architekturu aplikací, která zjednodušuje vývoj a údržbu. Každá z těchto vrstev má svou specifickou roli v rámci ORM. [34]

5.2.1.1 Entity

Entity jsou základní stavební bloky modelu, které reprezentují data jednoho řádku tabulky. Každá entita musí implementovat rozhraní *Nextras\Orm\Entity\IEntity*. Orm nabízí předdefinovanou třídu *Nextras\Orm\Entity\Entity*, která toto rozhraní implementuje a poskytuje další užitečné funkce. Entity slouží jako kontejnery dat a definují strukturu a vztahy mezi daty. [34]

5.2.1.2 Repositáře

Repositáře tvoří vrstvu, která se stará o entity. Repositáře spravují entity a poskytují API pro jejich načítání, filtrování a ukládání (persistenci). Repositáře jsou klíčové pro manipulaci s entitami, neboť poskytují abstrakci nad databázovými operacemi a umožňují vývojářům pracovat s daty na vyšší úrovni abstrakce. [34]

5.2.1.3 Mappery

Mappery jsou backendem Orm. Poskytují interakci s databázovou vrstvou a zajišťují komunikaci mezi aplikací a databází. Pro abstrakci spojení s databází využívá Nextras Orm knihovnu Nextras\Dbal, což umožňuje efektivní a flexibilní práci s různými typy databází. [34]

II. PRAKTICKÁ ČÁST

6 AKTUÁLNĚ EXISTUJÍCÍ ŘEŠENÍ

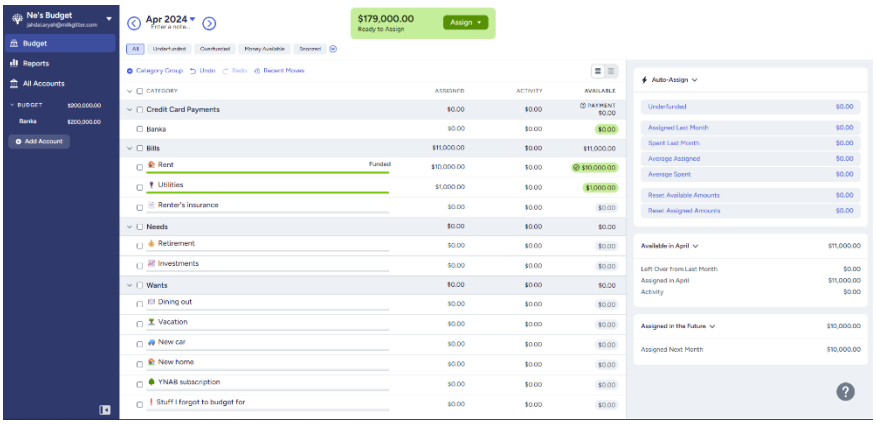
V první kapitole praktické části své bakalářské práce jsem se zabýval zkoumáním aktuálně existujících řešení v oblasti správy osobních financí. Narazil jsem na celou řadu problémů, které výrazně omezily výběr vhodných kandidátů. Jeden z hlavních problémů byla lokalizace – zjistil jsem, že mnohé zahraniční webové stránky, jako například NerdWallet, jsou dostupné pouze pro určité regiony bez možnosti načtení dat pro uživatele z jiných oblastí. Dalším problémem byly požadavky na zadání osobních údajů, jako je číslo pojištění nebo telefonní číslo, což zásadně eliminovalo potenciální kandidáty z mého výběru.

Prozkoumal jsem celou řadu aplikací nabízejících služby správy osobních financí. Mezi ně patřily YNAB (You Need A Budget), PocketSmith v demo režimu a Spendee. Pro svou práci jsem následně zvolil aplikaci Wallet, na základě řady doporučení. Ta se stala klíčovou inspirací pro moji bakalářskou práci, zejména v uživatelské přívětivosti a široké škále nabízených funkcí.

6.1 YNAB

YNAB, zkratka pro "You Need A Budget", je aplikace pro správu osobních financí s důrazem na rozpočtování a finanční plánování. Metoda, na které YNAB stojí, vede uživatele k zodpovědnému přístupu k životu, na základě reálných příjmů, jim umožňuje plánovat výdaje a spořit na budoucí cíle či nepředvídané situace. Toto zaměření na rozpočtovou část z YNAB činí silný nástroj pro ty, kdo hledají pomoc při zlepšování své finanční disciplíny a usilují o lepší přehled o svých financích.

Přestože je YNAB často doporučováno na internetu jako užitečný nástroj pro správu financí, pro moje specifické potřeby bylo vyhodnoceno jako nevhodné.

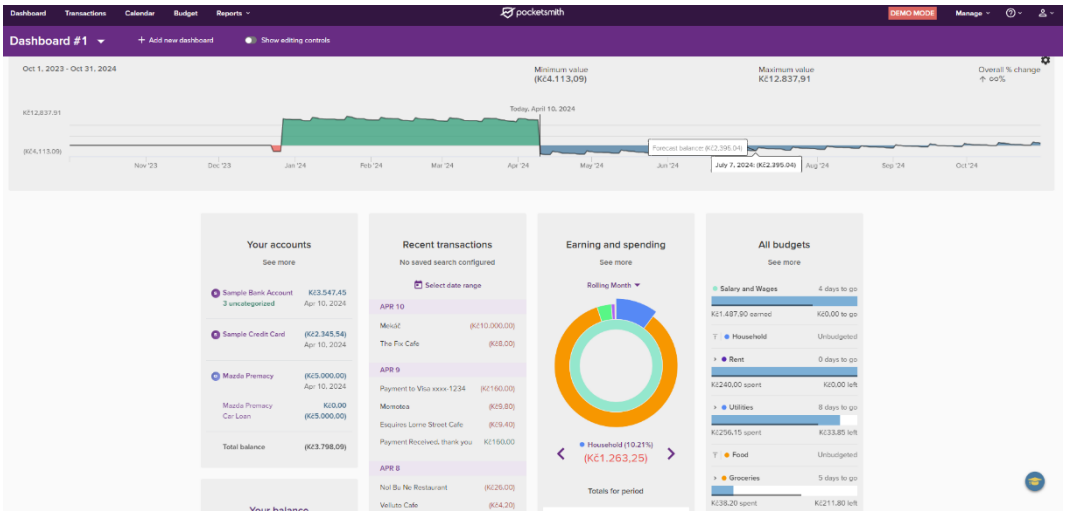


Obrázek 9. Dashboard webové aplikace YNAB

6.2 PocketSmith

PocketSmith umožňuje uživatelům efektivně monitorovat jejich příjmy, výdaje, majetek a čistou hodnotu, poskytuje flexibilitu v personalizaci rozpočtů a umožňuje předpovídání toku peněz až na 60 let dopředu, přičemž všechny tyto informace si lze snadno prohlížet v přehledném kalendáři. Aplikace přichází s řadou tarifních plánů, včetně bezplatné verze, která podporuje správu až dvou účtů s možností manuálního zadávání dat, umožňuje vytvořit až 12 rozpočtů a nabízí šestiměsíční finanční prognózu. Placené verze pak rozšiřují možnosti o automatické stahování bankovních transakcí, e-mailovou podporu, možnost propojení bankovních účtů z různých zemí a přístup k rozšířeným přehledům.

PocketSmith působí jako nejkomplexnější z dostupných řešení. Nabízí širokou škálu nástrojů pro správu financí. Avšak, jak je patrné z vizuálního přehledu na obrázku 10, uživatelské rozhraní může být méně přehledné kvůli přítomnosti mnoha widgetů různých velikostí na dashboardu, které jsou jen omezeně přizpůsobitelné. Například, i když je možné upravit dashboard na třísloupcové zobrazení z původní čtyřsloupcové konfigurace, šířka a výška sloupců zůstává nezměněna.



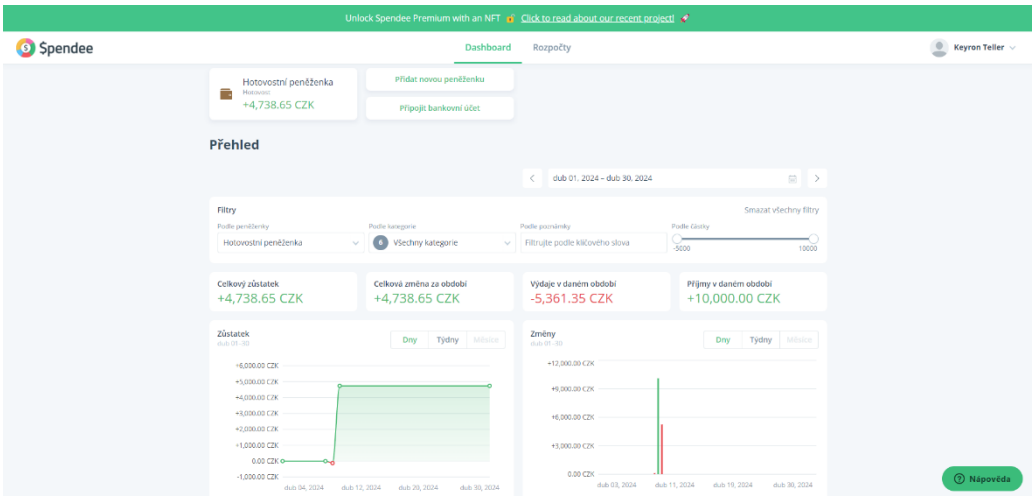
Obrázek 10 – Dashboard webové aplikace PocketSmith

6.3 Spendee

Spendee je webová aplikace zaměřená na správu osobních financí, která se vyznačuje jednoduchou registrací, přívětivým a čistým uživatelským rozhraním. Aplikace umožňuje snadné vytváření rozpočtů pro různé kategorie a intuitivní zadávání nových transakcí. Kategorie v aplikaci jsou jasně rozděleny na příjmy a výdaje, což usnadňuje sledování finančního

toku. Tyto vlastnosti činí Spendee atraktivní volbou pro ty, kteří hledají efektivní nástroj pro každodenní správu svých financí.

Aplikace má i své nevýhody. Pro uživatele, kteří chtějí sdílet účet s dalšími uživateli, vytvářet více než jeden rozpočet, spravovat více účtů nebo využívat automatické příkazy, je nezbytná placená verze. Tato omezení mohou být pro některé uživatele významnou překážkou. Dále, transakce v aplikaci jsou omezeny pouze na kategorie, což sice zjednodušuje jejich zadávání, ale může komplikovat zpětné vyhledávání specifických informací. Tento fakt, společně s tím, že transakce nelze vytvářet přímo z hlavního dashboardu může některé funkcionality aplikace omezit.



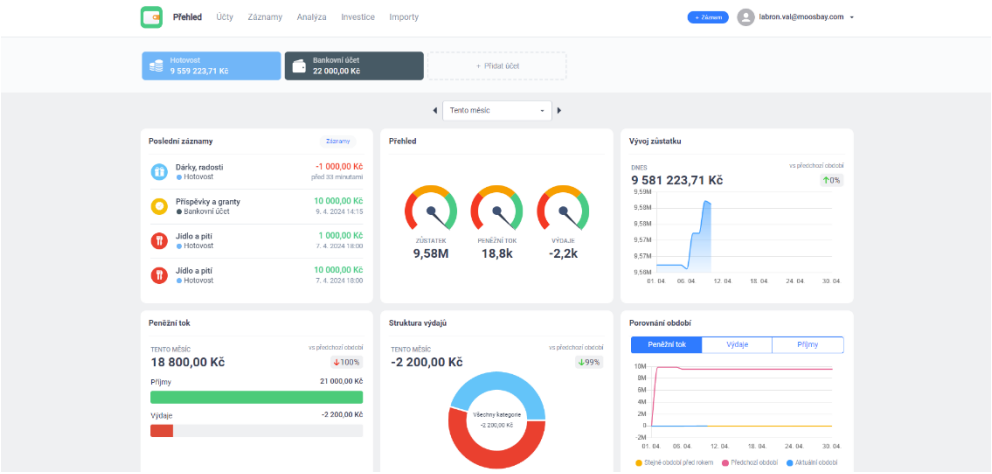
Obrázek 11. Dashboard webové aplikace Spendee

6.4 Wallet by BudgetBakers

Wallet je mobilní a webová aplikace vyvinutá Janem Müllerem z České republiky. [35] Design aplikace je minimalistický a usiluje o maximální přehlednost, což umožňuje uživatelům vytvářet transakce z jakéhokoli místa na webu s lehkostí a bez zbytečného hledání. V základní bezplatné verzi nabízí aplikace podporu až pro tři účty, neomezený počet transakcí, přehledné filtry pro snadné třídění transakcí a nástroje pro detailní analýzy.

Premium verze přidává funkce pro vytváření automatických transakcí a možnost sdílení účtů mezi více uživateli, což může být obzvláště užitečné pro rodinné finance nebo malé podniky.

Z uvedených aplikací považuji Wallet za nejlepší volbu v bezplatné kategorii, jelikož poskytuje rozsáhlé možnosti bez jakýchkoli dalších nákladů.



Obrázek 12. Dashboard webové aplikace Wallet

7 NÁVRH VLASTNÍHO ŘEŠENÍ

Pro návrh vlastního řešení byla zvolena aplikace Wallet jako východisko pro inspiraci, nicméně se zaměřením na eliminaci omezení, která jsou spojena s její placenou verzí. Cílem aplikace ExpenseInsight je zahrnout intuitivní správu transakcí s podporou pro základní CRUD (Create, Read, Update, Delete) operace, s rozšířením o schopnost automatizace opakovaných transakcí na základě nastavitelného období.

Jednou z klíčových funkcionalit, které byly považovány za esenciální, je možnost sdílení finančních účtů s neomezeným počtem osob, což rozšiřuje možnosti spolupráce mezi uživateli. Součástí je i tvorba šablon pro snadné generování pravidelných plateb, což zvyšuje efektivitu a šetří čas uživatelů.

S ohledem na osobní preference uživatelů byla zařazena podpora pro oba vizuální režimy – světlý a tmavý. To přispívá ke zlepšení uživatelského komfortu v jakémkoliv osvětlení.

Na úvodní stránce (dashboardu) byla klíčová flexibilita v možnosti personalizace zobrazení widgetů, které mají uživatelé možnost zobrazovat, skrývat a upravovat podle svých potřeb. Nechybí ani funkce pro výběr časového rozsahu, ve kterém se budou data widgetů prezentovat.

Další, pro uživatelský komfort podstatný prvek návrhu, byla implementace funkcionalit pro výběr a filtrování dat na dashboardu podle konkrétního účtu, a možnost přepínání mezi zobrazením jednoho nebo všech účtů.

Název ExpenseInsight, byl zvolen ve spolupráci s umělou inteligencí ChatGPT. Reflektuje klíčovou hodnotu aplikace – poskytovat uživatelům hlubší přehled o jejich nákladech. Tento název se ideálně hodí pro koncept aplikace, jejíž hlavním cílem je transparentnost a přehlednost ve financích.

7.1 Návrh UI

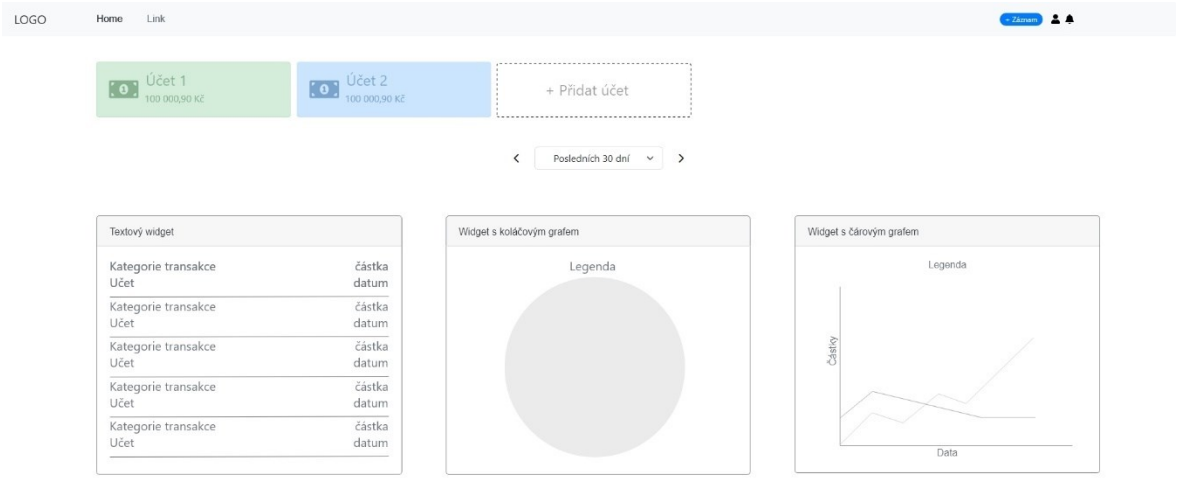
Při vytváření designu webového rozhraní jsem využil program Adobe XD, kam jsem importoval styly pro Bootstrap 5.3 a ikony z font-awesome.

7.1.1 Návrh Dashboardu

Inspiraci pro design dashboardu jsem našel v aplikaci Wallet, jehož jednoduchost a přehlednost se ukázaly jako ideální východisko. Ve svém návrhu jsem přizpůsobil rozvržení políček s účty tak, aby byly na rozdíl od kompaktnějšího uspořádání v aplikaci Wallet, vyšší a

poskytovaly více prostoru. Zaměřil jsem se také na barevné schéma s cílem, aby barvy políček odpovídaly barvě přiřazené danému účtu, s odstínem na pozadí a tmavší variantou pro text.

Rozmístění widgetů bylo zvoleno s ohledem na různá zařízení: na desktopu tři widgety v řadě, na tabletu dva a na mobilním telefonu jeden. Ve vizualizaci lze vidět návrhy pro tři hlavní typy widgetů – textový widget zobrazující přehled transakcí, widget s koláčovým grafem reprezentující kategorizaci výdajů či příjmů a widget s čárovým grafem, který ukazuje vývoj financí v čase.



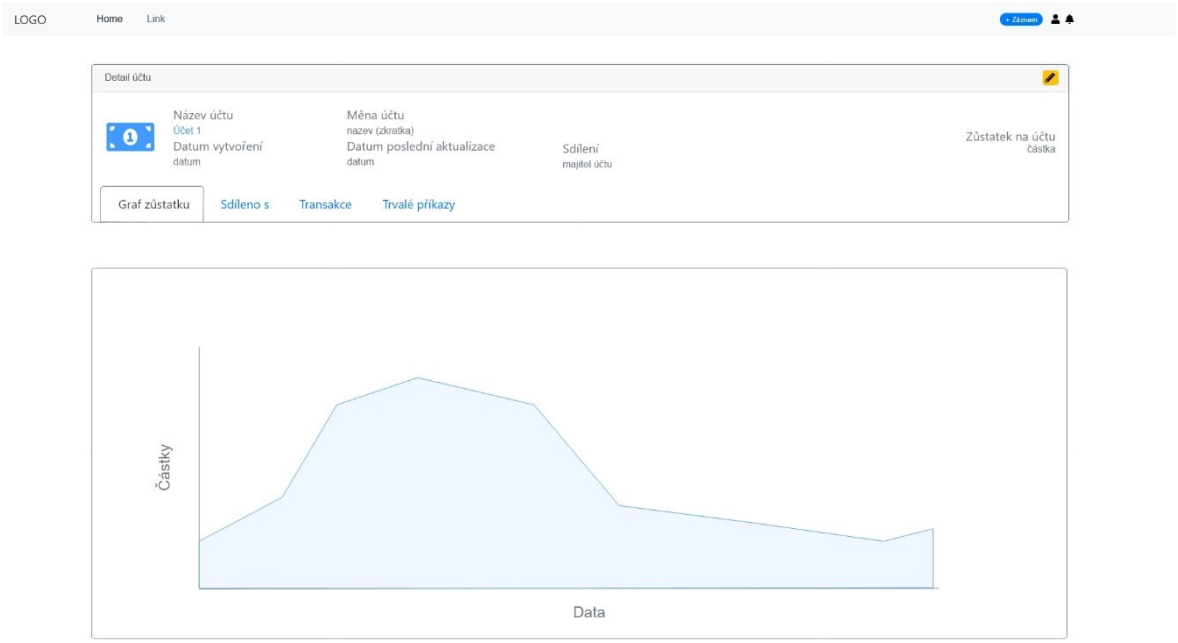
Obrázek 13. Návrh Dashboardu

7.1.2 Návrh detailu účtu

Design detailní stránky účtu byl vytvořen s ohledem na přehlednost a intuitivní ovládání. Pro vrchní část stránky jsem zvolil Bootstrapovou komponentu "card", která esteticky prezentuje klíčové informace účtu. V této sekci se projeví vybraná barva účtu, která je aplikována na ikonu účtu, název účtu a příslušný graf.

Detaily účtu zahrnují název, datum vytvoření, datum poslední aktualizace, měnu účtu a aktuální zůstatek. Kromě toho jsou v této sekci k dispozici ovládací tlačítka, která umožňují uživatelům rychlý přístup k dalším funkcím spojeným s účtem – zobrazení grafu zůstatku, přehledu transakcí a trvalých příkazů. Pro majitele účtu je zde navíc přístupné tlačítko „Sdíleno s“, které poskytuje informace o sdílení účtu s ostatními uživateli.

Dolní část stránky je navržena jako flexibilní prostor. Zde se po kliknutí na ovládací prvky z horní sekce zobrazují příslušné interaktivní panely. Tato oblast je dynamicky aktualizována a poskytuje informace v závislosti na zvolené akci uživatele.

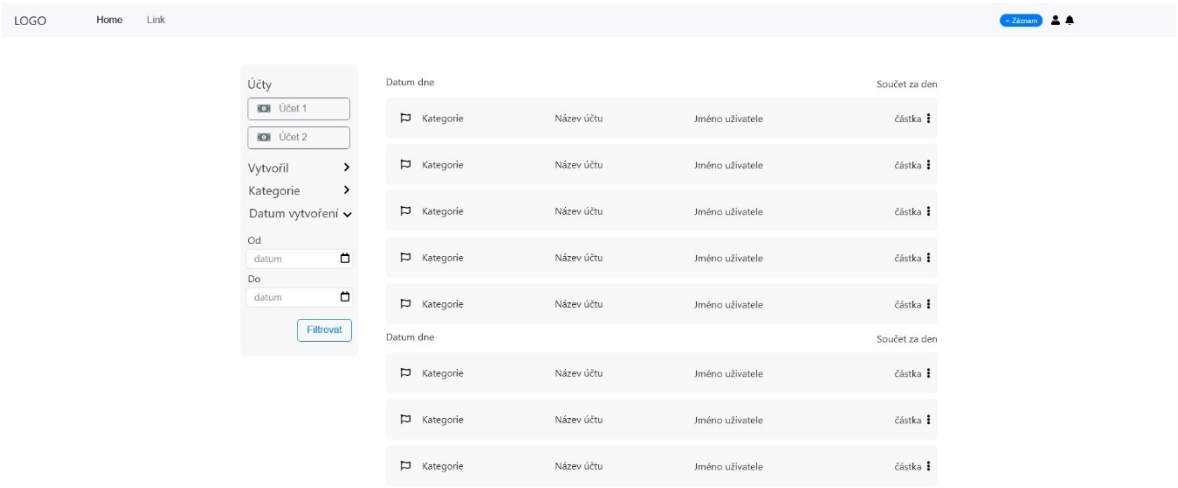


Obrázek 14. Návrh detailu účtu

7.1.3 Návrh přehledu transakcí

Stránka s přehledem transakcí byla koncipována s důrazem na efektivní filtrování a uspořádání zobrazených dat. V levé části uživatelského rozhraní se nachází filtry, umožňující uživatelům personalizovat seznam transakcí podle různých kritérií. Uživatelé mohou vybírat transakce na základě specifického účtu, autora transakce, kategorie, či časového období.

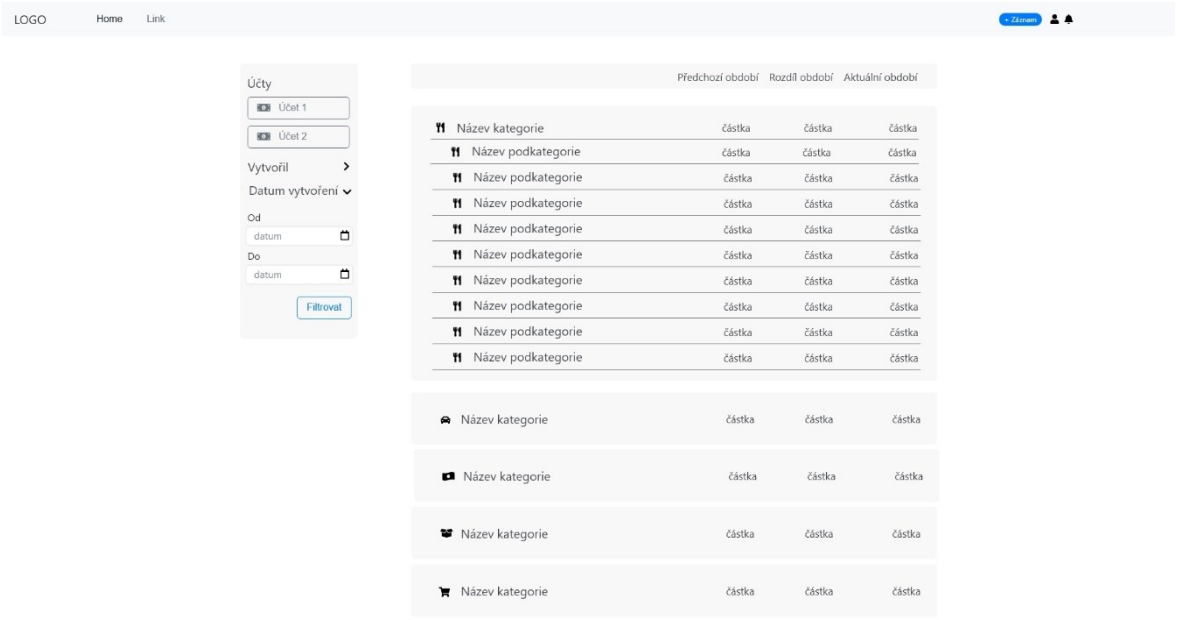
V pravé části stránky jsou transakce systematicky rozvrženy podle data provedení s uvedením souhrnné denní sumy, což poskytuje rychlý přehled o finančních pohybech. Pod každým datem následuje seznam jednotlivých transakcí včetně informací o kategorii, názvu účtu, jménu uživatele a výši finanční transakce.



Obrázek 15. Návrh přehledu transakcí

7.1.4 Návrh analýzy

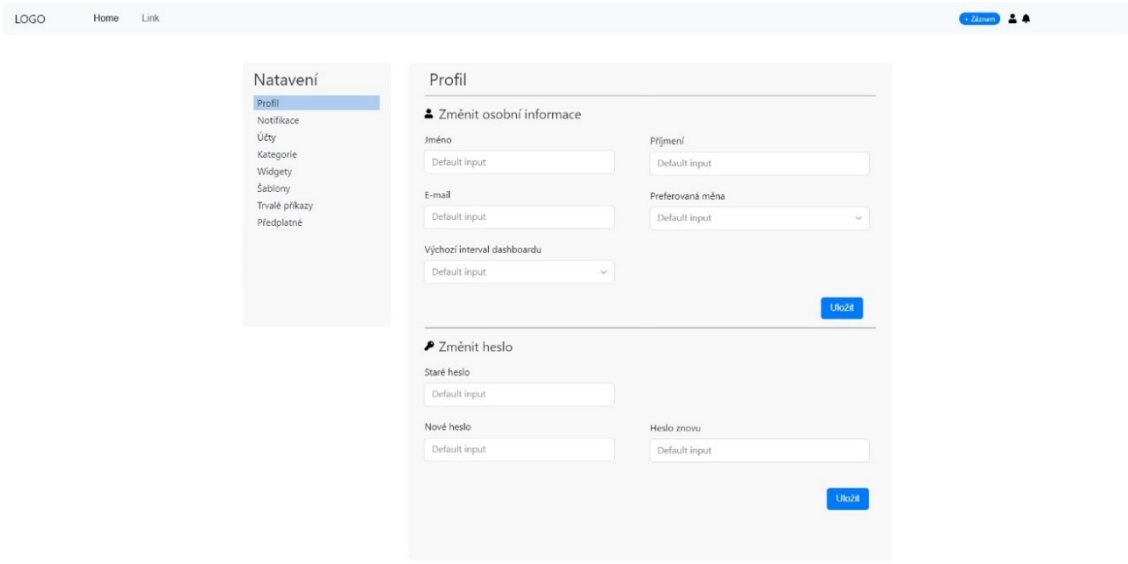
Návrh stránky analýzy nabízí podobnou filtraci jako u filtrace transakcí, s tím rozdílem, že zde chybí filtrace kategorií. Fokus je zde určen na vizualizaci finančních dat v kategoriích, s možností rozkliknout podkategorie pro podrobnější informace. Interaktivní prvky návrhu umožňují srovnání výdajů mezi aktuálním a předchozím obdobím.



Obrázek 16. Návrh analýzy

7.1.5 Návrh nastavení

V sekci Nastavení je implementován boční navigační panel. Ten usnadňuje přechod mezi jednotlivými konfiguračními možnostmi aplikace. Pro editaci profilu je navržen uživatelský formulář, který umožňuje aktualizaci osobních informací, jako jsou jméno, příjmení, e-mail a preferovaná měna, stejně jako výchozí nastavení pro interval zobrazení na dashboardu.

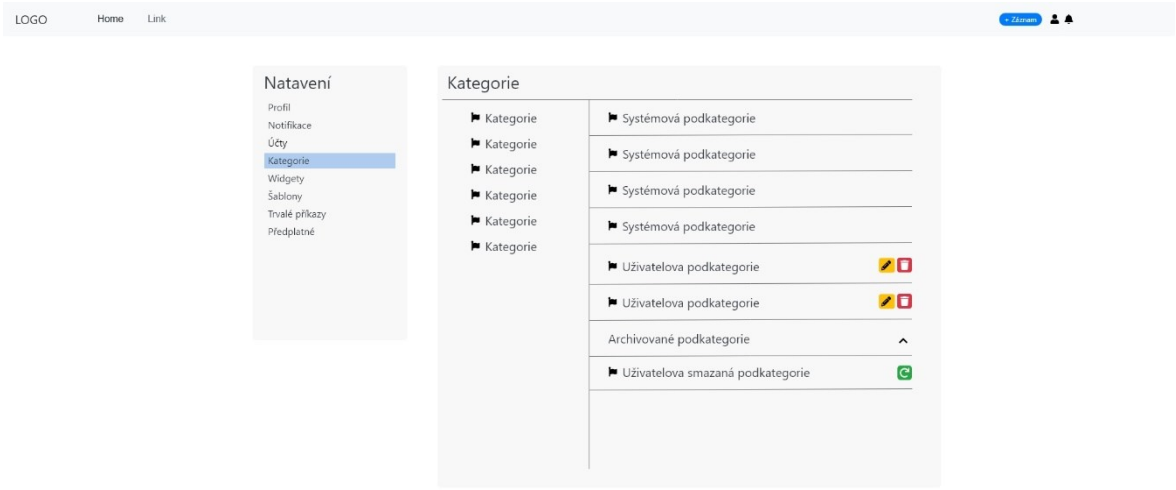


Obrázek 17. Návrh úpravy profilu

Další část návrhu se věnuje prezentaci účtů. Zde jsou data každého účtu přehledně řazena do tabulkového formátu, doplněna o funkce pro úpravu pořadí zobrazení účtů na hlavní stránce pomocí táhla.

Obrázek 18. Návrh seznamu účtů

Závěrečná část designu je zaměřena na správu kategorií. Uživatelům umožňuje vytvářet vlastní podkategorie, které jsou přiřazeny k hlavním kategoriím. Usnadňuje také editování a odstraňování těchto podkategorií. Archivované podkategorie, které mají asociované transakce, jsou stále zobrazeny v sekci archivováno.



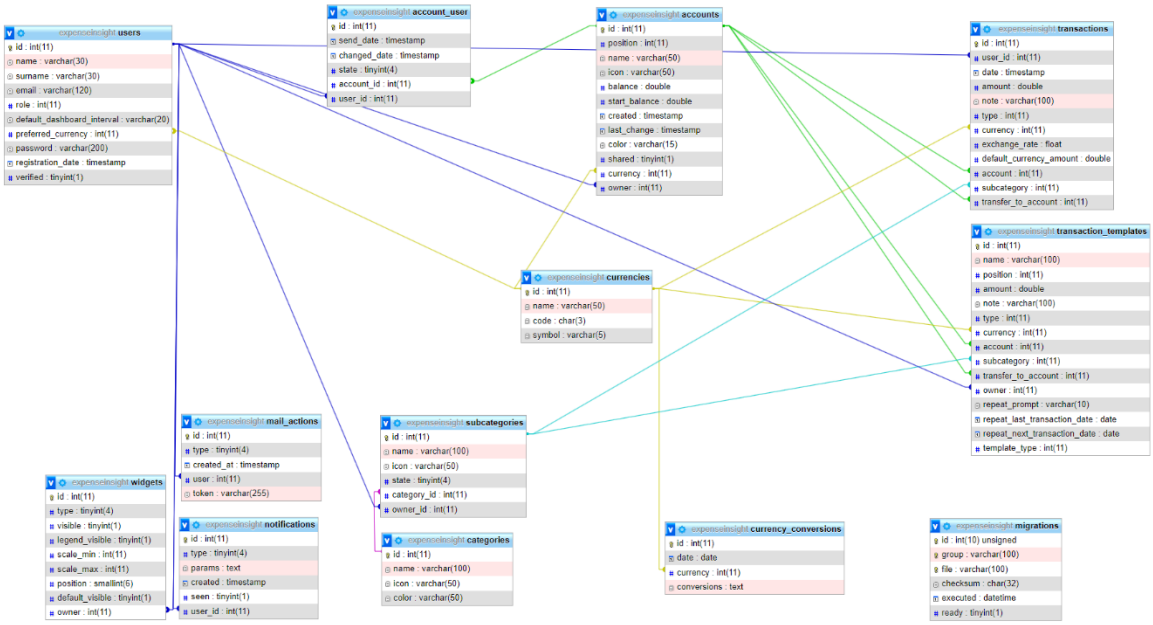
Obrázek 19. Návrh nastavení podkategorie

7.2 Návrh databáze

Databázové schéma navržené pro aplikaci poskytuje komplexní rámec pro správu a analýzu finančních dat. Zahrnuje následující tabulky:

- users: Uchovává profilové informace uživatelů, včetně autentizačních údajů a nastavení aplikace.
- accounts: Reprezentuje účty vlastněné uživatelem, s detaily o účtu a celkovém zůstatku.
- account_user: Mapuje vztah mezi uživateli a účty, které nevlastní, ale které jsou s nimi sdíleny.
- transactions: Zaznamenává finanční transakce prováděné na účtech.
- transaction_templates: Obsahuje šablony transakcí, včetně opakovaných transakcí a předplatných.
- currencies: Obsahuje seznam podporovaných měn aplikací.
- currency_conversions: Ukládá historická data o měnových konverzích pro různé dny a měny.
- categories a subcategories: Tyto tabulky uchovávají kategorie a podkategorie pro klasifikaci transakcí, kde uživatelé mohou vytvářet vlastní podkategorie.
- mail_actions: Sleduje e-mailové akce, jako jsou požadavky na změnu hesla či e-mailové verifikace, spolu s tokeny pro ověřování těchto akcí.
- notifications: Uchovává uživatelské notifikace související s aktivitou účtu.

- widgets: Umožňuje personalizaci nastavení widgetů, které si uživatel zvolí na svém dashboardu.



Obrázek 20. Návrh databáze

8 IMPLEMENTACE VLASTNÍHO ŘEŠENÍ

Pro základ webové aplikace byl zvolen framework Nette. Výběr byl motivován předešlými zkušenostmi a ambicí překonat v rámci bakalářské práce kvalitu předchozích projektů. K obohacení uživatelského zážitku byla integrována JavaScriptová knihovna Naja, která je doplněna o Chart.js pro efektivní tvorbu grafů. Pro stylování stránek byl zvolen Bootstrap ve verzi 5.3.3, využívající jeho nové funkce jako je podpora pro světlý a tmavý režim, což přináší moderní vzhled a flexibilitu uživatelského rozhraní. Pro multijazyčnou podporu bylo do backendu začleněno *contributte/translation*, umožňující aplikaci fungovat jak v anglickém, tak v českém jazyce. Pro efektivní manipulaci s daty bylo využito Nextras ORM, jehož jednoduchost implementace a schopnost zpracovávat složité databázové dotazy poskytují aplikaci vysoký výkon. Jako databázový systém byl zvolen MySQL, preferovaný pro jeho uživatelskou přívětivost a dostatečnou kapacitu pro potřeby bakalářské práce.

8.1 Struktura projektu

Projekt není modulový. Všechny důležité soubory jsou uspořádány v hlavním modulu, který se dále člení převážně do presenterů. Základní Nette adresářovou strukturu (viz Obrázek 3) jsem doplnil o celou řadu složek, které jsem uznal za vhodné, pro lepší přehlednost.

Assets: Tato složka obsahuje JavaScriptové a SCSS soubory, které je nutné před nasazením do produkčního prostředí překompilovat. Výsledné soubory jsou umístěny ve složce www.

Components: V této složce jsou uloženy menší samostatné celky rozdělené do podsložek *Controllers*, *Forms*, *Widgets* a *Mails*. Obsahují třídy zajišťující funkcionalitu jednotlivých controllerů a příslušné Latte šablony, které je možné dynamicky vytvářet na libovolném presenteru.

Helpers: Zde se nachází vlastní Latte filtry, včetně HTML helperu, který například formátuje finanční částky a zaokrouhluje je na předem daný počet desetinných míst.

Langs: Složka obsahuje lokalizační soubory ve formátu *nazev.jazyk.neon*, kde každý jazyk reprezentuje jednu lokalizaci. V tomto případě aplikace podporuje češtinu a angličtinu.

Model: Tato složka představuje datový model aplikace vytvořený pro Nextras ORM a obsahuje Entities, Mappers a Repositories. Dále se zde nachází soubor *Model.php*, který slouží jako centrální manažer pro správu repositářů a mapperů dat. V podsložce Models jsou umístěny modely, které nejsou přímo spojeny s databází, ale slouží pro zjednodušení práce s daty, jako je například struktura pro dialogová okna s výběrem.

Migrations: Složka obsahuje podsložky s migračními skripty, které jsou založeny na SQL souborech spustitelných prostřednictvím *migrations.php*.

node_modules: Složka obsahující moduly nainstalované pomocí *npm*, které jsou využívány především ve vývojovém prostředí pro správu závislostí a build procesů.

8.2 Konfigurační soubory Neon

Veškeré konfigurační soubory ve formátu NEON jsou organizovány v adresáři *config*. Tento adresář zahrnuje čtyři základní soubory – *common*, *local*, *production* a *services*, které společně tvoří kompletní nastavení prostředí a služeb aplikace.

8.2.1 Common

Soubor představuje základní konfiguraci pro Nette framework, včetně specifikací jako jsou doba platnosti session, striktní typová kontrola v Latte, nastavení presenteru pro ošetření chyb a export konfigurace závislostí (DI).

8.2.2 Local

V tomto souboru jsou uloženy přístupové údaje pro lokální vývojové prostředí. Konfigurace zahrnuje parametry pro Nextras DBAL, jako jsou typ databázového driveru, hostitel, název databáze, přihlašovací jméno a heslo, a také *connectionTz* pro nastavení časové zóny aplikace.

8.2.3 Production

Soubor *production* obsahuje přístupové údaje k produkční databázi, které jsou nastaveny podobně jako v souboru *local*, ale s informacemi specifickými pro provozní prostředí.

8.2.4 Services

Nejrozsáhlejší soubor, zahrnuje definice rozšíření (extensions), které konfigurují funkcionality pro překlady, Nextras DBAL, ORM a migrace. Dále obsahuje specifikaci služeb (services), včetně cest k třídám a komponentám, jako jsou *RouterFactory*, *HelpersHtml* a *formFactory*, které jsou do aplikace zavedeny prostřednictvím závislostí. Zde je také definována cesta k modelu pro třídy využívající Nextras ORM, nastavení jazykových překladů včetně podporovaných jazyků a cest k překladovým souborům. Poslední sekce souboru *services* se věnuje konfiguraci migrací.

8.3 Datový model

Pro manipulaci s daty v aplikaci jsem zvolil Nextras ORM, což je nástroj, který umožňuje efektivní a jednoduchou práci s datovými objekty. Klíčovou součástí Nextras ORM je soubor *Model.php*. Ten slouží jako centrální místo pro správu repositářů (viz Obrázek 21). Je důležité, aby každý repositář měl svůj odpovídající mapper a entitu se shodným názvem, což zajistí korektní mapování a přístup k datům v databázi.

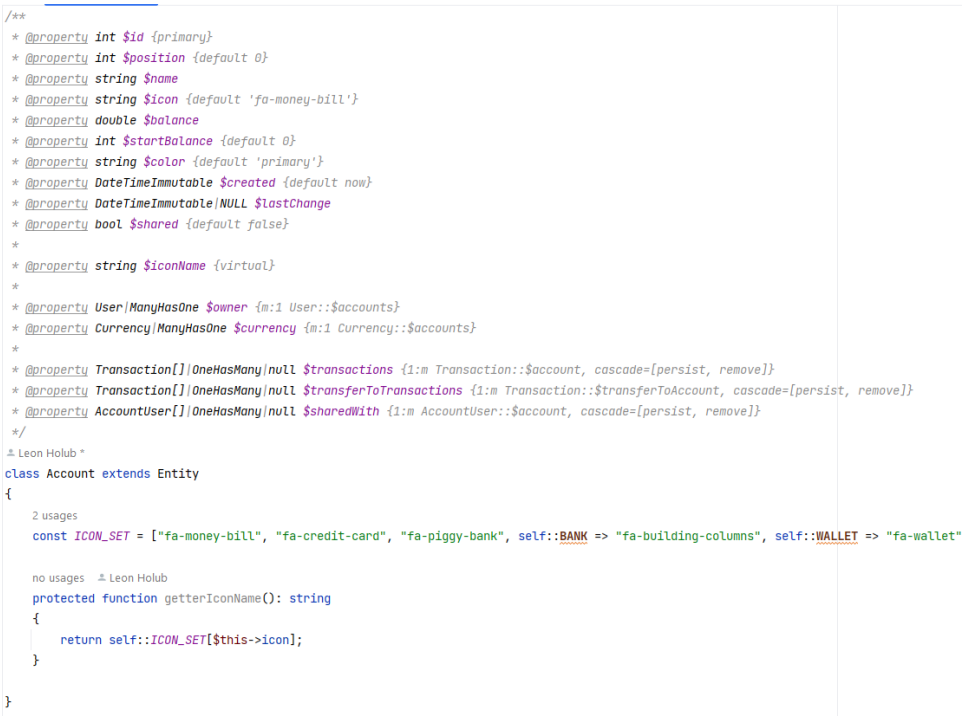
```php
use Nextras\Orm\Model\Model;
/**
 * @property-read UserRepository $userRepository
 * @property-read AccountRepository $accountRepository
 * @property-read TransactionRepository $transactionRepository
 * @property-read CategoryRepository $categoryRepository
 * @property-read SubcategoryRepository $subcategoryRepository
 * @property-read CurrencyRepository $currencyRepository
 * @property-read CurrencyConversionRepository $currencyConversionsRepository
 * @property-read TransactionTemplateRepository $transactionTemplateRepository
 * @property-read AccountUserRepository $accountUserRepository
 * @property-read NotificationRepository $notificationRepository
 * @property-read WidgetRepository $widgetRepository
 * @property-read MailActionRepository $mailActionRepository
 */
Leon Holub
class Orm extends Model
{

}
```

Obrázek 21. Obsah souboru *Model.php*

V rámci implementace softwarového projektu jsou klíčové komponenty jako repositáře, mappery a entity nezbytné pro správnou funkčnost a strukturu aplikace. Tyto komponenty se liší primárně v atributech a datových typech, což vyžaduje pečlivé navržení každé z nich, aby odpovídaly specifickým potřebám aplikace. Jako příklad významné implementace jsem zvolil komponentu Account, která demonstruje rozsáhlé využití možností frameworku Nextras ORM.

8.3.1 Entita účtu

Jak lze vidět (viz Obrázek 22) entita účtu, pojmenovaná anglicky Account, je přístupná prostřednictvím definovaných vlastností, které jsou specifikovány anotacemi v syntaxi PhpDoc, zapsanými mezi /** a */. Každá proměnná je opatřena anotací *@property*, signalizující atribut, který může, ale nemusí být přímo součástí databázové struktury.



```php
/**
 * @property int $id {primary}
 * @property int $position {default 0}
 * @property string $name
 * @property string $icon {default 'fa-money-bill'}
 * @property double $balance
 * @property int $startBalance {default 0}
 * @property string $color {default 'primary'}
 * @property DateTimeImmutable $created {default now}
 * @property DateTimeImmutable|NULL $lastChange
 * @property bool $shared {default false}
 *
 * @property string $iconName {virtual}
 *
 * @property User|ManyHasOne $owner {m:1 User::$accounts}
 * @property Currency|ManyHasOne $currency {m:1 Currency::$accounts}
 *
 * @property Transaction[]|OneHasMany|null $transactions {1:m Transaction::$account, cascade=[persist, remove]}
 * @property Transaction[]|OneHasMany|null $transferToTransactions {1:m Transaction::$transferToAccount, cascade=[persist, remove]}
 * @property AccountUser[]|OneHasMany|null $sharedWith {1:m AccountUser::$account, cascade=[persist, remove]}
 */
 Leon Holub *
class Account extends Entity
{
    2 usages
    const ICON_SET = ["fa-money-bill", "fa-credit-card", "fa-piggy-bank", self::BANK => "fa-building-columns", self::WALLET => "fa-wallet"

    no usages  Leon Holub
    protected function getterIconName(): string
    {
        return self::ICON_SET[$this->icon];
    }

}
```

Obrázek 22. Implementace entity účtu

Primární klíč je identifikován pomocí závorek *{primary}*, což indikuje jeho automatické inkrementování. U atributu *$position* použití *{default 0}* naznačuje, že v případě nezadání hodnoty pozice bude do databáze automaticky vložena hodnota 0. Tento princip lze aplikovat i na další defaultní hodnoty, jako jsou textové řetězce nebo aktuální datum a čas pomocí klíčového slova *now*.

Vlastnost *$iconName* má přiřazenou anotaci *{virtual}*, což představuje atribut, jenž není přímo součástí databáze. Pro takové atributy je nutné definovat funkci *getter* s názvem *getterNazevVlastnosti()*, která vrací odpovídající datový typ. V tomto případě funkce vrací název ikony z předem definovaného setu hodnot.

Vlastnosti entit mohou být navíc spojeny s jinými entitami, což umožňuje modelovat vztahy mezi daty v databázi. Jako příklad lze uvést vztah účtu a uživatele, kde vlastnost *$owner* v entitě účtu odkazuje na entitu uživatele. Tato vlastnost je typu *ManyHasOne*. To indikuje, že mnoho účtů může sdílet jednoho majitele, tedy jeden uživatel může mít více účtů. Tento model vztahu vyžaduje, aby v entitě, na kterou se odkazuje, existovala zrcadlová vlastnost typu *OneHasMany*. To znamená, že v entitě *User* musí být definována vlastnost, která ukazuje na množství účtů, které tento uživatel vlastní.

```
* @property Account[]|OneHasMany|null $accounts {1:m Account::$owner}
```

8.3.2 Repositář účtu

Repositář účtu představuje klíčovou komponentu v architektuře aplikace, zajišťující komunikaci mezi databází a entitami reprezentující data. Díky využití PHPDoc anotací pro definici metod a vlastností nabízí repositář účtu přehlednou a standardizovanou formu pro práci s datovým modelem. Všechny repositáře jsou odvozeny od základní třídy *Repository*, která dále dědí funkcionalitu z *Nextras ORM*. Tato hierarchie umožňuje repositářům přistupovat k metodám, jako jsou *getModel()* pro získání modelu a *getMapper()* pro získání mapperu, což jsou základní stavební bloky pro manipulaci s daty.

Konkrétně *AccountRepository* rozšiřuje tyto možnosti o specifické metody a vlastnosti pro práci s entitami účtů, jako je *$mapper* s anotací *@property-read*, která explicitně označuje, že *mapper* je určen pouze pro čtení, a přetíženou metodu *getMapper()* vracející *AccountMapper*.

```php
/**
 * @property-read AccountMapper $mapper
 * @method  AccountMapper getMapper()
 */
19 usages  Leon Holub *
class AccountRepository extends Repository
{
    no usages  Leon Holub
    public static function getEntityClassNames(): array
    {
        return [Account::class];
    }

    no usages  new *
    public function onAfterInsert(Account|IEntity $account): void
    {
        $account->startBalance = $account->balance;
    }

    2 usages  Leon Holub
    public function findMyAccounts(int $userId): ICollection
    {
        return $this->findBy(['owner' => $userId])->orderBy(['position' => 'ASC']);
    }
}
```

Obrázek 23. Implementace repositáře účtu

Funkce *getEntityClassNames()*, která je nezbytnou metodou každého repositáře, vrací pole entit, které repositář spravuje. V případě *AccountRepository* je tato metoda implementována tak, aby vracela pole entit účtů.

Zajímavou funkcionalitou je možnost repositáře přepisovat standardní metody nextras ORM, například *onAfterInsert*. Tato metoda zajišťuje, že po vytvoření nového účtu v databázi je jeho počáteční bilance nastavena na hodnotu, kterou uživatel specifikoval. Tím se

eliminuje potřeba manuálně nastavovat počáteční a aktuální bilanci v aplikaci, což zvyšuje přehlednost a efektivitu kódu.

Repositář nabízí prostor pro definici specifických funkcí, které se mohou zabývat transformací dat získaných z databáze. Takové funkce umožňují data nejen získat, ale i upravit do formy, která nejlépe vyhovuje potřebám aplikace. Příkladem je metoda *findMyAccounts*, kterou jsem implementoval v repositáři účtů. Tato metoda přijímá jako argument identifikátor uživatele a slouží k vyhledání a selekci účtů, které patří právě tomuto uživateli. Funkce tedy efektivně filtruje data podle specifického kritéria a vrací výsledek v podobě kolekce účtů, což je příkladný způsob, jak v repositáři pracovat s daty na míru konkrétním požadavkům.

8.3.3 Mapper účtu

Mapper účtu představuje nástroj pro správu dat mezi databází a aplikací. Každý mapper dědí od základní třídy Mapper, která je součástí frameworku Nextras ORM.

V rámci maperu byla implementována přetížená funkce *createConventions*, která umožňuje mapování embedovatelných vlastností do databáze. Embedovatelná vlastnost představuje možnost vložit do entity objekt, který má své vlastní atributy. Příkladem může být vlastnost *Repeat* v entitě šablony transakce, která se opakuje v definovaném intervalu. Tato vlastnost je deklarována pomocí *@property Repeat|null $repeat {embeddable}*, a samotná třída *Repeat* obsahuje atributy jako *prompt*, *lastTransactionDate* a *nextTransactionDate*.

Konvence v maperu znamenají, že data se uloží do sloupce repeat*_prompt* v tabulce *transaction_templates*, kde *repeat* je název embedovatelné třídy a následující název po podtržítku odpovídá jménu vlastnosti v této třídě.

Každý mapper má možnost definovat název tabulky, do které bude data mapovat, což není povinný údaj a pokud není specifikován, použije se název dle jména mapperu. Mappery pro specifické entity mohou navíc definovat vlastní metody pro selekci dat, které přesahují běžné funkce jako getBy() nebo findBy(). Například, ve zmíněném případě účtu byla

implementována funkce *findAllAccountsWithUser*, která přijímá ID uživatele a vrací kolekci účtů spojených s tímto uživatelem, ať už je tam uživatel majitel, nebo je s ním pouze sdíleno.

```
class AccountMapper extends Mapper
{
    protected $tableName = 'accounts';

    5 usages  Leon Holub *
    public function findAllAccountsWithUser(int $userId): ICollection
    {
        $builder = $this->builder();
        $builder->joinLeft( toExpression: "account_user", onExpression: "account_user.account_id = accounts.id");
        $builder->andWhere( expression: "account_user.user_id = %i", $userId);
        $builder->andWhere( expression: "account_user.state = %i", ...args: AccountUser::STATE_ACCEPTED);
        $builder->orWhere( expression: "accounts.owner = %i", $userId);
        $builder->andWhere( expression: "accounts.state = %i", ...args: Account::STATE_ACTIVE);
        $builder->addGroupBy( expression: "accounts.id");
        $builder->orderBy( expression: "CASE WHEN accounts.owner = %i THEN 0 ELSE 1 END", $userId);
        $builder->addOrderBy( expression: "accounts.position");
        $builder->addOrderBy( expression: "account_user.state");
        return $this->toCollection($builder);
    }
```

Obrázek 24. Část implementace mapperu účtu

V každé takové funkci se nejprve musí definovat builder, který staví databázový dotaz. Výsledná metoda musí zahrnovat návratovou hodnotu, která převádí výsledek buď do kolekce nebo do konkrétní entity.

Práce s daty prostřednictvím mapperu a repositáře se liší v tom, že mapper přistupuje přímo k názvům sloupců v databázi. Naopak, repositář využívá názvy definovaných entit, což reflektuje strukturu objektů v aplikaci.

8.4 MVP model

V rámci projektu byla architektura webové aplikace organizována do osmi primárních presenterů a dvou hlavních layoutů, v souladu s modelem MVP (Model-View-Presenter). Veškerá funkcionalita presenterů je odvozena od centrálního presenteru pojmenovaného *BasePresenter*, který slouží jako základ a integruje repositáře potřebné pro většinu ostatních presenterů, jako je například *transactionRepository* pro vytváření transakcí dostupných z různých částí aplikace. *BasePresenter* také zahrnuje *FormFactory* pro vytváření standardních formulářů, nastavení překladače, načítání transakčních dat a správu uživatelských notifikací. Zabezpečuje také autentizační kontrolu přístupu uživatele.

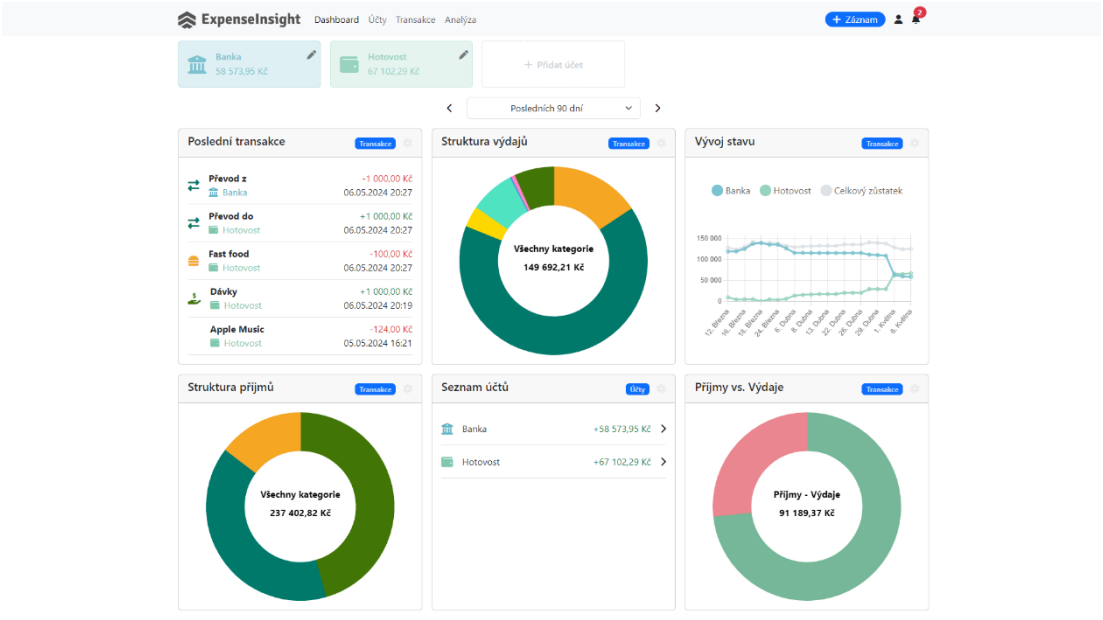
Dále BasePresenter poskytuje funkce pro odhlášení uživatele (*actionLogOut*), změnu jazyka aplikace (*handleChangeLocale*) a další specifické funkce, jako je zjišťování kurzů měn a jejich přepočet.

Ostatní presentery potom rozšiřují funkcionalitu BasePresenteru a přidávají specifické funkce podle svého účelu a jsou kategorizovány do čtyř hlavních skupin:

1. Pro nepřihlášené uživatele: K této kategorii náleží *SecurityPresenter*, využívající minimalistický design a specifické sady skriptů a stylů prostřednictvím layoutu *@securityLayout.latte*.
2. Pro přihlášené uživatele: Do této skupiny patří presentery *Accounts*, *Home*, *Settings* a *Transactions*, přičemž každý se zaměřuje na specifickou funkcionalitu svých akcí.
3. Automatizace: *CronPresenter* spadá do této kategorie a je určen pro automatizované úkoly, jako je vytváření transakcí z trvalých příkazů a generování souvisejících notifikací. Tento presenter je na serveru spouštěn denně.
4. Error stránky: Standardní Nette presentery pro zpracování chyb, Error4xx a Error, jsou používány pro zachycení chyb na produkčním serveru, jako jsou chyby 404 nebo nečekané serverové problémy.

8.4.1 Domovská stránka

Domovská stránka aplikace, ovládaná *HomePresenterem*, je designována s jediným účelem – poskytnout uživatelům komplexní přehled o jejich finančním stavu. *HomePresenter* funguje na principu jednoho defaultního pohledu (akce), avšak jeho funkcionalita je rozšířena o WidgetRepository, který má za úkol správu widgetů. Tento repositář umožňuje dynamické zobrazení widgetů na dashboardu na základě uživatelské konfigurace a preferencí, včetně nastavení viditelnosti, parametrů a pořadí.



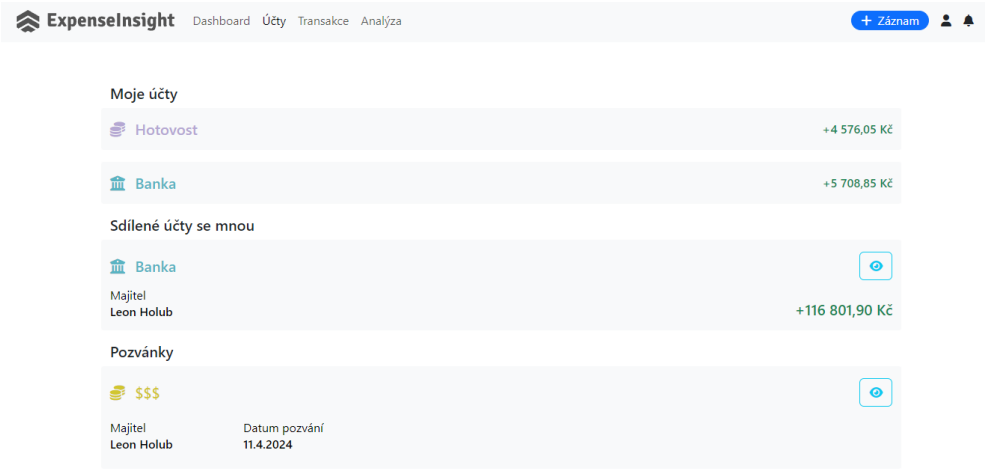
Obrázek 25. Domovská stránka

8.4.2 Účty

Správu účtů v aplikaci zajišťuje komponenta AccountsPresenter, která obsahuje dvě akce: default a detail.

8.4.2.1 Akce default

Primární akce default slouží k zobrazení seznamu všech účtů asociovaných s přihlášeným uživatelem. Zde jsou prezentovány účty, které uživatel vlastní, účty sdílené s ním ostatními uživateli a pozvánky k připojení se k dalším účtům. U sdílených účtů je uživateli umožněn náhled na jméno vlastníka a aktuální zůstatek, a také je možné otevřít detail daného účtu. U pozvánek na sdílené účty má uživatel informaci pouze o majiteli a má možnost přijmout nebo odmítnout přístup prostřednictvím modálního okna.



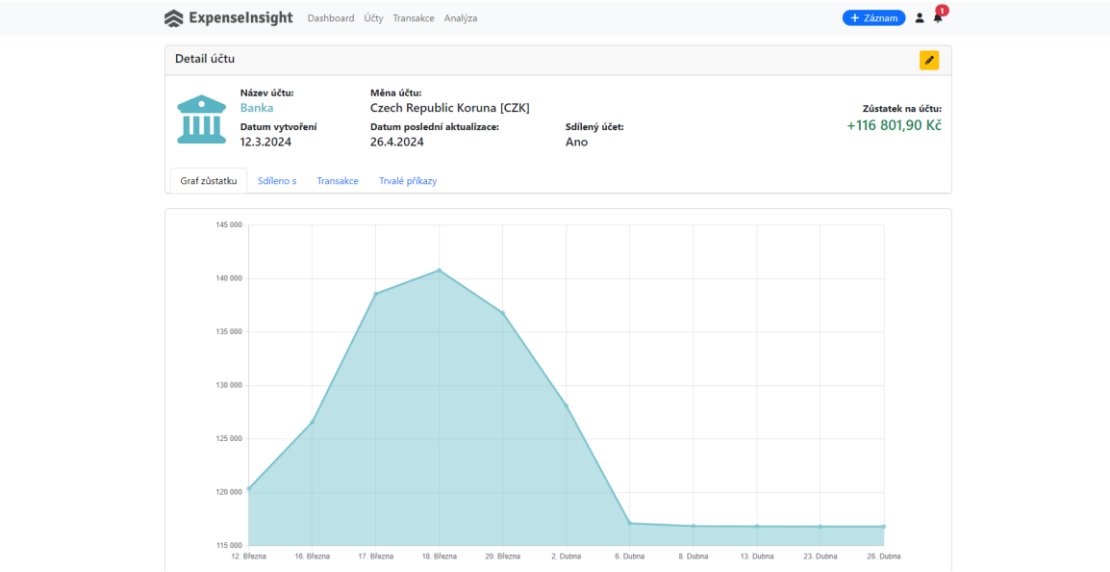
Obrázek 26. Stránka s účty

8.4.2.2 Akce detail

Akce detail poskytuje uživatelům rozšířené informace o konkrétním účtu, s možnostmi dynamického zobrazení různých typů dat spojených s tímto účtem. Struktura a obsah této akce jsou navrženy (viz Obrázek 14) a detailně popsány v odstavci 7.1.2 „Návrh detailu účtu“. Uživatelé mohou prostřednictvím interaktivních tlačítek v horní části stránky měnit obsah zobrazovaný v spodní části. Tato tlačítka umožňují přepínání mezi zobrazením grafu zůstatku, seznamem transakcí, informacemi o sdílených účtech, a trvalými příkazy a předplatnými spojenými s účtem.

V sekci transakcí je zobrazen seznam posledních 30 transakcí, s možností načíst další záznamy, pokud jsou dostupné. Trvalé příkazy a předplatná jsou prezentována tak, že vlastníci účtů vidí kompletní seznam trvalých příkazů a předplatných pro všechny uživatele

asociované s daným účtem, zatímco sdílení uživatelé mají přístup pouze k těm, které se jich přímo týkají.



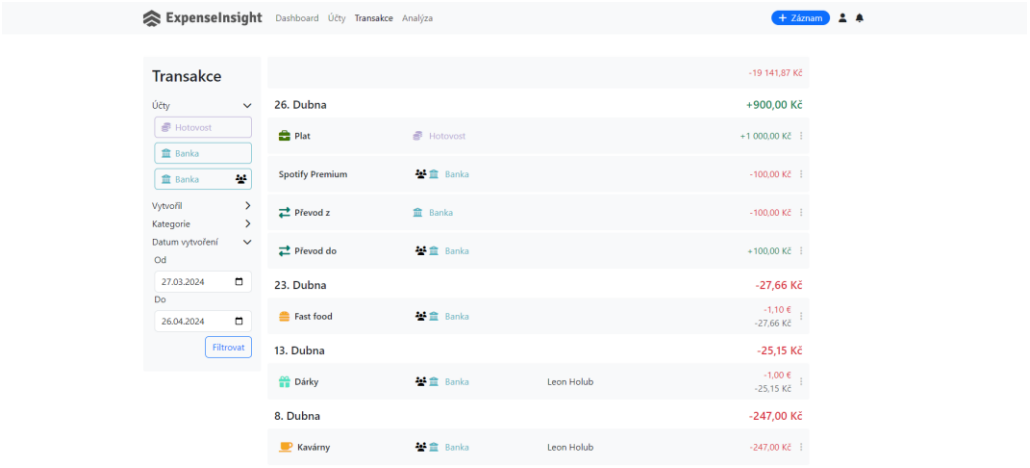
Obrázek 27. Stránka detailu účtu

8.4.3 Transakce

Správu a zobrazení transakcí v aplikaci zajišťuje *TransactionsPresenter*, který disponuje dvěma hlavními akcemi: *default* a *categories*. Obě tyto akce využívají komponentu formuláře k filtraci transakcí, která poskytuje uživatelům možnost výběru na základě účtu, autora transakce, a časového rozmezí. Komponenta ve svém konstruktoru přijímá parametr *$withCategories*, jenž určuje, zda umožnit uživatelům filtrovat i kategorie transakcí.

8.4.3.1 Akce default

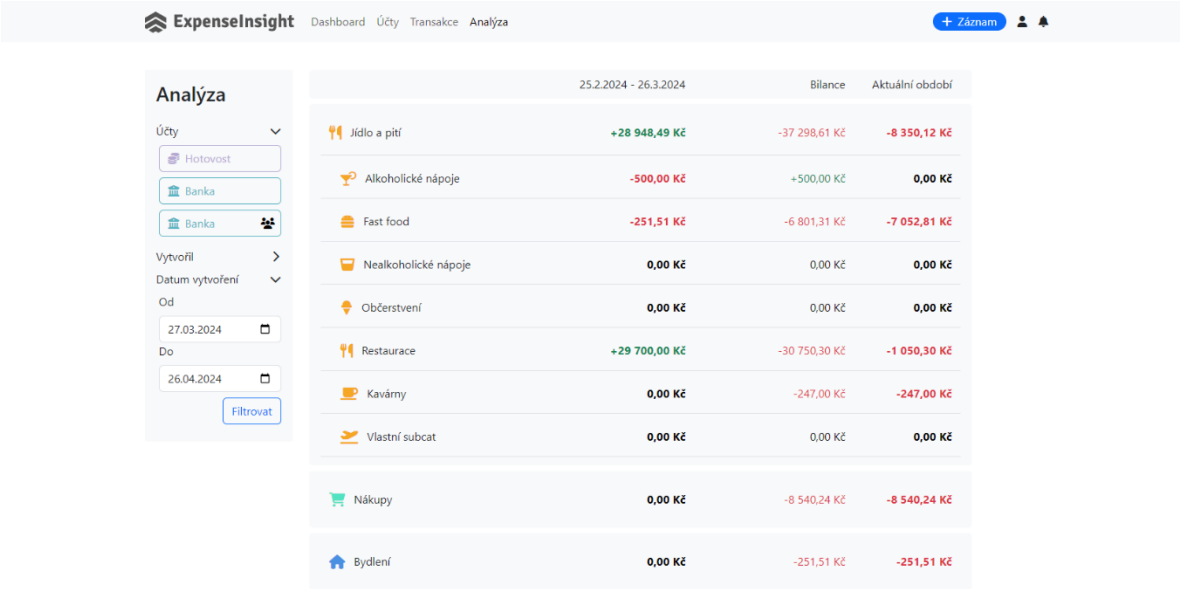
V rámci akce default je uživatelům prezentován seznam transakcí za vybrané období. Tyto transakce jsou rozděleny do kategorií podle dnů. Pro každý den je vypočtena a zobrazena celková suma, která je barevně odlišena podle toho, zda je kladná či záporná. Jednotlivé transakce jsou vypisovány podle specifikace dané návrhem (viz Obrázek 16) a částka každé transakce je obarvena dle toho, zda se jedná o příjem nebo výdaj. Pokud se jedná o účet, který uživatel nevlastní je vedle ikony účtu zobrazena navíc ikona skupiny lidí, reprezentující, že se jedná o cizí účet.



Obrázek 28. Stránka transakcí

8.4.3.2 Akce categories

Tato akce, i když v systému pojmenovaná jako *categories*, je pro uživatele prezentována jako "Analýza", v adresáři stránky je znázorněna jako *analysis*. Akce využívá stejný filtr jako akce *default*, ale bez možnosti vybírat kategorie, protože jejím účelem je poskytnout uživatelům analýzu výdajů podle kategorií. Uživatelé si zvolí období a systém poskytne srovnání s předchozím odpovídajícím obdobím. Zobrazuje přehled výdajů podle kategorií a srovnávací rozdíly mezi obdobími.



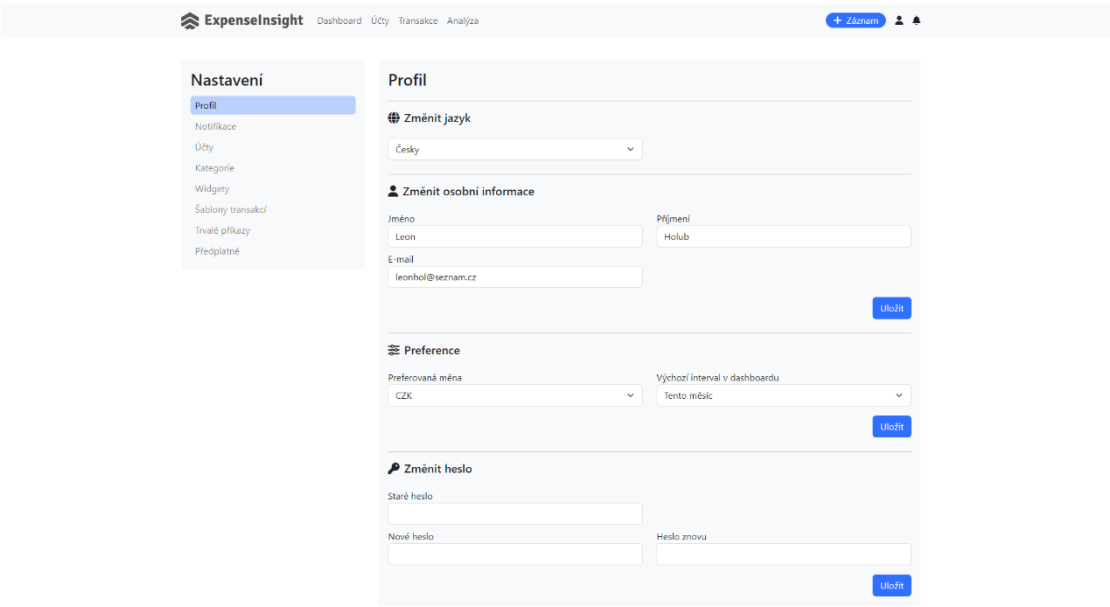
Obrázek 29. Stránka analýzy

8.4.4 Nastavení

Správu nastavení v aplikaci zajišťuje *SettingsPresenter*. Ten obsluhuje osm různých akcí, z nichž každá využívá komponentu *SettingsNavBar* pro zobrazení bočního navigačního panelu. Nastavení v aplikaci je rozděleno do tří hlavních typů stránek, každý s vlastním specifickým stylem a funkcionalitou.

8.4.4.1 Úprava profilu

Design stránky pro úpravu profilu byl upraven v souladu s návrhem (viz Obrázek 17). Navíc byla přidána dodatečná sekce umožňující změnu jazyka aplikace a sekce osobních informací byla rozdělena do dvou. Pro zjednodušení procesu byly implementovány tři tlačítka pro ukládání. Ta vizuálně oddělují uložení různých částí nastavení. Nicméně všechny tlačítka ve skutečnosti odesílají celý formulář, a na serverové straně v PHP se rozhoduje, zda se má provést i kontrola změny hesla. Toto řešení bylo implementováno s cílem logicky oddělit jednotlivé sekce nastavení a poskytnout uživatelům jasnější rozdělení mezi změnou hesla, úpravou osobních údajů a úpravou preferencí zobrazování. Díky tomu má uživatel dojem, že může změnit heslo nezávisle na ostatních informacích a obráceně, což zvyšuje přehlednost a uživatelský komfort při interakci s nastavením profilu.

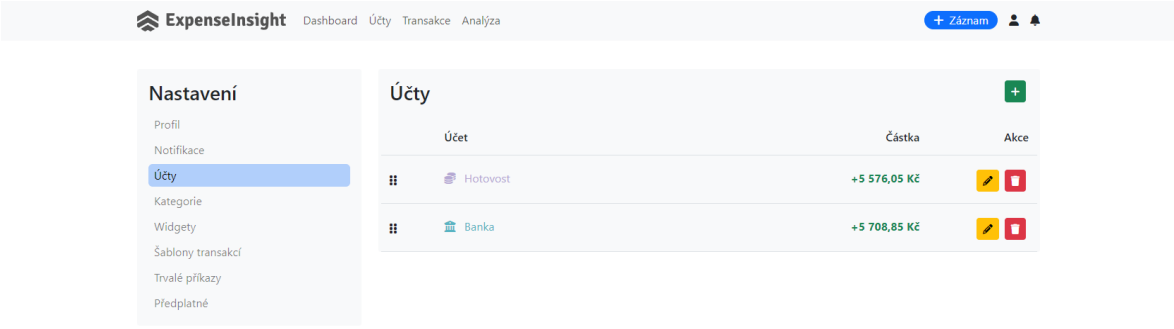


Obrázek 30. Stránka nastavení profilu

8.4.4.2 Seznamové výpisy

Stránky pro seznamové výpisy následují navržený styl (viz Obrázek 18). Zahrnují seznamy s možnostmi editace, smazání a případné změny pořadí záznamů, což je umožněno ikonou

se šesti tečkami. Pro šablony transakcí je k dispozici tlačítko "play" pro okamžité provedení transakce. Tento typ seznamového výpisu obsahuje nastavení pro notifikace, účty, widgety, šablony transakcí, trvalé příkazy a předplatné.

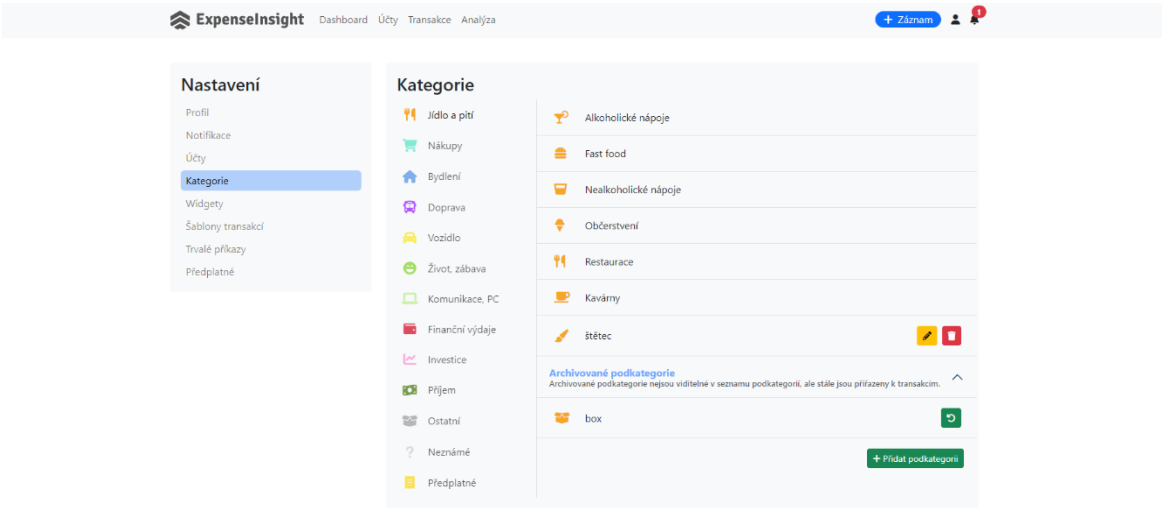


Obrázek 31. Stránka nastavení s účty

8.4.4.3 Nastavení kategorie

Stránka nastavení kategorie je navržena podle vizuálního návrhu (viz Obrázek 19). Na levé straně stránky je uveden výpis názvů kategorií, které slouží jako interaktivní přepínače. Kliknutím na jakoukoliv kategorii dojde k aktualizaci seznamu podkategorií na pravé straně, které odpovídají vybrané kategorii.

Tento proces je zefektivněn použitím dynamických snippetů. Každý panel s podkategoriemi má přidělený jedinečný identifikátor odpovídající kategorii, kterou reprezentuje. Když uživatel provede změnu (přidání, odebrání, úpravu) v podkategoriích, data jsou uložena a přes AJAX je regenerován pouze konkrétní panel s daným identifikátorem. Toto řešení výrazně zrychluje interakci s daty a minimalizuje zatížení serveru, jelikož není nutné překreslovat všechny kategorie a podkategorie při každé změně.



Obrázek 32. Stránka nastavení s kategoriemi

8.5 Komponenty

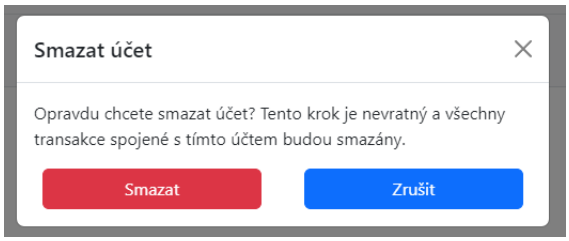
Adresář *Components* je strukturovaný do čtyř podadresářů, které obsahují různé typy komponentů používaných v aplikaci. Podadresáře zahrnují *Controllers*, *Forms*, *Widgets* a *Mails*.

8.5.1 Controllers

Adresář *controllers* obsahuje komponenty, které jsou zodpovědné za prezentaci dat a jejich manipulaci, včetně navigace mezi stránkami nebo ajaxových volání. Každý controller je implementován jako třída dědící od *Nette/.../Control* a obsahuje metodu render, která zajistí vykreslení příslušné šablony Latte. Mezi controllery patří:

8.5.1.1 DeleteModal

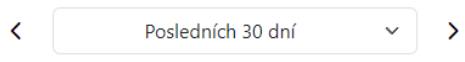
Jedná se o komponentu pro vytváření modálního okna s výzvou k potvrzení nebo zrušení akce. Používá třídu *DeleteWarning*. Ta umožňuje definovat nadpis, popis akce, texty tlačítek pro potvrzení a zrušení, odkazy a určit, zda se má použít AJAX. Komponenta se využívá napříč celou aplikací a neslouží jen jako potvrzení mazání, ale i například jako volení odpovědi na pozvání ke sdílení účtu.



Obrázek 33. Modální okno pro smazání účtu

8.5.1.2 SelectDashboardDate

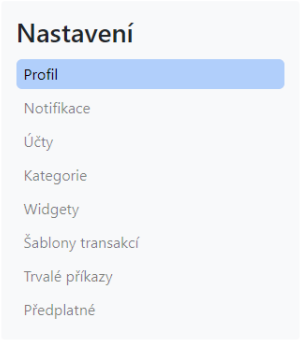
Komponenta umístěná na dashboardu umožňuje uživatelům měnit časové období, ze kterého chtějí data widgetů získávat. Umožňuje posun datumů a zobrazuje aktuálně zvolené období s možností výběru z různých intervalů.



Obrázek 34. Komponenta pro volbu intervalu

8.5.1.3 SettingsNavBar

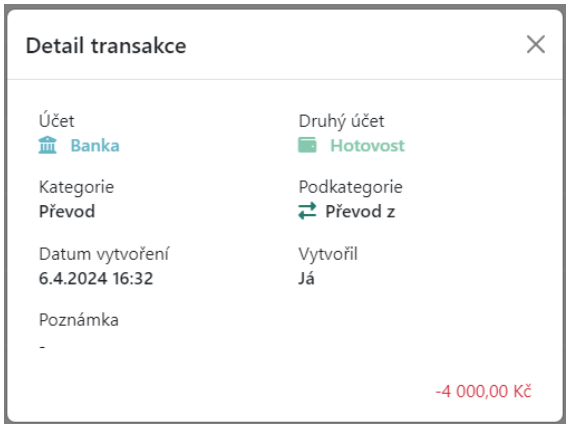
Tato komponenta se stará o vykreslení boční navigace v sekci nastavení. Její hlavní funkce je zobrazit navigační prvky a indikovat, na jaké stránce se uživatel nachází. Tato komponenta se vykresluje v každé akci nastavení, čímž snižuje jako ostatní komponenty nutnost vysoké repetice kódu.



Obrázek 35. Boční navigace v nastavení

8.5.1.4 TransactionDetail

Detailní zobrazení transakce v modálním okně, včetně informací jako částka, účet, datum a autor. Controller předává tyto informace do šablony, která následně prezentuje všechny detaily transakce.



Obrázek 36. Detail transakce

8.5.2 Forms

V sekci *Forms* nalezneme definice všech formulářových komponent aplikace. Podobně jako controllery, každý formulář je reprezentován třídou a přidruženým souborem Latte. Třídy formulářů dědí od *Nette\Forms\Control* a obsahují metodu *render* pro vykreslení

uživatelského rozhraní formuláře. Oproti controllerům však formuláře nabízí navíc metodu *createComponent{Nazev}Form*, která slouží k nastavení formulářových prvků, pravidel validace a dalších specifických atributů. Pro zajištění základní funkcionality formuláře je využívána FormFactory.

8.5.2.1 FormFactory

Tato třída je zodpovědná za poskytování společných funkcí a nastavení používaných napříč různými formuláři. Konstruktor *FormFactory* přijímá překladač, který se používá pro lokalizaci textů včetně chybových hlášení. Metoda *create* přijímá instanci presenteru a zakládá nový *Nette\Application\UI\Form*, jenž má přednastavený překladač, a definuje výchozí chování při chybě formuláře.

V případě chybného vyplnění formuláře *defaultErrorFunction* nastaví chybové hlášky jako *flash* zprávy typu danger a aktivuje *redrawFlashes* pro jejich zobrazení, pokud byla vyvolána AJAXem.

Metoda *sendSuccessPayload* je určena pro situace, kdy je formulář správně vyplněný a data uložena. Tato metoda informuje klienta o úspěšném procesu a umožňuje tak uzavření modálního okna pomocí JavaScriptu.

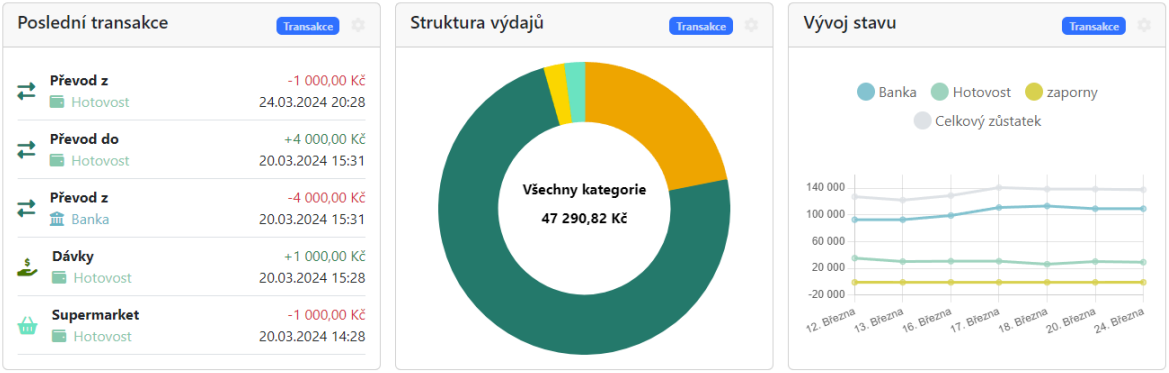
8.5.3 Widgets

Adresář *Widgets* je v aplikaci strukturován do dvou podkategorií: *Charts* a *Texts*.

Podadresář *Charts* obsahuje widgety zaměřené na grafické zobrazování dat ve formě grafů. To umožňuje uživatelům získat vizuální prezentaci jejich finančního vývoje nebo rozdělení výdajů a příjmů. Pro vykreslování grafů se využívá javascriptová knihovna *chart.js*, upravena o vlastní plugin pro zobrazení legendy jako html tagy a o plugin pro výpis názvu a hodnoty uprostřed koláčového grafu.

Podadresář *Texts* zahrnuje textové widgety, které poskytují informace jako souhrn posledních transakcí nebo detaily o účtech.

Widgety jsou koncipovány podobně jako controllery a každý z nich využívá metodu *render* pro vykreslení obsahu. Tato metoda přijímá objekt *Widget* obsahující nastavení widgetu. Renderování pak reflektuje individuální nastavení uživatele pro každý typ widgetu, což umožňuje jednoduchou personalizaci zobrazených dat na dashboardu.



Obrázek 37. Ukázka widgetů na hlavní stránce

8.5.4 Mails

Adresář *mails* je klíčový pro funkci odesílání emailů v aplikaci a obsahuje třídu *Mailer* spolu s příslušnými Latte šablonami pro tvorbu obsahu emailů.

8.5.4.1 Třída Mailer

Mailer je třída odpovědná za sestavování a odesílání emailů. V konstruktoru třídy jsou definovány vstupní parametry – překladač pro lokalizaci obsahu emailu, emailová adresa příjemce, název šablony, která se má použít, titulek emailu a pole parametrů a odkazů potřebných pro generování obsahu emailu. Tato třída zajišťuje, že všechny potřebné informace jsou správně předány do šablony a že výsledný obsah emailu je korektně sestaven a odeslán.

8.5.4.2 Latte šablony

Pro renderování emailů se používají šablony Latte. Tyto šablony transformují vstupní data a parametry do finální formy emailu, který je připraven k odeslání. Všechny atributy a odkazy, které jsou součástí parametrů předaných do šablony, jsou v šabloně zpracovávány jako samostatné proměnné. Například, pokud je v poli parametrů hodnota *"title" => "Titulek"*, tak se v šabloně tento titulek vypisuje jako *{$title}*.

8.5.4.3 Opakující se části emailů

Každá konkrétní šablona emailu importuje dva další soubory: *mailHeader* a *mailFooter*. Tyto soubory obsahují části emailu, které se v každém odeslaném emailu opakují. Tímto způsobem je zajištěna konzistence emailové komunikace.

8.6 Helpers

V rámci frameworku Nette je možné definovat vlastní pomocné filtry (helpers) pro úpravu a formátování výstupů dat v šablonách Latte. V Latte jsou to základní funkce jako noescape pro výpis textu bez escapování HTML znaků, round pro zaokrouhlení číselných hodnot, upper pro konverzi textu na velká písmena a mnoho dalších.

Kromě standardních filtrů Latte, byla v mém projektu implementována řada specifických pomocných funkcí, které rozšiřují funkčnost šablonovacího systému.

8.6.1 Vlastní filtr

Nejpoužívanější filtr napříč aplikací je filtr *money*, který je navržen pro formátování číselných hodnot jako finančních částek. Přijímá číslo a symbol měny jako parametry, přičemž může také akceptovat parametry pro přidání znaménka plus nebo mínus a pro nastavení barvy textu. Funkce nejprve určí, zda je částka kladná či záporná, na základě čehož nastaví příslušnou barvu textu. Poté číslo zaokrouhlí na požadovaný počet desetinných míst a vrátí ho ve formátu HTML span s nastavenou barvou a hodnotou, doplněnou o měnový symbol.

```
public
static function money(float $amount, string $currencySymbol, string $sign =
"+", string|null $color = null): string
{
    if ($amount < 0) {
        $sign = "";
    }

    if ($color === null) {
        $color = $sign === '+' ? 'success' : 'danger';
        if ($amount == 0) {
            $color = 'body-emphasis';
            $sign = "";
        }
    }
    $formattedAmount = number_format($amount, 2, ',', ' ');

    return "<span class='text-$color'>$sign$formattedAmount
$currencySymbol</span>";
}
```

Obrázek 38. Ukázka filtru *money*

9 DOSAŽENÝ VÝSLEDEK

Webová aplikace ve své finální verzi je připravena pro každodenní používání. Navržena je takovým způsobem, aby ji bylo možné v budoucnu rozšířit o nové funkce – další widgety, rozpočtové nástroje nebo podpora dalších jazyků.

9.1 Porovnání s Wallet

Oproti aplikaci Wallet od BudgetBakers je můj projekt ExpenseInsight čistě webový. Přesto, pokud porovnáme obě aplikace ve verzi zdarma, ExpenseInsight nabízí rozsáhlejší funkcionalitu a tím zvyšuje uživatelský komfort.

Tabulka 1. Porovnání ExpenseInsight a Wallet

Kategorie	ExpenseInsight	Wallet
Platforma	Čistě webová	Webová i mobilní
Účty	Neomezený počet Sdílení s uživateli Více informací o účtu	Omezený počet (3) Sdílení pouze v placené verzi Méně informací o účtu
Dashboard	Menší množství widgetů Editace widgetů Volby z definovaných období	Větší množství widgetů Nelze editovat widgety Volby jakéhokoliv rozmezí datumů
Tvorba transakcí	Nelze nastavit tag Jakákoliv měna Automatické transakce Uložit šablonu při tvorbě transakce nebo v nastavení	Lze nastavit tag Jakákoliv měna Nepodporuje automatické transakce Uložit šablonu pouze v nastavení
Analýza financí	Rozdíl období sumou Pouze číselná	Rozdíl období v procentech Lze zobrazit grafy

9.1.1 Porovnání základních funkcionalit

ExpenseInsight se vyznačuje tím, že podporuje sdílení účtů s neomezeným počtem dalších uživatelů. To rozšiřuje možnosti spolupráce. Navíc přidává systém notifikací, který uživatele informuje o změnách provedených hosty na jejich účtu. V aplikaci je implementováno 12 druhů notifikací, každá s možností překladu do jazyka aktuálně přihlášeného uživatele.

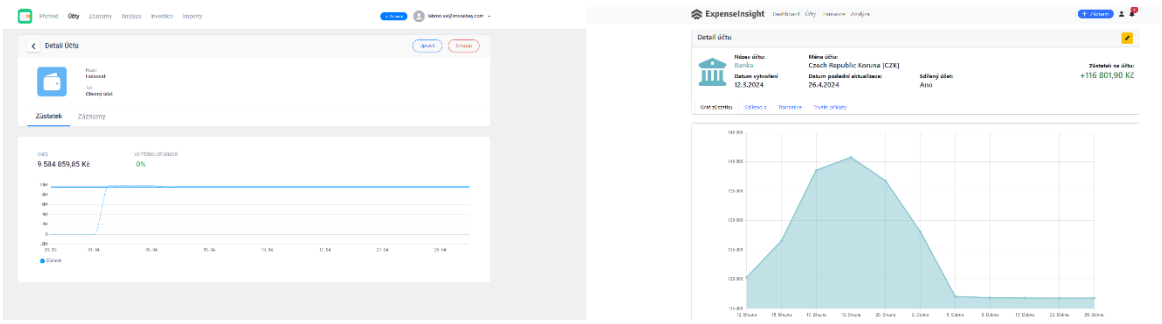
V aplikaci Wallet, lze nastavit preferovanou měnu pouze při registraci. ExpenseInsight umožňuje uživatelům měnu kdykoli změnit a přepočítat hodnoty. To je užitečná vlastnost při změně bydliště do jiné země. Další flexibilitu přináší možnost úpravy osobních údajů, což Wallet nenabízí.

9.1.2 Porovnání účtů

Zatímco Wallet omezuje počet účtů ve své bezplatné verzi na tři, ExpenseInsight takové omezení nemá. Uživatelé mohou účty přizpůsobit podle vlastních preferencí, včetně volby názvu, barvy a ikony z předdefinovaných možností. Funkce jako synchronizace s bankou a import dat z Excelu, které Wallet nabízí pouze v placené verzi, nejsou v ExpenseInsight zahrnuty.

V ExpenseInsight je spodní panel robustnější, doplněný o další panely, které reflektují systémové funkce opakujících se transakcí a sdílení účtů. V této aplikaci mohou majitelé účtů vidět, s kým sdílí účet a spravovat opakující se transakce, zatímco ostatní uživatelé mají přístup pouze k informacím o svých vlastních opakujících se transakcích.

V porovnání (viz Obrázek 39) lze vidět, že ExpenseInsight nabízí komplexnější přehled informací o účtu v porovnání s aplikací Wallet.

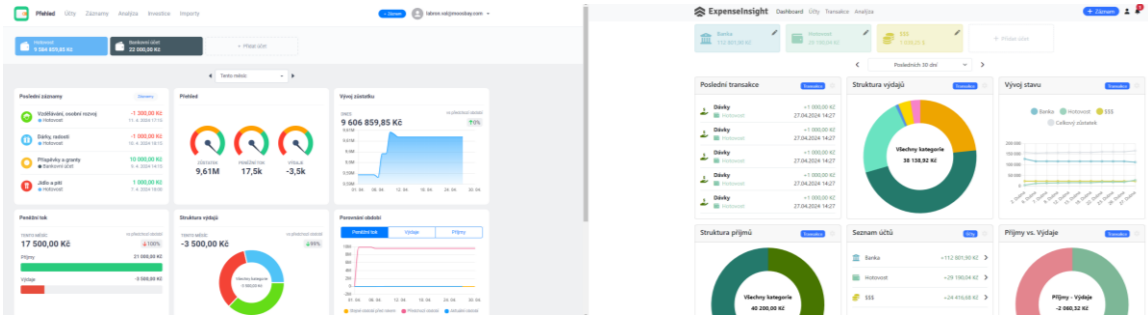


Obrázek 39. Porovnání detailu účtu mezi Wallet a ExpenseInsight

9.1.3 Porovnání Dashboardu

Design dashboardu je na první pohled obdobný. ExpenseInsight však přináší celou řadu vylepšení ve funkčnosti, jako je například možnost vrátit se k zobrazení všech účtů jednoduchým kliknutím na již vybraný účet. Ačkoli nabízí méně widgetů, jsou ty dostupné, více přizpůsobitelné a poskytují klíčové informace. Uživatelé mohou ovlivnit zobrazení grafů, včetně legendy, os x a y, u textových widgetů si mohou zvolit, zda chtějí částky ve výchozí měně nebo přepočítané do preferované měny.

Obě aplikace umožňují změnu časového období pro zobrazení dat, ale zatímco Wallet nabízí i výběr konkrétních dat, ExpenseInsight poskytuje přednastavené možnosti, které jsou pro většinu uživatelů dostatečné.



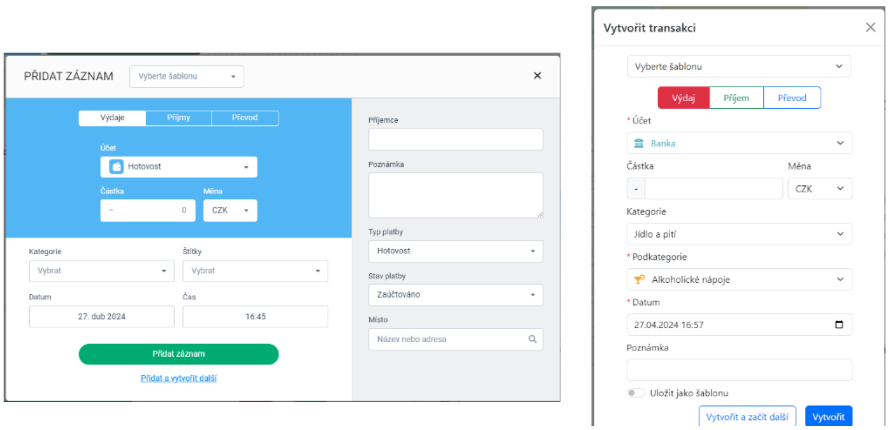
Obrázek 40. Porovnání dashboardu mezi Wallet a ExpenseInsight

9.1.4 Porovnávání transakcí

V obou aplikacích, Wallet i ExpenseInsight, je uživatelům umožněno snadno přistupovat k vytváření transakcí z jakékoliv části aplikace díky dobře viditelnému tlačítku umístěnému v uživatelském rozhraní. Wallet posouvá uživatelský zážitek o krok dále tím, že dovoluje přiřazovat transakcím tagy, čímž nabízí další úroveň organizace a personalizace.

Na rozdíl od Wallet, ExpenseInsight se nezaměřuje na tagy. Obě platformy sdílejí podobný způsob odeslání formulářů pro transakce. Uživatelé mohou vybírat mezi tlačítky pro rychlé dokončení transakce s okamžitým zavřením formuláře nebo tlačítkem, které umožňuje ponechat formulář otevřený pro zadání další transakce.

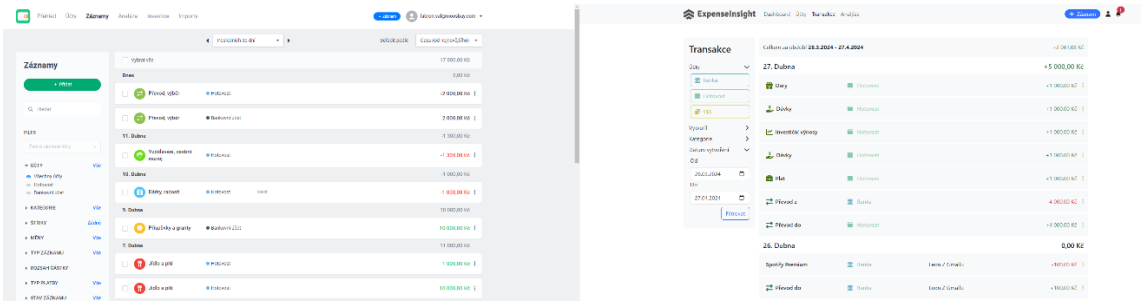
Design tvorby transakce ve Wallet je možná více esteticky přitažlivý, což může přilákat uživatele hledající vizuálně příjemné rozhraní. Nicméně, obě aplikace poskytují robustní základnu pro efektivní správu finančních transakcí s důrazem na jednoduchost a přímou manipulaci s finančními daty.



Obrázek 41. Porovnání tvorby transakce mezi Wallet a ExpenseInsight

9.1.4.1 Záznamy Transakcí

V oblasti záznamů transakcí jsou Wallet a ExpenseInsight konzistentní v tom, že obě aplikace poskytují uživatelům přehled o celkové sumě transakcí za jednotlivé dny a umožňují následné detailní zobrazení transakcí pod touto sumou. Možnost úpravy nebo odstranění transakcí je standardní funkcionalitou v obou systémech. Klíčový rozdíl spočívá v možnosti ExpenseInsight filtrvat transakce podle autora, což je přínosné v kontextu sdílených účtů.

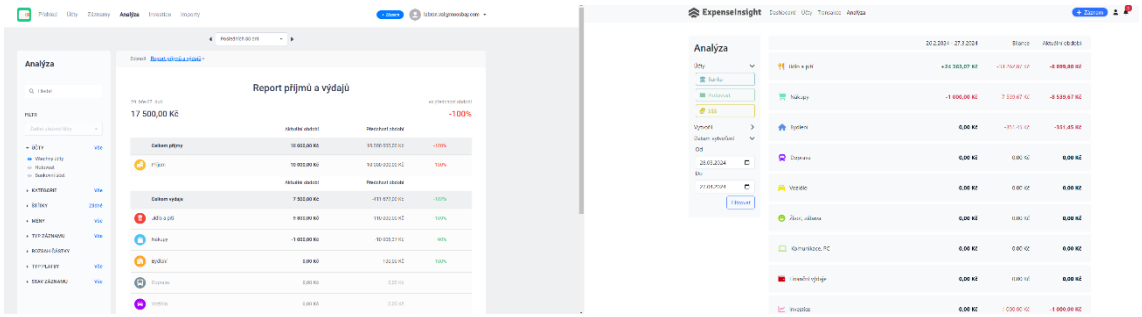


Obrázek 42. Porovnání seznamu transakcí mezi Wallet a ExpenseInsight

9.1.4.2 Analýza Financí

V rámci analýzy poskytují obě aplikace srovnatelné filtry. ExpenseInsight však zjednodušuje zobrazení tím, že se zaměřuje výhradně na sumu utracenou v každé kategorii, na rozdíl od Wallet, která nabízí také vizualizaci pomocí grafů. Wallet rovněž poskytuje srovnání s předchozím obdobím s procentuálním vyhodnocením nárůstu nebo poklesu výdajů. Systém reportů ve Wallet vykazuje chyby ve výpočtu procent, zejména při srovnávání záporných a kladných období.

Pro ExpenseInsight byl zvolen přístup založený na srovnání absolutních hodnot mezi obdobími, což znamená, že uživatelé vidí přímou finanční bilanci mezi aktuálním a předchozím obdobím bez použití procentuálních změn. Tento přístup byl zvolen na základě přesnosti a praktičnosti. Poskytuje uživatelům okamžité pochopení finančního rozdílu a je intuitivnější pro plánování a správu rozpočtu.



Obrázek 43. Porovnání analýzy mezi Wallet a ExpenseInsight

9.1.4.3 Opakované transakce

Aplikace ExpenseInsight umožňuje uživatelům vytvářet opakující se transakce a předplatné, které jsou dostupné pro jakýkoliv účet, bez ohledu na to, zda je uživatel jeho majitelem. Majitel účtu má možnost tyto transakce vidět v přehledu a popřípadě je smazat. Uživatelé mohou nastavit opakované transakce s výběrem z předdefinovaných období opakování a specifikovat data začátku a konce. Tyto transakce jsou automaticky provedeny ve stanovených termínech, přičemž systém generuje notifikace, které uživatele informují o každém provedení transakce.

9.2 Uživatelské testování aplikace

K ověření funkčnosti a uživatelské přívětivosti aplikace ExpenseInsight bylo uskutečněno testování s omezenou skupinou uživatelů. Toto testování bylo zorganizováno ve formě specifických úkolů, které účastníci měli splnit, aby se ověřila intuitivnost rozhraní a funkčnost aplikace. Uživatelé zaznamenávali své zkušenosti, případné obtíže při splnění úkolů a poskytovali zpětnou vazbu na odhalené chyby.

Mezi hlavní připomínky patřilo omezení počtu základních podkategorií. Uživatelé sice ocenili existující možnost přidávání vlastních podkategorií, avšak vyjádřili potřebu rozšíření standardně nabízených základních podkategorií. Tato zpětná vazba byla přijata a aplikace byla upravena tak, aby zahrnovala širší nabídku základních podkategorií, což zvýšilo její flexibilitu a uživatelskou adaptabilitu.

Během testování nebyly odhaleny žádné zásadní technické problémy. Menší nedostatky, které byly během testování identifikovány, byly rychle adresovány a opraveny, což vedlo k další stabilizaci aplikace. Toto uživatelské testování bylo klíčové pro dokončení finálních úprav a zvýšení celkové spokojenosti uživatelů s ExpenseInsight, čímž byla potvrzena vysoká uživatelská hodnota a funkčnost aplikace.

ZÁVĚR

V rámci mé bakalářské práce byla vyvinuta webová aplikace ExpenseInsight, zaměřená na efektivní správu osobních financí. Aplikace nabízí rozšířené funkce, jako jsou opakované transakce, sdílení účtů a tvorba šablon, které jsou často dostupné jen v placených verzích konkurenčních řešení. Cílem bylo překonat omezení existujících aplikací a nabídnout uživatelům vyšší míru přizpůsobení a funkcionalitu bez nutnosti platby. Webová aplikace je dostupná na https://expenseinsight.leonholub.cz.

V teoretické části byla popsána terminologie webových aplikací a PHP frameworků, se speciálním zaměřením na Nette framework včetně šablonovacího systému Latte. Byly představeny technologie jako Nextras ORM a JavaScriptová knihovna Naja, které aplikace využívá.

Praktická část práce zahrnovala průzkum existujících řešení, s aplikací Wallet jako hlavním srovnávacím modelem. Byly navrženy designy v Adobe XD, vytvořen návrh databáze, a nakonec bylo řešení implementováno. Projekt byl strukturován s využitím Nextras ORM a MVP modelu Nette, byly popsány klíčové komponenty a práce s formuláři a e-maily.

Výsledná aplikace ExpenseInsight nabízí uživatelům prémiové funkce bez poplatků, zvyšuje flexibilitu při úpravách dashboardu a poskytuje jasnější informace o finančních rozdílech prostřednictvím konkrétních částek namísto procent. Díky možnosti sdílení účtů aplikace zároveň zajišťuje majitelům účtů komplexní přehled o všech transakcích a trvalých příkazech. Dále umožňuje vytváření automatických transakcí s možností nastavení začátku, konce a frekvence opakování. Aplikace podporuje více jazyků a měn, a to včetně automatického přepočtu dle aktuálního kurzu.

Nicméně, aplikace stále má oblasti pro zlepšení, jako je import transakcí z bankovních účtů a lepší správa uživatelských preferencí v analýze a filtrování transakcí. Rozšíření systému oznámení o možnost zasílání e-mailů by zvýšilo její uživatelskou přívětivost. Budoucí rozvoj by mohl zahrnovat vytvoření API pro snadnější integraci s mobilními zařízeními, což by posílilo uživatelský zážitek a rozšířilo dostupnost aplikace.

SEZNAM POUŽITÉ LITERATURY

[1] KOĎOUSKOVÁ, Barbora. Web, webová stránka a webová aplikace, v čem je rozdíl? Online. Rascasone. 2021. Dostupné z: https://www.rascasone.com/cs/blog/web-webova-aplikace-rozdil/. [cit. 2024-03-25].

[2] VOLLE, Adam. Web application. Online. Encyclopedia Britannica. 2022. Dostupné z: https://www.britannica.com/topic/Web-application. [cit. 2024-03-25].

[3] BARBOŘÍK, Tom. Jaký je rozdíl mezi webem a webovou aplikací? Online. 2023. Dostupné z: https://www.coreapp.cz/blog/jaky-je-rozdil-mezi-webem-a-webovou-aplikaci. [cit. 2024-03-25].

[4] KOĎOUSKOVÁ, Barbora. Typy webových aplikací, kterou zvolit pro vlastní vývoj? Online. Rascasone. 2021. Dostupné z: https://www.rascasone.com/cs/blog/typy-webovych-aplikaci/. [cit. 2024-03-25].

[5] CODECADEMY TEAM. What is a Web App? Online. Codecademy. Dostupné z: https://www.codecademy.com/article/what-is-a-web-app. [cit. 2024-03-25].

[6] CODECADEMY TEAM. What Is a Framework? Online. Codecademy Blog. 2021. Dostupné z: https://www.codecademy.com/resources/blog/what-is-a-framework/. [cit. 2024-03-26].

[7] Top 10 PHP frameworks. Online. Www.javatpoint.com. 2019. Dostupné z: https://www.javatpoint.com/top-10-php-frameworks. [cit. 2024-03-27].

[8] Usage statistics of server-side programming languages for websites. Online. In: W3Techs. 2024. Dostupné z: https://w3techs.com/technologies/overview/programming_language. [cit. 2024-03-26].

[9] NONIS, Susith. 10 Best PHP Frameworks You Should Know About. Online. MonoVM.com. 2023. Dostupné z: https://monovm.com/blog/10-best-php-frameworks/. [cit. 2024-03-26].

[10] Blade Templates. Online. The PHP Framework For Web Artisans. 2024. Dostupné z: https://laravel.com/docs/11.x/blade. [cit. 2024-03-26].

[11] KONEČNÝ, Martin. Lekce 4 - Obrana proti útoku XSS v PHP. Online. Obrana proti útoku XSS v PHP. Dostupné z: https://www.itnetwork.cz/tutorial-bezpecnost-v-php--utoku-xss-a-obrana. [cit. 2024-03-26].

[12] The flexible, fast, and secure PHP template engine. Online. Dostupné z: https://twig.symfony.com/. [cit. 2024-03-27].

[13] GRUDL, David. Nette – Pohodlný a bezpečný vývoj webových aplikací v PHP. Online. Nette Framework. Dostupné z: https://nette.org/cs/. [cit. 2024-03-29].

[14] GRUDL, David. Historie Nette. Online. Nette Framework. Dostupné z: https://nette.org/cs/history. [cit. 2024-03-29].

[15] GRUDL, David. Tým a přispěvatelé. Online. Nette Framework. 2024. Dostupné z: https://nette.org/cs/contributors. [cit. 2024-03-29].

[16] GRUDL, David. David Grudl. Online. David Grudl. 2024. Dostupné z: https://davidgrudl.com/cs. [cit. 2024-03-29].

[17] GRUDL, David. Údržba a kompatibilita s PHP. Online. Nette Framework. 2024. Dostupné z: https://nette.org/cs/maintenance. [cit. 2024-03-29].

[18] GRUDL, David. Jak fungují aplikace? Online. Nette Dokumentace. Aktualizováno 19.3.2024. Dostupné z: https://doc.nette.org/cs/application/how-it-works. [cit. 2024-03-30].

[19] GRUDL, David. Bootstrap. Online. Nette Dokumentace. 2017, aktualizováno 19.3.2024. Dostupné z: https://doc.nette.org/cs/application/bootstrap. [cit. 2024-03-31].

[20] GRUDL, David. Začínáme s Tracy. Online. Tracy Ladicí Nástroj. 2018, aktualizováno 21.1.2024. Dostupné z: https://tracy.nette.org/cs/guide. [cit. 2024-03-31].

[21] GRUDL, David. Presentery. Online. Nette Dokumentace. 2011, aktualizováno 21.1.2024. Dostupné z: https://doc.nette.org/cs/application/presenters. [cit. 2024-03-31].

[22] GRUDL, David. Životní cyklus presenteru. Online. In: Nette. 2011. Dostupné z: https://files.nette.org/git/application/4.0/lifecycle.svg. [cit. 2024-03-31].

[23] GRUDL, David. Šablony. Online. Nette Dokumentace. 2023. Dostupné z: https://doc.nette.org/cs/application/templates. [cit. 2024-04-01].

[24] GRUDL, David. Latte je synonymum bezpečnosti. Online. Latte šablony. 2023. Dostupné z: https://latte.nette.org/cs/safety-first. [cit. 2024-04-01].

[25] GRUDL, David. Latte tagy (makra). Online. Latte šablony. 2015, aktualizováno 21.1.2024. Dostupné z: https://latte.nette.org/cs/tags. [cit. 2024-04-01].

[26] GRUDL, David. Routování. Online. Nette Dokumentace. 2011, aktualizováno 21.1.2024. Dostupné z: https://doc.nette.org/cs/application/routing. [cit. 2024-04-01].

[27] GRUDL, David. Interaktivní komponenty. Online. Nette Dokumentace. 2011, aktualizováno 21.1.2024. Dostupné z: https://doc.nette.org/cs/application/components. [cit. 2024-04-01].

[28] GRUDL, David. AJAX. Online. Nette Dokumentace. 2011, aktualizováno 21.1.2024. Dostupné z: https://doc.nette.org/cs/application/ajax. [cit. 2024-04-01].

[29] PUDIL, Jiří. Install and set up Naja. Online. Naja. 2021, aktualizováno 24.2.2024. Dostupné z: https://naja.js.org/#/guide/01-install-setup-naja. [cit. 2024-04-01].

[30] PUDIL, Jiří. History. Online. Naja. 2017, aktualizováno 29.9.2021. Dostupné z: https://naja.js.org/#/history. [cit. 2024-04-01].

[31] TVRDÍK, Jan; ŠKRÁŠEK, Jan a MATĚJKA, David. Nextras components. Online. Nextras. Dostupné z: https://nextras.org/. [cit. 2024-04-01].

[32] TVRDÍK, Jan; ŠKRÁŠEK, Jan a MATĚJKA, David. Configuring Nette DI Extension. Online. Dbal (main). 2023. Dostupné z: https://nextras.org/dbal/docs/main/config-nette. [cit. 2024-04-01].

[33] ŠKRÁŠEK, Jan. Nextras Orm. Online. Orm (main). 2014, aktualizováno 14.3.2024. Dostupné z: https://nextras.org/orm/docs/main/. [cit. 2024-04-01].

[34] ŠKRÁŠEK, Jan. Architecture. Online. Orm (main). 2020, aktualizováno 23.5.2021. Dostupné z: https://nextras.org/orm/docs/main/architecture. [cit. 2024-04-01].

[35] Team - Wallet by BudgetBakers - Your New Personal Finance Manager. Online. Wallet by BudgetBakers - Your New Personal Finance Manager. 2022. Dostupné z: https://budgetbakers.com/team/. [cit. 2024-04-21].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
DBAL	Database Abstraction Layer
DI	Dependency injection
DIC	Dependence injection container
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LTS	Long-term Support
MS SQL	Microsoft SQL Server
MVC	Model view controller
NEON	Formát konfiguračních souborů v Nette
npm	Node Package Manager
ORM	Object Relational Mapping
PHP	Hypertext Preprocessor
PhpDoc	PHP Documentation
SEO	Search Engine Optimization
SQL	Structured Query Language

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD S BAKALÁŘSKOU PRACÍ

Struktura obsahu přiloženého CD:

- Adresář Text bakalářské práce – obsahuje text bakalářské práce ve formátu PDF.
- Adresář Webová aplikace – obsahuje zdrojové kódy webové aplikace včetně návodu na spuštění