GYMNASIUM JANA KEPLERA

Parléřova 2/118, 169 00 Praha 6

Webová kalkulačka pro výpočet podílu uživatele na státním rozpočtu ČR

Maturitní práce

Autor: Tomáš Hozda

Třída: R8.A

Školní rok: 2020/2021

Předmět: Informatika

Vedoucí práce: Bc. Emil Miler

Praha, 2023

GYMNASIUM JANA KEPLERA
Kabinet informatiky

ZADÁNÍ MATURITNÍ PRÁCE

Student:	**Tomáš Hozda**
Třída:	**R8. A**
Školní rok:	**2022/2023**
Vedoucí práce:	**Emil Miler**
Název práce:	**Webová kalkulačka pro výpočet podílu uživatele na státním rozpočtu ČR**

Pokyny pro vypracování:

Cílem práce je vytvořit webovou aplikaci, která bude fungovat jako kalkulačka podílu jednotlivce na státním rozpočtu ČR. Uživatel zadá své příjmy a výdaje (různé kategorie podle zdanění) a kalkulačka vypočítá, kolik peněz za daný rok odvedl do státní rozpočtu a jak s nimi stát naložil (za co je utratil). Aplikace bude spravovat databázi vybraných státních výdajů a zobrazovat v přepočtu, jak se uživatel na nich podílel. Databázi výdajů, stejně jako státní rozpočty a průměrné hodnoty pro dané roky, bude možné upravovat a přidávat skrze aplikaci. Aplikace se dokáže vypořádat i s faktem, když stát utratí více, než jsou jeho příjmy. Výsledkem tak bude, že uživatel z vypočítané statistiky získá přehled o tom, jaké množství peněz odvede státu, kolik je z nich reálného užitku a k jakému účelu.

Doporučená literatura:

[1] TAUBERER, Joshua. *Open Government Data*. **2. vydání, 2014. Dostupné z: https://opengovdata.io/**

[2] ESPOSITO, Dino. *Modern Web Development*. **Microsoft Press, 2016. ISBN: 9781509300013**

URL repozitáře:

https://github.com/Tomdrs/statni_rozpocet

student *vedoucí práce*

V Praze dne 29. 9. 2022

Prohlášení

Prohlašuji, že jsem svou práci vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů. Nemám žádné námitky proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Praze dne 24. března 2023 Tomáš Hozda

Poděkování

Rád bych poděkoval vedoucímu práce, rodině, především bratrovi Lukášovi, a přátelům za konzultaci práce a podporu při tvoření projektu.

Abstrakt

Cílem práce je vytvořit webovou aplikaci, která funguje jako kalkulačka podílu jednotlivce na státním rozpočtu ČR. Uživatel zadá své příjmy a výdaje (různé kategorie podle zdanění) a kalkulačka vypočítá, kolik peněz za daný rok odvedl do státní rozpočtu a jak s nimi stát naložil (za co je utratil). Aplikace spravuje databázi vybraných státních výdajů a zobrazí v přepočtu, jak se uživatel na nich podílel. Databázi výdajů stejně jako státní rozpočty a průměrné hodnoty pro dané roky je možné upravovat a přidávat skrze aplikaci. Aplikace se dokáže vypořádat i s faktem, když stát utratí více, než jsou jeho příjmy. Výsledkem tak je, že uživatel z vypočítané statistiky získá obrázek o tom, jaké množství peněz odvede státu, kolik je z nich reálného užitku a k jakému účelu.

Klíčová slova

daně, web, aplikace, kalkulačka, Flask, AlpineJS, PicoCSS, Nix

Abstract

The goal of the project is to create a web application, that will serve as a calculator of an individual's participation in the state budget of Czech Republic. The user will enter their income and expenses (different categories based on taxation), and the calculator will compute, how much of his money went into the state budget, and how the state used them (that is, what it spent the money on). The application manages a database of selected state investments, and displays, how the user participated on them. The database of expenses, just like budgets and average state levies is possible to modify, add, and delete through the web interface. The app has no issue with the possibility, that the state may spent more money, than is its income. The result is that the user gets a picture, of how much money is levied from them into the state budget, and how much of it has an actual use (and which uses those are).

Keywords

taxes, web, application, calculator, Flask, AlpineJS, PicoCSS, Nix

Obsah

[1] Pallets. *Flask*. URL: `https://flask.palletsprojects.com/en/2.2.x/`.
[2] Caleb Porzio. *Alpine.js*. URL: `https://alpinejs.dev/`.
[3] Gani. *PocketBase*. URL: `https://pocketbase.io/`.

1. Teoretická část

V době, kdy jsem vymýšlel, co bych mohl vytvořit jako svůj maturitní projekt, se v médiích probírala možnost nákupu nových stíhacích letounů pro českou armádu. Odhadem se uváděla celková částka 48 miliard Kč.[1] Slyšel jsem mnoho názorů o tom, jak moc peněz to je a že by bylo lepší peníze investovat jinam. Zamyslel jsem se tedy nad tím, kde stát peníze získává a za co je pak utrácí. V souvislosti s tím mě napadlo, kolik vlastně člověk sám odvede na daních do státního rozpočtu a jak by se tedy podílel na investicích a výdajích.

Rozhodl jsem se tedy, že jako svůj maturitní projekt vytvořím webovou kalkulačku, která by se zabývala problematikou státního rozpočtu. Vytvořil jsem si představu, že uživatel do kolonek formuláře na webu zadá své příjmy a výdaje, kalkulačka údaje zpracuje a vypočítá z nich, jakou částku uživatel odvedl celkově za rok, který si vybere v nabídce, do státního rozpočtu. Uživatel by na stránce měl mít možnost prohlížet si údaje o státním rozpočtu z nabídky zpracovaných let. Dále jsem chtěl, aby kalkulačka zobrazila, prostřednitcvím jakých daní uživatel peníze státu odvedl. Tyto údaje by měly být společně v tabulce s celkovými státními příjmy. Vedle zobrazení státních výdajů jsem chtěl pak vypsat údaje o tom, jak se uživatelem odvedená částka rozpočítala na výdaje. Dále jsem na webu chtěl uvést některé konkrétní investice státu, kolik za ně stát utratil a jak se na nich podílel uživatel. Pro lepší celkovou funkcionalitu kalkulačky jsem se rozhodl, že kalkulačka musí mít i správu databáze údajů, které zobrazuje.

Z teoretické části jsem se tedy musel zaměřit především na daňový systém a údaje o státním rozpočtu. K tomu jsem také musel dohledat konkrétní investice státu a všechny tyto části spojit funkčně do sebe. Pro vytvoření funkční kalkulačky jsem tedy potřeboval získat údaje o státním rozpočtu za vybrané roky, údaje o konkrétních investicích, od uživatele přijmout údaje o jeho příjmech a výdajích a podle pravidel daňového systému vypočítat jeho podíl, který by pak kalkulačka rozložila mezi jednotlivé výdaje a investice.

1.1 Státní rozpočet

Svoji teoretickou analýzu jsem začal hledáním údajů o státním rozpočtu. Stát musí být v těchto záležitostech transparentní, a tak na webu Ministerstva financí jsou k dispozici tabulky plnění státního rozpočtu, které uvádí jednotlivé důležité položky příjmů a výdajů státního rozpočtu.[2] U každé položky je uvedeno, jaká hodnota se pro daný rok očekávala a jaká finálně hodnota skutečně byla. Já jsem samozřejmě pracoval se skutečnými hodnotami, aby kalkulačka byla přesnější. Rozhodl jsem se pro práci s údaji z let 2020, 2021 a 2022, protože v době dokončení projektu mají již vyplněné skutečné údaje. Počítal jsem s tím, že v rámci správy databáze bude pak možné přidat údaje i pro jiné roky.

Tyto údaje jsem si potřeboval pouze přepsat do databáze. Zpočátku jsem se potýkal s tím, že v souboru s údaji bylo více tabulek a uváděly různé příjmy. Na internetu jsem však poté dohledal, že

[1] Lubomír Světnička. *Vyjednávání o stíhačkách F-35 pokračují. Česko sonduje možnosti financování.* URL: https://www.idnes.cz/zpravy/nato/stihaci-lockheed-f-35-armada-obran-cupakova-usa-pilot.A230214_194540_zpr_nato_inc.

[2] Ministerstvo financí České republiky. *Plnění státního rozpočtu.* URL: https://www.mfcr.cz/cs/verejny-sektor/statni-rozpocet/plneni-statniho-rozpoctu.

existuje rozpočtové určení daní.[3] Částka, kterou daňový poplatník odvede na daních, nemíří pouze do státního rozpočtu, ale dělí se. Část putuje do rozpočtu obcí, část do krajských rozpočtů, a teprve zbytek skončí ve státním rozpočtu. Některé daně se dokonce odvádí pouze do obecních a krajských rozpočtů. Veřejné zdravotní pojištění se také nepočítá do příjmů státního rozpočtu.[4] Pochopil jsem tedy, že do databáze si potřebuji vypsat údaje z tabulky, která zohledňuje opravdu pouze příjmy státního rozpočtu a ne celkové daňové příjmy.

1.2 Konkrétní státní investice

Vedle údajů o státním rozpočtu jsem také potřeboval získat příklady konkrétních investic a jejich náklady. Myslím si, že pro uživatele jsou snadno představitelnější a pochopitelnější konkrétní investice (např. nákup stíhacích letounů nebo dostavba Pražského okruhu) než obecné pojmy státních výdajů (např. neinvestiční transfery rozpočtům územní úrovně nebo ostatní běžné výdaje). Velmi dobře mi v tomto ohledu pomohl národní investiční plán na roky 2020 až 2050.[5] Plán jsem prošel a vybral jsem si z něj důležité a velké investice, které jsem zapsal do databáze. Samozřejmě je možné přidávat další investice z plánu prostřednictvím správy databáze aplikace.

Při vybírání investic z plánu jsem si ale uvědomil problém složitosti výpočtu podílu jednotlivce na těchto investicích. Investice se většinou realizují mnoho let a není jasné, jakým přesným způsobem je stát financuje. Moje aplikace pracuje vždy s údaji za jeden rok, a i když z celkových výdajů státního rozpočtu vím, kolik putovalo na investice, tak nevím, kolik na jaké. Rozhodl jsem se tedy, že pro výpočet budu vycházet z předpokladu, že investice se celá realizovala a zaplatila pouze v roce, pro který si uživatel počítá své odvody a zobrazuje údaje o státním rozpočtu. U konkrétní investice by se tak zobrazil procentuální podíl na celkové částce, která za daný rok mířila na investice, a podle toho částka, jakou by se uživatel podílel. Nejedná se tak o reálnou částku, ale i tak poskytuje představu o nákladnosti investice a velikosti podílu uživatele aplikace.

1.3 Výpočet celkové odvedené částky na daních

S již získanými obecnými informacemi o státních rozpočtech jsem se mohl zaměřit přímo na jednotlivce. Částku, kterou odvede do státního rozpočtu, musím určit z údajů, které mi poskytne. Potřeboval jsem si tedy rozmyslet, které údaje pro výpočet potřebuji od uživatele požadovat. Zaměřil jsem se tedy na státní příjmy a podíval se, které daně přináší peníze do rozpočtu. Z těch jsem si vybral ty, na kterých se podílí jednotlivec sám - DPH, daň z příjmu fyzických osob, pojistné na sociální zabezpečení, daň z hazardu a spotřební daň z paliva, tabáku a alkoholu. Podle toho jsem si určil, jaké údaje od uživatele budu požadovat, abych konkrétní daň vypočítal. Pokud uživatel některé údaje neposkytne, bude kalkulačka pracovat s údaji odpovídajícími průměrnému občanovi ČR. Zdroje, odkud jsem tyto údaje čerpal, jsou uvedené v README.md.

[3] Finanční správa. *Rozpočtové určení daní*. URL: https://www.financnisprava.cz/cs/dane/danovy-system-cr/rozpoctove-urceni-dani.

[4] Ministerstvo financí České republiky. *Plnění státního rozpočtu*. URL: https://www.mfcr.cz/cs/verejny-sektor/statni-rozpocet/plneni-statniho-rozpoctu.

[5] Úřad vlády České republiky. *Národní investiční plán České republiky 2020-2050*. URL: https://www.vlada.cz/assets/media-centrum/aktualne/Narodni-investicni-plan-CR-2020_2050.pdf.

1.3.1 DPH

DPH se dělí podle procentuálního odvodu do tří kategorií - základní sazba, první snížená sazba a druhá snížená sazba. V druhé snížené sazbě se odvádí daň ve výši 10 %, a to především z vodného a stočného, nákupu knih, hudby, léků a z ubytovacích služeb. V první snížené sazbě to je 15 % a vyměřuje se pro potraviny a gastronomické služby, veřejnou dopravu a léčební pomůcky. Základní sazba je 21 % a vztahuje se na všechny ostatní nákupy a útraty.[6] Od uživatele tedy potřebuji zjistit, kolik průměrné měsíčně utratí za dané kategorie.

1.3.2 Daň z příjmu

Daň z příjmu fyzických osob se dnes vypočítává z hrubého příjmu. Z hrubého příjmu tvoří 15 %. Výpočet probíhá tak, že si spočítáme 15 % z hrubého příjmu a odečteme od toho slevu na poplatníka - 2570 Kč měsíčně. Do roku 2020 se daň vypočítávala ze superhrubé mzdy - 15 % z 1,338násobku hrubé mzdy. Pokud si poplatník nárokuje slevu na dítě, odečte se i ta. Sleva na první dítě je 15 204 Kč za rok, na druhé 22 320 Kč za rok a na každé další 27 840 Kč za rok. Dříve byl maximální součet omezen na 60 300 Kč za rok, od roku 2022 sleva na děti nemá horní hranici. Ze slevy na děti je možné vyměřit daňový bonus. Tedy pokud sleva na děti převyšuje daň z příjmů, stát musí uhradit daňovému plátci rozdíl. V extrémních případech může stát díky slevě na děti zaplatit jednotlivci více, než by jinak jednotlivec celkově odvedl do státního rozpočtu. Na slevu má nárok pouze jeden z rodičů. K dani z příjmů patří ještě daň z kapitálových příjmů. To je také 15 %, ale vyměřuje se pouze, pokud za rok plátce zaznamená zisk z kapitálových příjmů vyšší než 100 000 Kč.[7] Pro výpočet musí uživatel zadat svůj hrubý příjem, počet dětí, pokud na ně uplatňuje slevu, a kapitálové příjmy.

1.3.3 Sociální pojištění

Při výpočtu odvodu na sociální pojištění záleží na tom, jestli je uživatel zaměstnanec nebo OSVČ. Výpočet se podle toho liší. V případě OSVČ jsem se rozhodl částku nepočítat, protože nevím, z jakého vyměřovacího základu si ji uživatel počítá. Po uživateli, který je OSVČ, tedy požaduji, aby vyplnil kolonku s tím, kolik si na pojištění platí. Stejně tak požaduji, aby kolonku vyplnili ti, co pracují jako zaměstnanci, ale na zkrácený úvazek. Opět nevím, z čeho bych daň vypočítal. Pro klasické zaměstnance je to pak jednoduché. Zaměstnanec na sociální pojištění sám odvádí 6,5 % ze své hrubé mzdy a k tomu za něj zaměstnavatel odvádí ještě 24,8 % z hrubé mzdy. Ve své aplikaci zohledňuji i peníze odvedené zaměstnavatelem, protože je také vnímám jako odměnu za práci zaměstnance.[8]

[6] Perlament České republiky. *235/2004 Sb. Zákon o dani z přidané hodnoty*. URL: https://www.podnikatel.cz/zakony/zakon-c-235-2004-sb-o-dani-z-pridane-hodnoty/uplne/#prilohy.

[7] Finanční správa. *Daň z příjmů*. URL: https://www.financnisprava.cz/cs/dane/dane/dan-z-prijmu.

[8] Ministerstvo práce a sociálních věcí. *Sociální pojištění*. URL: https://www.mpsv.cz/socialni-pojisteni.

1.3.4 Spotřební daně a daň z hazardu

Daň z hazardu je 23 %[9], spotřební daň z paliv je v průměru 11,5 Kč/l, spotřební daň z piva je 0,32 Kč/l, spotřební daň z lihovin je 322,5 Kč na litr ethanolu a spotřební daň za jednu cigaretu je 2,9 Kč[10]. Uživatel tedy do kolonek zadá průměrnou částku, kterou prohraje na hazardu, kolik utratí za paliva, kolik vypije piva a lihovin a kolik vykouří cigaret za měsíc.

1.3.5 Finální částka

V instrukcích jsem upřesnil, že uživatelé mají do každé kolonky vyplnit průměrnou měsíční hodnotu údaje. Pomocí výše uvedených procentuálních sazeb kalkulačka spočítá, kolik peněz uživatel odvedl na daních za vybrané položky a samozřejmě to vynásobí dvanácti, aby výsledkem byl údaj za celý rok. U vybraných daní také zohlední rozpočtové určení daní. Z prostředků odvedených prostřednictvím DPH a daně z příjmů fyzických osob míří do státního rozpočtu pouze 64,38 %[11]. U daně z hazardu je to 70 %[12]. Výsledkem výpočtů je celková částka, kterou uživatel za rok odvedl do státního rozpočtu.

1.4 Rozbor uplatnění odvedené částky

Pokud má aplikace údaje o státním rozpočtu, konkrétních investicích a částku zaplacenou uživatelem na daních, může začít zobrazovat informace o uplatnění odvedených peněz. V úvodu teoretické analýzy jsem uvedl, co by kalkulačka měla zobrazovat - kolik prostřednictvím jakých daní uživatel odvedl do rozpočtu, jak se peníze přerozdělily na výdaje a jak by se uživatel hypoteticky podílel na konkrétních investicích.

1.4.1 Analýza kompozice odvedené částky

Uživateli se vedle položek státních příjmů zobrazí, kolika korunami se na této položce podílel. Výpočet odpovídá postupům, které jsem uvedl u výpočtu celkové částky. Pouze zde aplikace zobrazuje částky odvedené na jednotlivých daní a nesčítá je.

1.4.2 Analýza reálného uplatnění odvedené částky

Uživateli se vedle položek státních výdajů zobrazí, kolika korunami se na této položce podílel. Výpočet je velmi jednoduchý. U každé položky aplikace vypočítá, jaký podíl položka tvoří z celkových

[9]Parlament České republiky. *187/2016 Sb. Zákon o dani z hazardních her*. URL: https://www.zakonyprolidi.cz/cs/2016-187.

[10]Wikipedie. *Spotřební daň*. URL: https://cs.wikipedia.org/wiki/Spot%5C%C5%5C%99ebn%5C%C3%5C%AD_da%5C%C5%5C%88#Reference.

[11]Finanční správa. *Rozpočtové určení daní*. URL: https://www.financnisprava.cz/cs/dane/danovy-system-cr/rozpoctove-urceni-dani.

[12]Parlament České republiky. *187/2016 Sb. Zákon o dani z hazardních her*. URL: https://www.zakonyprolidi.cz/cs/2016-187.

výdajů. Toto procento aplikuje pak na částku odvedenou uživatelem na daních. Částka se tedy rozpočítá mezi jednotlivé položky ve stejném poměru, jako je to u celkových státních výdajů.

1.4.3 Hypotetický podíl na konkrétních investicích

Zde kalkulačka vezme celkové náklady konkrétní investice a spočítá, jaký procentuální podíl by náklady tvořily z peněz vyčleněných na investice za daný rok. Korunový podíl uživatele je výsledkem násobení vypočteného procenta a částky, kterou uživatel odvedl na daních a putovala na investice. V sekci 1.2 vysvětluji nepřesnosti tohoto výpočtu.

2. Implementace

2.1 Zvolené technologie

Na implementaci projektu jsem zvolil následující technologie:

- Programovací jazyk Python
- Programovací jazyk JavaScript
- Flask framework
- AlpineJS framework
- PicoCSS styly
- PocketBase

2.1.1 Programovací jazyk Python

Python jsem zvolil z více důvodů. Mé dva hlavní důvody byly praktičnost a technický aspekt. Python je pro mě praktický, protože jsem se s ním dříve setkal při výuce informatiky, a tak jsem se spoléhal, že si jej připomenu a bude pro mě jednodušší jej používat, než abych se učil nový jazyk, se kterým vůbec nemám zkušenosti.

Technický aspekt, proč jsem zvolil Python, je kvůli přesnosti jeho matematiky. Na semináři jsme se učili o standardu IEEE-754 a jeho nepřesnostech, a podle mého vyhledávání na internetu se tato čísla používají jako výchozí čísla s desetinnou čárkou ve většině populárních programovacích jazyků. Python používá čísla s neomezenou přesností, což mi pomáhá dělat přesnější výpočty. Vím, že pro ostatní programovací jazyky existují knihovny, které přidávají další číselné typy, ale to by byla komplexita navíc. Je mi také jasné, že čísla s neomezenou přesností jsou mnohem méně výkonné, než IEEE-754, ale pro účely mého projektu není výkon kritickým aspektem.

2.1.2 Programovací jazyk JavaScript

Jednou z mých obecných motivací při volbě použitých technologií byla jednoduchost. Nechtěl jsem používat nástroje navíc a zbytečně velké technologie. Nejsem sice odborníkem na tuto problematiku, ale NodeJS a další webové runtimy mi připadaly pro můj projekt jako přehnané. S JavaScriptem jsem se také setkal na školní informatice. Z těchto důvodů jsem se rozhodl zvolit JavaScript, i když si jsem vědom některých jeho slabin a nepředvídatelného chování.

S použitím JavaScriptu přímo v prohlížeči, bez žádného mezikroku jako NodeJS, Bun, nebo Deno, ve spolupráci s frameworkem AlpineJS se mi podařilo mít na frontendu pouze jeden jazyk a pouze jednu malou knihovnu jako závislost.

2.1.3 Flask framework[1]

V Pythonu jsem slyšel o dvou knihovnách pro tvorbu webových aplikací - Flask a Django. Django se mi zdálo jako moc složité pro moje účely, a to byl hlavní důvod, proč jsem zvolil Flask. Pro Flask také existuje kurz na Codecademy, který má velký překryv s dalšími kurzy, které jsem splnil v přípravě na maturitní projekt, volba Flasku mi tedy ušetřila čas.

2.1.4 AlpineJS framework[2]

O této knihovně jsem se dozvěděl z youtube videa na kanálu *Fireship*.[3] Líbilo se mi, že s ním můžu vepisovat chování rovnou do HTML, bez toho, aniž bych musel psát velké množství JavaScriptu. AlpineJS se dá do libovolné stránky vložit pomocí <*script*> tagu, díky čemuž jsem se opět vyhnul studiu NodeJS a dalších, složitějších technologií.

Přístup AlpineJS mi připadá vizuálně přehledný, a přestože se jedná o relativně malý framework, neměl jsem pocit, že by mi v něm chyběla nějaká funkcionalita.

Způsob přemýšlení, který AlpineJS vyžaduje, byl pro mě ze začátku nezvyklý. V Alpine se většina akcí a interaktivity definuje pomocí atributů na HTML.

Například:

```
<section
class="user_buttons"
x-data="{ is_logged: false, is_admin: false }"
x-init="
is_logged = window.localStorage.getItem('pocketbase_auth') !== null;
is_admin = window.localStorage.getItem('pocketbase_auth') !== null;
"
>
    <a
     role="button"
     href="/login"
     x-show="!is_logged"
    >
     Login
    </a>
    <a
     role="button"
     href="/admin"
     x-show="is_logged && is_admin"
    >
        Admin
    </a>
```

[1] Pallets. *Flask*. URL: https://flask.palletsprojects.com/en/2.2.x/.
[2] Caleb Porzio. *Alpine.js*. URL: https://alpinejs.dev/.
[3] Fireship. *Fireship*. URL: https://www.youtube.com/@Fireship.

```
    <a
        role="button"
        href="#"
        x-show="is_logged"
        x-on:click.prevent="pb_logout(client); is_logged = false;"
    >
     Logout
    </a>
</section>
```

Základní proces vývoje je takový, že si definuji na HTML prvcích, které mají být interaktivní, atribut **x-data**, ve kterém mám uložené v JavaScriptové objektové syntaxi proměnné patřící k tomuto prvku.

K těmto proměnným se pak mohu odkazovat v jiných atributech na tomto prvku a na prvcích, které jsou potomky tohoto prvku.

Dalším krokem je přidávání atributů jako **x-show** (podmínky, za kterých se má prvek zobrazovat) a **x-if** (podmínky, za kterých má prvek vůbec existovat), a tam, kde potřebuji zobrazit text nějaké proměnné, parametr **x-text**.

Například, zde je textový *input* a výpis textu, který je v něm napsaný:

```
<div x-data="{ vstup: '' }">
   <p x-text="vstup"></p>
   <input type="text" x-model="vstup">
</div>
```

AlpineJS je reaktivní, takže každá změna v inputu se automaticky zpropaguje skrz proměnnou **vstup**.

To mě vede k posledním dvěma atributům, které byly klíčové na straně AlpineJS. Atribut **x-init** diktuje kód, který se spustí, když se prvek poprvé načte, a atribut **x-effect** se spustí, když se stránka načte, a pokaždé, když dojde ke změnám v reaktivních proměnných definovaných v **x-data**, které jsou v kódu tohoto atributu zmíněny.

V některých případech se hodí nemít jednotný **x-effect**, protože tím může docházet k vykonávání zbytečné (nebo i nežádoucí) práce. Pro tyto situace je možné použít *kouzelnou funkci* (dle terminologie AlpineJS) **$watch('jmeno_promenne', hodnota => akce)**, která umožňuje reagovat na individuální změny proměnných.

Tyto atributy používám k volání svého JavaScriptového kódu, který mám rozčleněný do dalších souborů.

2.1.5 PicoCSS

PicoCSS jsem zvolil jako minimalistický CSS framework. Nechtěl jsem všechny styly psát ručně, protože nejsem odborník na design a typografii, ale nechtěl jsem importovat velké a složité frameworky jako Bootstrap, které bych se musel učit. O PicoCSS jsem se také dozvěděl z YouTube

videa[4]. Kromě jeho jednoduchosti a malé velikosti mě zaujalo především to, že nevyžaduje, abych všude psal CSS třídy a učil se, jak se jmenují. PicoCSS definuje styly pro HTML elementy a používá další kontextové informace k identifikaci, jak by měl být daný prvek zobrazen.

Například zobrazení karty, kterou jde skrýt pod její nadpis, vypadá takto:

```
<details>
  <summary>Nadpis</summary>
  <p>obsah</p>
  <p>atd.</p>
  <p>atd.</p>
</details>
```

Tuto funkcionalitu jsem používal na formátování svého layoutu způsobem, který mi připadal velmi praktický. Nevýhodou tohoto způsobu je však to, že zneužívá HTML5 sémantické prvky, které pak nemají nutně úplně logickou strukturu.

Například toto je sekce, kde zobrazuji formulář, příjmy a výdaje státu. Potřeboval jsem vzhled, který je nadefinovaný na tagu **<article>**, i když vím, že ani jeden z těchto prvků není článkem:

```
<section class="main" x-data>
    <article class="calc-form form">
       ...
    </article>
    <article class="prijmy" x-data>
       ...
    </article>
    <article class="vydaje" x-data>
       ...
    </article>
</section>
```

Nemyslím si, že by to toto byl zas tak pádný nedostatek v případě mojí aplikace, protože se nejedná o aplikaci určenou ke čtení nástroji, pro které jsou sémantické tagy důležité. Kdyby můj projekt spravoval například blogy, novinové články nebo odborné příspěvky, bylo by správné použití sématických tagů mnohem důležitější.

Nezvyklým aspektem pro mě bylo rozlišování CSS stylů na základě parametrů jako **role="button"** nebo **aria-busy="true"**.

2.1.6 PocketBase

Ještě než jsem se pustil do implementace projektu, mi bylo jasné, že budu potřebovat databázi. Z hodin informatiky a semináře jsem věděl o databázích jako je SQLite nebo PostgreSQL, ale připadaly mi moc složité. Rozhodl jsem se proto najít nějaké řešení, které by bylo jednodušší a přineslo

[4]Fireship. *Fireship*. URL: https://www.youtube.com/@Fireship.

by mi nejen databázi samotnou, ale i kvalitní podporu jak pro Python, tak pro Javascript, a pomohlo mi se správou uživatelů. Slyšel jsem o projektu Firebase od Googlu, ale ten se mi zdál moc složitý a nelíbila se mi myšlenka, že by můj projekt závisel na Googlu.

Nakonec jsem se, opět na Youtube kanálu Fireship, dozvěděl o projektu PocketBase. PocketBase má webové API, pro které existuje JavaScriptový i Pythonový klient, má přehledné webové rozhraní, zabudovaný koncept uživatelů a jejich správy a umožňuje mi specifikovat pravidla tak, abych měl jistotu, že k databázi mohu přistupovat i z frontendu bez toho, aniž bych se vystavil riziku, že mi některý z uživatelů databázi zlomyslně vymaže.

Pro jazyky JavaScript a Dart generuje PocketBase ukázky, jak provádět různé úkony na příslušné tabulce. Například, u akce pro přidání nové investice, vygeneruje webové rozhraní toto:

```
import PocketBase from 'pocketbase';

const pb = new PocketBase('https://pocketbase.gjk.cat');

...

// example create data
const data = {
    "start_year": 123,
    "name": "test",
    "cost": 123
};

const record = await pb.collection('investments').create(data);
```

Díky těmto ukázkám jsem se v PocketBase rychle zorientoval. Velmi užitečné bylo také nastavení pravidel, které používám u uživatelských investic:

```
# create action - custom rule
user_id = @request.auth.id

# update action - custom rule
user_id = @request.auth.id

# delete action - custom rule
user_id = @request.auth.id
```

Díky těmto pravidlům mám jistotu, že každý normální uživatel může přidávat, upravovat a odebírat pouze svoje investice, a nepotřebuji tuto akci provádět skrze svůj Flaskový backend, který by data a přístupy validoval manuálně.

PocketBase má zabudovaný webserver a jde v něm vyvíjet plnohodnotné aplikace pomocí rozšíření napsaných v jazyce Go. Protože však Flask vyžaduje WSGI server a já neumím Go, nemohl jsem toto využít.

Domnívám se, že pro znalce Go by bylo možné celou aplikaci napsat bez jakéhokoliv separátního backendu, a mít celý projekt jako jeden monolitický celek spravovaný PocketBasem.

Velmi užitečnou funkcí na PocketBase je také *Request log*, díky kterému jsem mohl zkoumat, proč něco, co zkouším, nefunguje. Když jsem se snažil přijít na to, jak správně filtrovat v databázi, vyhledávač požadavků byl další ergonomickou funkcí PocketBase.

PocketBase také obsahuje integraci pro přihlašování skrze služby jako je Google účet, Facebook účet, Gitlab, nebo Twitch, ale do těchto integrací jsem se nepouštěl.

Je nutné podotknout, že PocketBase je primárně interaktivní nástroj, schéma databáze jsem si musel manuálně exportovat do JSON souboru, import musí být také ruční.

2.2 Postup implementace

Pro moji práci jsou klíčová data získaná od státu. Bez správných dat a správné organizace těchto dat bych nemohl provádět výpočty. Začal jsem tedy svůj postup od databáze tvorbou jednotlivých tabulek a naplňováním těchto tabulek daty.

Všechny tabulky prošly několika verzemi, než jsem s nimi byl spokojený (a stejně jsem musel ještě během vývoje zbytku aplikace dělat v tabulkách drobné úpravy). Tabulku na uživatelské investice jsem přidal až na závěr.

Když jsem byl spokojený s daty, ze kterých moje aplikace vychází, vytvořil jsem nový Flaskový projekt. Pro základní nastavení projektu jsem použil nástroj Nix[5], o kterém jsem se dozvěděl díky bratrovi, který mi doporučil a nainstaloval NixOS jakožto *"nerozbitný Linux"*. Je nutné podotknout, že Nix a NixOS není to samé. Termín Nix označuje dvě věci - Nix jazyk a Nix balíčkovač, který je na tom jazyce postavený. NixOS je distribuce postavená na Nixu - balíčkovači. Nix jakožto balíčkovač, build systém a nástroj pro vytváření konzistentních prostředí a konfigurace není nijak závislý na NixOS a funguje na libovolné linuxové distribuci (i macOS v omezené míře), podobně jako Docker.

Rozdíl mezi Dockerem a Nixem je ten, že Docker je imperativní, psaní Dockerfile souborů můžeme přirovnat k psaní instrukcí v receptu do kuchařky, kdežto Nix je deklarativní, specifikuji, co potřebuji, a to mám.

Pro účely mého projektu mi stačilo napsat pouze soubor *shell.nix*, který mi vytvoří prostředí, které má potřebné závislosti, nakonfigurované proměnné prostředí a vstoupí do virtuálního Python prostředí.

Zde je moje Nix konfigurace pro tento projekt:

```
{ pkgs ? import <nixpkgs> { } }:

pkgs.mkShell {
  buildInputs = [
    pkgs.coreutils-full
    pkgs.python3Full
```

[5]kolektiv NixOS. *Nix*. URL: https://nixos.org/.

```
    pkgs.python3.pkgs.virtualenv
    pkgs.python310Packages.waitress
    pkgs.pocketbase
  ];
  shellHook = ''
    virtualenv -p python3 env
    source env/bin/activate
    pip install -r requirements.txt
  '';
  FLASK_APP = "src/main.py";
  POCKETBASE_URL = "https://pocketbase.gjk.cat/";
}
```

Pole *buildInputs* specifikuje, jaké programy potřebuji mít v prostředí, *shellHook*, co se má stát při spuštění tohoto shellu, a ostatní jsou proměnné prostředí.

V této době jsem ještě neměl dokončený online kurz Flasku, tak jsem tento framework teprve zkoušel. První endpoint vracel pouze string *"ahoj"*.

Později jsem přidal knihovnu pro PocketBase a začal psát soubor s matematickými funkcemi pro výpočty jednotlivých statistik (*tax_math.py*). Vytvořil jsem několik endpointů, které prakticky pouze přeposílaly data z PocketBase. Těmto endpointům jsem se teoreticky mohl vyhnout, ale vzhledem k tomu, že jsem data z tabulek potřeboval číst i ve Flaskové aplikaci, jsem je vytvořil a volal je rovnou ve Flasku, místo toho, abych si je posílal přes frontend.

Moje teze je, že tento přístup je rychlejší a efektivnější, když se PocketBase i Flask nachází na stejném stroji. Tyto endpointy používám i na frontendu, když už jsem je měl připravené. Pro vnitřní použití ve Flasku jsem je ale musel rozdělit na dvě funkce, kdy jedna získává data a druhá je flaskovým endpointem, který je posílá jako odpověď.

Například, zde je tento pár pro získání dat pro rozpočet na daný rok:

```
def fetch_rozpocet_na_rok(rok):
    rozpocty = fetch_all_for_table('budgets')

    spravny_rozpocet =
      next(rozpocet for rozpocet in rozpocty if rozpocet.year == rok)
    return Budget(spravny_rozpocet)

@app.route("/rozpocet/<int:rok>")
def rozpocet_na_rok(rok):
    rozpocet = fetch_rozpocet_na_rok(rok)
    return rozpocet.to_json()
```

Kvůli konverzi na JSON jsem si modely opsal do Pythonu a naimplementoval na nich manuální převod následujícím způsobem:

```
from flask import jsonify
```

```
class Budget:
    def __init__(self, record):
        print(record)
        self.year = record.year
        self.total_income = record.total_income
        self.total_expense = record.total_expense
        self.vat = record.vat
        self.consumer_taxes = record.consumer_taxes
        self.income_tax_individual = record.income_tax_individual
        self.income_tax_company = record.income_tax_company
        self.mandatory_social_insurance = record.mandatory_social_insurance
        self.other_tax = record.other_tax
        self.other_nontax_income = record.other_nontax_income
        self.consumer_tax_oils = record.consumer_tax_oils
        self.consumer_tax_tobacco = record.consumer_tax_tobacco
        self.consumer_tax_solar_energy = record.consumer_tax_solar_energy
        self.trash_fees = record.trash_fees
        self.gambling_tax = record.gambling_tax
        self.income_from_eu = record.income_from_eu
        self.income_connected_with_eu = record.income_connected_with_eu
        print(dir(self))         # debugging
        print(self.total_income) #

    def to_json(self):
        return jsonify(
            year=self.year,
            total_income=self.total_income,
            total_expense=self.total_expense,
            vat=self.vat,
            consumer_taxes=self.consumer_taxes,
            income_tax_individual=self.income_tax_individual,
            income_tax_company=self.income_tax_company,
            mandatory_social_insurance=self.mandatory_social_insurance,
            other_tax=self.other_tax,
            other_nontax_income=self.other_nontax_income,
            consumer_tax_oils=self.consumer_tax_oils,
            consumer_tax_tobacco=self.consumer_tax_tobacco,
            consumer_tax_solar_energy=self.consumer_tax_solar_energy,
            trash_fees=self.trash_fees,
            gambling_tax=self.gambling_tax,
            income_from_eu=self.income_from_eu,
            income_connected_with_eu=self.income_connected_with_eu,
        )
```

Uznávám, že tento způsob je trochu repetitivní a že může docházet k chybám kvůli překlepům. Složitější řešení, která jsem na internetu potkal, mi ale připadala nepřehledná pro tuto aplikaci.

Tyto základní endpointy jsem testoval pomocí programu **cURL**, v této chvíli neexistovalo ještě kon-

krétní spojení frontendu a backendu.

Posléze jsem začal realizovat i vzhled frontendu. Nejdřív jsem psal pouze HTML, framework PicoCSS jsem přidal později, a AlpineJS jsem doopravdy začal používat, až když jsem měl hlavní prvky vzhledu svojí aplikace.

Můj první experiment s frameworkem AlpineJS bylo dynamické zobrazování roku v textu footeru stránky:

```
<footer>
    Tomáš Hozda © <span x-text="new Date().getFullYear()"></span>
</footer>
```

Finální vzhled vznikl částečně jako bug. Rozložení příjmů, výdajů a formuláře na údaje uživatele měly být tři rovnocenné sloupce stejné barvy, ale kvůli špatně uzavřeným HTML tagům byl prostřední sloupec šedivý a zasunutý za těmi postraními. Tento vzhled se mi zalíbil, tak jsem se pro něj po opravě této chyby rozhodl.

Nejnovější koncepty v CSS pro mě byly *grid layout* a *flexbox*. Flexbox jsem použil ve dvou případech:

- Pro zobrazování více prvků najednou tak, aby se poskládaly správně na obrazovku (např. dlaždice s investicemi)
- Pro centrování prvků v jejich rodičích

Příklad prvního použití u dlaždic investic:

```
flexperiment {
    display: flex;
    flex-flow: row wrap;
    justify-content: center;
    width: 100vw;
    margin-left: calc((100vw - 100%) / -2);
    margin-right: calc((100vw - 100%) / -2);
}

.flexperiment article {
    flex-grow: 0;
}
```

A centrování prvků:

```
.center-example {
    display: flex;
    align-items: center;
    justify-content: center;
}
```

Intenzivní použití AlpineJS bylo až ke konci práce, když jsem potřeboval adekvátně spojit frontend a backend. V některých případech jsem narazil na nekonzistentní chování v AlpineJS. Zmíním dva příklady.

Pomocí atributu **x-for** na HTML tagu <*template*> je možné generovat prvky pro každý element pole. Používám jej vždy při zobrazování prvků nějaké tabulky z PocketBase. Když dochází k přidávání a odebírání elementů v poli bez toho, aniž by bylo vytvořeno nové pole, AlpineJS nedokáže poznat, o které prvky se jedná, a dochází k nezobrazování nebo duplikovanému zobrazení prvků.

V těchto případech je zapotřebí specifikovat nějakou vlastnost prvků pole jako klíč:

```
<template x-for="data in budgets" :key="data.id">
  ...
</template>
```

Druhá nekonzistence, se kterou jsem se potýkal, je reaktivnost v atributech **x-effect**. Když je kód moc komplexní a figuruje v něm mnoho reaktivních proměnných, tak je možné, že AlpineJS nedetekuje všechny odkazy a nepouští kód ve všechny správné momenty. Tento problém jsem ve všech případech vyřešil pomocí výše zmíněné **$watch()** funkce sledováním jednotlivých proměnných.

Posledním problémem, který jsem řešil, byl vývoj mobilní verze. Po přečtení mnoha článků jsem dospěl k závěru, že detekovat, zda se nacházím na mobilním zařízení, není vůbec jednoduchá záležitost. Použití *@media queries* podle šířky displeje není jistota, protože mobilní zařízení mají různá rozlišení.

Pomocí *@media queries*[6] nebo JavaScriptu je také možné zjistit, zda obrazovka podporuje ovládání dotykem. Některé prohlížeče ale hlásí podporu dotyku vždy, i když zařízení dotykové není, nebo naopak nehlásí podporu dotyku, i když zařízení dotykovou obrazovku má.

Nakonec jsem se rozhodl si napsat funkci v JavaScriptu, která rozhodne o detekci mobilního zařízení pomocí objektu *screen*:

```
function detect_mobile() {
    return screen.orientation.type == 'portrait-primary'
        || screen.orientation.type == 'portrait-secondary';
}
```

Tato funkce vrací *true*, pokud orientace obrazovky je na výšku, v *portrait* módu. Obrazovka v portrétovém módu je pouze na mobilních zařízeních. U stolního počítače i monitor namontovaný vertikálně je v prohlížeči interpretovaný jako *landscape*.

Tento způsob detekce má některá omezení. Velmi staré prohlížeče Edge, staré mobilní prohlížeče Safari a samozřejmě Internet Explorer tento objekt nepodporují.

[6] Mozilla Corporation. *Using media queries*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries.

3. Technická dokumentace

3.1 Požadavky

Projekt je možné zprovoznit dvěma způsoby, za použití Nixu a bez něj. Postup s použitím Nixu je jednodušší a spolehlivější, ale je použitelný pouze na Linuxových distribucích (a případně Macu), protože pro Windows nemá Nix podporu.

S Nixem jsou závislosti pro spuštění následující:

- Rozumná Linuxová distribuce (nebo MacOS)
- Nix
- Git

V případě postupu bez Nixu je možné použít i Windows, ale je zapotřebí explicitně nainstalovat více závislostí:

- Python 3 (projekt byl vyvíjen s Pythonem 3.10.9, ale neměly by být problémy s kompatibilitou s novějšími verzemi)
- PocketBase
- Flask
- waitress (na deployment)

Následující instrukce jsou pro Linuxové operační systémy, jakožto pro platformu, která se nejvíc hodí na vývoj a provoz aplikací.

3.1.1 Git

Git by měl být předinstalovaný na většině distribucí. Pokud není, zpravidla je možné jej nainstalovat pomocí balíčkovače systému:

```
# debianovité distribuce
sudo apt-get install git

# archovité distribuce
sudo pacman -S git

# void linux
sudo xbps-install -Sy git
```

Pro zprovoznění maturitního projektu takhle git stačí, v praxi si však uživatel často bude potřebovat nakonfigurovat jméno, email, a SSH klíč.

3.1.2 Nix

Nix je možné nainstalovat pomocí balíčkovače na vaší distribuci, například:

```
# debianovité distribuce
sudo apt-get install nix

# archovité distribuce
sudo pacman -S nix

# void linux
sudo xbps-install -Sy nix
```

Doporučený je ale způsob z webové stránky Nixu[1], který nainstaluje Nix přes Nix:

```
$ sh <(curl -L https://nixos.org/nix/install) --daemon
```

Aby uživatelé nepotřebovali na provoz Nixu práva superuživatele, vyplatí se pustit nixového démona:

```
# nix-daemon
```

Každý uživatel si potom musí nastavit proměnnou **NIX_REMOTE** na hodnotu **daemon**:

```
echo 'export NIX_REMOTE=daemon' >> $HOME/.bashrc #nebo jinak pro jiné shelly
```

Funkčnost Nixu můžeme ověřit například následujícím způsobem:

```
$ nix-shell -p neo-cowsay
[nix-shell:~]$ cowsay kůň
 _____
< kůň >
 -----
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

3.1.3 Python

Programovací jazyk Python je, podobně jako Git, na většině distribucí předinstalován. V případě, že tomu tak není, můžeme se ve většině případů opět spolehnout na nativní balíčkovač systému:

[1] https://nixos.org

```
# debianovité distribuce
sudo apt-get install python

# archovité distribuce
sudo pacman -S python

# void linux
sudo xbps-install -Sy python
```

Backend aplikace neobsahuje žádný složitý Python, kde by připadala v úvahu citlivost na verzi Pythonu. Projekt byl vyvinut s Pythonem 3.10.9, tuto verzi a každou novější můžeme považovat za jistotu.

Je důležité podotknout, že některé instalace Pythonu jsou spíš minimalistické a neobsahují nástroje jako *pip* a *venv/virtualenv*, které je potom potřeba doinstalovat separátně.

3.1.4 PocketBase[2]

PocketBase je asi nejméně známý software, co je potřeba nainstalovat pro zprovoznění aplikace. V případě, že nebude dostupný v balíčkovači linuxové distribuce, je zapotřebí jej nainstalovat buď stažením archivu z webu projektu:

`https://pocketbase.io/docs/`

Nebo, v případě, že distribuce obsahuje Go buď nainstalované nebo v repozitáři balíčkovače:

```
go get github.com/pocketbase/pocketbase
```

Pro práci s PocketBase je zapotřebí ještě nainstalovat Pythonovou PocketBase knihovnu. Nejjednodušší je vytvořit virtuální prostředí a nainstalovat PocketBase pomocí *pipu* ze souboru *requirements.txt*:

```
virtualenv -p python3 env
source env/bin/activate
pip install -r requirements.txt
```

3.1.5 Flask

Podobně jako klientskou knihovnu PocketBase je možné Flask nainstalovat ze souboru *requirements.txt* pomocí *pipu*. Vzhledem k tomu, že použití Flasku v tomto projektu není komplexní, měly by fungovat i verze z balíčkovače distribuce.

[2]Gani. *PocketBase*. URL: `https://pocketbase.io/`.

3.1.6 waitress

WSGI server *waitress* je nutný nainstalovat pouze na produkční provoz aplikace, pro vývoj a testování aplikace stačí server zabudovaný v příkazu *flask run*. Program *waitress* se dá nainstalovat z balíčkovače některých linuxových distribucí, nebo pomocí pipu:

```
pip install waitress
```

3.2 Instalace a spouštění

Pro spuštění projektu je nejdříve potřeba naklonovat repozitář:

```
git clone https://github.com/Tomdrs/statni_rozpocet
```

Dobrovolným krokem je spustit lokální instanci PocketBase:

```
pocketbase serve
```

Tato instance PocketBase bude prázdná. Postup pro naimportování databázového schématu je následující:

1. Otevřete v prohlížeči adresu `http://localhost:8090/_/`
2. Vytvořte první admin účet podle instrukcí na obrazovce
3. V kartě *Settings* v sidebaru na levé straně naimportujte schéma ze souboru *pb_schema.json* v repozitáři
4. Případně rozšiřte nebo upravte data v jednotlivých tabulkách

Kromě lokální instance PocketBase je možné použít instanci na adrese `https://pocketbase.gjk.cat`. Dalším krokem je nastavení proměnných prostředí:

```
export FLASK_APP="src/main.py"
export POCKETBASE_URL="https://pocketbase.gjk.cat" # nebo lokální instance
```

a vytvořit a vstoupit do virtuálního Python prostředí, pokud k tomu nedošlo dříve:

```
virtualenv -p python3 envhttps://github.com/Tomdrs/statni_rozpocet
source env/bin/activate
pip install -r requirements.txt
```

3.2.1 Spouštění s Nixem

Pro spuštění s Nixem není nutné explicitně vstupovat do virtuálního Python prostředí nebo nastavovat proměnné prostředí. Výchozí hodnoty jsou v souboru *shell.nix*.

Měly by stačit následující dva příkazy:

```
$ nix-shell
[nix-shell:~/statni_rozpocet]$ flask run
```

3.2.2 Spouštění bez Nixu

Bez Nixu je nutné vstoupit do virtuálního Python prostředí, nastavit proměnné prostředí a vytvořit soubor *static/config.js* s proměnnou *POCKETBASE_URL*. Kompletní postup je následující:

```
export FLASK_APP="src/main.py"
export POCKETBASE_URL="https://pocketbase.gjk.cat" # nebo lokální instance
virtualenv -p python3 envhttps://github.com/Tomdrs/statni_rozpocet
source env/bin/activate
pip install -r requirements.txt
echo "let POCKETBASE_URL = \"$POCKETBASE_URL\";" > static/config.js
flask run
```

V případě úspěchu by měl příkaz *flask run* zobrazit následující text:

```
 * Serving Flask app 'src/main.py'
 * Debug mode: off
WARNING: This is a development server. Do not use it in
a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000 * Serving Flask app 'src/main.py'
Press CTRL+C to quit
```

Nyní by otevření URL adresy `http://127.0.0.1:5000` mělo zobrazit aplikaci.

3.3 Uživatelská dokumentace

3.3.1 Spuštění aplikace

Projekt je webová aplikace, stačí v prohlížeči otevřít URl, na které běží Flask. V případě lokálního běhu aplikace se jedná o URL `http://127.0.0.1:5000`.

Aplikace se skládá ze tří webových stránek:

- Index (Hlavní stránka)
- Stránka přihlašování/registrace
- Admin stránka

3.3.2 Hlavní stránka

Hlavní stránka má tři hlavní sekce:

- Formulář se zobrazením celkových dat rozpočtu
- Formulář na přidání vlastní investice
- Zobrazení investic

Vpravo nahoře na stránce se, v závislosti na stavu uživatele, zobrazují tlačítka *Login*, *Admin* a *Logout*.

Formulář se zobrazením celkových dat rozpočtu

Pod výběrem roku se nalézají tři sloupce.

První sloupec zobrazuje příjmy rozpočtu, druhý formulář na údaje, třetí výdaje rozpočtu.

Aplikace zná průměrné hodnoty pro všechny parametry formuláře, proto aplikace okamžitě vypočítá participaci na rozpočtu a výdajích. Uživatel dodá informace ze svého života, které zná.

Hodnoty není potřeba potvrdit, výpočet se aktualizuje automaticky s postupným vyplňováním formuláře.

Pro návrat na výchozí hodnotu průměrného občana stačí smazat vyplněnou hodnotu.

V případě, že jsou zadány nesprávné údaje, zobrazuje formulář chybu.

Formulář na přidání investice

Tato funkcionalita je dostupná pouze, pokud je uživatel přihlášený buď jako normální uživatel, nebo jako administrátor.

Tento formulář umožňuje přidávat uživatelské investice. Po vyplnění kolonek *Název investice*, *Celkové náklady investice (v mld. Kč)* a *Rok počátku realizace* kliknutí na tlačítko *Přidat* se pod formulářem objeví nová investice.

Investice, které normální uživatel přidá sám pro sebe, jsou předřazeny vždy dostupným investicím, které přidali administrátoři.

V případě, že je uživatel přihlášen jako administrátor, přidává formulář sdílené investice.

Zobrazení investic

Závěrem stránky je vyhledávací pole a galerie dlaždic jednotlivých investic.

Vyhledávací pole filtruje investice. Prohledávané informace jsou jméno a rok spuštění projektu. Pole je interaktivní, filtr se aplikuje automaticky během psaní.

Pokud je uživatel přihlášený, zobrazují se nejdříve jeho investice. U dlaždic pro tyto investice je přidáno tlačítko *Odebrat*, které investici smaže.

Administrátor má tlačítko *Odebrat* u každé investice.

Dlaždice zobrazují rok počátku investice, náklady v miliardách korun, podíl rozpočtu, pokud by byla investice v celé míře splacena ve zvoleném roce, a kolika korunami by se v tomto roce uživatel na realizaci investice podílel.

3.3.3 Stránka přihlášení a registrace

Tato stránka obsahuje kartu s dvěma panely, jeden pro přihlášení, druhý pro registraci.

Formulář registrace umožňuje vytvořit v rámci aplikace nového normálního uživatele.

Pokud je registrace úspěšná, tedy pokud uživatel zadal správné údaje, dojde k přesměrování na formulář přihlášení.

V produkci je možné nakonfigurovat v PocketBase SMTP server, aby posílal potvrzovací emaily po registraci.

Po přihlášení pomocí formuláře dojde k přesměrování na hlavní stránku.

Formulář přihlášení je platný jak pro normální uživatele, tak pro administrátory.

Administrátorské účty lze vytvořit pouze prostřednictvím admin panelu PocketBase na url *<adresa_pocketbase>:<port>/_/*.

3.3.4 Administrátorská stránka

Tato stránka je dostupná pouze přihlášeným administrátorům. Přístup bez přihlášení zobrazí uživateli hlášku, že stránka je určena pouze pro administrátory.

Administrátorům stránka zobrazí rozhraní pro upravování všech tabulek, které jsou uloženy v PocketBase.

Každá tabulka zobrazuje všechny uložené hodnoty a políčka pro přidání nového záznamu. Vedle každého záznamu jsou dvě tlačítka - tužka pro úpravu, křížek pro smazání daného záznamu.

Kliknutí na tlačítko tužky přepne řádek do editovatelného módu, kde jsou všechny buňky přeměněny na upravitelná políčka. Hodnoty se v databázi aktualizují automaticky, není potřeba změny potvrzovat.

Závěr

Když se s již hotovým projektem ohlédnu na začátek a na cíle, které jsem si vytyčil, myslím, že jsem na něm odvedl dobrou práci. Projekt splňuje funkcionalitu, kterou jsem slíbil. Uživatel tak opravdu po zadání svých příjmů a výdajů vidí, kolik a jakým způsobem odvedl do státního rozpočtu peněz a co se pak s nimi stalo. Správa databáze rozpočtů a investic funguje správně, aplikace se tak dá i nadále rozvíjet tím, že se budou přidávat další investice a rozpočty na minulé i budoucí roky. Samozřejmě s častými změnami v zákonech matematika kalkulačky nemůže být plně funkční na budoucí léta i ta předcházející před rokem 2020, pokud nebude ve zdrojovém kódu modifikována. Daňový systém a finanční správa státu jsou natolik složité, že aplikace nemá šanci být naprosto přesná, ale myslím si, že uživateli poskytne velmi zajímavý obrázek o státních financích a o tom, jak se sám na nich podílí.

Když jsem s projektem začínal, měl jsem jenom drobné zkušenosti s Pythonem a žádný jiný programovací jazyk jsem neuměl. Bylo mi tedy jasné, že pro splnění projektu se vše budu muset naučit. Plnil jsem tedy internetové výukové kurzy, sledoval jsem videa a hledal články, které by mi pomohly naučit se alespoň základní práci s prostředky, jež jsem se rozhodl využívat. Mnoho informací a způsobů práce jsem si musel vyhledávat a učit se ještě v průběhu, takže postup byl pomalejší, než bych si přál, a prací jsem strávil o dost více času, než jsem si původně myslel, ale nakonec se to vyplatilo a povedlo se mi projekt dokončit. Výsledkem pro mě tak není pouze aplikace, kterou jsem vytvořil, ale také získané zkušenosti minimálně základní práce s Pythonem, HTML, CSS, Javascriptem, Alpinem, Flaskem a Pocketbasem. Nerozvíjel jsem se ale pouze v technických dovednostech, ale také ve znalostech ekonomických a finančních. Původně jsem si vlastně téma projektu vybral, protože mě ekonomická a finanční témata zajímají a v budoucnu bych se jim chtěl nadále věnovat. S prací na projektu jsem tak nahlédl do světa daňového systému a pochopil, jak složitý vlastně je. Věřím tak, že zkušenosti získané prací na projektu využiji i v budoucnosti.

S výsledkem, který jsem projektu věnoval, jsem spokojený. Ačkoliv se mi občas nedařilo nebo mi dělalo potíže něco vymyslet, vyvinul jsem velké úsilí a pokračoval, dokud jsem nedosáhl cíle. Mnoho také vděčím svému bratrovi, který mi díky svým zkušenostem v oblasti programování dokázal poradit, když jsem měl potíže, a vždy se mnou ochotně projekt konzultoval. Výsledkem je tak funkční aplikace, která poskytuje zajímavé údaje, a mně přinesla její tvorba mnohé zkušenosti.

Seznam použité literatury

[Cor] Mozilla Corporation. *Using media queries*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries.

[Česa] Ministerstvo financí České republiky. *Plnění státního rozpočtu*. URL: https://www.mfcr.cz/cs/verejny-sektor/statni-rozpocet/plneni-statniho-rozpoctu.

[Česb] Úřad vlády České republiky. *Národní investiční plán České republiky 2020-2050*. URL: https://www.vlada.cz/assets/media-centrum/aktualne/Narodni-investicni-plan-CR-2020_2050.pdf.

[Fir] Fireship. *Fireship*. URL: https://www.youtube.com/@Fireship.

[Gan] Gani. *PocketBase*. URL: https://pocketbase.io/.

[Nix] kolektiv NixOS. *Nix*. URL: https://nixos.org/.

[Pal] Pallets. *Flask*. URL: https://flask.palletsprojects.com/en/2.2.x/.

[Por] Caleb Porzio. *Alpine.js*. URL: https://alpinejs.dev/.

[repa] Parlament České republiky. *187/2016 Sb. Zákon o dani z hazardních her*. URL: https://www.zakonyprolidi.cz/cs/2016-187.

[repb] Perlament České republiky. *235/2004 Sb. Zákon o dani z přidané hodnoty*. URL: https://www.podnikatel.cz/zakony/zakon-c-235-2004-sb-o-dani-z-pridane-hodnoty/uplne/#prilohy.

[spra] Finanční správa. *Daň z příjmů*. URL: https://www.financnisprava.cz/cs/dane/dane/dan-z-prijmu.

[sprb] Finanční správa. *Rozpočtové určení daní*. URL: https://www.financnisprava.cz/cs/dane/danovy-system-cr/rozpoctove-urceni-dani.

[Svě] Lubomír Světnička. *Vyjednávání o stíhačkách F-35 pokračují. Česko sonduje možnosti financování*. URL: https://www.idnes.cz/zpravy/nato/stihaci-lockheed-f-35-armada-obran-cupakova-usa-pilot.A230214_194540_zpr_nato_inc.

[věc] Ministerstvo práce a sociálních věcí. *Sociální pojištění*. URL: https://www.mpsv.cz/socialni-pojisteni.

[Wik] Wikipedie. *Spotřební daň*. URL: https://cs.wikipedia.org/wiki/Spot%5C%C5%5C%99ebn%5C%C3%5C%AD_da%5C%C5%5C%88#Reference.

Seznam obrázků

Seznam tabulek