

# **Vývoj webových aplikací pomocí frameworku SvelteKit**

Radim Horčíčka

---

Bakalářská práce  
2023



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

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

Akademický rok: 2022/2023

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Radim Horčíčka  
Osobní číslo: A20435  
Studijní program: B0613A140020 Softwarové inženýrství  
Forma studia: Prezenční  
Téma práce: Vývoj webových aplikací pomocí frameworku SvelteKit  
Téma práce anglicky: Web Application Development Using the SvelteKit Framework

## Zásady pro vypracování

1. Vypracujte literární rešerši týkající se moderních vývojových javascriptových nástrojů Svelte a SvelteKit.
2. Zaměřte se na architekturu a princip fungování zmíněných nástrojů. Uvedte srovnání s jiným aktuálně používaným javascriptovým frameworkem.
3. Navrhněte demonstrační webovou aplikaci a stanovte funkcionální a nefunkcionální požadavky.
4. Pomocí frameworku SvelteKit implementujte demonstrační aplikaci dle stanovených požadavků, věnujte se také zabezpečení webové aplikace.
5. Popište důležité kroky implementace a v závěru shrňte výhody a nevýhody využívání zvoleného frameworku.

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

Seznam doporučené literatury:

1. HOLTHAUSEN, Simon, 2022. Svelte a beginner's guide. B.m.: Sitepoint.
2. SvelteKit [online]. [cit. 2022-11-30]. Dostupné z: <https://kit.svelte.dev/>
3. Svelte [online]. [cit. 2022-11-30]. Dostupné z: <https://svelte.dev/>
4. Vite [online]. Evan You & Vite Contributors, c2019 [cit. 2022-11-30]. Dostupné z: <https://vitejs.dev/>
5. Rendering on the Web [online]. Miller, 2019 [cit. 2022-11-30]. Dostupné z: <https://web.dev/rendering-on-the-web/>
6. HAVERBEKE, Marijn. Eloquent JavaScript: A Modern Introduction to Programming. 3rd Edition. No Starch Press, 2018. ISBN-13: 9781593279509.

Vedoucí bakalářské práce: **Ing. Radek Vala, Ph.D.**  
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **2. prosince 2022**

Termín odevzdání bakalářské práce: **26. května 2023**



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

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

Ve Zlíně dne 7. prosince 2022

**Jméno, příjmení: Radim Horčíčka**

**Název bakalářské práce: Vývoj webových aplikací pomocí frameworku SvelteKit**

**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.

Ve Zlíně, dne

.....  
podpis studenta

## **ABSTRAKT**

Tato bakalářská práce se zabývá moderními vývojovými nástroji Svelte a SvelteKit pro tvorbu webových aplikací. V teoretické části je popsána jejich architektura a princip fungování, včetně srovnání s jinými JavaScriptovými frameworky. V praktické části je navržena a implementována demonstrační webová aplikace s funkčními a nefunkčními požadavky. Dále je také věnována pozornost zabezpečení webové aplikace a detailnímu popisu implementačních kroků.

Klíčová slova: SvelteKit, Svelte, framework, web

## **ABSTRACT**

This bachelor's thesis focuses on modern development tools, Svelte and SvelteKit, for web application development. The theoretical part describes their architecture and functioning principles, including a comparison with other JavaScript frameworks. A demo web application is designed and implemented in the practical part, considering both functional and non-functional requirements. Additionally, attention is given to securing the web application and providing detailed descriptions of the implementation steps.

Keywords: SvelteKit, Svelte, framework, web

Chtěl bych moc poděkovat panu Ing. Radku Valovi, Ph.D. za jeho cenné rady a připomínky v průběhu psaní mé bakalářské práce. Vážím si toho, že mě přijal a byl můj vedoucí. Dále chci poděkovat svojí rodině, která na mě myslela a byla mi oporou.

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

# OBSAH

<b>ÚVOD.....</b>	<b>9</b>
<b>I TEORETICKÁ ČÁST .....</b>	<b>10</b>
<b>1 SEZNAMOVACÍ KAPITOLA .....</b>	<b>11</b>
1.1 PROBLEMATIKA VÝBĚRU NÁSTROJE PRO TVORBU WEBOVÝCH APLIKACÍ.....	11
1.2 SVELTE.....	11
1.3 SVELTEKIT.....	12
1.3.1 Routing.....	12
1.3.2 Načítání dat .....	17
1.3.3 Sdílející složka lib .....	19
1.3.4 Forms.....	19
1.3.5 Snapshots.....	23
1.3.6 Možnosti přednačení odkazů .....	24
1.3.7 data-sveltekit-preload-data .....	24
1.3.8 data-sveltekit-preload-code .....	25
1.3.9 data-sveltekit-reload .....	25
1.3.10 Ostatní data-sveltekit-* možnosti u odkazu .....	25
<b>2 SROVNÁNÍ OPROTI OSTATNÍM JAVASCRIPTOVÝM FRAMEWORKŮM.....</b>	<b>26</b>
2.1 REACT.....	26
2.2 ANGULAR.....	26
<b>II PRAKTICKÁ ČÁST .....</b>	<b>28</b>
<b>3 NÁVRH DEMOSTRAČNÍ WEBOVÉ APLIKACE.....</b>	<b>29</b>
3.1 FUNKCIONÁLNÍ POŽADAVKY .....	29
3.1.1 Registrace .....	29
3.1.2 Přihlášení.....	30
3.1.3 Odhlášení.....	30
3.1.4 Smazání účtu .....	30
3.1.5 Vytvoření textového příspěvku .....	30
3.1.6 Smazání textového příspěvku .....	30
3.1.7 Přepínání mezi tmavým a světlým režimem .....	30
3.1.8 Zobrazení profilu.....	30
3.1.9 Posun uživatele k formuláři v domovské stránce.....	30
3.2 NEFUNKCIONÁLNÍ POŽADAVKY.....	31
3.2.1 Multiplatformní webová aplikace .....	31
3.2.2 Responzivní webová aplikace .....	31
3.2.3 Ucelený vývojový stack .....	31
3.2.4 Použití JavaScriptu.....	31
3.3 PŘÍPADY UŽITÍ.....	32
3.4 DATABÁZOVÝ MODEL .....	36

<b>4</b>	<b>IMPLEMENTACE APLIKACE .....</b>	<b>38</b>
4.1	PŘÍPRAVA PROSTŘEDÍ PRO VÝVOJ.....	38
4.1.1	Vývojové prostředí.....	38
4.1.2	Instalace SvelteKit .....	38
4.1.3	Vytvoření nového projektu .....	38
4.2	ADRESÁŘOVÁ STRUKTURA.....	39
4.2.1	Počáteční struktura projektu.....	39
4.2.2	Příprava struktury projektu .....	40
4.3	KONFIGURACE PROJEKTU .....	40
4.3.1	Konfigurace knihovny Lucia.....	40
4.3.2	Konfigurace databázového nástroje Prisma .....	42
4.3.3	Ostatní knihovny .....	42
4.4	ZABEZPEČENÍ WEBOVÉ APLIKACE .....	43
4.5	POPIS DEMONSTRAČNÍ WEBOVÉ APLIKACE .....	45
4.5.1	Root stránka Home.....	45
4.5.2	Veřejná stránka profilu uživatele .....	48
4.5.3	Sekce nastavení .....	49
4.5.4	Stránka pro registrování .....	50
4.5.5	Stránka pro přihlášení .....	50
4.5.6	Informační přehled .....	51
4.5.7	SEO .....	51
	<b>ZÁVĚR .....</b>	<b>52</b>
	<b>SEZNAM POUŽITÉ LITERATURY .....</b>	<b>53</b>
	<b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK .....</b>	<b>55</b>
	<b>SEZNAM OBRÁZKŮ .....</b>	<b>56</b>
	<b>SEZNAM TABULEK.....</b>	<b>57</b>
	<b>SEZNAM PŘÍLOH.....</b>	<b>58</b>



## ÚVOD

S vývojem světových technologií se staly webové aplikace u některých firem nebo podnikatelů naprosto nepostradatelnou součástí jejich činnosti. Způsob, jak rychle obsloužit mnoho zákazníků ve stejnou chvíli s pomocí webové aplikace, bychom asi těžko nahrazovali. Framework SvelteKit přináší nový způsob, jak vyvíjet rychle, moderně a jednoduše.

Začátek teoretické části popisuje problematiku výběru nástroje pro tvorbu webových aplikací. Následně se práce věnuje frameworku SvelteKit, u kterého jsou popsány klíčové vlastnosti a funkcionality. V neposlední řadě je SvelteKit porovnán vůči jiným JavaScriptovým frameworkům.

Praktická část práce obsahuje návrh a implementaci demonstrační webové aplikace. Nejprve jsou stanoveny funkcionální a nefunkcionální požadavky, které aplikace musí splňovat. Poté je navržen databázový model a v závěru je podrobně popsán postup vývoje demonstrační webové aplikace.

Hlavním cílem této bakalářské práce je představit a zdůraznit výhody frameworku SvelteKit a ukázat, jak jednoduše ho lze implementovat pomocí demonstrační webové aplikace. Přínosem práce jsou informace pro vývojáře a projektové manažery, kteří hledají efektivní způsob vývoje webových aplikací.

## **I. TEORETICKÁ ČÁST**

## 1 SEZNAMOVACÍ KAPITOLA

SvelteKit je postaven na moderním JavaScriptovém frameworku Svelte. To znamená, že využívá technologii Svelte pro rychlejší vývoj a lepší výkon. A zároveň SvelteKit, jakožto nástroj, přináší nové funkce a možnosti, jak si usnadnit tvorbu webových aplikací.

### 1.1 Problematika výběru nástroje pro tvorbu webových aplikací

S nápadem na vytvoření webové aplikace se pojí také otázka výběru vhodného nástroje. Existuje mnoho faktorů, které mohou být při výběru nástroje pro webovou aplikaci brány v úvahu. Protože některé faktory mohou být pro jednotlivé aplikace a vývojové týmy důležitější než jiné, může být tento proces složitý. Proto je nutné zvážit výhody a nevýhody jednotlivých možností pro konkrétní případy.

Prvním faktorem při výběru nástroje může být jeho cena. Mezi ty nejpopulárnější patří open-source JavaScriptové frameworky jako Angular, React nebo Vue. Tyto frameworky jsou poskytovány zdarma a jejich zdrojové kódy lze veřejně dohledat. To znamená, že jejich tvůrci nevydělávají přímo prodejem produktu. Jejich příjmy tvoří sponzorství, provozování placených služeb, jako například online podpora, dále mohou tvůrci provozovat reklamy, nebo zpoplatnit některé knihovny, šablony nebo pluginy. Díky tomu, že jsou nástroje zdarma k užívání, může vzniknout větší komunita než u těch zpoplatněných. Tím vzniká silnější ekosystém, je zde více dokumentace, článků v diskusních fórech nebo návodů. Ekosystém může být jeden z dalších faktorů, ke kterým lze přihlížet, při výběru vhodného nástroje. Samozřejmě je určit, na jaké platformě bude webová aplikace běžet. Pokud se bude používat i na mobilních zařízeních, je vhodné vybrat nástroj, který podporuje responzivní design. Výběr nástroje pro tvorbu webových aplikací také ovlivňuje vývojářský tým a jeho zkušenost s daným nástrojem. Pokud je nástroj nový, může být vývoj aplikace za jeho pomoci pomalejší v porovnání s osvědčenými nástroji, které jsou již populární. Na to navazuje faktor, který se týká plánování do budoucna. Je vhodné vybrat nástroj s perspektivou do budoucna, aby byl podporován a aktualizován i po své první verzi.

### 1.2 Svelte

JavaScriptový open-source framework Svelte pro tvorbu webových aplikací je novým nástrojem, jak zefektivnit vývoj nebo jak zvýšit výkon aplikace. Zakladatelem byl v roce 2016 Rich Harris, který neustále s týmem tento framework vylepšuje. Jeho největší odlišností je, že nepoužívá virtuální DOM pro aktualizaci webové stránky, ale v době

kompile generuje kód, který využívá přímé manipulace s reálným DOM (Document Object Model). To znamená, že Svelte při každé změně v aplikaci nepřepisuje celý virtuální DOM, ale pouze aktualizuje potřebné prvky na stránce, což vede k výraznému zrychlení a zlepšení výkonu aplikace. Zlepšení přispívá i ten fakt, že v době překladu jsou komponenty aplikace přeloženy do čistého JavaScriptu, HTML a CSS kódu a při kompilaci se veškerá interakce uživatele s aplikací převádí na optimalizované JavaScriptové funkce. Svelte je natolik chytrý, že umí poznat a automaticky odstranit z výsledného kódu nevyužité komponenty a kódy. Z důvodu generování kódu během kompilace může být Svelte označen jako kompilátor, přestože má mnoho rysů frameworku. [1] [2]

Pro rychlejší a příjemnější vývoj webových aplikací nabízí Svelte intuitivnější a kratší zápis kódu, než je u ostatních frameworků. Rich Harris na stránkách Svelte porovnává jednoduché části webových aplikací, které jsou psány různými frameworky. Výsledkem je, že React má mnohem delší zápis i u jednoduché komponenty, zatím co Svelte ho má kratší a ve výsledku přehlednější. [3]

Svelte má na svých stránkách kromě velmi pěkně udělaných návodů, ukázek a dokumentace taky reference na webové stránky, které používají Svelte. Mezi ně spadají například weby jménem *Avast.com*, *Chess.com*, *IBM.com*, *Philips.co.uk* nebo *nytimes.com*. [4]

### 1.3 SvelteKit

V říjnu roku 2020 oznámil Rich Harris nástupce za framework pro vývoj webových aplikací jménem Sapper. Jméno nástupce je SvelteKit a jeho první beta verze byla k dispozici v březnu 2021. Tento nový framework je také postavený na Svelte a jeho předností je optimalizace pro výkon a načítání. Jedná se tedy znovu o primárně webový frontend framework, ale umožňuje i vývoj backendu pomocí serverless funkcí. [5]

Tento nástroj jménem SvelteKit nabízí vývojářům celou škálu funkcí, od základního routingu, který má za úkol aktualizovat UI při kliknutí na odkaz, až po různé způsoby renderování stránek. K tomu všemu je podporován vývoj v jazyce TypeScript. [6]

#### 1.3.1 Routing

SvelteKit využívá pro routing takzvaný filesystem-based router, je to jednoduchý způsob, jak definovat routes pro webové stránky pomocí struktury složek a souborů v projektu. To znamená, že vývojář nemusí manuálně definovat routes v kódu, ale stačí mu pouze řešit logiku struktury složek a souborů v projektu.

`src/routes` – je root route, ve které se nachází soubory, které ve výsledku zobrazují uživateli domovskou stránku. Dále obsahuje složky, které představují další stránky webu. Například adresa `src/routes/about` je adresa stránky jménem *about* a její logika je napsána ve složce `/about`. Pokud bude chtít vývojář dynamickou stránku, může využít *slugů*, které představují dynamický parametr v URL adrese stránky. Pro nastavení dynamické webové stránky pomocí parametru stačí přidat do složky odpovídající stránky další složku, která bude mít název slugu v hranatých závorkách. Pomocí adresy `src/routes/blog/[slug]` vznikne dynamická stránka *blog*, která bude měnit svůj obsah nebo chování podle slugu na konci URL adresy.

Příkladem by byla webová stránka, která má domovskou stránku *home* a stránku *kontakty*: Projekt obsahuje `src/routes/+page.svelte`, kde je napsána logika pro homepage, a obsahuje `/src/routes/kontakty/+page.svelte`, kde se nachází html kód pro stránku *kontakty*. Díky vytvoření této struktury složek a souborů v projektu nám SvelteKit automaticky mapuje routes webu. To znamená, že pro jednoduché přesměrování na domovskou stránku nebo do kontaktů stačí vývojáři napsat do html následující:



```
1 <a href="/">Home</a>
2 <a href="/kontakty">Kontakty</a>
```

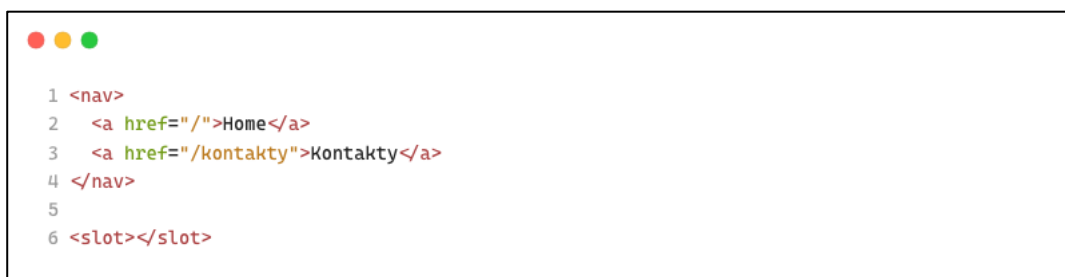
Obrázek 1 Kód v html pro přepínání mezi stránkami ve filesystem based routingu

### 1.3.1.1 *+page*

Soubor jménem `+page.svelte` definuje stránku naší aplikace. Ve výchozím nastavení jsou stránky vykreslovány jak na serveru (SSR) pro počáteční požadavek, tak v prohlížeči (CSR) pro další navigaci. Pokud stránka potřebuje pro úplné zobrazení nějaká data, je potřeba přidat modul s funkcí *load*. Tato funkce je napsána v souboru `+page.js` a může na serveru a v prohlížeči vykonávat různé úlohy v závislosti na dostupných zdrojích nebo prostředí. Pokud vývojář bude chtít například manipulovat s citlivými daty v databázi, je vhodné napsat tuto funkci *load* do souboru `+page.server.js`. Ten, na rozdíl od obyčejného `+page.js`, obsahuje kód pouze pro serverovou stranu.

### 1.3.1.2 Layout

Aby vývojář nemusel zbytečně psát znovu se opakující kód, který se vyskytuje na více stránkách webové aplikace, například menu nebo footer, může využít takzvaného layoutu. Je to soubor, který představuje šablonu, do které lze přidávat unikátní obsah pro určitou stránku. Tento obsah představuje komponenta `<slot></slot>`, která se nachází v souboru `src/routes/+layout.svelte`.



```
1 <nav>
2   <a href="/">Home</a>
3   <a href="/kontakty">Kontakty</a>
4 </nav>
5
6 <slot></slot>
```

Obrázek 2 Přidání navigace pro všechny stránky pomocí layout

### 1.3.1.3 Slug

Část URL adresy, která identifikuje konkrétní stránku nebo obsah, se ve SvelteKit nazývá Slug. Když uživatel navštíví stránku s daným slugem, SvelteKit použije slug pro načtení dat nebo pro načtení obsahu z API a vygeneruje stránku s těmito daty.

Pro ukázkou je do menu přidán seznam uživatelů („Uživatelé“), ve kterém pomocí *fetch* požadavku získáme z API jménem DummyJSON přezdívky uživatelů a zobrazíme je na stránce. Adresa k tomuto seznamu je `http://127.0.0.1:5173/users`. Pro využití slugu se při rozkliknutí přezdívky otevře stránka s jeho podrobnějšími detaily. Slug v URL bude představovat userID. Struktura v projektu pro tento slug je `src/routes/users/[userId]`. Díky tomu stránka obsahuje přezdívku, email a věk uživatele, který byl zadán na pozici slugu. Výsledná adresa při vývoji na localhost pro zobrazení uživatele s userID číslo jedna je `http://127.0.0.1:5173/users/1`



```
1 export const load = async () => {
2   const data = await (await fetch('https://dummyjson.com/users?limit=5%27')).json()
3   return data;
4 };
```

Obrázek 3 Obsah v `src/routes/users/+page.ts` pro získání uživatelů

```
1 <script lang="ts">
2   export let data;
3   const { users } = data;
4 </script>
5
6 <h1>Seznam uživatelů</h1>
7
8 {#each users as user}
9   <p><a href="/users/{user.id}">{user.username}</a></p>
10 {/each}
```

Obrázek 4 Obsah v src/routes/users/+page.svelte pro zobrazení uživatelů

```
1 export const load = async ({fetch, params}) => {
2   const fetchUser = async (id: any) => {
3     const response = await fetch(`https://dummyjson.com/users/${id}`);
4
5     const data = response.json()
6     return data;
7   };
8
9   return {
10     user: await fetchUser(params.userId)
11   };
12 }
```

Obrázek 5 Obsah v src/routes/users/[userId]/+page.ts pro získání dat uživatele

```
1 <script>
2   export let data;
3   const {user} = data;
4 </script>
5
6 <h1>{user.username}</h1>
7 <p>{user.email}</p>
8 <p>{user.age}</p>
```

Obrázek 6 Obsah v src/routes/users/[userId]/+page.svelte

### 1.3.1.4 Error

SvelteKit poskytuje taky možnosti, jak nakládat s různými typy chyb. V různých případech lze vykreslit uživateli různé chybové hlášení v závislosti na místě výskytu, typu chyby nebo celkově na povaze příchozího požadavku pro webový server.

Chyby se podle dokumentace SvelteKit dělí na očekávané (expected errors) a neočekávané (unexpected errors). V obou případech chyba představuje jednoduchý objekt, obsahující HTTP status a chybovou hlášku. Property v objektu chyby lze rozšířit o vlastní libovolné hodnoty. K tomu je potřeba upravit Error interface v *src/app.d.ts* tak, aby obsahoval požadovanou property navíc.

Očekávané chyby jsou v kódu aktivovány pomocí *error*, který je importován z knihovny *@sveltejs/kit*. Protože je chyba očekávaná, můžeme ji přímo v kódu aplikace zachytit a uživateli ji ukázat. Pokud do struktury aplikace přidáme *+error.svelte*, můžeme vykreslit uživateli unikátní stránku pro danou chybu zachycenou v *+page.server.ts*.

Ukázka zachycuje očekávanou chybu, kdy uživatel zadal do webové URL id uživatele, který neexistuje.

```
1 import {error} from '@sveltejs/kit'
2
3 export const load = async ({fetch, params}) => {
4   const fetchUser = async (id: any) => {
5     const response = await (await fetch(`https://dummyjson.com/users/${id}`));
6
7     if (!response.ok) {
8       throw error(404, {message: User with id '${id}' not found,
9                           code: 'USER_NOT_FOUND'});
10    }
11
12    const data = response.json()
13    return data;
14  };
15
16  return {
17    user: await fetchUser(params.userId)
18  };
19 }
```

Obrázek 7 Zachycení chyby pro neexistující ID v *src/routes/users/[userId]/+page.server.ts*



The image shows a code editor window with a white background and a black border. At the top left, there are three colored circles (red, yellow, green) representing window controls. The code is written in a light blue monospaced font. It consists of 11 lines of SvelteKit code. Line 1: `<script lang="ts">`. Line 2: `import { page } from '$app/stores';`. Line 3: `</script>`. Line 4: . Line 5: `<h1>Jejda!</h1>`. Line 6: `<p>{$page.status}: {$page.error?.code}</p>`. Line 7: `<p>{$page.error?.message}</p>`. Line 8: . Line 9: `<pre>`. Line 10: `{JSON.stringify($page,null,2)}`. Line 11: `</pre>`.

Obrázek 8 Soubor `src/routes/users/[userId]+error.svelte` pro vykreslení chyby uživateli.

Neočekávané chyby představují bezpečnostní riziko, proto se při jejich zobrazování uživateli neukazuje chybová hláška, ale pouze její zobecnění („Internal Error“) a chybový status s číslem 500. Tyto neočekávané chyby se v aplikaci volají pomocí: `„throw new Error();“`, na rozdíl od očekávané chyby: `„throw error(404, {message: `not found`});“`.

Pokud chyba nastane při načítání kořenového layoutu (`src/+layout.svelte`), nebo při vykreslování `+error.svelte` stránky, SvelteKit zobrazí nouzovou chybovou stránku. Pokud bychom ji chtěli upravit, musíme vytvořit soubor `src/error.html`.

### 1.3.1.5 Redirect

Pro přesměrování lze ve SvelteKit využít jednoduchého příkazu: `„throw redirect(308, '/')“`, který uživatele přesměruje na vývojářem určenou stránku, pod určeným stavovým kódem třídy 3xx. Ve zmíněném příkazu bude uživatel přesměrován na hlavní root stránku pod stavovým kódem 308, který představuje přesměrování s trvalou platností (anglicky Permanent Redirect) .

### 1.3.2 Načítání dat

SvelteKit dokumentace nabízí postupy pro načítání dat, které vytvářejí jak obsah, tak i vzhled nebo chování stránky.

#### 1.3.2.1 load

Funkce `load` ve SvelteKit je určena k načítání potřebných dat pro správné vykreslení stránky. Tato funkce se nachází ve stejné složce jako `+page.svelte`, a to v souboru `+page.js`. V případě, že stránka obsahuje layout a ten potřebuje pro své vykreslení načíst data, funkce `load` bude v souboru `+layout.js`. Dva zmíněné příklady načítání dat pomocí `load` běží jak na

serveru, tak i v prohlížeči. To znamená, že když uživatel poprvé navštíví webovou stránku, data se načítají na serveru během Server-Side-Renderingu (SSR) a uživatel dostane jako odpověď na požadavek již vykreslenou stránku.

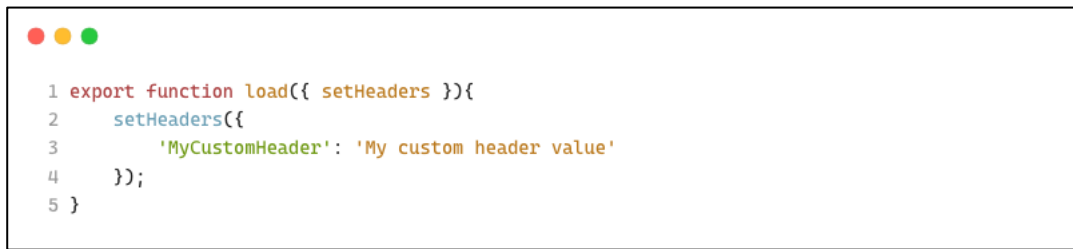
Pokud uživatel bude chtít obnovit stránku, nastane hydratace a načtená data se znovu použijí k obnovení stránky na straně klienta, aby se minimalizovalo množství dat, která musí být znovu načtena ze serveru. Po prvním načtení stránky se funkce *load* spouští pouze v prohlížeči. Výstupem univerzálního *load* je objekt, který obsahuje jakákoliv data (vlastní třídy, konstruktor komponent).

Oproti univerzálnímu *load*, je tu ještě server *load* funkce, která vždycky pracuje na straně serveru. Využívá se při načítání dat z databáze s údaji, které mohou být zneužity. Proto připojení k databázi probíhá pouze na serveru, tím pádem není přístupné v klientském JavaScriptu. Serverové *load* funkce vrací serializovaná data, která lze reprezentovat jako JSON. [7]

### 1.3.2.2 Nastavení hlaviček HTTP

Součástí HTTP požadavku nebo odpovědi je hlavička, která se vyskytuje na začátku požadavku nebo odpovědi, pod stavovým kódem a nad samotným HTML kódem stránky. Hlavička je nositelem různých doplňujících informací, které mohou být důležité pro správný chod webové aplikace. U požadavků mohou být hlavičky *HTTP reffer*, které nesou informaci o stránce, ze které uživatel přišel. Nebo hlavička *User Agent*, která informuje o typu prohlížeče. Příkladem hlavičky u odpovědi je *Age*, která informuje o tom, jak dlouho byla odpověď uložena v proxy cache. [8] [9]

SvelteKit nabízí možnost jak tyto hlavičky, anglicky headers, v odpovědích nastavit. Příkladem by byla funkce *load* v souboru *+page.server.js*. Obsahovala by v parametru objekt s metodou *setHeaders*. Tato metoda by se volala v těle funkce a obsahovala by argument, který je objekt obsahující klíč a hodnotu, přičemž klíč je název hlavičky, kterou chceme nastavit. Tím pádem soubor *+page.svelte*, který je na stejné úrovni jako *+page.server.js*, vždycky před svým vykreslením ve webové aplikaci využije funkci *load*, která nastavuje naši požadovanou hlavičku. Hlavičky lze zobrazit v nástrojích pro vývojáře v patřičném webovém prohlížeči. [7]

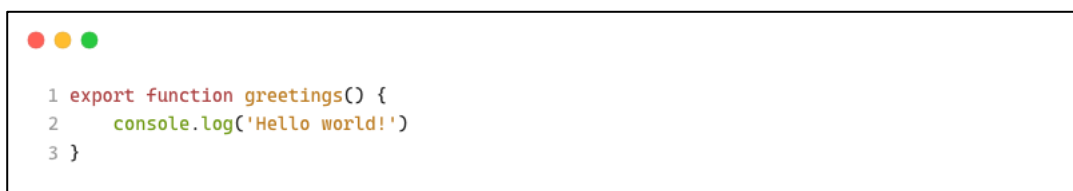


```
1 export function load({ setHeaders }) {  
2   setHeaders({  
3     'MyCustomHeader': 'My custom header value'  
4   });  
5 }
```


Obrázek 9 Nastavení hlavičky ve funkci load

### 1.3.3 Sdílející složka lib

Když je potřeba použít stejný kód na mnoha místech, lze jej napsat pouze jednou, do jednoho souboru a ten potom načíst pomocí importu s prefixem typu „`./.././../`“. SvelteKit nabízí možnost tohoto globálního sdílení jakémukoliv modulu ve složce *src*, a to pomocí volání aliasu složky *lib*. Ve zkratce to znamená, že pokud webová aplikace bude využívat stejný kód na více místech, vytvoří se ve složce *src* nová složka *lib*. Do té se napíše kód, který bude požadovaný pro sdílení napříč všemi soubory v *src*. Pro importování je důležité zadat adresu „`$lib/[jméno_souboru].js`“.



```
1 export function greetings() {  
2   console.log('Hello world!')  
3 }
```

Obrázek 10 Sdílená funkce v `src/lib/greet.js`

```
1 <script>  
2   import { greetings } from '$lib/lib.js';  
3   greetings();  
4 </script>
```

Obrázek 11 Import funkce pomocí `$lib`

### 1.3.4 Forms

Pro získání vstupních dat od uživatele SvelteKit nabízí funkci *form*, která poskytuje tuto výměnu informací mezi internetovým prohlížečem a serverem. Element formuláře může obsahovat různé atributy. Jedním z nich je atribut *method*. Ten definuje, kterou metodou požadavku HTTP se mají data odeslat. Další atribut uvnitř elementu `<form>` je *action*. Do

tohoto atributu se zapisuje místo, kam mají být odeslána data z formuláře. V případě, že atributy nejsou vývojářem zapsány, nastává defaultní stav, ve kterém *method* atribut je požadavek typu GET a *action* atribut odesílá data z formuláře na stejnou stránku, na které se nachází. Metoda HTTP požadavku jménem GET je ale velmi nebezpečná, protože data z formuláře by byla součástí URL adresy. To představuje bezpečnostní riziko například u formulářů, kde uživatel zadává svoje heslo. Proto se využívá metoda POST, která vrací data v těle HTTP odpovědi. Pokud vývojář chce použít více akcí na jedné stránce, může vytvořit pojmenované *actions*. Tyto server-side *actions* se přidávají pod unikátním jménem do souboru `+page.server.js` v části, kde se exportuje konstanta *actions*. Po vytvoření pojmenovaných *actions* nelze mít defaultní možnost, která zpracuje data z formuláře. Je pouze jedna defaultní *action* nebo jedna a více pojmenovaných *action*. Pro vývojáře je také k dispozici nová funkce, která informuje uživatele o tom, že něco bylo špatně zadáno. Funkce se nazývá *fail* a vrací data z *action* spolu s příslušným stavovým kódem HTTP. V ukázce je jednoduchý formulář pro seznam úkolů, který uživateli umožňuje přidávat nové úkoly do seznamu a taky je mazat po jednom nebo hromadně. Tyto akce se budou jmenovat *addTodo*, *removeTodo*, *clearTodos*. Uživatel zároveň nemůže přidat do seznamu prázdný řetězec.



```
1 import { fail } from '@sveltejs/kit'
2 import { addTodo, clearTodos, getTodos, removeTodo } from '$lib/server/database'
3
4 export function load () {
5   const todos = getTodos()
6   return { todos }
7 }
8
9 export const actions = {
10   addTodo: async ({ request }) => {
11     const formData = await request.formData()
12     const todo = String(formData.get('todo'))
13
14     if (!todo) {
15       return fail(400, { todo, missing: true })
16     }
17
18     addTodo(todo)
19     return { success: true }
20   },
21
22   removeTodo: async ({ request }) => {
23     const formData = await request.formData()
24     const todoId = Number(formData.get('id'))
25     removeTodo(todoId)
26     return { success: true }
27   },
28
29   clearTodos: () => {
30     clearTodos()
31   }
```

Obrázek 12 Ukázka tří actions v +page.server.ts pro formulář

```
1 <script lang="ts">
2   import { enhance } from '$app/forms';
3   export let data;
4   export let form;
5 </script>
6
7 <form method="POST" action="?/addTodo" use:enhance>
8   <div class="form-group">
9     <input type="text" name="todo" value={form?.todo ?? ''} />
10    {#if form?.missing}
11      <p class="error">This field is required</p>
12    {/if}
13
14    <div class="button-group">
15      <button type="submit">Add</button>
16      <button formaction="?/clearTodos" type="submit">Clear</button>
17    </div>
18  </div>
19 </form>
20
21 {#if form?.success}
22   <p>New todo added!</p>
23 {/if}
24
25 <ul>
26   {#each data.todos as todo}
27     <li>
28       <span>{todo.text}</span>
29       <form method="POST" action="?/removeTodo">
30         <input type="hidden" name="id" value={todo.id} />
31         <button class="delete" type="submit">delete</button>
32       </form>
33     </li>
34   {/each}
35 </ul>
36
37
38 <style>
39   .error {
40     color: red;
41   }
42 </style>
```

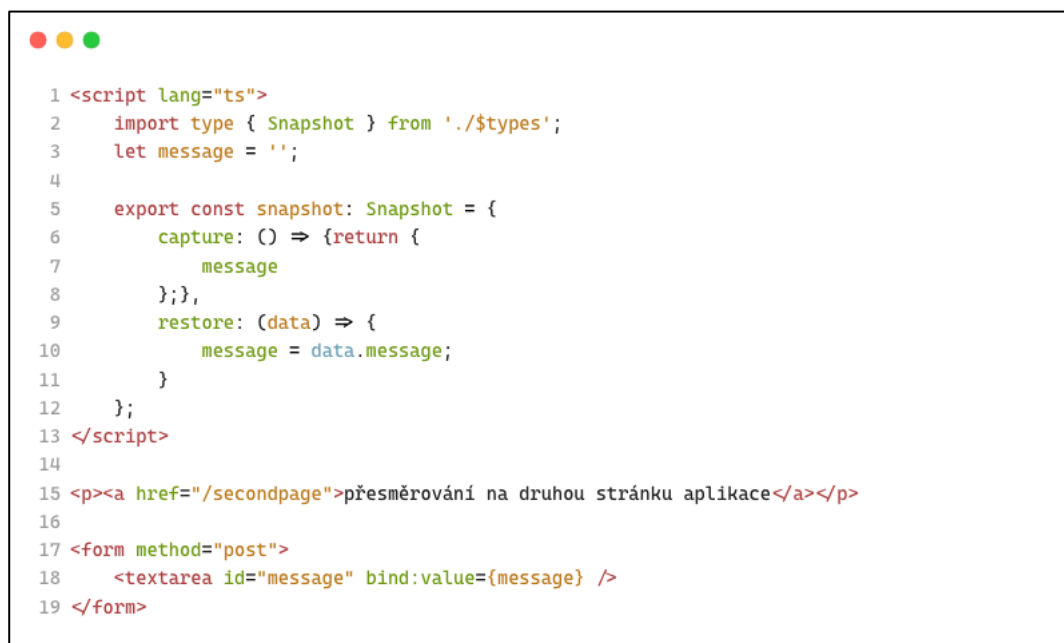
Obrázek 13 Kód v +page.svelte pro formulář

Zmíněný postup implementace formuláře funguje i bez JavaScriptu na straně klienta, ale kvůli tomu se stránka znovu načte, když nastane nějaká akce. Pro zabránění tomuto chování SvelteKit nabízí progresivní vylepšení formulářů ve webových aplikacích, které ale vyžaduje, aby uživatel měl povolený JavaScript v prohlížeči. Pokud se stane, že uživatel nemá JavaScript povolený, formulář bude fungovat s opětovným načítáním stránky při jeho používání. Progresivní vylepšení, které uživateli přinese pocit jednostránkové webové

aplikace (SPA)/nativního chování prohlížeče se po importu *enhance* z *\$app/forms* přidává do elementu `<form>`, kde stačí opravdu jenom napsat *use:enhance*.

### 1.3.5 Snapshots

Při práci s formuláři může uživatel jednoduše ztratit svá rozepsaná data při přechodu na jinou stránku ve webové aplikaci. Pokud uživatel rozepíše formulář a překlikne před jeho odesláním na jiný odkaz, nepomůže mu ani zpětné tlačítko „vrátit zpět“ v prohlížeči a všechno rozepsané ve formuláři ztratí. Název Snapshots, který by se dal přeložit jako zachycení snímku, nemá daleko od jeho principu fungování. Pro uložení rozepsaného formuláře nebo třeba dlouze scrollované pozice v boční navigaci stránky funkce Snapshots zachytí a uloží stav všech elementů v Document Object Model, neboli DOM state. Tento stav může být poté znovu zobrazen při zpětném vrácení se na stránku. Pro aplikaci tohoto chování musíme prvně v souboru *+page.svelte* nebo *+layout.svelte*, záleží na tom kde se vyskytují data k uložení, naimportovat Snapshots z modulu *\$types*, poté ve scriptovací části vytvořit proměnnou s defaultní prázdnou hodnotou, která následně bude nahrazena rozepsaným textem. Název proměnné se poté uvede k příslušnému formulářovému poli pomocí elementu *bind*, do kterého se zapisuje vstup. Aby SvelteKit mohl zachytit a následně obnovit data, musí být do script části exportován *snapshot* objekt s metodami *capture* a *restore*. Funkce *capture* nemá žádné vstupní parametry a vrací aktuální hodnotu proměnné definované na začátku. Pokud uživatel přejde zpátky, druhá funkce *restore* je zavolána s uloženou hodnotou ihned po aktualizaci stránky. To znamená, že funkce *restore* dosadí do proměnné uloženou hodnotu z předešlého zachycení a tato hodnota je potom ve formuláři uživatele díky tomu, že se propojí pomocí elementu *bind* přímo do konkrétního elementu.

A screenshot of a code editor window with a white background and a black border. The editor has three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in a light-colored font with syntax highlighting. It shows a TypeScript script for SvelteKit snapshots, including imports, variable declarations, and HTML tags for a link and a form.

```
1 <script lang="ts">
2   import type { Snapshot } from './$types';
3   let message = '';
4
5   export const snapshot: Snapshot = {
6     capture: () => {return {
7       message
8     }};
9     restore: (data) => {
10       message = data.message;
11     }
12   };
13 </script>
14
15 <p><a href="/secondpage">přesměrování na druhou stránku aplikace</a></p>
16
17 <form method="post">
18   <textarea id="message" bind:value={message} />
19 </form>
```

Obrázek 14 Snapshots kód v +page.svelte

### 1.3.6 Možnosti přednačtení odkazů

Pro úsporu drahocenných milisekund nabízí SvelteKit funkci, která může uživateli rychleji vykreslit stránku v dané webové aplikaci po přesměrování z kliknutí na odkaz. Tyto funkce mají předponu *data-sveltekit-*\* a vkládají se jako atribut do HTML tagu `<a>`. Tento způsob urychlení lze použít i u `<form>` s HTTP požadavkem typu GET.

### 1.3.7 data-sveltekit-preload-data

Atribut pro přednačtení dat může mít dvě hodnoty – *hover* nebo *tap*. První hodnota *hover* už podle názvu přednačte stránku z odkazu po najetí kurzoru myši na odkaz. V případě dotykového zařízení, preload začne po zaregistrování eventu *touchstart*, který představuje přiložení prstu na odkaz, SvelteKit naimportuje kód stránky, potom volá *load* funkce, které potřebují *fetch*, a to vše začne ještě před oddělením prstu z odkazu. V případě hodnoty *tap*, preload preload začíná po stisknutí levého tlačítka myši nad kurzorem, ale nečeká se na jeho odkliknutí. Defaultní šablona projektu ve SvelteKit má nastavený *hover* preload pro každý odkaz v elementu `<body></body>`. V případě, že uživatel má režim snížené spotřeby dat, preload nebude fungovat.



### 1.3.8 data-sveltekit-preload-code

Vývojář využívá této funkce, pokud chce načíst pouze potřebný JavaScript. Jeho atribut může nabývat 4 hodnot. A to sice *hover*, *tap*, *eager* a *viewport*. První dvě hodnoty fungují obdobně jako v *data-sveltekit-preload-data*. Co se týče hodnoty *eager*, ta slouží k okamžitému přednačení kódu ihned po načtení stránky, na které se odkaz vyskytuje. To ale může být nevýhodné v případě stránky, která má hodně odkazů. Proto může vývojář využít hodnotu *viewport*. Díky tomu se kód stránky načte poté, co uživatel bude odkaz fyzicky uvidět na obrazovce. Neboli, pokud bude odkaz vidět v pohledové oblasti prohlížeče. Tím pádem se kód začne stahovat potom, co se uživatel dostane posouváním do spodní části webové stránky, kde se vyskytuje odkaz.

### 1.3.9 data-sveltekit-reload

Protože SvelteKit obvykle přeskakuje mezi stránkami bez jejich znovu načtení (refresh), může někdy vývojář potřebovat opačné chování při kliknutí na odkaz. To může udělat za pomoci atributu *data-sveltekit-reload*.

### 1.3.10 Ostatní data-sveltekit-\* možnosti u odkazu

Atribut *data-sveltekit-replacestate* umožňuje nahradit aktuální položku v historii prohlížeče novou položkou. To se může hodit například v případech, kdy není žádáno, aby byla vytvořena nová položka v historii po každém kliknutí na odkaz. Pro lepší představu, pokud je uživatel na stránce č. 1 a klikne na odkaz stránky č. 2, na které klikne na upravený odkaz stránky č. 3 s atributem *data-sveltekit-replacestate*. Tak při kliknutí na tlačítko zpět přeskočí stránku č. 2 a vyskytne se na stránce č. 1.

## 2 SROVNÁNÍ OPROTI OSTATNÍM JAVASCRIPTOVÝM FRAMEWORKŮM

SvelteKit rozhodně není jediný framework pro tvorbu webových aplikací. Tato kapitola se věnuje popisováním rozdílů u ostatních vývojových frameworků.

### 2.1 React

Jeden z nejvíce používaných a nejoblíbenějších frameworků na světě je React.js od společnosti Meta. V roce 2013, ještě za názvu společnosti jako Facebook, vyšly open-source zdrojové kódy pro širší veřejnost. I přes většinu negativních ohlasů v začátcích Reactu, je dnes tento framework velice oblíbený, podle průzkumu od Stack Overflow z roku 2022, více než 40 % vývojářů preferuje framework React pro tvorbu webových aplikací. Přes 4,8 miliónů webových stránek využívá tento nástroj a mezi ně patří i známé firmy jako Airbnb, Uber, Netflix a samozřejmě taky samotná Meta. [10] [11]

React se zaměřuje na tvorbu uživatelského rozhraní za pomoci komponent, které rozdělují aplikaci na menší znovupoužitelné části, díky kterým pak mohou vzniknout složitější celky. Pro aktualizaci těchto komponent ve webové aplikaci využívá React Virtuální Document Object Model neboli ve zkratce VDOM. Ten představuje abstraktní webovou aplikaci, kde každá komponenta je reprezentována jako strom uzlů, který v sobě nese informace o stavu a vlastnostech komponentu. Pokud se komponenty aktualizují, vzniká proces sjednocení, který vypočítá rozdíl mezi aktuálním stromem uzlů a novým stromem uzlů, který představuje změněnou komponentu. Rozdíl slouží k určení minimálního počtu operací, které jsou potřeba pro změnu uživatelského rozhraní. Jakmile je rozdíl vypočítán, React aktualizuje v DOM pouze ty rozdíly, které byly vypočítané pomocí VDOM. To znamená, že zlepšuje výkon aplikace tím, že zasahuje do reprezentace HTML stránky (DOM) v prohlížeči až na základě rozdílů mezi stromy uzlů. Pro lepší vývoj a manipulaci s komponenty je v Reactu využívána rozšířená JavaScriptová syntaxe JSX. Ta umožňuje vývojářům vkládat kód podobný HTML přímo do JavaScriptu. Kompilaci z JSX do JavaScriptu za vývojáře udělá React. [12]

### 2.2 Angular

Do rodiny frameworků pro tvorbu webových aplikací, založených na komponentové architektuře, přidala oficiálně v roce 2016 společnost Google další framework jménem Angular 2.0, který byl nástupcem staršího nástroje AngularJS. V současné době se tento

nový framework označuje pouze jako Angular a nezmiňuje se jeho verze. Jak již zmíněno, Angular pracuje podobně jako React, protože webová aplikace se skládá ze znovu použitelných komponentů. Pro rychlejší a bezpečnější vývoj se programuje v jazyce TypeScript, který je syntaktickou nadstavbou pro jazyk JavaScript. To znamená, že TypeScript umí funkce z JavaScriptu, ale navíc přidává další funkce a vlastnosti, a to zejména silnější typování a objektově orientované prvky. Pro samotnou logiku aplikace se v Angularu využívají takzvané služby. Jsou to třídy, které poskytují určitou funkcionalitu například komunikaci nebo správu dat mezi klientem a serverovým API. Pokud chce komponenta použít službu, nemusí si ji sama vytvářet, stačí aby požádala Angular, ten pak službu vytvoří a vloží do komponenty za použití návrhového vzoru Dependency Injection. Díky tomu je kratší kód, rychlejší vývoj a lepší znovu použitelnost. Další funkce, kterou framework Angular nabízí je routing, který umožňuje snadno a efektivně přepínat mezi různými zobrazeními nebo komponentami bez nutnosti opakovaného načítání stránky. Angular dále poskytuje vývojářům několik různých typů testování, jako jsou unit testy nebo integrační testy. Výhodou je, že testování lze provádět bez nutnosti spouštění aplikace v prohlížeči, tím pádem jsou testy rychlejší a jednodušší na otestování. Vývojář může využít i dalších funkcí jako jsou Server-side rendering, Pre-rendering, Forms pro získání uživatelského vstupu a validaci dat, Web workers pro výpočetní operace mimo hlavní vlákno za účelem rychlé a responsivní aplikace, nebo funkce která pomáhá dělat animace. [13]

## **II. PRAKTICKÁ ČÁST**

### 3 NÁVRH DEMOSTRAČNÍ WEBOVÉ APLIKACE

Cílem webové aplikace bude vytvořit prostředí pro uživatele, kde můžou vytvářet a mazat své příspěvky. Důležitou součástí bude i knihovna Lucia, která usnadňuje implementaci autentizace a správy uživatelů ve frameworku SvelteKit. Uživatelé se budou moci registrovat, přihlašovat, vytvářet příspěvky, mazat příspěvky nebo smazat všechny svoje údaje z databáze, která je zprostředkována pomocí moderního open-source databázového nástroje jménem Prisma. Tento nástroj pracuje s relačními databázemi a usnadňuje vývoj, správu a bezpečnost databáze. V neposlední řadě bude webová aplikace kontrolovat data zadané uživatelem do formuláře pomocí knihovny pro validaci dat jménem Zod. [14] [15]

#### 3.1 Funkcionální požadavky

Webová aplikace musí splňovat funkcionální požadavky, které jsou detailněji popsány v této kapitole.

Tabulka 1 Funkcionální požadavky stručně

ID požadavku	Popis požadavku pro uživatele
FP1	Uživatel má možnost se zaregistrovat do systému.
FP2	Uživatel se může přihlásit ke svému účtu.
FP3	Uživatel s povoleným Javascriptem se může odhlásit.
FP4	Uživatel s povoleným Javascriptem může odstranit svůj účet.
FP5	Uživatel může vytvořit nový příspěvek.
FP6	Uživatel s povoleným Javascriptem může mazat své příspěvky
FP7	Uživatel s povoleným Javascriptem může měnit světelné režimy.
FP8	Uživatel může zobrazit cizí profil.
FP9	Uživatel s povoleným Javascriptem může využít tlačítko pro posun.

##### 3.1.1 Registrace

Uživatel má možnost vytvořit si pomocí hesla a přezdívky svůj vlastní účet ve webové aplikaci. Registrace je uskutečněna v případě, že uživatelův vstup splňuje podmínky, jako je unikátní přezdívka nebo délka textu. Při nesplňujících podmínkách je uživatel upozorněn přímo na stránce s registračním formulářem.

### **3.1.2 Přihlášení**

Zaregistrovaným uživatelům je umožněno se přihlásit do svého účtu. Přihlášení vyžaduje uživatelskou přezdívku a heslo, které zadal při registraci. Na případně špatně zadané údaje je uživatel upozorněn ve webové aplikaci.

### **3.1.3 Odhlášení**

Webová aplikace umožňuje se uživateli odhlásit ze svého účtu. Při odhlášení jsou vymazány session cookies.

### **3.1.4 Smazání účtu**

Přihlášený uživatel má možnost si permanentně odstranit účet společně s jeho údaji a příspěvky ve webové aplikaci.

### **3.1.5 Vytvoření textového příspěvku**

Přihlášený uživatel má možnost vytvořit nový textový příspěvek. Příspěvek musí obsahovat nadpis a textovou zprávu, kterou chce uživatel napsat. Tyto údaje zároveň musí splňovat podmínky pro textovou délku.

### **3.1.6 Smazání textového příspěvku**

Přihlášený autor příspěvku může svůj příspěvek smazat. Po odstranění není příspěvek na domovské stránce, na stránce autora nebo v databázi.

### **3.1.7 Přepínání mezi tmavým a světlým režimem**

Uživatel má možnost přepínat mezi světlými režimy na každé stránce webové aplikace. Přepnutí režimu ihned po provedení ovlivní vzhled webové aplikace.

### **3.1.8 Zobrazení profilu**

Uživatel má možnost zobrazit svůj nebo cizí profil, který bude obsahovat pouze příspěvky uživatele, kterého je onen daný profil.

### **3.1.9 Posun uživatele k formuláři v domovské stránce**

Pokud je uživatel přihlášený, tlačítko s fixní pozicí a nápisem „create post“ jej přenesení do horní části domovské stránky, kde se nachází formulář pro vytvoření příspěvku. Pokud je uživatel nepřihlášený, tlačítko ho přesměruje na stránku s přihlášením.

## 3.2 Nefunkcionální požadavky

Webová aplikace bere v úvahu i nefunkcionální požadavky, které vedou k lepšímu uživatelskému zážitku.

Tabulka 2 Nefunkcionální požadavky stručně

ID požadavku	Popis požadavku pro uživatele
NP1	Webová aplikace je multiplatformní
NP2	Webová aplikace je responzivní
NP3	Ucelený vývojový stack
NP4	Pro správnost webové aplikace se vyžaduje Javascript v prohlížeči

### 3.2.1 Multiplatformní webová aplikace

Webová aplikace podporuje běh na různých platformách a zařízeních, jako jsou desktopové počítače, mobilní telefony nebo tablety.

### 3.2.2 Responzivní webová aplikace

Webová aplikace je responzivní a přizpůsobitelná různým velikostem obrazovek, aby poskytovala optimální a příjemnou uživatelskou zkušenost.

### 3.2.3 Ucelený vývojový stack

Pomocí frameworku SvelteKit je webová aplikace tvořena jako ucelený vývojový stack, který kombinuje jak klientskou, tak serverovou funkcionalitu. Díky tomu je lepší správa aplikace, její škálovatelnost, rychlost vývoje nebo načítání.

### 3.2.4 Použití JavaScriptu

Aplikace vyžaduje podporu pro JavaScript pro správné fungování a interaktivní prvky. Nepřihlášený uživatel bez povoleného JavaScriptu má stále možnost sledovat příspěvky na hlavní stránce.

### 3.3 Případy užití

Případy užití představují scénáře, které popisují interakci mezi uživatelem a webovou aplikací.

Tabulka 3 Případ užití registrace

Název	UC001 - Registrace
Aktéři	Uživatel, webová aplikace
Předpoklady	Uživatel není přihlášený.
Hlavní scénář	1. Uživatel je na stránce a v horním menu klikne na tlačítko Sign in, poté přejde pomocí odkazu na registrační formulář. 2. Uživatel vyplní a odešle registrační formulář. 3. Aplikace ověří údaje a vytvoří nový účet a session. A session id uloží do prohlížeče uživatele.
Alternativní sc.	2a. Uživatel zadá neplatné nebo nesprávné údaje. 2b. Aplikace zobrazí chybovou hlášku.
Výstup	Uživatel úspěšně dodělá registraci a získá nový uživatelský účet. Poté je přesměrován na domovskou stránku už jako přihlášený uživatel.

Tabulka 4 Případ užití přihlášení

Název	UC002 - Přihlášení
Aktéři	Uživatel, webová aplikace
Předpoklady	Uživatel není přihlášený, ale je zaregistrovaný a má přístupové údaje.
Hlavní scénář	1. Uživatel je na stránce a v horním menu klikne na tlačítko Sign in. 2. Uživatel vyplní a odešle přihlašovací formulář. 3. Aplikace ověří údaje a přihlásí uživatele. Vytvoří session a její id uloží do prohlížeče uživatele.
Alternativní sc.	2a. Uživatel zadá neplatné nebo nesprávné údaje. 2b. Aplikace zobrazí chybovou hlášku.
Výstup	Uživatel se úspěšně přihlásí do svého účtu. Je přesměrován na domovskou stránku



Tabulka 5 Příklad užití odhlášení

Název	UC003 - Odhlášení
Aktéři	Uživatel, webová aplikace
Předpoklady	Uživatel je přihlášený. Uživatel má povolený JavaScript v Prohlížeči.
Hlavní scénář	1. Uživatel je na stránce Settings a dívá se na záložku sign out. 2. Uživatel klikne na červené tlačítko Sign out. 3. Uživatel potvrdí, že se chce odhlásit v okně modal pomocí confirm tlačítka. 4. Aplikace odstraní session id z prohlížeče na kterém je uživatel.
Alternativní sc.	3a. Uživatel klikne na tlačítko Cancel a modal se zavře
Výstup	Uživatel je přesměrován na domovskou stránku jako nepřihlášený uživatel.

Tabulka 6 Příklad užití vytvoření příspěvku

Název	UC004 – Vytvoření příspěvku
Aktéři	Uživatel, webová aplikace
Předpoklady	Uživatel je přihlášený.
Hlavní scénář	1. Uživatel je na domovské stránce. 2. Uživatel vyplní a odesílá formulář.  3. Webová aplikace ověří vstupy. 4. Webová aplikace uloží příspěvek do databáze. 5. Webová aplikace zobrazí nový příspěvek na domovské stránce.
Alternativní sc.	3a. Uživatel nesplnil požadavky pro přijetí příspěvku (počet znaků). 3b. Uživateli se zobrazí ve formuláři chybová hláška. 3c. Uživatel upraví špatně zadané údaje a znovu posílá formulář.
Výstup	Nový příspěvek je uložen v databázi a je na hlavní stránce.

Tabulka 7 Příklad užití smazání příspěvku

Název	UC005 – Smazání příspěvku
Akteři	Uživatel, webová aplikace
Předpoklady	Uživatel je přihlášený a už vytvořil aspoň jeden vlastní příspěvek. A má povolený JavaScript
Hlavní scénář	1. Uživatel je na domovské stránce a vidí kartu svého příspěvku, který chce smazat. 2. Uživatel u vybraného příspěvku klikne na tlačítko delete.  3. Uživatel v potvrzovacím modal klikne na tlačítko confirm. 4. Aplikace odstraní příspěvek z databáze.
Alternativní sc.	3a. Uživatel klikne na tlačítko cancel, zavře se mu modal a může znovu kliknout na tlačítko delete pro odstranění příspěvku.
Výstup	Příspěvek je úspěšně smazán z databáze a není vidět na domovské stránce a na profilu uživatele.

Tabulka 8 Příklad odstranění uživatele

Název	UC006 – Odstranění uživatele
Akteři	Uživatel, webová aplikace
Předpoklady	Uživatel je přihlášený ve webové aplikaci a má povolený JavaScript.
Hlavní scénář	1. Uživatel je na stránce Settings v sekci delete account. 2. Uživatel klikne na červené tlačítko Delete my profile. 3. Uživatel klikne v potvrzovacím modalu na tlačítko confirm. 4. Aplikace odstraní příspěvky uživatele následně session id v prohlížeči a potom jeho v databázi. 5. Aplikace přesměruje uživatele na login stránku jakožto nepřihlášeného.
Alternativní sc.	3a. Uživatel klikne na tlačítko cancel a vidí znovu celou stránku v sekci delete my account.
Výstup	Uživatel je na přihlašovací stránce, jeho příspěvky a údaje nejsou v databázi, tím pádem i v aplikaci. Nepůjde se mu přihlásit přihlašovacími údaji jako před smazáním účtu, dokud se znovu nezaregistruje.

Tabulka 9 Příklad užití přesměrování pomocí menu

Název	UC007 – Přesměrování pomocí menu
Aktéři	Uživatel, webová aplikace
Předpoklady	Uživatel je na webové stránce.
Hlavní scénář	1. Uživatel vidí v aplikaci na každé stránce nalevo navigační menu, které obsahuje dva odkazy (Home, About) pro nepřihlášeného uživatele a 4 odkazy pro přihlášeného (navíc je Profile a Settings). 2. Uživatel klikne na libovolnou stránku nabízenou v postranním menu. 4. Aplikace přesměruje uživatele, pokud se jedná o stránku vyžadující autorizaci, zkontroluje, zda je uživatel přihlášen a pak ho přesměruje.
Alternativní sc.	1a. Uživatel pro zobrazení musí kliknout na zabalené navigační menu v horní levé části stránky. 1b. Aplikace vysune z levé strany navigační menu. 2a. Uživatel klikne mimo vysunuté menu. 2b. Aplikace zabalí menu
Výstup	Uživatel je přesměrován na požadovanou stránku s původními pravomoci.

Tabulka 10 Příklad užití změna světelného režimu

Název	UC008 – Změna světelného režimu
Aktéři	Uživatel, webová aplikace
Předpoklady	Povolený javascript v prohlížeči.
Hlavní scénář	1. Uživatel klikne na tlačítko v horní liště webové aplikace. 2. Aplikace nastaví světelný režim.
Alternativní sc.	
Výstup	Uživateli se změní barva prostředí v celé webové aplikaci.

Tabulka 11 Příklad užití tlačítko pro posun na stránce

Název	UC009 – Tlačítko pro posunutí na stránce
Aktéři	Uživatel, webová aplikace
Předpoklady	Uživatel je na hlavní domovské stránce Home a má povolený JavaScript.
Hlavní scénář	1. Uživatel klikne na tlačítko create post 2. Uživatel je přesunut do horní části k formuláři.
Alternativní sc.	1a. Uživatel není přihlášen a kliknutí na tlačítko create post ho přesměruje na stránku login k přihlašovacímu formuláři.
Výstup	Uživatel je posunut k části, kde se nachází formulář pro vytvoření příspěvku.

### 3.4 Databázový model

Pro práci s databázovým modelem je využíván moderní ORM (Object Relational Mapping) nástroj jménem Prisma. Díky objektově relačnímu mapování je zajištěna automatická konverze dat mezi relační databází a objektově orientovaným jazykem. Proto lze jednoduše vytvořit a upravovat datové modely v souboru *schema.prisma*. Projekt využívá SQLite databázi, přičemž databázový soubor je v lokálním umístění *dev.db*. Schéma obsahuje popisuje čtyři databázové modely. První model *AuthUser* reprezentuje autentizovaného uživatele a obsahuje pole pro unikátní *id* a *username*, dále *auth\_session*, *auth\_key* a *auth\_post*, která odkazují na další modely. Model *AuthSession* slouží k reprezentaci autentizační relace uživatele v rámci webové aplikace. Představuje informace o aktivní relaci, která je navázána na konkrétního uživatele. To se dá využít u implementace funkce pro přihlašování, odhlašování a podobně. Atributy v tomto modelu jsou unikátní *id*, dále *user\_id*, *auth\_user*, který je vazbou na model *AuthUser*. Nebo také atributy sloužící jako časová razítka *active\_expires* a *idle\_expires*. Tyto razítka udávají platnost relace. Třetí popsáný model v souboru *schema.prisma* je *AuthKey*. Tento model pomáhá reprezentovat autentizační klíč uživatele v rámci aplikace, který se využívá například u přihlašování. Mezi atributy patří jednoznačný identifikátor autentizačního klíče jménem *id*, hashované heslo uživatele *hashed\_password*, identifikátor uživatele, kterému autentizační klíč patří pod názvem *user\_id*, dále je *primary\_key*, který nabývá hodnot pravda/nepravda, podle toho zda je klíč primární, časové razítko *expires* a nakonec atribut *auth\_user* s vazbou na *AuthUser*. Poslední model je *AuthPost*, díky němuž lze vytvářet a mazat příspěvky od uživatele. Obsahuje pole *id*, které je opět unikátní, název příspěvku *title*, obsah příspěvku *content*, *create\_at* nabývací výchozí hodnotu aktuálního času s datem při vytvoření, nebo *author\_id* a *author\_username* a nakonec *author*, což je vazba na model *AuthUser*, který reprezentuje autentizovaného uživatele.

```
1 generator client {
2   provider = "prisma-client-js"
3 }
4
5 datasource db {
6   provider = "sqlite"
7   url      = "file:./dev.db"
8 }
9
10 model AuthUser {
11   id          String    @id @unique
12   auth_session AuthSession[]
13   auth_key    AuthKey[]
14   username    String    @unique
15   auth_post   AuthPost[]
16
17   @@map("auth_user")
18 }
19
20 model AuthSession {
21   id          String    @id @unique
22   user_id     String
23   active_expires BigInt
24   idle_expires BigInt
25   auth_user   AuthUser  @relation(references: [id], fields: [user_id], onDelete:
    Cascade)
26
27   @@index([user_id])
28   @@map("auth_session")
29 }
30
31 model AuthKey {
32   id          String    @id @unique
33   hashed_password String?
34   user_id     String
35   primary_key Boolean
36   expires     BigInt?
37   auth_user   AuthUser  @relation(references: [id], fields: [user_id], onDelete:
    Cascade)
38
39   @@index([user_id])
40   @@map("auth_key")
41 }
42
43 model AuthPost {
44   id          Int        @id @default(autoincrement())
45   title       String
46   content     String
47   created_at  DateTime   @default(now())
48   author_id   String
49   author_username String
50   author      AuthUser   @relation(references: [id], fields: [author_id])
51
52   @@map("auth_post")
53 }
```

Obrázek 15 Prisma schéma s modely

## 4 IMPLEMENTACE APLIKACE

Praktická kapitola implementace aplikace je důležitou součástí této bakalářské práce. Obsahem je postup od základního vytváření projektu až po jeho finální podobu s popisem a řešením problematiky, která se během vývoje vyskytuje.

### 4.1 Příprava prostředí pro vývoj

Aplikaci lze vyvíjet v různém vývojovém prostředí s různými nástroji, na různých operačních systémech.

#### 4.1.1 Vývojové prostředí

Pracovat na projektu s frameworkem SvelteKit lze v mnoha vývojových prostředí (*Integrated Development Environment - IDE*). Na stránkách v oficiální dokumentaci je doporučen Visual Studio Code od firmy Microsoft, který lze zdarma stáhnout a používat bez jakýchkoliv omezení.

#### 4.1.2 Instalace SvelteKit

Pro spuštění a zpracovávání serverových funkcí potřebuje SvelteKit poskytnout prostředí od *Node.js*. Jedná se o asynchronní JavaScriptový runtime, který poskytuje prostředí pro běh JavaScriptu na straně serveru. Obsahuje také správce balíčků *Node Package Manager (npm)*, který slouží k instalaci a správě balíčků, které v tomto případě představují framework. Dále *Node.js* poskytuje nástroj *Vite*, který je používán jako vývojový server a překladač *Svelte* komponent do JavaScriptu.

#### 4.1.3 Vytvoření nového projektu

Po nainstalování *Node.js* otevřeme příkazový řádek ve složce, ve které budeme chtít mít náš projekt. Do terminálu bychom normálně napsali pouze: „*npm create svelte@latest [název\_projektu]*“, ale protože náš projekt bude využívat knihovny *Skeleton UI*, napíšeme příkaz: „*npm create skeleton-app@latest [název\_projektu]*“, díky tomu nám *Skeleton CLI* umožní si vybrat výchozí šablonu projektu, zvolit možnost type checkingu pomocí *TypeScript*, kterou využijeme, a nebo přidání *Prettier* pro formátování kódu. Tím pádem vzniká rychlý a snadný start s přednastavenou konfigurací. Poté se v příkazovém řádku v nově vytvořeném projektu spustí příkaz *npm install*, který nainstaluje všechny závislosti v *package.json* potřebné pro běh a vývoj aplikace. Posledním krokem je spuštění vývojového serveru, který se nám po jakékoli uložené změně v projektu sám automaticky

sestaví a obnoví webovou stránku. Stránku lze sledovat ve webovém prohlížeči na adrese: „localhost:[ČÍSLO\_PORTU]“. Tento krok udělá příkaz: „npm run dev“.

## 4.2 Adresářová struktura

Vzhledem k tomu, že SvelteKit je založený na file-based routeringu, je o to víc důležitá struktura složek a souboru v projektu.

### 4.2.1 Počáteční struktura projektu

Po vytvoření projektu vzniká složka s námi zadaným názvem. V této složce jsou pak i další. Všechny jsou vytvořeny automaticky po vytvoření projektu. Jako první lze popsat *./svelte-kit* složku, která obsahuje soubory a konfigurace související se SvelteKit frameworkem. Složka pod ní je */node\_modules*. Ta v sobě má závislosti a moduly, které jsou nainstalovány pomocí nástroje *npm* (Node Package Manager). Tyto závislosti jsou potřebné pro správný chod projektu a jsou specifikovány ve složce projektu pod souborem *package.json*. Další složka po boku dvou předchozích je *static*. Tato složka slouží pro uložení statických souborů, jako jsou obrázky, ikony a další soubory, které nevyžadují zpracování nebo kompilaci. Čtvrtá složka jménem */src*, je nejspíše nejdůležitější pro vývoj webové aplikace pomocí frameworku SvelteKit. V ní se nachází zdrojový kód webové aplikace. Zdrojový kód představují různé soubory, jako jsou komponenty, styly, konfigurace a další soubory, které jsou součástí implementace. Hlavně se zde vyskytuje ještě další složka, do které vývojář zasahuje nejspíše nejvíce. Je to složka */src/routes* a už podle názvu lze usoudit, že obsahuje složky, soubory a konfigurace související s jednotlivými stránkami aplikace. Každá nová složka v ní představuje novou stránku ve webové aplikaci a její název je zároveň její adresou v prohlížeči. Tyto složky v sobě mají opět vlastní komponenty a konfigurace. Zároveň stránka dědí ze souborů, které jsou přímo ve složce */src/routes*. Například soubor */src/routes/+layout.server.ts* má v sobě exportovanou *load* funkci, která něco vrací. K tomu, co vrátí má přístup jak root stránka v */src/routes*, tak všechny další stránky ve složce */src/routes/\**. Pro vývojáře je k dispozici i složka */src/lib* pro jeho vlastní pomocné komponenty, knihovny, konstanty a další pomocné nebo často opakující se soubory používané ve webové aplikaci. Tato složka je dostupná globálně v projektu. Abychom si detailněji popsali soubory ve složce */src*, můžeme zmínit a popsat soubor */src/app.d.ts*. Jak koncovka souboru naznačuje, jedná se o globální definice typů a rozhraní pro webovou aplikaci. Dalším globálním nastavením v projektu je i soubor */src/app.postcss*, ve kterém lze nastavit globální styly webové aplikace. Poslední soubor, který je ve složce */src*, je

/src/app.html. Jedná se o šablonu stránky, o HTML dokument, který obsahuje několik důležitých značkovacích míst pro správné fungování aplikace. Ještě je důležité zmínit dva soubory v hlavní složce projektu, a to sice [název\_projektu]/*package.json*, který již byl zmíněn. Ten obsahuje potřebné balíčky a repozitáře, které se mají nainstalovat. Druhý soubor je */config.svelte.ts*, díky kterému může vývojář nakonfigurovat různé chování a nastavení webové aplikace. Může například nastavit předzpracování různých souborů, bezpečnostní nastavení nebo middleware funkce, které slouží pro vlastní manipulaci s požadavky a odpověďmi na straně serveru. [16]

#### 4.2.2 Příprava struktury projektu

Po nainstalování potřebných balíčků pro správný chod webové aplikace pomocí příkazu *npm install*, si připravíme strukturu projektu. Přidáme složky a soubory, do kterých zatím nebudeme vkládat žádné konfigurace. Díky tomu se budeme blížit finální struktuře projektu. Jako první krok začneme tím, že do projektu přidáme prázdnou složku */prisma*, do té později budeme vkládat soubor *schema.prisma* pro vytvoření modelů v databázi *SQLite*, která se vytvoří v této složce také. Nyní se posuneme do složky */src*. Nejdříve do ní přidáme soubor */src/hooks.server.ts*, který bude později manipulovat s požadavky na serverové straně a zajišťovat autentizaci údajů před zpracováním požadavků. To by bylo, co se týče souborů ve složce */src* všechno, proto se přesuneme na vytváření složek. A začneme vytvářením globální sdílejší složky */src/lib*. Zde vytvoříme složku */lib/components* a *lib/server*. Složku pro úschovu komponentů zatím necháme prázdnou a v průběhu vývoje do ní přidáme soubory. Ve složce */lib/server* vytvoříme soubor *lucia.ts*, kde bude později probíhat inicializace knihovny Lucia a její exportování.

### 4.3 Konfigurace projektu

Cílem konfigurace v projektu je nastavení a propojení různých komponent webové aplikace, tak, aby spolupracovaly správně. Do toho spadá definování parametrů, cest, adaptérů a dalších potřebných informací, které umožňují komponentám pracovat společně a poskytovat požadovanou funkcionalitu. V případě naší webové aplikace bude potřeba konfigurovat Lucia, Prisma, Zod validation.

#### 4.3.1 Konfigurace knihovny Lucia

Na začátek je důležité si knihovnu stáhnout pomocí příkazu *npm install lucia-auth*. A potom stáhnout prisma adaptér příkazem *npm install @lucia-auth/adapters-prisma*. V dokumentaci



na webových stránkách Lucia je popsán postup inicializace. Do již námi vytvořeného souboru `/lib/lucia.ts` vložíme ukázkový kód z jejich návodu. A nad něj inicializujeme nový prisma client. Inicializace Lucia Auth v kódu probíhá pomocí funkce `lucia()`, která inicializuje a konfiguruje její instanci. Tato funkce zároveň definuje tři vlastnosti. Jako první konfiguruje adaptér pro databázi. V našem případě je to adaptér Prisma s vytvořenou instancí na začátku souboru. Díky tomu, že jsme ve složce `/lib`, můžeme tohoto prisma klienta globálně exportovat. Dále se určuje vlastnost prostředí, ve kterém aplikace běží. Pokud je webová aplikace spuštěna v režimu vývoje, nastaví se prostředí na `DEV`. V opačném případě se nastaví `PROD`. Poslední vlastnost, kterou kód z dokumentace ve funkci nastavuje je middleware, který říká, že se pro integraci Lucia Auth do SvelteKit použije `sveltekit()`. Kód z dokumentace si ještě rozšíříme ve funkci `lucia()` o transformační funkci `transformDatabaseUser()`, která slouží k transformaci dat uživatele získaných z databáze do požadovaného formátu. V tomto případě je přeměna `id` a `username` z `userData` na vlastnosti `userId` a `username`. Po inicializaci a konfiguraci Lucia Auth ji musíme ještě exportovat. Vytvořenou instanci nabývá proměnná `auth`, která se exportuje a díky tomu je možné v dalším kódu přistupovat k autentizačním funkcím a metodám. Důležitou částí je ještě exportování typu `Auth`, který je odvozen od `typeof auth`, což poskytuje informace o typu této proměnné. Tohoto modulu `auth` využijeme při nastavování serverových hooků v souboru `/src/hooks.server.ts`. Opět můžeme zkopírovat kód z dokumentace Lucia, zkopírujeme serverový hook `handle`, což je funkce, která přebírá objekt s parametry `event` a `resolve`. Parametr `event` v sobě nese informaci o aktuálním požadavku a `resolve` slouží jako funkce pro pokračování ve zpracování požadavku. Tělo funkce obsahuje přidání instance `auth` ze souboru `/lib/lucia.ts` do objektu `event.locals.auth`, pomocí funkce `handleRequest()`. Tato funkce vrací objekt `AuthRequest`, který poskytuje sadu metod, které usnadňují validaci příchozích požadavků. Díky tomuto postupu máme možnost pracovat s autentizací a autorizací uživatelů v aplikaci. Posledním důležitým krokem pro správnou integraci Lucia je vložení potřebných typů do projektu. To usnadňuje práci s autentizací a umožňuje to našemu vývojovému prostředí lépe poskytovat podporu a detekování chyb při psaní kódu. Tohoto kroku docílíme tím, že do souboru `/src/app.d.ts` vložíme kód z dokumentace na stránkách Lucia. Přidaný kód rozšiřuje původní dokument v sekci `App` namespace o rozhraní `Locals`, ve kterém je přidán `auth` z modulu `lucia-auth`. Tímto způsobem se zajišťuje, že autentizační informace jsou dostupné v rámci kontextu serverových funkcí a dalších částí aplikace. Dále do souboru `/src/app.d.ts` přidáme namespace Lucia, ve kterém je definován typ `Auth` a `UserAttributes`, do kterého napíšeme `username: string`; . Na konec souboru se

ještě musí přidat velice důležitá část, a to sice export složené závorky `{}`, neboli export prázdného objektu, který zajistí, že soubor bude brán jako modul. [17]

### 4.3.2 Konfigurace databázového nástroje Prisma

Jelikož jsme si jako adaptér zvolili Prisma, podnikneme další kroky pro konfiguraci naší webové aplikace. Pokud vývojář již provedl krok instalace balíčku pomocí `npm` i `@lucia-auth/adapter-prisma`, může ve složce `/prisma`, kterou jsme dříve vytvořili, udělat nový soubor `/prisma/schema.prisma`. Do něho může vývojář zapisovat schéma pro svou databázi. Vzhledem k tomu, že jsme soubor se schématem nevytvořili pomocí příkazu, musíme do něj ještě přidat generátor, který vytvoří Prisma klienta pro přístup k databázi. Dále specifikujeme poskytovatele databáze, kterým bude SQLite a adresu databáze, která bude v aktuálním adresáři pod názvem `/dev.db`. Jako další věc si zkopírujeme modely `AuthUser`, `AuthSession` a `AuthKey`, které jsou už předepsané na webových stránkách Lucia a vložíme je do schématu. K nim přidáme model `AuthPost`, který je popsán v části této bakalářské práce, která se věnuje databázovému modelu. V příkazovém řádku spustíme příkaz `npx prisma generate`, ten nám vygeneruje TypeScript definice, které zajišťují, že webová aplikace používá správné datové typy při komunikaci s databází, což zvyšuje bezpečnost, spolehlivost a produktivitu při vývoji webové aplikace. Poté spustíme další příkaz, který nám vytvoří `.db` soubor dle specifikace v `/prisma/schema.prisma`. Je to příkaz `npx prisma db push` a se specifikacemi výše zmíněnými, nám vytvoří soubor `/prisma/dev.db` a nastaví strukturu databáze tak, aby odpovídala definici ve schématu. Při každé změně v `/prisma/schema.prisma` je důležité tento příkaz znovu spustit a tím pádem vytvořit novou migraci databáze. Po tomhle příkazu následuje další: „`npx prisma generate`“. Ten slouží k vytvoření Prisma klienta, který je definován v souboru `/prisma/schema.prisma` a poskytuje rozhraní pro komunikaci s databází. [18]

### 4.3.3 Ostatní knihovny

Vzhledem k tomu, že jsme na začátku při vytváření projektu využili Skeleton CLI, nemusíme psát žádné další manuální příkazy. Můžeme napříč projektu využívat výhod, které nám tento nástroj umožňuje. Pomocí příkazu `npm install zod` si dopředu stáhneme validační knihovnu Zod pro validaci dat od uživatelů.

## 4.4 Zabezpečení webové aplikace

Dnes je bezpečnost klíčovou součástí vývoje webových aplikací a vývojáři musí této oblasti věnovat zvláštní pozornost. Nedostatečné zabezpečení může i vážně ohrozit funkčnost webové stránky a přinést finanční nebezpečí jak uživatelům, tak majitelům. Stávají se i úniky citlivých dat, krádeže identit nebo ohrožení na zdraví. Tato kapitola se věnuje popisu některých hrozeb a jejich následnému zabránění v rámci možností frameworku SvelteKit.

### 4.4.1 Cross-Site-Scripting (XSS)

Je typ kybernetického útoku, který zneužívá nedostatečně ošetřeného vstupu od uživatelů na straně serveru. Zneužití spočívá v tom, že útočník na webové stránce vloží a spustí nebezpečný kód, většinou typu JavaScript, a díky tomu může napadnout a ovládat celou stránku, což mu umožňuje provádět nebezpečné akce přímo v prohlížeči uživatele. SvelteKit a Svelte nabízejí možnosti, jak tyto útoky minimalizovat.

#### 4.4.1.1 Ochranný Svelte mechanismus

Svelte zajišťuje ochranu pomocí bezpečného zpracování dat poskytnutých od uživatele. Framework tyto data buďto escapuje, což znamená, že převede speciální znaky na jejich bezpečnou textovou reprezentaci, nebo provádí sanitaci, která spočívá v mazání potencionálně nebezpečných částí dat, kterými jsou například scripty. [19]

#### 4.4.1.2 Nastavení Content-Security-Policy (CSP)

Ochrana před útoky Cross-Site-Scripting může být i nastavení hlaviček u HTTP odpovědí, které jsou zasílány klientovi. Bezpečnostní mechanismus Content-Security-Policy umožňuje vývojáři webové stránky definovat, jaké zdroje mohou být na stránce načítány. Zdroji se rozumí například scripty, styly nebo obrázky. Framework SvelteKit nabízí vývojářům možnost, jak tento bezpečnostní mechanismus přidat do svého projektu. Konfigurace se řeší v souboru *svelte.config.js* vnořením objektu *csp* do objektu *kit*. [19] [20]

#### 4.4.1.3 Autentizační knihovna Lucia

Pro jednoduchou správu uživatelů a jejich relací lze zdarma využít Lucia knihovny. Ta vývojáři poskytuje jednoduchou implementaci některých osvědčených bezpečnostních postupů. Jedním z nich je využívání relací (sessions), díky kterým lze validovat a sledovat uživatele. Při přihlášení Lucia uloží do cookies novou session id, která je uložena i v databázi v tabulce *auth\_session*. V databázi je k *id* relace ještě na rozdíl od uložení v cookies přidáno

políčko pro *user\_id*. Díky tomu lze bezpečně ověřovat uživatele po dobu jeho přihlášení. Což nás vede ke stavu, kdy se uživatel z webové aplikace odhlásí, nebo mu vyprší časová relace. U tohoto nastává nebezpečí, kdy jsou data právě jako *session\_id* stále uložena i po odhlášení v prohlížeči. Toho by mohl jednoduše zneužít každý, kdo by měl přístup k zařízení, například ke školnímu počítači. Proto je důležité zajistit efektivní odhlášení a tato data z prohlížeče smazat. Lucia to dělá tak, že nejprve pomocí *invalidateSession()* zruší relaci s uživatelem a potom příkazem *setSession(null)* smaže všechny session cookies, čímž zmizí hodnota *session\_id*, která po opětovném přihlášení bude navíc úplně jiná. Do locals je společně s relací ukládán i *event*, pomocí kterého Lucia umožňuje snadný a konzistentní přístup k *RequestEvent*, bez nutnosti jej opakovaně předávat mezi různými částmi kódu. To usnadňuje serverovou validaci pro příchozí požadavky, například v *load* funkci, která má parametr *locals*. [19] [21] [22]

#### 4.4.1.4 Validace uživatelského vstupu pomocí Zod

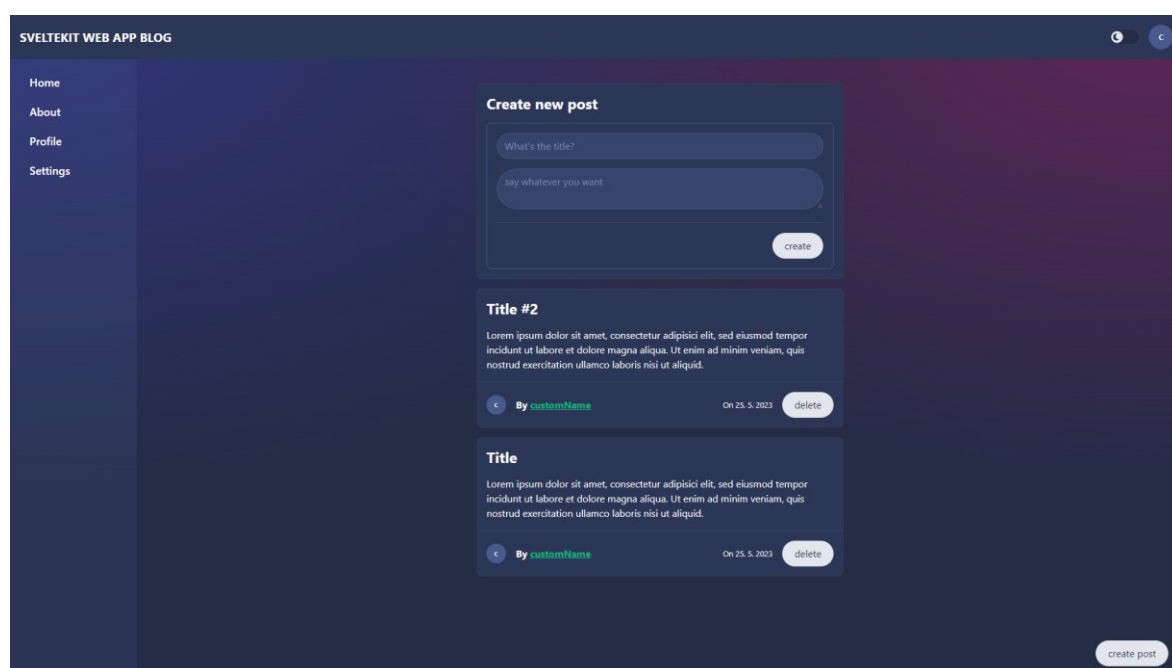
Webová aplikace na straně serveru ověřuje vstupní data od uživatele pomocí knihovny Zod. Jedná se o nástroj zdarma, který je užitečný pro zajištění konzistence a správnosti dat. Pomáhá snižovat chybovost a zabezpečuje, že data splňují očekávané požadavky. Ty lze nastavit pomocí validačního schéma, které si vývojář nastaví podle potřeb tak, aby popisovaly očekávanou strukturu a typ dat. Schémata mohou definovat různá pravidla, jako jsou povinná pole, omezení délky vstupů nebo jejich typ. Zod po definování validačního schématu využije funkci, která porovná data od uživatele s požadovaným schématem. Pokud jsou data neplatná, knihovna Zod poskytne informace o chybách, které může vývojář vrátit uživateli přímo do prohlížeče na webovou stránku. [23]

## 4.5 Popis demonstrační webové aplikace

Kapitola popisuje postup vytváření demonstrační webové aplikace podle zadaných požadavků.

### 4.5.1 Root stránka Home

Jako první začneme tvorbou domovské stránky, která bude obsahovat všechny příspěvky uživatelů. Příspěvek bude moci vytvořit přihlášený uživatel a zároveň ho může i smazat. Stránka bude jinak vypadat a fungovat pro přihlášeného a nepřihlášeného uživatele. Logika této stránky se řeší přímo ve složce `/src/routes`.



Obrázek 16 Ukázka domovské stránky

#### 4.5.1.1 Layout pro webovou stránku

Vytvořením souboru `+layout.svelte` můžeme aplikovat šablonu pro celou webovou stránku. V případě této webové aplikace jsem funkci souboru využil k postrannímu navigačnímu menu a k hlavičce webové stránky, která bude aplikovaná i na ostatní routes. Pro implementaci navigačního menu a rozložení stránky využijeme UI prvků z knihovny Skeleton. První prvek bude *AppShell*, což je responzivní ovládání pro rozložení stránky. *AppShell* může obsahovat klidně všechny sekce (*header*, *sidebarLeft*, *sidebarRight*, *footer*, *pageHeader*, *pageFooter*) a dokud do nich něco nevložíme, na stránce se to neprojeví. My budeme využívat *header* a *sidebarLeft* pro horní lištu a postranní navigaci. Do fragmentu

pro slot *header* vložíme header element, který je určený pro layout horní lišty *AppBar*. Obsah headeru je nastavený tak, že obsahuje název webové stránky, tlačítko pro změnu světelného režimu a přihlašovací tlačítko nebo ikonu s počátečním písmenkem jména uživatele. To se mění v souladu se stavem uživatele stránky (přihlášený/nepřihlášený). Potom je pomocí Tailwind stylování nastaveno, že menší zařízení budou mít v horní liště ještě rozbalovací menu, které při kliknutí spustí další komponentu *Drawer* z knihovny *Skeleton UI*, která zobrazí překryvný panel. V tomto panelu bude námi vytvořená komponenta *<Navigation>*, představující obsah postranního menu. Tuto komponentu přidáme také do *AppShell* slotu pro *sidebarLeft*. Protože komponenta se bude měnit podle toho, jestli je uživatel přihlášený, přidáme jí do property exportovanou proměnnou *data*, která bude obsahovat status uživatele. Tato šablona layout určuje obsah stránky, na kterou se šablona aplikuje pomocí elementu *<slot />*. Já ho mám vložený mezi nevyužitým *pageHeader* a *pageFooter* slotem. Jak bylo zmíněno, layout vypadá jinak podle stavu uživatele. Aby to soubor *+layout.svelte* věděl, musíme vytvořit nový soubor *+layout.server.ts*, který bude obsahovat funkci *load*, ve které se zjišťuje pomocí session cookies, zda-li je uživatel přihlášen. Objekt uživatel se potom promítne v exportované proměnné *data*, která je k dispozici dalším stránkám webové aplikace, protože se nachází přímo v root složce */routes*.

#### 4.5.1.2 Hlavní obsah domovské stránky

Pro vizualizaci samotné domovské stránky využijeme soubor */+page.svelte* ve složce */src*. Stránka opět mění funkčnost a vzhled v závislosti na typu uživatele. Pokud se jedná o nepřihlášeného uživatele, uvidí příspěvky a fixní tlačítko *create post*, které ho po kliknutí přesměruje na stránku s přihlášením. Tato stránka ještě poskytuje možnost přesměrování na profil autora, který vlastní příspěvek. To nastane pokud dojde ke kliknutí na jeho přezdívku. V případě přihlášeného uživatele tlačítko *create post* posune uživatele k formuláři pro vytvoření nového příspěvku. Tento formulář provádí akci *createPost*, kterou později popíšeme v souboru *+page.server.ts*. Podobně jako u layoutu, soubor *+page.svelte* potřebuje ke svému zobrazení načíst nějaká data. Místo uživatelů se ale načítají všechny příspěvky z databáze a jejich pořadí se ještě otáčí, aby nejnovější příspěvek byl vždycky nahoře. Jelikož v tomto souboru manipulujeme s databází, na jeho začátku importujeme instanci *prisma* klientu ze souboru */lib/server/lucia.ts*. Objekt *data* nyní v souboru *+page.svelte* obsahuje uživatele a seřazené příspěvky. *Svelte* nabízí možnost reaktivního přiřazení pomocí znaků „*\$:*“. V našem projektu se do objektu *post* přidávají příspěvky z objektu *data*. Protože využíváme reaktivní označení, při aktualizaci objektu *data*, se objekt

*post* aktualizuje sám automaticky a uživatel uvidí změnu. Co se týče formulářů, tak jsou na této webové stránce dva. Prvním je formulář pro vytváření příspěvku, který využívá JavaScriptového zlepšení *use:enhance*. Při odeslání formuláře se provádí v souboru *+page.server.ts* akce *createPost*. Obsahuje dvě vstupní pole, do kterých může uživatel zadat vstup. Tyto pole mají svůj atribut *name*, díky kterým lze odlišit vstupy a uživateli vracet chybová hlášení.

Akce, které může stránka vykonávat, jsou definované v souboru *+page.server.ts* pod funkcí *load*. Než zde popíšu *Actions*, je ještě důležitá část, ve které se definuje schéma pro validaci vstupů od uživatele pomocí knihovny *Zod*. Ve schématu jsem nastavil následující požadavky. Nadpis příspěvku musí být vyplněn, musí obsahovat minimálně 3 znaky a nesmí přesáhnout délku 12 znaků. K tomu všemu se z tohoto stringu od uživatele oddělávají zbytečné bílé mezery pomocí metody *.trim()*. Další požadavky jsou definovány pro vstup *content*, neboli tělo příspěvku. To musí být opět vyplněno, musí mít minimální velikost 2 znaky a maximálně 230 znaků. Při nedodržení některého z požadavků je vytvořena vlastní chybová hláška, kterou budeme v *Action createPost* vracet uživateli.

Protože akce nebude jenom jedna, nevytváříme defaultní *Action*, ale pojmenovanou. Akce *createPost* pracuje s objektem *request* a *locals*. Z *request* získáme zadaná data od uživatele a pomocí *locals* ověříme na začátku, zdali se jedná o přihlášeného uživatele. Jako další se akce pokusí o vytvoření příspěvku. Může zde nastat varianta, kdy neprojde *Zod* validace, nebo se do databáze příspěvek úspěšně nahraje. V případě známe chyby instance *ZodError*, objekt *form* v *+page.svelte* změní stav *Error* a toho využijeme u zobrazování chybové hlášky, která se zobrazí pokud se splní podmínka výskytu chyby ve *form*. Zároveň má chyba svůj název společně s vlastní chybovou hláškou a ta se přiřazuje ke konkrétnímu poli. Je-li příspěvek úspěšně nahrán do databáze, uživatel pocítí výhodu *use:enhance* a na stránce se mu zobrazí jeho příspěvek bez jakéhokoliv znovu načtení stránky.

Do druhého formuláře na stránce uživatel nic nezadá, na stránce není zobrazen, je totiž v těle tlačítka, které také využívá *Actions*, tentokrát akci *deletePost*. Když uživatel klikne na tlačítko *delete*, otevře se mu modal okno s potvrzením. Tento modal jsme importovali z knihovny *Skeleton UI* a funguje následovně. Do kořenového layoutu přidáme komponent *<Modal />*, dále tlačítko *delete* bude spouštět funkci, která modal okno otevře. Pokud uživatel potvrdí odstranění, odešle se formulář s hodnotami příspěvku, se kterými se pracuje na serveru. Tam akce *deletePost* na začátku, obdobně jako *createPost*, ověřuje uživatele.

Potom zkoumá, zda uživatel vlastní příspěvek, který chce smazat a pokud ne, vyhodí *error*. V dalším kroku se odstraňuje příspěvek podle jeho *id* v databázi.

Poslední věcí na stránce, která stojí za to zmínit, je fixní *create post* tlačítko. To znamená, že pozice tlačítka je stále v pravém dolním rohu. Při kliknutí na tlačítko se spustí funkce, která přesune uživatele do horní části domovské stránky, přesněji k formuláři pro vytváření příspěvků. Pokud uživatel není přihlášený, je přesměrován na stránku s přihlášením.

Pro očekávané i neočekávané chyby, které nastanou v sekci root složky, můžeme nastavit a nastylovat vlastní chybovou stránku. Stačí do projektu přidat soubor *+error.svelte* a ten si nastavit podle potřeb. V případě, že nastane očekávaná chyba, kdy uživatel zadá do URL adresy neexistující stránku, můžeme tuto chybu zachytit a uživateli vypsat vlastní chybovou hlášku pro určitou konkrétní chybu.

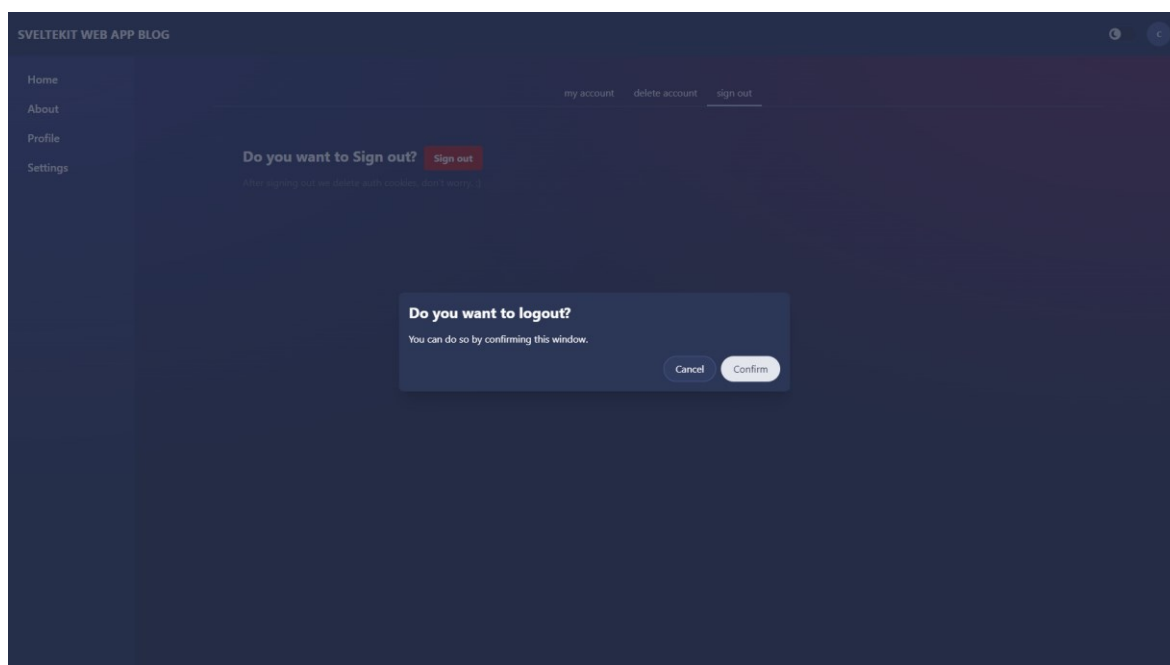
#### 4.5.2 Veřejná stránka profilu uživatele

Tato stránka se bude v navigačním layout menu zobrazovat pouze přihlášeným uživatelům. Ti ale budou moci navštěvovat cizí profily přihlášených uživatelů. Pomocí nastavení složky jako *slug* bude její obsah vytvářen dynamicky, podle parametru z URL adresy. Nejprve ale vytvoříme složku */routes/profile*, ve které bude *load* funkce sloužící k přesměrování přihlášeného uživatele na jeho profil. Pokud nepřihlášený uživatel zadá adresu */profile* bez parametru, jak si za chvíli ukážeme, bude přesměrován na domovskou stránku. Teď vytvoříme onu *slug* složku a pojmenujeme jí v hranatých závorkách */profile/[usernameProfile]*. To nám umožní pracovat s parametrem předaným v URL adrese a podle toho načítat data nebo vykreslovat stránku. Smyslem této stránky je ukázat pouze příspěvky od jednoho uživatele, proto do *slug* složky přidáme vlastní *+page.server.ts* soubor, ve kterém prvně načteme uživatele podle přezdívky z databáze a potom jeho příspěvky. Podobně jako u domovské stránky ještě zachytíme očekávanou chybu. Do *load* funkce dáme podmínku, která při nenalezení uživatele v databázi vyhodí chybu. Příkaz pro vyhození chyby bude obsahovat stavový kód a chybovou hlášku, která se bude měnit podle zadaného parametru v adrese. Protože jsme nastavili už v *load* chybovou hlášku, nově vytvořený *+error.svelte* soubor ve *slug* složce */[usernameProfile]* zůstane s defaultním výpisem chyby.



### 4.5.3 Sekce nastavení

Stránka nastavení uživatele poskytne skupinu záložek, které budou představovat informaci o účtu, možnost si trvale smazat účet nebo se odhlásit. Přičemž potvrzování těchto akcí se bude provádět skrze modal okno. Vytvoříme novou složku `/settings` a v ní soubor `+page.svelte` a `+page.server.js`. Pro rychlé přepínání mezi různými volbami využijeme další komponentu z knihovny Skeleton UI. Stránka se bude skládat z komponenty `<TabGroup>`, která v jejím záhlaví obsahuje jednotlivé záložky `<Tab>`. Ty v této sekci vytvářejí navigaci. Při kliknutí na záložku se změní hodnota proměnné `tabSet`, podle které se zobrazuje kontent vybrané záložky. Pro odhlášení a smazání je obdobná implementace jako u tlačítka pro odstranění příspěvku v domovské stránce. Kliknutí na tlačítko spustí modal okno a to po potvrzení odešle formulář. V `+page.server.ts` se na začátku ve funkci `load` ověřuje, zda je uživatel přihlášen, jestli není, je přesměrován na přihlašovací stránku. Potom záleží, kterou `action` formulář provádí. Při odhlášení zkontroluje relaci, potom ji zneplatní. Následuje odstranění cookies nesoucí hodnotu o bývalé relaci. U volby trvalého odstranění účtu `action deleteUser` provede autorizaci. Následně, pomocí instance klienta Prisma odstraní všechny příspěvky, kde je uživatel autorem a pak i samotného uživatele. Na konci je přesměrování na domovskou stránku.



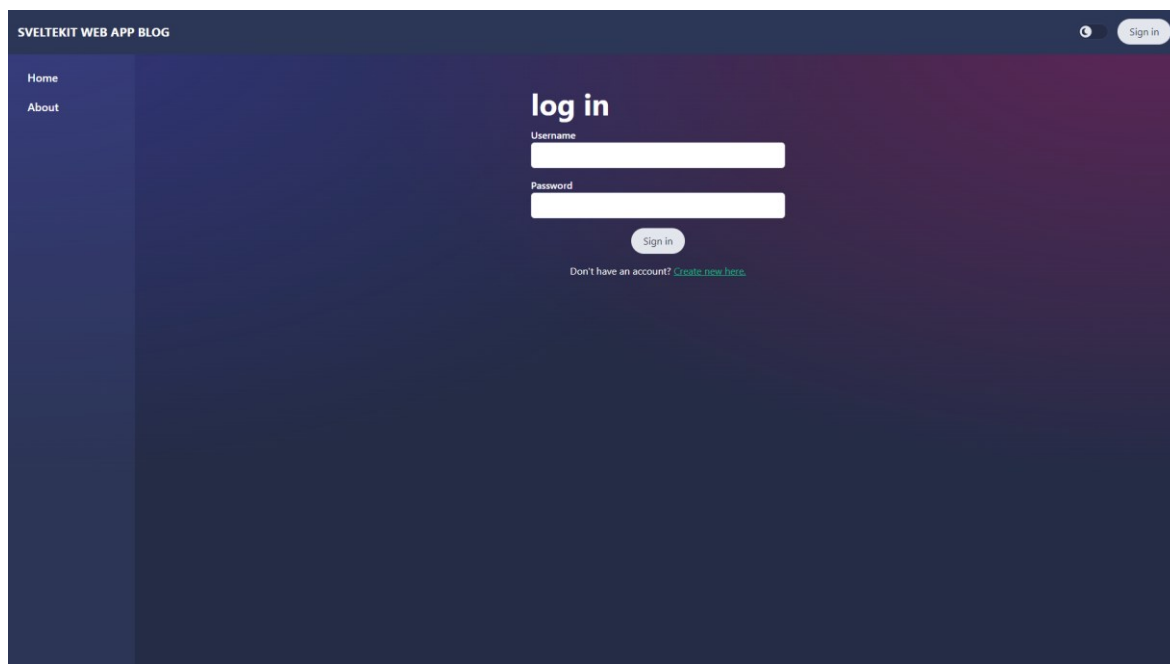
Obrázek 17 Potvrzovací modal při odhlášení

#### 4.5.4 Stránka pro registrování

Pokud si chce uživatel vytvořit nový profil, tak na této stránce vyplní formulář, který ho případně upozorní na špatně zadané vstupy. Logika je v demonstrační webové aplikaci řešena ve složce `/signup` a obsahuje `+page.svelte`, `+page.server.ts` a `+error.svelte`. Přesuneme se rovnou k popisu jediné defaultní akce, kterou spouští odeslání formuláře. Ta na začátku do objektu `form` vkládá vstupní data od uživatele. Následně je validuje pomocí našeho předem definovaného schéma. To je skoro podobné jako `createPostSchema`, akorát u nastavení přezdívky nepovoluje mezeru. Při nesprávně zapsaném vstupu je uživatel upozorněn přímo ve formuláři. Pokud byl vstup zadán správně, je vytvořena uživateli relace a následně uložena do cookies, takže je jako přihlášený.

#### 4.5.5 Stránka pro přihlášení

Přihlašovací formulář je skoro stejný jako registrační, akorát se po validaci vstupů ověřuje uživatel na základě jeho hesla a přezdívky. Jsou zde zachycené nové chyby, jako například neexistující uživatel nebo špatné heslo. Název stránky v projektu je `login`.



Obrázek 18 Přihlašovací stránka s formulářem

#### 4.5.6 Informační přehled

Poslední stránku přidáme pro přihlášené i nepřihlášené uživatele. Bude sloužit k rychlému seznámení se s funkcemi webové aplikace. Zároveň se využije komponenta `<Stepper>` z knihovny Skeleton UI, která uživatelsky přívětivě reprezentuje informace v krocích.

#### 4.5.7 SEO

Ke každé stránce jsem přidal komponentu *SEO* z globální složky */lib*. Ta stránkám nastavuje *title*, ale hlavně poskytuje strukturovaná data ve formátu, který je vhodný pro vyhledávače. To znamená, že vyhledávače lépe rozumí obsahu stránky a mohou ji více a lépe zviditelnit ve výsledcích vyhledávání.

## ZÁVĚR

V teoretické části byla představena problematika výběru nástroje pro tvorbu webových aplikací a podrobněji byl zkoumán framework SvelteKit. Byly popsány klíčové vlastnosti a funkcionality SvelteKit, včetně směřování, načítání dat a odesílání formulářů.

Praktická část práce se zaměřila na návrh a implementaci demonstrační webové aplikace. Nejprve byly definovány funkcionální a nefunkcionální požadavky. Poté byl navržen databázový model a v závěru byl popsán postup vývoje demonstrační webové aplikace. Pro vývoj bylo využito možností, které nabízí framework SvelteKit a nástroje s ním spojené.

Během implementace jsem došel k následujícím zjištěním: Práce se SvelteKit je poměrně svižná, protože framework obsahuje spoustu funkcí, které jsou v praxi často využívány a jednoduše se implementují. Další skvělá věc je, že vývoj frameworku neustále pokračuje. To ale může mít za následek určité nevýhody. Například to, že starší návody a postupy mohou být po provedení aktualizace zastaralé a neoptimální. Ke všemu je sice psána dokumentace na oficiálních stránkách, ale občas je psána až moc přátelsky a málo odborně. Naštěstí SvelteKit disponuje skvělou a aktivní komunitou, která je přítomná na různých komunikačních kanálech, kde vývojáři mohou klást otázky ohledně nesrovnalostí, nebo navrhopvat změny a vylepšení. I přestože jsem se s některými postupy teprve seznámil, rychle jsem si je oblíbil. Zejména způsob řízení přechodů mezi stránkami a psaní serverové části.

## SEZNAM POUŽITÉ LITERATURY

- [1] Getting started with Svelte. In: *MDN Web Docs* [online]. individual mozilla.org contributors, 1998 [cit. 2023-04-13]. Dostupné z: [https://developer.mozilla.org/en-US/docs/Learn/Tools\\_and\\_testing/Client-side\\_JavaScript\\_frameworks/Svelte\\_getting\\_started](https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Svelte_getting_started)
- [2] JEBAVÝ, Josef. Úvod do technologie Svelte: Nástroj k vývoji technicky vyspělých webových aplikací. In: *SvelteJS* [online]. 2019 [cit. 2023-04-13]. Dostupné z: <https://www.sveltejs.cz/>
- [3] HARRIS, Rich. Write less code: The most important metric you're not paying attention to. In: *Svelte* [online]. [cit. 2023-04-13]. Dostupné z: <https://svelte.dev/blog/write-less-code>
- [4] *Svelte: CYBERNETICALLY ENHANCED WEB APPS* [online]. [cit. 2023-04-13]. Dostupné z: <https://svelte.dev/>
- [5] HARRIS, Rich. SvelteKit is in public beta. In: *Svelte* [online]. [cit. 2023-04-13]. Dostupné z: <https://svelte.dev/blog/sveltekit-beta>
- [6] Introduction. In: *SvelteKit* [online]. [cit. 2023-04-13]. Dostupné z: <https://kit.svelte.dev/docs/introduction>
- [7] Loading data. In: *SvelteKit: web development, streamlined* [online]. [cit. 2023-04-13]. Dostupné z: <https://kit.svelte.dev/docs/load>
- [8] Age: HTTP Response - Age. In: *MDN Web Docs* [online]. individual mozilla.org contributors, 1998 [cit. 2023-04-13]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Age>
- [9] ŠTRÁFELDA, Jan. HTTP Hlavička. In: *Jan Štráfelda: Průvodce online projektem* [online]. [cit. 2023-04-13]. Dostupné z: <https://www.strafelda.cz/http-hlavicka>
- [10] ZAIDI, Nasir. Top Companies using ReactJS in 2022. In: *Walturn* [online]. [cit. 2023-04-16]. Dostupné z: <https://www.walturn.com/insights/top-companies-using-reactjs-in-2022>
- [11] HÁMORI, Ferenc. The History of React.js on a Timeline. In: *RisingStack* [online]. 2022 [cit. 2023-04-16]. Dostupné z: <https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/>
- [12] WILLOUGHBY, John. All Things React. In: *Telerik* [online]. 2023 [cit. 2023-04-16]. Dostupné z: <https://www.telerik.com/blogs/all-things-react>
- [13] *Angular developer guides* [online]. 2021 [cit. 2023-04-24]. Dostupné z: <https://angular.io/guide/developer-guide-overview>
- [14] Introduction. In: *Lucia* [online]. [cit. 2023-05-23]. Dostupné z: <https://lucia-auth.com/start-here/introduction?sveltekit>
- [15] What is Prisma?. In: *Prisma* [online]. Prisma Data, 2023 [cit. 2023-05-23]. Dostupné z: <https://www.prisma.io/docs/concepts/overview/what-is-prisma>
- [16] HARRIS, Rich. Project structure. In: *SvelteKit: web development, streamlined* [online]. [cit. 2023-05-23]. Dostupné z: <https://kit.svelte.dev/docs/project-structure>
- [17] Getting started. In: *Lucia: Authentication, simple and clean* [online]. 2023 [cit. 2023-05-23]. Dostupné z: <https://lucia-auth.com/start-here/getting-started?sveltekit>
- [18] Prisma. In: *Lucia* [online]. 2023 [cit. 2023-05-23]. Dostupné z: <https://lucia-auth.com/adapters/prisma?sveltekit>

- [19] PRATS, Diego. Web app development security best practices. In: *Developer Docs* [online]. 2023 [cit. 2023-05-24]. Dostupné z: <https://internetcomputer.org/docs/current/developer-docs/security/web-app-development-security-best-practices>
- [20] Configuration: csp. In: *SvelteKit: web development, streamlined* [online]. 2023 [cit. 2023-05-24]. Dostupné z: <https://kit.svelte.dev/docs/configuration#csp>
- [21] Sessions. In: *Lucia* [online]. 2023 [cit. 2023-05-24]. Dostupné z: <https://lucia-auth.com/basics/sessions?sveltekit>
- [22] Handle requests. In: *Lucia* [online]. 2023 [cit. 2023-05-24]. Dostupné z: <https://lucia-auth.com/basics/handle-requests?sveltekit>
- [23] Zod: TypeScript-first schema validation with static type inference. In: *Zod* [online]. [cit. 2023-05-24]. Dostupné z: <https://zod.dev/>

## SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

URL	Uniform Resource Locator
CSP	Content Security Policy
XSS	Cross Site Scripting
IDE	Integrated development environment
CLI	Command Line Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
DOM	Document Object Model
VDOM	Virtual Document Object Model
UI	User Interface
API	Application Programming Interface
CSS	Cascading Style Sheets
JSX	JavaScript XML
SSR	Server Side Rendering
CSR	Client Side Rendering
ORM	Object Relational Mapping

## SEZNAM OBRÁZKŮ

Obrázek 1 Kód v html pro přepínání mezi stránkami ve filesystem based routingu.....	13
Obrázek 2 Přidání navigace pro všechny stránky pomocí layout .....	14
Obrázek 3 Obsah v src/routes/users/+page.ts pro získání uživatelů .....	14
Obrázek 4 Obsah v src/routes/users/+page.svelte pro zobrazení uživatelů .....	15
Obrázek 5 Obsah v src/routes/users/[userId]/+page.ts pro získání dat uživatele .....	15
Obrázek 6 Obsah v src/routes/users/[userId]/+page.svelte .....	15
Obrázek 7 Zachycení chyby pro neexistující ID v src/routes/users/[userId]/+page.server.ts .....	16
Obrázek 8 Soubor src/routes/users/[userId]/+error.svelte pro vykreslení chyby uživateli..	17
Obrázek 9 Nastavení hlavičky ve funkci load .....	19
Obrázek 10 Sdílená funkce v scr/lib/greet.js .....	19
Obrázek 11 Import funkce pomocí \$lib .....	19
Obrázek 12 Ukázka tří actions v +page.server.ts pro formulář .....	21
Obrázek 13 Kód v +page.svelte pro formulář.....	22
Obrázek 14 Snapshots kód v +page.svelte.....	24
Obrázek 15 Prisma schéma s modely .....	37
Obrázek 16 Ukázka domovské stránky .....	45
Obrázek 17 Potvrzovací modal při odhlášení .....	49
Obrázek 18 Přihlašovací stránka s formulářem .....	50



**SEZNAM TABULEK**

Tabulka 1 Funkcionální požadavky stručně .....	29
Tabulka 2 Nefunkcionální požadavky stručně.....	31
Tabulka 3 Příklad užití registrace .....	32
Tabulka 4 Příklad užití přihlášení .....	32
Tabulka 5 Příklad užití odhlášení .....	33
Tabulka 6 Příklad užití vytvoření příspěvku.....	33
Tabulka 7 Příklad užití smazání příspěvku .....	34
Tabulka 8 Příklad odstranění uživatele.....	34
Tabulka 9 Příklad užití přesměrování pomocí menu .....	35
Tabulka 10 Příklad užití změna světelného režimu .....	35
Tabulka 11 Příklad užití tlačítko pro posun na stránce.....	35

## SEZNAM PŘÍLOH

Příloha P I: CD se zdrojovým kódem

