SPŠT Střední průmyslová škola Třebíč

Maturitní práce

WEBOVÁ APLIKACE PRO PŘIHLÁŠKY DO DOMOVA MLÁDEŽE

Profilová část maturitní zkoušky

Studijní obor: Informační technologie
Třída: ITB4
Školní rok: 2025 Jan Prokůpek

Zadání práce

Cílem práce je návrh a implementace webové aplikace pro evidenci přihlášek zájemců o ubytování v domově mládeže a pro podporu komunikace mezi žadateli a administrátory. Aplikace bude sloužit k centralizované správě procesu podávání žádostí a poskytne uživatelům přehledný a transparentní způsob sledování stavu jejich přihlášky. Uživatelé budou mít možnost vytvářet a spravovat své žádosti prostřednictvím systému ticketů, kde bude zobrazena časová osa jednotlivých kroků řízení včetně vyjádření ze strany administrátora. Administrátorská část aplikace umožní pracovníkům domova mládeže spravovat přijaté žádosti, komunikovat se zájemci a vyhodnocovat je také na základě dojezdových vzdáleností či dalších relevantních kritérií.

Aplikace bude vyvíjena s využitím frameworku Next.js v jazyce TypeScript, pro tvorbu uživatelského rozhraní bude použit TailwindCSS a databázová vrstva bude řešena prostřednictvím PostgreSQL a ORM Prisma. Součástí práce bude také návrh databázového modelu a architektury aplikace, implementace uživatelských i administrátorských modulů a ošetření základních bezpečnostních aspektů.

Významnou součástí projektu bude nasazení hotové aplikace do produkčního prostředí na servery Střední průmyslové školy Třebíč, její praktické ověření v reálném provozu a zpracování uživatelské i technické dokumentace, která bude sloužit pro následnou údržbu a využívání systému v praxi.

ABSTRAKT

Maturitní práce má za úkol usnadnit proces přihlašování žáků a správu přihlášek vychovateli vytvořením aplikace pro správu přihlašovacího procesu. Úvod krátce popisuje aktuální řešení a problémy s ním spojené. V teoretické části jsou shrnuty technologie, které byly při vývoji použity a důvody, proč byly vybrány pro vývoj. Praktická část pojednává o shrnutí vlastní implementace projektu, strategiích, které byly při vývoji použity a průběhem nasazením aplikace na server Střední průmyslové školy Třebíč.

KLÍČOVÁ SLOVA

přihlašovací formulář, domov mládeže, webová aplikace

ABSTRACT

The graduation thesis aims to simplify the process of student registration and application management for educators by creating an application for managing the registration process. The introduction briefly describes the current solution and the problems associated with it. The theoretical part summarizes the technologies used during development and explains the reasons why they were chosen. The practical part discusses the implementation of the project itself, the strategies used during development, and the process of deploying the application on the server of the Secondary Technical School in Třebíč.

KEYWORDS

application form, dormitory, web application

PODĚKOVÁNÍ

Děkuji vedoucímu práce Mgr. Matěji Brožkovi za cenné rady a odborné vedení při zpracování této práce.

PROHLÁŠENÍ

Prohlašuji, že jsem tuto práci vypracoval/a samostatně a uvedl/a v ní všechny prameny, literaturu a ostatní zdroje, které jsem použil/a. Pro vypracování této práce byla použita šablona pro citování podle normy ČSN ISO 690-2022, kterou vytvořili Marta Zizienová a Adam Zizien, licencovaná pod Creative Commons Zero v1.0 Universal.

V Třebíči dne 15. května 2024

Podpis autora

Obsah

Úvod

Správa přihlašovacích formulářů do domova mládeže je často náročný a časově intenzivní proces, který vyžaduje efektivitu při zpracování žádosti a případné komunikaci se žadateli. Cílem této práce je navrhnout a implementovat webovou aplikaci za použití moderních technologií, která tento proces zjednoduší a zpřehlední jak pro žadatele, tak pro vychovatele domova mládeže. Historicky bylo přihlašování do domova mládeže řešeno prostřednictvím papírových formulářů, které byly vyplňovány ručně a posílány e-mailem. V posledních letech se však začali hledat alternativy, které by umožnily digitalizaci tohoto procesu. Minulý rok bylo pilotně zavedeno zasílání přihlášek prostřednictvím Google Forms, což přineslo určité zlepšení. Nicméně tento systém má své limity, zejména pokud jde o evidenci a archivování přijatých žádostí. Přesně tyto problémy se snaží tato práce řešit vytvořením specializované webové aplikace.

Mezi primární funkce aplikace patří možnost vytváření a správu přihlášek žadateli, kteří budou moci sledovat stav své žádosti v reálném čase. Vychovatelé na domově mládeže pak získají nástroje pro správu přihlášky, automatizovanou komunikaci se žadateli a přehled o všech přijatých žádostech. Aplikace bude také obsahovat funkce pro hodnocení a výběr žadatelů na základě bodů, které budou automaticky uděleny na základě odpovědí na otázky.

Výsledná aplikace by měla být přínostná pro všechny strany, jmenovitě pro žadatele, kteří získají pohodlnou a transparentní cestu k podání přihlášky, a pro vychovatele, kteří budou mít efektivní nástroj pro správu a zpracování přihlášek.

1 Problematika stávajícího řešení

Jak již bylo zmíněno v úvodu, aktuální stav systému pro přihlašování a správu přihlášek pro domov mládeže je neoptimální a obsahuje množství funkcí, které lze implementovat pro celkové zlepšení uživatelské přívětivosti a pohodlnosti. Pro jednoduché ustanovaní těchto funkcí je však se potřeba podívat na všechny typy, které kdy byly zvažovány pro použití a následně tato data zohlednit při návrhu nového systému.

1.1 Fyzické přihlášky

Fyzické přihlášky, také zvané papírové přihlášky, jsou již dnes považovány za staromódní, avšak je potřeba si z nich vzít důležité poznámky o tom, jaké nevýhody přinášeli a ty následně zvážit při implementaci našeho řešení.

Při porovnání s ostatními řešeními mají bezkonkurenčně nejvíce nevýhod. Mezi nevýhody definitivně patří:

- Nutnost fyzického doručení přihlášky na určené místo, což může být pro některé žadatele komplikované.
- Obtížná správa a archivace papírových přihlášek. Papírové dokumenty se těžce hledají, třídí a uchovávají, což dokáže zvýšit administrativní zátěž pro vychovatele.
- Omezené možnosti pro automatizaci procesu hodnocení a kalkukace bodů na základě odpovědí žadatelů.

1.2 Google Forms

Google Forms je online nástroj pro tvorbu formulářů, sběr a statistiku dat. Nabízí široké možnosti přizpůsobení formulářů, integraci s dalšími službami Google a automatické shromažďování odpovědí do přehledných tabulek. [1]

Mezi hlavní výhody Google Forms patří hlavně intuitivní uživatelské rozhraní, jednoduché nastavení a možnost rychlého sdílení formulářů prostřednictvím odkazu. Dále nabízí automatický export dat do tabulek. Všechny tyto funkce výrazně usnadňují přihlašovací proces, avšak přichází i funkce které jsou vitální a chybí. Pro školní rok 2025/2026 byly Google Forms pilotně využity jako nástroj pro sběr přihlášek. Část procesu přihlašování z administrativní strany tvoří např. archivace přijatých přihlášek

ve formátu PDF[1], či komunikace se žadateli. Tyto funkce však Google Forms nenabízí, což vede k nutnosti manuálního zpracování.

1.3 Externí řešení

Externí řešení patří zdaleka k nejvhodnějším možnostem, jak řešit přihlašovací proces. Mezi hlavní výhody patří především fakt, že produkty lze přizpůsobit na míru dle požadavků a potřeb domova mládeže. Nabízejí též opravu chyb, aktualizace a technickou podporu, což může být velice užitečné pro zajištění hladkého chodu systému. Mezi nevýhody však patří především finanční náročnost, jelikož externí řešení často vyžadují měsíční nebo roční poplatky za podporu a údržbu.

[1] Tento krok byl řešen automaticky za pomocí zautomatizovaného skriptu v Google Sheets, z vlastní zkušenosti bylo toto řešení však velice chybové a často se muselo upravovat.

2 Stanovení požadavků na novou aplikaci

Při diskuzi o tvorbě nové aplikacei byly též stanoveny požadavky na funkce, které by měla aplikace obsahovat, odvíjely jsme se především od problémů, které přinášela stávající řešení. Tyto požadavky můžeme pro přehlednost rozdělit do dvou hlavních kategorií: **požadavky na uživatelské funkce** a **požadavky na administrátorské funkce**. Na základě těchto požadavků bude následně proveden výběr technologií a návrh architektury aplikace.

2.1 Uživatelské funkce

- Zobrazení formuláře pro přihlášení do domova mládeže s možností vyplnění a odeslání přihlášky.
- Možnost sledování stavu přihlášky v reálném čase prostřednictvím *ticket systému*.

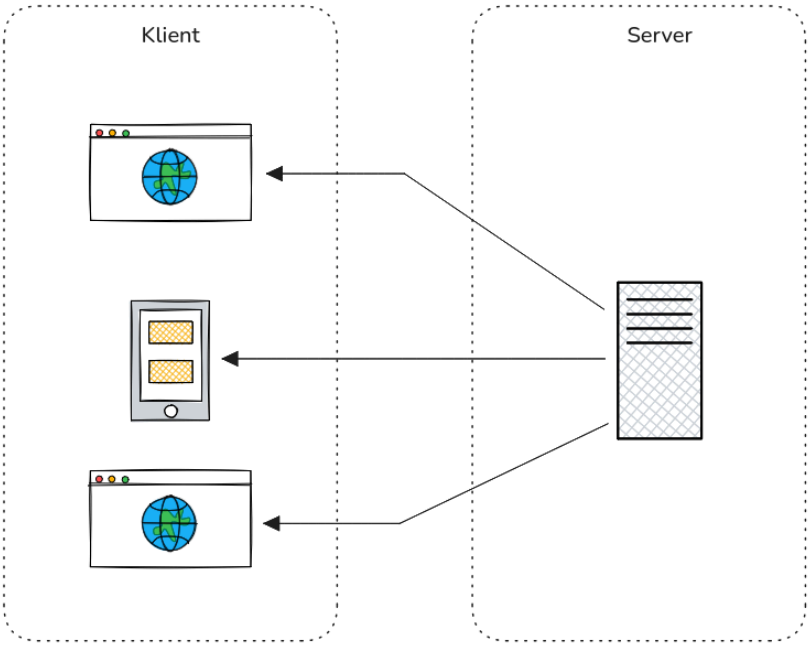
2.2 Administrátorské funkce

- Zobrazení přehledu všech přijatých přihlášek s možností filtrování, či přidávání poznámek.
- Možnost komunikace se žadateli prostřednictvím integrované funkce pro zasílání zpráv za pomoci e-mailu.
- Automatické bodování přihlášek na základě odpovědí žadatelů.
- Generování a archivace přijatých přihlášek společně s možností exportu do PDF.
- Možnost úpravy formuláře pro přihlášení dle aktuálních potřeb.
- Zabezpečení přístupu k administrátorským funkcím pomocí autentizačního systému s RBAC[2], který zajistí různé úrovně přístupu pro různé role vychovatelů.

[2]Role-Based Access Control je systém pro efektivní správu přístupu k zabezpečeným informacím pomocí rolí a oprávnění. [2]

3 Architektura webové aplikace

Vývoj jakékoliv aplikace by měl začít návrhem její architektury. Ta dokáže přibližně nastínit, jak do sebe budou jednotlivé části aplikace zapadat a také se od ní odvíjí výběr technologií, které budou při vývoji použity. Pro webové aplikace je de-facto standardem architektura **client-server**, která dělí aplikaci na dvě hlavní části – klientskou a serverovou.



Obrázek 1: Vizualizace client-server architektury

3.1 Klient

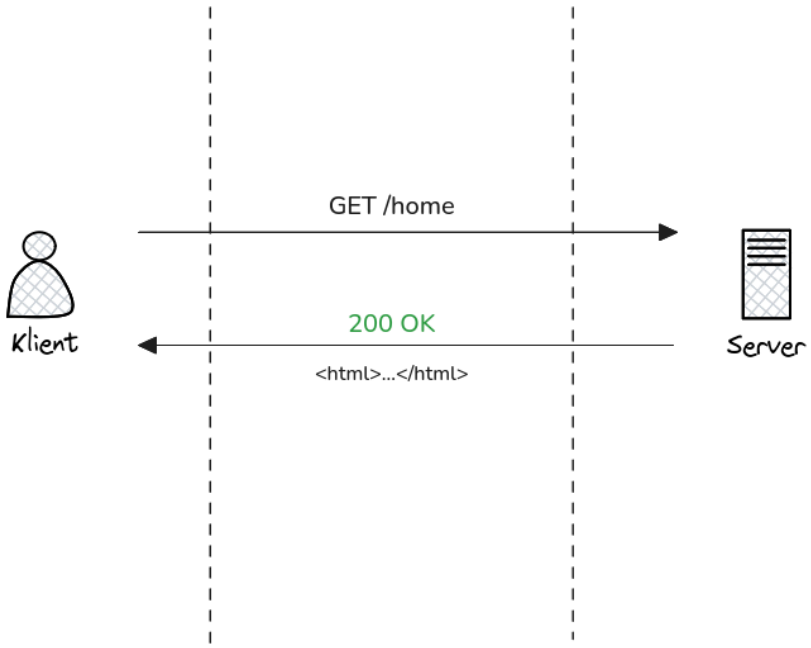
Klientská část aplikace je zodpovědná za interakce s uživatelem. V případě webové aplikace se jedná kód, který běží přímo v prohlížeči uživatele. Jejím hlavním úkolem je zobrazit uživatelské rozhraní a zpracovávat interakce způsobené uživatelem.

3.2 Server

Serverová část aplikace běží na vzdáleném zařízení (serveru) a je zodpovědná primárně za úkony, které považujeme za nebezpečné při vykonávání na straně klienta, jako je např. přístup k databázi, validace dat, autentizace uživatelů či vnitřní logika aplikace. Server je také zodpovědný za poskytnutí dat samotné webové stránky (HTML, CSS, JavaScript), které následně klient uživateli zobrazí a umožní mu s nimi interagovat.

3.3 Komunikace mezi klientem a serverem

Klient a server spolu může komunikovat několika protokoly, mezi nejpopulárnější patří HTTP/HTTPS, GraphQL či WebSocket. Pro většinu webových aplikací, je však nejvhodnější volbou využití HTTP/HTTPS protokolu. Tento protokol umožňuje klientovi odesílat požadavky na server a přijímat odpovědi, což je ideální pro většinu scénářů webových aplikací.



Obrázek 2: Vizualizace komunikace mezi klientem a serverem pomocí HTTP protokolu

Každý požadavek odeslaný klientem obsahuje metodu (mezi nejběžněji používané patří GET, POST, PUT a DELETE), URL adresu, hlavičky a případné tělo požadavku[3]. Server následně zpracuje požadavek a vrátí odpověď obsahující stavový kód, hlavičky a případně tělo odpovědi s daty.

[3]Tělo požadavku je obvykle přítomno u metod jako POST a PUT, které odesílají data na server.

3.4 Autentizace a autorizace

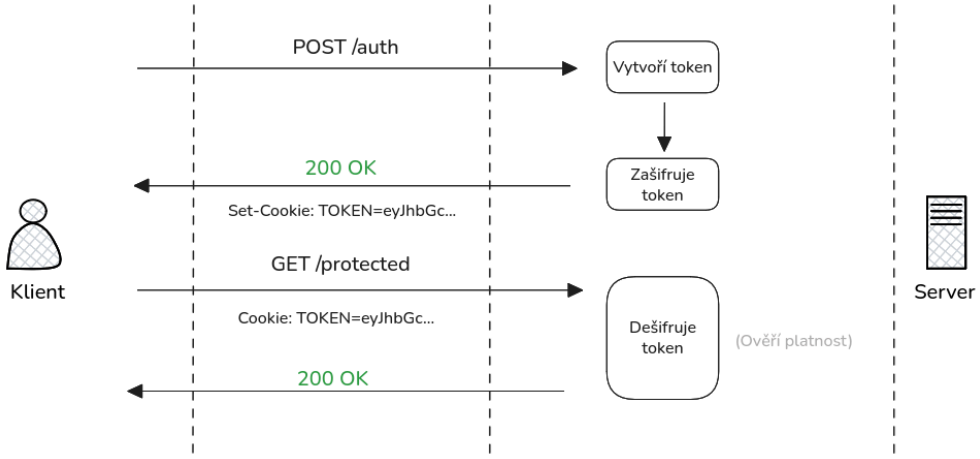
Pro zabezpečení přístupu k funkcím, jež jsou určeny pouze pro oprávněné klienty (uživatele), je potřebná implementace systému pro autentizaci a autorizace. Autentizace je proces ověření identity uživatele, zatímco autorizace určuje, jaké akce může autentizovaný uživatel provádět.

V dnešní době dělíme autentizaci na 2 primární typy – **session-based authentication** a **JWT (JSON Web Tokens)**. Často se však můžeme setkat i termíny jako je **stateful** a **stateless authentication**, tyto termíny však přímo popisují, zda server uchovává stav o přihlášeném uživateli (stateful) nebo nikoliv (stateless).

3.4.1 JWT (JSON Web Tokens)

JWT je standard pro bezpečnou **stateless** autentizaci (tj. neuchovává stav přihlášeného uživatele na serveru). Autentizace je rozdělena na 2 hlavní kroky – přihlášení a ověření tokenu. Při přihlášení klient vyšle požadavek na server s přihlašovacími údaji (např. uživatelské jméno a heslo). Server ověří tyto údaje a pokud jsou správné, vygeneruje JWT token(obsahující informace o uživateli a jeho oprávněních ve formátu JSON), tento token je zašifrován za pomocí asymetrického klíče a následně odeslán zpět klientovi. Klient si tento token uloží (např. do localStorage nebo cookies) a při každém dalším požadavku na server ho přiloží v hlavičce `Authorization`. Server následně ověří platnost tokenu (např. kontrolou podpisu a expirace) a pokud je token platný, povolí přístup k požadovaným zdrojům.

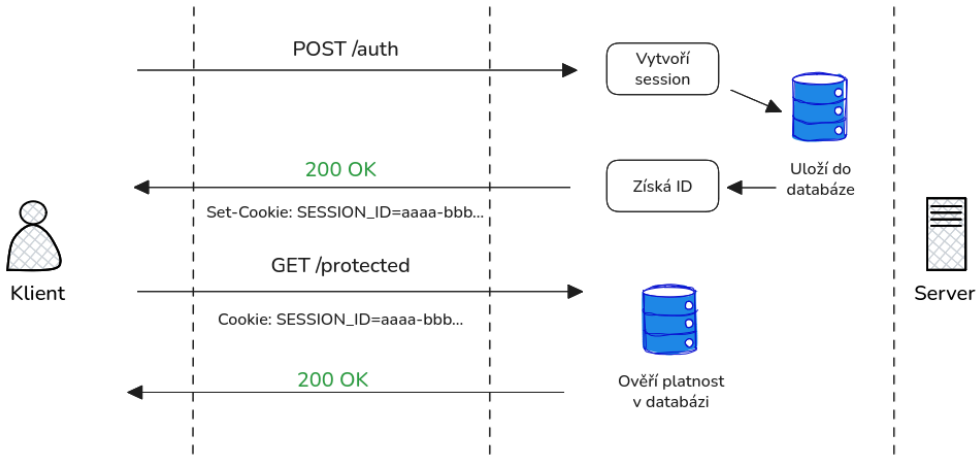
Oproti session-based autentizace má tento způsob hlavní nevýhodu v tom, že server nemá možnost uživatele odhlásit před vypršením platnosti tokenu, jelikož server neuchovává žádný stav o přihlášeném uživateli.



Obrázek 3: Vizualizace JWT autentizace

3.4.2 Session-based authentication

Druhým hojně užívaným způsobem pro autentizaci je tzv. session-base autentizace. Tento styl je oproti JWT **stateful**, ukládá tedy stav přihlášeného uživatele na serveru (v databázi). Při přihlášení klient odešle požadavek na server s přihlašovacími údaji. Server ověří tyto údaje a pokud jsou správné, vytvoří novou session (relaci) pro uživatele a vygeneruje unikátní identifikátor session (session ID). Tento identifikátor je následně odeslán zpět klientovi, který si ho uloží do cookies. Při každém dalším požadavku na server klient automaticky přiloží cookies obsahující session ID. Server následně ověří platnost session ID (např. kontrolou, zda session stále existuje v databázi) a pokud je platné, povolí přístup k požadovaným zdrojům.



Obrázek 4: Vizualizace session-based autentizace

4 Technologie použité při vývoji

Při vývoji moderní webové aplikace je klíčové zvolit technologie, které kromě splnění požadavků na funkcionalitu zajistí i dlouhodobou udržitelnost, škálovatelnost a bezpečnost aplikace. Následující technologie byly vybrány na základě jejich výhod a existujících zkušeností.

4.1 TypeScript

TypeScript je programovací jazyk, který je nadstavbou JavaScriptu a přidává mu statické typování. To přidává řadu výhod, jako např. bezpečné typy, či lepší čitelnost kódu. [3]

```
1 const greeting: string = "Ahoj, Světe!";
2 console.log(greeting);
```

Výpis 1: Ukázka kódu v TypeScriptu

Díky TypeScriptu je možné odhalit chyby již během vývoje, což vede k vyšší kvalitě kódu a snížení počtu chyb v produkčním prostředí. TypeScript je svými funkcemi také podporován ve většině moderních vývojových nástrojů, což usnadňuje práci vývojářům.

4.2 Next.js

Next.js je webový framework, který je postaven na Reactu a umožňuje tvorbu kompletních webových aplikací s podporou pokročilých funkcí, jako je **Server-Side Rendering** (SSR), nebo **Server Actions**. [4]

4.2.1 React

React je knihovna pro tvorbu uživatelských rozhraní, který umožňuje vytváření komponent založených na stavech a vlastnostech přímo v JavaScriptu či TypeScriptu. [5] Jedná se o jeden z nejpoužívanějších nástrojů pro vývoj webových aplikací. Díky přímé integraci v Next.js umožňuje efektivní tvorbu dynamických a interaktivních uživatelských rozhraní.

React umožňuje tvorbu *znovupoužitelných komponent*. Tyto komponenty jsou prosté funkce nebo třídy[4], které přijímají vstupní data(*props*, též známo v HTML jako atributy). Komponenty mohou také spravovat svůj vlastní stav – *state*, což umožňuje vytváření interaktivních prvků uživatelského rozhraní.

Každý soubor s příponou `.tsx` nebo `.jsx` představuje soubor podporující speciální syntaxi JSX, ta umožňuje kombinovat kód podobný HTML přímo do JavaScriptu/ TypeScriptu. Tento kód je následně přeložen do nativního JavaScriptu, který je vykonáván v prohlížeči.

```tsx
// hello-world.tsx
"use client";
import React from "react";

function HelloWorld() {
  const [greeting, setGreeting] = React.useState("Ahoj, Světe!");

  return (
    <div>
      <h1>{greeting}</h1>
      <button onClick={() => setGreeting("Ahoj, Next.js!")}>
        Změnit pozdrav
      </button>
    </div>
  );
}
```

Výpis 2: Ukázka komponenty v Reactu

[4] V moderním Reactu je doporučeno používat výhradně funkční komponenty.

4.2.2 React Server Components

React Server Components (RSC) je speciální typ komponenty v React, která umožňuje vykonávání kódu komponenty na serveru místo v prohlížeči. V Next.js lze rozlišit RSC a běžné komponenty na straně klienty pomocí direktivy `"use client"` umístěné na začátku souboru. Pokud tato direktiva chybí, automaticky se React automaticky považuje všechny komponenty definované v daném souboru jako serverové komponenty. [6]

Tento speciální typ komponenty umožňuje vývojářům přistupovat ke zdrojům na serveru, jako je databáze nebo souborový systém přímo z komponenty, aniž by bylo nutné vytvářet API rozhraní pro komunikaci. Jednou z úskalí RSC je, že tyto komponenty nemohou používat interaktivní prvky(např. `onClick` události nebo `useState` hook).

Využití samotné RSC také prodlužuje dobu načítání stránky, protože React čeká na dokončení obsluhy serverové komponenty před tím, než odešle výsledná data do prohlížeče klientovi. Pro zvýšení uživatelské přivětivosti (UX) existuje proto tzv. *Suspense* komponenta, ta dokáže zobrazit náhradní obsah (např. indikátor načítání) zatímco server čeká na dokončení vykonání RSC.

```tsx
// server-component.tsx
import React from "react";

async function ServerComponent() {
  const data = await fetchDataFromDatabase();

  return (
    <div>
      <h1>Data ze serveru:</h1>
      <pre>{JSON.stringify(data, null, 2)}</pre>
    </div>
  );
}

function Page() {
  return (
    <div>
      <h1>Moje stránka s RSC</h1>
      <React.Suspense fallback={
        <div>Načítání dat ze serveru...</div>
      }>
        <ServerComponent />
      </React.Suspense>
    </div>
  )
}
```

Výpis 3: Ukázka React Server Component a jejího použití s Suspense

4.2.3 Server Actions v Next.js

Server Actions(česky Serverové akce nebo Funkce na straně serveru) nahrazují potřebu vytváření samostatné API na serveru, kterou by bylo nutné z klientské strany volat. Místo toho lze funkce, jež jsou definovány ve speciálním souboru volat přímo z komponent na straně klienta. Next.js interně automaticky vytvoří potřebné API na pozadí.[7]

Jako příklad můžeme vytvořit jednoduchou funkci, jejíž úkolem bude vrátit aktuální čas ze serveru. Server Actions jsou definovány v souborech které začínají direktivou `"use server"`.

```
1  "use server";
2
3  export async function getServerTime() {
4    return new Date().toISOString();
5  }
```

Výpis 4: Ukázka Server Action v Next.js

Funkci lze následně importovat a volat přímo z komponenty na straně klienta.

```
1  "use client";
2  import React from "react";
3  import { getServerTime } from ...;
4
5  function TestovaciKomponenta() {
6    const cas = React.use(getServerTime());
7
8    return <div>Aktuální čas ze serveru: {cas}</div>;
9  }
```

Výpis 5: Ukázka komponenty v Next.js využívající Server Action

4.3 Prisma ORM

Prisma je moderní ORM (Object-Relational Mapping) nástroj pro TypeScript, který slouží k interakcemi s databází za pomocí automaticky generovaného typovaného API. [8]

Jedná se o jednu z nejpopulárnějších možností pro práci s databázemi v TypeScriptu, a díky své jednoduchosti pro vykonávání jednoduchých CRUD operací byla ideální volbou pro tento projekt.

Každý projekt definuje své schéma databáze v souboru zakončeného příponou `.prisma`. Tento soubor obsahuje modely, které reprezentují tabulky a jejich vztahy v databázi. Prisma následně na základě tohoto schématu generuje typované API pro interakci s databází.

```
model User {
  id       Int      @id @default(autoincrement())
  email    String   @unique
  name     String?
}
```

Výpis 6: Ukázka schématu databáze v Prisma ORM s modelem `User`

```
const prisma = new PrismaClient();
await prisma.user.create({
  data: {
    email: "<email>",
    name: "<name>"
  }
});
```

Výpis 7: Ukázka použití generovaného kódu pro práci s modelem `User`

Prisma podporuje širokou škálu databázových systému, včetně PostgreSQL, MySQL, SQLite a dalších. [8]

4.3.1 PostgreSQL

PostgreSQL je relační SQL databázový systém, který byl společně s Prismou zvolen jako hlavní databázové řešení pro tento projekt. Jedná se o jeden z nejpoužívanějších databázových systémů s pokročilými funkcemi, jako je podpora transakcí, bezpečnost pomocí Row-Level Security (RLS), pokročilé datové typy (JSON, UUID), či Full--Text Search.

4.4 TailwindCSS

TailwindCSS je podpůrný CSS framework pro moderní webový vývoj, který umožňuje rychlé a efektivní vytváření uživatelských rozhraní za pomocí tříd s předdefinovanými styly. [9]

TailwindCSS umožňuje vývojářům vytvářet responzivní a přizpůsobitelná rozhraní bez nutnosti psaní vlastního CSS kódu od nuly. Poskytuje širokou škálu tříd, které pokrývají různé aspekty stylování, jako je rozvržení, barvy, responzivita, typografie a další.

Seznam použité literatury

[1] GOOGLE LLC. *Google Forms: Online formuláře pro rychlé statistiky*. Online. Dostupné z: https://workspace.google.com/products/forms/. [cit. 2025-10-24]

[2] MALE, Vinay. Decoding Role-Based Access Control (RBAC). Online. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology.* 2025, vol. 11, s. 2082–2090. Dostupné z: https://doi.org/10.32628/CSEIT251112211. [cit. 2025-10-24]

[3] MICROSOFT. TypeScript: JavaScript with syntax for types. Online. Dostupné z: https://www.typescriptlang.org/. [cit. 2025-10-24]

[4] VERCEL INC. Next.js: The React Framework for the Web. Online. Dostupné z: https://nextjs.org/. [cit. 2025-10-24]

[5] META PLATFORMS, INC. React: The library for web and native user interfaces. Online. Dostupné z: https://react.dev/. [cit. 2025-10-24]

[6] META PLATFORMS, INC. React Server Components. Online. Dostupné z: https://react.dev/reference/rsc/server-components. [cit. 2025-10-30]

[7] VERCEL INC. Server Actions in Next.js. Online. Dostupné z: https://nextjs.org/docs/13/app/api-reference/functions/server-actions. [cit. 2025-10-24]

[8] PRISMA INC. Prisma: Next-generation ORM for Node.js and TypeScript. Online. Dostupné z: https://www.prisma.io/orm. [cit. 2025-10-30]

[9] TAILWIND LABS, INC. Online. Dostupné z: https://tailwindcss.com/. [cit. 2025-11-12]

Seznam obrázků

Seznam tabulek