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, React, Next.js

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, React, Next.js

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. Při přípravě této práce jsem použil ChatGPT (https://chat.openai.com/) a Github Copilot (https://github.com/features/copilot) za účelem kontroly kódu, překladů v aplikaci a hledání chyb. Po použití tohoto nástroje jsem provedl kontrolu obsahu a přebírám za něj plnou zodpovědnost.

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 ovládacího panelu.
- Automatické zasílání notifikací e-mailem při změně stavu přihlášky (např. přijetí, zamítnutí).
- Využití stejného uživatelského účtu pro podání více přihlášek (např. sourozenci, jiné ročníky), bez nutnosti nové registrace.

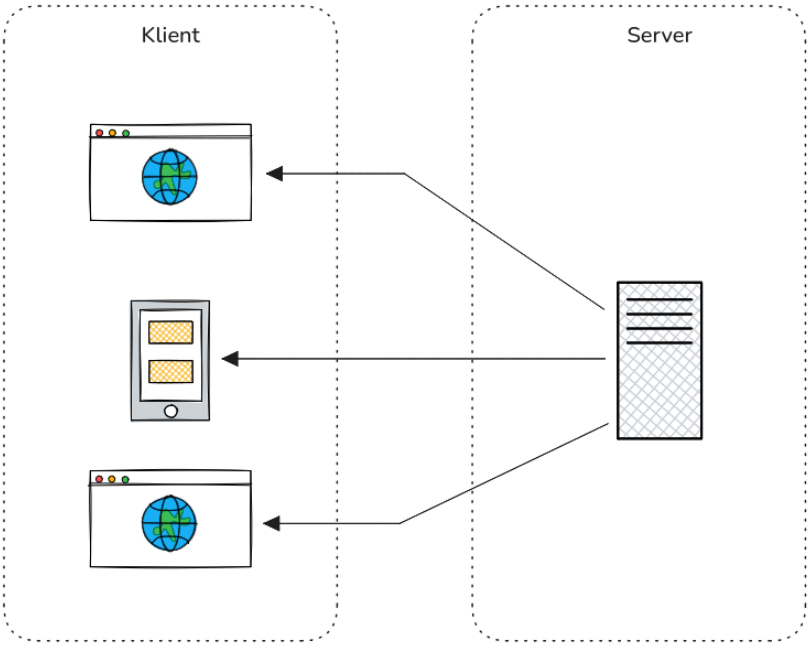
2.2 Administrátorské funkce

- 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, či jiných formátů.
- Jednoduchá úprava studijních oborů a ročníků, pro minimalizaci intervence správce systému.
- 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

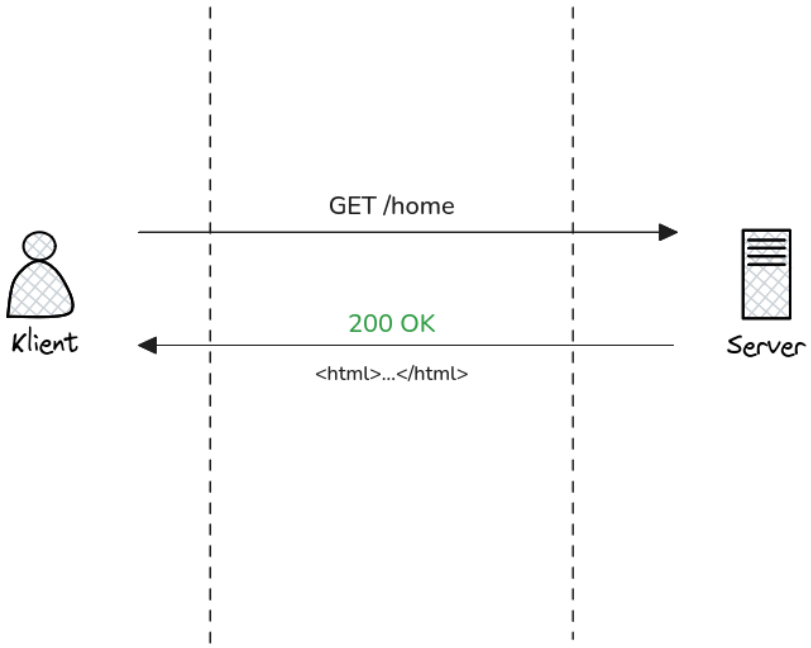
Klientská část aplikace je zodpovědná za interakce s uživatelem. V případě webové aplikace se jedná kód, který běží přímo v prohlížeči uživatele. Jejím hlavním úkolem je zobrazit uživatelské rozhraní a zpracovávat interakce způsobené uživatelem.

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

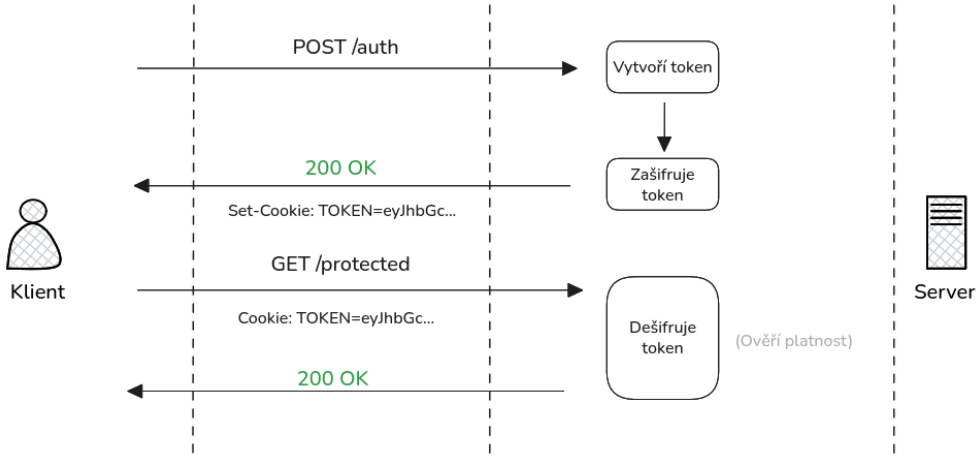
Pro zabezpečení přístupu k funkcím, jež jsou určeny pouze pro oprávněné klienty (uživatele), je potřebná implementace systému pro autentizaci a autorizace. Autentizace je proces ověření identity uživatele, zatímco autorizace určuje, jaké akce může autentizovaný uživatel provádět.

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.

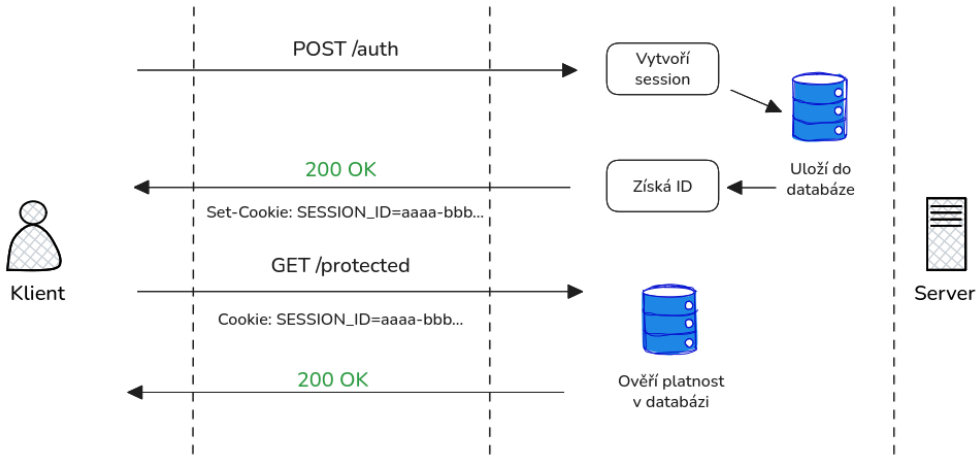
Oproti session-based autentizace má tento způsob hlavní nevýhodu v tom, že server nemá možnost uživatele odhlásit před vypršením platnosti tokenu, jelikož server neuchovává žádný stav o přihlášeném uživateli.



Obrázek 3: Vizualizace JWT autentizace

3.4.2 Session-based authentication

Druhým hojně užívaným způsobem pro autentizaci je tzv. session-based 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 moderních verzích knihovny React je doporučeno používat výhradně funkční komponenty.

4.2.2 React Server Components

React Server Components (RSC) je speciální typ komponenty v Reactu, 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 přívětivosti (UX) existuje proto tzv. *Suspense* komponenta, ta dokáže zobrazit náhradní obsah (např. indikátor načítání) zatímco server čeká na dokončení vykonání RSC.

RSC lze do jisté míry přirovnat ke klasickému PHP – které kód vykonává na serveru a uživateli pošle až hotovou HTML stránku. Oproti PHP však RSC umožňuje kombinovat serverový a klientský kód v rámci jedné aplikace, což přináší větší flexibilitu a možnosti pro vývojáře.

```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* (Full-textové vyhledávání). Tyto funkce dělají PostgreSQL ideální volbou pro moderní webové aplikace.

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.

5 Implementace webové aplikace

5.1 Návrh databáze a datového modelu

Jak již bylo zmíněno v kapitole o použitých technologiích, pro práci s databází byla zvolena ORM Prisma. Samotný databázový model pak vychází z výchozího modelu používaného knihovnou Better Auth (viz Kapitola 5.3). Tento model byl následně rozšířen o další sloupce a tabulky, které byly za potřeba pro implementace webové aplikace.

TODO DOPLNIT SCHEMA DATABAZE

Pro validaci většiny vstupních dat (klientských i serverových), která jsou později zasazena do databázového modelu byla použita knihovna Zod[5]. Ta umožňuje definovat schémata pro validaci dat a poskytuje jednoduché API pro ověřování datových struktur. Zod je plně kompatibilní s TypeScriptem a je použit jak na straně klienta, tak i na straně

[5] https://zod.dev/

serveru. Na straně serveru je Zod použit primárně v kombinaci s Next-Safe-Action (viz Kapitola 5.3.1.1) pro validaci dat přicházejících z klienta.

5.2 Inicializace aplikace

Při prvním spuštění aplikace dojde k takzvanému *bootstrappingu* aplikace. Tento proces zahrnuje kroky nutné pro funkce aplikace, jako je připojení k databázi, načtení konfiguračních proměnných, či vytvoření základních struktur potřebných pro běh aplikace.

Tento krok také vytvoří výchozí administrátorský účet (hlavního vychovatele) aplikace, pokud v databázi ještě žádný neexistuje. Uživatel je vytvořen na základě konfiguračních proměnných, které jsou nastaveny v souboru `.env` v kořenovém adresáři projektu.

5.3 Přihlašovací formulář a autentizace

Po úvodním otevření aplikace je uživatel automaticky přesměrován na přihlašovací stránku, kde se lze přihlásit, nebo zaregistrovat nový účet. Pro úspěšnou registraci je potřeba zadat platnou e-mailovou adresu, uživatelské jméno a heslo. Uživatel má téže možnost si vybrat, zda-li si má prohlížeč zapamatovat heslo pro příští návštěvy aplikace. Po úspěšné registraci je uživatel přesměrován do samotného uživatelského rozhraní aplikace.

Pokud uživatel již účet má, může se přihlásit zadáním e-mailové adresy a hesla. V případě zadání neplatných údajů je uživatel informován o chybě a je vyzván k opětovnému zadání správných údajů.

5.3.1 Autentizace a autorizace pomocí knihovny Better Auth

Pro implementaci přihlašovacího formuláře, autentizace i autorizace byla zvolena knihovna Better Auth[6]. Ta nabízí jednoduché a rychlé řešení přihlašování v aplikacích postavených na nejen Reactu a Next.js. Better Auth využívá session-based autentizace, takže jsou do cookies ukládány session identifikátor, které server ověřuje při každém požadavku.

[6] https://www.better-auth.com/

Knihovna je modulární a podporuje přidávání *pluginů*, které rozšiřují její funkce. Jedním z těchto pluginů je i plugin pro podporu RBAC, který umožňuje definovat různé role uživatelů a jejich oprávnění v aplikaci. Aplikace byla rozdělena na 3 hlavní role:

- `guest` (žadatel) – role pro běžného uživatele, tento uživatel vidí pouze do žadatelského rozhraní a nemá přístup k většině serverové API.
- `user` (vychovatel) – role pro vychovatele domova mládeže, uživatel s touto rolí vidí administrátorské rozhraní a má částečný přístup k serverovému API. Některé funkce, jako například konfigurace aplikace, archivace, nebo správa uživatelů jsou omezeny.
- `admin` (administrátor – hlavní vychovatel) – role pro hlavního vychovatele domova mládeže, uživatel s touto rolí má plný přístup k administrátorskému rozhraní a serverovému API. Může spravovat uživatele, nastavovat konfiguraci aplikace a provádět další administrativní úkony.

5.3.1.1 Middleware pro ochranu veřejné API

Je nutno zmínit, že aplikace obsahuje API, kterou lze dosáhnout z veřejné sítě. Tato API je vygenerovaná Next.js z Serverových akcí a je tedy přístupná z klientské části aplikace. Tuto API je nutno ochránit před neoprávněným přístupem, což je zajištěno pomocí knihovny Next-Safe-Action[7] (dále již jen jako NSA).

NSA je knihovna, která abstrahuje serverové akce v Next.js a přidává jim možnost validace, zachycování chyb a zachycování požadavků za chodu [10].

V konfiguraci NSA lze použít funkci `.use()` pro definování *middleware* (prostředník pro zachycování požadavků). Tento *middleware* je vykonán před samotnou serverovou akcí a může být použit pro různé účely, jako je kontrola autentizace uživatele, logování požadavků, či manipulace s daty požadavku.

5.4 Rozhraní pro žadatele

V režimu žadatele, tedy uživatele s rolí `guest`, má uživatel přístup k velmi omezeným funkcím samotné aplikace.

[7] https://next-safe-action.dev/

5.4.1 Boční navigační menu

Boční navigační menu je hlavním navigačním prvkem aplikace. Umožňuje uživateli přístup k různým sekcím aplikace, jako je přehled přihlášek, možnosti uživatele, nebo odhlášení z aplikace. Navigační menu je navrženo tak, aby bylo responzivní vůči různým velikostem obrazovky a zařízení.

5.4.2 Sekce *Moje přihlášky*

V sekci *Moje přihlášky* lze vytvářet nové přihlášky, prohlížet si již vytvořené přihlášky a sledovat jejich stav. Vytvořené přihlášky jsou rozděleny do zobrazovacích karet, které obsahují rozkliknutelné karty s přehledem informací o přihlášce, jako je datum vytvoření, stav přihlášky a počet získaných bodů, a další.

5.4.3 Formulář pro vytvoření přihlášky

Formulář pro vytvoření přihlášky je dostupný po kliknutí na tlačítko *Nová přihláška* v sekci *Moje přihlášky*. Na formulář se uživatel nedostane, pokud je v administrátorském panelu nastaveno, že přihlášky nejsou momentálně přijímány, nebo již vypršela lhůta pro podání přihlášky.

Formulář obsahuje několik sekcí, které pokrývají různé části přihlášky: údaje o žadateli, údaje o zákonných zástupcích a další otázky týkající se přihlášky. Každá sekce obsahuje různé typy vstupních polí, jako jsou textová pole, výběrové seznamy, přepínače a další. Pro postoupení do další sekce je vždy potřeba vyplnit všechna povinná pole tak, aby podléhala schématu validace. Po úspěšném vyplnění všech sekcí a odeslání formuláře je přihláška uložena do databáze a uživatel je přesměrován zpět do sekce *Moje přihlášky*, kde může sledovat stav své přihlášky. Přihlášku po odeslání již není možné upravovat.

Při každém odeslání formuláře je formulář zařazen do ročníku, jenž je aktuálně otevřen pro přijímání přihlášek. Žadatelskému rodnému číslo je také přiřazeno číslo *evidenční*, to je automaticky vygenerováno (nebo při opětovném podání přihlášky znovu použito) na základě počtu již přijatých přihlášek v daném ročníku.

5.4.4 Nastavení profilu

Sekce je užita pro správu uživatelského profilu. Uživatel zde může měnit své osobní údaje, jako je jméno, e-mailová adresa a heslo. Pro změnu hesla je potřeba zadat heslo aktuální a nové heslo.

5.5 Rozhraní pro vychovatele

Rozhraním pro vychovatele se rozumí administrativní část aplikace, která umožňuje spravovat přihlášky a vykonávat další administrativní úkony. Toto rozhraní je pouhým rozšířením uživatelského rozhraní pro žadatele, tímpádem má vychovatel stále přístup k podávání přihlášek a ostatním úkonům, které by za normálních okolností žadatel měl.

5.5.1 Nastavení chování aplikace

Nastavení aplikace se nadále dělí na několik podčástí (sekcí). Většina těchto nastavení je dostupna pouze pro uživatele s rolí *hlavní vychovatel*.

5.5.1.1 Sekce „Obecné“

Sekce obsahuje hlavní nastavení samotné aplikace, a to konkrétně:

- **Přístup k přihlašovacímu formuláři** – vybrání mezi možnostmi:
  - „Otevřeno pro všechny (ignorovat uzávěrku)“ – umožňuje přístup k formuláři všem přihlášeným uživatelům.
  - „Otevřeno pro vychovatele (ignorovat uzávěrku)“ – přístup k přihlášce je možný pouze uživatelům s rolí *user* a vyšší.
  - „Uzavřeno po datu konce přihlašování“ – obsah je dostupný pouze do vypršení předem stanovené lhůty.
  - „Uzavřeno“ – k obsahu nemá přístup žádný typ uživatele.
- **Datum konce přihlašování** – nastavením datumu a času znemožníte přístup k formuláři po daném termínu. Toto pravidlo však vejde v platnost pouze v případě, že je přístup k formuláři nastaven na „Uzavřeno po datu konce přihlašování“.
- **Název domény** – toto pole je pouze estetické. Vzhledem k univerzálnímu návrhu aplikace bylo přidáno toto pole, aby žadatelé dokázali jednoduše odlišit, pro jaké internátní zařízení aplikaci využívají.

- **Potvrzení při odeslání přihlášky** – slouží jako text, který musí uživatelé odsouhlasit před odesláním samotné přihlášky.

5.5.1.2 Sekce „Ročníky“

V této sekci lze spravovat ročníky, které jsou v aplikaci uloženy. Ročníky lze archivovat, či nastavit jako výchozí. Výchozím ročníkem se rozumí ročník, pro který se budou ukládat nové přihlášky, či generovat nová evidenční čísla. V aplikaci vždy existuje alespoň jeden ročník a ročník a také je vždy nastaven jeden ročník jako výchozí.

5.5.1.3 Sekce „Studijní obory“

Sekce sloužící pro správu studijních oborů. Každý obor má jméno, krátkou formu jména, délku studia a možnost nastavení, zda-li se jedná o obor, ve kterém začíná výuka brzy (tento fakt poté ovlivňuje samotné bodování přihlášky).

5.5.1.4 Sekce „Účty“

Sekce pro správu uživatelských účtů. Uživatelé mohou být přidáváni, odebíráni a lze upravovat jejich role.

5.5.1.5 Sekce „E-maily“

Sekce pro správu e-mailových šablon, které jsou používány pro komunikaci s žadateli. Lze také nastavit výchozího odesílatele a předmět, se kterým bude e-mail odeslán. Je nutno podotknout, že tato část neslouží pro nastavení připojení k SMTP serveru, to je řešeno pomocí konfiguračních proměnných v souboru `.env`.

5.5.2 Zobrazení přijatých přihlášek

Vychovatelé mají přístup k přehlednému seznamu všech přihlášek. Seznam je zde implementován pomocí přehledné tabulky s podporou stránkování, filtrování a vyhledávání. Každá přihláška je zobrazena v řádku tabulky s možností rozkliknutí pro zobrazení detailních informací o přihlášce. Z této stránky lze také přihlášky mazat.

5.5.2.1 Detail přihlášky

Každou přihlášku lze rozkliknout pro zobrazení detailních informací. V detailu přihlášky jsou zobrazeny všechny informace, které žadatel zadal při vyplňování přihlášky,

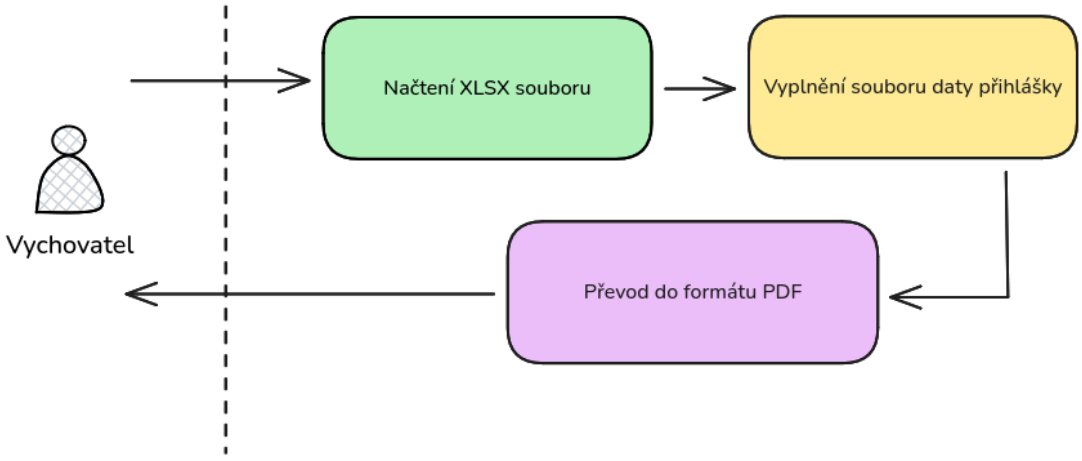
včetně automaticky vygenerovaného bodového ohodnocení a stavu přihlášky. Vychovatel zde má také možnost měnit stav přihlášky (např. přijatá, zamítnutá) a přidávat poznámky. V případě, že uživatel, který přihlášku vytvořil, špatně vyplnil libovolný údaj, má možnost údaj v přihlášce opravit.

5.5.2.2 Export přihlášek do PDF

Export přihlášky do PDF formátu je poněkud záludná záležitost, jelikož generování PDF na straně serveru v prostředí Node.js není příliš, ani efektivní. Pro tento účel bylo zvoleno řešení pomocí kombinace přihlášky ve formátu XLSX a následné konverze do PDF pomocí nástroje LibreOffice.

Proces generování PDF probíhá ve třech krocích:

- Jednorázové nahrání přihláškové šablony XLSX do aplikace.
- Při požadavku na export přihlášky je vytvořena kopie šablony, do které jsou následně vloženy údaje z přihlášky. Vložení údajů probíhá pomocí knihovna **ExcelJS**, která umožňuje manipulaci s XLSX soubory v prostředí Node.js. V šabloně jsou přepsány položky s předem definovanými názvy buněk (např. buňka s obsahem `$application_id$` bude nahrazena univerálním identifikátorem přihlášky).
- Upravený XLSX soubor je následně vyexportován do PDF pomocí knihovny **libreoffice-convert**, který abstrahuje volání LibreOffice z příkazové řádky pro konverzi mezi různými formáty dokumentů.



Obrázek 5: Ukázka toku exportu přihlášky do formátu PDF

Řešení pomocí mezisouboru XLSX se může zdát jako redundantní a zbytečně komplikované, avšak v současné době neexistuje žádné široce používané řešení pro vyplňování (přepisování) PDF šablon v prostředí Node.js. Tento přístup tedy představuje kompromis mezi složitostí implementace a funkčností.

5.5.3 Archivace

V systému lze ročníky archivovat, což znamená, že přihlášky v daném ročníku již nebudou moci být upravovány ani přidávány nové přihlášky. Archivaci ročníků lze spravovat v sekci nastavení aplikace. Nelze aktivovat ročník, který je nastaven jako výchozí.

Archivace ročníku nemá vliv na již vygenerovaná evidenční čísla přihlášek, ta zůstávají nadále platná a jsou spojena s daným ročníkem. Archivace slouží tedy převážně k ochraně dat před nechtěnými úpravami a organizaci dat v systému.

5.5.4 Evidenční čísla

Evidenční čísla jsou unikátní identifikátory přihlášek, které mohou být generovány při odeslání přihlášky. Evidenční číslo je tvořeno kombinací prvního ročníku, ve kterém byla pro dané rodné číslo vygenerována přihláška a pořadového čísla přihlášky v daném ročníku. Tento výrok tedy implikuje, že pro každé rodné číslo, bude číslo evidenční vygenerováno pouze jednou, a při opětovném podání přihlášky v dalším ročníku, bude použito již existující evidenční číslo.

2025/1

Obrázek 6: Příklad evidenčního čísla přihlášky (první přihláška v roce 2025)

V programu lze importovat existující evidenční čísla pro předem definovaná rodná čísla. Tento import je užitečný v případech, kdy docházelo k tvoření evidenčních čísel mimo systém (např. ručně) a je potřeba tato čísla synchronizovat s databází aplikace. Import probíhá pomocí CSV souboru, který obsahuje dva sloupce – rodné číslo a evidenční číslo.

5.5.5 Massmail

Massmail je funkce, která umožňuje hromadné odesílání e-mailů všem žadatelům, nebo vybraným skupinám žadatelů na základě různých kritérií (např. stav přihlášky, ročník, atd.). Tato funkce je užitečná pro komunikaci s velkým počtem žadatelů najednou, například pro informování o změnách v přijímacím řízení, nebo pro zasílání potvrzení o přijetí přihlášky. Funkce je dostupná pro všechny vychovatele a skrývá se pod položkou *Hromadné e-maily* v bočním navigačním menu. E-maily lze posílat ve formátu prostého textu, nebo HTML.

6 Vývoj a nasazení

6.1 Vývojové nástroje

6.1.1 Docker

6.1.2 Užití statických analýzátorů kódu

6.1.2.1 Biome.js

6.1.2.2 SonarQube

6.1.3 Aktivní monitorování

6.1.3.1 Grafana a Loki

6.2 Plánování vývoje pomocí GitHub Issues

6.3 Verzovací systém Git

6.3.1 GitHub

6.4 Nasazení na produkční server

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]

[10] EDOARDO RANGHIERI. Online. Dostupné z: https://next-safe-action.dev/. [cit. 2025-12-25]

Seznam obrázků

Seznam tabulek