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

Zadání maturitní práce je přílohou této práce.

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. Chtěl bych také poděkovat své rodině a blízkým přátelům za jejich podporu během celého období tvorby této práce.

V Třebíči dne 11. 2. 2026

Podpis autora

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/featuassets/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 11. 2. 2026

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 v 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 dle uvedených odpovědí na otázky.

Výsledná aplikace by měla být přínostná pro všechny strany. Jmenovitě pak 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

Při analýze stávajících mechanismů podávání přihlášek se fyzická (papírová) forma jeví jako procesně nejméně efektivní varianta. Její limity však zároveň definují požadavky na modernizaci a digitalizaci celého systému. Mezi kritické nedostatky, které je nutné v rámci návrhu nového řešení eliminovat nebo je omezit, patří zejména:

- Nutnost fyzického doručení přihlášky na určené místo. To může být pro některé žadatele komplikované.
- Obtížná správa a archivace fyzických 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. Nutnost manuálního zpracování vede k chybovosti a daleko vyšší časové náročnosti.

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 a nejflexibilnějším 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 patří především finanční náročnost, jelikož externí řešení často vyžadují měsíční nebo roční poplatky za podporu a údržbu.

Vlastnost	Fyzické přihlášky	Google Forms	Externí řešení	Studentský projekt
Snadná manipulace s přihláškou	ne	ano	ano	ano
Automatizace hodnocení přihlášek	ne	částečně	ano	ano
Archivace přihlášek	ne	částečně	ano	ano
Komunikace se žadateli	ne	ne	možná	ano
Implementace na míru	ne	ne	ano	ano
Možnost expanze funkcí	částečně	ne	ano	ano
Náklady	nízké	nízké	vysoké	nízké

Tabulka 1: Jednoduché porovnání řešení přihlašovacího procesu do domova mládeže

2 Stanovení požadavků na novou aplikaci

Při diskuzi o tvorbě nové aplikace byly též stanoveny požadavky na funkce, které by měla aplikace obsahovat. Vycházeli 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 funkce administrátorské**. Na základě těchto požadavků byly následně vybrány technologie užité pro vývoj aplikace.

[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.

2.1 Uživatelské funkce

Uživatelskými funkcemi se rozumí funkce, které jsou přístupné a využívané samotnými žadateli o ubytování.

- Zobrazení přehledného formuláře pro podání přihlášky, který bude obsahovat všechny potřebné informace a otázky pro správné vyplnění žádosti. Zároveň bude obsahovat validaci pro zajištění správnosti a úplnosti zadaných dat společně s kolonkami pro nahrání potřebných souborů.
- Možnost sledování přihlášku, včetně jejího stavu, v reálném čase. Žadatelé budou mít jednoduchý přehled o tom, v jaké fázi se jejich žádost nachází, a budou informováni o případných změnách stavu.
- Využití stejného účtu pro podání a správu více přihlášek (předpokládá se, že uživatelé budou podávat přihlášky pro více svých dětí a ve více letech).

2.2 Administrátorské funkce

Do administrátorských funkcí spadají funkce, které jsou standardně přístupné pouze pro vychovatele a pracovníky domova mládeže, kteří mají na starosti správu přihlášek.

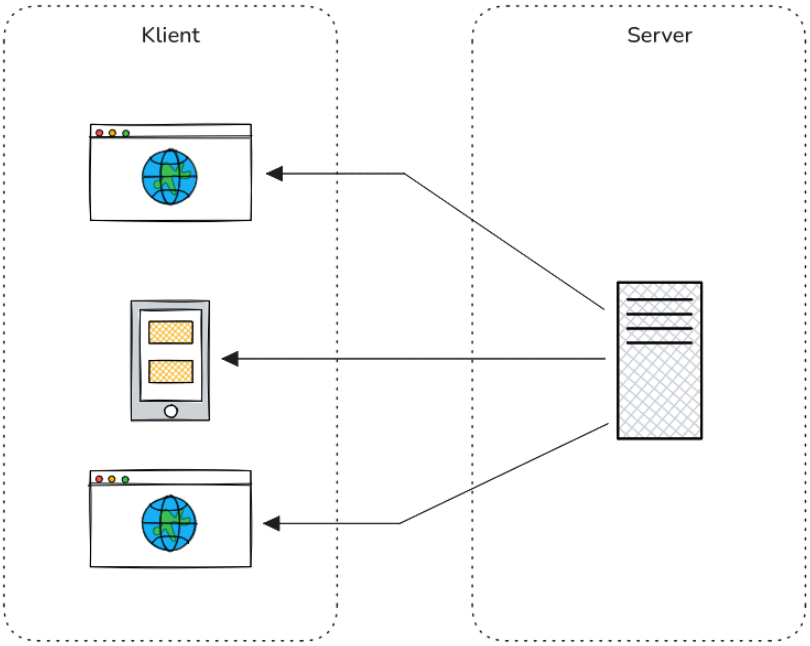
- Možnost spravovat přijaté žádosti, včetně možnosti prohlížet a upravovat informace o žadatelích, měnit stav žádostí a přidávat poznámky k jednotlivým žádostem.
- Automatizace procesu hodnocení a kalkulace bodů na základě odpovědí žadatelů. Aplikace bude automaticky udělovat body za jednotlivé odpovědi na otázky v přihlášce, což usnadní a zrychlí proces hodnocení žádostí.
- Možnost komunikace se žadateli přímo z aplikace, a to ať už prostřednictvím e-mailu, tak i za pomoci funkce pro hromadnou korespondenci, která využije interního školního mailového serveru pro odesílání zpráv.
- Zabezpečení přístupu k administrátorským funkcím pomocí autentizačního systému s RBAC[2], který zajistí různé úrovně přístupu pro různé role vychovatelů.

3 Architektura webové aplikace

Vývoj jakékoliv aplikace začíná návrhem její architektury. Ta dokáže přibližně nastínit, jak do sebe budou jednotlivé části aplikace zapadat a také se od ní odvíjí výběr

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

technologií, které budou při vývoji použity. Fungování moderních webových aplikací se opírá o rozdělení odpovědnosti mezi uživatelské rozhraní (*client-side*) a aplikační logiku běžící na serveru (*server-side*).



Obrázek 1: Vizualizace client-server architektury

3.1 Klient

Klientská část aplikace, někdy nazývaná také *front-end*, představuje kód a soubory, které jsou přeneseny do prohlížeče uživatele a vykonávány přímo na jeho zařízení. Hlavním účelem těchto dat je vykreslit uživatelské rozhraní, zpracovávat interakce s cílovým uživatelem a zpětně komunikovat se serverovou částí aplikace.

3.2 Server

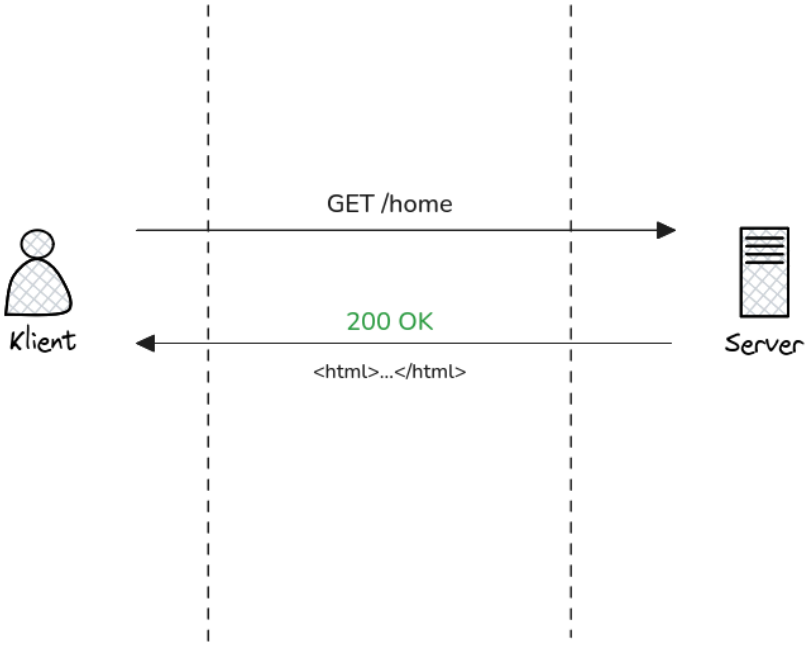
Serverová část aplikace, také označována jako *back-end* je část aplikace, která běží na vzdáleném zařízení (server), není tedy přístupná přímo uživateli. Serverová část aplikace je nejčastěji zodpovědná za interní zpracování dat, komunikaci s databází, autentizaci uživatelů a poskytování dat klientovi (včetně samotné webové stránky). Obecně by se tedy dalo říci, že tato část aplikace provádí úkony, které jsou považovány za nebezpečné (např. zápis do databáze) nebo neověřitelné (např. validace dat) při vykonávání na straně klienta.

3.3 Komunikace mezi klientem a serverem

Komunikace mezi klientem a serverem na aplikační vrstvě může používat různé protokoly, mezi nejběžněji používané protokoly patří HTTP (HyperText Transfer Protocol), WebSocket a GraphQL. V této práci probíhá komunikace mezi klientem a serverem primárně pomocí protokolu HTTP.

HTTP protokol dělí komunikaci na dvě hlavní části: požadavek (*request*) a odpověď (*response*). Klient – prohlížeč odešle požadavek na server, který tento požadavek zpracuje a následně odešle zpět odpověď. Požadavek i odpověď obsahují různé informace:

Klientský požadavek je definován především metodou (např. GET pro čtení či POST pro zápis), která určuje typ operace, a cílovou adresou URL identifikující konkrétní zdroj. Nedílnou součást tvoří hlavičky, nesoucí metadata typu autentizačních údajů či formátu dat, a volitelně také tělo požadavku obsahující samotná data k odeslání. Server následně na tento podnět reaguje strukturou, jejímž klíčovým prvkem je stavový kód (např. 200 OK či 404 Not Found), ten poskytuje okamžitou informaci o výsledku zpracování. Obdobně jako u požadavku, i odpověď obsahuje specifické hlavičky a zpravidla i tělo, které nese klientem vyžádaný obsah, nejčastěji ve formátu JSON nebo HTML.



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

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.

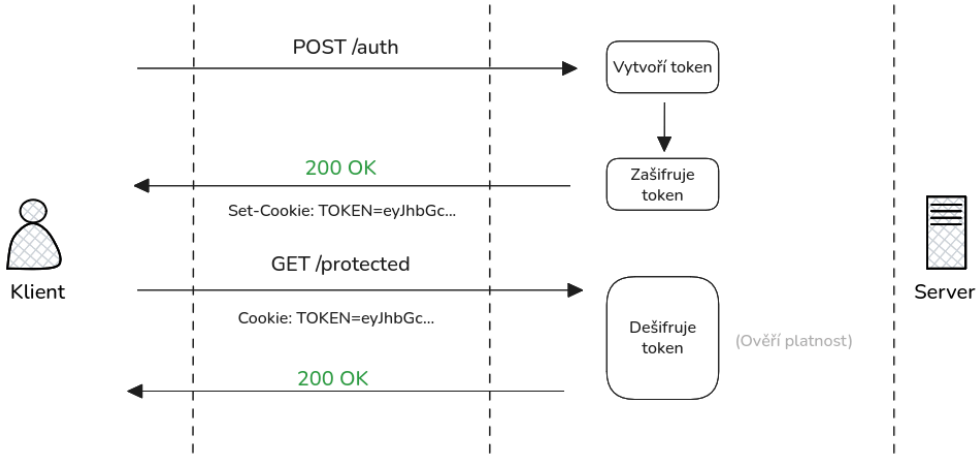
V dnešní době dělíme autentizaci na dva primární typy – **session-based authentication** a **JWT (JSON Web Tokens)**. Často se však můžeme setkat i termíny jako je **stateful** a **stateless authentication**, tyto termíny však přímo popisují, zda server uchovává stav o přihlášeném uživateli (stateful) 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 dva 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 [3].

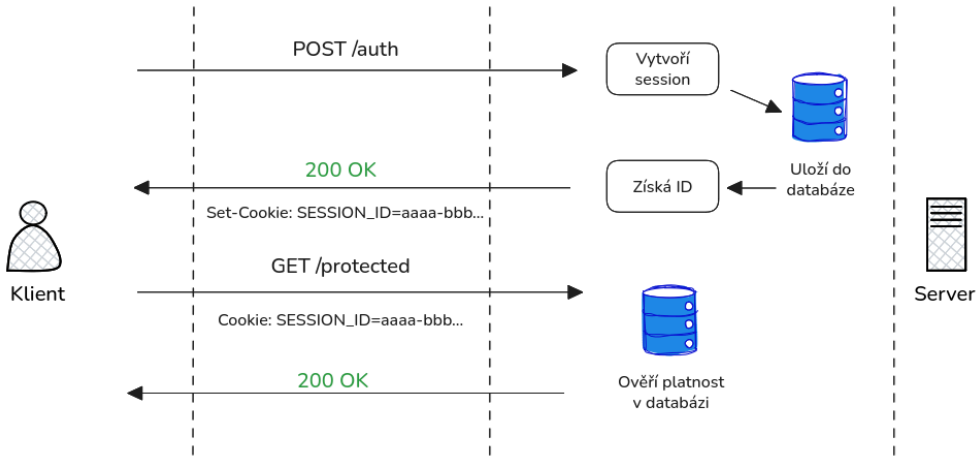
Oproti session-based autentizace má tento způsob hlavní nevýhodu v tom, že server nemá možnost uživatele odhlásit před vypršením platnosti tokenu, jelikož server neuchovává žádný stav o přihlášeném uživateli. Této funkce lze však dosáhnout implementací blacklistu neplatných tokenů na straně serveru, v takovém případě však JWT ztrácí svůj hlavní benefit – **stateless** charakter.



Obrázek 3: Vizualizace JWT autentizace

3.4.2 Session-based autentizace

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

Vlastnost	JWT Token	Session
Uchovávání stavu na serveru	ne	ano
Možnost odhlášení uživatele před vypršením platnosti	ne	ano
Škálovatelnost	vysoká	nízká

Tabulka 2: Porovnání standardů pro autentizaci a autorizaci

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 staticky typované rozšíření JavaScriptu, které přidává podporu pro typy, třídy, rozhraní a další funkce, které lze běžně najít ve striktně staticky a objektově orientovaných jazycích. Přidává silnou typovou kontrolu, která se prosazuje již přes samotnou transpilací kódu do JavaScriptu, což pomáhá odhalit chyby již během vývoje a zvyšuje kvalitu kódu [4].

TypeScript byl zvolen pro tento projekt z několika důvodů. Prvním z nich je fakt, že je hojně využíván v moderním webovém vývoji, což zajišťuje širokou podporu a

množství knihoven a nástrojů, které jsou s ním kompatibilní [5]. Dále přináší výhody v podobě lepší čitelnosti a údržby kódu, což je klíčové pro dlouhodobou udržitelnost projektu. TypeScript také umožňuje využívat pokročilé funkce a syntaxi, které nejsou nativně podporovány v JavaScriptu, což může zefektivnit vývoj a zlepšit celkovou strukturu kódu.

4.1.1 Typy a rozhraní

Jak již bylo v na začátku kapitoly zmíněno, TypeScript přináší do JavaScriptu rozsáhlo podporu typů. Typy jsou definovány pomocí klíčového slova `type` a umožňují vývojářům definovat vlastní datové struktury, které mohou být použity k zajištění správnosti dat v aplikaci. Mezi vybrané základní předdefinované datové typy pak patří:

- `string` pro textové řetězce
- `number` pro číselné hodnoty
- `boolean` pro logické hodnoty (true/false)
- `array` pro pole hodnot
- `enum` pro výčtové typy
- `any` pro hodnoty, které mohou být jakéhokoliv typu[3],
- `void` pro funkce, které nevracejí žádnou hodnotu
- `null` a `undefined` pro reprezentaci neexistujících nebo nedefinovaných hodnot

Pomocí TypeScript lze však specifikovat i složitější datové struktury. Typy lze kombinovat pomocí logických operátorů `&` (průnik typů) a `|` (sjednocení typů). Komplexní datové struktury lze pak definovat pomocí rozhraních (`interface`), které či `type` aliasů, které umožňují definovat strukturu objektů a jejich vlastností.

```
1 interface User {
2   id: number;
3   name: string;
4 }
5 type UserId = User["id"];
```

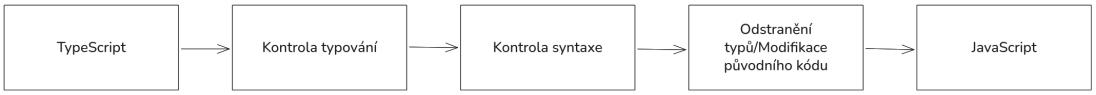
Výpis 1: Ukázka práce s rozhraními – definice rozhraní a typu odkazujícího na vlastnost rozhraní

[3]Použití typu `any` se nedoporučuje [6]. Jeho použití vede k vypnutí statické kontroly typů pro danou proměnnou, což může vést k chybám, které by jinak mohli být odhaleny během vývoje.

,

4.1.2 Transpilace

Pojmem *transpilace* se rozumí proces převodu kódu z jednoho programovacího jazyka do druhého. V případě TypeScriptu se jedná o převod kódu napsaného v TypeScriptu zpět do běžného JavaScriptu, který lze následně vykonávat nativně v prohlížeči či v serverovém prostředí (Node.js, Bun…). Transpilace umožňuje vývojářům využívat pokročilé funkce a syntaxi TypeScriptu, zatímco výsledný kód zůstává kompatibilní s širokou škálou prostředí, která podporují JavaScript.



Obrázek 5: Vizualizace procesu transpilace TypeScriptu do JavaScriptu

4.1.3 Standard ECMAScript

Vývoj syntaxe a funkcí v JavaScriptu (a tedy i TypeScriptu) je řízen standardem ECMAScript, který je pravidelně aktualizován a přináší modernizace syntaxe a funkcionalit do JavaScriptu [7]. TypeScript nabízí podporu pro verzi ECMAScript standardu až do ES5 [8]. Mezi některé moderní funkce, které standard ECMAScript přinesl v posledních letech, zejména patří šipkové funkce, třídy, moduly, podpora asynchroního programování [9].

Funkce	TypeScript	JavaScript
Statické typování	ano	ne
Podpora pro rozhraní a typy	ano	ne
Podpora pro moderní syntaxi	ano	částečně
Zachycení chyb během vývoje	ano	ne
Kompatibilita s prohlížeči	ne	ano
Nastavení	z počátku	žádné
Hromadné změny kódu	ano	složitější

Tabulka 3: Porovnání vybraných aspektů skriptovacích jazyků a srovnání TypeScriptu a JavaScriptu

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* [10].

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 [11]. 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.

[4] V moderních verzích knihovny React je doporučeno používat výhradně funkční komponenty.

```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 1: Ukázka komponenty v Reactu

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. [12]

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 2: 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.[13]

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 3: 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 4: 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 [14].

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 5: Ukázka schématu databáze v Prisma ORM s modelem `User`

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

Výpis 6: Ukázka použití generovaného kódu pro práci s modelem `User`

Prisma podporuje širokou škálu databázových systému, včetně PostgreSQL, MySQL, SQLite a dalších [14].

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 [15].

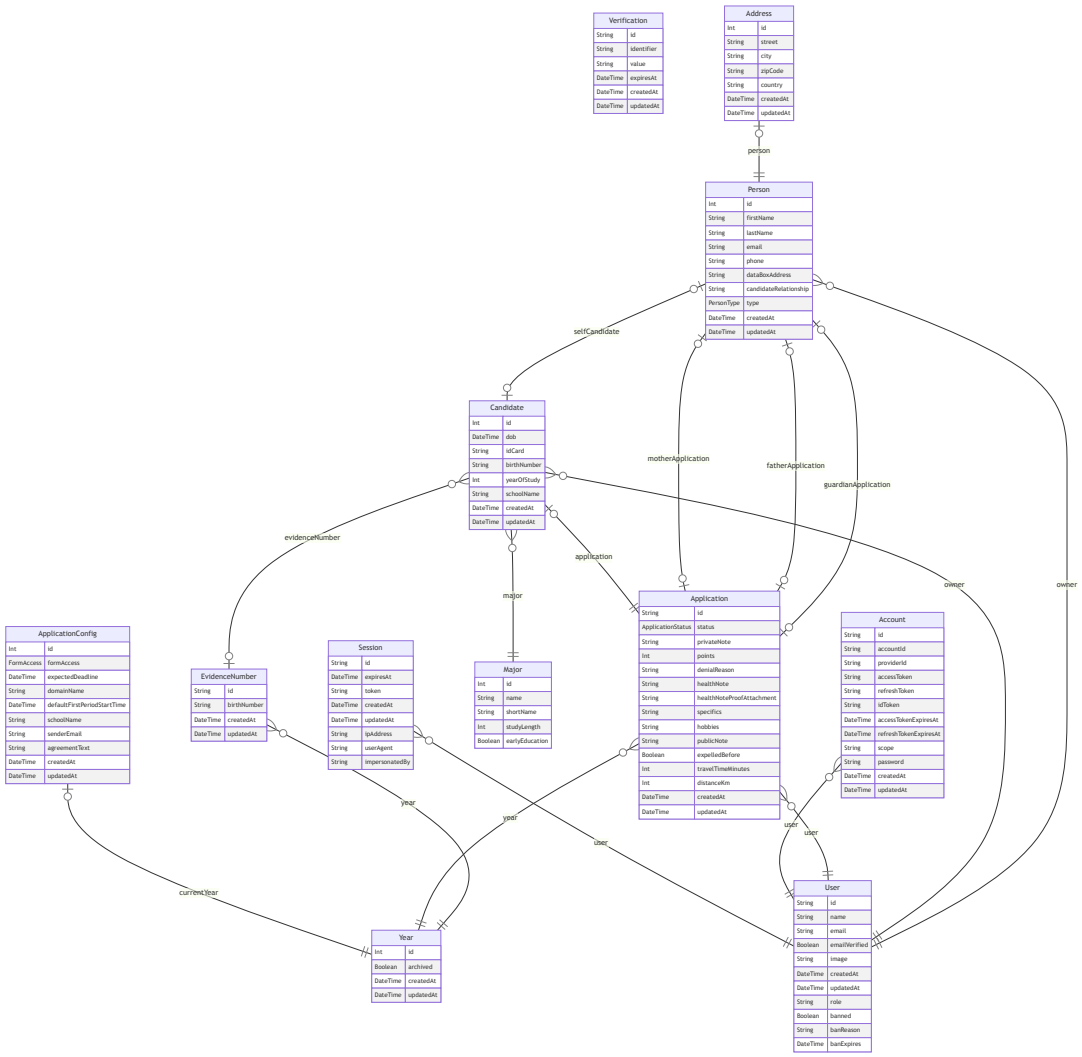
TailwindCSS umožňuje vývojářům vytvářet responzivní a přizpůsobitelná rozhraní bez nutnosti psaní vlastního CSS kódu od nuly. Poskytuje širokou škálu tříd, které

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 5.4). Tento model byl následně rozšířen o další sloupce a tabulky, které byly za potřebí pro implementaci webové aplikace.



Obrázek 6: Databázový model aplikace jako Entity-Relationship Diagram

Mezi některé důležité tabulky v modelu patří:

- `User` – tabulka pro ukládání informací o uživatelích, jako je e-mail, heslo, role a další.
- `Application` – tabulka pro ukládání informací o přihláškách, jako je stav přihlášky, datum podání, a další.
- `ApplicationConfig` – tabulka pro ukládání konfiguračních nastavení aplikace, jako jsou stav přijímání přihlášek, datum uzávěrky apod.
- `Person` – tabulka pro ukládání informací o osobách, které jsou spojeny s přihláškami (např. žadatel a zák. zástupce).

5.2 Kontrola a validace dat

Protože jsou data zadaná uživateli často náchylná k chybovosti, je důležité zabezpečit formuláře a API rozhraní kontrolou a validací vstupních dat. Validaci je důležité provádět duálně – jak na straně klienta (pro rychlou odezvu bez nutnosti komunikace se serverem), tak i na straně serveru (pro zajištění bezpečnosti a integrity dat). Důvodem pro dvojitý přístup k validaci je fakt, že validace na straně klienta může být snadno obejita (např. pomocí nástrojů pro vývojáře v prohlížeči), zatímco validace na straně serveru obejita být nemůže a zajišťuje správnost dat, která jsou uložena v databázi.

Pro práci s validací dat byla zvolena knihovna Zod[5]. Pomocí Zodu lze jednoduče definovat schéma pro validaci dat a následně použít toto schéma pro validaci dat jak na straně klienta, tak i na straně serveru. Zod umožňuje definovat jak komplexní, tak jednoduché datové struktury, je jednoduše použitelný a nabízí širokou škálu vestavěných validací pro různé datové typy (např. e-mail, URL, čísla, atd.). Zod zároveň využívá výhod TypeScriptu a z modelu lze automaticky odvodit typy, což zajišťuje konzistenci mezi validací a typovou kontrolou v celé aplikaci [16].

[5] https://zod.dev/

```
1 import { z } from "zod";
2
3 const userSchema = z.object({
4   email: z.string().email(),
5   password: z.string().min(8),
6   name: z.string().optional(),
7 });
8 type User = z.infer<typeof userSchema>;
```

Výpis 2: Příklad užití knihovny Zod

5.3 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.

5.4 Přihlášovací 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.

5.4.1 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.

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.

5.4.1.1 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 [17].

V konfiguraci NSA lze použít funkci `.use()` pro definování *middleware* (prostředník pro zachycování požadavků). Tento *middleware* je vykonán před samotnou serverovou

[6] https://www.better-auth.com/
[7] https://next-safe-action.dev/

akcí a může být použit pro různé účely, jako je kontrola autentizace uživatele, logování požadavků, či manipulace s daty požadavku.

5.5 Rozhraní pro žadatele

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

5.5.1 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.

5.5.2 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.

5.5.3 Formulář pro vytvoření přihlášky

Formulář pro vytvoření přihlášky je dostupný po kliknutí na tlačítko *Nová přihláška* v sekci *Moje přihlášky*. K formuláři lze přistoupit pouze v případě, že nejsou splněny žádné podmínky pro zamezení přístupu k formuláři (např. uzávěrka přihlášek, či globální zamezení přístupu pro uživatele s rolí `guest`).

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.

5.5.4 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.

5.6 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.

5.6.1 Nastavení chování aplikace

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

5.6.1.1 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.

5.6.1.2 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.

5.6.1.3 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).

5.6.1.4 Sekce „Účty“

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

5.6.2 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.

5.6.2.1 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.

5.6.2.2 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 běžné, ani efektivní. Aktuálně existují 2 možné řešení pro generování PDF z předhotovené šablony: přes vyplňování buněk v XLSX a přes vyplňování formulářových polí přímo v PDF. V jednoduchosti, efektivitě zpracování i kvalitě výsledného PDF se ukázalo druhé řešení jako lepší, a proto také bylo zvoleno pro implementaci této funkce. Pro tento účel byla zvolena knihovna PDF-LIB[8], která umožňuje jednoduchou manipulaci s PDF dokumenty, včetně vyplňování formulářových polí. Celý proces generování PDF probíhá následovně:

- Předem je vytvořena šablona PDF, která obsahuje formulářová pole pro všechny potřebné údaje z přihlášky. Tato pole jsou předem definována webovou aplikací. Šablona je nahrána do systém přes nastavení aplikace.
- Pro vygenerování PDF konkrétní přihlášky, či sady přihlášek lze použít tlačítko pro export do PDF, které se nachází jak ve stránce pro detaily přihlášky, tak i v přehledu přihlášek. Po kliknutí na tlačítko se spustí proces generování PDF, který načte šablonu PDF, vyplní formulářová pole daty z přihlášky a vygeneruje výsledný PDF dokument, který je následně odeslán uživateli ke stažení.

Interně je pro generování PDF vytvořena speciální API, která přijímá ID přihlášky (nebo skupiny ID) a vrací vyplněný PDF dokument. Všechny PDF jsou zpracovány v separatním vlákně (*worker thread*), aby nedocházelo k blokování hlavního vlákna serveru a tím i zhoršení výkonu aplikace. Všechny přihlášky se zároveň cachují do lokálního úložiště, aby se zrychlil přístup k již vygenerovaným PDF dokumentům (cache se automaticky přemazává při případných změnách přihlášek).

Hlavní výhodou užití knihovny PDF-LIB oproti řešení s XLSX je fakt, že XLSX řešení má podstatně horší časovou komplexitu ($O(C \cdot R)$), zatímco PDF-LIB nabízí konstantní časovou komplexitu ($O(1)$) pro vygenerování jednoho PDF dokumentu, což je zásadní pro zajištění rychlé odezvy aplikace. Při testování obou řešení se ukázalo, že průměrná odezva pro vygenerování PDF pomocí PDF-LIB byla přibližně 600 ms, zatímco pro řešení s XLSX to bylo přibližně 5.5 sekundy.

[8] https://pdf-lib.js.org/

5.6.3 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.

5.6.4 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 7: Příklad evidenčního čísla přihlášky (první přihláška v roce 2025)

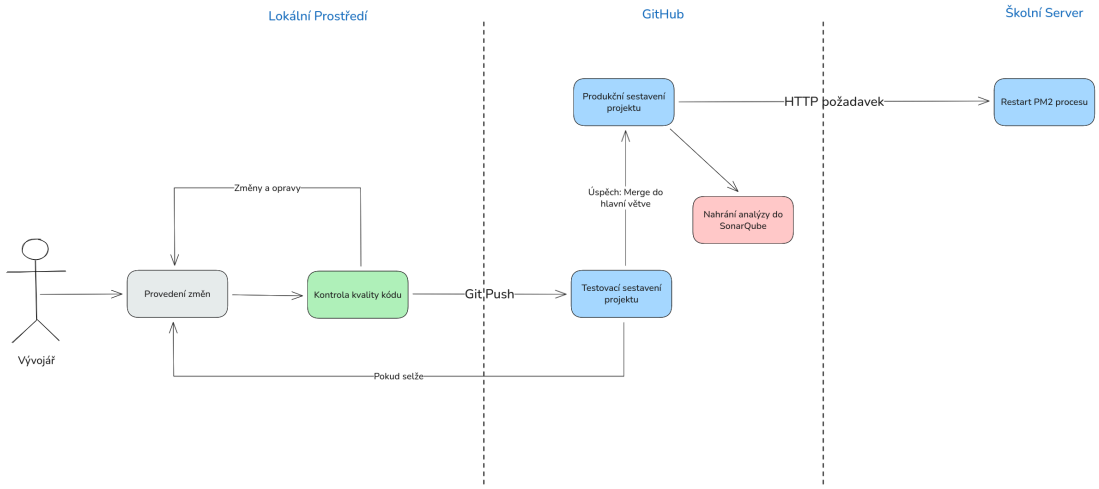
V programu lze importovat existující evidenční čísla pro předem definovaná rodná čísla. Tento import je užitečný v případech, kdy docházelo k tvoření evidenčních čísel mimo systém (např. ručně) a je potřeba tato čísla synchronizovat s databází aplikace. Import probíhá pomocí CSV souboru, který obsahuje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

5.6.5 Massmail

Massmail (neboli hromadné e-maily) 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. K funkci lze přistoupit pomocí bočního navigačního menu. E-maily nabízí základní funkce formátování, jako je tučný text, kurzíva, odrážky a další.

6 Vývoj a nasazení

Při tvorbě projektu by měl být kladen důraz na efektivní vývojový proces, který zahrnuje nástroje pro jednodušší lokální vývoj, zajištění kvality kódu, monitorování aplikace po nasazení a automatizaci nasazení. V následující kapitole je proto popsán přesný vývojový proces, nástroje použité pro zajištění kvality kódu a samotný automatizovaný proces nasazení aplikace na produkční server.



Obrázek 8: Schéma procesu vývoje aplikace

6.1 Vývojové nástroje

6.1.1 Docker

Ve vývojovém prostředí lze aplikaci i služby nutné pro její běh (např. databáze) spustit pomocí Dockeru. Pro tento účel je v kořenovém adresáři projektu umístěn soubor `docker-compose.dev.yml`, který definuje potřebné služby a jejich konfiguraci. Tento soubor lze použít pro rychlé a jednotné spuštění databáze Postgres a podpůrného SMTP serveru MailHog.

Společně se souborem `docker-compose.dev.yml` je kořenovém adresáři projektu umístěn i soubor `Dockerfile`, který definuje sestavení samotné aplikace do Docker obrazu. Tento obraz je následně použit v `docker-compose.yml` pro spuštění aplikace v produkčním prostředí. Nutno podotknout, že na produkčním serveru je aplikace spuštěna přes *LXC kontejner*, nikoliv jako Docker kontejner. Toto rozhodnutí bylo učiněno na základě konzultace s administrátorem lokálního školního serveru.

6.1.2 Užití statických analýzátorů kódu

Pro zajištění kvality kódu, dodržování standardů programování a odhalování potenciálních chyb byly do vývojového procesu integrovány *nástroje pro statickou analýzu kódu*. Tyto nástroje analyzují zdrojový kód bez jeho spuštění a poskytují zpětnou vazbu vývojářům o možných problémech, jako jsou chyby v syntaxi, nedodržování konvencí, či potenciální bezpečnostní rizika, tímto se liší od analyzérů dynamických, jako jsou testy, nástroje pro analýzu výkonu (profilery) nebo ladicí nástroje (debuggery).

6.1.2.1 Biome

Biome je moderní nástroj pro statickou analýzu kódu, který podporuje různé programovací jazyky, včetně TypeScriptu. Biome nabízí funkce, jako je formátování kódu podle předem stanovených pravidel, detekce chyb v syntaxi, analýza kvality kódu podle definovaných standardů pro statickou analýzu webovách aplikací[9] a další [18].

Součástí lokální konfigurace Biome je i integrace s verzovacím systémem Git pomocí *pre-commit hooku*. Z názvu je patrné, že se jedná o skript, nebo jinou akci, která bude vykonána před tím, než dojde k vytvoření nového *commitu* v lokálním repoziráři. V případě Biome je tento *hook* použit pro automatické spuštění analýzy kódu a případné opravy nalezených problémů.

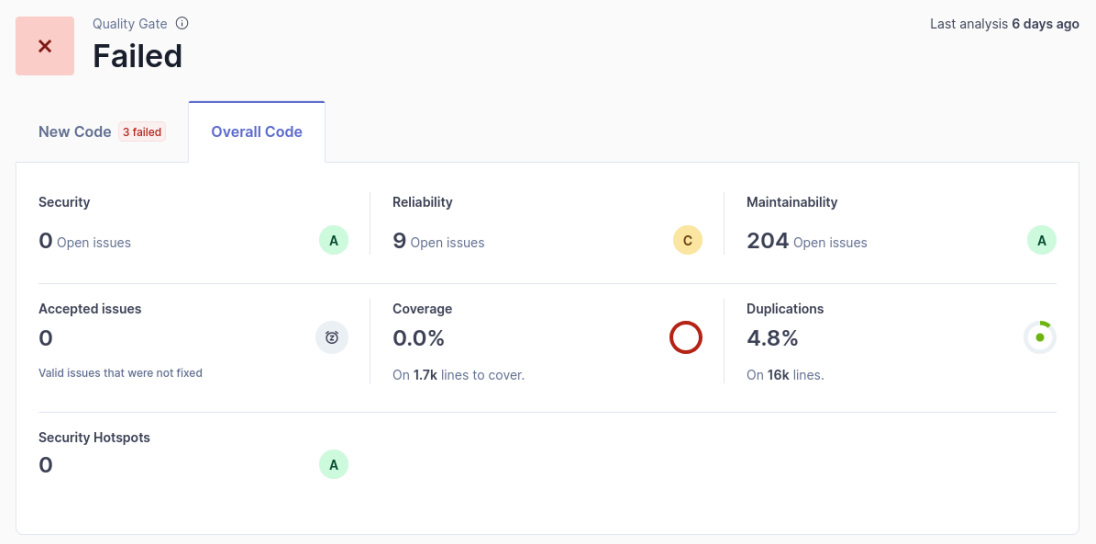
Další funkcí Biome je integrace s editory a vývojovými prostředími (jako je Visual Studio Code, nebo JetBrains WebStorm) pomocí rozšíření. Tato integrace umožňuje vývojářům získávat zpětnou vazbu o kvalitě kódu přímo během psaní kódu, což usnadňuje dodržování standardů a zlepšuje kvalitu kódu již v raných fázích vývoje, ještě než je samotný nástroj spuštěn manuálně.

Celé nastavení Biome je uloženo v kořenovém adresáři projektu v souboru `biome.json`, kde jsou definována pravidla pro analýzu kódu, formátování a další.

6.1.2.2 SonarQube

SonarQube je platforma pro inspekci kvality kódu v softwarových projektech. Podporuje širokou škálu programovacích jazyků, včetně TypeScriptu, a nabízí funkce jako je analýza kódu, detekce chyb, sledování metrik kvality a další [19].

[9] Tato pravidla jsou často převzata z jiných nástrojů, jako je ESLint. Kompletní seznam pravidel a jejich odůvodnění lze nalézt v oficiální dokumentaci Biome.



Obrázek 9: Souhrn analýzy zdrojového kódu v SonarQube

Pro integraci SonarQube do vývojového procesu byl SonarQube nasazen na vlastní server a byl vytvořen nový projekt. Pro integraci mezi repozitářem na platformě GitHub a SonarQube byla vytvořena nová *GitHub Action*, tedy automaticky vykonaný činnost. Ta je nastavena, aby se spustila při každém aktualizace zdrojového kódu na hlavní větvi repozitáře. Tato akce provede analýzu kódu pomocí SonarQube a výsledky jsou následně odeslány na server SonarQube, kde jsou dostupné pro další přehledy a analýzy.

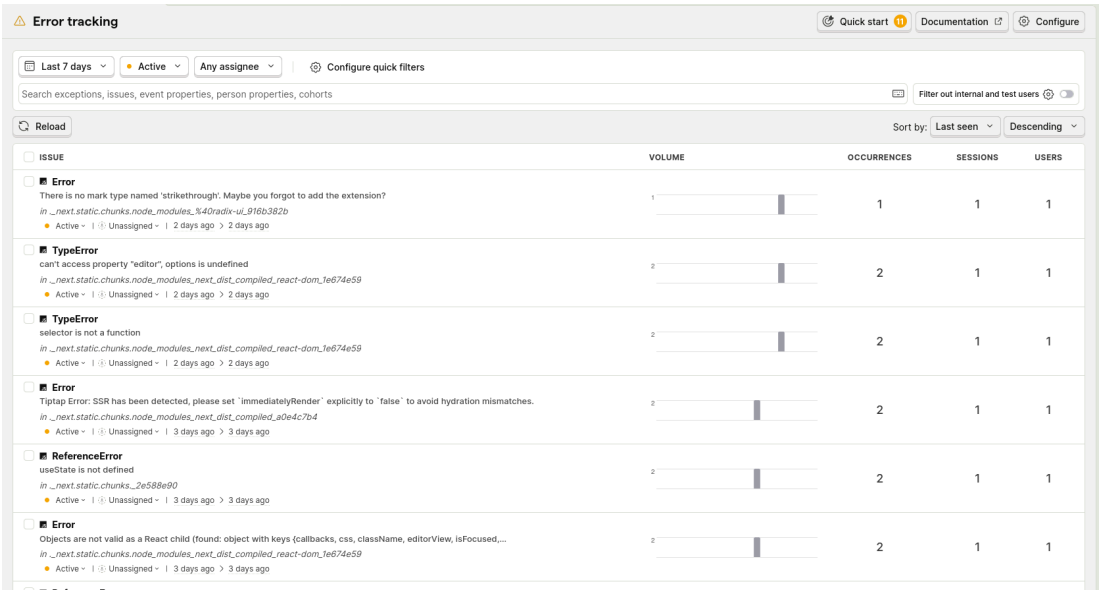
6.1.3 Aktivní monitorování

Mezi nástroje pro aktivní monitorování webových aplikací řadíme nástroje, které aktivně sledují chování aplikace v reálném čase a poskytují zpětnou vazbu o výkonu, chybách a dalších důležitých aspektech aplikace. Tyto nástroje pomáhají vývojářům identifikovat a řešit problémy (například sledováním kroků uživatele, které vedly k vyvolání chyby), optimalizovat výkon a zlepšovat uživatelskou zkušenost za běhu aplikace.

6.1.3.1 PostHog

PostHog je open-source platforma pro analýzu chování uživatelů a aktivní monitorování webových aplikací. Umožňuje vývojářům sledovat interakce uživatelů, analyzovat chování aplikace, zajišťovat neodladěné chyby a získávat cenné informace pro optimalizaci výkonu a zlepšení uživatelské zkušenosti [20].

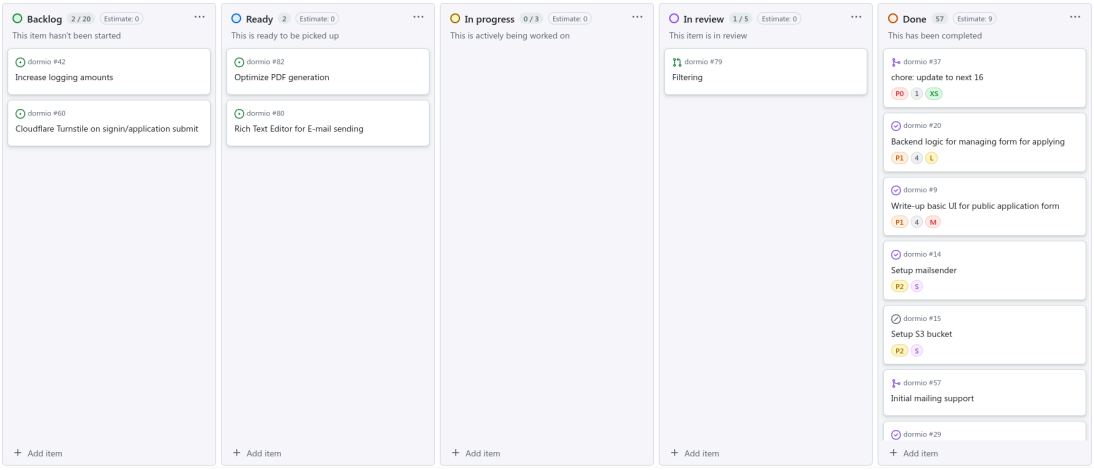
PostHog byl do vývojového procesu integrován použitím oficiální knihovny. Hlavním cílem integrace bylo vytvořit možnost získávání zpětné vazby o chování uživatelů a sledování chyb, které by mohly nastat během používání aplikace. Tato integrace umožňuje lépe porozumět chybám a problémům, které uživatelé mohou zažívat. Integrací zároveň pomůžeme rychleji problémy opravovat a případně i předcházet jejich vzniku v budoucnu.



Obrázek 10: Panel pro sledování nastalých problémů při běhu projektu v platformě PostHog

6.2 Plánování vývoje pomocí GitHub Projects

Správné plánování vývoje je jednou z klíčových činností pro úspěšný a nezanedbaný vývoj jakéhokolik softwarového projektu. Pro tento účel byla zvolena platforma GitHub Projects, která umožňuje vytváření projektů, úkolů a sledování jejich stavu přímo v rámci repozitáře na GitHubu [21].



Obrázek 11: Plánovací tabulka pro vývoj aplikace na platformě GitHub Projects

Struktura plánovací tabulky byla rozdělena do několika sloupců:

- *Backlog* – sloupec pro úkoly, které jsou naplánovány, ale ještě nebyly zahájeny a aktuálně nejsou žádné kapacity pro jejich řešení.
- *Ready* – sloupec pro úkoly, které jsou připraveny k řešení a čekají na zahájení práce.
- *In Progress* – sloupec pro úkoly, na kterých se aktuálně pracuje.
- *In Review* – sloupec pro úkoly, které byly dokončeny a čekají na schválení (např. kontrolou kódu).
- *Done* – sloupec pro úkoly, které byly dokončeny a schváleny.

Každému z úkolů je propojen s příslušnou *issue* (problémem) nebo *pull requestem* (žádostí o sloučení kódu) v repozitáři, což umožňuje snadné sledování pokroku a stavu jednotlivých úkolů přímo z plánovací tabulky. Zároveň je ke každému úkolu možní přiřadit různé typy štítků (labels), které pomáhají kategorizovat úkoly podle jejich povahy (např. bug, feature, enhancement) a priority.

Samostatné *issues* a *pull requesty* jsou pak spravovány pomocí standardních funkcí platformy GitHub a lze je opět kategorizovat do štítků, ty ale nyní slouží pro identifikaci typu problému či změny, kterou daný *issue* nebo *pull request* řeší (například problém se štítkem `ui/ux` značí problém, který se týká uživatelského rozhraní).

6.3 Verzovací systém Git

Git je distribuovaný verzovací systém, který umožňuje sledování změn v souborech a koordinaci práce mezi více vývojáři na jednom projektu [22]. Pro tento projekt byl

zvolen Git jako hlavní nástroj pro správu verzí kódu, jelikož je nabízí širokou škálu funkcí pro efektivní spolupráci a správu kódu, zároveň je dobře integrován s platformou GitHub, která slouží jako hostitelská služba pro repozitář projektu. Git zároveň patří mezi nejrozšířenější verzovací systémy, které jsou v současnosti používány ve vývoji softwaru [5].

6.4 Nasazení na produkční server

Nasazení aplikaci na produkční systém probíhá pomocí automazivaného procesu, který je spuštěn při každé aktualizaci hlavní větve v repozitáři na platformě GitHub. Tento proces je implementován pomocí *GitHub Actions*, což je nástroj pro automatizaci pracovních postupů přímo v rámci platformy GitHub [23].

V prvním kroku procesu nasazení je spuštěna akce, která ověří spustitenost kódu testovacím sestavením. Pokud je sestavení úspěšné, akce odešle HTTP požadavek na otevřenou API jednoduchého webového serveru běžícího na produkčním serveru. Tento požadavek slouží jako spouštěč skriptu, který je zodpovědný za znovunasazení aplikace. Skript následně vykoná následující kroky, přesně v tomto pořadí:

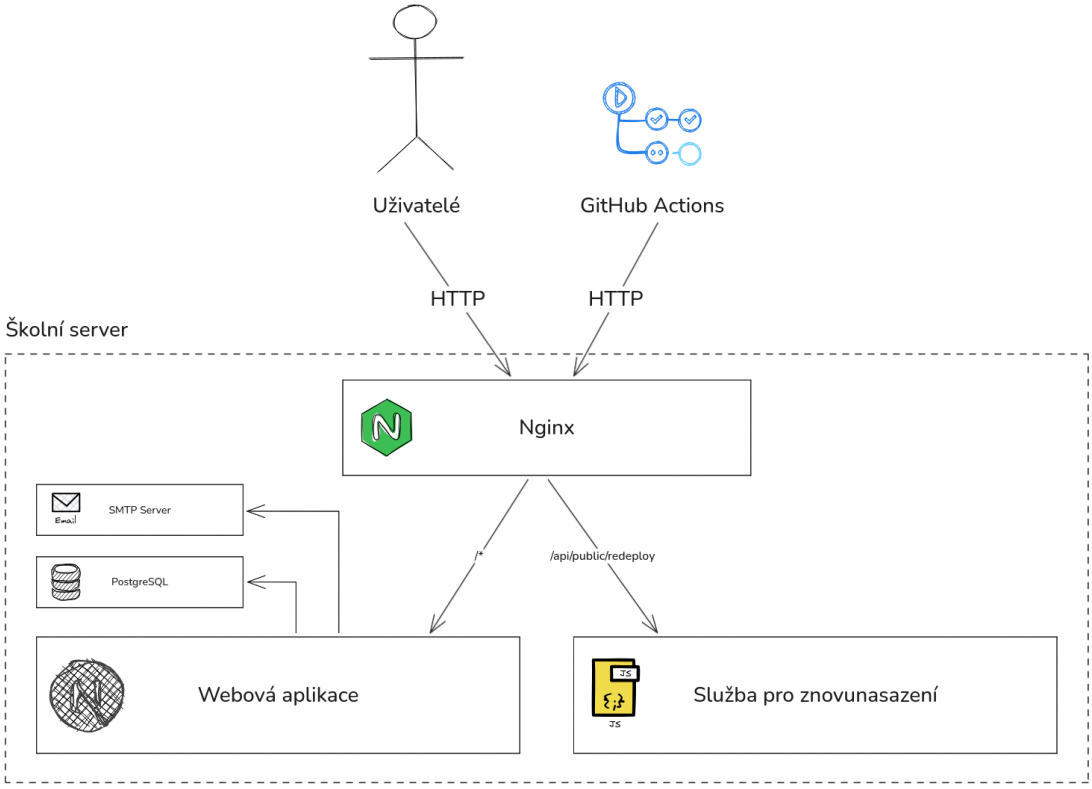
- Zastaví běžící instance aplikace (pokud existuje), vypne pm2 proces.
- Aktualizuje zdrojový kód aplikace uložený na serveru pomocí `git pull` pro získání nejnovějších změn z repozitáře.
- Nainstaluje všechny potřebné závislosti (pokud došlo k jejich změně).
- Aplikuje případné změny v databázovém schématu pomocí migrací.
- Nastartuje aplikaci pomocí pm2 a znovu ji zpřístupní uživatelům.

6.4.1 Architektura produkčního serveru

Produkční server je hostován jako LXC kontejner na hlavním školním serveru. Tento kontejner je nakonfigurován tak, aby poskytoval izolované prostředí pro běh aplikace, což zajišťuje bezpečnost a stabilitu aplikace. Kontejner obsahuje všechny potřebné závislosti a konfigurace pro běh aplikace, včetně databázového serveru PostgreSQL a přístupu na SMTP server pro odesílání e-mailů přes školní doménu.

Pro přístup k aplikaci z veřejné sítě je na školním serveru nastavena služba Nginx, která poskytuje reverzní proxy pro směrování HTTP požadavků na správný port, na kterém aplikace běží. V případě služby pro znovunasazení je pak v konfiguraci reverzní

proxy nastaveno směrování požadavků na specifickou cestu a přístup je omezen na klíč pro ověření. Pro bezpečnostní účely je také implementována cesta pro *kill-switch*, která umožňuje pouhým posláním HTTP požadavku na specifickou cestu okamžitě vzdáleně zastavit běh aplikace, což je užitečné v případě zjištění závažné chyby, nebo bezpečnostního incidentu.



Obrázek 12: Schéma architektury produkčního serveru

Závěr

Cílem této maturitní práce bylo představit systém Dormio. Jedná se o webovou aplikaci určenou pro správu přijímacího řízení do domovů mládeže a internátních zařízení. Práce se soustředila na návrh, implementaci a nasazení této aplikace s důrazem na moderní technologie a osvědčené postupy ve vývoji softwaru. Cílem také bylo zajistit, aby aplikace byla uživatelsky přívětivá jak pro žadatele, tak i pro vychovatele, a aby splňovala požadavky na bezpečnost a spolehlivost.

Při vývoji jsem získal cenné zkušenosti s návrhem databázových modelů, uživatelských rozhraní, implementací kódu na straně klienta i serveru a také se správou životního cyklu vývoje softwaru. Samotná práce mě pak naučila lépe přemýšlet o

architektuře webových aplikací a výběru vhodných technologií pro konkrétní úkoly. Věřím tedy, že se tento projekt ověří jako užitečný nástroj pro domovy mládeže a internátní zařízení.

Plně doufám v to, že tento projekt bude nadále udržován a rozvíjen tak, aby se stal univerzálním systémem pro správu domovů mládeže a jeho funkcionalita se rozšiřovala o další užitečné funkce, které by mohly usnadnit práci vychovatelům.

Seznam použité literatury

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

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

[3] OKTA, INC. Introduction to JSON Web Tokens. online. 2026. Dostupné z: https://www.jwt.io/introduction#what-is-json-web-token-structure.

[4] MICROSOFT. TypeScript: JavaScript with syntax for types. online. Dostupné z: https://www.typescriptlang.org/.

[5] STACK OVERFLOW. *Stack Overflow Developer Survey 2025*. online. 2025. https://survey.stackoverflow.co/2025/.

[6] MICROSOFT INC. *TypeScript Declaration Files: Do's and Don'ts*. online. Dostupné z: https://www.typescriptlang.org/docs/handbook/declaration-files/do-s-and-don-ts.html. [citováno 2026-02-11].

[7] ECMA INTERNATIONAL. *ECMAScript®: The Standardized Scripting Language for Web Development*. online. 2026. https://www.ecma-international.org/publications-and-standards/standards/ecma-262/.

[8] MICROSOFT INC. *TypeScript Compiler Options: Target*. online. Dostupné z: https://www.typescriptlang.org/tsconfig/#target. [citováno 2026-02-11].

[9] ECMA INTERNATIONAL. *ECMA-262: ECMAScript® 2015 Language Specification*. online. 2015. Dostupné z: https://www.ecma-international.org/ecma-262/6.0/.

[10] VERCEL INC. Next.js: The React Framework for the Web. online. Dostupné z: https://nextjs.org/.

[11] META PLATFORMS, INC. React: The library for web and native user interfaces. online. Dostupné z: https://react.dev/.

[12] META PLATFORMS, INC. React Server Components. online. Dostupné z: https://react.dev/reference/rsc/server-components.

[13] VERCEL INC. Server Actions in Next.js. online. Dostupné z: https://nextjs.org/docs/13/app/api-reference/functions/server-actions.

[14] PRISMA INC. Prisma: Next-generation ORM for Node.js and TypeScript. online. Dostupné z: https://www.prisma.io/orm.

[15] TAILWIND LABS, INC. online. Dostupné z: https://tailwindcss.com/.

[16] COLINHACKS. *Zod: TypeScript-first schema validation with static type inference*. online. 2023. https://zod.dev/.

[17] RANGHIERI, Edoardo. online. Dostupné z: https://next-safe-action.dev/.

[18] BIOME DEVELOPERS AND CONTRIBUTORS. Biome, toolchain of the web. online. Dostupné z: https://biomejs.dev/.

[19] SONARSOURCE SA. SonarQube: Code quality and security. online. Dostupné z: https://www.sonarsource.com/products/sonarqube/.

[20] POSTHOG, INC. PostHog: We make dev tools for product engineers. online. Dostupné z: https://posthog.com/.

[21] GITHUB, INC. GitHub Projects: Project planning for developers. online. Dostupné z: https://github.com/features/project-management/.

[22] TORVALDS, Linus a Junio C. HAMANO. *Git: Fast Version Control System*. online. Dostupné z: https://git-scm.com/. [citováno 2026-01-28].

[23] GITHUB, INC. GitHub Actions: Automate, customize, and execute your software development workflows right in your repository. online. 2026. Dostupné z: https://github.com/features/actions.

Seznam obrázků

Seznam tabulek