

Webová aplikace pro správu harmonogramu zotavovacích akcí

Pavel Bařina



Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Pavel Bařina
Osobní číslo: A22466
Studijní program: B0613A140020 Softwarové inženýrství
Forma studia: Prezenční
Téma práce: Webová aplikace pro správu harmonogramu zotavovacích akcí
Téma práce anglicky: Web Application for Managing Recovery Event Schedules

Zásady pro vypracování

- Vypracujte literární rešerši na dané téma.
- Zvolte vhodný framework pro vývoj webové aplikace a databáze.
- Vytvořte grafický návrh uživatelského rozhraní aplikace a navrhnete databázovou strukturu pro ukládání dat.
- Implementujte funkcionalitu pro tvorbu, správu a organizaci harmonogramů s možností jejich exportu.
- Věnujte pozornost zabezpečení webové aplikace.

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

Seznam doporučené literatury:

1. SCAMBRAY, Joel a SHEMA, Mike. *Hacking bez tajemství: webové aplikace*. Computer Press, 2003. ISBN 80-7226-769-8.
2. WELLING, Luke a KUKLÍNEK, Jan. *PHP a MySQL rozvoj webových aplikací*. 3. vyd. Softpress, 2005. ISBN 80-86497-83-6.
3. DOMES, Martin. *Tvorba internetových stránek pomocí HTML, CSS a JavaScriptu*. Computer Press, 2004. ISBN 80-86686-39-6.
4. TIDWELL, Jenifer; BREWER, Charles a VALENCIA, Aynne. *Designing Interfaces*. 3rd Edition. O'Reilly Media, 2020. ISBN 9781492051961.
5. MCDONALD, Malcolm. *Web Security for Developers*. No Starch Press, 2020. ISBN 1593279949.
6. SONMEZ, John. *The Complete Software Developer's Career Guide*. Simple Programmer, 2017. ISBN 9780999081419.

Vedoucí bakalářské práce: **Ing. Tomáš Kadavý, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **27. října 2024**
Termín odevzdání bakalářské práce: **2. června 2025**



L.S.

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

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

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA BAKALÁŘSKÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení bakalářské práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – bakalářskou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem řádně citoval; v případě publikace výsledků budu uveden jako spoluautor;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.
- při tvorbě této práce jsem použil nástroj generativního modelu AI ChatGPT 4.0, dostupné na <https://chat.openai.com> za účelem optimalizace kódu a lepšího formulování textu. Po použití tohoto nástroje jsem provedl kontrolu obsahu a přebírám za něj plnou zodpovědnost.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Planora je webová aplikace určená k efektivnímu plánování harmonogramů zotavovacích akcí. Aplikace je postavena na moderním PHP (Hypertext Preprocessor) frameworku Laravel a využívá relační databázový model MySQL (Structured Query Language) pro ukládání a správu dat. Hlavní funkce zahrnují intuitivní uživatelské rozhraní umožňující vytváření, editaci, mazání a správu jednotlivých harmonogramů. Uživatelé mohou harmonogramy sdílet mezi více uživateli a exportovat je do PDF (Portable Document Format) formátu pro případný tisk či offline použití. Díky webovému rozhraní je Planora dostupná na široké škále zařízení, včetně počítačů, notebooků, tabletů a mobilních telefonů. Aplikace tak poskytuje komplexní řešení pro efektivní organizaci a koordinaci zotavovacích akcí.

Klíčová slova: harmonogram, Laravel, MySQL, Planora, plánovač, webová aplikace

ABSTRACT

Planora is a web-based application designed for efficient planning of recovery schedules. The application is built on the modern PHP framework Laravel and uses the MySQL relational database model for data storage and management. The main functionalities include an intuitive user interface that allows creating, editing, deleting and managing individual schedules. Users can share schedules between multiple users and export them to PDF format for printing or offline use. Thanks to its web-based interface, Planora is accessible on a wide range of devices, including PCs, laptops, tablets and mobile phones. The app thus provides a comprehensive solution for the efficient organisation and coordination of recovery events.

Keywords: schedule, Laravel, MySQL, Planora, planner, web application

Rád bych na tomto místě poděkoval vedoucímu mé bakalářské práce, Ing. Tomáši Kadavému, Ph.D., za jeho odborné vedení, vstřícnost a ochotu kdykoliv pomoci nebo poradit během celé doby vzniku této práce.

Mé poděkování patří také mé rodině, spolužákům a přítelkyni za jejich neutuchající podporu a motivaci po celou dobu studia.

V neposlední řadě bych chtěl vyjádřit své díky všem, kteří se podíleli na testování vytvořené aplikace a poskytli mi cennou zpětnou vazbu, jež byla klíčová pro úspěšné dokončení tohoto projektu.

OBSAH

ÚVOD.....	13
I TEORETICKÁ ČÁST	14
1 ZOTAVOVACÍ AKCE A JEJICH ORGANIZACE	15
1.1 Co JE POTŘEBA DODRŽET PŘI ORGANIZACI ZOTAVOVACÍ AKCE	15
1.1.1 Ohlašovací povinnost	15
1.1.2 Hygienické požadavky	15
1.1.3 Zdravotní zabezpečení	16
1.1.4 Personální požadavky.....	17
1.2 POTŘEBA EFEKTIVNÍHO PLÁNOVÁNÍ A HARMONOGRAMŮ	17
2 ANALÝZA A VÝBĚR FRAMEWORKU	19
2.1 KRITÉRIA PRO VÝBĚR FRAMEWORKU	19
2.2 ANALÝZA JEDNOTLIVÝCH FRAMEWORKŮ.....	20
2.2.1 Laravel.....	20
2.2.2 Angular.....	23
2.2.3 React.....	26
2.2.4 Vue.js	29
2.3 ZVOLENÍ TECHNOLOGIE PRO VÝVOJ	32
3 POUŽITÉ TECHNOLOGIE.....	34
3.1 LARAVEL.....	34
3.1.1 Historie Laravelu.....	34
3.1.2 Nastavení vývojového prostředí a vytvoření projektu	35
3.1.3 Routování a controllery	37
3.1.4 Blade templates pro frontend	41
3.2 MYSQL	41
3.2.1 Instalace MySQL	42
3.2.2 Základní operace	43
II PRAKTICKÁ ČÁST	46
4 ANALÝZA POŽADAVKŮ	47
4.1 CÍLOVÁ SKUPINA UŽIVATELŮ	47
4.2 FUNKČNÍ A NEFUNKČNÍ POŽADAVKY APLIKACE	47
4.2.1 Funkční požadavky	47
4.2.2 Nefunkční požadavky.....	48
4.3 NÁVRH UŽIVATELSKÝCH ROLÍ.....	49
5 NÁVRH APLIKACE	50
5.1 DATABÁZOVÝ MODEL	50
5.2 UŽIVATELSKÉ ROZHRANÍ	54

5.2.1	Přihlašování	54
5.2.2	Registrace	55
5.2.3	Zapomenuté heslo a obnovení hesla.....	56
5.2.4	Kontaktní formulář	56
5.2.5	Uživatelský profil	57
5.2.6	Administrace	58
5.2.7	Dashboard.....	61
6	IMPLEMENTACE.....	63
6.1	ARCHITEKTURA SYSTÉMU	63
6.1.1	Vytvoření projektu	63
6.1.2	Architektura MVC.....	63
6.1.3	Struktura projektu.....	64
6.2	BACKENDOVÁ IMPLEMENTACE	64
6.2.1	Autentizace a registrace uživatelů	65
6.2.2	Uživatelské role a oprávnění	65
6.2.3	Databázová logika a Eloquent ORM.....	65
6.2.4	Uzamykání harmonogramu	65
6.2.5	Administrační část.....	66
6.3	FRONTENDOVÁ IMPLEMENTACE	67
6.3.1	Hlavní layout a struktura stránky	67
6.3.2	Navigační menu.....	67
6.3.3	Dashboard a zobrazení harmonogramu	68
6.3.4	Uzamykání harmonogramu	69
6.3.5	Sdílení a export harmonogramu	69
6.3.6	Administrační část.....	70
6.4	ZABEZPEČENÍ APLIKACE	70
6.4.1	Autentizace a autorizace uživatelů	71
6.4.2	Bezpečné ukládání hesel	71
6.4.3	Řízení přístupu a oprávnění.....	71
6.4.4	Ochrana proti útokům.....	71
6.4.5	Zamykání harmonogramů	72
6.5	TESTOVÁNÍ APLIKACE	72
6.6	NASAZENÍ APLIKACE	73
7	UŽIVATELSKÁ DOKUMENTACE.....	74
7.1	REGISTRACE A PŘIHLÁŠENÍ.....	74
7.2	VYTVÁŘENÍ A SPRÁVA HARMONOGRAMU	74
7.3	SDÍLENÍ HARMONOGRAMU	74
7.4	EXPORT DO PDF.....	74
7.5	UŽIVATELSKÝ PROFIL.....	75
7.6	KONTAKTNÍ FORMULÁŘ	75
7.7	ADMINISTRACE SYSTÉMU	75
	ZÁVĚR.....	76

SEZNAM POUŽITÉ LITERATURY.....	77
SEZNAM OBRÁZKŮ	79
SEZNAM TABULEK.....	80
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	81
SEZNAM PŘÍLOH.....	82

ÚVOD

Pro úspěšné řízení různých typů akcí, včetně akcí zotavovacích, jsou klíčovými prvky efektivní plánování a organizace času. Tyto akce vyžadují pečlivě sestavené harmonogramy, díky kterým je možné spolehlivě koordinovat celý chod akce. Tradiční metody plánování, jako jsou papírové diáře, kalendáře či nástěnky, často narážejí na své limity, zejména pokud jde o možnosti sdílení s více lidmi a zachování přehlednosti. Na popud těchto nedostatků byla vytvořena webová aplikace Planora, která slouží jako moderní nástroj pro správu harmonogramů zotavovacích akcí.

Motivací k vývoji této aplikace byla potřeba efektivnějšího nástroje pro plánování a správu harmonogramů zotavovacích akcí, která vzešla z osobní zkušenosti s organizací těchto akcí. Současné způsoby postrádají potřebnou flexibilitu a přehlednost. Dalším důvodem bylo získání nových zkušeností s moderními technologiemi určenými pro vývoj moderních webových aplikací, především s frameworkem Laravel. Tyto zkušenosti vedly k prohloubení znalostí v oblasti webového vývoje.

Teoretická část této práce se zaměřuje na problematiku plánování a organizace zotavovacích akcí. Nejprve jsou definovány základní pojmy související se zotavovacími akcemi. Následuje analýza současných řešení využívanými v praxi a identifikace jejich nedostatků, které vedli k vývoji nové aplikace. Důležitou součástí je také analýza a výběr vhodného frameworku pro vývoj webové aplikace. Jsou zde porovnány současné nejpoužívanější frameworky a následně jsou zhodnoceny jejich pozitiva a negativa. V neposlední řadě zde jsou popsány technologie, které byly využity pro vývoj aplikace.

Praktická část se zaměřuje na samotný proces vývoje webové aplikace Planora. Je zde popsáno, jak probíhal návrh a následně samotná implementace aplikace. Taktéž zde jsou představeny základní funkce, kterými aplikace disponuje a rovněž je věnována pozornost zabezpečení aplikace.

Cílem této bakalářské práce je navrhnout a implementovat webovou aplikaci Planora pro efektivní plánování harmonogramů zotavovacích akcí. Aplikace by měla poskytnout uživatelsky přívětivé rozhraní, které je dostatečně bezpečné a umožňuje správu, sdílení a exportování harmonogramů.

I TEORETICKÁ ČÁST

1 ZOTAVOVACÍ AKCE A JEJICH ORGANIZACE

Zotavovací akce je dle českého právního řádu definována jako organizovaný pobyt 30 a více dětí ve věku do 15 let na dobu delší než 5 dnů, jehož účelem je posílit zdraví dětí, zvýšit jejich tělesnou zdatnost nebo získat specifické znalosti či dovednosti. [1]

Za zotavovací akci lze považovat letní dětské tábory, školy v přírodě nebo různá sportovní soustředění.

1.1 Co je potřeba dodržet při organizaci zotavovací akce

Při organizaci zotavovací akce v České republice je nutné dodržovat několik legislativních a hygienických požadavků. Dodržením těchto požadavků by měla být zajištěna bezpečnost a zdraví všech účastníků takové akce. V následujících kapitolách jsou shrnuty nejpodstatnější body, které je potřeba při organizaci zotavovací akce dodržet.

1.1.1 Ohlašovací povinnost

Nejpozději jeden měsíc před zahájením zotavovací akce je pořadatel povinný informovat příslušný orgán ochrany veřejného zdraví (krajskou hygienickou stanici) o následujících skutečnostech. [2]

- Termín a místo konání zotavovací akce
- Počet dětí zúčastněných na zotavovací akci
- Způsob jejího zabezpečení pitnou vodou
- Způsob zajištění stravování účastníků zotavovací akce

1.1.2 Hygienické požadavky

Hygienické zajištění zotavovací akce lze považovat za nejdůležitější bod, na který musí každý organizátor takové akce myslet. Pro zajištění bezpečí a zdraví všech účastníků je potřeba dodržovat několik norem pro ubytování, stravování, osobní hygienu a sanitární zařízení. [1]

Ubytování

Prostor určený k ubytování dětí by měl být dostatečně velký, dobře větraný a musí umožňovat bezpečný a komfortní pohyb. V případě stanových táborů nebo jiného přírodního

ubytování je potřeba stavby dostatečně ochránit proti nepříznivým povětrnostním podmínkám a škůdcům. [1]

Stravování

Při přípravě stravy přímo na místě konání zotavovací akce musí být k dispozici vhodné kuchyňské zázemí, které jsou zastřešené. Patří sem jídelna, kuchyně a sklad. Tyto prostory musí být od sebe odděleny. Veškeré potraviny musí být pečlivě skladovány a je nezbytné dodržovat přísné teplotní a skladovací podmínky, aby nedocházelo ke kontaminaci jídla. Navíc musí být všechny potraviny zdravotně nezávadné a musí odpovídat zásadám zdravé výživy dětí. [1]

Osobní hygiena

Pro umožnění optimální osobní hygieny je na zotavovacích akcích potřeba dodržovat několik hygienických podmínek. V umývárně je potřeba zajistit jeden výtokový kohout nebo umyvadlo na 5 osob. Při sprchování je nutné zajistit, aby se osoby sprchovaly odděleně podle pohlaví. Maximální počet osob na jednu sprchovou hlavici je 30 a osoba by měla mít možnost se osprchovat alespoň jedenkrát týdně. [1]

Sanitární zařízení

Velmi důležitým požadavkem při organizaci zotavovacích akcí je zajištění dostatečného počtu toalet. Obecně platí, že na každých 15 osob musí připadat alespoň jedna toaleta. Toalety musí být oddělené podle pohlaví a vybavené dostatečným množstvím hygienických prostředků, jako jsou toaletní papír, mýdlo a voda na umytí rukou.

V případě používání latrín je nezbytné latríny každý den překrýt vrstvou zeminy a před opuštěním tábora je kompletně zasypat. Dále je nutné dodržovat minimální vzdálenosti, například záchody musí být umístěny alespoň 25 metrů od studny s pitnou vodou a minimálně 50 metrů od stravovacích zařízení.

1.1.3 Zdravotní zabezpečení

Zajištění zdravotního zabezpečení na zotavovacích akcích je klíčové pro bezpečnost a zdraví všech účastníků. Na každé zotavovací akci je nezbytné mít zdravotníka. Velký důraz je však třeba dbát i na zdravotní způsobilost účastníků.

Zdravotník

Zdravotníkem zotavovacích akcí může být osoba starší 18 let, která splňuje alespoň jednu z následujících podmínek [3]:

- Má způsobilost k výkonu zdravotnického povolání (lékař, zubní lékař, všeobecná sestra, porodní asistentka, zdravotnický záchranář nebo zdravotnický asistent).
- Je studentem oboru všeobecné lékařství nebo zubní lékařství po úspěšném ukončení třetího ročníku.
- Absolvovala akreditovaný kurz pro zdravotníky zotavovacích akcí a má platné oprávnění k výkonu funkce zdravotníka.

Zdravotník musí být taktéž zdravotně způsobilý k výkonu své funkce. Toto dokládá posudkem od lékaře, jehož platnost jsou 2 roky.

Zdravotní způsobilost účastníků

Před účastí na zotavovací akci je každý účastník povinen doložit posudek o zdravotní způsobilosti. Tento posudek zpravidla vystavuje obvodní lékař a má platnost 1 rok. Dále je nutné předložit vyplněné prohlášení o bezinfekčnosti podepsané zákonným zástupcem. Toto prohlášení potvrzuje, že účastník nejeví žádné známky onemocnění a v posledních 14 dnech nebyl ve styku s osobou nemocnou infekčním onemocněním. [4]

1.1.4 Personální požadavky

Vedoucí a personál zotavovacích akcí musí být tělesně i duševně způsobilí. Každý člen personálu se musí podrobit lékařské prohlídce a získat potvrzení o schopnosti vykonávat tuto činnost. Bez tohoto potvrzení nesmí žádná osoba na zotavovací akci ani vypomáhat. [1]

Výjimku z povinnosti lékařského potvrzení mají pouze pedagogičtí a zdravotničtí pracovníci. Osoby zapojené do přípravy a podávání stravy musí mít platný zdravotní průkaz. [1]

1.2 Potřeba efektivního plánování a harmonogramů

Základním kamenem úspěšně zorganizované zotavovací akce je efektivní plánování. Správně sestavený harmonogram je klíčový k hladkému průběhu akce, bezpečnosti všech účastníků a minimalizaci problémů, které by mohly vzniknout v důsledku nedostatečného plánování.

Absence dobře strukturovaného harmonogramu může vést k nedorozuměním mezi vedoucími, zbytečnému zdržování nebo dokonce hádkám. Je nutné, aby bylo vždy jasné definováno, kdy a kde se má daná aktivita odehrávat. Stejně tak je důležité, aby měla každá aktivita přiřazenou zodpovědnou osobu, která za její průběh nese odpovědnost. V ideálním

případě by každá aktivita měla mít jak suchou, tak mokrou variantu, aby bylo možné program přizpůsobit případnému nepříznivému počasí.

Nedostatečné plánování může způsobit vážnější problémy, například nepřípravenost na změny počasí, špatné načasování stravování, nevyhovující podmínky pro odpočinek nebo nedostatek času na přesuny mezi aktivitami. To může negativně ovlivnit nejen komfort a bezpečnost účastníků, ale i celkový průběh akce. Těmto komplikacím však lze předejít díky pečlivě připravenému harmonogramu.

2 ANALÝZA A VÝBĚR FRAMEWORKU

V této kapitole jsou analyzovány a porovnány populární frameworky současné doby používané pro vývoj moderních webových aplikací. Mezi ně patří Laravel, Angular, React nebo Vue.js, které se řadí mezi osvědčené technologie s velkým množstvím využití. V následujících podkapitolách budou tyto technologie hodnoceny. [5]

Hlavním cílem je vyhodnotit jejich silné a slabé stránky a na základě této analýzy vybrat vhodnou technologii, která bude využita pro vývoj webové aplikace pro správu harmonogramů zotavovacích akcí.

2.1 Kritéria pro výběr frameworku

Na začátku bylo stanoveno několik základních kritérií, kterými by měl vhodný framework disponovat, aby bylo docíleno adekvátní efektivity, výkonnosti a dlouhodobé udržitelnosti aplikace. Mezi hlavní kritéria patří [6]:

- Snadnost osvojení vlastností nové technologie a dostupná dokumentace: Díky kvalitní dokumentaci je jednodušší řešit dílčí problematiku při vývoji a celková potřeba času k vývoji a osvojení nových technologií se tím zkracuje.
- Výkonnost a škálovatelnost: Zvolený framework by měl efektivně pracovat s menšími ale i rozsáhlejšími aplikacemi a umožňovat snadné škálování podle potřeb rostoucí aplikace.
- Podpora pro fullstack vývoj: Možnost spravovat jak frontend, tak backend v rámci jednoho ekosystému je důležité kritérium, které bylo stanoveno před vývojem samotné aplikace. Některé frameworky se specializují pouze na jednu část aplikace, zatímco jiné poskytují komplexní řešení pro celý vývojový stack.
- Bezpečnost a údržba: Velkou roli při výběru vhodné technologie hraje její bezpečnost, resp. Frekvence aktualizací, zpětná kompatibilita verzí nebo vestavěné bezpečnostní mechanismy. Tyto prvky určují, jak dobře framework zvládá každodenní bezpečnostní výzvy a jak snadné je aplikaci udržovat zabezpečenou.
- Popularita a komunita: Aktivní a rozsáhlá komunita znamená lepší dostupnost podpory a rychlejší řešení problémů. Zároveň platí, že populární frameworky jsou častěji aktualizovány a mají širší ekosystém rozšíření a tím se zvyšuje jejich dlouhodobá udržitelnost.

2.2 Analýza jednotlivých frameworků

V této části budou podrobně rozebrány vybrané frameworky s ohledem na kritéria stanovená v předchozí kapitole. Každý framework bude stručně představen, popsán z hlediska svého využití v praxi a jeho silné a slabé stránky budou shrnuty do tabulek.

Následující podkapitoly se zaměří na Laravel, Angular, React a Vue.js, tedy technologie, které byly zvažovány pro vývoj aplikace.

2.2.1 Laravel

Laravel je open-source PHP framework určený k vývoji webových aplikací. Je navržen tak, aby zrychlil proces vývoje zejména kvůli stručné syntaxi. Primárně se jedná o backendový framework, který poskytuje mnoho vlastností pro vývoj moderních webových aplikací, jako je routování, validace, cachování, ukládání souborů a mnoho dalšího. Ve světě patří mezi nejrozšířenější objektově orientované PHP frameworky. [7][8]

Charakteristika a klíčové vlastnosti

Laravel je framework se stručnou syntaxí, která usnadňuje psaní i čtení kódu. Díky rozsáhlé a dobře strukturované dokumentaci se v něm vývojáři snadno orientují a mohou se ho rychle naučit. Pokud již mají zkušenosti s PHP, mohou s Laravelem začít pracovat téměř okamžitě. Pro úplné začátečníky v PHP je Laravel zároveň skvělou volbou, protože nejen usnadňuje vstup do tohoto jazyka, ale také učí klíčové koncepty webového vývoje a objektově orientovaného programování, na kterém je framework postaven. [8]

Z pohledu výkonnosti poskytuje Laravel nástroje pro optimalizaci, jako je cachování, práce s databázemi prostřednictvím Eloquent ORM (Object-Relational Mapping), řazení do front, vysílání a testování. Tyto vlastnosti mu pomáhají zvládat velkou zátěž i složité funkce a je ideální volbou pro rozsáhlé webové aplikace, které vyžadují rychlou odezvu a vysokou stabilitu. Navíc obsahuje vestavěné nástroje pro testování, což usnadňuje ladění a optimalizaci výkonu. [8]

I když je Laravel primárně backendový framework, umožňuje vývoj i fullstack aplikací. Jeho vestavěná podpora pro Blade templating engine umožňuje efektivní správu frontendu, zatímco backendové funkce jako Eloquent ORM usnadňují práci s databázemi. Kromě toho Laravel nabízí Livewire, framework pro tvorbu dynamických a interaktivních frontendů přímo v PHP, který umožňuje dosáhnout podobné živosti a moderního uživatelského zážitku jako u JavaScriptových frameworků, například Vue nebo React. [7]

Bezpečnost a údržba aplikací jsou v Laravelu klíčovými aspekty. Poskytuje vestavěné mechanismy pro ochranu proti běžným bezpečnostním hrozbám, jako je SQL injection, cross-site scripting (XSS) a cross-site request forgery (CSRF). Blade templating engine automaticky escapuje výstup, čímž minimalizuje riziko XSS útoků, a framework generuje CSRF tokeny pro ochranu proti podvrženým požadavkům. Laravel také zajišťuje bezpečné ukládání hesel pomocí hashovacích algoritmů Bcrypt a Argon2, nabízí šifrování dat prostřednictvím OpenSSL a umožňuje ověřování e-mailů uživatelů. Dále obsahuje ochranu proti útokům hrubou silou a DDoS (Distributed Denial of Service) zajištěnou prostřednictvím omezení rychlosti požadavků (Rate Limiting). Vzhledem k jasné struktuře a dodržování návrhových vzorů je údržba kódu efektivní, přehledná a snadno rozšiřitelná. [7]

Laravel má aktivní a rostoucí komunitu vývojářů po celém světě. Díky tomu je k dispozici množství balíčků a rozšíření, které usnadňují a urychlují vývoj. Komunita také poskytuje podporu prostřednictvím fór, tutoriálů a konferencí, což usnadňuje řešení problémů a sdílení znalostí. [8]

Použití

Tento framework se často využívá pro vývoj široké škály webových aplikací, od menších projektů až po rozsáhlé enterprise systémy. Jeho flexibilita a bohatá sada funkcí umožňují nasazení v různých typech aplikací, a právě to z něj činí populární volbu mezi vývojáři. Mezi možné příklady použití patří [9]:

- E-commerce platformy: Laravel je ideální volbou pro e-commerce systémy zejména pro svou flexibilitu, škálovatelnost a rozsáhlý ekosystém nástrojů. Umožňuje snadnou integraci funkcí, jako je správa produktů, platební systémy nebo logistika. Velké množství balíčků a aktivní komunita usnadňují vývoj a přizpůsobení e-shopů specifickým potřebám. Navíc poskytuje vestavěné mechanismy pro zabezpečení, výkon a správu databází, a to je nezbytné pro stabilní provoz online obchodů.
- Systémy pro správu obsahu (CMS): Laravel je vhodný pro tvorbu systémů pro správu obsahu (CMS) především svou flexibilitou a rozsáhlou sadou funkcí. Jeho modularita a rozšiřitelnost z něj činí spolehlivý základ pro robustní CMS systémy.

- **RESTful API (Application Programming Interface):** Laravel je často využíván pro tvorbu RESTful API pro svoji integraci s mobilními aplikacemi a frontendovými frameworky. Díky své syntaxi a bohaté sadě nástrojů, jako je Eloquent ORM, usnadňuje správu databází a zjednodušuje vývoj API. Vestavěné funkce pro autentizaci a autorizaci poskytují Laravelu bezpečné prostředí pro komunikaci mezi klientem a serverem. Navíc podpora pro verzování API a middleware umožňuje řízení přístupu a údržbu různých verzí API. [10]

Výhody a nevýhody

Laravel je oblíbený PHP framework, který nabízí řadu výhod pro vývoj webových aplikací, ale má i některé nevýhody. Přehled výhod a nevýhod je uveden v Tabulce 1 na konci této kapitoly. [11][12]

Výhody

- **Jednoduchost kódu:** Laravel poskytuje rozsáhlou knihovnu předprogramovaných funkcionalit, což minimalizuje množství potřebného kódu a usnadňuje vývoj robustních webových aplikací.
- **Škálovatelnost:** Framework je vysoce škálovatelný. To umožňuje jeho použití pro projekty různých velikostí, od malých, až po velké webové aplikace.
- **Bezpečnost:** Laravel nabízí vysokou úroveň zabezpečení, což chrání aplikace před běžnými bezpečnostními hrozbami.
- **Bohatá sada nástrojů:** Framework disponuje širokou škálou nástrojů pro vývojáře, která zjednodušuje a urychluje proces vývoje.
- **Snadné učení:** Vzhledem k jednoduché a přehledné syntaxi a rozsáhlé dokumentaci je Laravel relativně snadno osvojitelný i pro méně zkušené vývojáře.

Nevýhody

- **Náklady na vývojáře:** Najmutí zkušených Laravel vývojářů může být nákladné. Kvůli tomu se můžou zvýšit celkové náklady na projekt.
- **Omezená podpora:** Přestože má Laravel aktivní komunitu, oficiální podpora může být omezená. To občas může způsobit problémy při řešení specifických problémů.
- **Časté aktualizace:** Framework vyžaduje časté aktualizace, které mohou být náročné na údržbu a může způsobit problémy s kompatibilitou.

- Rychlost: V některých případech může být Laravel pomalejší ve srovnání s jinými frameworky, což může ovlivnit výkon aplikace.

Tabulka 1 Souhrn výhod a nevýhod Laravelu

Výhody	Nevýhody
Jednoduchost kódu	Náklady na vývojáře
Škálovatelnost	Omezená podpora
Bezpečnost	Časté aktualizace
Bohatá sada nástrojů	Rychlost
Snadné učení	

2.2.2 Angular

Angular je open-source framework vyvinutý společností Google pro tvorbu moderních webových aplikací primárně potom pro vývoj jednostránkových webových aplikací (SPA). Svou přehlednou syntaxí si dává za cíl zefektivnit a urychlit vývoj nových funkcionalit bez zbytečné složitosti. Staví na komponentové architektuře se službami. Nejedná se o JavaScriptový jazyk, nýbrž o TypeScript. [13]

Charakteristika a klíčové vlastnosti

Tento framework je přístupný i pro začínající vývojáře díky přehledné dokumentaci a široké komunitě. Oficiální dokumentace nabízí přehledné návody a příklady, které umožňují pochopení základních konceptů frameworku. Doba potřebná k osvojení Angularu závisí na předchozích zkušenostech s webovým vývojem, ale s pravidelným studiem a praxí lze dosáhnout pokročilé úrovně během několika měsíců. [14]

Angular poskytuje řadu nástrojů pro optimalizaci výkonu aplikací, jako je detekce změn a minimalizace zátěže při vykreslování komponent. Tento proces je zautomatizován, avšak pokud je spouštěn příliš často, může docházet ke zpomalení aplikace. K dispozici jsou také nástroje jako Angular DevTools a podpora pro Chrome DevTool, které pomáhají ladit výkon aplikací. [15]

Nejčastěji je Angular využíván jako frontendová část ve fullstack vývoji, kde spolupracuje s různými backendovými technologiemi, jako jsou Node.js, .NET nebo Java. Umožňuje vytvářet single-page aplikace (SPA) s přehledným uživatelským rozhraním a podporuje modularitu. Tím usnadňuje správu rozsáhlých projektů. Svými funkcemi, jako

je datová vazba, validace formulářů, routování a dependency injection, poskytuje vývojářům efektivní nástroje pro tvorbu moderních webových aplikací. Je tak důležitou součástí fullstack ekosystému, kde zajišťuje dynamický a interaktivní frontend, zatímco o backend se starají jiné technologie. [16]

Framework obsahuje vestavěné mechanismy pro ochranu proti běžným bezpečnostním hrozbám, jako jsou útoky typu XSS a umožňuje implementaci postupů, jako je Content Security Policy (CSP). Díky pravidelným aktualizacím a aktivní komunitě je Angular neustále vylepšován a udržován. [17]

Angular je velmi oblíbený mezi vývojáři a má rozsáhlou komunitu, která poskytuje množství knihoven, nástrojů a rozšíření. Podle průzkumu Stack Overflow Developer Survey patří mezi nejpoužívanější front-end frameworky, což potvrzuje jeho stabilitu a širokou podporu. Aktivní komunita nejenže přispívá k jeho neustálému rozvoji, ale také usnadňuje sdílení znalostí a řešení problémů. To vše z něj činí ideální volbu pro rozsáhlejší projekty. [18]

Použití

Angular nachází uplatnění v široké škále webových i mobilních aplikací, od menších projektů po rozsáhlé podnikové systémy. Jeho flexibilita a rozsáhlý ekosystém nástrojů z něj činí preferovanou volbu mezi vývojáři. Mezi nejčastější scénáře využití patří [19]:

- **Jednostránkové aplikace (SPA):** Tento framework je ideální pro vývoj aplikací s plynulým uživatelským prostředím, kde se obsah načítá a aktualizuje dynamicky bez potřeby obnovení celé stránky. To umožňuje propracovaný routing a správa detekce změn aplikace.
- **Komplexní enterprise systémy:** Organizace s vysokými nároky na škálovatelnost a bezpečnost často volí Angular pro vývoj rozsáhlých informačních systémů. Jeho modularita, znovupoužitelnost komponent a jasně definovaná architektura umožňují efektivní správu kódu v dlouhodobých projektech.
- **Mobilní aplikace:** S pomocí technologie NativeScript nebo Ionic lze Angular využít i pro vývoj mobilních aplikací. NativeScript umožňuje tvorbu plně nativních aplikací pro Android a iOS, zatímco Ionic poskytuje hybridní řešení, které sdílí značnou část kódu napříč platformami.

- Progresivní webové aplikace (PWA): Angular podporuje vývoj moderních PWA, které kombinují výhody nativních aplikací s flexibilitou webu. Poskytují rychlé načítání, offline režim a možnost instalace přímo z prohlížeče. Pro usnadnění vývoje Angular nabízí vestavěný nástroj *@angular/pwa*.
- Server-side rendering (SSR): Pro zlepšení výkonnosti a optimalizaci vyhledávače lze Angular kombinovat s Angular Universal pro předběžné vykreslení obsahu na serveru. Tím se urychluje načítání stránky a zlepšuje SEO (Search Engine Optimization).

Výhody a nevýhody

Angular patří mezi často využívané frameworky pro vývoj webových aplikací a přináší jak silné stránky, tak určité výzvy. Přehled výhod a nevýhod je uveden v Tabulce 2 na konci této kapitoly. [18]

Výhody

- Modularita a opakovatelné použití kódu: Díky komponentově orientované architektuře umožňuje Angular snadné opakované použití kódu a urychluje vývoj.
- Podpora TypeScriptu: Angular je postaven na TypeScriptu. To přináší výhody jako statické typování a lepší čitelnost kódu.
- Aktivní komunita a podpora: Široká uživatelská základna a podpora od společnosti Google zajišťují pravidelné aktualizace a rozsáhlou dokumentaci.

Nevýhody

- Strmá křivka učení: Angular je komplexní framework, jehož osvojení může být náročné, zejména pro začátečníky.
- Velikost aplikace: Aplikace postavené na Angularu mohou být objemnější, což může ovlivnit výkon, zejména u menších projektů.
- Složitost migrace: Přejít mezi různými verzemi Angularu může být komplikovaný a vyžadovat značné úsilí.

Tabulka 2 Souhrn výhod a nevýhod Angularu

Výhody	Nevýhody
Modularita a opakovatelné použití kódu	Strmá křivka učení
Podpora TypeScriptu	Velikost aplikace
Aktivní komunita a podpora	Složitost migrace

2.2.3 React

React je open-source JavaScriptová knihovna vyvinutá společností Meta pro tvorbu interaktivních uživatelských rozhraní, zejména jednostránkových aplikací (SPA). Využívá virtuální DOM (Document Object Model) pro aktualizaci UI (User Interface) a komponentovou architekturu, která podporuje znovupoužitelnost kódu. Pro tvorbu komponent používá JSX syntaxi podobnou HTML (HyperText Markup Language), integrovaná do JavaScriptu. React je flexibilní, podporuje různé backendové technologie a často se kombinuje s Reduxem nebo React Routerem pro routování. [20][21]

Charakteristika a klíčové vlastnosti

React je obecně považován za přístupnou knihovnu pro vývojáře, zejména pro ty, kteří již mají základní znalosti HTML, CSS (Cascading Style Sheets) a JavaScriptu. Jeho komponentová architektura a použití JSX usnadňují vytváření interaktivních uživatelských rozhraní. Přestože může být pro začátečníky zpočátku náročné pochopit některé pokročilé koncepty, jako je správa stavu, díky rozsáhlé dokumentaci a aktivní komunitě je proces učení efektivnější. [22]

React nabízí různé techniky pro optimalizaci výkonu aplikací. Memorování komponent pomocí *React.memo* zabraňuje zbytečnému znovuvykreslování. Správa stavu minimalizuje počet komponent, které se musí aktualizovat při změně dat. Asynchronní rendering umožňuje rozdělení vykreslování na menší části, a tím zlepšuje odezvu aplikace. Implementací těchto metod lze dosáhnout plynulejšího vykreslování. [23]

Jako front-endová knihovna se React zaměřuje primárně na tvorbu uživatelských rozhraní. Sama o sobě neobsahuje nástroje pro back-end vývoj, jako je správa databází nebo serverová logika. Pro vytvoření full-stack aplikace je tedy nutné React kombinovat s dalšími technologiemi, které pokrývají serverovou stranu aplikace. [20]

React sám o sobě nenabízí mnoho vestavěných bezpečnostních mechanismů. Automaticky chrání proti Cross-Site Scriptingu (XSS) tím, že escapuje hodnoty při vykreslování. Přesto je důležité validovat a čistit uživatelský vstup, aby se předešlo XSS útokům. Doporučuje se také vyhnout přímému vkládání HTML do DOMu, omezit použití nebezpečných URL (Uniform Resource Locator) a používat bezpečné metody pro zpracování odkazů. [24]

Díky široké komunitní podpoře a rozsáhlé knihovně vzdělávacích materiálů patří React mezi nejpoužívanější frontendové technologie. Od svého open-source uvedení Facebookem v roce 2013 si získal stabilní postavení na trhu a dnes pohání přes 52 milionů webových stránek. Jeho komponentová architektura a možnost opětovného použití kódu urychlují vývoj a zjednodušují správu aplikací, což přispívá k jeho trvalé popularitě. [25]

Použití

Komponentová architektura a kompatibilita s různými backendy činí React skvělou volbou pro vývoj komplexních webových aplikací s bohatým uživatelským rozhraním. Snadná integrace do stávajících systémů a možnost znovupoužití komponent zajišťují konzistentní vzhled a vývoj. Mezi nejčastější scénáře využití patří [26]:

- Mobilní aplikace: S využitím React Native lze vytvářet nativní mobilní aplikace pro iOS i Android. React Native umožňuje vývojářům použít stejné principy jako při tvorbě webových aplikací, čímž se vývoj urychluje a snižují se náklady.
- AR/VR aplikace: React lze využít i pro vývoj rozšířené a virtuální reality. Pomocí knihoven jako React VR (Virtual Reality) je možné vytvářet 3D aplikace kompatibilní s VR zařízeními, jako jsou Oculus Rift nebo HTC Vive.
- Sociální sítě: Rychlé renderování komponent na straně serveru a efektivní aktualizace uživatelského rozhraní bez nutnosti obnovování celé stránky činí React vhodným pro vývoj sociálních sítí. Umožňuje například okamžité aktualizace interakcí, jako je změna počtu "lajků" na příspěvcích.
- E-commerce aplikace: React se často využívá pro vývoj e-commerce platformy kvůli své modulární architektuře, která umožňuje snadnou správu a aktualizaci jednotlivých komponent bez ovlivnění zbytku aplikace.

Výhody a nevýhody

React i přes některé nevýhody zůstává jedním z nejpoužívanějších nástrojů pro vývoj moderních webových aplikací pro svou flexibilitu, škálovatelnost a silnou komunitní podporu. Přehled výhod a nevýhod je uveden v Tabulce 3 na konci této kapitoly. [27]

Výhody

- Komponentová architektura: React umožňuje vývoj pomocí znovupoužitelných, nezávisle spravovaných komponent. Usnadňuje škálování aplikací a jejich údržbu.
- Možnost přizpůsobení: Díky své flexibilitě React usnadňuje tvorbu aplikací na míru, včetně integrace interaktivních prvků, jako jsou grafy, widgety nebo živé datové zdroje.
- Budoucí perspektiva: React je připraven na integraci s novými technologiemi, jako je strojové učení (Machine Learning) a umožňuje například implementaci chatbotů přímo v aplikacích.
- Redux pro správu stavu: U Single Page Applications (SPA) React spolu s Reduxem zajišťuje konzistentní řízení stavu. Přispívá ke stabilitě aplikací a lepšímu uživatelskému požitku.

Nevýhody

- Složitost pro začátečníky: React obsahuje pokročilé koncepty, jako jsou JSX, props, lifecycle metody a hooks, které mohou být pro začátečníky náročné na pochopení.
- Integrace s jinými technologiemi: Pro plnohodnotné využití je často nutné integrovat další nástroje, jako Redux, React Router nebo middleware, což může zkomplikovat vývoj.
- Vysoká závislost na JavaScriptu: Vývojáři z jiných jazykových prostředí mohou mít problémy s přechodem na React kvůli jeho silné vazbě na JavaScript.
- Velikost aplikací: Kvůli velkému množství závislostí a využívání virtuálního DOMu může React vytvářet objemné aplikace a výsledkem jsou delší časy načítání.
- Snížený výkon na starších zařízeních a slabých sítích: React používá client-side rendering. Kvůli tomu může docházet k delšímu vykreslování na méně výkonných zařízeních nebo při pomalejším připojení k internetu.

Tabulka 3 Souhrn výhod a nevýhod Reactu

Výhody	Nevýhody
Komponentová architektura	Složitost pro začátečníky
Možnosti přizpůsobení	Integrace s jinými technologiemi
Budoucí perspektiva (ML, AI)	Vysoká závislost na JavaScriptu
Redux pro správu stavu	Velikost aplikací
	Snížený výkon na starších zařízeních a slabých sítích

2.2.4 Vue.js

Vue.js je JavaScriptový framework pro tvorbu uživatelských rozhraní. Staví na standardech HTML, CSS a JavaScriptu a poskytuje deklarativní, komponentově orientovaný programovací model, který umožňuje vyvíjet uživatelská rozhraní libovolné složitosti. Mezi jeho hlavní vlastnosti patří deklarativní renderování, kdy Vue.js rozšiřuje standardní HTML o syntaxi šablon umožňující popsat výstup HTML na základě stavu JavaScriptu, a reaktivita, která automaticky sleduje změny stavu v JavaScriptu a aktualizuje DOM při těchto změnách. [28]

Charakteristika a klíčové vlastnosti

Vue.js je snadno naučitelný framework, protože ke startu stačí základní znalost HTML, CSS a JavaScriptu, bez nutnosti hlubšího porozumění dalším technologiím, jako jsou JSX v Reactu nebo TypeScript v Angularu. Je podporován v populárních editorech kódu. Nováčky mají k dispozici aktivní komunitu na Discordu a fórech. Kromě toho existuje mnoho průvodců a kurzů přímo na oficiální stránce Vue.js, což výrazně usnadňuje proces učení. [29]

Díky využití virtuálního DOM se Vue.js zaměřuje na aktualizaci pouze nezbytných částí uživatelského rozhraní, čímž výrazně zlepšuje výkon. Tento přístup umožňuje rychlejší vykreslování stránek a plynulejší interakci s aplikací. Testy ukazují, že Vue.js má o 22 % lepší alokaci paměti, je o 19 % rychlejší při spuštění a o 36 % rychlejší při operacích s DOM ve srovnání s Reactem. To z něj činí vhodné řešení pro vývoj dynamických aplikací s vysokými nároky na rychlost. [29]

Vue.js není fullstack framework, ale frontendový framework zaměřený na tvorbu uživatelských rozhraní. Pro vývoj fullstack aplikací je nutné jej kombinovat s backendovými technologiemi, jako jsou například Node.js, Laravel, Django nebo Ruby on Rails.

Vue.js však nabízí podporu pro server-side rendering (SSR) skrze Nuxt.js. Ten usnadňuje integraci s backendem a zlepšuje výkon aplikací. Díky tomu lze Vue.js snadno propojit s různými API a backendovými službami, na základě čeho je vhodným nástrojem pro fullstack vývoj ve spojení s dalšími technologiemi. [28]

Bezpečnost ve Vue.js závisí na správném zacházení s uživatelským vstupem. Framework sice automaticky chrání proti XSS útokům, ale je nutné dávat pozor při použití v-html, dynamických atributů a externího obsahu. Není doporučeno používat nedůvěryhodné šablony a neměly by se vykreslovat `<script>` tagy s uživatelským vstupem, protože to může vést k bezpečnostním hrozbám. Pro maximální ochranu je doporučeno vždy čistit data a dodržovat osvědčené bezpečnostní postupy. [28]

Vue.js je populární zejména kvůli aktivní komunitě, která ho neustále podporuje a rozvíjí. Členové komunity jsou aktivní na Discordu a fórech, kde si vzájemně pomáhají a sdílí své znalosti. Důkazem rozsáhlé uživatelské základny je i více než 97 tisíc tagů souvisejících s Vue.js na Stack Overflow. To ukazuje, že framework má silnou podporu a široké využití mezi vývojáři. [29]

Použití

Vue.js je flexibilní a modulární framework, který lze použít pro širokou škálu projektů. Jednoduchá syntaxe se hodí jak pro malé webové aplikace, tak pro rozsáhlé podnikové systémy. Vue.js nabízí vysoký výkon, škálovatelnost a efektivní správu stavu aplikace. Tyto vlastnosti z něj činí vhodný nástroj pro moderní vývoj webových a mobilních aplikací. Mezi nejčastější scénáře využití patří [30]:

- Jednostránkové aplikace (SPA): Vue.js je ideální pro jednostránkové aplikace, které vyžadují interaktivitu a správu dynamického obsahu. Komponentové architektura a reaktivní datový model umožňují Vue.js plynulé aktualizace obsahu bez nutnosti obnovy celé stránky.
- Progresivní webové aplikace (PWA): Progresivní webové aplikace (PWA) se stávají stále populárnější volbou pro firmy, které chtějí nabídnout uživatelský zážitek podobný nativním mobilním aplikacím. Nástroje jako Vue Native umožňují vývoj těchto aplikací.
- Mobilní aplikace: Využití Vue.js nachází i ve vývoji mobilních aplikací například v kombinaci s NativeScriptem a Vue Native. Tyto nástroje umožňují vývoj

hybridních aplikací s využitím Vue.js syntaxe, což vývojářům usnadňuje práci při tvorbě aplikací pro iOS i Android.

- SSR a e-commerce: Vue.js je výbornou volbou pro e-commerce platformy díky podpoře server-side renderingu (SSR), zejména ve spojení s frameworkem Nuxt.js. Tento přístup umožňuje rychlejší načítání stránek a lepší SEO optimalizaci, přičemž právě to hraje klíčovou roli u online obchodů. Serverově renderovaný obsah také přispívá k lepšímu uživatelskému zážitku, zejména na mobilních zařízeních s pomalejším připojením.

Výhody a nevýhody

Mezi hlavní výhody Vue.js patří jednoduchá syntaxe a rozsáhlá komunita. Na druhou stranu může Vue.js trpět omezenou podporou ze strany velkých korporací, menší dostupností pokročilých balíčků a potenciálními problémy s rozsáhlými aplikacemi. Přehled výhod a nevýhod je uveden v Tabulce 4 na konci této kapitoly. [29]

Výhody

- Snadnost učení: Pro svou jednoduchou syntaxi, přehlednou strukturu a kvalitní dokumentaci je Vue.js přístupný i pro začátečníky.
- Flexibilita a modularita: Vue.js lze snadno integrovat do existujících projektů nebo použít pro rozsáhlé aplikace díky komponentové architektuře.
- Reaktivita a výkonnost: Vue.js využívá virtuální DOM, reaktivní data binding a umožňuje rychlé aktualizace uživatelského rozhraní.
- Silná komunita a dokumentace: Vývojáři mají k dispozici rozsáhlou dokumentaci, tutoriály a aktivní komunitu na fórech a Discordu.

Nevýhody

- Menší ekosystém: Ve srovnání s Reactem a Angularem může Vue.js postrádat některé pokročilé knihovny a nástroje.
- Slabší podpora velkých společností: Na rozdíl od Reactu a Angularu nemá Vue.js silného firemního podporovatele. To může ovlivnit dlouhodobou stabilitu.
- Problémy s rozsáhlými projekty: U velkých aplikací může být správa stavu složitější, přestože existují nástroje jako Vuex nebo Pinia.

- Nižší dostupnost zkušených vývojářů: Kvůli menšímu rozšíření Vue.js v porovnání s Reactem může být složitější najít zkušené Vue.js vývojáře pro větší projekty.

Tabulka 4 Souhrn výhod a nevýhod Vue.js

Výhody	Nevýhody
Snadnost učení	Menší ekosystém
Flexibilita a modularita	Slabší podpora velkých společností
Reaktivita a výkonnost	Problémy s rozsáhlými projekty
Silná komunita a dokumentace	Nižší dostupnost zkušených vývojářů

2.3 Zvolení technologie pro vývoj

Na základě provedené analýzy byl pro vývoj aplikace zvolen framework Laravel. Hlavním důvodem je jeho podpora fullstack vývoje, což znamená, že zvládne jak backend, tak frontend v jednom ekosystému. Díky Blade templating engine a Livewire lze vytvářet dynamické aplikace bez nutnosti kombinace více technologií.

Další výhodou je snadné osvojení a skvělá dokumentace. Laravel staví na PHP, takže pokud už s tímto jazykem člověk pracoval, přechod na tento framework je poměrně hladký. Navíc má stručnou syntaxi, která pomáhá udržovat kód čistý a přehledný.

Laravel je také dobře škálovatelný a zabezpečený. Obsahuje optimalizované nástroje pro práci s databázemi, podporu cachování a vestavěnou ochranu proti běžným hrozbám jako SQL injection, XSS nebo CSRF útoky.

Velkým plusem je silná komunita a široká podpora. Laravel má rozsáhlý ekosystém balíčků a aktivní vývojářskou komunitu, takže je snadné najít řešení na nejrůznější problémy.

S ohledem na tyto faktory byl Laravel vybrán jako nejvhodnější framework pro tento projekt. Souhrnné srovnání jednotlivých frameworků podle klíčových kritérií uvádí Tabulka 5.

Tabulka 5 Porovnání frameworků podle klíčových kritérií

Kritérium	Laravel	Angular	React	Vue.js
Fullstack podpora	Ano	Ne	Ne	Ne
Snadnost osvojení	Snadné	Strmá křivka učení	Střední náročnost	Snadné
Škálovatelnost	Výborná	Výborná	Výborná	Dobrá
Bezpečnost	Vysoká	Vysoká	Omezená	Omezená
Výkon	Optimalizovaný	Střední	Rychlý	Rychlý
Komunita a podpora	Velká a aktivní	Silná	Velká	Menší než React a Angular
Použití v praxi	Webové aplikace, API, CMS, e-commerce, podnikové systémy	SPA, enterprise aplikace, PWA, mobilní aplikace	SPA, sociální sítě, e-commerce, mobilní aplikace	SPA, PWA, menší aplikace, mobilní aplikace

3 POUŽITÉ TECHNOLOGIE

Tato kapitola se věnuje použitým technologiím v aplikaci Planora. Popisuje použitý PHP framework Laravel, jeho historii, vývojové prostředí, systém routování, práci s controllery, middleware a šablonovacím systémem Blade. Dále se zaměřuje na databázový systém MySQL, jeho vlastnosti, postup instalace a základní databázové operace, které byly využity při návrhu a implementaci aplikace.

3.1 Laravel

V předchozí kapitole byla provedena analýza populárních frameworků, včetně Laravelu, kde byly popsány jeho základní vlastnosti, výhody a nevýhody. Tato kapitola se zaměřuje na konkrétní technologické aspekty Laravelu, které jsou důležité pro vývoj moderních webových aplikací. Kapitola obsahuje stručný náhled do historie Laravelu a jeho vývoje, návod pro nastavení vývojového prostředí a vytvoření nového projektu. Následně je popsán systém routování, práce s controllery a middleware. Součástí kapitoly je také část věnovaná Blade templating systému pro frontend, který usnadňuje práci se šablonami v Laravelu.

3.1.1 Historie Laravelu

Pro pochopením významu vzniku Laravelu je důležité znát jeho historický kontext a vývoj PHP frameworků. Vývoj Laravelu byl z velké části ovlivněn předchozími technologiemi a reagoval na jejich nedostatky. [31]

Počátky moderních webových frameworků

V roce 2004 se poprvé objevil framework Ruby on Rails, který přinesl koncept MVC (Model-View-Controller), RESTful API a další principy, které ovlivnily vývoj webových aplikací. Tento přístup inspiroval vznik mnoha frameworků i v PHP ekosystému. [31]

Vzestup PHP frameworků

V reakci na popularitu Ruby on Rails začaly vznikat PHP frameworky, které se snažily přinést podobné výhody. Prvním z nich byl v roce 2005 CakePHP, následovaný Symfony, CodeIgniterem, Zend Frameworkem a dalšími. Každý z nich měl jiný přístup. Například CodeIgniter se soustředil na rychlý vývoj a jednoduchost a Symfony nebo Zend zase na robustnost a škálovatelnost pro enterprise aplikace. [31]

CodeIgniter byl jedním z nejpopulárnějších frameworků díky své jednoduchosti a kvalitní dokumentaci. Postupem času ale začal zaostávat, protože pomalu implementoval moderní prvky PHP, jako byly *namespaces* nebo *Composer*. To vedlo v roce 2010 Taylora Otwellu k rozhodnutí vytvořit vlastní framework – Laravel. [31]

Vznik a vývoj Laravelu

Laravel byl poprvé vydán v červnu 2011 a přinesl inovace jako Eloquent ORM pro práci s databázemi, closure-based routování a podporu autentizace. Vývoj probíhal rychle a během následujících verzí byly přidány klíčové funkce [31]:

- Laravel 2 a 3 (2011–2012) – podpora kontrolerů, testování, IoC container, migrace
- Laravel 4 (2013) – kompletní přepis frameworku s využitím Composeru, zavedení mailové komponenty, databázového seedování
- Laravel 5 (2015) – vylepšená struktura projektu, plánovač úloh, Socialite pro autentizaci
- Laravel 6+ (2019) – přechod na SemVer verzování, pravidelné menší aktualizace místo velkých změn

Nejnovější verzí Laravelu je Laravel 11, vydaná v březnu 2024. Tato verze přináší zjednodušenou adresářovou strukturu a nový oficiální WebSocket server Laravel Reverb. Laravel dodržuje SemVer verzování, kdy nové hlavní verze vycházejí každý rok v prvním čtvrtletí, zatímco menší aktualizace a opravy mohou vycházet i týdně. Laravel 11 je podporován bezpečnostními aktualizacemi po dobu dvou let a opravami chyb po dobu 18 měsíců. [7]

3.1.2 Nastavení vývojového prostředí a vytvoření projektu

Aby bylo možné vyvíjet webové aplikace v Laravelu, je důležité správně nastavit vývojové prostředí. Laravel nabízí několik způsobů, jak zajistit konzistentní a optimalizované prostředí pro vývoj na různých operačních systémech.

Pro správné fungování Laravelu je nutné mít nainstalované PHP ve verzi 8.1 nebo vyšší spolu s klíčovými rozšířeními, jako jsou OpenSSL, PDO, Mbstring, Tokenizer, XML, Ctype, JSON (JavaScript Object Notation) a BCMath. Kromě těchto požadavků je klíčovým nástrojem také Composer, správce závislostí pro PHP, který funguje obdobně jako NPM pro Node.js a je nezbytný pro instalaci, aktualizaci a správu balíčků potřebných pro vývoj v Laravelu. [31]

Možnosti vývojového prostředí

Existuje několik způsobů, jak nastavit lokální vývojové prostředí pro Laravel, přičemž každý z nich nabízí odlišné výhody v závislosti na požadavcích projektu a operačním systému vývojáře. [31]

- PHP vestavěný server: Nejjednodušší způsob spuštění Laravel aplikace je použití vestavěného serveru PHP pomocí příkazu `php -S localhost:8000 -t public`. Tento způsob je vhodný pro základní testování, ale neposkytuje další nástroje pro správu databází nebo konfiguraci prostředí.
- Artisan serve: Laravel poskytuje vlastní příkaz pro spuštění serveru pomocí Artisan CLI (Command Line Interface) `php artisan serve`. Tato metoda je prakticky identická s vestavěným serverem PHP, ale jednodušší na použití.
- Laravel Sail: Sail je oficiální vývojové prostředí Laravelu postavené na Dockeru, které je dostupné na všech operačních systémech. Obsahuje předkonfigurované prostředí s webovým serverem, databází a dalšími nástroji. Aplikaci lze spustit příkazem `./vendor/bin/sail up`. Jeho výhoda spočívá v jednotném vývojovém prostředí bez nutnosti manuální instalace všech závislostí.
- Laravel Valet: Valet je nástroj určený především pro macOS. Umožňuje snadné nastavení lokálního vývojového prostředí s podporou vlastní domény pro každý projekt. Po instalaci lze spustit projekt příkazem `valet park`.
- Laravel Homestead: Homestead je předkonfigurovaný virtuální stroj postavený na Vagrantu, který simuluje produkční prostředí Laravel aplikací a je ideální pro vývojáře pracující na Windows, kteří potřebují stabilní a konzistentní vývojové prostředí. Nabízí kompletní vývojovou sadu zahrnující PHP, MySQL, Redis, PostgreSQL a další služby bez nutnosti manuální konfigurace.

Vytvoření nového projektu

Pro vytvoření nového Laravel projektu existují tři hlavní způsoby, všechny prováděné za pomoci příkazového řádku: pomocí Laravel Installeru, pomocí Composer create-project nebo s využitím Laravel Sail. [31]

- Vytvoření projektu pomocí Laravel Installeru: Nejdoporučovanější metodou je použití Laravel Installeru, který je potřeba nejprve nainstalovat pomocí Composeru příkazem `composer global require "laravel/installer"`. Po instalaci lze nový projekt

vytvořit příkazem *laravel new projectName*. Tímto se vytvoří složka *projectName*, která již obsahuje základní Laravel projekt.

- Vytvoření projektu pomocí Composer create-project: Druhou možností je využití Composer create-project, který umožňuje instalaci Laravelu přímo prostřednictvím Composeru. Projekt se vytvoří příkazem *composer create-project laravel/laravel projectName*. Tento příkaz vytvoří složku *projectName* se základním Laravel projektem.
- Vytvoření projektu pomocí Laravel Sail: Pro vývojáře, kteří plánují používat Laravel Sail je možné Laravel nainstalovat a zároveň spustit Sail pomocí příkazu *curl -s "https://laravel.build/example-app" | bash*. Po dokončení instalace je potřeba se přepnout do složky projektu a spustit Sail. Toho lze docílit příkazy *cd example-app* a *./vendor/bin/sail up*.

3.1.3 Routování a controllery

Routování je klíčovou součástí každé webové aplikace, která zajišťuje zpracování požadavků od uživatelů a jejich směrování na správné části aplikace. Laravel nabízí robustní systém routování, který umožňuje snadno definovat cesty, parametry a metody HTTP, a zároveň poskytuje integraci s controllery pro lepší organizaci kódu. [31]

Model-View-Controller (MVC) v Laravelu

Laravel je postaven na architektuře MVC (Model-View-Controller), která odděluje jednotlivé části aplikace [31]:

- Model – reprezentuje data a práci s databází.
- View – odpovídá za prezentaci dat uživateli.
- Controller – zpracovává HTTP požadavky, komunikuje s modelem a vrací odpověď prostřednictvím view.

Definování rout v Laravelu

Routy v Laravelu se definují v souborech *routes/web.php* pro webové aplikace s uživatelským rozhraním a *routes/api.php* pro REST API, které pracuje s JSON odpověďmi. Pokud chceme definovat routu, tak nejjednodušší způsob je pomocí closure funkce. Následující kód zaregistruje GET požadavek na / a vrátí textovou odpověď. [31]

```
Route::get('/', function () {  
    return 'Hello, World!';  
});
```

HTTP metody v routách

Laravel podporuje různé HTTP metody [31]:

- GET – získání zdroje.
- POST – vytvoření nového zdroje.
- PUT/PATCH – aktualizace existujícího zdroje.
- DELETE – odstranění zdroje.
- OPTIONS – dotaz na podporované HTTP metody pro danou URL.

V následující ukázce je zobrazen příklad použití některých HTTP metod.

```
Route::post('/submit', function () {  
    return 'Data submitted';  
});  
  
Route::put('/update/{id}', function ($id) {  
    return "Updating record with ID: $id";  
});  
  
Route::delete('/delete/{id}', function ($id) {  
    return "Deleting record with ID: $id";  
});
```

Použití controllerů v Laravelu

Místo definování všech rout v closure funkcích je vhodnější použít controllery, které seskupují logiku souvisejících požadavků. Nový controller lze vytvořit pomocí Artisan CLI příkazem `php artisan make:controller TaskController`. Tím se vygeneruje soubor `TaskController.php` v adresáři `app/Http/Controllers`. Takto vygenerovaný controller může vypadat následovně [31]:

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class TaskController extends Controller
{
    public function index()
    {
        return "Hello from TaskController!";
    }
}
```

V dalším kroku je potřeba napojit nově vygenerovaný controller na požadovanou routu.

```
use App\Http\Controllers\TaskController;

Route::get('/tasks', [TaskController::class, 'index']);
```

Pro CRUD (Create, Read, Update, Delete) operace lze využít resource controller, který automaticky vygeneruje standardní metody. Nový resource controller lze vytvořit pomocí příkazu `php artisan make:controller TaskController --resource`. Následně lze všechny CRUD operace napojit jedním příkazem, čímž se automaticky vytvoří routy pro zobrazení seznamu `index()`, formuláře pro vytvoření `create()`, uložení nového záznamu `store()`, zobrazení konkrétního záznamu `show()`, editaci `edit()`, aktualizaci `update()` a odstranění `destroy()`. Níže je ukázka kódu pro napojení CRUD operací. [31]

```
Route::resource('tasks', TaskController::class);
```

Middleware

Middleware v Laravelu slouží k filtrování a zpracování HTTP požadavků ještě před jejich předáním aplikaci. Nejčastěji se využívají pro autentizaci uživatelů, omezení přístupu k určitým částem aplikace nebo pro úpravu požadavků a odpovědí. [31]

Middleware lze aplikovat na skupinu rout pomocí metody *middleware()*. Následující ukázka kódu zajistí, že přístup k *dashboard* a *account* stránkám bude umožněn pouze přihlášeným uživatelům. [31]

```
Route::middleware('auth')->group(function() {  
    Route::get('dashboard', function () {  
        return view('dashboard');  
    });  
    Route::get('account', function () {  
        return view('account');  
    });  
});
```

Middleware lze přiřadit přímo v konstruktoru controlleru, což přispívá k lepší přehlednosti kódu. Stejně jako u rout se přidává pomocí metody *middleware()*, přičemž je možné určit, na které metody se má vztahovat *only()* a na které ne *except()*. Pokud je potřeba aplikovat mnoho výjimek *only()* a *except()*, může být vhodnější vytvořit samostatný controller pro dané routy. Níže je ukázka controlleru obsahující middleware. [31]

```
class DashboardController extends Controller  
{  
    public function __construct()  
    {  
        $this->middleware('auth');  
  
        $this->middleware('admin-auth')->only('editUsers');  
  
        $this->middleware('team-member')->except('editUsers');  
    }  
}
```

3.1.4 Blade templates pro frontend

Blade je šablonovací systém Laravelu, který je oproti nativnímu PHP jednodušší a přehlednější. Jeho hlavní výhodou je přehledná syntaxe a podpora dědičnosti šablon. Blade využívá dvojité složené závorky `{{ }}` pro bezpečné vypisování dat, přičemž automaticky escapuje HTML, zatímco zápis `{!! !!}` umožňuje vypsat neescapovaný obsah. [31]

Blade podporuje běžné řídicí struktury jako `@if`, `@foreach`, `@for` nebo `@while`, přičemž k dispozici jsou i speciální direktivy jako `@unless`, která má opačný význam jako `@if`, nebo `@forelse`, která kombinuje `@foreach` s možností definovat obsah pro případ, kdy je pole prázdné. [31]

Důležitou součástí Blade je dědičnost šablon. Základní rozložení stránky lze definovat v hlavní šabloně pomocí `@yield` nebo `@section`, které určují místo, kam budou vloženy konkrétní části podřízených šablon. Dědičnost se realizuje pomocí `@extends`. [31]

Další funkcí je vkládání částí šablon pomocí `@include`, `@includeIf` nebo `@each`, díky čemuž je možné opakované použití menších komponent, například tlačítek či menu. [31]

Blade podporuje stacky (`@push`, `@prepend`, `@stack`) pro snadné přidávání skriptů a stylů do konkrétní části šablony z různých míst aplikace, aniž by bylo nutné upravovat hlavní šablonu. Blade šablony v Laravelu také umožňují vytvářet vlastní direktivy pomocí `Blade::directive`. Tím zajistí možnost definovat zkratky pro opakované části kódu, například podmínky nebo formátování výstupu. [31]

3.2 MySQL

MySQL je open-source relační databázový systém. Jeho autorem je Michael Widenius, který systém vytvořil v roce 1995. O pět let později, v roce 2000 byl zpřístupněn pod licencí GNU Public License (GPL), což vedlo k jeho bezplatnému využívání a skokovému nárůstu popularity. O vývoj a vlastnictví MySQL se stará společnost MySQL AB, která uvádí, že jejich databázový systém je nainstalován na více než 4 milionech zařízení po celém světě. Mimo to, že je systém dostupný zdarma, si získal popularitu především pro svou spolehlivost, vysoký výkon a široké možnosti využití. Tyto vlastnosti z něj činí jeden z nejpoužívanějších databázových systémů na světě. [32]

Vysoká rychlost a škálovatelnost řadí MySQL mezi nejvýkonnější databázové systémy, které umí pracovat jak s malými objemy dat, tak s rozsáhlými databázemi s miliardami řádků a desítkami tisíc tabulek. V porovnání s konkurenčními systémy, jako jsou Oracle nebo

MS SQL, dosáhlo MySQL v testech magazínu eWEEK vynikajících výsledků v oblasti výkonu a škálovatelnosti. [32]

Klíčovým prvkem jeho architektury jsou úložné enginy, které propojují SQL příkazy s databázovým úložištěm a umožňují například transakční operace nebo zpětné vrácení změn. Kromě flexibility je jedním z hlavních přínosů MySQL také jeho neustálý vývoj a pravidelné aktualizace, které zajišťují vyšší stabilitu, rychlost a rozšiřování funkcí. Uživatelé mají přístup jak ke stabilní verzi, která je důkladně testována, tak k experimentální variantě obsahující nejnovější funkce, které se postupně začleňují do oficiálního vydání. [32]

3.2.1 Instalace MySQL

MySQL lze instalovat na širokou škálu operačních systémů, včetně Linuxu, FreeBSD, Unixu, Mac OS a Windows. Nejnovější stabilní verzi lze vždy stáhnout z oficiálních stránek MySQL. Před samotnou instalací je nutné rozhodnout se mezi binární nebo zdrojovou distribucí. Binární verze je jednodušší na instalaci, zatímco zdrojová distribuce umožňuje větší přizpůsobení konfigurace. [32]

Instalace na Unix a Linux

Při instalaci na Linux nebo Unix se doporučuje použít binární distribuci, ale v případě potřeby lze MySQL nainstalovat i ze zdrojových souborů. K tomu je nutné mít nainstalované nástroje jako GNU gunzip, tar a gcc. Proces zahrnuje stažení a rozbalení balíčku, vytvoření uživatelské skupiny *mysql*, konfiguraci parametrů pomocí *./configure*, kompilaci (*make* a *make install*), nastavení oprávnění a inicializaci databáze skriptem *mysql_install_db*. Po instalaci lze MySQL spustit příkazem *mysqld_safe &* a pro automatické spouštění při startu systému je nutné přidat skript do */etc/init.d/*. [32]

Instalace na Windows

Pro Windows existuje několik instalačních balíčků, přičemž nejdoporučovanější je Windows Essential Package. Instalace probíhá přes standardní instalační program s příponou *.msi* nebo pomocí manuálně rozbaleného archivu. Po instalaci se konfigurace provádí pomocí MySQL Server Instance Configuration Wizard, který umožňuje nastavit port, úložiště a další parametry. MySQL lze spustit příkazem *net start mysql*, a pro zajištění automatického spuštění při startu systému lze použít *mysqld-nt --install*. [32]

Instalace na Mac OS

Novější verze Mac OS často obsahují MySQL předinstalované, ale pokud je potřeba jej doinstalovat, stačí stáhnout PKG balíček a nainstalovat jej pomocí Finderu. Spuštění MySQL se provádí příkazem `sudo /usr/local/mysql/bin/mysqld_safe`. Pro automatické spuštění při startu systému je třeba přidat StartupItem. [32]

Post instalační kroky

Po instalaci je doporučeno provést několik bezpečnostních nastavení [32]:

- Změna hesla pro uživatele root.

```
SET PASSWORD FOR root@localhost=PASSWORD('new_password');
```

- Odstranění anonymních uživatelů.

```
DELETE FROM mysql.user WHERE User='';
```

- Vytvoření uživatelských účtů s omezenými oprávněními pomocí

```
GRANT SELECT ON *.* TO username IDENTIFIED BY 'password';
```

3.2.2 Základní operace

Po úspěšné instalaci MySQL je dalším krokem práce s databází a tabulkami. MySQL podporuje mnoho způsobů interakce, z nichž nejzákladnějším je použití klienta *mysql*, který umožňuje spravovat databáze přímo z příkazového řádku. [32]

Vytvoření databáze a tabulek

Před zahájením práce s databází v MySQL je nutné nejprve vytvořit databázi a definovat v ní tabulky, které budou uchovávat jednotlivé záznamy. V této sekci bude ukázán názorný postup vytvoření databáze a tabulek pomocí SQL příkazů. [32]

- Pro vytvoření databáze slouží SQL příkaz:

```
CREATE DATABASE example_database;
```

- Po vytvoření databáze je nutné ji vybrat, aby bylo jasné, se kterou databází se bude pracovat.

```
USE example_database;
```

- Následně je možné vytvořit tabulku uvnitř databáze, kde budou uloženy potřebné informace.

```
CREATE TABLE example (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(100),  
    year YEAR,  
);
```

Vložení dat

Po vytvoření databázových tabulek je dalším krokem jejich naplnění daty. Nejjednodušším způsobem, jak přidat nové záznamy do tabulek, je použití příkazu INSERT. Pomocí něj lze vložit jeden nebo více záznamů najednou. [32]

- K vložení dat do tabulky se používá klíčové slovo INSERT a následující SQL syntaxe:

```
INSERT INTO example (name, year)  
VALUES ('example name', 2025);
```

Dotazování dat

Jakmile jsou v databázových tabulkách uložena data, je potřeba je efektivně získávat a zobrazovat. K tomu slouží příkaz SELECT, který umožňuje načítání dat z tabulek podle různých kritérií. [32]

Nejjednodušší způsob, jak zobrazit všechna uložená data v tabulce, je použití hvězdičky (*), která reprezentuje výběr všech sloupců. Často je však užitečné zobrazit pouze vybrané sloupce nebo filtrovat výsledky podle určitých podmínek pomocí klauzule WHERE. [32]

- Pro zobrazení všech záznamů v tabulce se používá příkaz SELECT s hvězdičkou (*), která označuje výběr všech sloupců.

```
SELECT * FROM example;
```

- Pokud chceme zobrazit pouze určité záznamy na základě podmínky, použijeme klauzuli WHERE, která filtruje výsledky dotazu.

```
SELECT * FROM example WHERE year = 2025;
```

Aktualizace dat

Během práce s databází často nastává situace, kdy je potřeba upravit existující záznamy, například opravit chybný údaj nebo aktualizovat informace. K tomuto účelu slouží příkaz UPDATE, který umožňuje změnit hodnoty v jednom nebo více sloupcích na základě zadané podmínky. [32]

Pokud je potřeba změnit více hodnot najednou, lze je oddělit čárkou v rámci SET klauzule. Abychom aktualizovali pouze konkrétní řádky, používáme klauzuli WHERE, která specifikuje podmínky výběru. [32]

- Pro změnu existujících dat se používá příkaz UPDATE, za kterým následuje název tabulky. Pomocí klauzule SET určujeme, které sloupce a na jaké hodnoty se mají změnit. Pokud chceme upravit pouze konkrétní řádky, můžeme přidat podmínku pomocí WHERE.

```
UPDATE example SET year = 2002 WHERE name = 'example_name';
```

Mazání dat

Při správě databáze může nastat situace, kdy je potřeba odstranit záznamy. K tomu slouží příkaz DELETE, který umožňuje smazat konkrétní řádky v tabulce na základě zadané podmínky. [32]

Abychom předešli nechtěnému smazání všech dat, je doporučeno vždy použít klauzuli WHERE, která určuje, které řádky mají být odstraněny. Pokud je záznam provázán s jinými tabulkami pomocí cizích klíčů, je nutné zajistit správné odstranění souvisejících dat. [32]

- Pro odstranění záznamů z tabulky se používá příkaz DELETE, za kterým následuje název tabulky. Pomocí klauzule WHERE určujeme, které řádky mají být smazány, aby nedošlo k odstranění všech dat.

```
DELETE FROM example WHERE name = 'example_name';
```

II PRAKTICKÁ ČÁST

4 ANALÝZA POŽADAVKŮ

Jako první krok před návrhem a vývojem webové aplikace bylo nutné provést analýzu požadavků, vycházející z potřeb cílových uživatelů a charakteru činností, které budou v rámci aplikace vykonávat. Cílem této analýzy je definovat základní funkce systému, identifikovat uživatelské role a stanovit technické a provozní podmínky, za nichž bude aplikace spolehlivě fungovat.

4.1 Cílová skupina uživatelů

Aplikace Planora je určena především vedoucím zotavovacích akcí, jako jsou letní tábory, školy v přírodě, příměstské kempy nebo další organizované pobyty pro děti. Tito uživatelé typicky zajišťují programové a organizační plánování těchto akcí a potřebují nástroj, který jim umožní efektivní tvorbu a správu harmonogramu, včetně jeho sdílení v rámci týmu.

Aplikace vznikla původně jako řešení pro potřeby jednoho skautského střediska, avšak její univerzální struktura umožňuje využití i mimo skautské prostředí. Využít ji tedy může každý organizátor, který při plánování postupuje podobně – tj. strukturuje program do aktivit, přiděluje odpovědnosti a potřebuje srozumitelnou koordinaci v týmu.

Cílovou skupinou jsou tedy zejména vedoucí a organizátoři dětských a mládežnických akcí, kteří hledají digitální nástroj pro přehlednou a týmově sdílenou správu programu.

4.2 Funkční a nefunkční požadavky aplikace

Před návrhem a vývojem aplikace bylo nezbytné stanovit základní požadavky, které měla aplikace splňovat. Tyto požadavky jsou rozděleny na funkční, tedy konkrétní funkce, které má aplikace uživatelům nabízet, a nefunkční, které se týkají zejména technických, bezpečnostních a uživatelských aspektů aplikace.

4.2.1 Funkční požadavky

- Autentizace a registrace uživatelů – uživatelé se musí přihlásit ke svému účtu, přičemž je zajištěno ověření e-mailové adresy.
- Uživatelské profily – každý uživatel má možnost upravit svůj profil a v případě potřeby i smazat účet.
- Správa harmonogramů – uživatelé mohou vytvářet, upravovat, mazat, zobrazovat, sdílet a exportovat harmonogramy zotavovacích akcí.

- Zamykání harmonogramu – pro zamezení souběžné úpravy více uživateli je zaveden mechanismus uzamčení, který zajistí, že v daný okamžik může harmonogram upravovat pouze jeden uživatel.
- Správa aktivit v harmonogramu – do harmonogramu lze přidávat jednotlivé aktivity s jejich podrobným popisem.
- Zodpovědné osoby a oprávnění k úpravám – k jednotlivým aktivitám lze přiřadit zodpovědnou osobu, která nemusí být nutně registrovaným uživatelem. V případě potřeby však lze blok programu navázat na konkrétního uživatele v systému, přičemž následně může daný blok upravovat pouze tento uživatel.
- Export harmonogramu do PDF – harmonogram lze vyexportovat a stáhnout jako přehledný dokument pro offline použití.
- Role a oprávnění – systém rozlišuje běžné uživatele a administrátory s možností spravovat účty ostatních uživatelů.
- Kontaktní formulář – přihlášení uživatelé mohou odesílat zprávy správci aplikace přes kontaktní formulář.

4.2.2 Nefunkční požadavky

- Dostupnost přes webové rozhraní – aplikace je přístupná z jakéhokoli zařízení s internetovým připojením prostřednictvím běžného prohlížeče.
- Responzivní design – rozhraní aplikace je optimalizováno pro použití na mobilních telefonech, tabletech i počítačích.
- Bezpečnost – implementována je ochrana proti běžným typům útoků (např. CSRF, XSS), bezpečné ukládání hesel (hashování) a řízení přístupu přes middleware.
- Založeno na Laravelu – aplikace je vyvíjena pomocí PHP frameworku Laravel, který zajišťuje moderní architekturu a bezpečnostní standardy.
- Modularita kódu – kód je strukturován podle principů MVC (Model-View-Controller), což usnadňuje budoucí rozšíření a údržbu.
- Podpora více uživatelů – systém umožňuje souběžné přihlašování více uživatelů a správu dat bez kolizí.

- Možnost nasazení na běžný hosting – aplikaci je možné snadno nasadit na běžný PHP/MySQL webhosting nebo cloudové služby.

4.3 Návrh uživatelských rolí

S ohledem na charakter aplikace a způsob jejího využití byla zvolena jednoduchá struktura uživatelských rolí. Systém rozlišuje pouze dvě role – běžného uživatele (user) a administrátora (admin). Toto rozdělení přehledně odděluje běžné funkce určené koncovým uživatelům od nástrojů určených pro správu systému a zajišťuje kontrolu nad přístupem k citlivým operacím.

- Uživatel (user) – má přístup k vlastnímu účtu, může vytvářet, upravovat, zobrazovat, sdílet a exportovat harmonogramy ve svém vlastnictví, přidávat aktivity, spravovat svůj profil a komunikovat přes kontaktní formulář. Pokud mu je nasdílen harmonogram s oprávněním k úpravám a zároveň mu autor přiřadí konkrétní blok programu, může upravovat pouze tento vymezený blok.
- Administrátor (admin) – má oproti běžnému uživateli rozšířená oprávnění. Může spravovat uživatelské účty (úprava, mazání), nastavovat oprávnění a monitorovat kontaktní zprávy. Admin má také přístup ke všem harmonogramům, bez ohledu na jejich vlastnictví nebo přiřazení. Tyto harmonogramy smí admin mazat, anebo může ručně na harmonogram aplikovat zámek, čímž zamezí přístup k harmonogramu ostatním uživatelům.

5 NÁVRH APLIKACE

Návrh aplikace představuje klíčovou fází celého vývoje, během níž byly definovány struktury, komponenty a principy, které tvoří základ funkčnosti i vzhledu výsledného systému. V této kapitole jsou popsány hlavní části návrhu od datového modelu zajišťujícího logické a konzistentní uložení informací, až po uživatelské rozhraní, jehož podoba se vyvíjela od vizuálního prototypu až k finální implementaci. Důraz byl kladen na přehlednost, uživatelskou přívětivost a snadnou orientaci v systému z hlediska technického provedení i vizuální prezentace. Kapitola zároveň reflektuje vývojové změny, které nastaly v průběhu návrhu, a porovnává původní představy s reálným výsledkem.

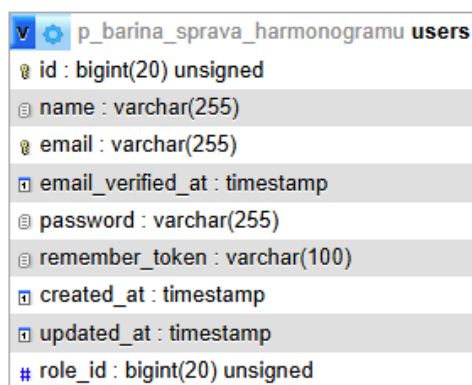
5.1 Databázový model

Databázový model tvoří základní kostru celé aplikace a zajišťuje správu dat souvisejících s harmonogramy, jejich obsahem a uživatelskými přístupy. Návrh je postaven na relační databázi MySQL a respektuje principy normalizace i přehledné rozdělení datových vazeb.

Databáze obsahuje následující tabulky:

Users

Tabulka *users* (Obrázek 1) uchovává informace o registrovaných uživateli aplikace. Každý uživatel má své jméno, e-mail, zašifrované heslo a referenci na tabulku *roles*, která definuje jeho oprávnění (*user* / *admin*).

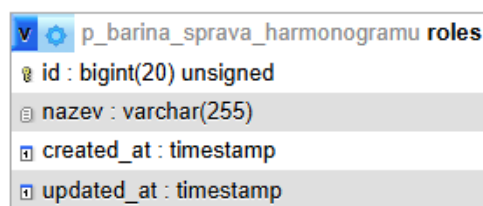


id	: bigint(20) unsigned
name	: varchar(255)
email	: varchar(255)
email_verified_at	: timestamp
password	: varchar(255)
remember_token	: varchar(100)
created_at	: timestamp
updated_at	: timestamp
role_id	: bigint(20) unsigned

Obrázek 1 – Tabulka *users*

Roles

Tabulka *roles* (Obrázek 2) obsahuje předdefinované úrovně oprávnění uživatelů. Toto rozdělení umožňuje aplikaci rozlišovat mezi běžnými uživateli a administrátory, přičemž každý uživatel má přiřazenou roli přes cizí klíč *role_id*.

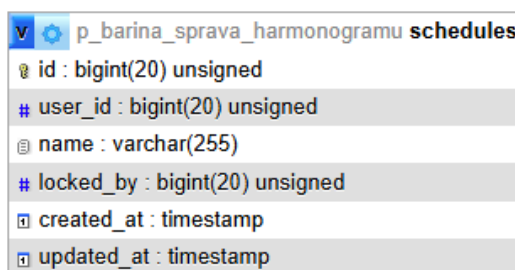


p_barina_sprava_harmonogramu roles	
id	bigint(20) unsigned
nazev	varchar(255)
created_at	timestamp
updated_at	timestamp

Obrázek 2 – Tabulka roles

Schedules

Tabulka *schedules* (Obrázek 3) reprezentuje jednotlivé harmonogramy. Každý záznam je svázán s konkrétním uživatelem (*user_id*), který je jeho vlastníkem. Volitelný sloupec *locked_by* umožňuje dočasné uzamčení harmonogramu pro úpravy pouze jedním uživatelem.

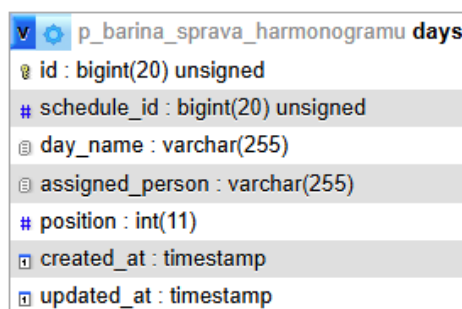


p_barina_sprava_harmonogramu schedules	
id	bigint(20) unsigned
user_id	bigint(20) unsigned
name	varchar(255)
locked_by	bigint(20) unsigned
created_at	timestamp
updated_at	timestamp

Obrázek 3 – Tabulka schedules

Days

Každý harmonogram se může skládat z více dnů. Dny jsou reprezentovány tabulkou *days* (Obrázek 4), kde každý záznam obsahuje název dne, jeho pozici v rámci harmonogramu a volitelně také přiřazenou osobu zodpovědnou za daný den.

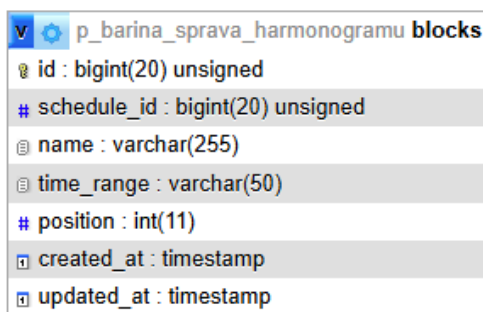


p_barina_sprava_harmonogramu days	
id	bigint(20) unsigned
schedule_id	bigint(20) unsigned
day_name	varchar(255)
assigned_person	varchar(255)
position	int(11)
created_at	timestamp
updated_at	timestamp

Obrázek 4 – Tabulka days

Blocks

Každý den se dále dělí na časové bloky (např. dopoledne, odpoledne, večer). Tabulka *blocks* (Obrázek 5) definuje jednotlivé bloky včetně jejich názvu, časového rozmezí a pořadí. Tyto bloky tvoří rámec, do kterého se vkládají konkrétní aktivity.

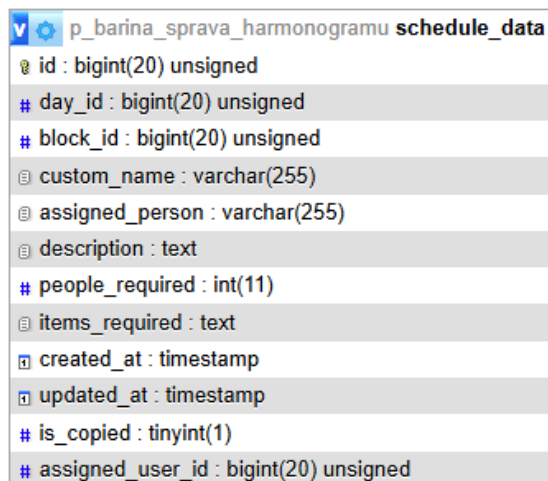


p_barina_sprava_harmonogramu blocks	
	id : bigint(20) unsigned
#	schedule_id : bigint(20) unsigned
	name : varchar(255)
	time_range : varchar(50)
#	position : int(11)
	created_at : timestamp
	updated_at : timestamp

Obrázek 5 – Tabulka blocks

Schedule Data

Jádrem celého harmonogramu je tabulka *schedule_data* (Obrázek 6), která obsahuje konkrétní programové aktivity. Každý záznam je propojen s konkrétním dnem (*day_id*) a blokem (*block_id*). Kromě názvu a popisu aktivity lze definovat odpovědnou osobu, potřebný počet lidí, materiál, podrobný popis a také, kdo má právo tuto aktivitu upravovat (*assigned_user_id*).



p_barina_sprava_harmonogramu schedule_data	
	id : bigint(20) unsigned
#	day_id : bigint(20) unsigned
#	block_id : bigint(20) unsigned
	custom_name : varchar(255)
	assigned_person : varchar(255)
	description : text
#	people_required : int(11)
	items_required : text
	created_at : timestamp
	updated_at : timestamp
#	is_copied : tinyint(1)
#	assigned_user_id : bigint(20) unsigned

Obrázek 6 – Tabulka schedule_data

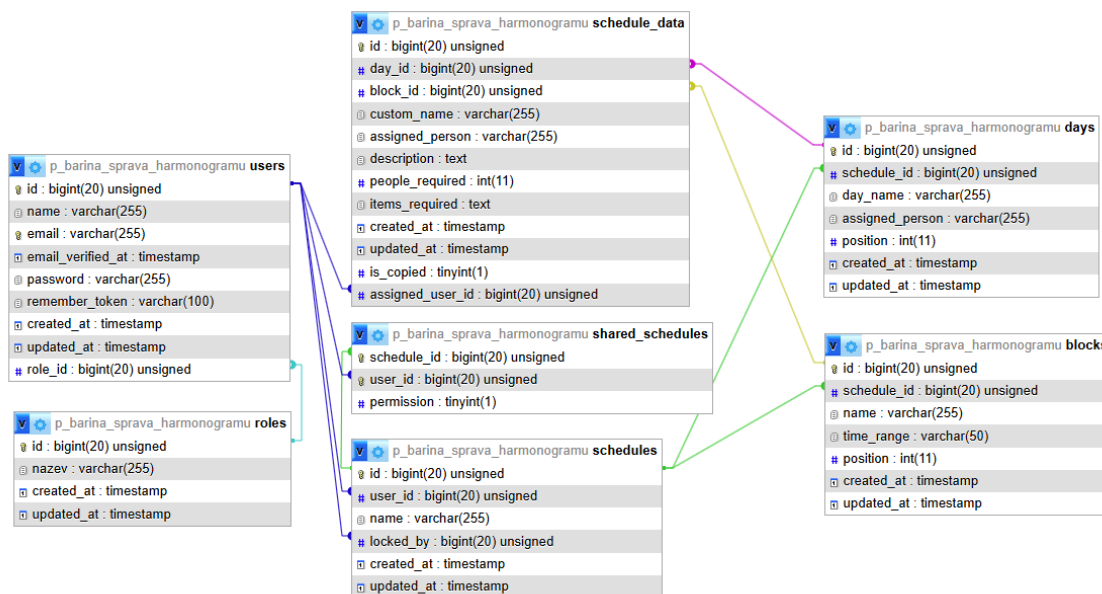
Shared Schedules

Tabulka *shared_schedules* (Obrázek 7) umožňuje sdílení harmonogramů s jinými uživateli. Každý záznam obsahuje cizí klíč na *schedules* i *users* a příznak *permission*, který určuje, zda má daný uživatel k harmonogramu pouze čtecí přístup, nebo může harmonogram i upravovat.

p_barina_sprava_harmonogramu shared_schedules	
	schedule_id : bigint(20) unsigned
	user_id : bigint(20) unsigned
	permission : tinyint(1)

Obrázek 7 – Tabulka *shared_schedules*

Na Obrázku 8 je znázorněna celková struktura databáze včetně relací mezi jednotlivými tabulkami. Diagram ilustruje hierarchii harmonogramu (harmonogram – dny a bloky – aktivity) i vazby na uživatele a jejich oprávnění. Barevně rozlišené spojnice znázorňují cizí klíče, které zajišťují konzistenci dat a provázání jednotlivých entit systému.



Obrázek 8 – Diagram celé databáze

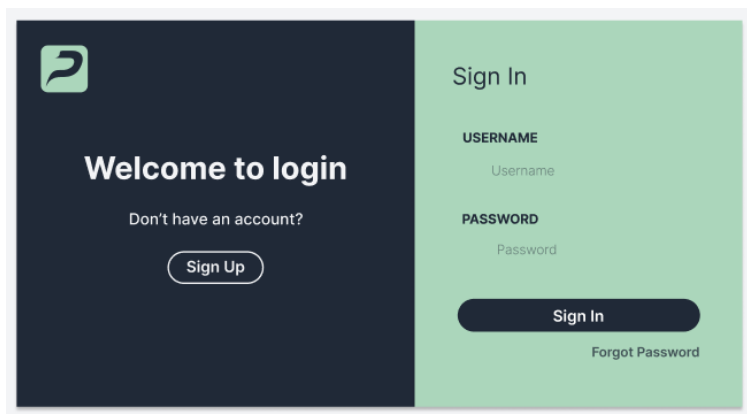
5.2 Uživatelské rozhraní

Před implementací uživatelského rozhraní byl vytvořen návrh ve formě vizuálního prototypu s pomocí nástroje Figma. Tento návrh sloužil jako výchozí model pro návrh logiky, rozmístění prvků a vizuální styl aplikace. Hlavními cíli návrhu byly přehlednost, jednoduchost a optimalizace pro různá zařízení.

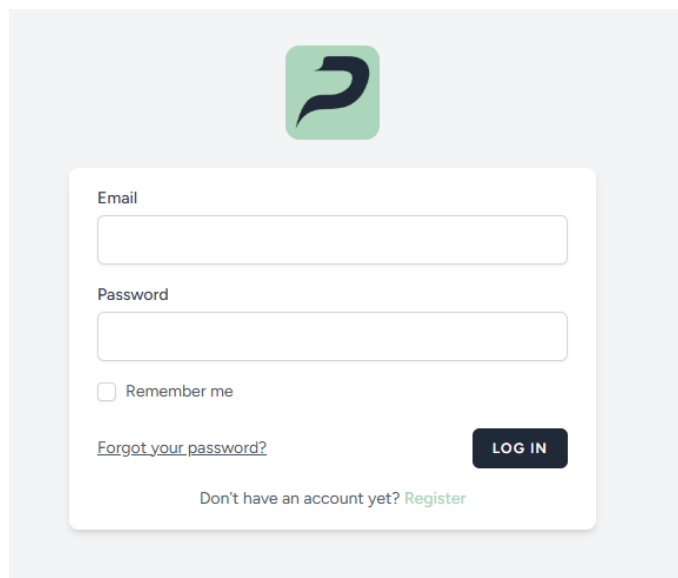
V průběhu vývoje došlo ke změnám některých prvků rozhraní, zejména z důvodu technických omezení nebo upřesnění požadavků. V následujících podkapitolách je vždy porovnán původní návrh s výsledným zpracováním v rámci aplikace.

5.2.1 Přihlašování

Původní návrh přihlašovací stránky (Obrázek 9) pracoval s dvousloupcovým rozvržením a výrazným barevným kontrastem, který mohl u uživatele navozovat dojem aktivovaného tmavého režimu, přestože aplikace žádný takový režim v současné době nepodporuje. Zároveň se ukázalo, že navržené rozložení nevyhovovalo menším obrazovkám – zejména na mobilních zařízeních působilo nepřehledně a neprakticky. Ve výsledné implementaci (Obrázek 10) byl proto zvolen jednodušší a přehlednější design s centrálním zarovnáním, který je snadno použitelný na všech typech zařízení.



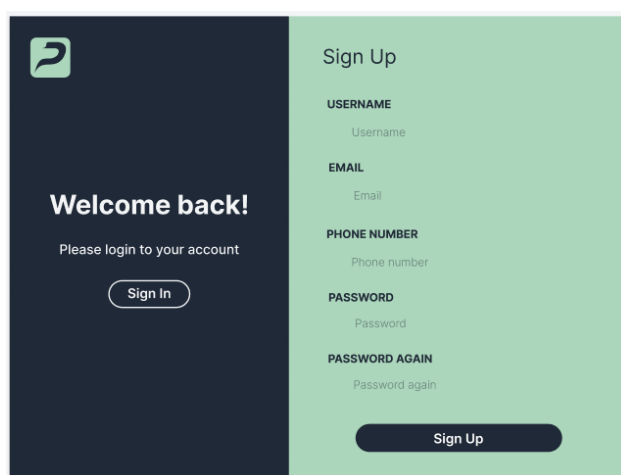
Obrázek 9 – Návrh přihlašovacího formuláře

The image shows a login form on a light gray background. At the top center is a green square logo with a white stylized 'P' or '2' shape. Below the logo is a white rounded rectangle containing the form fields. It has an 'Email' label above a text input field, a 'Password' label above another text input field, a checkbox labeled 'Remember me', a link 'Forgot your password?' in green, and a dark blue 'LOG IN' button. At the bottom of the white box is the text 'Don't have an account yet? Register' in green.

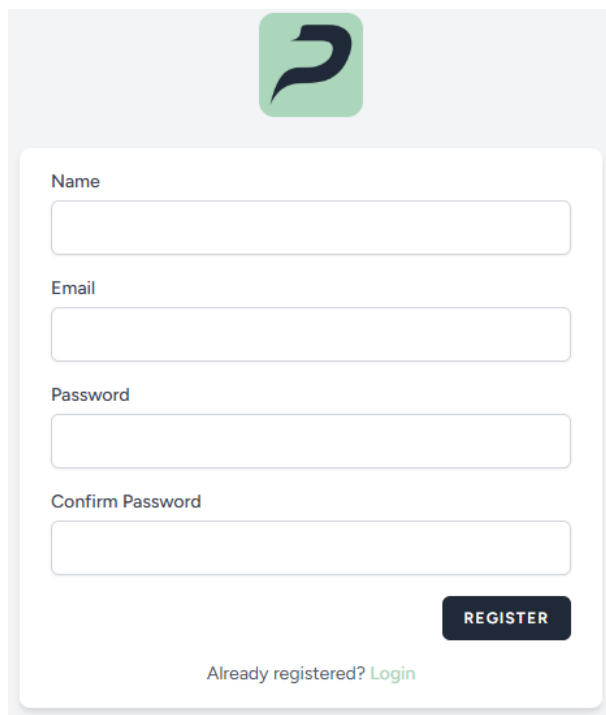
Obrázek 10 – Finální podoba přihlašovacího formuláře

5.2.2 Registrace

Původní návrh registrační stránky (Obrázek 11) opět pracoval s rozděleným layoutem a použitím výrazného barevného kontrastu, což však i v tomto případě působilo dojmem tmavého režimu, který aplikace v současné době nepodporuje. Finální verze (Obrázek 12) byla zjednodušena nejen z důvodu lepšího zobrazení na mobilních zařízeních, ale také kvůli úpravě samotného formuláře. Pole pro zadání telefonního čísla bylo odstraněno, jelikož z hlediska funkce aplikace nebylo potřebné. Výsledná podoba je přehlednější a více odpovídá minimalistickému stylu celé aplikace.

The image shows a registration form with a split layout. The left side has a dark blue background with a green logo at the top, the text 'Welcome back!', 'Please login to your account', and a 'Sign In' button. The right side has a light green background with the title 'Sign Up' and five form fields: 'USERNAME' (Username), 'EMAIL' (Email), 'PHONE NUMBER' (Phone number), 'PASSWORD' (Password), and 'PASSWORD AGAIN' (Password again). A dark blue 'Sign Up' button is at the bottom right.

Obrázek 11 – Návrh registračního formuláře



The image shows a registration form with a light gray background. At the top center is a green square logo with a white stylized 'S' or 'Z' shape. Below the logo is a white rounded rectangle containing the form fields. The fields are labeled 'Name', 'Email', 'Password', and 'Confirm Password', each with a corresponding text input box. To the right of the 'Password' and 'Confirm Password' fields is a dark blue button with the word 'REGISTER' in white capital letters. At the bottom center of the white box, there is a link that says 'Already registered? Login' in a small, green font.

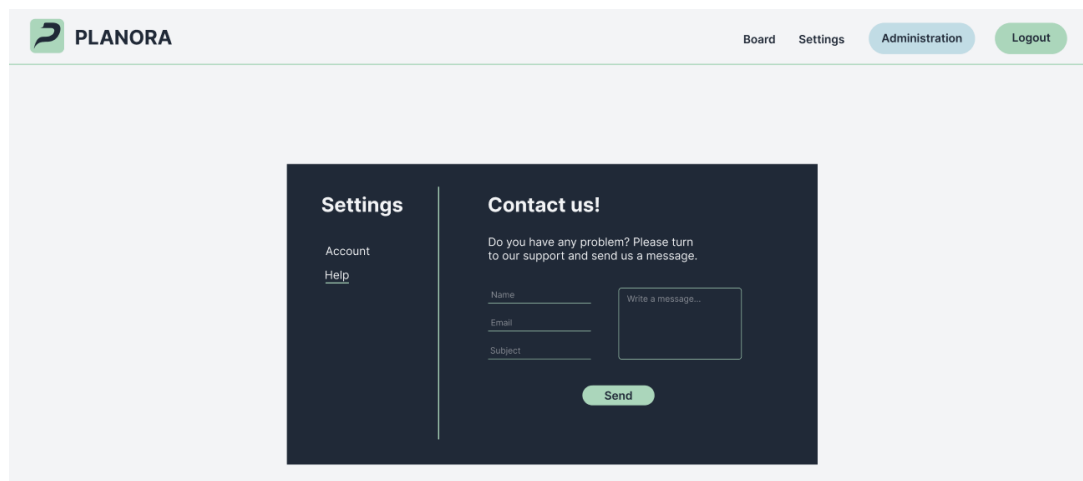
Obrázek 12 – Finální podoba registračního formuláře

5.2.3 Zapomenuté heslo a obnovení hesla

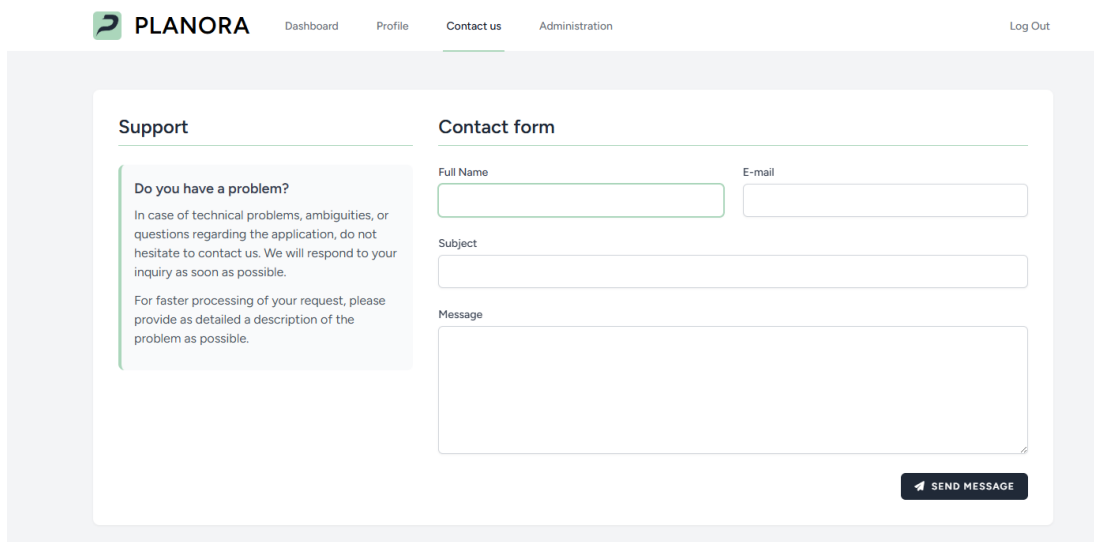
Původní návrh formulářů pro zapomenuté heslo a obnovení odpovídal prvotnímu návrhu přihlašovací stránky (Obrázek 9), avšak v konečné implementaci došlo ke sjednocení designu s ostatními formuláři v aplikaci. Hlavním důvodem úpravy bylo opět zajištění lepší přístupnosti a správného zobrazení na menších zařízeních. Design je díky tomu přehlednější a bez zbytečných barevných prvků, které by mohly působit rušivě.

5.2.4 Kontaktní formulář

V případě kontaktního formuláře došlo mezi návrhem a finální implementací k několika zásadním změnám. Původně byl formulář navržen jako součást záložky „Settings“ (Obrázek 13), kde byl umístěn vedle nastavení uživatelského účtu. Ve výsledné verzi byl však přesunut do samostatné položky (Obrázek 14) v hlavním menu, což přispělo ke zpřehlednění struktury rozhraní a lepší orientaci uživatele. Kromě toho se změnila také barevná paleta – původní návrh využíval tmavé pozadí, které však omezovalo možnosti případné budoucí implementace tmavého režimu. Finální verze proto přešla na světlý design, který nabízí lepší výchozí základ pro podporu více barevných témat včetně tmavého režimu.



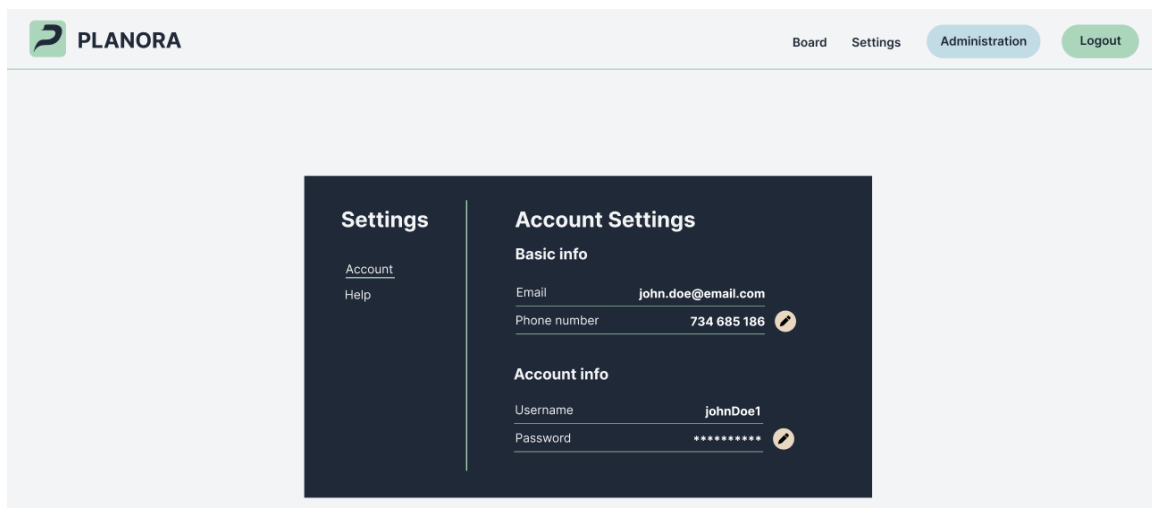
Obrázek 13 – Návrh kontaktního formuláře



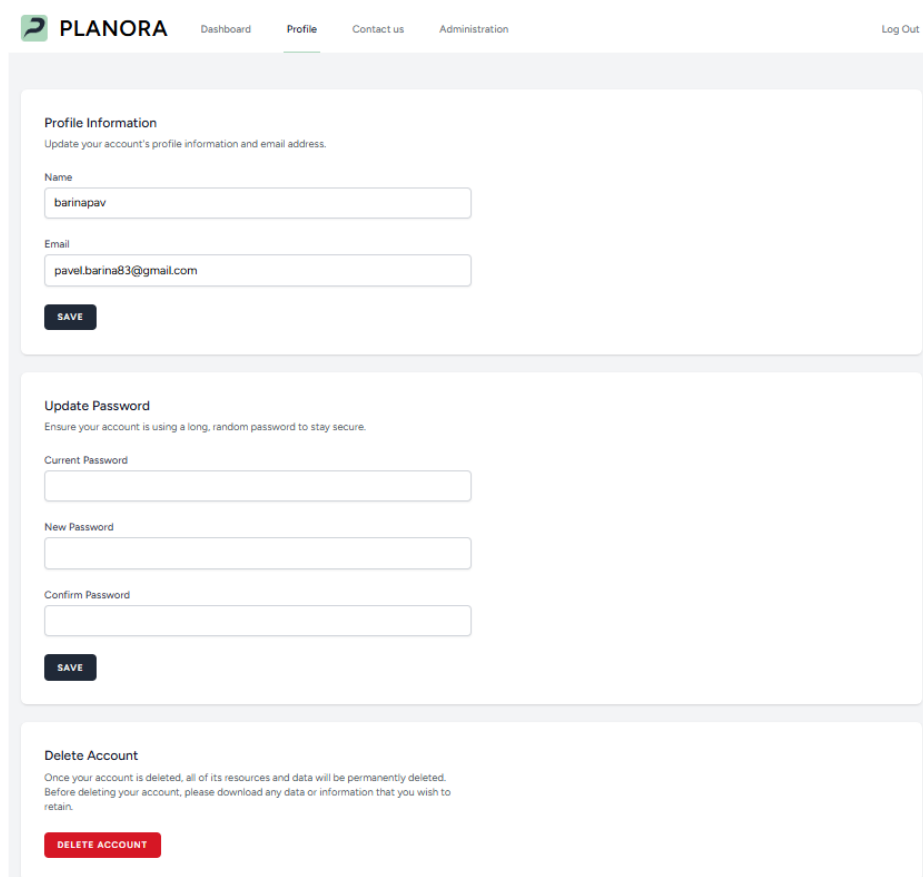
Obrázek 14 – Finální podoba kontaktního formuláře

5.2.5 Uživatelský profil

Uživatelský profil byl v návrhu řešen jako jedna ze záložek v rámci sekce „Settings“ (Obrázek 15), společně s kontaktním formulářem. Ve finální verzi (Obrázek 16) aplikace došlo k oddělení těchto funkcí do samostatných položek v hlavním menu, čímž se zjednodušila navigace. Výraznou změnou prošla také barevná paleta. Původní tmavé pozadí bylo nahrazeno světlým, které lépe zapadá do celkového vizuálního stylu aplikace a zároveň usnadňuje případné budoucí zavedení tmavého režimu. Kromě vizuálních úprav došlo i ke změně rozsahu funkcí: v návrhu bylo možné upravovat pouze heslo a telefonní číslo, zatímco e-mail a uživatelské jméno byly jen pro čtení. Ve finální implementaci je možné upravovat jak jméno, tak e-mail, měnit heslo a také zcela smazat uživatelský účet.



Obrázek 15 – Návrh uživatelského profilu

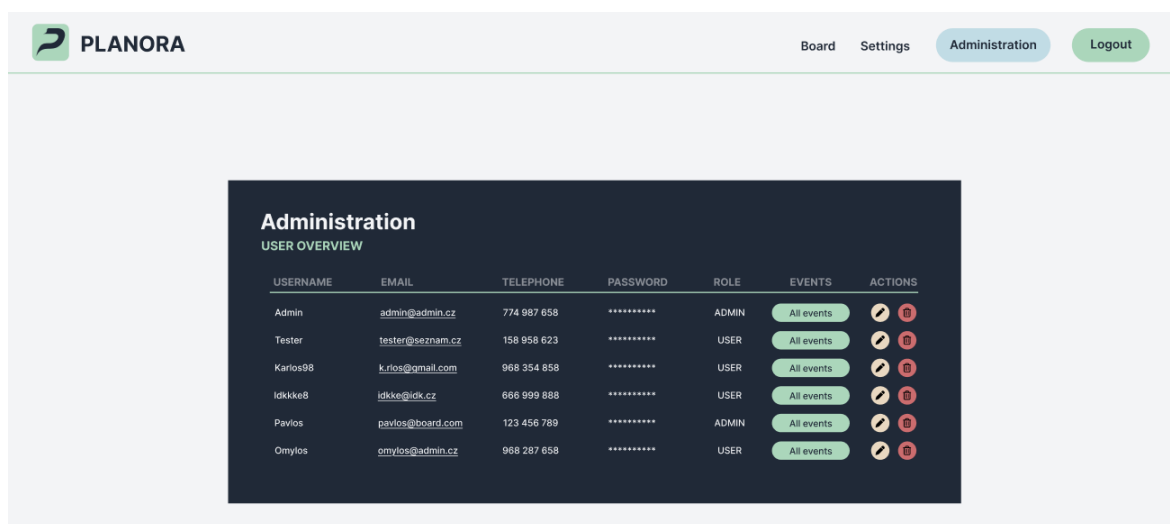


Obrázek 16 – Finální podoba uživatelského profilu

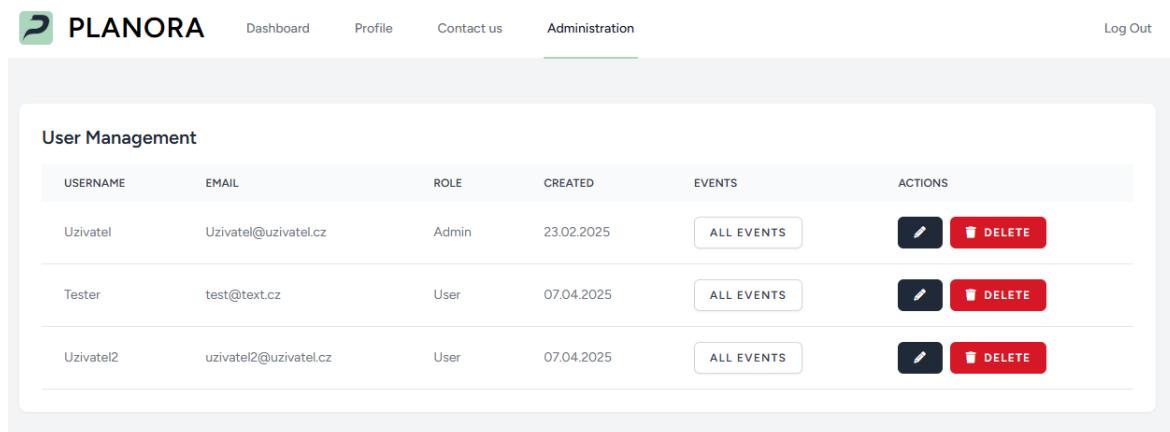
5.2.6 Administrace

V administrační části pro správu uživatelů došlo mezi návrhem a finální verzí k několika důležitým změnám. Stejně jako u ostatních částí aplikace byla upravena barevná paleta, přičemž finální verze využívá světlejší a vizuálně čistší vzhled. Výraznější změna však nastala v samotném obsahu tabulky. Z návrhu (Obrázek 17) byly odstraněny sloupce

s telefonním číslem a šifrovaným heslem, které by z bezpečnostního a praktického hlediska neměly být zobrazovány. Naopak do finální podoby (Obrázek 18) byl přidán údaj o datu vytvoření uživatelského účtu, což je užitečné zejména při správě většího množství uživatelů. Zůstaly zachovány všechny klíčové funkce: přímý přístup k přehledu harmonogramů konkrétního uživatele, možnost úpravy údajů a odstranění uživatele ze systému.



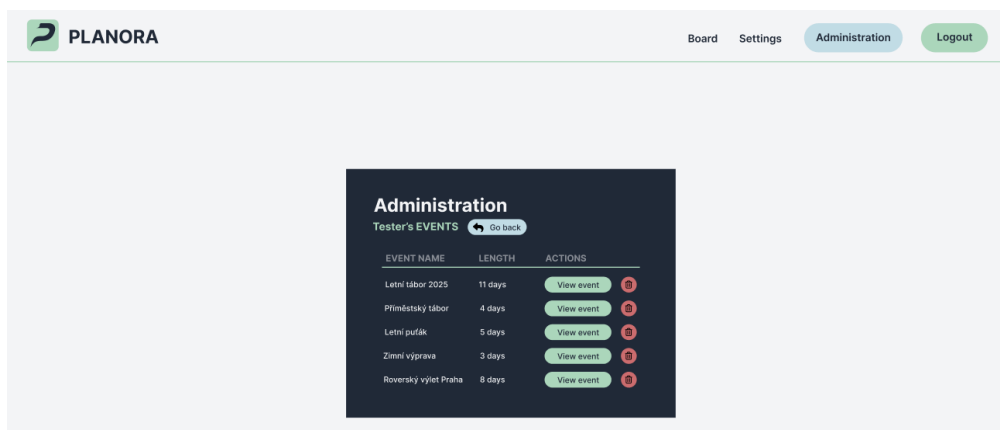
Obrázek 17 – Návrh administrace pro správu uživatelů



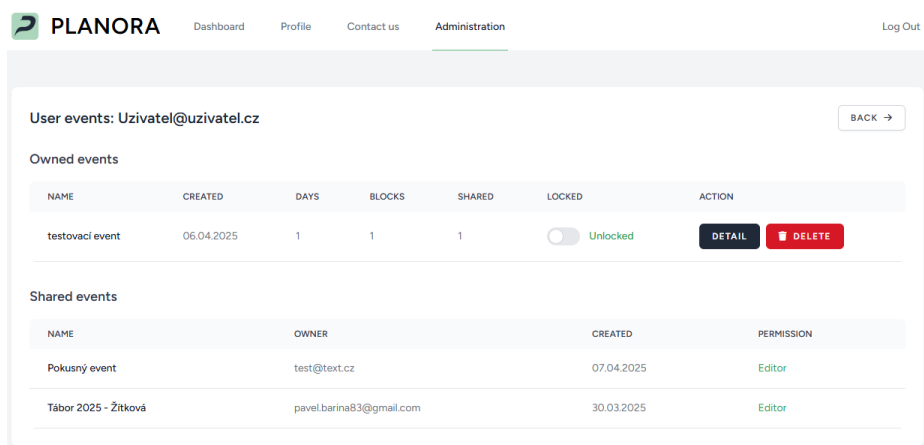
Obrázek 18 – Finální podoba administrace pro správu uživatelů

Přehled harmonogramů v administrační části prošel výraznou úpravou oproti původnímu návrhu (Obrázek 19). Ten zobrazoval pouze základní informace, jako je název harmonogramu a jeho délku. Také nabízel možnost prohlédnutí detailu nebo smazání daného záznamu. Finální verze (Obrázek 20) poskytuje uživatelům podstatně více informací: kromě názvu a data vytvoření je zobrazen také počet dnů, počet bloků, počet sdílení a stav uzamčení daného harmonogramu. Tím se výrazně zlepšila informovanost administrátora při správě většího množství dat. Kromě vlastních harmonogramů zobrazuje systém také ty, které byly

danému uživateli sdíleny jinými uživateli aplikace. Z každého záznamu je dále možné přejít do detailu harmonogramu (Obrázek 21), kde jsou uvedeny podrobnosti o vlastníkovi, celkové struktuře a sdílení. Také je zde přehledně zobrazen celý program ve vizuální podobě. Barevné schéma bylo sjednoceno se zbytkem aplikace.



Obrázek 19 – Návrh administrace pro správu harmonogramů



Obrázek 20 – Finální podoba administrace pro správu harmonogramů

The screenshot displays the PLANORA administration interface. At the top, there is a navigation bar with links: Dashboard, Profile, Contact us, and Administration (which is highlighted). A 'Log Out' link is also present. The main content area is titled 'Event detail: testovací event' and includes a 'BACK' button and a 'DELETE' button. Below this, there are two panels: 'Event information' and 'Shared with users'. The 'Event information' panel shows details such as Owner (Uzivatel@uzivatel.cz), Created date (06.04.2025 13:32:35), Number of days (5), and Number of blocks (5). The 'Shared with users' panel shows a user (pavel.barina83@gmail.com) with an 'Editor' role. Below these panels is a 'Harmonogram' section, which is a table showing a schedule for 5 days. The table has columns for time slots: BUDÍČEK (7:30), SNÍDANĚ (8:00), DOPolední BLOK (9:00), OBĚD (12:00), and ODPOlední BLOK (14:00). The rows represent different days and users, with activities like 'Buzení taborníků všichni', 'Snídaně všichni', 'Dopolední program 1 vedoucí 1', 'Oběd všichni', and 'Odpolední program 1 vedoucí 6'.

	BUDÍČEK 7:30	SNÍDANĚ 8:00	DOPolední BLOK 9:00	OBĚD 12:00	ODPOlední BLOK 14:00
1. DEN Tester	Buzení taborníků všichni	Snídaně všichni	Dopolední program 1 vedoucí 1	Oběd všichni	Odpolední program 1 vedoucí 6
2. DEN Admin	Buzení taborníků všichni	Snídaně všichni	Dopolední program 2 vedoucí 2	Oběd všichni	Odpolední program 2 vedoucí 7
3. DEN Uživatel	Buzení taborníků všichni	Snídaně všichni	Dopolední program 3 vedoucí 3	Oběd všichni	Odpolední program 3 vedoucí 8
4. DEN Uživatel2	Buzení taborníků všichni	Snídaně všichni	Dopolední program 4 vedoucí 4	Oběd všichni	
5. DEN tester	Buzení taborníků všichni	Snídaně všichni	Dopolední program 5 vedoucí 5	Oběd všichni	Odpolední program 5 vedoucí 10

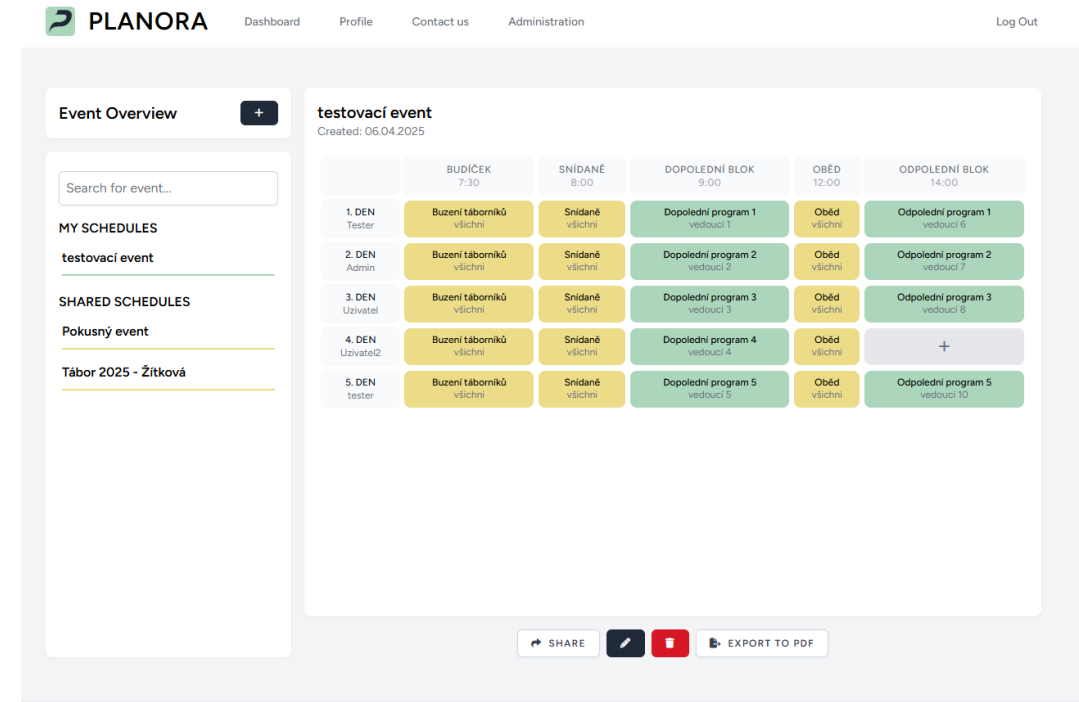
Obrázek 21 – Finální podoba administrace pro detailní přehled harmonogramu

5.2.7 Dashboard

V hlavním dashboardu nedošlo oproti původnímu návrhu k zásadním změnám. Koncept zobrazení harmonogramu ve formě tabulky s časovým rozvržením zůstal zachován téměř beze změny. Nejvýraznější úpravu představuje změna barevné palety a doplnění navigačního menu, kde byly do horní lišty přidány nové položky „Profile“ a „Contact us“, které dříve figurovaly pouze jako součást podstránky „Settings“. Dále došlo k drobným úpravám vzhledu tlačítek a dalších ovládacích prvků, nicméně celkové rozvržení a funkčnost dashboardu zůstaly věrné původnímu návrhu. Výsledná podoba tak zachovává původní myšlenku, ale lépe zapadá do sjednoceného vizuálního stylu celé aplikace. Drobné úpravy vzhledu a rozložení jsou patrné na Obrázku 22 a 23



Obrázek 22 – Návrh dashboardu



Obrázek 23 – Finální podoba dashboardu

6 IMPLEMENTACE

Tato kapitola popisuje praktickou realizaci webové aplikace Planora. Zaměřuje se na architekturu systému, způsob vytvoření projektu, implementaci backendové a frontendové logiky, zabezpečení dat a samotné nasazení aplikace do produkčního prostředí. Jsou zde popsány klíčové technické kroky, použité nástroje a hlavní funkce aplikace.

6.1 Architektura systému

Aplikace Planora je postavena na frameworku Laravel, který využívá architekturu MVC (Model–View–Controller). Tento přístup odděluje zodpovědnosti jednotlivých částí systému a přispívá k přehlednosti i snadné údržbě.

6.1.1 Vytvoření projektu

Aplikace byla vytvořena klasickým způsobem přes příkazový řádek s využitím nástroje Composer a Laravel Installer. Na základě oficiální dokumentace Laravelu byl projekt inicializován příkazem:

```
laravel new sprava-harmonogramu
```

Tímto příkazem se vytvořila kompletní adresářová struktura Laravel projektu, připravená pro další vývoj. Využití této metody zajišťuje správnou inicializaci základních komponent frameworku a usnadňuje budoucí rozšiřování projektu.

6.1.2 Architektura MVC

Laravel staví na architektuře Model–View–Controller, která umožňuje přehledné oddělení jednotlivých částí systému:

- Model zajišťuje práci s databází. Pomocí Eloquent ORM byly vytvořeny modely pro uživatele (User), role (Role), harmonogramy (Schedule), dny (Day), bloky (Block) a aktivity (ScheduleData). Každý model odpovídá jedné databázové tabulce a umožňuje snadné provádění operací jako je vkládání, úprava nebo mazání záznamů.
- View představuje šablony uživatelského rozhraní. V aplikaci byly využity Blade šablony, které umožňují vytvářet přehledné a bezpečné HTML výstupy s podporou dědičnosti a vkládání komponent. Zobrazení jednotlivých stránek (např. přihlašovací

formulář, dashboard nebo administrace) bylo realizováno právě pomocí těchto Blade souborů.

- Controller slouží jako prostředník mezi modelem a zobrazením. Zajišťuje zpracování uživatelských požadavků, volá odpovídající modelové operace a připravuje data pro zobrazení. Například třída ScheduleController zpracovává logiku vytváření, úprav nebo mazání harmonogramů.

6.1.3 Struktura projektu

Základní adresářová struktura Laravelu byla dodržena v následující podobě:

- app/Models/ – obsahuje definice datových modelů využívajících Eloquent ORM.
- app/Http/Controllers/ – obsahuje controllery, které zajišťují aplikační logiku.
- resources/views/ – zde se nachází Blade šablony pro jednotlivé stránky uživatelského rozhraní.
- routes/web.php – soubor definující webové routy a mapující URL na konkrétní controllery a metody.
- database/migrations/ – obsahuje soubory definující strukturu databázových tabulek v čitelném PHP formátu.
- .env – soubor pro nastavení konfigurace prostředí (např. e-mailová služba, připojení k databázi).

Tato struktura umožňuje přehlednou organizaci kódu, snazší údržbu a rozšiřitelnost systému. Díky využití Laravel Artisan CLI bylo možné rychle generovat modely, controllery, migrace i další součásti systému.

6.2 Backendová implementace

Backend aplikace Planora je postaven na frameworku Laravel, který nabízí moderní nástroje pro správu uživatelů, zabezpečení, správu databáze a směrování požadavků. Použitím Eloquent ORM bylo možné definovat relační modely a jejich vzájemné vztahy bez nutnosti psát komplexní SQL dotazy. V této části jsou popsány hlavní oblasti backendové logiky aplikace.

6.2.1 Autentizace a registrace uživatelů

Pro správu přihlášení, registrace a obnovy hesla byla využita vestavěná autentizační funkcionality Laravelu rozšířená o ověření e-mailové adresy. Po registraci je uživateli zaslán ověřovací e-mail s odkazem, který musí potvrdit, než se mu zpřístupní hlavní část aplikace. Tím je zajištěna validita e-mailových adres a zvýšena bezpečnost celého systému.

Aplikace pro tyto účely využívá middleware *auth* a *verified*, které kontrolují přihlášení a potvrzení e-mailu. Všechny klíčové routy aplikace jsou tak chráněny proti neoprávněnému přístupu.

6.2.2 Uživatelské role a oprávnění

Aplikace rozlišuje dvě základní role – *user* a *admin*. Role jsou uloženy v tabulce *roles* a přiřazeny jednotlivým uživatelům přes *role_id*.

Rozlišení oprávnění je realizováno pomocí middleware a podmínek ve view i controllerech. Například přístup k administrátorské sekci je možný pouze pro uživatele s rolí *admin*, zatímco běžní uživatelé mají omezený přístup pouze ke svým datům nebo k datům, která jim byla sdílena.

6.2.3 Databázová logika a Eloquent ORM

Datová vrstva aplikace je postavena na Eloquent ORM, které umožňuje snadnou práci s databází prostřednictvím PHP objektů. Například práce s harmonogramy, bloky nebo aktivitami se realizuje přes relační metody jako *hasMany*, *belongsTo* nebo *hasOne*.

Typickým příkladem může být model *Schedule*, který definuje vztah k uživateli *belongsTo(User::class)* a ke svým dnům *hasMany(Day::class)*, což umožňuje srozumitelné dotazy typu:

```
$schedule->days()->with('blocks.scheduleData')->get();
```

Tímto způsobem je možné načíst celý harmonogram včetně bloků a jednotlivých aktivit v jediném dotazu.

6.2.4 Uzamykání harmonogramu

Aby se předešlo kolizím při úpravách harmonogramu více uživateli současně, byl implementován mechanismus uzamčení. Každý harmonogram má ve své databázové struktuře sloupec *locked_by*, který uchovává ID uživatele, jenž má harmonogram aktuálně „otevřený“ pro editaci.

Při pokusu o úpravu aplikace kontroluje, zda je harmonogram zamčen a kým. Pokud je uzamčen jiným uživatelem, úprava není umožněna a zobrazí se varování. Pokud je harmonogram volný nebo jej má zamčený aktuální uživatel, změny jsou povoleny. Tento mechanismus zabraňuje nežádoucím konfliktům a zajišťuje integritu dat.

6.2.5 Administrační část

Další významnou součástí backendu je administrační sekce, která je přístupná pouze uživatelům s rolí administrátora. Role uživatele je určena vazbou na tabulku *roles*, kde se nachází např. hodnota „Admin“. Oprávnění jsou ověřována prostřednictvím podmínek v kontrolerech a šablonách, a zejména pomocí vlastního middleware s názvem AdminMiddleware.

Tento middleware zajišťuje, že přístup k administraci bude automaticky odepřen uživatelům bez potřebné role. V případě, že podmínka není splněna, je uživatel přesměrován zpět na dashboard s chybovým hlášením. Kontrola probíhá následovně:

```
public function handle(Request $request, Closure $next): Response
{
    if (auth()->check() && auth()->user()->role->nazev ===
        'Admin') {
        return $next($request);
    }

    return redirect('/dashboard')->with('error', 'Access
        denied.');
```

AdminMiddleware je aplikován na všechny routy spadající pod prefix */admin*. Díky tomu je dosaženo centrálního řízení přístupu bez nutnosti opakovat podmínky v každém kontroleru. V kombinaci s Blade podmínkami na frontendu tak systém zajišťuje, že administraci může využívat výhradně oprávněná osoba.

Oprávněná osoba má poté přístup k několika funkcím, které vedou například k zobrazení seznamu všech uživatelů a všech vytvořených harmonogramů v systému. Pomocí těchto funkcí může správce systému:

- mazat nebo upravovat uživatelské účty
- kontrolovat nebo mazat harmonogramy jiných uživatelů
- zobrazovat sdílené relace mezi uživateli (včetně práv čtení / úprav)

- ručně odemknout zamčené harmonogramy, pokud došlo k selhání uvolnění zámku

6.3 Frontendová implementace

Uživatelské rozhraní aplikace Planora je vytvořeno s využitím Blade šablonovacího systému, který je součástí frameworku Laravel. Blade umožňuje snadno kombinovat HTML strukturu s PHP logikou, přičemž zachovává přehlednost a bezpečnost výstupu. Výhodou je také podpora dědičnosti šablon a jejich komponentové rozdělení, což výrazně usnadňuje údržbu i rozšiřování kódu.

Celý frontend je stylován pomocí frameworku Tailwind CSS, který umožňuje rychlé a přehledné definování vzhledu bez nutnosti vytvářet vlastní CSS třídy. Responzivita je zajištěna přímo pomocí Tailwind tříd jako *sm:*, *md:* nebo *lg:*, čímž je zajištěno správné zobrazení jak na počítačích, tak mobilních zařízeních.

6.3.1 Hlavní layout a struktura stránky

Blade šablony tvoří základní strukturu frontendové části aplikace. Všechny stránky využívají společný layout definovaný v souboru `layouts/app.blade.php`, kde jsou umístěny společné části jako hlavička, navigace, import stylů a skriptů včetně komponenty *x-toaster-hub* pro zobrazení toast notifikací. Obsah jednotlivých stránek se do této šablony vkládá následovně:

```
<main class="flex-grow">
    {{ $slot }}
</main>
```

Tento přístup umožňuje každé podstránce vložit svůj specifický obsah bez nutnosti duplikovat společné části jako hlavičku nebo patičku.

6.3.2 Navigační menu

Navigace je implementována v samostatné šabloně `layouts/navigation.blade.php` a pomocí `@include` se vkládá do hlavního layoutu. Navigace zohledňuje roli přihlášeného uživatele. Běžným uživatelům zobrazuje odkazy na dashboard, profil a kontaktní formulář. Administrátorům navíc umožňuje přístup do administrace.

Ukázka podmíněného zobrazení v navigaci:

```
@if (Auth::check() && Auth::user()->role->nazev === 'Admin')
    <div class="pt-2 pb-3 space-y-1">
        <x-responsive-nav-link :href="route('admin.index')"
            :active="request()->routeIs('admin.*')">
            {{ __('Administration') }}
        </x-responsive-nav-link>
    </div>
@endif
```

Díky tomu má každý typ uživatele přístup pouze k relevantním částem aplikace.

6.3.3 Dashboard a zobrazení harmonogramu

Po přihlášení uživatele je hlavní stránkou aplikace *dashboard.blade.php*. Tato stránka zobrazuje seznam harmonogramů, které uživatel vlastní nebo které mu byly nasdíleny. Harmonogramy jsou v rámci postranního menu rozděleny na vlastní a sdílené, přičemž v každé kategorii je zobrazen jejich název. Uživatel může pomocí vyhledávacího pole filtrovat harmonogramy podle názvu bez nutnosti obnovování stránky.

Kliknutím na konkrétní položku v seznamu je uživatel přesměrován na detail vybraného harmonogramu. V případě, že uživatel nemá žádný harmonogram, je mu zobrazena informace o prázdném stavu:

```
@forelse($schedules as $schedule)
    <a href="{{ route('schedule.show', $schedule) }}">
        <h4>{{ $schedule->name }}</h4>
    </a>
@empty
    <p>You don't have any events yet</p>
@endforelse
```

Vizualizace samotného harmonogramu je realizována tabulkovou strukturou, kde řádky reprezentují jednotlivé dny a sloupce časové bloky (například dopoledne, odpoledne, večer). Do těchto buněk jsou vloženy konkrétní aktivity, které obsahují název, popis, zodpovědnou osobu a další metadata. Výpis probíhá iterativně s využitím smyček *@foreach* a přístupem ke vztahům mezi modely, např. *\$day->blocks* nebo *\$block->scheduleData*.

6.3.4 Uzamykání harmonogramu

Při editaci nebo mazání harmonogramu hraje důležitou roli mechanismus zamykání, jehož hlavní část je realizována na straně backendu. Ten brání současnému přístupu více uživatelů ke stejnému harmonogramu, čímž předchází kolizím při ukládání dat. Frontendová část tento stav pouze reflektuje a zajišťuje uživatelskou informovanost.

Například při pokusu o otevření modálního okna pro úpravu harmonogramu je odeslán požadavek na backend, který vrátí buď data harmonogramu, nebo chybu s informací, že záznam je aktuálně uzamčen jiným uživatelem. Pokud je harmonogram uzamčen, formulář nenačte a zobrazí se překryvná informační vrstva s hláškou „Event is currently being edited by someone else.“. Uživatel má možnost pokusit se akci zopakovat kliknutím na tlačítko „Try again“, které odešle nový požadavek a ověří, zda je již harmonogram opět dostupný.

Tato logika je implementována například v souboru `editEvent.js`, kde metoda `openEditModal()` kontroluje odpověď serveru. Pokud server vrátí chybu, je v Blade šabloně modalu aktivován prvek `#errorOverlay`, který informuje uživatele a zároveň skryje obsah. Zobrazení chyby je řízeno následovně:

```
if (data.error) {  
    document.getElementById('errorOverlay').style.display =  
    'flex';  
  
    document.querySelector('button[type="submit"][form="editEventForm"]').style.display = 'none';  
}
```

Zobrazené okno pro editaci tak není nikdy aktivováno, dokud jiný uživatel nedokončí svoji práci. Při zavření modálního okna navíc frontend pomocí `fetch` požadavku zavolá endpoint pro odemčení harmonogramu (`/schedule/{id}/unlock`), čímž uvolní zámek a umožní editaci dalšímu uživateli.

6.3.5 Sdílení a export harmonogramu

Součástí uživatelského rozhraní je funkce sdílení harmonogramu s dalšími uživateli systému. Uživatel, který harmonogram vlastní, má pod jeho detailem přístup k několika akčním tlačítkům, mezi něž patří mimo jiné editace, mazání, export a právě sdílení. Po kliknutí na tlačítko „Share“ se otevře modální okno, v němž může uživatel specifikovat e-mailové adresy osob s nimiž chce harmonogram sdílet a oprávnění (pouze pro čtení, nebo editor).

Pokud je harmonogram nasdílen s oprávněním pro úpravy, může jeho vlastník při editaci konkrétní aktivity přiřadit tuto aktivitu právě jednomu ze sdílených uživatelů. Toto přiřazení probíhá v rámci rozhraní pro úpravu harmonogramu a pracuje se zde s výběrovým polem, které obsahuje pouze ty uživatele, kterým byl harmonogram sdílen s oprávněním k úpravám. Tímto způsobem je zajištěno, že k jednotlivým částem harmonogramu mají přístup pouze ti uživatelé, kterým byla výslovně svěřena odpovědnost.

Export harmonogramu je umožněn prostřednictvím tlačítka „Export“, které vygeneruje přehledný PDF dokument obsahující celý rozvrh ve strukturované podobě, která je nastavena v souboru *schedule.blade.php*. Tento dokument slouží zejména pro offline použití nebo tisk, například jako fyzická kopie denního programu tábora.

6.3.6 Administrační část

V případě, že je přihlášen administrátor, zobrazí se mu v hlavním menu další položka „Administration“. Tato sekce umožňuje správu uživatelů a harmonogramů přímo přes webové rozhraní. Administrátor vidí seznam všech registrovaných uživatelů s jejich e-mailem a datem registrace. U každého uživatele jsou k dispozici akční tlačítka pro úpravu nebo smazání účtu.

V rámci administračního rozhraní je dále možné zobrazit přehled všech harmonogramů včetně informací o jejich vlastníkově, počtu dnů, bloků a sdílených uživatelů. Při kliknutí na konkrétní záznam je administrátor přesměrován do detailu harmonogramu, kde má přístup pouze pro čtení. Veškeré administrátorské rozhraní je vytvořeno pomocí Blade šablon a zachovává jednotný vizuální styl s hlavní aplikací.

Zobrazení administračních prvků v navigačním menu je řízeno podmínkou, která kontroluje, zda má uživatel přiřazenu roli „Admin“. Tato podmínka je již podrobněji rozebrána v části popisující navigační menu výše.

6.4 Zabezpečení aplikace

Bezpečnost byla při návrhu a vývoji aplikace Planora jednou z priorit. Aplikace implementuje několik úrovní ochrany, které pokrývají ověřování uživatelů, řízení přístupu, bezpečné ukládání hesel a ochranu proti běžným typům útoků. Laravel jako použitý framework nabízí řadu vestavěných bezpečnostních mechanismů, které byly v projektu využity.

6.4.1 Autentizace a autorizace uživatelů

Pro autentizaci byla využita vestavěná funkcionalita Laravelu, která zahrnuje registraci, přihlášení, potvrzení e-mailu a reset hesla. Přístup k aplikaci je umožněn pouze ověřeným uživatelům a většina aplikačních rout je chráněna middlewarem *auth* a *verified*.

Oprávnění k jednotlivým částem aplikace jsou řízena na základě uživatelské role, která je definována v databázi. V systému existují dvě role – běžný uživatel a administrátor. Administrátor má přístup ke všem harmonogramům a správě uživatelů, zatímco běžní uživatelé mají přístup jen ke svým a sdíleným harmonogramům. Tyto rozdíly jsou vyhodnocovány v controllerech a také na úrovni zobrazení pomocí podmínek ve Blade šablonách.

6.4.2 Bezpečné ukládání hesel

Veškerá hesla jsou při registraci a změně hesla automaticky ukládána v hashované podobě pomocí algoritmu Bcrypt. Laravel při tom využívá komponentu *Hash*, která zabezpečuje hesla proti zpětnému dekódování. V databázi se tak nikdy nenachází žádná hesla v čitelné podobě.

Při registraci je zároveň kladen důraz na kvalitu hesla. Formulář kontroluje, zda zadané heslo splňuje minimální bezpečnostní kritéria. Heslo musí obsahovat alespoň osm znaků, jedno velké a malé písmeno, číslici a speciální znak. Tato pravidla jsou validována jak na straně frontendového formuláře, tak i serverově při zpracování požadavku. Tím je zajištěno, že do systému nevstupují slabá nebo snadno prolomitelná hesla.

6.4.3 Řízení přístupu a oprávnění

Přístup k jednotlivým částem aplikace je omezen nejen podle uživatelské role, ale i podle vlastnictví nebo sdílení harmonogramu. U harmonogramů jsou uživatelé oprávnění upravovat pouze ty, které sami vytvořili nebo které jim byly nasdíleny s právem úprav. Při pokusu o neoprávněný přístup jsou uživatelé přesměrováni.

6.4.4 Ochrana proti útokům

Aplikace je chráněna proti nejběžnějším webovým hrozbám. Laravel automaticky escapuje výstup ve všech Blade šablonách, čímž minimalizuje riziko XSS útoků. Dále využívá CSRF tokeny, které se generují pro každý formulář a jsou automaticky ověřovány při odeslání. Tím je zajištěno, že požadavky na server pocházejí z oprávněného zdroje.

Kromě toho je využíváno také omezování přístupu k citlivým routám pomocí middleware a kontrol přihlášení, čímž je zamezeno nepřihlášeným uživatelům v přístupu k obsahu. Validace vstupních dat probíhá na serverové straně a tím se zvyšuje robustnost aplikace i v případě manipulace s frontendem.

6.4.5 Zamykání harmonogramů

Specifickou součástí ochrany dat v aplikaci Planora je mechanismus zamykání harmonogramů, který slouží k zajištění konzistence při souběžné práci více uživatelů. Účelem této funkce je zabránit tomu, aby více uživatelů upravovalo stejný harmonogram současně, což by mohlo vést k přepsání nebo ztrátě dat.

Při pokusu o úpravu harmonogramu je nejprve na straně serveru ověřeno, zda není záznam právě upravován jiným uživatelem. Pokud ano, klientský požadavek obdrží informaci o zámku a uživateli je zobrazeno upozornění s možností opakování pokusu o přístup. Harmonogram je po úspěšném otevření automaticky označen jako uzamčený pomocí záznamu v databázovém poli *locked_by*, které obsahuje ID uživatele provádějícího úpravy. Po ukončení práce se zámek uvolní.

Z hlediska správy je možné harmonogram v krajním případě odemknout ručně prostřednictvím administračního rozhraní. Tento přístup kombinuje technické ověření na backendu s odpovídající logikou na frontendové straně, čímž chrání data před nechtěným přepisem v prostředí s více uživateli.

6.5 Testování aplikace

Testování aplikace probíhalo ve více fázích. Nejprve byla dokončena první pracovní verze, která obsahovala veškeré plánované funkce. Následně byla provedena manuální kontrola každé části přímo v aplikaci, při níž byly odhaleny a opraveny první nedostatky. Po interním ověření byla aplikace nasazena na produkční server a zpřístupněna širší skupině uživatelů. Primárně spolužákům, kolegům z práce a především vedoucím z domovského skautského střediska, pro které je aplikace primárně určena.

Účastníci testování dostali za úkol aktivně aplikaci procházet a hledat případné chyby nebo problematické části. Tento přístup pomohl identifikovat řadu konkrétních problémů. Mezi problémy patřilo například opakované odeslání ověřovacího e-mailu po registraci, nedostatečné ztmavení pozadí za modálními okny na mobilních zařízeních nebo nefunkční

drag and drop v jednom z modálních dialogů. Všechny tyto chyby byly následně analyzovány a opraveny. Nakonec byla nasazena upravená verze.

Cílem testování bylo ověřit funkčnost aplikace z pohledu běžného uživatele, zajistit správné propojení a integritu dat a zároveň dosáhnout maximálně plynulého a srozumitelného uživatelského rozhraní. Důraz byl kladen na stabilitu systému a na použitelnost na různých typech zařízení.

6.6 Nasazení aplikace

Aplikace Planora je v současné době nasazena v ostrém prostředí a dostupná veřejnosti na adrese <https://planora.pavel-barina.cz>. Nasazení bylo realizováno na webhostingovém serveru spravovaném pomocí HestiaCP, kde má aplikace vyhrazenou vlastní databázi MySQL a běží v prostředí podporujícím PHP 8.1.

Pro potřeby provozu nebyly nutné žádné zásahy do kódu. Po nakonfigurování `.env` souboru, připojení k databázi a vygenerování klíče aplikace pomocí příkazu `php artisan key:generate` byla aplikace připravena k použití. Migrace databázových tabulek byla provedena pomocí Laravel příkazů `migrate` a `db:seed`.

Aplikace je plně připravena pro vícenásobné použití více uživateli současně. Díky architektuře postavené na Laravelu a podpoře session řízení je možné, aby uživatelé pracovali s harmonogramy paralelně z různých zařízení nebo prohlížečů bez negativního ovlivnění běhu aplikace. Mechanismus zamykání harmonogramů navíc zajišťuje bezpečnost při souběžné editaci sdílených záznamů.

Do budoucna lze aplikaci snadno přesunout na libovolný jiný server s podporou Laravelu, případně nasadit pomocí kontejnerizační technologie (např. Docker) nebo spravovaného prostředí jako Laravel Forge, pokud by bylo potřeba lépe škálovat nebo automatizovat nasazení.

7 UŽIVATELSKÁ DOKUMENTACE

Aplikace Planora je určena především vedoucím a organizátorům zotavovacích akcí, kteří potřebují efektivní nástroj pro plánování programu. Uživatelské rozhraní je navrženo s důrazem na jednoduchost, přehlednost a intuitivní ovládání. Tato kapitola shrnuje základní funkcionalitu aplikace z pohledu běžného uživatele.

7.1 Registrace a přihlášení

Před prvním použitím je nutné se zaregistrovat pomocí e-mailu a hesla. Při registraci systém vyžaduje silné heslo obsahující kombinaci malých a velkých písmen, číslic a speciálních znaků. Po registraci je uživateli odeslán potvrzovací e-mail s odkazem pro ověření účtu. Přístup do aplikace je umožněn až po jeho potvrzení.

7.2 Vytváření a správa harmonogramu

Po přihlášení je uživateli zpřístupněn dashboard, kde může vytvářet nové harmonogramy pomocí tlačítka „+“. Harmonogram je možné kdykoli upravit, přejmenovat nebo smazat. Při editaci se pracuje s přehlednou tabulkovou strukturou, kde řádky reprezentují dny a sloupce jednotlivé bloky programu. U každé aktivity lze vyplnit název, popis, počet osob, materiál a zodpovědnou osobu.

7.3 Sdílení harmonogramu

Každý harmonogram může být sdílen s dalšími uživateli prostřednictvím jejich e-mailových adres. Sdílení probíhá přes modální okno, kam se vkládají e-maily oddělené čárkou a vybírá se úroveň oprávnění – pouze čtení nebo možnost úprav. Uživatelé s oprávněním pro úpravy mohou být následně přiřazeni k jednotlivým aktivitám, které pak mohou upravovat pouze oni.

7.4 Export do PDF

Hotový harmonogram lze kdykoli stáhnout ve formátu PDF pomocí tlačítka „Export“. Exportovaný dokument zachovává strukturu harmonogramu včetně přiřazených odpovědných osob a je vhodný pro tisk nebo offline použití.

7.5 Uživatelský profil

V sekci „Profile“ může uživatel upravit své jméno, e-mailovou adresu a heslo. K dispozici je také možnost smazat svůj účet, což nevratně odstraní všechna jeho data. Smazání účtu vyžaduje potvrzení aktuálním heslem.

7.6 Kontaktní formulář

Přihlášení uživatelé mohou kontaktovat správce aplikace prostřednictvím formuláře dostupného v hlavním menu. Tento formulář slouží pro dotazy, zpětnou vazbu nebo nahlášení technických problémů.

7.7 Administrace systému

Uživatelé s administrátorským oprávněním mají navíc přístup k administrační sekci, kde mohou spravovat všechny uživatelské účty a harmonogramy v systému. V části „Administration“ je zobrazen seznam registrovaných uživatelů včetně základních údajů a datumu registrace. Administrátor má možnost upravit nebo smazat jakýkoli účet.

Součástí administrace je také přehled všech vytvořených harmonogramů. Zde jsou nejen vlastní harmonogramy, ale i ty, které vytvořili ostatní uživatelé. U každého harmonogramu jsou zobrazeny informace o vlastníkovi, počtu dnů, bloků, sdílení a stav uzamčení. Administrátor může harmonogram ručně uzamknout nebo odstranit, případně vstoupit do jeho detailního náhledu, kde je celý program vizuálně zobrazen.

Díky těmto nástrojům má administrátor plnou kontrolu nad obsahem systému a může zasahovat v případě potřeby. Například při neaktivních účtech nebo při technických potížích s editací harmonogramu.

ZÁVĚR

Tato bakalářská práce se zabývala návrhem a realizací webové aplikace pro správu harmonogramů zotavovacích akcí, jako jsou letní tábory a akce podobného charakteru. Cílem práce bylo zjednodušit plánování programu a sestavování denního rozvrhu pro vedoucí těchto akcí. Důraz byl kladen nejen na vytvoření funkčního nástroje, ale také na návrh intuitivního a uživatelsky přívětivého rozhraní, které odpovídá reálným potřebám organizátorů a vedoucích zotavovacích akcí.

V teoretické části práce bylo vysvětleno, co je zotavovací akce a jaké požadavky je nutné dodržet při jejich plánování. Dále byla provedena analýza dostupných frameworků s cílem identifikovat ten, který nejlépe odpovídá požadavkům plánované aplikace. Nakonec byly detailně popsány technologie použité při realizaci aplikace – Laravel jako hlavní framework pro vývoj aplikace a MySQL jako databázová platforma.

Praktická část práce začala analýzou požadavků, která zahrnovala definici cílové skupiny, funkčních a nefunkčních požadavků a návrh uživatelských rolí. V části věnované návrhu aplikace byly popsány rozdíly mezi původním grafickým návrhem a finální implementací. Kapitola implementace se podrobně zaměřila na nejdůležitější funkce v rámci backendové a frontendové části aplikace. Dále byly popsány bezpečnostní prvky a způsob nasazení aplikace do provozního prostředí. Součástí praktické části je také stručná uživatelská příručka ve formě dokumentace.

Po dokončení vývoje byla aplikace podrobena víceúrovňovému testování, které zahrnovalo manuální ověření funkcionalit autorem a zpětnou vazbu od reálných uživatelů z cílové skupiny. Na základě připomínek byly odstraněny zjištěné nedostatky a provedeny potřebné úpravy, aby aplikace odpovídala očekáváním uživatelů a byla připravena pro ostré nasazení.

Závěrem lze říci, že všechny stanovené cíle byly splněny. Výsledná aplikace poskytuje přehledný, funkční a praktický nástroj pro efektivní správu harmonogramů zotavovacích akcí. Další rozšíření by mohlo zahrnovat například přidání tmavého režimu, podporu více jazyků, možnost přihlášení prostřednictvím Google účtu nebo systému SkautIS, integraci s Google kalendářem, implementaci notifikací a případný vývoj mobilní aplikace pro snadnější přístup a lepší uživatelský zážitek na mobilních zařízeních.

SEZNAM POUŽITÉ LITERATURY

- [1] AION CS - INFO@AION.CZ. *Vyhláška Ministerstva zdravotnictví o hygienických požadavcích na zotavovací akce pro děti*. Online. 2001. Dostupné z: <https://www.zakonyprolidi.cz/cs/2001-106>. [cit. 2025-03-03].
- [2] KRAJSKÁ HYGIENICKÁ STANICE MORAVSKOSLEZSKÉHO KRAJE SE SÍDLEM V OSTRAVĚ. *KHSMK*. Online. 2009. Dostupné z: <https://www.khsova.cz/obcanum/detail/615>. [cit. 2025-03-03].
- [3] HLAVINKA, Filip. *Předpis: Směrnice ke zdravotnickému zabezpečení akcí a k souvisejícím kvalifikacím*. Online. 2024. Dostupné z: <https://krizovatka.skaut.cz/spisovna/smernice/vzdelavani/smernice-ke-zdravotnickemu-zabezpeceni-akci/2024-09/predpis>. [cit. 2025-03-03].
- [4] SEKRETARIÁT ČRDM. *Zotavovací akce*. Online. 2017. Dostupné z: <https://poradna.crdm.cz/tabory/zotavovaci-akce-565>. [cit. 2025-03-03].
- [5] SENCHÁ, Team. *Web Application Development – Top 10 Frameworks in 2025*. Online. 2024. Dostupné z: <https://www.sencha.com/blog/web-application-development-top-frameworks/>. [cit. 2025-03-06].
- [6] SKOUMAL, Vladislav. *Jak vybrat framework?* Online. 2015. Dostupné z: <https://www.skoumal.com/cs/jak-vybrat-framework/>. [cit. 2025-03-06].
- [7] LARAVEL. *Laravel Documentation*. Online. Dostupné z: <https://laravel.com/docs/12.x>. [cit. 2025-03-08].
- [8] RUNCLOUD. *What is Laravel? Explain it like I'm five*. Online. 2024. Dostupné z: <https://runcloud.io/blog/what-is-laravel>. [cit. 2025-03-08].
- [9] AHMED, Shahzeb. *Laravel Ecommerce Packages or Ready-Made CMS: Which One to Choose for Online Store?* Online. 2023. Dostupné z: <https://www.cloudways.com/blog/laravel-ecommerce/>. [cit. 2025-03-08].
- [10] CASTELO, André. *Laravel API Tutorial: How to Build and Test a RESTful API*. Online. Dostupné z: <https://www.toptal.com/laravel/restful-laravel-api-tutorial>. [cit. 2025-03-08].
- [11] SHAH, Jigar. *Advantages And Disadvantages Of Laravel Development*. Online. 2024. Dostupné z: <https://dolphinwebsolution.com/blog/advantages-and-disadvantages-of-laravel-development/>. [cit. 2025-03-08].
- [12] STAFF, Guru. *The Pros and Cons of Laravel*. Online. 2021. Dostupné z: <https://www.guru.com/blog/the-pros-and-cons-of-laravel/>. [cit. 2025-03-08].
- [13] DESHPANDE, Chinmayee. *What Is Angular? A Guide to the Angular Framework*. Online. 2024. Dostupné z: <https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular>. [cit. 2025-03-09].
- [14] PANDYA, Manav. *How Long Does it Take to Learn Angular?* Online. 2024. Dostupné z: <https://www.scholarhat.com/tutorial/angular/how-long-will-it-take-to-learn-angular>. [cit. 2025-03-09].
- [15] ANGULAR TEAM. *Runtime performance optimization*. Online. Dostupné z: <https://angular.dev/best-practices/runtime-performance>. [cit. 2025-03-09].
- [16] SYNAPSE TEAM. *What is the relationship between angular and full stack development?* Online. 2023. Dostupné z: <https://synapseteam.com/q/what-is-the-relationship-between-angular-and-full-stack-development/>. [cit. 2025-03-09].

- [17] MULLASSERY, Rakesh. *Understanding Security in Angular Applications: Best Practices and Implementation*. Online. 2024. Dostupné z: <https://medium.com/@rakesh.mr.0341/understanding-security-in-angular-applications-best-practices-and-implementation-edc773863d8a>. [cit. 2025-03-09].
- [18] ENGINEERING. *The Good and the Bad of Angular Development*. Online. 2022. Dostupné z: <https://www.altexsoft.com/blog/the-good-and-the-bad-of-angular-development/>. [cit. 2025-03-09].
- [19] GŁUSZEK, Łukasz. *What Apps Can I Build With Angular? A Short Guide for Entrepreneurs and Managers*. Online. 26.11.2024. Dostupné z: <https://www.netguru.com/blog/angular-mobile-apps>. [cit. 2025-03-09].
- [20] DESHPANDE, Chinmayee. *The Best Guide to Know What Is React*. Online. 2024. Dostupné z: <https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs>. [cit. 2025-03-10].
- [21] MÁCA, Jindřich. *Lekce 1 - Úvod do React*. Online. Dostupné z: <https://www.itnetwork.cz/uvod-do-react>. [cit. 2025-03-10].
- [22] NOBLE DESKTOP. *How Difficult is it to Learn React?* Online. Dostupné z: <https://www.nobledesktop.com/learn/react/how-difficult-is-it-to-learn-react>. [cit. 2025-03-10].
- [23] META OPEN SOURCE. *Render and Commit*. Online. Dostupné z: <https://react.dev/learn/render-and-commit>. [cit. 2025-03-10].
- [24] VUNNAM, Naresh. *ReactJS Security Best Practices*. Online. 2023. Dostupné z: <https://medium.com/whatfix-techblog/reactjs-security-best-practices-6542b71a5577>. [cit. 2025-03-10].
- [25] GOR, Chintan. *Why ReactJS Has Become Popular Among Developers?* Online. 2024. Dostupné z: <https://www.esparkinfo.com/blog/why-reactjs-become-popular.html>. [cit. 2025-03-10].
- [26] SETHI, Mohak. *Types of React Apps and Their Popular Examples*. Online. Dostupné z: <https://www.turing.com/kb/types-of-react-apps-and-their-popular-examples>. [cit. 2025-03-10].
- [27] T. WILLIAMS, Alexander. *The Pros and Cons of Using React Today*. Online. 2023. Dostupné z: <https://thenewstack.io/the-pros-and-cons-of-using-react-today/>. [cit. 2025-03-10].
- [28] VUE.JS. *Vue.js Documentation*. Online. Dostupné z: <https://vuejs.org/guide/introduction>. [cit. 2025-03-11].
- [29] ALTEXSOFT EDITORIAL TEAM. *The Good and the Bad of Vue.js Framework Programming*. Online. 28.9.2022. Dostupné z: <https://www.altexsoft.com/blog/pros-and-cons-of-vue-js/>. [cit. 2025-03-11].
- [30] GRZYBOWSKA, Kaja. *What is Vue.js used for? Top Applications in Modern App Development*. Online. Dostupné z: <https://www.monterail.com/blog/what-is-vue-js-used-for>. [cit. 2025-03-11].
- [31] STAUFFER, Matt. *Laravel: Up & Running: A Framework for Building Modern PHP Apps, 3rd Edition*. 3rd Edition. Beijing: O'Reilly Media, 2023. ISBN 9781098153267.
- [32] DYER, Russell J. T. *MYSQL in a nutshell*. Beijing: O'Reilly, 2005. ISBN 0596007892.

SEZNAM OBRÁZKŮ

Obrázek 1 – Tabulka users.....	50
Obrázek 2 – Tabulka roles	51
Obrázek 3 – Tabulka schedules	51
Obrázek 4 – Tabulka days	51
Obrázek 5 – Tabulka blocks	52
Obrázek 6 – Tabulka schedule_data	52
Obrázek 7 – Tabulka shared_schedules.....	53
Obrázek 8 – Diagram celé databáze	53
Obrázek 9 – Návrh přihlašovacího formuláře.....	54
Obrázek 10 – Finální podoba přihlašovacího formuláře	55
Obrázek 11 – Návrh registračního formuláře	55
Obrázek 12 – Finální podoba registračního formuláře	56
Obrázek 13 – Návrh kontaktního formuláře	57
Obrázek 14 – Finální podoba kontaktního formuláře.....	57
Obrázek 15 – Návrh uživatelského profilu	58
Obrázek 16 – Finální podoba uživatelského profilu	58
Obrázek 17 – Návrh administrace pro správu uživatelů.....	59
Obrázek 18 – Finální podoba administrace pro správu uživatelů.....	59
Obrázek 19 – Návrh administrace pro správu harmonogramů	60
Obrázek 20 – Finální podoba administrace pro správu harmonogramů	60
Obrázek 21 – Finální podoba administrace pro detailní přehled harmonogramu.....	61
Obrázek 22 – Návrh dashboardu	62
Obrázek 23 – Finální podoba dashboardu	62

SEZNAM TABULEK

Tabulka 1 Souhrn výhod a nevýhod Laravelu	23
Tabulka 2 Souhrn výhod a nevýhod Angularu	26
Tabulka 3 Souhrn výhod a nevýhod Reactu	29
Tabulka 4 Souhrn výhod a nevýhod Vue.js	32
Tabulka 5 Porovnání frameworků podle klíčových kritérií	33

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AI	Artificial Intelligence (Umělá inteligence)
API	Application Programming Interface (Programovací rozhraní aplikací)
CLI	Command Line Interface (Rozhraní příkazového řádku)
CMS	Content Management System (Systém pro správu obsahu)
CRUD	Create, Read, Update, Delete (Vytvořit, Číst, Aktualizovat, Smazat)
CSS	Cascading Style Sheets (Kaskádové styly)
CSP	Content Security Policy (Zásady zabezpečení obsahu)
CSRF	Cross-Site Request Forgery (Podvržení požadavku napříč stránkami)
DOM	Document Object Model (Objektový model dokumentu)
DDoS	Distributed Denial of Service (Distribuované odepření služby)
HTML	HyperText Markup Language (Značkovací jazyk pro tvorbu webu)
JSON	JavaScript Object Notation (Zápis objektů v jazyce JavaScript)
MVC	Model-View-Controller
ORM	Object-Relational Mapping (Objektově-relační mapování)
PWA	Progressive Web Application (Progresivní webová aplikace)
PDF	Portable Document Format
PHP	Hypertext Preprocessor
SEO	Search Engine Optimization (Optimalizace pro vyhledávače)
SPA	Single Page Application (Jednostránková aplikace)
SQL	Structured Query Language
UI	User Interface (Uživatelské rozhraní)
URL	Uniform Resource Locator
VR	Virtual Reality (Virtuální realita)
XSS	Cross-Site Scripting

SEZNAM PŘÍLOH

Příloha P I: Zdrojové kódy aplikace

PŘÍLOHA P I: ZDROJOVÉ KÓDY APLIKACE

- Zdrojové kódy aplikace nahrané v systému IS/STAG