

Open source plánovač směn pro malý podnik

Adam Novák

Bakalářská práce
2025



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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: Adam Novák
Osobní číslo: A22712
Studijní program: B0613A140020 Softwarové inženýrství
Forma studia: Prezenční
Téma práce: Open source plánovač směn pro malý podnik
Téma práce anglicky: Open Source Shift Scheduler for Small Business

Zásady pro vypracování

- Popište základní principy a problematiku vývoje klient-server webových aplikací.
- Vyberte a zdůvodněte volbu vhodného vývojového nástroje pro implementaci webové aplikace pro plánování směn, popište jeho strukturu, výhody a základní funkce.
- Stanovte funkcionální a nefunkcionální požadavky na aplikaci, navrhnete uživatelské rozhraní aplikace a databázovou strukturu.
- Implementujte aplikaci pro plánování směn dle stanovených požadavků.
- Shrňte dosažené výsledky a vhodnost zvoleného přístupu a zhodnotte zabezpečení webové aplikace.

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

Seznam doporučené literatury:

1. STAUFFER, Matt. Laravel: up & running: a framework for building modern PHP apps. Third edition. Beijing: O'Reilly, 2023. ISBN 978-1-098-15326-7.
2. KAVANAGH, Michael J. a JOHNSON, Richard David (ed.). Human resource information systems. Fourth edition. Los Angeles: Sage, [2018]. ISBN 978-1-5063-5145-2.
3. ZANDSTRA, Matt. PHP 8 objects, patterns, and practice: mastering OO enhancements, design patterns, and essential development tools. Sixth edition. New York: Apress, [2021]. ISBN 978-1-4842-6790-5.
4. LAURENČÍK, Marek. Tvorba www stránek v HTML a CSS. Průvodce. Praha: Grada Publishing, 2019. ISBN 978-80-271-2241-7.
5. ALLEN, Ashley. Battle Ready Laravel: A guide to auditing, testing, fixing, and improving your Laravel applications. 2022. ISBN 979-8371096036.
6. PHP Manual. Online. 2024. Dostupné z: <https://www.php.net/manual/en/index.php>. [cit. 2024-10-22].
7. Laravel documentation. Online. 2024. Dostupné z: <https://laravel.com/docs/11.x>. [cit. 2024-10-22].
8. Laracast. Online. 2024. Dostupné z: <https://laracasts.com/>. [cit. 2024-10-22].

Vedoucí bakalářské práce: **Ing. Radek Vala, 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(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.
- při tvorbě této práce jsem použil nástroj generativní model AI ChatGPT; <https://chatgpt.com/> za účelem zlepšení formulace textu. Po použití těchto nástrojů jsem provedl kontrolu obsahu a přebírám za něj plnou zodpovědnost.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato bakalářská práce se zaměřuje na návrh a realizaci open source webové aplikace pro plánování směn v malých podnicích. Cílem práce je navrhnout, implementovat a popsat systém, který usnadní správu zaměstnanců, směn a provozoven a přinese vyšší efektivitu a přehlednost do každodenního provozu. Práce se zabývá volbou vhodného technologického řešení, konkrétně PHP frameworku Laravel, návrhem databázové struktury, správou uživatelských rolí a oprávnění a implementací hlavních funkcionalit aplikace. Důraz je kladen na bezpečnost systému, uživatelskou přívětivost a možnost budoucího rozšíření, například o mobilní aplikaci nebo nové moduly. Na základě implementace je provedeno zhodnocení dosažených výsledků, přínosů a případných omezení navrženého řešení pro praxi malých firem.

Klíčová slova: open source, plánovač směn, správa směn, malý podnik, laravel, webová aplikace

ABSTRACT

This bachelor thesis focuses on the design and implementation of an open source web application for shift scheduling in small businesses. The aim of the thesis is to design, implement and describe a system that will facilitate the management of employees, shifts and premises and bring greater efficiency and clarity to day-to-day operations. The thesis deals with the selection of a suitable technological solution, namely the PHP framework Laravel, the design of the database structure, the management of user roles and permissions and the implementation of the main functionalities of the application. Emphasis is placed on the security of the system, user-friendliness and the possibility of future extension, for example, with a mobile application or new modules. Based on the implementation, an evaluation of the achieved results, benefits and possible limitations of the proposed solution for the practice of small companies is made.

Keywords: open source, shift planner, shift management, small business, laravel, web application

Rád bych touto cestou poděkoval všem, kteří mě během psaní této bakalářské práce podporovali a motivovali.

Na prvním místě děkuji vedoucímu mé bakalářské práce Ing. Radku Valovi, Ph.D., za jeho odborné vedení, cenné rady, které mi věnoval po celou dobu zpracování této práce.

Dále děkuji své rodině a blízkým za podporu, pochopení a povzbuzení během studia i při psaní této práce.

Velké poděkování patří také mým spolužákům, kteří mi byli oporou při studiu. Jejich ochota sdílet poznatky, konzultovat témata byla pro mě velmi cenná.

OBSAH

ÚVOD.....	11
I TEORETICKÁ ČÁST	12
1 ZÁKLADNÍ PRINCIPY A PROBLEMATIKA VÝVOJE KLIENT-SERVER WEBOVÝCH APLIKACÍ.....	13
1.1 ARCHITEKTURA KLIENT-SERVER	13
1.1.1 Jednovrstvá architektura.....	13
1.1.2 Dvouvrstvá architektura	14
1.1.3 Třívrstvá architektura	14
1.1.4 Srovnání architektur: Klient-Server vs. Peer-to-Peer.....	15
1.2 HTTP PROTOKOL	16
1.2.1 Request-response cyklus	17
1.2.2 HTTP metody	17
2 VÝVOJOVÉ NÁSTROJE PRO IMPLEMENTACI WEBOVÝCH APLIKACÍ.....	19
2.1 PŘEHLED DOSTUPNÝCH FRAMEWORKŮ PRO WEBOVÉ APLIKACE.....	19
2.1.1 ASP.NET CORE	19
2.1.2 Symfony	21
2.1.3 Express.js	23
2.2 KRITÉRIA VÝBĚRU VÝVOJOVÉHO FRAMEWORKU	24
2.2.1 Funkcionalita.....	25
2.2.2 Usnadnění vývoje a údržby	25
2.2.3 Opakované použití kódu	25
2.2.4 Konzistentní struktura a styl kódu.....	25
2.2.5 Vestavěné bezpečnostní mechanismy	25
2.2.6 Efektivita a úspora nákladů	26
2.2.7 Škálovatelnost	26
2.2.8 Komunitní podpora a dokumentace	26
2.2.9 Rizika a omezení frameworků	26
2.3 PŘEDSTAVENÍ FRAMEWORKU LARAVEL	26
2.3.1 Historie a charakteristika.....	27
2.3.2 Struktura Laravel frameworku	27
2.3.3 Architektura MVC v kontextu frameworku Laravel.....	28
2.3.4 Výhody využití Laravel frameworku	29
II PRAKTICKÁ ČÁST.....	31
3 ZDŮVODNĚNÍ VOLBY FRAMEWORKU LARAVEL	32
4 ANALÝZA A NÁVRH APLIKACE.....	34
4.1 SPECIFIKACE POŽADAVKŮ	34
4.1.1 Funkcionální požadavky	34
4.1.2 Nefunkcionální požadavky.....	35
4.2 NÁVRH UŽIVATELSKÉHO ROZHRANÍ	36

4.3	NÁVRH DATABÁZOVÉ STRUKTURY	39
4.3.1	Popis tabulek a relací.....	40
5	IMPLEMENTACE APLIKACE.....	49
5.1	AUTENTIZACE	49
5.1.1	Obnova hesla	50
5.2	AUTORIZACE – SPRÁVA ROLÍ A OPRÁVNĚNÍ	50
5.2.1	Základní princip	50
5.2.2	Rozšíření modelu Role:	51
5.2.3	Balíček Spatie.....	51
5.2.4	Ochrana přístupu pomocí middleware	54
5.3	MODELÝ.....	56
5.4	MIGRACE A SEEDERY.....	57
5.5	FRONTEND.....	59
5.5.1	Kalendář	59
6	ZHODNOCENÍ ZABEZPEČENÍ	61
6.1	OCHRANA PROTI WEBOVÝM ÚTOKŮM.....	61
6.1.1	CSRF – Cross-Site Request Forgery	61
6.1.2	XSS – Cross-Site Scripting	62
6.1.3	SQL Injection	62
6.2	BEZPEČNOST UŽIVATELSKÝCH ÚČTŮ A ROZHRANÍ API	62
6.2.1	Hashování hesel.....	62
6.2.2	JWT podpora a zabezpečení API	62
6.3	OCHRANA RELACÍ A PŘIHLÁŠENÍ	63
6.4	VALIDACE VSTUPNÍCH DAT	63
6.5	OMEZENÍ A NÁVRHY NA ROZŠÍŘENÍ ZABEZPEČENÍ	64
7	ZÁVĚR A ZHODNOCENÍ.....	65
7.1	SHRNUTÍ VÝSLEDKŮ	65
7.2	MOŽNÉ ROZŠÍŘENÍ APLIKACE	65
7.3	UKÁZKY UŽIVATELSKÉ ROZHRANÍ.....	67
	ZÁVĚR.....	70
	SEZNAM POUŽITÉ LITERATURY	71
	SEZNAM OBRÁZKŮ	73
	SEZNAM TABULEK.....	74
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	75
	SEZNAM PŘÍLOH.....	76

ÚVOD

Pro efektivní řízení provozu malých podniků, které zahrnují správu směn a plánování personálu, jsou klíčové dobře nastavené procesy a přehledná organizace práce. Ruční metody, jako jsou papírové plány, tabulky v Excelu nebo nástěnky, často narážejí na své limity – zejména pokud jde o aktuálnost, možnost sdílení dat mezi více lidmi a bezpečnost. Tyto nedostatky vedly k motivaci vytvořit moderní webovou aplikaci, která umožní jednoduché a efektivní plánování směn v prostředí více provozoven.

Impulzem k realizaci této aplikace byla snaha usnadnit práci administrátorům i zaměstnancům malých podniků, kde současné nástroje často postrádají potřebnou flexibilitu a přehlednost. Další motivací byl zájem o prohloubení znalostí v oblasti webového vývoje a získání praktických zkušeností s frameworkem Laravel.

Teoretická část této práce se zaměřuje na problematiku vývoje klient-server webových aplikací. Nejprve jsou popsány základní principy této architektury, její výhody a nevýhody. Dále je provedena analýza vybraných vývojových nástrojů a zdůvodněna volba frameworku Laravel. Je zde popsána jeho struktura, hlavní funkce a přínosy, které použití tohoto frameworku nabízí.

Praktická část se zabývá návrhem a implementací aplikace pro plánování směn. Jsou zde stanoveny funkcionální a nefunkcionální požadavky na aplikaci, navrženo uživatelské rozhraní a popsána databázová struktura systému. Součástí je také implementace hlavních funkcí aplikace, včetně správy uživatelů, směn a provozoven. Závěr práce hodnotí dosažené výsledky, provádí zhodnocení zvoleného přístupu a analyzuje úroveň zabezpečení aplikace.

Cílem této bakalářské práce je navrhnout, implementovat open source webovou aplikaci pro plánování směn, která poskytne bezpečné a rozšiřitelné prostředí pro správu pracovních rozpisů v malých podnicích.

I TEORETICKÁ ČÁST

1 ZÁKLADNÍ PRINCIPY A PROBLEMATIKA VÝVOJE KLIENT-SERVER WEBOVÝCH APLIKACÍ

Webové aplikace se staly nedílnou součástí moderního digitálního světa a jejich význam neustále roste. Tyto aplikace jsou zpravidla postaveny na architektuře klient-server, která představuje základní distribuovaný model pro vývoj webových řešení. Tento architektonický přístup rozděluje funkcionalitu aplikace mezi dvě hlavní komponenty – klientskou část běžící v prohlížeči koncového uživatele a serverovou část fungující na vzdáleném serveru. [1]

1.1 Architektura klient-server

Architektura klient-server představuje model počítačové sítě, kde mnoho klientů žádá o služby a získává je od centralizovaného serveru. Klientské počítače slouží jako rozhraní pro uživatele, kteří tímto způsobem požadují služby od serveru a vidí výsledky, které server poskytne. Servery vyčkávají na požadavky od klientů a poté na ně odpovídají. Ideálně poskytují standardizované a transparentní rozhraní. [1]

Klienti se obvykle nacházejí na pracovních stanicích nebo osobních počítačích, zatímco servery bývají umístěny na jiných místech sítě, často na výkonnějších strojích. Tento model fungování je velmi efektivní, pokud klienti a server mají specifické úkoly, které pravidelně plní. [1]

Díky tomu, že jak klientské, tak serverové počítače jsou inteligentní zařízení, je architektura klient-server zásadně odlišná od staršího modelu mainframe, kde veškeré výpočty prováděl centrální mainframe počítač, zatímco připojené terminály fungovaly pouze jako jednoduché zobrazovací jednotky bez vlastního výpočetního výkonu. [1]

1.1.1 Jednovrstvá architektura

V jednovrstvé architektuře jsou všechny součásti aplikace, včetně uživatelského rozhraní, obchodní logiky, databázové logiky a samotné databáze, integrovány do jednoho balíčku. Tento balíček funguje jako samostatná entita, která zajišťuje všechny funkce aplikace. [2]

Jednovrstvá architektura poskytuje spolehlivý výkon díky integrovaným službám, avšak její správa může být složitá kvůli rozptýlení dat a časté duplicitě práce. Obsahuje prezentační vrstvu, obchodní vrstvu a datovou vrstvu, které jsou propojeny v rámci jednoho softwarového balíčku. Data jsou obvykle ukládána lokálně nebo na sdíleném disku. [2]

1.1.2 Dvouvrstvá architektura

Dvouvrstvá architektura rozděluje aplikační logiku na dvě hlavní vrstvy. Databáze funguje jako samostatná jednotka, zatímco hlavní aplikace zahrnuje uživatelské rozhraní, obchodní logiku a databázovou logiku pro komunikaci s databází a zpracování aplikací. [2]

Tento model nabízí efektivnější prostředí než jednovrstvá architektura, protože nevyužívá prostředníka mezi klientem a serverem, což zrychluje komunikaci. Díky tomu se také snižuje riziko zmatků při interakci klientů se systémem. [2]

1.1.3 Třívrstvá architektura

Na rozdíl od dvouvrstvé architektury používá třívrstvá architektura middleware jako prostředníka mezi klientem a serverem. Middleware přijímá požadavky od klienta a následně je předává serveru ke zpracování. Server poté vrátí odpověď prostřednictvím middleware zpět klientovi.

Třívrstvá architektura se skládá ze tří klíčových vrstev:

1. Prezentační vrstva – Řízená klientskými zařízeními.
2. Aplikační vrstva – Spravovaná middlewarem a serverem, kde probíhá zpracování obchodní logiky.
3. Databázová vrstva – Ukládá a spravuje data.

Díky přítomnosti middleware poskytuje tato architektura vyšší úroveň bezpečnosti, neviditelné databázové struktury a lepší integritu dat. [2]

Tabulka 1 - Výhody a nevýhody třívrstvé architektury

Výhody	Nevýhody
Snadná správa – Centralizované ukládání souborů na serveru umožňuje efektivní správu a snadné vyhledávání.	Nižší robustnost – Pokud dojde k selhání hlavního serveru, celá síť přestane fungovat.
Přístupnost – Klienti mohou přistupovat k systému odkudkoli bez ohledu na platformu.	Náročná údržba – Servery vyžadují nepřetržitou údržbu a správa sítě musí být zajištěna odborníkem.
Škálovatelnost – Architektura umožňuje přidávání dalších klientů nebo serverů bez narušení provozu.	Vysoké náklady – Nastavení a údržba serverů je finančně náročná, což může omezit jejich dostupnost pro menší organizace.
Centralizované řízení – Informace jsou uloženy na jednom místě, což usnadňuje správu a aktualizace dat.	Přetížení sítě – Příliš mnoho požadavků na jeden server může způsobit jeho přetížení, což vede ke zpomalení připojení.
Zabezpečení – Centralizovaná správa přístupů a zálohování zvyšuje ochranu dat a umožňuje obnovu při jejich ztrátě.	

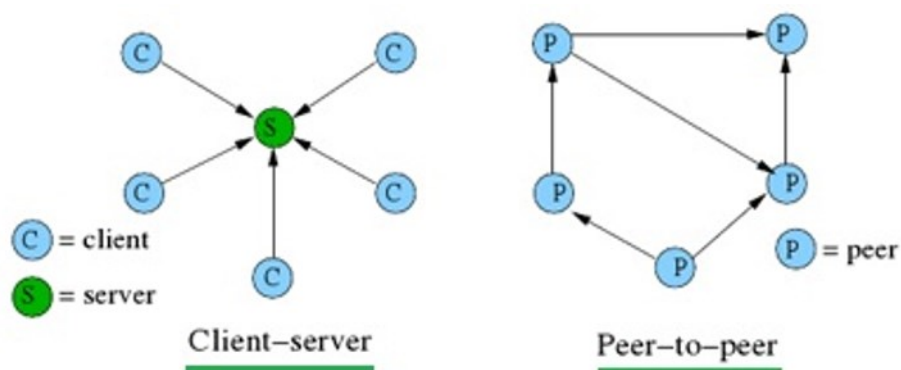
(zdroj: [2])

1.1.4 Srovnání architektur: Klient-Server vs. Peer-to-Peer

Klíčový rozdíl mezi architekturou klient-server a peer-to-peer spočívá v tom, že v klient-server modelu jsou jasně odděleny role klientů, kteří požadují služby, a serverů, které tyto služby poskytují. Naopak v peer-to-peer systémech jsou všechny uzly rovnocenné a mohou působit jak jako poskytovatelé služeb, tak jako jejich příjemci. [2][3]

Systémy klient-server vyžadují centrální souborový server a jejich zavedení je nákladnější ve srovnání s peer-to-peer systémy. Na druhou stranu klient-server model nabízí vyšší úroveň zabezpečení, protože přístupová práva spravuje dedikovaný souborový server. Naproti tomu v peer-to-peer sítích závisí bezpečnost na jednotlivých uživateli, což může vést k větším bezpečnostním rizikům. [2][3]

Peer-to-peer sítě navíc trpí poklesem výkonu, když roste počet uzlů, zatímco systémy klient-server jsou stabilnější a umožňují snadné škálování podle potřeb organizace. Výběr mezi těmito dvěma architekturami by měl být založen na konkrétních požadavcích a prostředí, kde mají být implementovány. [2]



Obrázek 1 - Grafické srovnání client-server vs Peer to peer

(zdroj:[4])

1.2 HTTP protokol

Hypertext Transfer Protocol (HTTP) představuje základní komunikační protokol, který umožňuje přenos dat mezi klientskými zařízeními (např. webovými prohlížeči) a servery. Jedná se o protokol aplikační vrstvy, který se historicky vyvíjel nad sadou protokolů TCP/IP – architektura, která tvoří páteř dnešního internetu. [5]

HTTP se používá vždy, když uživatel přistupuje k webovým stránkám. Prohlížeč v takovém případě odesílá požadavky (například na HTML dokument, obrázky, videa apod.) na cílový server. Tyto požadavky a odpovědi jsou standardizované a umožňují efektivní komunikaci mezi klientem a serverem. HTTP tvoří základ výměny informací na webu. Když uživatel zadá adresu webové stránky do prohlížeče nebo klikne na odkaz, prohlížeč odešle tzv. HTTP požadavek. Tento požadavek obsahuje například URL, metodu (např. GET, POST), verzi HTTP a případně další data (např. cookie nebo tělo požadavku). Server následně tento požadavek zpracuje pomocí tzv. HTTP démona – programu, který naslouchá požadavkům – a odešle zpět HTTP odpověď. Ta může obsahovat např. HTML dokument, stavový kód (např. 200 OK, 404 Not Found) nebo další metadata v hlavičkách. [5]

1.2.1 Request-response cyklus

Základním mechanismem komunikace mezi klientem a serverem ve webové aplikaci je tzv. request-response cyklus. V tomto procesu hraje zásadní roli HTTP protokol, který umožňuje přenos zpráv mezi klientským zařízením (např. webovým prohlížečem) a serverem, kde běží samotná webová aplikace. [6]

Celý cyklus začíná tím, že uživatel prostřednictvím svého prohlížeče odešle požadavek na konkrétní adresu URL. Tento požadavek je doručen webovému serveru, který ho předá aplikační vrstvě – v tomto případě například PHP frameworku jako je Laravel. Zde je požadavek nejprve analyzován tzv. routerem, který rozhoduje, jaký konkrétní kontroler a akce mají být spuštěny. Router zároveň extrahuje potřebné parametry z URL a předává je dál. [6]

Kontroler následně obslouží požadavek – často tím, že získá data z databáze nebo jiných zdrojů, zpracuje je pomocí aplikační logiky a připraví odpověď. Tato odpověď je obvykle zabalena do objektu typu Response, který obsahuje nejen samotný obsah (např. HTML, JSON), ale i metadata jako HTTP stavový kód nebo hlavičky. [6]

Tento response je poté odeslán zpět klientovi. Prohlížeč následně zpracuje přijatou odpověď a podle jejího typu ji zobrazí uživateli, případně provede další akce (například přesměrování nebo zobrazení chybového hlášení). [6]

Tento cyklus se opakuje při každé interakci uživatele s webovou aplikací, ať už jde o kliknutí na odkaz, odeslání formuláře nebo načtení nové stránky. Přesně definovaná struktura cyklu a standardizace protokolu HTTP umožňují robustní, předvídatelný a škálovatelný způsob komunikace mezi klientem a serverem. [6]

1.2.2 HTTP metody

HTTP protokol definuje sadu metod, které určují typ operace, které klient požaduje po serveru. Tyto metody, často označované jako „HTTP verb“ (slovesa), jsou základem pro komunikaci mezi klientem a serverem a hrají klíčovou roli při vývoji webových aplikací. [7]

Mezi nejběžnější HTTP metody patří [7]:

- **GET:** Slouží k získání reprezentace specifikovaného zdroje. Požadavky pomocí metody GET by měly pouze načítat data a neměly by mít žádné vedlejší účinky.

- **POST:** Používá se k odeslání entity na specifikovaný zdroj, často za účelem vytvoření nového záznamu nebo způsobení vedlejších efektů na serveru.
- **PUT:** Nahrazuje všechny aktuální reprezentace cílového zdroje obsahem požadavku. Pokud zdroj neexistuje, může být vytvořen.
- **DELETE:** Odstraňuje specifikovaný zdroj.
- **PATCH:** Aplikuje částečné úpravy na zdroj. Na rozdíl od PUT, který nahrazuje celý zdroj, PATCH modifikuje pouze specifikované části.

Tyto metody jsou úzce spojeny s operacemi CRUD (Create, Read, Update, Delete), které představují základní operace pro práci s daty v databázích [7]:

- **Create:** POST
- **Read:** GET
- **Update:** PUT nebo PATCH
- **Delete:** DELETE

Při návrhu RESTful API je důležité správně mapovat tyto metody na příslušné operace, aby byla zajištěna konzistence a předvídatelnost chování aplikace. Dále je třeba brát v úvahu vlastnosti jednotlivých metod [7][8]:

- **Bezpečnost (Safe):** Metody, které nemění stav serveru (např. GET, HEAD).
- **Idempotent:** Opakované provedení požadavku má stejný efekt jako jedno provedení (např. PUT, DELETE).
- **Možnost cachování:** Některé odpovědi na požadavky mohou být uloženy do cache pro budoucí použití (např. GET, HEAD).

2 VÝVOJOVÉ NÁSTROJE PRO IMPLEMENTACI WEBOVÝCH APLIKACÍ

V této kapitole budou představeny moderní nástroje a frameworky, které se používají při vývoji webových aplikací. Zaměření bude kladeno především na jejich praktické využití, architektonické vlastnosti a technické předpoklady pro implementaci různých typů projektů. Jednotlivé frameworky budou popsány z hlediska jejich vývojových možností, podpory multiplatformního nasazení, výkonnosti, bezpečnostních prvků a komunitní podpory. Součástí kapitoly je také přehled kritérií, které jsou vhodné zohlednit při výběru vývojového prostředí, a detailní analýza frameworku Laravel, který byl vybrán jako základ pro realizaci projektu.

2.1 Přehled dostupných frameworků pro webové aplikace

Podkapitola se zaměřuje na srovnání vybraných webových frameworků, které jsou v současnosti široce využívány při vývoji moderních webových aplikací. Cílem je poskytnout přehled jejich základních vlastností, výhod. Každý framework bude stručně charakterizován z hlediska své architektury, podpory nástrojů, flexibility a technického zázemí. Představeny budou technologie ASP.NET Core, Symfony, Express.js.

2.1.1 ASP.NET CORE

ASP.NET Core je moderní, open-source framework vyvinutý společností Microsoft pro vývoj webových aplikací, rozhraní API a mikroservis. Primárně cílí na potřeby současného vývoje, jako je práce s kontejnery, cloudové nasazení a multiplatformní vývoj. Od svého uvedení v roce 2016 prošel výrazným vývojem a stal se důležitou součástí sjednoceného .NET ekosystému. [9][10]

ASP.NET Core se vyznačuje multiplatformností, což znamená, že je možné jej provozovat nejen na systému Windows, ale také na Linuxu a macOS. Tím se otevírají možnosti pro týmy využívající různé operační prostředí. Díky vysokému výkonu a škálovatelnosti se ASP.NET Core řadí mezi nejrychlejší webové frameworky dostupné na trhu, což je podpořeno optimalizací zpracování požadavků i efektivní prací s databázemi. [9]

Framework využívá sjednocený vývojový model. To znamená, že bez ohledu na to, zda se jedná o webovou aplikaci, API nebo desktopový software, vývojář pracuje s jednotnou technologií. Tento přístup usnadňuje sdílení kódu a orientaci v projektech. Významnou

vlastností je také vestavěná podpora *dependency injection*, což přispívá k lepší udržitelnosti a testovatelnosti aplikací. [9]

Součástí ASP.NET Core je také technologie Blazor, která umožňuje tvorbu interaktivních webových uživatelských rozhraní za použití jazyku C# místo JavaScriptu. To je velkou výhodou zejména pro vývojáře, kteří již pracují v .NET prostředí. [9]

Mezi výhody patří [11]:

- **Multiplatformní podpora:** ASP.NET Core umožňuje vývoj a nasazení aplikací nejen na Windows, ale také na Linux a macOS, což zvyšuje flexibilitu při výběru hostitelského prostředí.
- **Výkonnost a škálovatelnost:** ASP.NET Core je optimalizován pro vysoký výkon díky funkcím, jako je just-in-time (JIT) kompilace a server-side caching, a je vhodný pro rozsáhlé aplikace s vysokou návštěvností.
- **Silná bezpečnost:** ASP.NET Core nabízí vestavěné mechanismy ochrany proti běžným zranitelnostem (např. XSS, CSRF) a těží z bezpečnostních standardů a pravidelných aktualizací od Microsoftu.
- **Bohatá integrace s Microsoft technologiemi:** Díky integraci s nástroji jako Windows Server, SQL Server a Active Directory lze snadno stavět aplikace, které těží z celého Microsoft ekosystému.
- **Moderní vývojové prostředí:** ASP.NET Core je podporován silným IDE, například Visual Studio, které nabízí funkce jako IntelliSense, debugging a správa projektů pro efektivnější vývoj.
- **Open-source a komunitní podpora:** I když je to Microsoft technologie, ASP.NET Core je open-source, což podporuje komunitní spolupráci a umožňuje rychle reagovat na zpětnou vazbu a nové požadavky.

Mezi nevýhody patří [11]:

- **Strmější učební křivka:** Pro začátečníky může být ASP.NET Core složitější na naučení, protože vyžaduje znalosti .NET frameworku a moderního objektově orientovaného programování.

- Potenciální závislost na Windows: Přestože je ASP.NET Core multiplatformní, některé pokročilé funkce a podnikové nasazení mohou být stále více optimalizovány pro prostředí Windows.
- Náklady na vývoj: Vývoj s ASP.NET Core může být nákladnější, protože často vyžaduje zkušené vývojáře a integraci s placenými Microsoft produkty a službami.
- Vendor lock-in: Používání ASP.NET Core v rámci Microsoft ekosystému může vést k závislosti na této platformě, což může ztížit případnou migraci na jiné technologie.

2.1.2 Symfony

Symfony je moderní, open-source PHP framework navržený pro vývoj škálovatelných, modulárních a efektivních webových aplikací. Od svého uvedení v roce 2005 společností SensioLabs si získal silné postavení mezi vývojáři, a to zejména díky své architektuře založené na vzoru MVC (Model-View-Controller) a vysoké flexibilitě. Symfony je široce využíván v komerční sféře – mezi známé uživatele patří například Spotify, National Geographic Travel či Swiss Open Government Data. [12]

Symfony staví na principu modularity a opakovaného použití komponent. Díky svému návrhu MVC umožňuje efektivní rozdělení aplikace do tří vrstev. Jádrem Symfony je soubor samostatných komponent – malých knihoven, které lze použít samostatně nebo v rámci frameworku. Tato modularita umožňuje vývojářům zvolit pouze ty části, které skutečně potřebují, což vede ke vzniku optimalizovaného kódu bez nadbytečné zátěže. [12]

Mezi výhody patří [13]:

- Úspora času při vývoji: Symfony výrazně zrychluje vývoj aplikací. To, co by bez frameworku zabralo tři dny, lze se Symfony zvládnout za jeden. To šetří čas, který lze využít pro jiné části projektu.
- Snadná údržba: Díky přehledné architektuře a dobře strukturovanému kódu je údržba aplikací jednodušší. Noví vývojáři se v projektu snáze zorientují a sníží se množství chyb díky osvědčenému kódu.
- Zvýšená bezpečnost: Symfony pravidelně vydává aktualizace zaměřené na bezpečnost, což z něj činí spolehlivý nástroj pro vývoj citlivých webových systémů.

- Vysoká flexibilita: Symfony využívá tzv. *bundles* – znovupoužitelné balíčky obsahující šablony, JavaScript, PHP soubory atd. Díky jejich oddělení a možnosti nezávislé konfigurace lze framework snadno přizpůsobit konkrétním potřebám projektu.
- Nízká závislost mezi komponentami: Kombinováním komponent a *bundles* vzniká flexibilní architektura s minimem vzájemných závislostí. To usnadňuje změny a přizpůsobení bez rizika rozbití jiných částí systému.
- Obchodní podpora: Symfony je vyvíjen společností SensioLabs, která nabízí profesionální služby, certifikace a školení. Díky tomu je framework oblíbený u firemních klientů.
- Spolehlivost: Symfony je dlouhodobě stabilní a používají jej i jiné významné platformy (např. Magento, Drupal). Díky své osvědčenosti je považován za důvěryhodný nástroj.
- Efektivní testování: Díky dobře navržené architektuře a možnosti znovupoužití komponent je testování kódu ve Symfony jednodušší a efektivnější, což podporuje stabilní běh aplikace.

Mezi nevýhody patří [13][14]:

- Nižší výkon při vysoké zátěži: Při provozu webových aplikací s velkým množstvím současných uživatelů může Symfony vykazovat nižší výkon, zejména na starších verzích PHP.
- Delší doba vývoje: Kvůli používání návrhových vzorů a vyšší komplexitě architektury může vývoj trvat déle, obzvláště v počátečních fázích projektu.
- Náročné na učení: Symfony má strmou křivku učení. Vyžaduje hlubší technické znalosti a časovou investici k plnému zvládnutí frameworku, což může být náročné pro začátečníky.
- Pomalejší tempo vývoje ve srovnání s Laravelem: Symfony sice nabízí široké možnosti přizpůsobení, ale právě tato flexibilita často zpomaluje práci. Vývojáři musí řadu funkcí nastavovat ručně, což celkově prodlužuje dobu, než je aplikace hotová.

- Náročnější proces konfigurace a spuštění: Projekty v Symfony obvykle vyžadují složité nastavení a důkladné pochopení celé architektury. Ve srovnání s frameworky, které nabízejí jednodušší a více automatizované kroky instalace, si Symfony žádá větší časové a znalostní úsilí.
- Větší objem rutinního kódu: Při práci se Symfony je často nutné psát více opakujícího se kódu, například tvořit dodatečné třídy či konfigurace. I když to dává vývojářům detailní kontrolu nad aplikací, může to negativně ovlivnit jejich efektivitu a znamenat více rutinní práce.

2.1.3 Express.js

Express.js je minimalistický, flexibilní a open-source framework pro Node.js, určený pro vývoj serverových aplikací a API. Jeho hlavní výhodou je jednoduchost a rychlost, s jakou lze vytvořit plně funkční backend pomocí jazyka JavaScript. Díky široké podpoře middleware, robustnímu systému směrování a integraci s šablonovacími nástroji si získal oblibu mezi vývojáři tvořícími RESTful API i celé webové aplikace. [15]

Express funguje jako tenká vrstva nad Node.js, která usnadňuje zpracování požadavků, směrování a správu odpovědí. Umožňuje vývojářům rychle navrhnout a strukturovat aplikace přesně podle jejich potřeb, bez předepsané architektury. Framework je vhodný jak pro prototypování, tak i pro plně škálovatelné produkční aplikace. [15]

Díky tomu, že Express běží na Node.js, umožňuje psát jak klientskou, tak serverovou logiku v jednom jazyce – JavaScriptu, což snižuje kontextové přepínání a zjednodušuje spolupráci mezi frontend a backend vývojáři. [15]

Mezi výhody patří [16][17]:

- Úspora času při vývoji: Express.js umožňuje rychlejší vývoj webových aplikací díky sdílení kódu mezi klientem a serverem, což zvyšuje efektivitu a zkracuje dobu potřebnou k nasazení.
- Vývoj aplikací v reálném čase: Framework je ideální pro tvorbu aplikací, které vyžadují práci v reálném čase, například chatovací systémy, streamovací služby nebo jednostránkové aplikace s dynamickým obsahem.
- Snadná implementace změn: Díky událostmi řízené architektuře a decentralizovanému přístupu je možné rychle přizpůsobit aplikaci měnícím se požadavkům trhu a provádět úpravy bez nutnosti zásadního přepracování.

- Škálovatelnost: Express.js poskytuje flexibilní základ pro růst a rozšiřování aplikace. S minimem závislostí a jednoduchou strukturou usnadňuje škálování podle potřeb firmy.
- Snadné učení: Jelikož je postaven na JavaScriptu, je pro vývojáře s touto znalostí snadné osvojit si základy Express.js a rychle začít tvořit backendové aplikace.
- Rychlý výkon: Express.js využívá rychlého běhového prostředí Node.js, které umožňuje efektivní zpracování souběžných požadavků, což je výhodné především u rozsáhlých webových platforem.
- Flexibilita a modularita: Framework je navržen tak, aby byl snadno přizpůsobitelný – vývojáři mohou využívat různé moduly, knihovny a pracovní postupy dle potřeby, aniž by byli svázáni přísnými pravidly.

Mezi nevýhody Express.js patří [16][17]:

- Malý a nezralý ekosystém: Některé části ekosystému Express.js mohou být méně vyvinuté nebo obsahovat složitý kód, což může vést k problémům s udržitelností a stabilitou.
- Nevhodnost pro CPU-náročné úkoly: Express.js není ideální volbou pro aplikace, které vyžadují intenzivní zpracování dat, například úpravu zvuku, videa nebo grafiky, protože je optimalizovaný spíše pro reaktivní aplikace.
- Složitější nasazení: Na rozdíl od technologií jako PHP vyžaduje Express.js více konfigurace a integrací při nasazování do produkčního prostředí, což může znamenat vyšší nároky na zkušenosti vývojářů.
- Menší komunita a podpora: I když popularita Express.js roste, stále nemá tak velkou komunitu a podporu jako některé zavedenější backendové technologie, což může zpomalit řešení problémů a hledání zdrojů.

2.2 Kritéria výběru vývojového frameworku

Při vývoji moderních webových aplikací představuje volba vhodného vývojového frameworku jedno z klíčových rozhodnutí. Frameworky nabízejí strukturované, a přitom flexibilní prostředí, které výrazně zjednodušuje celý proces vývoje – od návrhu až po údržbu. Díky předem definovaným nástrojům, knihovnám a osvědčeným postupům umožňují

soustředit se více na samotnou funkcionalitu aplikace. V této kapitole jsou popsána hlavní kritéria, která je vhodné při výběru frameworku zvážit. [18]

2.2.1 Funkcionalita

Při výběru frameworku je důležité řídit se skutečnými požadavky projektu. Pokud aplikace vyžaduje pouze základní směrování požadavků (*routing*), nemá smysl sahát po robustním full-stack řešení, které nabízí i funkce, jež zůstanou nevyužité. Výběr by měl vycházet z konkrétní potřeby, nikoliv pouze ze zvyklostí nebo oblíbenosti daného nástroje. Jasně definování požadavků umožňuje objektivně porovnat frameworky podle toho, zda poskytují přesně ty funkce, které daný projekt vyžaduje. [18][19]

2.2.2 Usnadnění vývoje a údržby

Frameworky obvykle nabízejí pevně danou strukturu projektu a množství předdefinovaných nástrojů jako jsou migrace, validace, směrování. To umožňuje vývojářům soustředit se na samotnou logiku aplikace, nikoliv na opakující se rutinní činnosti či kód. Díky standardizaci architektury a organizaci kódu je rovněž usnadněna následná údržba aplikace. [18]

2.2.3 Opakované použití kódu

Jednou z hlavních výhod frameworků je možnost opakovaně používat již vytvořené komponenty, funkce či knihovny. Tento přístup zkracuje dobu vývoje, zamezuje duplicitám v kódu a zvyšuje efektivitu celého vývojového procesu. [18]

2.2.4 Konzistentní struktura a styl kódu

Frameworky často vyžadují dodržování jednotného stylu psaní kódu a předem stanovených konvencí. Tento přístup usnadňuje orientaci v projektu, zjednodušuje týmovou spolupráci a umožňuje rychlejší zapojení dalších či nových vývojářů. Některé frameworky navíc podporují více jazyků. [18]

2.2.5 Vestavěné bezpečnostní mechanismy

Moderní frameworky integrují ochranu proti běžným zranitelnostem, jako jsou SQL Injection, Cross-Site Scripting (XSS) nebo Cross-Site Request Forgery (CSRF). Díky těmto prvkům lze snadněji vytvořit bezpečné aplikace bez nutnosti implementace všech zabezpečení manuálně. [18]

2.2.6 Efektivita a úspora nákladů

Přestože počáteční seznámení s frameworkem může být časově náročné, jejich použití vede ke značné úspoře nákladů v dlouhodobém horizontu. Opětovné použití knihoven, generování kódu a automatizace úloh umožňuje vývojářům soustředit se na jádro projektu a dokončit aplikaci rychleji. [18]

2.2.7 Škálovatelnost

Frameworky jsou zpravidla koncipovány tak, aby podporovaly růst projektu. Ať už jde o přidávání nových funkcí, větší počet uživatelů nebo vyšší zátěž, správně zvolený framework umožňuje projekt bez nutnosti zásadních úprav rozšiřovat. [18]

2.2.8 Komunitní podpora a dokumentace

Frameworky s rozsáhlou a aktivní komunitou vývojářů přinášejí mnoho výhod. Snadno dostupná dokumentace, návody, diskusní fóra a příklady z praxe pomáhají při řešení problémů či při osvojení si frameworku. Velikost komunity často ukazuje na jeho spolehlivost a dlouhodobou životaschopnost. [18][19]

2.2.9 Rizika a omezení frameworků

Přes řadu výhod je nutné počítat také s určitými nevýhodami. Mezi nejčastější patří:

- Strmá učící křivka – komplexní frameworky mohou být zpočátku náročné na pochopení.
- Závislost – změna frameworku po implementaci může být obtížná a nákladná.
- Omezené možnosti přizpůsobení – některé frameworky omezují vývojáře pevnou strukturou a konvencemi.
- Závislost na knihovnách třetích stran – frameworky často využívají externí balíčky, které mohou představovat riziko z hlediska kompatibility nebo dostupnosti. [18][19]

2.3 Představení frameworku Laravel

Laravel představuje webový aplikační framework charakteristický svou expresivní a elegantní syntaxí. Tento framework poskytuje strukturovaný základ a výchozí architekturu pro vývoj webových aplikací, což vývojářům umožňuje koncentrovat se na implementaci

klíčových funkcionalit, zatímco framework zajišťuje správu nižších vrstev a techničtějších aspektů vývoje. [20]

Framework Laravel si klade za cíl poskytovat optimální vývojářské prostředí při současném zajištění pokročilých funkcí včetně propracované *dependency injection*, abstraktní vrstvy pro databázové operace, systému front a plánovaných úloh, komplexního testovacího prostředí zahrnujícího unit a integrační testy a dalších pokročilých nástrojů. [20]

Laravel je navržen tak, aby vyhovoval různým úrovním zkušeností – je vhodný jak pro začínající vývojáře, tak pro zkušené programátory, jelikož umožňuje postupné rozšiřování znalostí a schopností v kontextu PHP webových technologií. Framework poskytuje progresivní vývojovou platformu umožňující realizaci široké škály webových projektů. [20]

2.3.1 Historie a charakteristika

Laravel je moderní PHP framework, který vytvořil Taylor Otwell s cílem zjednodušit vývoj webových aplikací, nabídnout elegantní syntaxi a zvýšit produktivitu vývojářů. Inspiraci čerpal z frameworku CodeIgniter, který byl oblíbený pro svou jednoduchost a lehkost. I když CodeIgniter nabízel dobrý základ, postrádal vestavěnou podporu pro autentizaci a autorizaci uživatelů, což Taylor vnímal jako zásadní nedostatek. Laravel proto od začátku integroval robustní systém autentizace, správu oprávnění a další prvky potřebné pro moderní aplikace, čímž vývojářům ušetřil čas a námahu spojenou s implementací těchto funkcí od nuly. [21]

Laravel byl navíc strategicky postaven na komponentech frameworku Symfony, čímž získal pevný základ a přístup k osvědčené funkcionalitě, aniž by ztratil svou flexibilitu. Díky tomu se Laravel postupně rozvinul z malého projektu v jeden z nejpoužívanějších PHP frameworků na světě. S každým dalším vydáním přináší nové nástroje a vylepšení, která reflektují aktuální potřeby vývojářů. Jeho přehledná dokumentace, silná komunita a důraz na čistý a udržitelný kód z něj dělají ideální volbu pro vývoj robustních webových aplikací a API. [21]

2.3.2 Struktura Laravel frameworku

Struktura Laravel frameworku je navržena tak, aby poskytovala základ pro vývoj malých i rozsáhlých aplikací. Přesto Laravel ponechává vývojářům svobodu v organizaci kódu, pokud je zajištěno jeho správné načítání přes Composer. Kořenová složka projektu obsahuje několik klíčových adresářů, jako je *app* pro aplikační logiku, *bootstrap* pro inicializaci frameworku, *config* pro konfigurační soubory, *database* pro migrace a seedery, *public* jako

vstupní bod aplikace, resources pro blade šablony a zdroje, routes pro definice tras a storage pro ukládání logů, cache či nahraných souborů. Složka tests obsahuje automatizované testy a vendor ukládá závislosti nainstalované pomocí Composeru. [22]

Uvnitř adresáře app se nachází většina obchodní logiky aplikace. Výchozí struktura obsahuje složky jako Http (kontrolery, middleware), Models (datové modely), Providers (poskytovatelé služeb) a podle potřeby přibývají další (např. Jobs, Events, Mail, Notifications, Policies, Rules). Laravel umožňuje generování tříd přes Artisan příkazy, což podporuje udržitelnost a konzistentní strukturu kódu. Díky modulárnímu uspořádání a možnosti přizpůsobení představuje struktura Laravelu dobře organizovaný základ, který zároveň zůstává flexibilní a rozšiřitelný. [22][23]

2.3.3 Architektura MVC v kontextu frameworku Laravel

Architektura MVC patří mezi nejpoužívanější návrhové vzory ve vývoji webových stránek. Laravel je oblíbený PHP framework, který tuto architekturu implementuje velice důsledně. Znalost MVC je zásadní pro tvorbu webových aplikací, které jsou snadno rozšiřitelné a udržitelné. [24]

Model v aplikaci reprezentuje data a veškerou logiku s daty spojenou. V Laravelu zajišťuje Model komunikaci s databází a operace nad daty. Stojí mezi aplikací a databází a nabízí objektově orientované rozhraní pro práci s informacemi. Laravel využívá Eloquent ORM (Object-Relational Mapping), který výrazně usnadňuje práci s databází v rámci modelů. Pomocí Eloquentu je možné snadno provádět operace jako vytváření, čtení, aktualizace a mazání (CRUD) databázových záznamů. Navíc umožňuje definovat vztahy mezi jednotlivými modely, což podporuje správu složitějších datových struktur. [24][25] [23]

View se stará o zobrazování dat uživateli a zpracování jeho vstupů. V Laravelu se pohledy vytváří s pomocí šablonovacího systému Blade, který nabízí přehlednou syntaxi pro tvorbu dynamických a opakovaně použitelných komponent uživatelského rozhraní. Pohledy jsou samostatné soubory s HTML a PHP kódem. Data získaná od Controlleru lze jednoduše zobrazit pomocí Blade syntaxe. Šablony podporují podmínky, cykly i vkládání částečných šablon, což přispívá k modularitě a flexibilitě rozhraní. [24; 25]

Controller funguje jako spojovací článek mezi Modelem a View. Přijímá požadavky od uživatele, zpracovává je, volá potřebné operace na Modelu a předává výsledky do View.

V Laravelu Controller definuje jednotlivé metody (akce) odpovídající konkrétním požadavkům uživatele. Obsahují logiku nutnou pro obsluhu požadavků, správu datového toku, validaci vstupů a provádění autorizace před manipulací s daty. [23][24]

2.3.4 Výhody využití Laravel frameworku

V této části jsou popsány hlavní výhody využití Laravel frameworku, které přispívají k jeho oblíbenosti mezi vývojáři a k efektivitě při vývoji aplikací.

Zaměření na vývojáře a radost z programování

Jedním z hlavních principů Laravelu je snaha zvýšit spokojenost vývojářů a urychlit tak vývojový proces. Otwell klade důraz na to, aby byl framework nejen funkční, ale také příjemný pro práci – například i tím, že podporuje psaní čitelného kódu. Výsledkem je prostředí, které eliminuje zbytečné bariéry mezi myšlenkou a realizací. Laravel je koncipován jako framework, který maximálně usnadňuje a zrychluje celý vývojový proces – od vytvoření prvního návrhu až po nasazení aplikace do produkčního prostředí. Oproti jiným frameworkům, které se často zaměřují na přísnou architektonickou čistotu nebo na potřeby velkých podnikových systémů, Laravel klade důraz především na praktické potřeby jednotlivých vývojářů. [25]

Jednoduchost a srozumitelnost

Laravel preferuje konvenci před konfigurací, což znamená, že pokud vývojář přijme výchozí nastavení frameworku, může vyvíjet aplikace bez nutnosti složitých konfigurací. Tato vlastnost nejen snižuje čas potřebný k vytvoření projektu, ale také snižuje možnost vzniku chyb a zjednodušuje údržbu. [25]

Součástí filozofie frameworku je i snaha o co nejjednodušší návrh – Laravel nevnucuje složité architektonické vzory, pokud nejsou nezbytné. Vývojář si může zvolit jednoduché přístupy, jako jsou globální funkce nebo *facade* vzory, a přitom framework neztrácí schopnost pokrýt i složitější scénáře. [25]

Ekosystém nástrojů a rozšiřitelnost

Jedním z nejvýraznějších rysů Laravelu je jeho bohatý ekosystém nástrojů, které pokrývají široké spektrum potřeb vývoje:

- Homestead a Valet pro lokální vývojové prostředí,
- Forge pro správu serverů,

- Envoyer pro bezvýpadkové nasazování,

specializované balíčky jako Cashier, Scout, Passport, Horizon, Nova nebo Dusk, které řeší specifické oblasti vývoje (např. platby, vyhledávání, API autentizaci, správu front, admin rozhraní, testování apod.). Tato výbava umožňuje vývojářům soustředit se na podstatu aplikace a přenechat rutinní úkoly frameworku. [25]

Aktivní a vstřícná komunita

Laravel těží z jedné z nejaktivnějších a nejprátelejších komunit ve světě PHP. Od video lekcí na Laracasts, přes blogy a podcasty, až po oficiální konference Laracon, je komunita Laravelu orientovaná na vzdělávání a vzájemnou pomoc. To výrazně snižuje bariéru vstupu pro nové uživatele a urychluje řešení problémů v praxi. [25]

II PRAKTICKÁ ČÁST

3 ZDŮVODNĚNÍ VOLBY FRAMEWORKU LARAVEL

Volba vývojového nástroje výrazně ovlivňuje efektivitu vývoje, údržbu i výslednou kvalitu aplikace. Na základě kritérií uvedených v kapitole 2.2 a praktických požadavků projektu byl zvolen PHP framework Laravel, který nejlépe splňuje technické, funkční i provozní potřeby systému. Následující přehled dokládá, jak Laravel naplnil jednotlivá kritéria:

Funkcionalita

Laravel nabízí kompletní sadu nástrojů pro vývoj moderní webové aplikace: routing, middleware, autentizaci, validaci, správu databáze pomocí migrací a seedrů a REST API. Všechny tyto nástroje byly v rámci projektu využity.

Usnadnění vývoje a údržby

Laravel definuje jednotnou strukturu projektu a přehledně odděluje jednotlivé vrstvy aplikace (modely, kontrolery, pohledy, routy). Díky příkazům, které nabízí *artisan* bylo možné generovat komponenty projektu automatizovaně, což zrychlilo vývoj a snížilo množství chyb.

Opakované použití kódu

Framework obsahuje předpřipravené moduly (např. validace vstupů, autentizace, správa rolí a oprávnění pomocí balíčku Spatie), které bylo možné snadno integrovat bez nutnosti psát vše od začátku. Projekt rovněž využívá factories a seedery pro generování testovacích dat. Tím bylo možné rychle připravit data nejen pro testování, ale i pro případné další vývojáře, kteří se k projektu připojí.

Konzistentní struktura a styl kódu

Laravel podporuje jednotný vývojový styl. Díky MVC architektuře a konvencím frameworku je projekt přehledný a snadno rozšiřitelný. Tato konzistence umožňuje i jinému vývojáři rychle se zorientovat v kódu.

Vestavěné bezpečnostní mechanismy

Framework má nativní ochranu proti CSRF, XSS, SQL Injection a nabízí hashování hesel pomocí bcrypt. Bezpečnostní vrstvy byly klíčové vzhledem k povaze aplikace, která pracuje s osobními údaji a plánováním směn.

Efektivita a úspora nákladů

Laravel jako open-source nástroj eliminuje licenční náklady. Díky bohaté sadě funkcí a výborné dokumentaci umožňuje rychlý vývoj a snižuje časovou náročnost údržby.

Škálovatelnost

Projekt je navržen s ohledem na více provozoven, uživatelské role i možnost budoucího rozšíření o mobilní aplikaci. Laravel poskytuje modularitu, která škálování systému neomezuje.

Jednoduchost nasazení

Na rozdíl od některých JavaScriptových frameworků (např. React nebo Vue.js ve spojení s Node.js), kde je často třeba nastavovat Webpack, bundlery, proxy servery a buildovací procesy, je nasazení Laravel aplikace výrazně jednodušší. Stačí PHP server s Apache nebo Nginx, databáze a prostředí s Composerem a NPM. Laravel běží stabilně i na běžném sdíleném hostingu, což je výhodné pro malé firmy bez vlastního serverového zázemí.

Komunitní podpora, dokumentace, aktuálnost

Laravel má mimořádně aktivní komunitu, bohatou oficiální dokumentaci a široké spektrum externích balíčků. Pravidelně vychází nové verze a dlouhodobě podporované verze (LTS) poskytují záruku bezpečnostních aktualizací po několik let. Tento aktivní vývoj zajišťuje, že Laravel zůstává aktuální s nejnovějšími technologickými standardy.

Rizika a omezení

Laravel má mírně vyšší nároky na pochopení architektury a některých pokročilých konceptů (např. dependency injection, service providery), zejména pro začínající vývojáře. Díky množství dostupných zdrojů, návodů a komunitní podpory však byla tato počáteční učicí křivka snadno zvládnutelná. Riziko uzamčení na jednu je zde minimální vzhledem k popularitě frameworku.

4 ANALÝZA A NÁVRH APLIKACE

Tato kapitola se zaměřuje na praktickou část vývoje aplikace. Cílem je definovat požadavky, které systém musí splňovat, a následně představit návrh jeho klíčových částí, jako je uživatelské rozhraní a databázová struktura. Nejprve jsou popsány funkcionální a nefunkcionální požadavky, které vycházejí z předpokládaného způsobu používání systému a jeho cílové skupiny. Poté následuje návrh samotné aplikace, který slouží jako základ pro její implementaci.

4.1 Specifikace požadavků

Specifikace požadavků představuje klíčovou fázi vývoje softwaru, ve které jsou definovány cíle a funkce systému. Správně nadefinované požadavky slouží jako východisko pro návrh, vývoj i budoucí údržbu systému. V této podkapitole jsou požadavky rozděleny do dvou skupin – funkcionální a nefunkcionální.

4.1.1 Funkcionální požadavky

Funkcionální požadavky popisují konkrétní schopnosti systému – tedy to, co uživatelé mohou v aplikaci vykonávat. V systému není funkcionalita pevně vázána na konkrétní uživatelské role (např. administrátor vs. zaměstnanec). Místo toho je přístup ke konkrétním funkcím řízen pomocí systému oprávnění, které lze spravovat dynamicky přímo v aplikaci.

Následuje přehled klíčových funkcí systému:

- Autentizace a bezpečnost
 - Možnost přihlášení uživatele pomocí e-mailu a hesla.
 - Ověření e-mailové adresy po registraci.
 - Možnost resetování zapomenutého hesla prostřednictvím e-mailu.
- Správa uživatelů
 - Registrace nových uživatelů prostřednictvím systému.
 - Přiřazování uživatelů k jedné nebo více pobočkám.
 - Dynamická správa rolí a přiřazování oprávnění jednotlivým uživatelům.
- Dostupnost
 - Uživatelé si mohou zaznamenat svou dostupnost do kalendáře.

- Správa směn
 - Tvorba směn pro konkrétní dny.
 - Přiřazení směn konkrétním uživatelům.
 - Zobrazení směn v přehledném kalendáři.
 - Individuální zobrazení směn přiřazených jednotlivým uživatelům.
- Správa poboček
 - Možnost vytváření nových poboček (zařízení).
 - Přiřazování uživatelů k jednotlivým pobočkám.

4.1.2 Nefunkcionální požadavky

Nefunkcionální požadavky definují vlastnosti systému, které nesouvisejí s konkrétními akcemi uživatelů, ale určují kvalitu, výkonnost, bezpečnost a provozní podmínky aplikace. Tyto požadavky zajišťují, že aplikace bude spolehlivá, přenositelná, udržitelná a použitelná v reálném nasazení.

Následuje přehled klíčových nefunkcionálních požadavků systému:

- Platformní a technologické požadavky
 - Aplikace bude provozována jako webová aplikace dostupná přes internet.
 - Nasazení proběhne na open-source infrastruktuře.
 - Backend je postaven na PHP frameworku Laravel, databáze na MySQL.
- Dostupnost
 - Aplikace bude dostupná 24/7 bez nutnosti lokální instalace na straně klienta.
- Bezpečnost
 - Ochrana proti běžným typům webových útoků:
 - CSRF (Cross-Site Request Forgery)
 - XSS (Cross-Site Scripting)
 - SQL Injection
 - Bezpečné ukládání hesel pomocí hashovací funkce (bcrypt).

- Ověření e-mailové adresy a kontrola oprávnění při každém přístupu k citlivým částem systému.
- Výkonnost
 - Systém zvládne obsloužit desítky paralelních uživatelů bez ztráty plynulosti.
 - Udržitelnost a rozšiřitelnost
 - Aplikace bude postavena modulárně s důrazem na čitelný a přehledný kód.
 - Použití migrací a seedů umožní snadné verzování databáze.
 - Podpora více rolí a provozoven bez potřeby zásahu do jádra aplikace.
- Nasazení a správa
 - Projekt bude mít připravené README.md s postupem nasazení na server.
 - Nasazení bude možné prostřednictvím Composeru, CLI nástrojů

4.2 Návrh uživatelského rozhraní

Uživatelské rozhraní bylo navrženo tak, aby bylo přehledné, jednoduché a aby byla jednoduchá orientace jak pro administrátory, tak pro běžné zaměstnance. Pro vizuální návrh byly vytvořeny wireframy v nástroji Visily, které sloužily jako hrubý podklad pro realizaci aplikace. Tyto návrhy zahrnují hlavní obrazovky a klíčové komponenty.

Po přihlášení uživatel vstupuje na hlavní přehledovou obrazovku (jak je vidět na obrázku č. 2 níže), která zobrazuje měsíční kalendář s vyznačenými směnami. Uživatel zde okamžitě vidí, kdy a na které pobočce má pracovat. V původním návrhu se zobrazovaly zejména časové údaje směny, ale v praxi se ukázalo užitečné k těmto informacím přiřadit také název pobočky. Pro uživatele, kteří pracují na více provozovnách, to významně zvyšuje přehlednost a snižuje riziko záměny směn, protože hned na první pohled vidí nejen čas, ale i konkrétní místo výkonu práce.

Leden 2025

po	út	st	čt	pá	so	ne
		1 12:00 - 17:00	2	3	4	5
6	7	8 12:00 - 17:00	9	10	11	12
13	14	15 08:00 - 16:45	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Obrázek 2 - Wireframe úvodního kalendáře

(zdroj: vlastní)

Dalším prvkem uživatelského rozhraní je hlavní menu (viz obrázek č. 3), které obsahuje navigační lištu umožňující přepínání mezi klíčovými sekcemi systému. Menu zahrnuje položky jako „Mé směny“, „Dostupnost“, „Zaměstnanci“, „Směny“, „Management poboček“. Díky této horní liště má uživatel rychlý přístup k požadovaným funkcím, a to bez nutnosti zdlouhavého proklikávání. Rozbalovací menu v pravém horním rohu navíc umožňuje výběr konkrétní pobočky. Navigační menu bylo v průběhu vývoje upraveno. Byly změněny názvy jednotlivých položek tak, aby lépe odpovídaly reálné funkčnosti.

☐	Mé směny	Dostupnost	Zaměstnanci	Směny ▾	Management poboček	Pobočka Praha ▾	Jméno Příjmení ▾
--------------------------	----------	------------	-------------	---------	--------------------	-----------------	------------------

Obrázek 3 - Wireframe menu

(zdroj: vlastní)

Pro zadávání časových preferencí slouží stránka plánování dostupnosti, kde si zaměstnanec volí, zda je dostupný celý den, pouze v určitý čas, nedostupný nebo na dovolené (viz obrázek č. 4 níže). Tyto preference se vizuálně zaznamenávají do kalendáře a jsou následně využívány při sestavování směn. Oproti tomuto návrhu je však v aktuální podobě systému rozšířena nabídka stavů ještě o jeden – nedostupný specifický čas, který umožňuje

uživateli přesně definovat, ve kterých hodinách nemůže pracovat. Současně bylo také upraveno tlačítko pro uložení dostupnosti, které je nově dostupné hned nad kalendářem.

Leden 2025

po	ut	st	ct	pa	so	ne
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Uložit dostupnost

Obrázek 4 - Wireframe stránky dostupnosti

(zdroj: vlastní)

Pro uživatele, kteří mají pravomoce editovat směny, slouží stránka pro editaci směn, kde mohou detailně plánovat směny pro konkrétní den a pobočku (viz obrázek č. 5 níže). Stránka zobrazuje časovou osu dne rozdělenou do hodinových bloků, na které jsou vizuálně přehledně znázorněny pracovní úseky jednotlivých zaměstnanců. Oproti původnímu návrhu je tato stránka rozšířena o input s výběrem konkrétního data, což usnadňuje přepínání mezi dny. Dále zde bylo doplněno modální okno pro editaci směny, které umožňuje rychlé a pohodlné upravování již přiřazených směn přímo z rozhraní, aniž by bylo nutné je mazat a zadávat znovu.

Obrázek 5 - Wireframe stránky editace směn

(zdroj: vlastní)

Další klíčovou součástí uživatelského rozhraní je stránka s přehledem již platných směn pro konkrétní pobočku a zvolený měsíc (viz obrázek č. 6 níže). Tabulka zobrazuje jednotlivé dny v měsíci v horní části a jména zaměstnanců ve sloupcích. U každého zaměstnance jsou pak přehledně vyznačeny časy jeho směn. Oproti původnímu návrhu bylo rozhraní této stránky v aplikaci dále rozšířeno o tlačítka umožňující přechod mezi jednotlivými měsíci.

Leden 2025

Zaměstnanec	1 st	2 čt	3 pá	4 so	5 ne	6 po	7 út	8 st	9 čt	10 pá	11 so	12 ne
Jméno Příjmení	14:00 16:00		14:00 16:00			14:00 16:00				14:00 16:00		14:00 16:00
Jméno Příjmení		14:00 16:00				14:00 16:00	14:00 16:00		14:00 16:00		14:00 16:00	
Jméno Příjmení						14:00 16:00		14:00 16:00				

Obrázek 6 - Wireframe přehledu směn pro pobočku

(zdroj: vlastní)

4.3 Návrh databázové struktury

Databázová struktura byla navržena s důrazem na modularitu, přehlednost. Návrh reflektuje požadavky na plánování směn ve více provozovnách a řízení přístupových práv.

Obrázek č. 7 níže, znázorňuje výsledný model databáze, který slouží jako technický základ implementace aplikační logiky. Podrobnosti k jednotlivým tabulkám a jejich vazbám jsou popsány v následujících podkapitolách.



Obrázek 7 - Struktura databáze

(zdroj: vlastní)

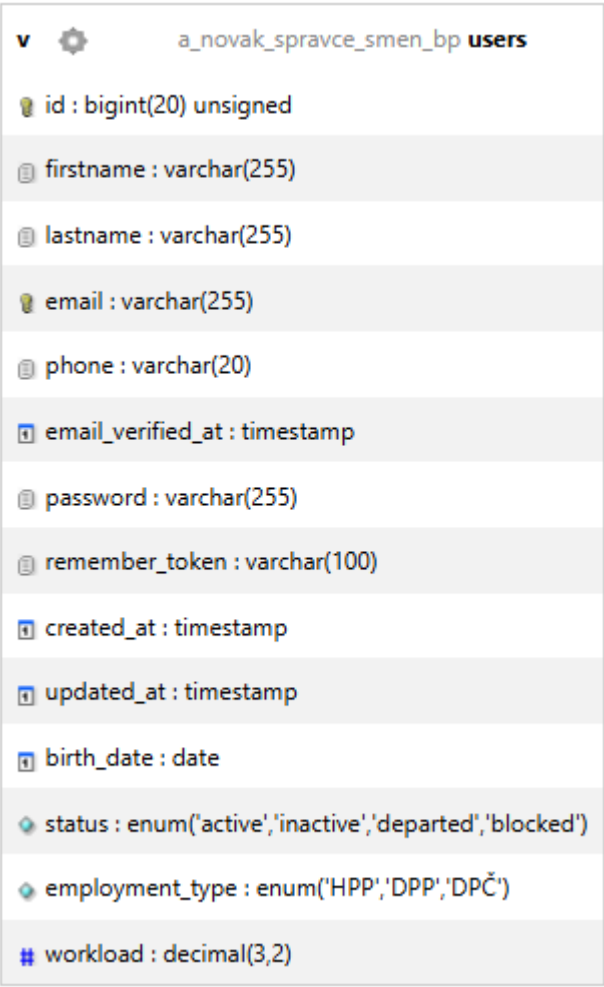
4.3.1 Popis tabulek a relací

Databázová vrstva tvoří páteř celé aplikace a zajišťuje konzistentní uložení všech důležitých entit i jejich vzájemných vztahů. V této podkapitole jsou podrobně popsány jednotlivé tabulky databáze včetně jejich funkcí a vazeb na ostatní části systému. Každé entitě odpovídá samostatná tabulka, přičemž pro realizaci složitějších vztahů – zejména typu N:M – jsou využity pomocné (pivotní) tabulky.

Popis vychází z návrhu databázového schématu (viz obrázek č. 7) a poskytuje náhled na strukturu uložených dat.

4.3.1.1 Tabulka users

Tabulka users představuje základní evidenci všech uživatelů systému, tedy zaměstnanců i administrátorů. Uchovává osobní údaje jako jméno, příjmení, e-mail, telefonní číslo a datum narození. Zároveň obsahuje technické údaje pro autentizaci a správu uživatelského účtu, včetně hesla, tokenu pro zapamatování relace a stav ověření e-mailu. Každý uživatel má typ pracovního poměru (HPP, DPP). Dále se eviduje jeho úvazek ve formátu desetinného čísla. Jeden uživatel může být přiřazen k více provozovnám prostřednictvím spojovací tabulky facility_user a může mít přiřazeno více směn (shifts). Uživatel je rovněž napojen na tabulku dostupností (availabilities) a relací přihlášení (sessions). Role a oprávnění jsou přiřazovány skrze polymorfní tabulky model_has_roles a model_has_permissions.



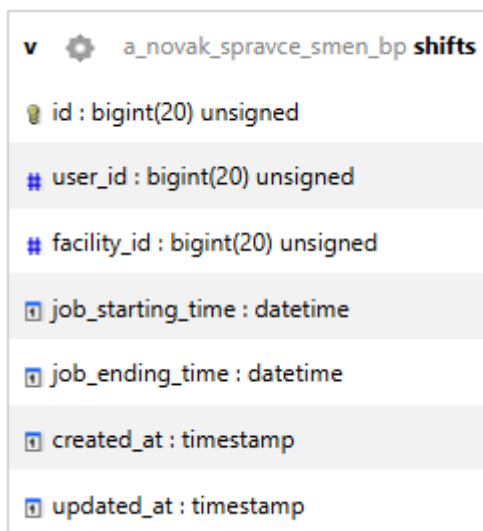
		a_novak_spravce_smen_bp	users
	id	: bigint(20)	unsigned
	firstname	: varchar(255)	
	lastname	: varchar(255)	
	email	: varchar(255)	
	phone	: varchar(20)	
	email_verified_at	: timestamp	
	password	: varchar(255)	
	remember_token	: varchar(100)	
	created_at	: timestamp	
	updated_at	: timestamp	
	birth_date	: date	
	status	: enum('active','inactive','departed','blocked')	
	employment_type	: enum('HPP','DPP','DPČ')	
	workload	: decimal(3,2)	

Obrázek 8 - Tabulka users

(zdroj: vlastní)

4.3.1.2 Tabulka shifts

Tabulka shifts reprezentuje směny, které jsou naplánované pro konkrétní uživatele v rámci určitých provozoven. Každá směna je určena začátkem a koncem v přesném datovém a časovém formátu. Směna je vždy přiřazena konkrétnímu uživateli (`user_id`) a zároveň konkrétní provozovně (`facility_id`), čímž vzniká přímá vazba na tabulky `users` a `facilities`.



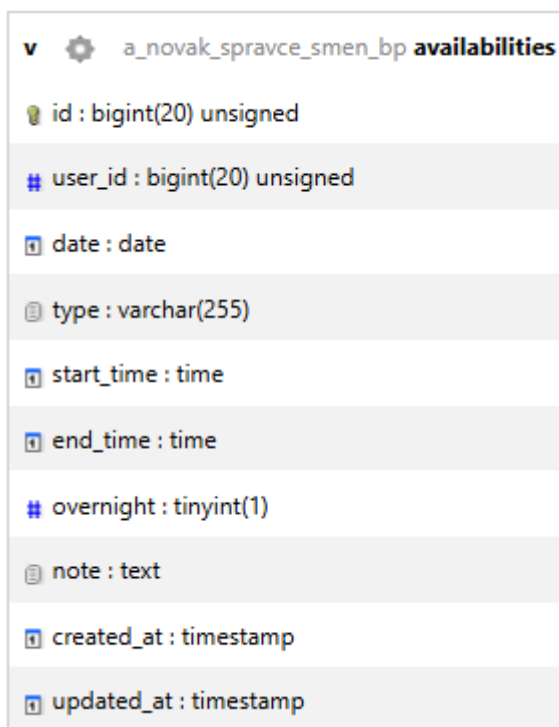
v	⚙	a_novak_spravce_smen_bp	shifts
🔑		id	: bigint(20) unsigned
#		user_id	: bigint(20) unsigned
#		facility_id	: bigint(20) unsigned
🕒		job_starting_time	: datetime
🕒		job_ending_time	: datetime
🕒		created_at	: timestamp
🕒		updated_at	: timestamp

Obrázek 9 - Tabulka shifts

(zdroj: vlastní)

4.3.1.3 Tabulka availabilities

Tabulka availabilities uchovává dostupnost uživatelů pro plánování směn. Každý záznam obsahuje datum, typ dostupnosti (např. dostupný, nedostupný), časový rozsah (od-do) a atribut, zda se jedná o přesah do dalšího dne. Dostupnost je vždy vázaná na konkrétního uživatele pomocí `user_id`.



v	⚙	a_novak_spravce_smen_bp	availabilities
🔑		id	: bigint(20) unsigned
#		user_id	: bigint(20) unsigned
📅		date	: date
📄		type	: varchar(255)
🕒		start_time	: time
🕒		end_time	: time
#		overnight	: tinyint(1)
📄		note	: text
🕒		created_at	: timestamp
🕒		updated_at	: timestamp

Obrázek 10 - Tabulka availabilities

(zdroj: vlastní)

4.3.1.4 Tabulka facilities

Tabulka facilities uchovává informace o jednotlivých provozovnách nebo pobočkách, kde uživatelé pracují. Každá provozovna má svůj název, lokaci a volitelnou barvu, která slouží pro vizuální odlišení například v kalendářovém zobrazení směn.

Kromě těchto základních údajů obsahuje tabulka i atribut is_main, který slouží k označení hlavní provozovny. Každý uživatel musí mít přístup alespoň k jedné provozovně, která je označena jako hlavní, což umožňuje správné fungování systému i v případech, kdy je více provozoven. Tento příznak je důležitý například pro výchozí zobrazení směn nebo výběr provozovny při přihlášení.

Provozovna může být propojena s více směnami (shifts) a také s více uživateli pomocí pivot tabulky facility_user, která umožňuje správu vztahů mezi uživateli a jednotlivými provozovami.

v	a_novak_spravce_smen_bp	facilities
id	: bigint(20) unsigned	
name	: varchar(255)	
location	: varchar(255)	
color	: varchar(255)	
created_at	: timestamp	
updated_at	: timestamp	
is_main	: tinyint(1)	

Obrázek 11 - Tabulka facilities

(zdroj: vlastní)

4.3.1.5 Tabulka facility_user

Tabulka facility_user slouží jako spojovací (pivot) tabulka pro realizaci vazby typu N:M (many-to-many) mezi entitami users a facilities. Umožňuje reprezentovat situaci, kdy jeden uživatel může být přiřazen k více provozovnám, a zároveň jedna provozovna může být sdílena více uživateli. Každý záznam v této tabulce tvoří konkrétní instanci propojení mezi daným uživatelem a provozovnou.

v	a_novak_spravce_smen_bp	facility_user
id	: bigint(20) unsigned	
user_id	: bigint(20) unsigned	
facility_id	: bigint(20) unsigned	
created_at	: timestamp	
updated_at	: timestamp	

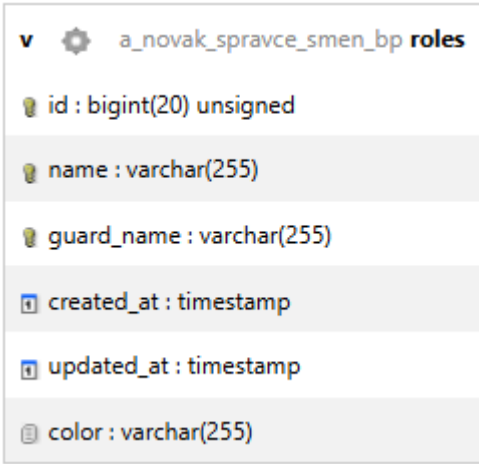
Obrázek 12 - Tabulka facility_user

(zdroj: vlastní)

4.3.1.6 Tabulka roles

Tabulka roles slouží k evidenci jednotlivých rolí uživatelů v systému (např. administrátor, manažer směn). Každá role má svůj název (name), název guardu a barvu v HEX formátu pro

vizuální rozlišení. Tabulka je součástí automaticky generované struktury balíčku Spatie Permissions pro Laravel, který usnadňuje správu rolí a oprávnění. Role jsou uživatelům přiřazovány prostřednictvím vazby many-to-many skrze spojovací tabulku `model_has_roles`.



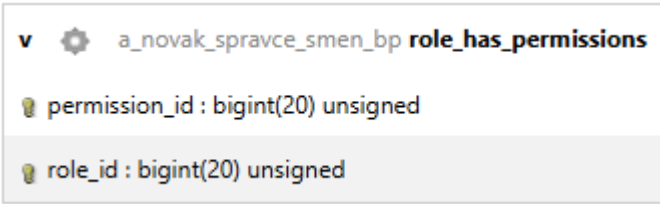
v  a_novak_spravce_smen_bp roles
 id : bigint(20) unsigned
 name : varchar(255)
 guard_name : varchar(255)
 created_at : timestamp
 updated_at : timestamp
 color : varchar(255)

Obrázek 13 - Tabulka roles

(zdroj: vlastní)

4.3.1.7 Tabulka `role_has_permissions`

Tabulka `role_has_permissions` zajišťuje přiřazení oprávnění jednotlivým rolím. Každý záznam obsahuje identifikátor oprávnění (`permission_id`) a identifikátor role (`role_id`), čímž vzniká přímá many-to-many vazba mezi entitami `roles` a `permissions`. Tato tabulka je také součástí automaticky generované struktury balíčku Spatie Permissions.



v  a_novak_spravce_smen_bp role_has_permissions
 permission_id : bigint(20) unsigned
 role_id : bigint(20) unsigned

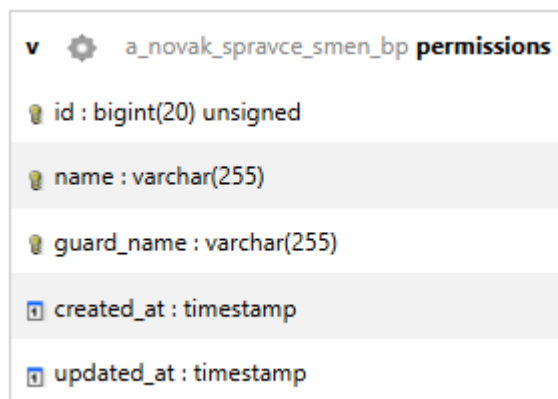
Obrázek 14 - Tabulka `role_has_permissions`

(zdroj: vlastní)

4.3.1.8 Tabulka `permissions`

Tabulka `permissions` uchovává seznam jednotlivých oprávnění, která definují specifické akce nebo přístupy v rámci systému (např. správa uživatelů, úprava směn). Každé oprávnění

má svůj název (*name*) a název guardu. Tato tabulka je součástí automaticky generované struktury balíčku Spatie Permissions.



v	a_novak_spravce_smen_bp	permissions
		id : bigint(20) unsigned
		name : varchar(255)
		guard_name : varchar(255)
		created_at : timestamp
		updated_at : timestamp

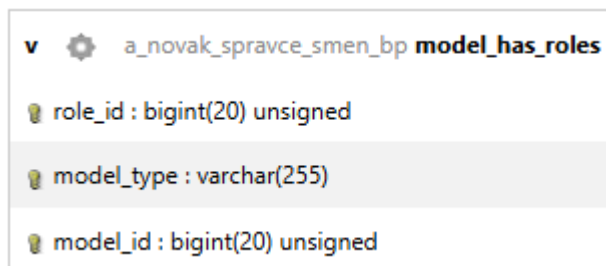
Obrázek 15 - Tabulka permissions

(zdroj: vlastní)

4.3.1.9 Tabulky *model_has_roles* a *model_has_permissions*

Tabulky *model_has_roles* a *model_has_permissions* slouží k přiřazování rolí a oprávnění jednotlivým instancím modelů – typicky uživatelům. Obě tabulky využívají many-to-many vazbu, která umožňuje přiřadit roli nebo oprávnění libovolnému modelu v systému (nejen uživatelům), prostřednictvím kombinace sloupců *model_type* (typ modelu, např. App\Models\User) a *model_id* (ID konkrétní instance). Tyto tabulky jsou také nedílnou součástí balíčku Spatie Permissions.

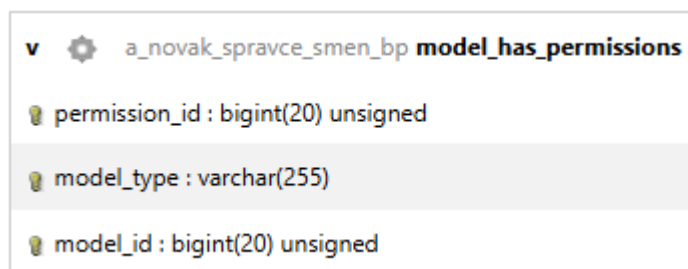
- *model_has_roles* propojuje modely s konkrétními rolemi.
- *model_has_permissions* propojuje modely s konkrétními oprávněními.



v	a_novak_spravce_smen_bp	model_has_roles
		role_id : bigint(20) unsigned
		model_type : varchar(255)
		model_id : bigint(20) unsigned

Obrázek 16 - Tabulka model_has_roles

(zdroj: vlastní)



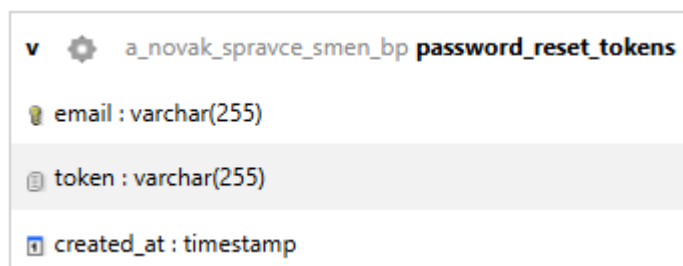
v	a_novak_spravce_smen_bp	model_has_permissions
	permission_id	bigint(20) unsigned
	model_type	varchar(255)
	model_id	bigint(20) unsigned

Obrázek 17 - Tabulka model_has_permissions

(zdroj: vlastní)

4.3.1.10 Tabulka password_reset_tokens

Tabulka password_reset_tokens slouží k uchovávání tokenů pro reset hesla. Každý záznam obsahuje e-mail uživatele, unikátní token a čas vytvoření. Tyto údaje se používají při procesu obnovení zapomenutého hesla, kdy je uživateli zaslán odkaz obsahující token, který ověřuje oprávněnost žádosti o změnu hesla. Tato tabulka je automaticky vytvořena migrací při instalaci starter kitu Laravel Breeze a je součástí výchozí konfigurace pro správu uživatelských účtů.



v	a_novak_spravce_smen_bp	password_reset_tokens
	email	varchar(255)
	token	varchar(255)
	created_at	timestamp

Obrázek 18 - Tabulka password_reset_tokens

(zdroj: vlastní)

4.3.1.11 Tabulka sessions

Tabulka sessions slouží k uchovávání informací o aktivních nebo historických relacích uživatelů. Každý záznam obsahuje jedinečný identifikátor relace (id), ID uživatele (user_id), IP adresu přihlášení, typ použitého prohlížeče (user_agent), uložený stav (payload) a čas poslední aktivity. Pomocí těchto údajů lze sledovat, kdy a odkud se uživatelé přihlašovali a zda je jejich relace stále aktivní.

Tato tabulka je automaticky generována Laravel frameworkem, pokud je jako session driver nastavena databáze (SESSION_DRIVER=database). V projektech využívajících

Laravel Breeze starter kit bývá často součástí výchozí konfigurace pro správu uživatelských relací.

v	a_novak_spravce_smen_bp	sessions
id	: varchar(255)	
#	user_id : bigint(20) unsigned	
@	ip_address : varchar(45)	
@	user_agent : text	
@	payload : longtext	
#	last_activity : int(11)	

Obrázek 19 - Tabulka sessions

(zdroj: vlastní)

5 IMPLEMENTACE APLIKACE

Tato kapitola podrobně popisuje proces implementace klíčových částí aplikace. Cílem je představit jednotlivé technologické komponenty a vysvětlit, jak byly využity k dosažení funkčního a bezpečného řešení.

5.1 Autentizace

Pro účely autentizace uživatelů v rámci systému byl využit oficiální starter kit Laravel Breeze, který nabízí jednoduchou, přehlednou a zároveň plně funkční implementaci autentizační logiky. Tento nástroj poskytuje základní mechanismy pro registraci, přihlášení, odhlášení, ověření e-mailu a správu hesla, a to jak po stránce backendové logiky, tak i s připraveným frontendem (v tomto případě s využitím Blade šablon).

Instalace Laravel Breeze probíhá pomocí následujících příkazů v konzoli:

```
composer require laravel/breeze --dev
php artisan breeze:install
npm install && npm run dev
php artisan migrate
```

Po spuštění těchto příkazů se automaticky vygenerují potřebné kontrolery, routy, Blade šablony a migrace pro práci s uživatelskými účty. Breeze využívá standardní Laravel middleware *auth* a *verified*, což usnadňuje zajištění přístupu pouze pro přihlášené nebo ověřené uživatele.

Po registraci nového uživatele se do databáze uloží záznam s neověřeným e-mailem pole *email_verified_at* zůstává *null*. Laravel automaticky odešle ověřovací e-mail prostřednictvím notifikačního systému *Illuminate\Notifications*, který obsahuje podepsaný odkaz pro ověření identity.

Tento proces je aktivovaný v modelu *User*, který implementuje rozhraní *MustVerifyEmail*:

```
class User extends Authenticatable implements MustVerifyEmail, JWTSubject
{
    use HasFactory, Notifiable, HasApiTokens, HasRoles;
}
```

Díky tomu Laravel automaticky zapne logiku pro ověření e-mailu a přidá odpovídající routy.

Laravel Breeze přidává routy potřebné pro ověření e-mailu, obnovení hesla do souboru *routes/auth.php*.

5.1.1 Obnova hesla

Součástí balíčku *Breeze* je i plně implementovaný mechanismus pro obnovu zapomenutého hesla. Uživatel může požádat o reset hesla prostřednictvím e-mailu, ve kterém mu je doručen odkaz na formulář pro zadání nového hesla. Laravel interně používá tabulku *password_reset_token* a příslušný *NewPasswordController* pro zajištění bezpečného zpracování požadavku.

5.2 Autorizace – Správa rolí a oprávnění

Pro správu přístupových práv uživatelů byla využita populární knihovna Spatie Laravel Permission. Tento open-source balíček poskytuje přehledné a robustní rozhraní pro definici rolí (roles) a oprávnění (permissions), které lze dynamicky přiřazovat jednotlivým uživatelům nebo jiným modelům.

5.2.1 Základní princip

Balíček zavádí čtyři hlavní tabulky:

- roles – definice jednotlivých rolí (např. admin, zaměstnanec),
- permissions – konkrétní oprávnění (např. „edit shifts“, „view users“),
- model_has_roles a model_has_permissions – pivot tabulky propojující role/oprávnění s konkrétními uživateli (polymorfně),
- role_has_permissions – určuje, jaká oprávnění náleží konkrétním rolím.

V projektu byly tyto tabulky vytvořeny pomocí migračního souboru *2025_02_28_201303_create_permission_tables.php*

Instalace balíčku proběhla dle oficiální dokumentace pomocí následujících příkazů:

```
composer require spatie/laravel-permission
php artisan vendor:publish --
    provider="Spatie\Permission\PermissionServiceProvider"
```

5.2.2 Rozšíření modelu Role:

Pro zvýšení přehlednosti a možnost barevného odlišení rolí v uživatelském rozhraní byla základní tabulka roles rozšířena o nový sloupec color. Tato úprava byla realizována vlastní migrací:

```
public function up(): void
{
    Schema::table('roles', function (Blueprint $table) {
        $table->string('color')->default('#47126B');
    });
}
```

Tato barva se následně využívá např. v tabulkách ve frontendovém rozhraní – například zaměstnanci s rolí „vedoucí“ mohou být označeni žlutě, zatímco administrátor červeně apod.

5.2.3 Balíček Spatie

Aplikace využívá výchozí Laravel guard web, jak je uvedeno v konfiguračním souboru *config/auth.php*:

```
'guards' => [
    'web' => [
        'driver' => 'session',
        'provider' => 'users',
    ],
    'api' => [
        'driver' => 'jwt',
        'provider' => 'users',
    ],
],
```

Guard zajišťuje, že oprávnění platí pouze v daném autentizačním kontextu (např. pro web vs. API).

V projektu je vědomě vypnuta podpora týmů, což zjednodušuje správu oprávnění. Konkrétně se v souboru *config/permission.php* nachází následující nastavení:

```
'teams' => false,
```

Toto omezení znamená, že role a oprávnění nejsou navázány na žádnou týmovou strukturu a aplikují napříč celou aplikací. Toto rozhodnutí má praktický dopad i na technickou stránku projektu. Pokud by byla podpora týmů zapnuta, balíček Spatie Permission by automaticky generoval dodatečné databázové tabulky, jako například tabulku teams a s ní i související pivot tabulky pro sledování vztahů mezi uživateli, rolemi a týmy.

Vzhledem k tomu, že aplikace není určena pro prostředí, kde by bylo nutné oddělovat oprávnění podle týmů, by zapnutí této funkce přineslo zbytečnou komplikaci. Naopak, vypnutím této funkce je dosaženo jednodušší správy práv, méně složitého schématu databáze. Celý systém se tak soustředí na to, aby role a oprávnění byly přehledně a jednotně aplikovány na všechny uživatele bez ohledu na to, do jakého týmu patří. Tento přístup je plně dostačující pro potřeby malého podniku, pro který je tento plánovač směn navržen.

V seederu *RoleSeeder.php* jsou definovány výchozí role a jim přiřazená oprávnění:

```

public function run()
{
    Permission::create(['name' => 'create facilities']);
    Permission::create(['name' => 'delete facilities']);
    Permission::create(['name' => 'view facilities']);
    Permission::create(['name' => 'edit facilities']);

    Permission::create(['name' => 'manage facility users']);

    Permission::create(['name' => 'view shifts']);
    Permission::create(['name' => 'view others shifts']);
    Permission::create(['name' => 'edit shifts']);

    Permission::create(['name' => 'view users']);
    Permission::create(['name' => 'create users']);
    Permission::create(['name' => 'edit users']);
    Permission::create(['name' => 'delete users']);

    Permission::create(['name' => 'view availability']);
    Permission::create(['name' => 'edit availability']);

    Permission::create(['name' => 'create roles']);
    Permission::create(['name' => 'delete roles']);
    Permission::create(['name' => 'view roles']);
    Permission::create(['name' => 'edit roles']);

    $admin = Role::create(['name' => 'Admin']);

    $admin->givePermissionTo(Permission::all());

    $manager = Role::create(['name' => 'Manažer']);
    $manager->givePermissionTo([
        'view shifts',
        'view others shifts',
        'edit shifts',
        'view availability',
        'edit availability',
    ]);

    $employee = Role::create(['name' => 'Zaměstnanec']);
    $employee->givePermissionTo([
        'view shifts',
        'view others shifts',
        'view availability',
        'edit availability'
    ]);
}

```

Role se uživatelům následně přiřazují v *ProductionSeeder.php* či v kontroleru *UserController.php*:

```

// ProductionSeeder.php
$admin->assignRole('Admin');

// UserController.php
$user->assignRole($validated['role']);

```

V Blade šablonách se oprávnění využívají nejen k ochraně konkrétních akcí nebo sekcí kódu, ale také například k dynamickému zobrazení prvků uživatelského rozhraní. Příkladem je generování menu, kde jsou jednotlivé položky viditelné pouze pro uživatele s příslušnou pravomocí. Takto lze zajistit, že například odkaz na seznam uživatelů nebo správu směn uvidí pouze ti, kdo mají přiděleno příslušné oprávnění. Ukázka kódu je níže:

```
@can('view users')
    <x-dropdown-link :href="route('users.index')" :active="request()->routeIs('users.index')">
        {{ __('Seznam zaměstnanců') }}
    </x-dropdown-link>
@endcan
```

Tento přístup zajišťuje konzistentní správu oprávnění nejen na backendu (přes middleware), ale i přímo ve vizuální vrstvě aplikace, což přispívá k lepšímu uživatelskému zážitku a vyšší bezpečnosti.

5.2.4 Ochrana přístupu pomocí middleware

V rámci aplikace je přístup k většině částem systému omezen pouze na uživatele, kteří jsou přihlášení a zároveň mají ověřený e-mail. K dosažení tohoto omezení se při definici rout používá middleware *auth* a *verified*. Tyto middleware jsou zahrnuty ve společné route group, která obaluje všechny routy kromě těch, které slouží k přihlášení a obnově zapomenutého hesla. Díky tomu je zajištěno, že všechny definované routy v této skupině budou mít automaticky aplikované podmínky, že:

- uživatel je přihlášen (middleware *auth*),
- uživatel má ověřený e-mail (middleware *verified*).

Ukázka této definice:

```
Route::middleware(['auth', 'verified', CheckAndSetFacility::class])->group(function () {
    // Seznam rout s middlewarem auth, verified a CheckAndSetFacility
    Route::get('/availability',
        [App\Http\Controllers\AvailabilityController::class, 'index'])
        ->name('availability.index')
        ->middleware('permission:view availability');
    // další chráněné routy...
});
```

Tento přístup významně zvyšuje bezpečnost aplikace, protože zabráňuje přístupu neautorizovaným uživatelům a zároveň zajišťuje, že uživatel skutečně vlastní e-mailovou adresu, kterou uvedl při registraci. Jedinými výjimkami jsou routy pro přihlášení a obnovu hesla, které z logiky věci nemohou vyžadovat přihlášení ani ověření e-mailu.

Z hlediska toho, že aplikace nabízí možnost správy více poboček (facilities), bylo navíc nutné aplikovat vlastní middleware s názvem *CheckAndSetFacility*. Tento middleware zajišťuje, že uživatel bude mít v session uloženou alespoň jednu hlavní pobočku, se kterou bude pracovat. Je to nezbytné z hlediska vykreslování a výběru dat, protože již zapsané směny, které jsou platné, jsou vždy svázány s konkrétní pobočkou. Middleware tedy pomáhá udržet konzistenci a správný kontext dat v rámci aplikace, což je klíčové pro správné fungování více-pobočkového systému.

Pro efektivní využití oprávnění pomocí middleware je třeba balíček správně nakonfigurovat. To se děje v souboru *bootstrap/app.php*, kde jsou v rámci konfigurace middleware zaregistrovány aliasy, které umožňují používat přímo v definicích rout následující middleware:

- `role` – ověřuje, zda má uživatel přiřazenou určitou roli,
- `permission` – ověřuje, zda má uživatel přiděleno konkrétní oprávnění,
- `role_or_permission` – ověřuje, zda má uživatel alespoň jednu z daných rolí nebo oprávnění.

Ukázka implementace middleware dle dokumentace v souboru *bootstrap/app.php*:

```
->withMiddleware(function (Middleware $middleware) {  
    $middleware->alias([  
        'role' => \Spatie\Permission\Middleware\RoleMiddleware::class,  
        'permission' =>  
        \Spatie\Permission\Middleware\PermissionMiddleware::class,  
        'role_or_permission' =>  
        \Spatie\Permission\Middleware\RoleOrPermissionMiddleware::class,  
    ] );  
})
```

Po této registraci lze velmi jednoduše přidat middleware přímo k jednotlivým routám nebo celým route skupinám. To znamená, že není nutné psát složité podmínky uvnitř kontrolerů – přístup k určitým částem aplikace je možné spravovat přímo na úrovni rout. Například:

```
Route::post('/users', [UserController::class, 'store'])  
    ->name('users.store')  
    ->middleware('permission:create users');
```

Tímto způsobem je zajištěno, že k vytváření nových uživatelů bude mít přístup pouze uživatel s příslušným oprávněním „create users“.

Výhodou tohoto přístupu je vysoká přehlednost a centralizace pravidel pro autorizaci – je ihned zřejmé, jaké oprávnění je potřeba k přístupu ke konkrétní routě. Navíc je tato metoda velmi dobře škálovatelná, protože přidávání nových oprávnění nebo úpravy existujících pravidel lze provést snadno a bez zásahu do samotné obchodní logiky aplikace.

Celkově tedy využití balíčku Spatie Permission spolu s middleware výrazně přispívá k bezpečnosti a udržitelnosti aplikace, zejména v případech, kdy je potřeba spravovat komplexnější hierarchii rolí a oprávnění.

5.3 Modely

Modely v Laravel frameworku reprezentují jednotlivé entity databáze a zajišťují logiku práce s daty prostřednictvím ORM systému Eloquent. V rámci aplikace slouží modely k obsluze klíčových entit: uživatelé, směny, provozovny a dostupnost. Každý model odpovídá jedné databázové tabulce a definuje vztahy mezi entitami.

User

Model User reprezentuje uživatele systému – zaměstnance, vedoucí i administrátory. Využívá několik Laravel traitů jako *HasRoles*, *HasApiTokens*, *Notifiable* a *HasFactory*, které poskytují podporu pro správu oprávnění, API autentizaci, notifikace a generování testovacích dat. Model implementuje rozhraní *MustVerifyEmail* pro ověřování e-mailové adresy uživatele a *JWTSubject* pro podporu autentizace pomocí JSON Web Tokenu pro budoucí použití API pro mobilní aplikaci.

Uživatelé mají definovaný vztah many-to-many s provozovnami prostřednictvím pivot tabulky *facility_user*. Dále jsou ve vztahu one-to-many se směny (každý uživatel může mít více směn) a také s dostupnostmi. Model také vymezuje atributy, které lze hromadně plnit, i ty, které se mají při serializaci skrývat (např. heslo a tokeny).

Facility

Model Facility představuje jednotlivé pobočky nebo provozovny, ve kterých pracují uživatelé. Každá provozovna může být barevně odlišena pomocí atributu *color* a může být označena jako hlavní (*is_main*). Tento atribut je důležitý zejména při výběru výchozí provozovny po přihlášení uživatele.

Provozovna je ve vztahu many-to-many s uživateli a one-to-many se směny, které se v dané provozovně plánují. Tento model pomáhá řídit rozdělení uživatelů napříč více místy výkonu práce a je klíčový pro správu přístupů i filtrování směn podle lokace.

Shift

Model Shift reprezentuje pracovní směny přiřazené uživatelům. Každá směna je určena přesným časovým rozmezím a přiřazena konkrétnímu uživateli a provozovně.

Směna je ve vztahu many-to-one jak k uživateli, tak k provozovně. Tento model je klíčový pro zajištění vazby mezi dostupností zaměstnanců, plánováním směn a přehledným kalendářovým zobrazením.

Availability

Model Availability zachycuje dostupnost uživatele v konkrétní den a čas. Každý záznam obsahuje datum, typ dostupnosti (např. dostupný či nedostupný), časové rozpětí.

Každá dostupnost náleží jednomu uživateli prostřednictvím vztahu many-to-one. Tento model je zásadní pro logiku plánování, protože umožňuje zobrazit nebo vyhodnotit, zda může být danému uživateli v konkrétním přiřazena směna.

5.4 Migrate a seedery

V rámci projektu je důležitou součástí přípravy databázové struktury využití migrací, seederů a továrních tříd (factories), které nabízí Laravel. Tento přístup zajišťuje konzistentní a opakovatelnou inicializaci databáze při vývoji a testování aplikace.

Migrate slouží k definici struktury databázových tabulek. Ve složce *database/migrations* se nachází více než deset souborů popisujících vznik tabulek a úpravy jejich struktur. Mezi hlavní migrate patří:

- `create_users_table.php` – Definuje základní tabulku uživatelů včetně polí jako jméno, email, heslo a další identifikační údaje.
- `create_facilities_table.php` – Vytváří tabulku pro zařízení (např. pracoviště), která mohou být přiřazena uživatelům.
- `create_shifts_table.php` – Obsahuje definici směn, včetně datumu a času začátku a konce.
- `create_facility_user_table.php` – Pivot tabulka propojující uživatele s jednotlivými zařízeními (many-to-many vztah).
- `create_availabilities_table.php` – Uchovává dostupnost uživatelů pro plánování směn.

- `create_permission_tables.php` – Vytváří potřebné tabulky pro správu oprávnění a rolí (spolu s Laravel Permission balíčkem od Spatie).

a další pomocné migrace jako např. `add_color_to_roles_tables.php`, které rozšiřují existující tabulky o nové atributy.

Seeder třídy naplňují databázi výchozími nebo testovacími daty. V projektu jsou umístěny ve složce `database/seeder`s a zahrnují:

- `DatabaseSeeder.php` – Hlavní vstupní bod pro spouštění všech seederů. V projektu se spouští seedery pro zařízení, uživatele, směny, role apod.
- `FacilitySeeder.php` – Vytváří výchozí zařízení (např. „Pobočka Praha“).
- `FacilityUserSeeder.php` – Přiřazuje uživatele k zařízení.
- `RoleSeeder.php` – Definuje role jako admin, vedoucí směn, zaměstnanec
- `ShiftSeeder.php` – Generuje testovací směny.
- `ProductionSeeder.php` – Slouží při produkční instalaci, vytváří základní pobočky, role a administrátorský účet.

Seeder třídy využívají modelové továrny (factories), které usnadňují generování realistických dat. Tyto factories umožňují snadno vytvářet fiktivní data pro testování. Ve složce `database/factories` jsou definovány:

- `UserFactory.php` – Generuje uživatele se jménem, e-mailem, heslem a dalšími náhodnými údaji pomocí knihovny *Faker*.
- `ShiftFactory.php` – Vytváří směny s náhodným začátkem a koncem v rámci definovaného časového rozsahu.

Ukázka kódu factories:

```
public function definition(): array
{
    return [
        'firstname' => fake()->firstName(),
        'lastname' => fake()->lastName(),
        'email' => fake()->unique()->safeEmail(),
        'email_verified_at' => now(),
        'phone' => fake()->phoneNumber(),
        'password' => static::$password ??= Hash::make('Password123@@'),
        'remember_token' => Str::random(10),
    ];
}
```

Použití migrací, seedrů a factories přináší následující výhody:

- Automatizace – Celý proces vytvoření databáze je opakovatelný.
- Rychlost vývoje – Díky factories lze rychle generovat data pro testování.
- Modularita – Jednotlivé části jsou přehledně rozděleny do samostatných tříd.

Pro naplnění databáze postačí spustit jednoduchý příkaz:

```
php artisan migrate:fresh --seed
```

5.5 Frontend

Frontend aplikace je postaven pomocí Blade šablonovacího systému, který je jedním ze standardních postupů při práci s frameworkem Laravel. Blade umožňuje snadno kombinovat PHP kód s HTML a nabízí praktické funkce jako dědění šablon, cykly, podmínky nebo includování komponent, což zajišťuje přehledný a udržitelný kód.

Jak je zmíněno v předchozích kapitolách, projekt využívá Laravel Breeze starterkit, který poskytuje předpřipravené řešení pro základní funkce, jako je registrace, přihlášení nebo reset hesla, a to včetně frontendové části těchto stránek. Breeze zároveň definuje základní strukturu aplikace – frontend je rozdělen na opakovaně použitelné komponenty, které se využívají napříč celou aplikací. Mezi takové komponenty patří například navigace, rozhraní pro přihlášené uživatele nebo komponenty tlačítek.

Dalším prvkem frontendové části je také využití Tailwind CSS. Laravel Breeze využívá tento framework pro stylování, a proto bylo při tvorbě projektu ponecháno tohot přístupu. Všechny nově vytvářené části frontendu rovněž využívají třídy Tailwindu. Tailwind CSS poskytuje jednoduchý a efektivní způsob, jak psát jednotné styly přímo v HTML souborech bez nutnosti vytvářet složité CSS soubory. Díky tomu je dosaženo jednotného vizuálního stylu celé aplikace a zároveň rychlé a efektivní tvorby uživatelského rozhraní.

5.5.1 Kalendář

Jednou z klíčových částí aplikace je kalendář, který slouží k přehlednému zobrazení směn a volby dostupnosti. Pro jeho implementaci byla použita javascriptová knihovna *FullCalendar*, která poskytuje řešení pro vykreslování kalendářů přímo ve webovém prohlížeči.

FullCalendar umožňuje nejen zobrazení směn, ale i interakci s uživatelem. V aplikaci je tohoto využito při plánování dostupnosti. Komunikace mezi frontendem a backendem

probíhá pomocí HTTP requestů. Při načítání kalendáře FullCalendar posílá AJAX požadavky na předem definované Laravel routy. Tyto routy směřují na funkce v kontroleru *ShiftController*. Tento kontroler vrací data ve formátu JSON, které FullCalendar využívá k vykreslení směn.

Ukázka komunikace frontendová část:

```
function loadEvents(month, year) {
    const monthKey = getMonthKey(year, month);
    ...
    fetch('/availability/events?' + new URLSearchParams({
        month: month,
        year: year
    }))
        .then(response => response.json())
        .then(data => {
            data.forEach(event => {
                ...
                calendar.addEvent(calendarEvent);
                ...
            });
        })
}
```

Na backendu se data připravují pro frontend následovně:

```
public function userShifts() {
    $user = Auth::user();

    $currentMonth = request('month', now()->month);
    $currentYear = request('year', now()->year);
    ...

    // Načtení směn pro vybraný měsíc a rok
    $shifts = $user->shifts()
        ->with('facility')
        ->whereMonth('job_starting_time', $currentMonth)
        ->whereYear('job_starting_time', $currentYear)
        ->get();

    $events = [];
    foreach ($shifts as $shift) {
        $events[] = [
            'title' => $shift->facility->name . ': <br>' .
                Carbon::parse($shift->job_starting_time)->format('H:i') . ' - ' .
                Carbon::parse($shift->job_ending_time)->format('H:i'),
            'start' => $shift->job_starting_time,
            'end' => $shift->job_ending_time,
        ];
    }

    if (request()->ajax()) {
        return response()->json($events);
    }
}
```

6 ZHODNOCENÍ ZABEZPEČENÍ

Bezpečnost webové aplikace je zásadním bodem zejména v systémech, které pracují s citlivými údaji o uživateli a řídí jejich pracovní rozvrhy. V této kapitole je analyzována úroveň zabezpečení vyvíjené aplikace, kde jsou popsány konkrétní ochranné mechanismy implementované pomocí Laravel frameworku, nativních funkcí a rozšíření.

6.1 Ochrana proti webovým útokům

Bezpečnost webové aplikace je zásadním aspektem, zvláště pokud pracuje s citlivými uživatelskými daty. V této kapitole jsou popsány konkrétní ochranné mechanismy, které byly v aplikaci implementovány pomocí Laravel frameworku.

6.1.1 CSRF – Cross-Site Request Forgery

Laravel automaticky chrání všechny formuláře proti tzv. CSRF útokům prostřednictvím middleware *VerifyCsrfToken*. Každý požadavek typu POST, PUT, PATCH nebo DELETE musí obsahovat platný CSRF token, jinak bude zamítnut. Tento token je generován pro každého uživatele a Laravel jej automaticky vkládá do formulářů pomocí Blade direktivy `@csrf`. Vývojář nemusí CSRF ochranu nastavovat ručně – stačí v šabloně použít `@csrf`, a Laravel vloží do formuláře skrytý `<input>` s tokenem, který bude při odeslání ověřen.

Ukázka formuláře aplikace, který využívá CSRF tokenu:

```
<form method="POST" action="{{ route('users.store') }}">
    @csrf
    <div class="grid grid-cols-2 gap-4">
        <div class="mb-4">
            <label for="firstname">Jméno:</label>
            <input type="text" id="firstname" name="firstname" value="{{
old('firstname') }}" required>
            @error('firstname')
            <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
            @enderror
        </div>
    </div>
    ...
</form>
```

Přítomnost `@csrf` zajistí, že při odeslání bude Laravel schopen ověřit, zda požadavek pochází od oprávněného uživatele v rámci aktivní relace. Tímto způsobem je zajištěno, že formuláře nemohou být zneužity skrze externí webové stránky, což výrazně zvyšuje bezpečnost celé aplikace.

6.1.2 XSS – Cross-Site Scripting

K ochraně proti XSS útokům Laravel využívá automatické escapování výstupu v Blade šablonách pomocí:

```
{{ $data }}
```

Jakékoliv HTML nebo JavaScript kódy vložené uživatelem jsou převedeny na neškodný textový výstup. Explicitní vypnutí escapování je možné pomocí {!! !!}, ale v této aplikaci se nikde nevyužívá, čímž je eliminováno riziko podstrčeného skriptu.

6.1.3 SQL Injection

Aplikace je postavena na ORM Eloquent, které veškeré databázové dotazy automaticky parametrizuje. Tím je eliminována možnost SQL injection, protože uživatelské vstupy nejsou nikdy přímo součástí SQL dotazů. Všechna filtrování, validace a dotazování jsou realizována bezpečně přes Eloquent nebo Query Builder.

6.2 Bezpečnost uživatelských účtů a rozhraní API

Bezpečnost uživatelských účtů a správně navržené zabezpečení rozhraní API jsou zásadní pro ochranu dat a přístupů v moderních webových aplikacích. Tato část práce se věnuje popisu způsobů, jakými aplikace chrání citlivé informace uživatelů a jak zajišťuje bezpečný přístup k (API).

6.2.1 Hashování hesel

Hesla uživatelů jsou ukládána v zahashované podobě pomocí algoritmu *bcrypt*, který je výchozím řešením v Laravelu. Tento způsob ukládání hesel výrazně zvyšuje bezpečnost, protože ani v případě úniku databáze není možné hesla zpětně rekonstruovat.

6.2.2 JWT podpora a zabezpečení API

Model *User* implementuje rozhraní *JWTSubject*, což umožňuje autentizaci pomocí JSON Web Tokenu (JWT). Tato forma autentizace je již v projektu ukázkově nastavena v API routách (*routes/api.php*) a připravena pro budoucí rozšíření systému – například o mobilní aplikaci. JWT autentizace umožňuje bezpečný přístup, kdy jsou uživatelské požadavky autorizovány pomocí podepsaného tokenu, nikoliv přihlašovacích údajů. Token je odeslán v hlavičce každého požadavku a ověřován na serveru, čímž se výrazně zvyšuje bezpečnost a škálovatelnost API rozhraní.

6.3 Ochrana relací a přihlášení

Aplikace využívá možnost Laravelu ukládat relace do databáze, což je bezpečnější a stabilnější varianta oproti výchozímu ukládání do souborů. Toto nastavení je konfigurováno v souboru `.env`, kde je použit klíč:

```
SESSION_DRIVER=database
```

Díky tomuto nastavení Laravel místo výchozího file driveru používá databázový, a relace jsou ukládány do tabulky `sessions`. Tato tabulka je automaticky generována. Tabulka `sessions` obsahuje informace jako je ID relace, ID přihlášeného uživatele, IP adresa, UserAgent, čas poslední aktivity.

Tato struktura umožňuje v budoucnu jednoduše implementovat pokročilé bezpečnostní funkce, pro správu více aktivních relací jednoho uživatele, ruční odhlášení z jiných zařízení, detekci podezřelého chování (např. přihlášení z neznámé IP adresy nebo zařízení), notifikace při přihlášení z nového zařízení.

Relace jsou při každém požadavku automaticky ověřovány a Laravel zajistí jejich platnost a expiraci dle nastavení `SESSION_LIFETIME` v minutách. V kombinaci s CSRF ochranou a validací emailové adresy tak Laravel poskytuje silný základ.

6.4 Validace vstupních dat

Validace vstupních dat je prvkem každé moderní webové aplikace, neboť zajišťuje, že systém pracuje pouze se smysluplnými informacemi. V projektu probíhá validace dat zejména na serverové straně, kde slouží k ochraně integrity databáze a k prevenci chyb způsobených uživatelskými vstupy.

Jako příklad lze uvést validaci dat při tvorbě a editaci poboček, která je realizována ve třídě `FacilityController`. Zde se využívá nástroj Laravel Validator, který umožňuje definovat sadu validačních pravidel a následně ověřit, zda příchozí data těmto pravidlům odpovídají.

```
$validated = $request->validate([
    'name' => 'required|string|max:255|unique:facilities,name,' .
    $facility->id,
    'color' => 'required|string|max:7|regex:/^[a-zA-F0-9]{6}$/ ',
    'is_main' => 'boolean'
]);
```

6.5 Omezení a návrhy na rozšíření zabezpečení

Aplikace má pevný základ zabezpečení, postavený na standardech Laravelu. Pro budoucí verze systému lze uvažovat o doplnění následujících bezpečnostních prvků:

- Dvoufaktorová autentizace (2FA) – např. pomocí TOTP nebo SMS.
- Audit log – zaznamenávání citlivých operací (např. mazání směn).
- Rate limiting a throttling – omezení počtu požadavků z jedné IP pro API i web.
- Správa přístupových tokenů a jejich expirační doby – zejména pro mobilní přístup nebo rozhraní třetích stran.
- CAPTCHA ochrana formulářů – především při registraci nebo obnově hesla.
- Logování podezřelých akcí – např. neúspěšných přihlášení, změn hesel

7 ZÁVĚR A ZHODNOCENÍ

V závěrečné kapitole této bakalářské práce jsou shrnuty dosažené výsledky vytvořené aplikace a naznačeny možné směry jejího budoucího rozvoje. Následující podkapitoly poskytují souhrn splněných úkolů, návrhy na rozšíření aplikace a vizuální ukázky výsledného řešení.

7.1 Shrnutí výsledků

V rámci této bakalářské práce byla navržena a implementována open source webová aplikace pro plánování směn v malých podnicích. Byla využita architektura klient-server s důrazem na přehlednost, bezpečnost a snadnou rozšiřitelnost. Implementace zahrnovala návrh databázové struktury, uživatelského rozhraní, systému správy uživatelů, směn a poboček, stejně jako zabezpečení proti běžným typům útoků. Součástí projektu byl rovněž vytvořen podrobný *README.md* soubor, který obsahuje návod k nasazení aplikace na virtuální server, aby bylo zajištěno hladké uvedení do provozu.

7.2 Možné rozšíření aplikace

Ačkoliv současná verze aplikace splňuje stanovené funkční a nefunkční požadavky, nabízí se řada možností, jak ji do budoucna rozšířit a dále zvýšit její praktickou využitelnost. Níže jsou uvedeny:

1. Jednou z perspektivních oblastí je implementace kontroly konfliktních směn mezi pobočkami. V praxi totiž může nastat situace, kdy je jeden zaměstnanec nedopatřením naplánován na směnu ve dvou různých provozovnách ve stejný čas, nebo s příliš krátkým časovým odstupem, který neumožňuje fyzický přesun. Automatizovaná kontrola těchto konfliktů by předešla podobným chybám, které mohou vzniknout při manuálním plánování, a přispěla by k vyšší efektivitě sestavovaných rozpisů směn.
2. Dalším rozšířením by mohlo být zavedení podmínek zákoníku práce. Systém by například dokázal upozornit administrátory na případy, kdy by plánované směny překračovaly maximální povolený počet hodin, neumožňovaly povinný odpočinek mezi směnami nebo jinak narušovaly pracovněprávní limity. Tento prvek by významně usnadnil dodržování legislativy a zjednodušil práci zaměstnavatelů.

3. Dále je možné doplnit evidenci přestávek přímo v rámci plánovaných směn, což by uživatelům umožnilo lépe plánovat nejen začátky a konce pracovních úseků, ale také jejich rozvržení uvnitř směny.
4. Z hlediska technického vývoje by aplikace mohla těžit z vylepšení responzivního přístupu, tedy optimalizace uživatelského rozhraní pro různá zařízení – od počítačů přes tablety až po mobilní telefony.
5. Další z možných směrů technického rozšíření je zlepšení konstrukce JavaScriptového kódu. Současná implementace využívá základní přístupy, které by bylo možné dále optimalizovat tak, aby byl kód přehlednější, udržitelnější a snadněji rozšiřitelný. V tomto kontextu se nabízí také přechod na balíček *Laravel Livewire*, což je moderní knihovna, která umožňuje tvorbu interaktivních uživatelských rozhraní bez nutnosti psát rozsáhlý vlastní JavaScript. Livewire umožňuje vytvářet dynamické komponenty, které komunikují se serverem přes AJAX, aniž by bylo potřeba se hluboce ponořit do JavaScriptu či frameworků jako Vue.js, Angular.
6. Dalším velmi perspektivním rozšířením je automatické generování rozpisů směn. V budoucnu by aplikace mohla obsahovat algoritmus, který na základě dostupnosti zaměstnanců, jejich preferencí a pravidel nastavených administrátorem automaticky navrhne optimální rozvrh směn. Takové automatické plánování by výrazně ušetřilo čas potřebný k manuální přípravě rozpisů a zároveň by mohlo zohlednit složité podmínky a omezení.
7. Ještě pokročilejší možností je implementace umělé inteligence (AI), která by se na základě historických dat mohla rozpoznávat vzorce a potřeby jednotlivých poboček a dokázala by plánovat směny s vyšší přesností a efektivitou. AI by mohla doporučovat optimální rozložení směn v závislosti na očekávané zátěži provozovny, předvídat potřebu personálu podle sezónnosti.

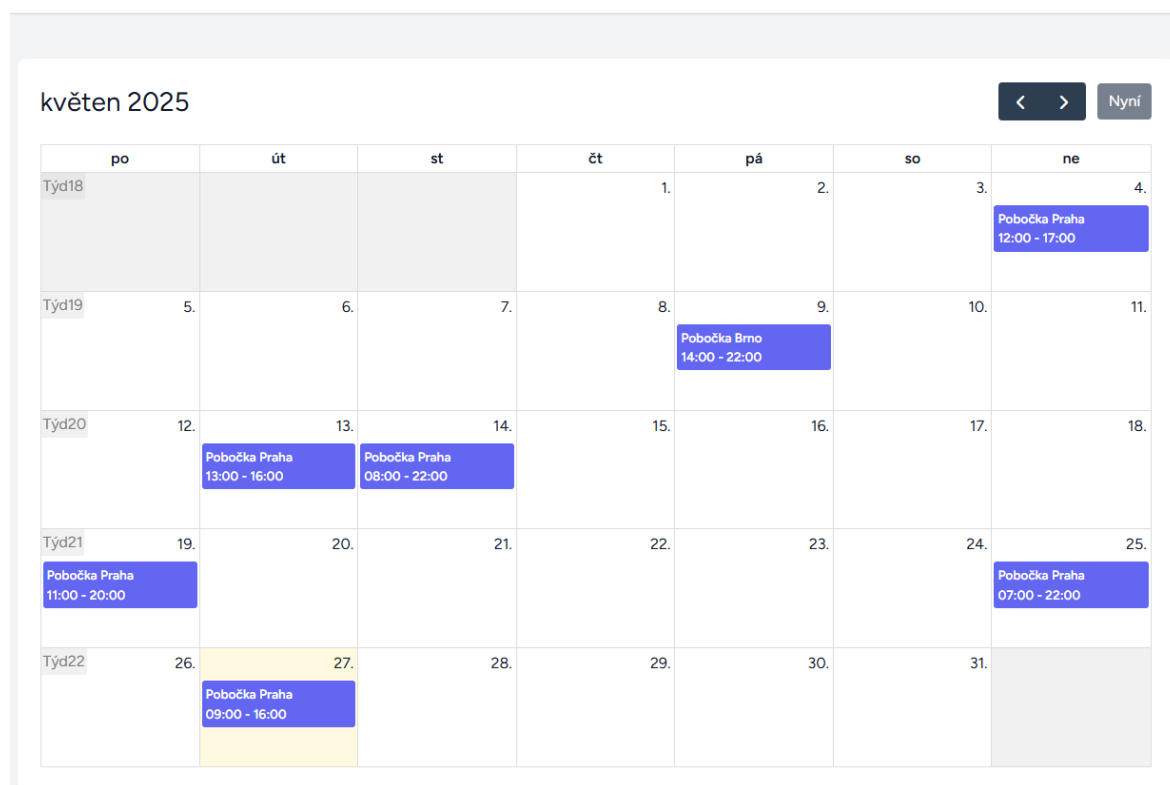
Celkově lze shrnout, že aplikace má díky své otevřené architektuře značný potenciál pro budoucí rozšiřování funkcionality, a to jak po stránce legislativní kompatibility, tak z hlediska uživatelského komfortu a provozní efektivity.

7.3 Ukázky uživatelské rozhraní

Pro lepší ilustraci výsledné podoby aplikace jsou v této kapitole zařazeny vybrané screenshoty, které demonstrují hlavní funkce a vzhled uživatelského rozhraní. Cílem této části je vizuálně doplnit popis implementace a ukázat praktické výsledky práce.

Je důležité poznamenat, že finální podoba aplikace se částečně liší od původního návrhu wireframe, který je popsán v kapitole 4.2. Tyto rozdíly vznikly na základě zkušeností z implementace a testování uživatelského rozhraní, aby lépe odpovídalo praktickým potřebám.

Můj kalendář



Obrázek 20 - Hlavní dashboard s přehledem směn

(zdroj: vlastní)

Plánování dostupnost

Uložit dostupnost

●
 Dostupný
celý den

⊘
 Nedostupný
specifický
čas

⚙
 Dovolená

🕒
 Dostupný
specifický
čas

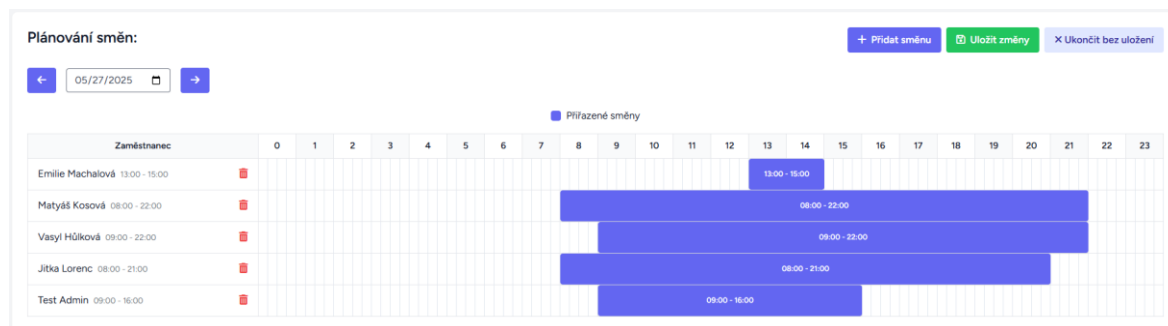
květen 2025

<
>
Nyní

	po	út	st	čt	pá	so	ne
Týd18				08:00 - 16:00	Celý den		
Týd19	5.	6.	08:00 - 16:00	08:00 - 16:00	Dovolena	10.	11.
Týd20	12.	08:00 - 16:00	Celý den	Celý den	Celý den	17.	18.
Týd21	19.	20.	21.	22.	23.	24.	25.
Týd22	26.		28.	29.	30.	31.	

Obrázek 21 - Stránka pro plánování dostupnosti

(zdroj: vlastní)



Obrázek 22 - Vizuální časová osa obsazenosti směny

(zdroj: vlastní)

Nastavení směny



Zaměstnanec

Vyberte zaměstnance



Vyberte zaměstnance

Ondřej Masný

Denisa Menšíková (08:00 - 16:00)

Simona Havelová (08:00 - 16:00)

Test Manager (Dovolená)

Věra Klein (Žádná dostupnost)

Eliška Juřicová (Žádná dostupnost)

Štěpán Kratochvíl (Žádná dostupnost)

Olga Záruba (Žádná dostupnost)

Test User (Žádná dostupnost)

Ranní (6-14)

Odpolední (14-22)

Noční (22-6)

Standardní (8-16:30)

Zrušit

Uložit

Obrázek 23 - Modální okno pro přiřazení směny dle požadavků

(zdroj: vlastní)

květen 2025																																		
Uživatel	1 Čt	2 Pá	3 So	4 Ne	5 Po	6 Út	7 St	8 Čt	9 Pá	10 So	11 Ne	12 Po	13 Út	14 St	15 Čt	16 Pá	17 So	18 Ne	19 Po	20 Út	21 St	22 Čt	23 Pá	24 So	25 Ne	26 Po	27 Út	28 St	29 Čt	30 Pá	31 So			
Emilie Machalová	-	-	-	-	-	-	-	-	08:00 17:00	-	-	14:00 16:00	13:00 21:00	-	-	-	-	-	-	-	-	14:00 15:00	08:00 22:00	-	-	-	-	13:00 15:00	-	-	-	-		
Věra Klein	-	-	-	06:00 16:00	-	-	-	-	11:00 20:00	-	-	09:00 15:00	13:00 16:00	06:00 16:00	-	-	-	-	-	-	-	-	-	-	-	12:00 17:00	-	-	-	-	-	-		
Denisa Menšíková	-	-	-	06:00 19:00	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	12:00 19:00	-	-	-	10:00 16:00	-	13:00 16:00	-	-	-	-	-	
Eliška Juřicová	-	-	14:00 20:00	-	-	-	-	-	-	-	-	14:00 18:00	-	-	-	-	-	-	-	12:00 17:00	-	-	-	-	-	-	-	-	-	-	-	-	-	
Štěpán Kratochvíl	-	-	-	-	09:00 15:00	-	-	-	-	-	-	-	07:00 16:00	13:00 16:00	-	-	-	-	-	14:00 21:00	-	-	-	-	-	-	-	-	-	-	-	-	-	
Simona Havelová	-	-	-	11:00 18:00	-	-	-	-	-	-	-	08:00 22:00	11:00 15:00	-	-	-	-	-	-	-	-	12:00 22:00	-	-	-	-	-	-	-	-	-	-	-	
Test User	-	-	-	-	-	-	-	-	-	-	-	-	06:00 15:00	-	-	-	-	-	-	-	-	-	-	-	-	08:00 20:00	-	-	-	-	-	-	-	
Test Manager	-	-	-	14:00 19:00	-	-	-	-	08:00 19:00	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	06:00 18:00	-	-	-	-	-	
Test Admin	-	-	-	12:00 17:00	-	-	-	-	-	-	-	-	13:00 16:00	08:00 22:00	-	-	-	-	-	11:00 20:00	-	-	-	-	-	07:00 22:00	-	09:00 16:00	-	-	-	-	-	

Obrázek 24 - Přehled směn zaměstnanců konkrétní pobočky

(zdroj: vlastní)

ZÁVĚR

Tato bakalářská práce byla zaměřena na návrh a implementaci open source webové aplikace pro plánování směn v malém podniku.

Úvod teoretické části práce se věnoval problematice klient-server architektury a obecně návrhu webových aplikací. Byly popsány jednotlivé architektury, principy HTTP komunikace a rozdíly mezi různými přístupy. Dále byly analyzovány dostupné frameworky, jako ASP.NET Core, Symfony a Express.js, přičemž hlavní část se soustředila na framework Laravel. Byla stručně zmíněna jeho historie, popsána jeho struktura.

Praktická část práce popsala klíčové funkce plánovače směn, především správu uživatelů, poboček, směn a oprávnění. Byly stanoveny funkční a nefunkční požadavky aplikace, navržena databázová struktura a uživatelské rozhraní. Dále byla podrobně popsána implementace jednotlivých modulů, včetně autentizace, autorizace a zabezpečení proti běžným webovým útokům. Součástí bylo také zhodnocení bezpečnosti aplikace a doporučení pro její případné budoucí rozšíření.

V poslední části práce bylo provedeno zhodnocení celého projektu, přičemž bylo konstatováno, že aplikace splnila stanovené cíle a poskytuje přehledné, bezpečné a rozšiřitelné prostředí pro plánování směn v malém podniku. Framework Laravel výrazně zjednodušil vývoj díky své přehledné struktuře, hotovým modulům. Výsledkem je moderní aplikace, která díky své open source povaze nabízí další možnosti úprav a rozšíření, například napojení na mobilní aplikace.

SEZNAM POUŽITÉ LITERATURY

- [1] ROUSE, Margaret. *Client/Server Architecture*. Online. 25.8.2020. Dostupné z: <https://www.techopedia.com/definition/438/clientserver-architecture>. [cit. 2025-02-28].
- [2] INTERVIEWBIT. *Client Server Architecture – Detailed Explanation*. Online. 3.6.2022. Dostupné z: <https://www.interviewbit.com/blog/client-server-architecture/>. [cit. 2025-02-28].
- [3] GEEKSFORGEEKS. *Difference between Client-Server and Peer-to-Peer Network*. Online. 20.8.2024. Dostupné z: <https://www.geeksforgeeks.org/difference-between-client-server-and-peer-to-peer-network/>. [cit. 2025-02-28].
- [4] RFWIRELESS-WORLD. *Client vs. Server: Understanding the Key Differences*. Online. Dostupné z: <https://www.rfwireless-world.com/terminology/networking-basics/client-vs-server-understanding-the-key-differences>. [cit. 2025-02-28].
- [5] AWATI, Rahul; CHAI, Wesley a FERGUSON, Kevin. *Definition What is HTTP and how does it work? Hypertext Transfer Protocol*. Online. 2025. Dostupné z: <https://www.techtarget.com/whatis/definition/HTTP-Hypertext-Transfer-Protocol>. [cit. 2025-02-28].
- [6] BODNÁR, Ján. *Úvod do frameworku Symfony: request-response cyklus*. Online. 2019. Dostupné z: <https://www.root.cz/clanky/uvod-do-frameworku-symfony-request-response-cyklus/>. [cit. 2025-03-12].
- [7] MDN CONTRIBUTORS. *HTTP request methods*. Online. 13.3.2025. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods>. [cit. 2025-03-25].
- [8] MDN CONTRIBUTORS. *Idempotent*. Online. 8.6.2023. Dostupné z: <https://developer.mozilla.org/en-US/docs/Glossary/Idempotent>. [cit. 2025-03-25].
- [9] PATEL, Ravi. *Exploring .NET 8 and ASP.NET Core: Features, Advantages, and Comparison with ASP.NET Framework*. Online. 2024. Dostupné z: <https://medium.com/@ravipatel.it/exploring-net-8-and-asp-net-core-features-advantages-and-comparison-with-asp-net-framework-a6b5743fc7be>. [cit. 2025-04-02].
- [10] UMBRACO. *What is ASP.NET Core?* Online. Dostupné z: <https://umbraco.com/knowledge-base/asp-dot-net-core/>. [cit. 2025-04-02].
- [11] KANEKAR, Kedar. *PHP vs ASP.NET: Which is Better?* Online. 2024. Dostupné z: https://www.turing.com/blog/php-vs-asp-net#a_s_p._n_e_t_pros_and_cons. [cit. 2025-04-02].
- [12] SMITH, Gracey. *What is Symfony? An Overview*. Online. 2025. Dostupné z: <https://www.theknowledgeacademy.com/blog/what-is-symfony/>. [cit. 2025-05-01].
- [13] PAREEK, Abhishek. *Symfony Web Development: Pros and Cons Explained*. Online. Dostupné z: <https://www.developers.dev/tech-talk/the-good-and-the-bad-of-symfony-web-development.html>. [cit. 2025-05-03].
- [14] KATARIYA, Jayanti. *Symfony vs Laravel: Choose the Best Framework for Your Project*. Online. Dostupné z: <https://www.moontechnolabs.com/blog/symfony-vs-laravel/>. [cit. 2025-05-24].

- [15] CODECADEMY TEAM. *Express.js Explained: What it is and How to Use it in Your JavaScript Project*. Online. Dostupné z: <https://www.codecademy.com/article/what-is-express-js>. [cit. 2025-05-04].
- [16] ZHIDKOV, Roman. *Node.js vs PHP: Which is Better?* Online. 13.02.2025. Dostupné z: <https://www.milesweb.com/blog/technology-hub/node-js-vs-php/>. [cit. 2025-05-04].
- [17] SHARMA, Itesh. *Node.js vs PHP: A Well Researched Comparison Guide*. Online. 2022. Dostupné z: <https://www.tatvasoft.com/outsourcing/2022/03/node-js-vs-php.html>. [cit. 2025-05-24].
- [18] TMA SOLUTIONS. *Software Development Frameworks: A Guide to 15 Popular Choices*. Online. 2024. Dostupné z: <https://www.tmasolutions.com/insights/software-development-frameworks>. [cit. 2025-05-20].
- [19] SOFTJOURN. *6 Things to Look for in a Framework*. Online. Dostupné z: <https://softjourn.com/insights/6-things-to-look-for-in-a-framework>. [cit. 2025-05-20].
- [20] OTWELL, Taylor. *Installation - Meet Laravel*. Online. Dostupné z: <https://laravel.com/docs/11.x>. [cit. 2025-04-27].
- [21] DEGNI, Riccardo. *Laravel: Celebrating 12 Years of Powering PHP Development*. Online. Dostupné z: <https://www.codemotion.com/magazine/languages/laravel-best-practices/>. [cit. 2025-04-27].
- [22] OTWELL, Taylor. *Directory Structure*. Online. Dostupné z: <https://laravel.com/docs/11.x/structure>. [cit. 2025-04-27].
- [23] HANIF, Fikri. *Understanding the MVC Architecture in Laravel: A Comprehensive Guide*. Online. Dostupné z: <https://fkrihnif.medium.com/understanding-the-mvc-architecture-in-laravel-a-comprehensive-guide-8f620cc139b6>. [cit. 2025-04-27].
- [24] GEEKSFORGEEKS. *Introduction to Laravel and MVC Framework*. Online. 2024, 20.3.2024. Dostupné z: <https://www.geeksforgeeks.org/introduction-to-laravel-and-mvc-framework/>. [cit. 2025-04-27].
- [25] STAUFFER, Matt. *Laravel: Up & Running: A Framework for Building Modern PHP Apps*. 3rd Edition. O'Reilly Media, 2023. ISBN 978-109815326-7.

SEZNAM OBRÁZKŮ

Obrázek 1 - Grafické srovnání client-server vs Peer to peer	16
Obrázek 2 - Wireframe úvodního kalendáře	37
Obrázek 3 - Wireframe menu	37
Obrázek 4 - Wireframe stránky dostupnosti	38
Obrázek 5 - Wireframe stránky editace směn	39
Obrázek 6 - Wireframe přehledu směn pro pobočku	39
Obrázek 7 - Struktura databáze	40
Obrázek 8 - Tabulka users	41
Obrázek 9 - Tabulka shifts	42
Obrázek 10 - Tabulka availabilities	43
Obrázek 11 - Tabulka facilities	44
Obrázek 12 - Tabulka facility_user	44
Obrázek 13 - Tabulka roles	45
Obrázek 14 - Tabulka role_has_permissions	45
Obrázek 15 - Tabulka permissions	46
Obrázek 16 - Tabulka model_has_roles	46
Obrázek 17 - Tabulka model_has_permissions	47
Obrázek 18 - Tabulka password_reset_tokens	47
Obrázek 19 - Tabulka sessions	48
Obrázek 20 - Hlavní dashboard s přehledem směn	67
Obrázek 21 - Stránka pro plánování dostupnosti	68
Obrázek 22 - Vizualní časová osa obsazenosti směny	68
Obrázek 23 - Modální okno pro přiřazení směny dle požadavků	69
Obrázek 24 - Přehled směn zaměstnanců konkrétní pobočky	69

SEZNAM TABULEK

Tabulka 1 - Výhody a nevýhody třívrstvé architektury	15
--	----

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

HTTP HyperText Transfer Protocol

HTML Hypertext Markup Language

URL Uniform Resource Locator

PHP Hypertext Preprocessor

JSON JavaScript Object Notation

JIT Just-In-Time

XSS Cross-Site Scripting

CSRF Cross-Site Request Forgery

IDE Integrated Development Environment

MVC Model-View-Controller

CLI Command Line Interface

SQL Structured Query Language

CSS Cascading Style Sheets

ORM Object-Relational Mapping

NPM Node Package Manager

AJAX Asynchronous JavaScript and XML

XML Extensible Markup Language

JWT JSON Web Token

API Application Programming Interface

AI Artificial Intelligence

SEZNAM PŘÍLOH

Příloha P I: Zdrojové kódy aplikace

PŘÍLOHA P I: ZDROJOVÉ KÓDY APLIKACE

Zdrojové kódy aplikace nahrané do systému IS/STAG