Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Systém plánování služeb pro nemocnice

Bakalářská práce

Autor: Martin Drábek
Studijní obor: ai3s-p

Vedoucí práce: Ing. Tomáš Nacházel, Ph.D.

Hradec Králové Duben 2025

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 19.4.2025

Martin Drábek

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Tomáši Nacházelovi, Ph.D. za metodické vedení práce, za jeho volný čas a rady, které mi při zpracování této práce věnoval. Také chci poděkovat své rodině a všem svým přátelům, kteří mě velmi podporovali během zpracování mé bakalářské práce.

Abstrakt

Tato bakalářská práce se zaměřuje na návrh a implementaci systému pro plánování služeb v nemocnicích.

V teoretické části je kladen důraz na obecný návrh informačního systému, jeho cíle a životní cyklus. Dále je popsán problém rozpisu služeb, známý jako *Nurse Rostering Problem* (NRP), a přehled algoritmů využívaných k jeho řešení. Součástí této práce je také přehled dostupného plánovacího softwaru na trhu.

Výsledkem této práce je návrh efektivního a adaptivního systému pro plánování služeb v nemocnicích, který bude schopen splnit administrativní i personální požadavky a přispět k optimalizaci provozu zdravotnických zařízení. Praktická část se věnuje výběru vhodných technologií pro vývoj aplikace, její implementaci a procesu testování.

Navržený systém byl úspěšně realizován a testován, přičemž splňuje stanovené požadavky na funkčnost, uživatelskou přívětivost a efektivitu plánování. Implementace poskytuje nástroj, který usnadňuje organizaci směn a umožňuje flexibilní přizpůsobení individuálním potřebám personálu. Výsledky této práce potvrzují, že vyvinutý systém představuje vhodné řešení pro optimalizaci plánování služeb.

Klíčová slova: Plánovací systém, Automatizace plánování, Plánování, Nemocniční systém

Abstract

Title: Service scheduling system for hospitals

This bachelor thesis focuses on the design and implementation of a service scheduling system for hospitals.

The theoretical part focuses on the general design of the information system, its objectives and life cycle. It also describes the service scheduling problem, known as

the Nurse Rostering Problem (NRP), and reviews the algorithms used to solve it. A review of the available scheduling software on the market is also included.

The result of this work is the design of an efficient and adaptive system for service scheduling in hospitals that will be able to meet administrative and staffing requirements and contribute to the optimization of the operation of healthcare facilities. The practical part deals with the selection of appropriate technologies for the development of the application, its implementation and the testing process.

The proposed system has been successfully implemented and tested, meeting the specified requirements for functionality, user-friendliness and scheduling efficiency. The implementation provides a tool that facilitates the organization of shifts and allows flexible adaptation to individual staff needs. The results of this work confirm that the developed system represents a suitable solution for optimizing service scheduling.

Key words: Scheduling system, Automation of Scheduling, Scheduling, Hospital System

Obsah

1 Úvod ... 1

2 Cíl a metodika práce ... 2

2.1 Využití nástrojů umělé inteligence ... 2

3 Návrh informačního systému ... 3

3.1 Cíle návrhu informačních systémů ... 3

3.2 Životní cyklus návrhu systému ... 4

3.3 Technologie pro webové aplikace ... 4

3.3.1 Architektura sytému ... 5

3.4 Frontend ... 6

3.4.1 HTML ... 7

3.4.2 CSS ... 7

3.4.3 JavaScript ... 8

3.4.4 Vue.js ... 8

3.4.5 AJAX ... 10

3.5 Backend ... 10

3.5.1 Node.js ... 11

3.5.2 RESTful API ... 12

3.5.3 JWT ... 12

3.6 Databáze MongoDB ... 13

3.7 Visual Studio Code ... 14

4 Plánování služeb ... 15

4.1 Způsoby řešení rozvrhu ... 15

4.1.1 Ruční řešení v tištěné formě ... 16

4.1.2 Ruční řešení pomocí softwaru ... 16

4.1.3 Poloautomatické řešení ... 16

4.1.4 Plně automatické řešení 17
5 Problém rozpisu služeb 17
5.1 Definice problému 18
6 Dostupný plánovací software na trhu 19
6.1 Matrickx 19
6.2 UKG software pro plánování zaměstnanců 20
6.3 Humanity 21
6.4 MakeShift 21
6.5 QGenda 22
7 Návrh webové aplikace systému 24
7.1 Vymezení požadavků na nemocniční systém 24
7.1.1 Řešení plánování služeb 24
7.1.2 Přívětivé UX/UI 25
7.1.3 Dodržování předpisů a zabezpečení dat 26
7.2 Uživatelské rozhraní 26
7.2.1 Drátové modely 26
7.2.2 Principy uživatelského dojmu 31
7.2.3 Grafický styl aplikace 32
7.3 Výběr technologií 33
7.3.1 Webová aplikace 33
7.3.2 Zdůvodnění výběru technologií 33
8 Implementace webové aplikace systému 34
8.1 Organizace projektu 34
8.1.1 Frontend struktura 35
8.1.2 Backend struktura 37
8.2 Implementace frontendu 39

8.2.1 Konfigurace projektu 40
8.2.2 Komunikace mezi komponentami 41
8.2.3 Autentizace uživatelů 43
8.2.4 Správa kalendáře a událostí 45
8.2.5 Automatizace plánování 54
8.2.6 Řešení skupin 58
8.3 Implementace backendu 61
8.3.1 Architektura backendu 61
8.3.2 Návrh databázových modelů 62
8.3.3 Operace s databází 63
8.3.4 Kontrolery 65
8.3.5 Middleware pro autentizaci a validaci 70
8.3.6 Směrování požadavků 72
8.4 Testování aplikace 74
9 Shrnutí a diskuse výsledků 75
10 Závěry a doporučení 76
11 Seznam použité literatury 77

Seznam obrázků

Obrázek 1: SPA architektura Zdroj: (4) 5
Obrázek 2: AJAX operace Zdroj: (51) přeloženo autorem 10
Obrázek 3: Drátový model úvodu Zdroj: autor 28
Obrázek 4: Drátový model přihlášení Zdroj: autor 29
Obrázek 5: Drátový model kalendáře Zdroj: autor 30
Obrázek 6: Drátový model skupin Zdroj: autor 31
Obrázek 7: Struktura frontend projektu Zdroj: autor 35
Obrázek 8: Stuktura backend projektu Zdroj: autor 37

Obrázek 9: Vývojový diagram automatizace plánování 1 Zdroj: autor 57
Obrázek 10: Vývojový diagram automatizace plánování 2 Zdroj: autor.....57

Seznam programových kódů

Programový kód 1: Definice cest v routeru Zdroj: autor.....42
Programový kód 2: Metoda pro přihlášení do systému Zdroj: autor 45
Programový kód 3: Metoda pro vytváření událostí Zdroj: autor 47
Programový kód 4: Metoda pro mazání událostí Zdroj: autor.....48
Programový kód 5: Metoda pro změnu událostí Zdroj: autor.....49
Programový kód 6: Metoda pro načtení událostí z backendu Zdroj: autor.....50
Programový kód 7: Pravidlo uzávěrky Zdroj: autor.....51
Programový kód 8: Omezení překrývání a sousedních událostí Zdroj: autor 52
Programový kód 9: Omezení počtu víkendů a svátků Zdroj: autor.....53
Programový kód 10: Algoritmus automatizace událostí Zdroj: autor 56
Programový kód 11: Metody pro správu skupin Zdroj: autor 59
Programový kód 12: Pravidlo pro kontrolu konfliktu událostí Zdroj: autor.....61
Programový kód 13: Metoda pro registraci uživatele Zdroj: autor.....64
Programový kód 14: Metoda pro přihlášení uživatele Zdroj: autor 65
Programový kód 15: Metoda pro uložení události do databáze Zdroj: autor 67
Programový kód 16: Metody pro smazání a změnu událostí Zdroj: autor.....68
Programový kód 17: Metoda pro získání všech událostí z databáze Zdroj: autor ...69
Programový kód 18: Metody pro verifikaci údajů a rolí Zdroj: autor 71

1 Úvod

Plánování služeb v nemocnicích představuje jeden z klíčových aspektů zajišťování efektivní a kvalitní zdravotní péče. Správně sestavený plán umožňuje optimalizovat využití personálních zdrojů, minimalizovat riziko vyhoření zaměstnanců a zajistit kontinuitu péče o pacienty. V současné době, kdy nemocnice čelí rostoucím nárokům na poskytování zdravotních služeb a zároveň omezeným zdrojům, se efektivní plánování stává nezbytností. Spolehlivý a uživatelsky přívětivý systém plánování oproti zastaralým metodám, včetně papírových rozpisů výrazně šetří čas, finanční prostředky a lidské zdroje.

Tato bakalářská práce se zabývá problematikou plánování služeb v nemocnicích a následně implementací konkrétního řešení prostřednictvím webové aplikace systému.

První část práce se věnuje návrhu a životnímu cyklu obecného informačního sytému. Dále jsou vysvětlena základní pravidla a pojmy související s plánováním služeb a běžné metody řešení rozvrhu. Následně je v práci nastíněn problém rozpisu služeb a jeho možná řešení pomocí jednotlivých algoritmů. Práce se také zaměřuje na analýzu vybraných plánovacích softwarů, které jsou využívány ve zdravotnických zařízeních.

Druhá část se soustředí na návrh a vývoj webové aplikace určené pro plánování služeb. Nejprve jsou definovány požadavky na vyvíjený systém a jeho odpovídající řešení. Následně jsou popsány principy uživatelského rozhraní, uživatelského dojmu a základy zabezpečení dat. Dále jsou představeny technologie a nástroje použité při vývoji aplikace. V dalších kapitolách je podrobně popsán proces vývoje frontendu a backendu. Závěrečná kapitola se zaměřuje na jednoduché principy testování aplikace.

2 Cíl a metodika práce

Cílem této bakalářské práce je poskytnout čtenářům přehled o problematice plánování a následně navrhnout a implementovat aplikaci, která umožní efektivní rozvržení služeb lékařů na základě zadaných parametrů a preferencí, přičemž budou dodržena všechna stanovená pravidla.

V teoretické části je nejprve popsán obecný návrh informačního systému, včetně definice problematiky plánování služeb, jeho omezení, správy a řešení rozvrhu. Dále se práce zabývá problematikou rozpisu služeb, přehledem algoritmů používaných k jeho řešení a analýzou dostupného plánovacího softwaru na trhu.

Praktická část se zaměřuje na vymezení požadavků na navrhovaný nemocniční systém, výběr vhodných technologií a popis použitého softwaru. Jsou zde představeny zvolené frameworky a nástroje pro backend, frontend a databázovou vrstvu. Následně je popsán celkový návrh webové aplikace a její implementace. Práce je zakončena závěrečným shrnutím dosažených výsledků.

2.1 Využití nástrojů umělé inteligence

Při tvorbě této bakalářské práce byly využity nástroje umělé inteligence v souladu s dohodou s vedoucím práce a etickými zásadami akademického psaní. Konkrétně byly použity nástroje pro opravu gramatiky a překlepů, přičemž veškeré úpravy byly následně ručně ověřeny a případně upraveny.

Pro překlad některých odborných textů z anglického jazyka do češtiny byl rovněž použit nástroj umělé inteligence, přeložené pasáže byly důkladně zkontrolovány a upraveny s ohledem na kontext práce. Veškeré informace získané za pomoci AI byly podrobeny kritickému zkoumání a ověřeny na základě relevantních zdrojů.

3 Návrh informačního systému

Návrh informačních systémů je proces definování architektury, komponent, modulů, rozhraní a dat pro systém, který má splňovat zadané požadavky. Zahrnuje převedení uživatelských požadavků do podrobného plánu, který je vodítkem pro fázi implementace. Cílem je vytvořit dobře organizovanou a efektivní strukturu, která splňuje zamýšlený účel a zároveň zohledňuje faktory, jako je škálovatelnost, udržovatelnost a výkonnost. Tento proces se řídí určitými pravidly a doporučeními. Podle těchto pravidel a doporučení se může tento proces nazvat metodikou. Metodika určuje, kdo by měl co, kdy a proč při vývoji a provozu informačních systémů dělat. Zde je také vhodné od sebe dokázat odlišit následující výrazy. (1)

- Metodika – identifikuje soubor principů a konceptů, které pokrývají celý cyklus.
- Metoda – pojednává o tom, co dělat, kdy a jak používat různé techniky.
- Nástroj – využívá se jako nástroj nebo pomůcka pro vývojáře
- Technika – definuje způsob, jak danou činnost provádět

3.1 Cíle návrhu informačních systémů

Jak uvádí autor článku (2), jeden z hlavních cílů, který se dodržuje při návrhu informačního systému, je mít systém praktický, aby byl zaměřen na daný soubor uživatelů, kterému odpovídá.

Systém by měl být navržen tak, aby byl přesný vůči téměř všem požadavkům, kolem kterých je navržen, ať už se jedná o funkční nebo nefunkční požadavky. Návrh systému by měl být úplný, tedy splňovat všechny požadavky uživatelů. Návrh systému musí být v souladu s pravidly, která jsou stanovena, aby nepřekračoval náklady na zdroje ani je nevyužíval nedostatečně, protože jak je známo, tak to povede k nízké propustnosti a delší době odezvy. Navržený systém by měl být po určitou dobu v blízkosti bezporuchového prostředí, aby se otestovala jeho míra spolehlivosti. Důležité je také dbát na dostatečnou optimalizaci, protože výpočetní čas a úložný prostor jsou klíčové pro správnou funkci jednotlivých komponent systému. Návrh systému by měl být přizpůsobitelný v čase a podle různých uživatelských potřeb zákazníků, o kterých se ví, že se budou v čase měnit. (3)

3.2 Životní cyklus návrhu systému

Článek (2) popisuje životní cyklus návrhu systému. Jedná se o komplexní proces, který popisuje kroky spojené s návrhem a vývojem daného systému, ať už se jedná o softwarovou aplikaci, hardwarové řešení nebo integrovaný systém kombinující oba způsoby. Zahrnuje řadu fází, které vedou inženýry při vytváření systému, jenž odpovídá potřebám uživatele a cílům organizace. Cílem je zajistit, aby byl konečný produkt spolehlivý, škálovatelný a udržovatelný.

Fáze životního cyklu návrhu systému jsou následující:

- Plánování – zahrnuje definování cílů a rozsah projektu, stanovení časového harmonogramu a rozpočtu.
- Studie proveditelnosti – hodnotí se, zda je projekt realizovatelný z technického, ekonomického a právního hlediska.
- Návrh systému – zaměřuje se na vytváření detailních specifikací systému
- Implementace – v této fázi se návrh systému přemění na skutečný kód, začne vývoj jednotlivých komponent.
- Testování – zahrnuje důkladné testování systému, aby se zjistily a odstranily chyby i nedostatky.
- Nasazení – jedná se o fázi nasazení do produkčního prostředí, tedy instalace systému, migrace dat, školení uživatelů a uvedení systému do provozu.
- Údržba a podpora – tato fáze zahrnuje opravu chyb, aktualizace softwaru, zlepšování výkonu a přizpůsobení systému novým požadavkům uživatelů

3.3 Technologie pro webové aplikace

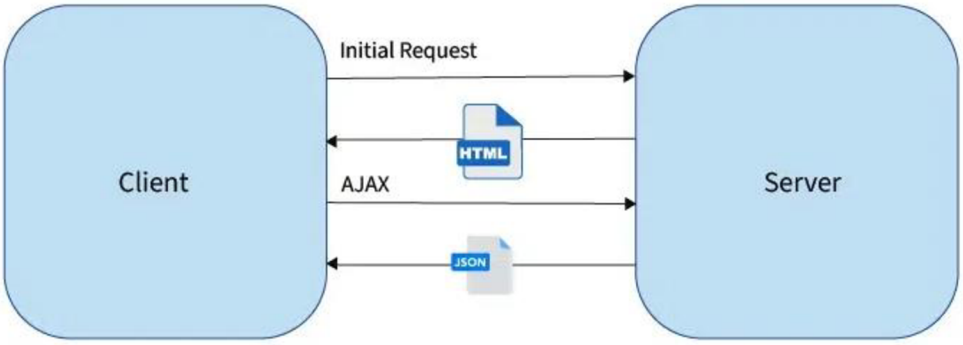
V této kapitole jsou podrobně popsány technologie používané při vývoji webových aplikací. Bude popsána architektura systému, která ukazuje propojení mezi jednotlivými vrstvami aplikace. Dále se kapitola zaměří na popis frontendu, backendu a jejich technologií, kde budou vysvětleny nástroje a frameworky použité pro vývoj uživatelského rozhraní a serverové logiky. A nakonec bude představeno databázové řešení a popis používaného vývojového prostředí.

3.3.1 Architektura sytému

Tato webová aplikace využívá architekturu Single Page Application (SPA) neboli jednostránková aplikace.

Jedná se o typ webové aplikace, která dynamicky načítá a aktualizuje obsah bez obnovování celé stránky. Na rozdíl od tradičních webových stránek, SPA využívají moderní technologie, které zlepšují uživatelský zážitek tím, že minimalizují přerušení a poskytují plynulejší rozhraní. Uživatelé mohou s aplikací komunikovat plynule, podobně jako při používání běžného softwaru pro počítače. Hlavní výhodou je eliminace opětovného načítání celé stránky, což má za následek citlivější a poutavější webové prostředí. (4) Toho je dosaženo tím, že prohlížeč získá všechny podstatné kódy HTML, JavaScript a CSS v jednom požadavku nebo aktualizuje potřebný obsah na základě akcí uživatele. Obrázek 1 vyobrazuje tento princip.

Když se na něco klikne ve SPA, odešlou se do prohlížeče pouze potřebné informace a prohlížeč je vykreslí. Tím se liší od tradičního načítání stránek, kdy server při každém kliknutí odešle do prohlížeče celou stránku.



Obrázek 1: SPA architektura
Zdroj: **(4)**

Rozdělují se na 3 typy vykreslování webových stránek.

Vykreslování na straně klienta:

Prohlížeč si nejprve vyžádá HTML ze serveru. Server pak rychle odpoví základním souborem HTML a propojenými styly/skripty. Uživatel nyní vidí prázdnou stránku nebo načítací okno, zatímco se provádí JavaScript. Aplikace načte data, vytvoří pohledy a vloží je do DOM.

Vykreslování na straně klienta může být u základních webových stránek pomalejší, protože využívá mnoho prostředků zařízení. Přesto je vhodné pro vytížené weby, protože uživatelům urychluje práci.

Vykreslování na straně serveru:

Prohlížeč nejprve požádá server o soubor HTML. Nyní server shromáždí potřebná data, sestaví SPA a okamžitě vytvoří soubor HTML. Nyní uživatel vidí obsah připravený k použití. Struktura jednostránkové aplikace přidává události, vytváří virtuální DOM a připravuje ostatní komponenty. Nyní je aplikace připravena k použití.

Volba vykreslování na straně serveru je ideální, aby se dosáhlo svižného fungování aplikace, aniž by se zatěžoval prohlížeč uživatele, čímž je zajištěn optimální výkon aplikace.

Generátor statických stránek:

Prohlížeč prvně vyžaduje HTML, generátor rychle poskytne předpřipravené statické stránky. Server zobrazí uživatelům statickou stránku pro rychlé načtení. SPA ve stránce načítají data a provádějí dynamické změny stránky. Po přidání dat je SPA připravena pro plynulou interakci uživatele.

Generátor je skvělý pro rychlé a efektivní statické stránky, ale nemusí být ideální pro vysoce dynamické webové stránky, kde se často pracuje s měnícím dynamickým obsahem. Jejich síla spočívá hlavně ve statických stránkách, což odpovídá názvu.

3.4 Frontend

Front-end aplikace, (5) obecně známé jako rozhraní aplikace, je vrstva nebo prvek, který má uživatel možnost používat, vidět a komunikovat s ním prostřednictvím tlačítek, obrázků, interaktivních prvků, navigačních nabídek a textu. Jakmile je front end aplikace vyvinut, může k ní uživatel přistupovat. Po otevření front-endu aplikace začne uživatel okamžitě pracovat s písmy, barvami a rozmístění rozhraní. Tyto prvky vedou uživatele při navigaci v aplikaci. Důležité je také zajistit, aby aplikace byla vhodná pro mobilní zařízení a dobře fungovala na zařízeních různých velikostí a rozlišení obrazovky. (6) Do frontendu patří samozřejmě i uživatelský dojem (UX), který

se zaměřuje na to, aby aplikace byla intuitivní, přístupná a snadno ovladatelná pro uživatele.

Pomocí HTML, kaskádových stylů (CSS) a JavaScriptu mohou vývojáři začlenit efekty a stylistické předvolby.

3.4.1 HTML

HyperText Markup Language neboli HTML (7) je nezákladnějším stavebním prvkem webu, jedná se o samotnou kostru webové aplikace. Definuje význam a strukturu webového obsahu. „Hypertext“ označuje odkazy, které propojují webové stránky mezi sebou, a to buď v rámci jedné webové stránky, nebo mezi více webovými stránkami. Odkazy jsou základním aspektem webu.

Jazyk HTML používá „značky“ k anotaci textu, obrázků a dalšího obsahu pro zobrazení ve webovém prohlížeči. Značky HTML obsahují speciální „prvky“, mezi základní patří např. <head>, <body>, <title>, <footer>, <p>, <img>, , <nav>, <ul>, <li> a mnohé další.

Element HTML je oddělen od ostatního textu v dokumentu pomocí „značek“, které se skládají z názvu elementu obklopeného znaky '<' a '>'. Každý prvek má svůj začátek a konec, konec daného prvku se značí lomítkem, tedy následovně: <title>Nadpis</title>. V názvu prvku uvnitř značky se nerozlišují velká a malá písmena. To znamená, že může být zapsán velkými a malými písmeny nebo jejich kombinací. Například značku <title> lze zapsat jako <Title>, <TITLE> nebo jiným způsobem. Nicméně konvencí a doporučeným postupem je psát značky malými písmeny.

3.4.2 CSS

Cascading Style Sheets neboli CSS (8) je jazyk stylů používaný k popisu prezentace dokumentu napsaného v jazyce HTML nebo XML. CSS popisuje, jak mají být prvky zobrazeny na obrazovce. CSS patří mezi základní jazyky otevřeného webu a je standardizován ve všech webových prohlížečích podle specifikací W3C. Má zásadní význam pro úplné přizpůsobení webové stránky, protože umožňuje upravovat a měnit každý prvek obsahu, včetně velikosti písma, mezer a animací.

Jelikož se jedná o kaskádový jazyk, tak ve stylech funguje dědičnost, tedy pokud se pro buňku tabulky nastaví modrá barva písma, tak tuto barvu automaticky dostanou i všechny odstavce textu v této buňce. Stejný CSS styl lze aplikovat na více stránek a prvků naráz, takže pokud se změní jeden řádek stylu, tak změny se samy projeví na všech stránkách, kde se daný styl využívá. CSS je možné vložit jak interně, tak externě, běžně se styly vkládají externě pro přehlednost a zjednodušení upravování, pomocí referenčního prvku link uvnitř sekce head:

```
'<link rel="stylesheet" href="style.css"> '
```

3.4.3 JavaScript

Jedná se o odlehčený interpretovaný programovací jazyk s prvotřídními funkcemi. (9) Nejznámější je jako skriptovací jazyk pro webové stránky, ale používá ho i mnoho jiných prostředí než prohlížeč, například Node.js, Apache CouchDB a Adobe Acrobat.

Dynamické možnosti jazyka JavaScript zahrnují konstrukci objektů za běhu, seznamy parametrů proměnných, proměnné funkcí, dynamické vytváření skriptů, introspekci objektů a obnovu zdrojového kódu. JavaScript není interpretovaná Java, oba programovací jazyky mají velmi odlišnou syntaxi, sémantiku a použití. Pro přidání JavaScriptu je potřeba vložit do HTML stránky prvek <script>, který umožní vytvářet funkce a interní logiku pro danou stránku.

JavaScript (10) je využíván jak na klientské, tak na serverové straně. Na straně klienta JavaScript primárně zajišťuje zobrazení uživatelského rozhraní, zatímco na serveru podporuje provoz složitých aplikací propojených s databázemi. JavaScript využívá koncept funkcionálního paradigmatu, který dovoluje ukládat funkce do obyčejných proměnných. Tato na první pohled jednoduchá vlastnost potom umožňuje předávat funkce jako argumenty jiným funkcím, to se nazývá callback, nebo dokonce použít funkci jako základ pro vytvoření objektu. Objektově orientované programování získává tímto zcela nový rozměr. Funguje také prototypové dědění, takže jeden objekt může sloužit jako vzor pro jiný objekt.

3.4.4 Vue.js

Jedná se o framework JavaScriptu (11) pro vytváření uživatelských rozhraní. Staví na standardních jazycích HTML, CSS a JavaScript a poskytuje deklarativní programovací

model založený na komponentech, který pomáhá efektivně vyvíjet uživatelská rozhraní libovolné složitosti.

Základní funkce, které využívá jsou deklarativní vykreslování, Vue rozšiřuje standardní HTML o syntaxi šablon, která umožňuje deklarativně popsat výstup HTML na základě stavu JavaScriptu. A reaktivita, pomocí které Vue automaticky sleduje změny stavu JavaScriptu a při změnách efektivně aktualizuje DOM. Vue poskytuje rozhraní pro změnu toho, co se na stránce zobrazí, na základě aktuální cesty k adrese URL bez ohledu na to, jak byla změněna. Použití front-end routeru navíc umožňuje záměrný přechod cesty prohlížeče, když dojde k určitým událostem prohlížeče jako kliknutí na tlačítka nebo odkazy.

Ve většině projektů Vue s nástrojem build-tool lze vytvářet komponenty Vue pomocí formátu souboru podobného HTML, který se nazývá Single-File Component. SFC Vue, jak název napovídá, zapouzdřuje logiku komponenty, šablonu a styly (CSS) do jediného souboru. Komponenty Vue lze vytvářet ve dvou různých stylech API neboli aplikační programové rozhraní: Options API a Composition API.

Pomocí rozhraní API Options se definuje logika komponenty použitím objektu možností, jako jsou data, metody a připojené prvky. Vlastnosti definované Options jsou na něm vystaveny uvnitř funkce, která ukazuje na instanci komponenty.

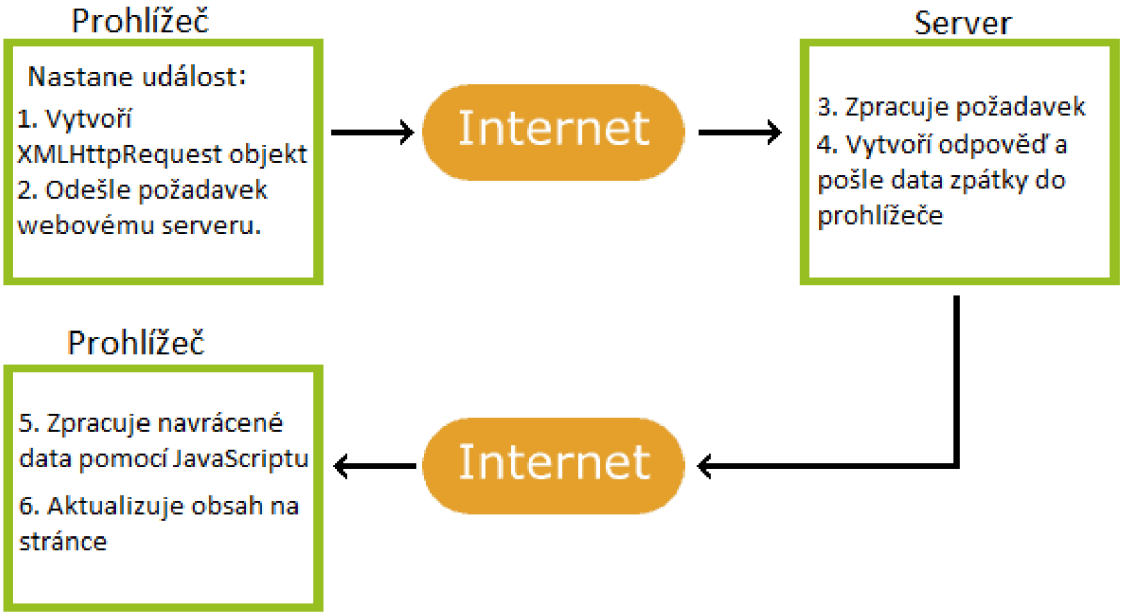
Zatímco pomocí rozhraní Composition API se definuje logika komponenty použitím importovaných funkcí API. Composition API se typicky používá jako <script setup>. Atribut setup je nápověda, díky níž Vue provádí transformace při kompilaci, které umožňují používat Composition API s menším množstvím šablon. Například importy a proměnné nebo funkce nejvyšší úrovně deklarované v <script setup> jsou přímo použitelné v šabloně.

Oba styly rozhraní API plně pokrývají běžné případy použití. Jedná se o různá rozhraní, která jsou poháněna stejným základním systémem. Ve skutečnosti je rozhraní API Options implementováno nad rozhraním API Composition. Základní koncepty a znalosti o rozhraní Vue jsou společné pro oba styly. V rámci projektu byl zvolen styl rozhraní API Options.

3.4.5 AJAX

Asynchronní JavaScript a XML (AJAX) (12) je technika vývoje webových stránek, která umožňuje webovým stránkám komunikovat s webovým serverem asynchronně, což znamená, že mohou odesílat a přijímat data na pozadí, aniž by zasahovaly do interakce uživatele na stránce. Tento dynamický přístup k vývoji webových stránek změnil způsob, jakým s nimi pracujeme, a učinil je interaktivnějšími a uživatelsky přívětivějšími.

Srdcem AJAXu je objekt XMLHttpRequest (XHR) v jazyce JavaScript, který slouží jako motor pro asynchronní komunikaci mezi klientem a serverem. Základní syntaxe požadavku AJAX (viz. Obrázek 2) zahrnuje vytvoření objektu XHR, zadání metody požadavku (například GET nebo POST), adresy URL a asynchronní zpracování odpovědi. Pomocí AJAXu je možné čtení dat z webového serveru po načtení stránky, aktualizace webové stránky bez jejího opětovného načtení a odesílání dat na webový server na pozadí.



Obrázek 2: AJAX operace
Zdroj: **(51)** přeloženo autorem

3.5 Backend

Backend (13) je serverová část webových stránek. Spravuje data a zajišťuje, aby vše na frontendové straně fungovalo správně. Jedná se o část aplikace, kde se nachází server

s databází a kde je vytvořena logika pro provádění operací. Zahrnuje hlavní vlastnosti a funkce aplikace na serveru. Uživatelé nevidí backend ani s ním přímo neinteragují; jedná se o funkce v pozadí. K částem a vlastnostem, které vyvinuli návrháři backendu, mají uživatelé nepřímý přístup prostřednictvím aplikace front-endu. Činnosti, jako je psaní rozhraní API, vytváření knihoven a práce se systémovými komponentami bez uživatelského rozhraní nebo dokonce systémy vědeckého programování, jsou také zahrnuty do backendu.

3.5.1 Node.js

Node.js (13) je multiplatformní běhové prostředí s otevřeným zdrojovým kódem pro spouštění kódu JavaScriptu mimo prohlížeč, které se používá k vytváření interaktivních a dynamických webových stránek. Nejedná se o framework ani programovací jazyk. Node.js se často používá pro vytváření back-endových služeb, jako jsou API, například webové aplikace nebo mobilní aplikace. Dodává se s bohatou knihovnou modulů prostřednictvím npm (Node Package Manager), pomocí které je možné si instalovat, aktualizovat a spravovat knihovny i balíčky, které obsahují již předpřipravené části kódu.

Jak uvádí příspěvek, (14) uživatelé Node.js se nemusí obávat zablokování procesu, protože neexistují žádné zámky. Téměř žádná funkce v Node.js neprovádí přímo I/O neboli vstup/výstup, takže proces nikdy neblokuje, s výjimkou případů, kdy se I/O provádí pomocí synchronních metod standardní knihovny Node.js. Jelikož nic neblokuje, je velmi výhodné pomocí Node.js vyvíjet škálovatelné systémy.

Volání start-the-event-loop zde neexistuje. Node.js jednoduše vstoupí do smyčky událostí po provedení vstupního skriptu. Node.js ukončí smyčku událostí, když už není třeba provádět žádná další zpětná volání.

3.5.1.1 Express.js

Express.js (15) je minimální a flexibilní framework pro webové aplikace Node.js, který nabízí řadu funkcí pro snadné vytváření webových a mobilních aplikací. Zjednodušuje vývoj aplikací na straně serveru tím, že nabízí snadno použitelné rozhraní API pro směrování, middleware a nástroje HTTP. I když je samotný Express poměrně minimalistický, vývojáři vytvořili kompatibilní middleware balíčky, které řeší téměř

jakýkoli problém při vývoji webu. Existují knihovny pro práci se soubory cookie, relacemi, přihlašováním uživatelů, parametry URL, daty POST, bezpečnostními hlavičkami a mnoha dalšími.

3.5.2 RESTful API

Přenos reprezentačního stavu neboli REST (16) je architektonický styl rozhraní pro programování aplikací, které používá požadavky HTTP pro přístup k datům a jejich užití. Tato data lze použít pro datové typy POST, GET, PUT, a DELETE, což se týká operací CRUD čili vytváření, čtení, aktualizace a mazání souvisejících se zdroji.

API je kód, který umožňuje dvěma softwarovým programům vzájemně komunikovat. Návrh rozhraní API určuje správný způsob, jak má vývojář napsat program neboli klienta, který používá rozhraní API k vyžádání služeb od jiné aplikace, obvykle serveru.

Technologie REST je obecně upřednostňována před jinými podobnými technologiemi. Je to proto, že REST využívá menší šířku pásma, takže je efektivnější při používání internetu.

Hlavními třemi prvky RESTful API jsou klient, server a zdroj. Klient je softwarový kód nebo aplikace, která požaduje zdroj od serveru. Server je softwarový kód nebo aplikace, která řídí zdroje a odpovídá na požadavky klienta na zdroj. Zdroje jsou jakákoli data nebo obsah, například text, video a obrázky, které server ovládá a zpřístupňuje v reakci na požadavky klienta. Reprezentace zdrojů musí být samo popisná, klient nemusí vědět, zda je zdroj zaměstnanec nebo zařízení. Měl by jednat na základě typu média přiřazeného ke zdroji.

Jeho důležitou vlastností je bezstavovost, která vyžaduje, aby každý požadavek klienta na server obsahoval všechny informace nezbytné k pochopení a dokončení požadavku. Server nemůže využívat žádné dříve uložené kontextové informace na serveru. Z tohoto důvodu musí klientská aplikace zcela uchovávat stav relace.

3.5.3 JWT

JSON Web Token (JWT) (17) je otevřený standard, který definuje kompaktní a samostatný způsob bezpečného přenosu informací mezi stranami ve formě objektu JSON. Tyto informace lze ověřovat a důvěřovat jim, protože jsou digitálně podepsané.

Protokoly JWT lze podepsat pomocí tajného klíče nebo dvojice veřejného a soukromého klíče.

Podepsané tokeny mohou ověřit integritu tvrzení v nich obsažených, zatímco šifrované tokeny tato tvrzení před ostatními stranami skrývají. Pokud jsou tokeny podepsány pomocí dvojice veřejného soukromého klíče, podpis také osvědčuje, že je podepsala pouze strana, která je držitelem soukromého klíče.

V tomto projektu je JWT využíván k autorizaci, což je nejčastější scénář použití JWT. Jakmile je uživatel přihlášen, každý další požadavek bude obsahovat token JWT a umožní mu přístup k trasám, službám a prostředkům, které jsou s tímto tokenem povoleny.

Struktura JWT se skládá ze tří části, které jsou odděleny tečkami, jedná se o hlavičku, náklad a podpis. Typické JWT vypadá následovně: xxxxx.yyyyy.zzzzz

Hlavička se obvykle skládá ze dvou částí: typu tokenu, což je JWT, a použitého podepisovacího algoritmu. Druhou částí tokenu je náklad, který obsahuje deklarace. Deklarace jsou prohlášení o entitě a další údaje. Existují tři typy deklarací: registrované, veřejné a soukromé deklarace. Poslední částí tokenu je podpis, který se vytváří tak, že se musí vzít zakódovaná hlavička, zakódovaný náklad, tajný kód, algoritmus uvedený v hlavičce a podepsat jej. Podpis slouží k ověření, že zpráva nebyla po cestě změněna, a v případě tokenů podepsaných soukromým klíčem může také ověřit, že odesílatel JWT je tím, za koho se vydává.

3.6 Databáze MongoDB

Jedná se o NoSQL databázi s otevřeným kódem, která poskytuje výkonné a flexibilní řešení pro zpracování moderních datových potřeb (18). Nabízí dynamický návrh schématu, který vývojářům umožňuje ukládat a spravovat data způsobem, který se bez problémů přizpůsobí současným požadavkům aplikací. Na rozdíl od tradičních relačních databází umožňuje dokumentově orientovaná architektura MongoDB větší agilitu a škálovatelnost, takže je preferovanou volbou pro podniky a vývojáře, kteří chtějí zpracovávat velké objemy nestrukturovaných nebo částečně strukturovaných dat.

Řadí se do kategorie databází NoSQL neboli nejen SQL, protože ukládání a načítání dat v MongoDB neprobíhá formou tabulek. Prostředí MongoDB poskytuje server, který je možné spustit a pak na něm vytvořit více databází pomocí MongoDB. Vzhledem k tomu, že se jedná o databázi NoSQL, jsou data uložena v kolekcích a dokumentech. Databáze obsahuje kolekce stejně jako databáze MYSQL obsahuje tabulky. Je možné vytvořit více databází a více kolekcí. Uvnitř kolekce se nachází dokumenty, které mají maximální velikost 16 MB.

Tyto dokumenty obsahují data, které jsou potřeba uložit do databáze MongoDB. Jedna kolekce může obsahovat více dokumentů a ty jsou bez schématu, což znamená, že není nutné, aby jeden dokument byl podobný druhému. Dokumenty se vytvářejí pomocí polí. Pole jsou dvojice klíč-hodnota v dokumentech, stejně jako sloupce v relační databázi. Hodnota polí může být libovolného datového typu, například double, string, boolean atd.

3.7 Visual Studio Code

Visual Studio Code, (19) často označovaný zkratkou VS Code je jedním z oblíbených textových editorů s otevřeným zdrojovým kódem. VS Code je jedním z renomovaných programů společnosti Microsoft, je zdarma a je kompatibilní se systémy Windows, Linux a MacOS.

Instalace VS Code je snadná a nezabere příliš mnoho místa, vzhledem k tomu, že tato platforma je vybavena robustními a aktuálními funkcemi, nelze její nízkou spotřebu považovat za chybu. Má integrovanou podporu pro JavaScript, Node.js a bohatý ekosystém rozšíření pro další programovací jazyky jako jsou C++, Java, Python a mnoho dalších.

Jelikož není možné, aby editor kódu poskytoval řešení pro všechny možné problémy, podporuje VS Code rozšíření. Rozšíření jsou funkce třetích stran, které mohou programátoři přidávat, aby zvýšili kvalitu, funkčnost a vzhled kódů. Prostřednictvím rozšíření VS mohou programátoři také přidávat jazyky, ladicí programy a další nástroje. Po instalaci se rozšíření přidá do editoru a je připraveno k použití. Pokud konkrétní rozšíření už není potřeba, je možné ho jednoduše odinstalovat prostřednictvím tlačítka pro správu.

4 Plánování služeb

Na začátku celé problematiky je důležité si definovat několik pravidel a základních pojmů, které jsou spojeny s návrhem plánovacího systému, aby bylo všemu plně porozuměno. (20)

Plánování lze obecně definovat jako přidělování zdrojů objektům tak, aby byly celkové náklady použitých zdrojů co nejvíce minimalizovány. Při plánování je důležité dosáhnout optimálního výsledku a splnit všechny zadaná pravidla a podmínky. Každá akce v rozvrhu má danou dobu trvání. Pokud se jedná o tento případ systému plánování služeb pro nemocnice, kde se sloužící lékaři hlásí na noční služby, tak mají jednotlivé akce v rozvrhu jednoznačně stanovené body začátku a konce. Zdroje jsou tedy vyjádřeny sloužícími lékaři, kterým se může přiřadit maximálně jedna směna na jeden den. Během tvorby rozvrhu služeb je potřeba brát v potaz určitá pravidla a omezení. Pravidla například zahrnují maximální počet po sobě jdoucích odpracovaných hodin nebo celkový počet odpracovaných hodin za měsíc. Tato pravidla se většinou řídí dle zákoníku práce (21), a je tedy třeba je bez výhrad dodržovat.

4.1 Způsoby řešení rozvrhu

Podle článku (22), existuje několik způsobů řešení problému plánování směn. Tyto metody lze rozdělit na:

- Ruční v tištěné formě
- Ruční pomocí softwaru na počítači
- Poloautomatický
- Plně automatický, může využívat umělou inteligenci

Všechny uvedené metody mají své určité výhody i nevýhody. Obecně je dáno, že čím více eviduje firma zaměstnanců ve svém rozvrhu, tím více je sestavení a správa finálního rozvrhu komplikovanější. Ruční formy nejsou obvykle velmi vhodné, v případě softwaru na počítači to při manuálním sestavováním vyžaduje mnoho práce a snadno se může vyskytnout lidská chyba. Na druhé straně jsou poloautomatické a plně automatické, které jsou již v dnešní době velmi rozsáhlé a využívané. Hojně se také pro plně automatické řešení využívá umělá inteligence, která se poslední dobou neustálé rozvíjí.

4.1.1 Ruční řešení v tištěné formě

Jak uvádí autor článku (22), ruční řešení návrhu rozvrhu se v dnešní době občas stále používá. Jedná se například o plánování služeb v terénu na nejjednodušší úrovni, které může rychle začít selhávat, když se zintenzivní objem, složitost nebo volatilita. Často se stává, že je denní rozvrh nucen se náhle změnit kvůli určitým okolnostem. Když k tomu dojde, tak je velmi obtížné přenastavit plán na zbytek dne vzhledem k počtu závislostí, náročnosti jednotlivých prací a možných vzdáleností mezi pracovišti. Jeden z hlavních důvodů ustavičného využívání ručního řešení v tištěné formě bývá v prostředích s nižší digitální gramotností, limitovaným přístupem k moderním technologiím a pro jednoduché nebo jednorázové úlohy není nutné, aby si menší firmy pořizovali příslušný software. Proto se i nadále v některých oblastech využívá ruční řešení plánování v tištěné formě.

4.1.2 Ruční řešení pomocí softwaru

V článku (22) se pojednává o tom, že ruční plánování se vyznačuje funkcemi "drag and drop", kdy uživatel může snadno přetahovat pracovní příkazy a úkoly do kalendářů a elektronicky zaměstnancům rozesílat práci. Systém obvykle provede některé automatické kontroly, jakmile je úkol dokončen. Například zkontroluje, zda má zaměstnanec přidělený k danému úkolu potřebné dovednosti k jeho provedení. Ruční plánování nabízí automatizovanější a předvídatelnější řešení než systém založený na papírových dokumentech a pomáhá podnikům šetřit čas a zvyšovat přesnost. Protože však do plánování výrazně zasahuje člověk, může se ukázat, že je obtížné prostřednictvím tohoto systému řídit složité procesy plánování.

4.1.3 Poloautomatické řešení

Článek (22) se zmiňuje o tom, že službu poloautomatického plánování stále vyvolává uživatel, aby následně spustil automatizovaný proces. Ačkoli je tento systém plánování automatický, tak stále vyžaduje určitou úroveň lidské interakce. Mnoho podniků ji upřednostňuje, protože jim umožňuje zachovat určitou úroveň kontroly a důvěry. Obsahuje nástroje pro rozhodování, které mohou poradit uživateli o dopadu přidání dané akce nebo úkolu do rozvrhu. Taková funkce se nazývá inteligentní poradce, která umožňuje uživatelům činit rychlá a informovaná rozhodnutí, aby byla zachována co nejlepší kvalita plánu.

4.1.4 Plně automatické řešení

Autor článku (22) uvádí, že plně automatické plánování je v dnešní době velmi rozšířené a využívané mnohými firmami. Některé společnosti nemusí mít čas nebo prostředky na to, aby přetahovaly pracovní příkazy do manuálních a poloautomatických systémů a kontrolovaly všechny podrobnosti. Může se také ukázat, že je obtížné udržet si přehled o plánování všech denních havarijních úkolů. Plně automatické plánování posouvá intuitivní povahu automatického plánování ještě dále. Takový nástroj rychle a snadno automaticky sestavuje plány a rozesílá práci zaměstnancům na základě předem definovaných parametrů. Tento proces zcela eliminuje manuální zásahy člověka.

5 Problém rozpisu služeb

Článek (23) uvádí rozpis služeb sester neboli „*Nurse Rostering problem*“ jako jeden z klíčových problémů, které musí každá nemocnice pravidelně řešit. Cílem celého procesu je najít ideální řešení rozvržení směn, které dosahuje nejlepšího poměru mezi kvalitou rozvrhu a potřebným časem k jeho dokončení. Nicméně stále existuje propast mezi výzkumem personálního plánování a jeho implementací v praxi, kterou lze snížit větším zapojením plánovačů a vývojářů softwaru. Jak uvádí příspěvek (24), tak již v dnešní době se hojně používají systémy, které jsou schopny téměř samostatně vytvářet a spravovat rozvrhy směn pro celý personál dané nemocnice. Pro řešení problému rozpisu služeb se používají hlavně následující algoritmy:

- **Heuristické**
  - Článek (25) pojednává o tom, že heuristika je přístup k řešení problému, který využívá praktický postup k vytvoření proveditelného řešení, které je dostatečně dobré pro rychlé vyřešení konkrétního problému a dosažení okamžitých cílů, nemusí však nutně jít o optimální řešení.
- **Optimalizační**
  - Autor článku (25) uvádí, že optimalizační model využívá inteligentní, automatizovaný proces k vytvoření optimálního řešení konkrétního problému, zohledňuje rozhodovací proměnné, stejně jako omezení a

klíčové ukazatele výkonnosti. Cílem řešení je nabídnout nejlepší možnou cestu k dosažení optimální výkonnosti, tedy maximalizovat výkonnost a celkovou ziskovost.

- **Genetický**
    - Autor článku (26) popisuje genetický algoritmus, který využívá adaptivní vyhledávací algoritmy, které se řadí do větší části evolučních algoritmů. Jedná se o inteligentní využití náhodného hledání opatřeného historickými daty k nasměrování hledání do oblasti lepšího výkonu v prostoru řešení. Jsou založeny na analogii s genetickou strukturou a chováním chromozomů populace. Běžně se používají ke generování kvalitních řešení optimalizačních problémů a problémů vyhledávání.

5.1 Definice problému

Autor článku (23) uvádí, že při řešeném reálném problému rozpisu služeb je cílem vytvořit plán rozpisu služeb zdravotnického zařízení pro stanovený plánovací horizont. Nejdůležitější pojmy definující tento problém jsou směny a periody, absence, dovednosti, smlouvy, lékaři, pracovní týmy a neslučitelnosti.

Směna je plánovaná pracovní doba lékaře, má čas začátku a trvání a spadá do určité periody. Periody jsou specifické části denní doby tak, že minimální a maximální počet povolených po sobě jdoucích směn je agregován na úrovni té stejné periody.

Absence lékařů mají různé důvody a je s nimi dále pracováno v rámci rozvrhu. Volný den je den, kdy má lékař normálně naplánované volno. Den odpočinku je konkrétní den volna po poslední noční směně v nočním sledu. Dovolená je doba, kdy si lékař dobrovolně bere den dovolené. Nepřítomnost je, když lékař pracuje na jiném oddělení nebo vykonává jinou službu.

Dovednost označuje profesní profil nebo službu. Dovednosti jsou spojeny se směnami, takže každá dovednost je přiřazena k seznamu možných směn. Lékař ve službě s danou dovedností může být přidělen pouze na směnu spojenou s touto dovedností.

Každý lékař pracuje na základě zvláštní pracovní smlouvy. Ve smlouvě je pro plánovací období stanoven: minimální a maximální počet pracovních dnů, minimální počet po

sobě jdoucích volných dnů, maximální počet po sobě jdoucích volných dnů, maximální počet pracovních nedělí, víkendů a svátků. Dále definuje minimální časovou vzdálenost mezi dvěma po sobě jdoucími nočními směnami, pracovní dny v týdnu, průměrnou denní pracovní dobu a minimální počet hodin odpočinku po daném počtu po sobě jdoucích směn.

Lékaři jsou charakterizovány pohlavím, kvalifikací, souborem proveditelných směn, pořadím posledních přidělených směn v období předcházejícím plánu horizontu. Někteří lékaři se mohou sdružovat do pracovních týmů, které shromažďují oblíbené kolegy, zatímco neslučitelnost se projevuje u dvojic lékařů, kteří mají mezi sebou nekompatibilitu.

Pracovní týmy a nekompatibility mezi lékaři mají různé úrovně. Všechny uvedené pojmy se pak promítají do finálního řešení jako proměnné, které se musí v rámci plánování směn zohledňovat a následně se celý proces stává velmi složitým.

6 Dostupný plánovací software na trhu

V dnešním dynamickém podnikatelském prostředí je efektivní plánování klíčovým faktorem úspěchu. Plánovací software nabízí organizacím nástroje pro optimalizaci jejich procesů, zlepšení efektivity a dosažení strategických cílů. Na trhu existuje několik plánovacích softwarů, které jsou rozdílné podle funkcionalit, odvětví, ve kterých jsou nejčastěji používány, a velikosti podniků, pro které jsou určeny. Pro tuto práci jsem se rozhodl zmínit uvedené plánovací softwary, které jsem vybral na základě jejich celkové popularity a co největšího možného využití pro konkrétní případ plánování služeb v rámci nemocnic. Tato část se zaměřuje na přehled nejvýznamnějších plánovacích softwarových řešení dostupných na trhu, jejich klíčové vlastnosti a výhody, které mohou přinést.

6.1 Matrickx

Matrickx (27) je plánovací software speciálně vyvinutý pro zdravotnictví, který automaticky vypočítá nejefektivnější plánování práce. Matrickx je již nyní online plánovačem nizozemského zdravotnictví. Nabízí možnost plánování poloautomaticky nebo plně automaticky podle daných pravidel, poskytuje organizaci přehled a nabízí strukturu.

Software pro plánování je naprogramován podle nejvyšších technologických standardů a je monitorován 24 hodin denně. Pracuje s modulárním systémem, takže vše může optimálně odpovídat specifickým přáním každé skupiny lékařů a specialistů. Umožňuje přidělení práv ostatním pro delegování celkové plánovací práce. Všechny provedené akce se zaznamenávají do záznamníku a systém sleduje, jak často kdo na jakém místě pracoval.

Pokud je specialista několik dní mimo službu, tak Matrickx odpočítává dny a uvádí, kdy přesně bude lékař opět k dispozici. Kromě toho lze zobrazit několik vizualizací plánování: týdenní plány, rozvrhy a řídicí panely, které lze upravit podle individuálních preferencí.

Matrickx je dostupný v rámci tří různých předplatných: Bright, Smart a Genius. Předplatné Bright obsahuje možnost pouze manuálního plánování, které je primárně pro soukromé kliniky. Předplatné Smart disponuje širší škálou možností a zároveň má k dispozici poloautomatické plánování. Nejprestižnější dostupné předplatné je Genius, které poskytuje všechny přístupné moduly a možnosti, odlišuje se od předplatného Smart cenou, a hlavně plně automatickým plánováním. Cena těchto předplatných není veřejně k dispozici a může se lišit podle domluvy.

6.2 UKG software pro plánování zaměstnanců

UKG (28) je plánovací software pro podniky všech velikostí. Umožňuje automatizovat proces vytváření a správy pracovních rozvrhů zaměstnanců ať se jedná o restauraci, maloobchod, zdravotnické zařízení nebo jiný podnik.

Je zde možnost sestavit flexibilní rozvrhy, které se přizpůsobí životu zaměstnanců, a zároveň využívá poznatky založené na datech, aby se přizpůsobil poptávce. Zobrazuje přehled o personálním pokrytí v reálném čase, umožňuje také rychle analyzovat nadbytečné a nedostatečné personální obsazení, aby bylo možné v danou chvíli provést vhodnější úpravy personálního obsazení.

Pomocí strojového učení umožňuje dlouhodobý pohled na budoucí personální a obchodní potřeby. Využívá pokročilou umělou inteligenci UKG Bryte k optimalizaci plánování, rozvrhování pracovních zdrojů a generování přesných předpovědí.

Plánovací nástroje UKG nabízejí rozšířený přehled a řadu definovatelných ukazatelů na základě údajů o plánování, obchodních údajů a informacích o lidech.

Nástroj UKG poskytuje dvě dostupné verze softwaru: UKG READY a UKG PRO. UKG READY je plánovací software pro menší nebo středně velké firmy, který rovněž obsahuje nástroje pro řízení lidských zdrojů a mzdy v rámci celé firmy. UKG PRO je plánovací software vhodný pro větší organizace, jedná se o vylepšenou verzi, která poskytuje všechny zmíněné služby od UKG READY, a navíc umožňuje řízení lidského kapitálu. Cena těchto produktů není veřejně k dispozici a může se lišit podle nabídky.

6.3 Humanity

Jedná se o dynamický plánovací software pro zaměstnavatele jménem Humanity (29). Poskytuje předpovědi na bázi umělé inteligence, automatizované plánování a je dostupný i skrze mobilní aplikaci. Umožňuje vytváření plánů a kontroluje zároveň náklady pomocí prediktivního předvídání a automatického plánování v reálném čase podle dostupnosti a dovedností. Pomáhá automatizovat komunikaci pro rychlé obsazení volných směn, sdílení aktualizací rozvrhu a zasílání připomínek ohledně směn. Snižuje riziko nedodržování předpisů a přizpůsobuje se pracovním zákonům a interním směrnicím. Obsahuje zabudované historické záznamy a auditní protokoly pro možnost sledování proběhlých změn v systému.

Humanity má svou vlastní aplikaci, která je dostupná na všech zařízeních, tím pádem mají zaměstnanci neustálý přístup do systému, dostávají automatizované upozornění o nejbližších termínech jejich směn a mají možnost okamžitého zabrání či změnění směn. Hojně se Humanity využívá jak v soukromém, tak i veřejném sektoru, například ve zdravotnictví, stavebnictví, maloobchodech nebo státní správě. Cena není přímo veřejně k dispozici, odvíjí se podle počtu uživatelů, kteří budou software ve firmě využívat.

6.4 MakeShift

Makeshift (30) je software pro plánování služeb zaměstnanců, který byl vyvinut speciálně pro plánování ve zdravotnictví. Společnost MakeShift byla založena na

přístupu zaměřeném na lidi s cílem pomoci manažerům a zaměstnancům zjednodušovat plánování. Software je vhodný pro menší, střední nebo velký firmy.

Jedna z hlavních funkcí softwaru je okamžitý volný přístup ke směnám, který umožňuje zaměstnancům si snadno prohlédnout volné směny, když si chtějí vzít směnu navíc. Také je k dispozici plánování na základě rotace, které dovoluje si lehce vytvořit rotační plán směn pro rovnoměrné rozložení pracovní zátěže. Další funkcí je možnost výměny směn zaměstnanců, kde si zaměstnanci mohou navzájem snadno vytvářet žádosti o výměnu svých směn. Software rovněž poskytuje správu a schvalování žádostí o volno nebo dovolenou zaměstnanců. Posílá upozornění na pauzy a přesčasy, aby se předešlo přepracování nebo vyhoření zaměstnanců. Umožňuje posílání oznámení zaměstnancům nebo odesílání individuálních zprávy. MakeShift je kromě toho dostupný v podobě aplikace na mobilních telefonech, aby ho všichni zaměstnanci mohli kdykoliv používat dle potřeby.

Podle přístupné statistiky software vede k 55% zkrácení času potřebného k sestavení rozvrhu a 83% zkrácení času potřebného k obsazení směn. Software také poskytuje integraci s předními personálními a mzdovými systémy, které automaticky synchronizují rozvrhy a pracovní výkazy.

MakeShift má k dispozici tři předplatné: Core, Advanced a Elite, ale nabízí i bezplatnou zkušební verzi. Předplatné Core začíná na $2.75 za zaměstnance na měsíc, které obsahuje uvedené základní nástroje softwaru. Předplatné Advanced je vylepšené Core, cena se odlišuje podle nabídky, obsahuje možnost přidání doplňků nebo modulů, pokročilé oprávnění a umělou inteligenci pro vytváření rozvrhu. Předplatné Elite je vylepšení předešlých balíčků, cena se opět různí podle domluvy, obsahuje navíc přístup k aplikačnímu programovému rozhraní, škálovatelnost, možnost přizpůsobení aplikace a modul umělé inteligence pro nejlepší automatické vytváření rozvrhu.

6.5 QGenda

Jedná se o komplexní platformu pro plánování provozu zaměstnanců pod názvem QGenda (31), která umožňuje centrálně řídit celou pracovní sílu ve zdravotnictví a

optimalizovat kapacitu pro poskytování péče v celém podniku. Poskytuje řešení pro nemocnice a zdravotnické systémy.

Mezi poskytované řešení patří automatizované plánování založené na pravidlech, podporuje jedinečné potřeby všech zaměstnanců, specializací a oddělení v rámci celého podniku. Umožňuje vytváření požadavků a výměn, zajišťuje, aby všechny výměny odpovídaly požadavkům na kvalifikace zaměstnanců. Pomocí mobilní aplikace QGenda je možnost spravovat a zobrazovat rozvrhy poskytovatelů kdekoli a kdykoli. Centralizovaný přehled o dostupnosti místností, vybavení a atributech péče v reálném čase umožňuje sdílení prostoru a zlepšuje využití místností nemocnice. Umožňuje zobrazení a správu bloků vyšetřoven, rozvrhů zaměstnanců a informací o pacientech v rámci jedné platformy. Platforma poskytuje centralizovaný a okamžitý přístup k aktuálním rozpisům pohotovostí pro všechna oddělení a zajišťuje, že v případě nouze jsou pacientům okamžitě k dispozici ti správní zaměstnanci. Automatizuje a zajišťuje kompletní mzdy na základě komplexních mzdových pravidel.

Podle dostupných statistik uvedená platforma snižuje čas potřebný na vytváření a měnění rozvrhů o 50%. Snižuje rovněž mzdové náklady na přesčasy a směny navíc o 25%. Prodejce neposkytuje veřejnou cenu a může se lišit dle domluvy.

7 Návrh webové aplikace systému

Při návrhu webové aplikace systému je nezbytné definovat způsob jeho realizace. Mezi klíčové aspekty návrhu patří vymezení konkrétních požadavků na systém, popis jeho fungování a logiky, zabezpečení přenášených dat, návrh uživatelského rozhraní a výběr použitých technologií pro frontend i backend aplikace.

7.1 Vymezení požadavků na nemocniční systém

Tato část se zaměřuje na definování konkrétních požadavků pro vývoj nemocničního systému určeného k plánování služeb. Efektivní systém pro plánování směn je klíčový pro optimalizaci pracovních procesů v nemocnicích. Správně navržený systém může významně přispět ke zvýšení produktivity a efektivity práce, snížení administrativní zátěže a minimalizaci rizika vyhoření zaměstnanců

Mezi základní požadavky na systém pro plánování služeb patří:

7.1.1 Řešení plánování služeb

V této části jsou uvedeny informace poskytnuté odborným konzultantem z oblasti v návaznosti na zadání práce. Požadavky zde uvedené slouží jako klíčové vstupní parametry pro návrh celého systému.

- Systém poskytuje každému lékaři svůj osobní účet, kde má k dispozici základní informace, kalendář plánování a svou přiřazenou sloužící skupinu.
- V části plánování je lékařům k dispozici rozvrh, pomocí kterého jsou schopni si naplánovat své noční služby. Mohou si zvolit pouze služby, které jsou plánované mezi 18. a 6. hodinou.
- Nelze vypsat služby na 2 po sobě jdoucí dny, pokud si daný lékař nechá tuto preferenci zapnutou.
- Každý lékař dané skupiny by měl mít zhruba stejné množství služeb, aby splnil minimální měsíční pracovní dobu. Zároveň by neměl mít více než 2 víkendy nebo svátky měsíčně, pokud si tuto preferenci zvolí.
- Lékaři si mohou plánovat své služby alespoň na 2 měsíce dopředu, ale musí to být do 15. dne předchozího měsíce, protože se každý 15. den měsíce zveřejňují služby na následující měsíc.

- Lékaři jsou v rámci systému rozděleni do několika skupin, každá skupina má svůj oddělený rozvrh a daný den slouží od každé skupiny jeden lékař. Jakýkoliv lékař může být součástí pouze jedné skupiny.
- Pokud si lékař nenaplánuje své služby, aby splnil minimální pracovní dobu, tedy alespoň 180 hodin měsíčně, systém mu samostatně přidělí službu dle potřeby.
- Pokud při plánování dojde ke kolizi požadavků služeb v rámci stejné skupiny dvou různých lékařů, tak systém schválí dříve zadaný požadavek a následně oznámí druhému lékaři, že dané datum již není k dispozici, společně se jménem lékařem, kterým je již obsazeno.

Systém zahrnuje také uživatele s rolí administrátora, který má vyšší úroveň oprávnění než běžní uživatelé. Administrátor disponuje přístupem k rozvrhům všech registrovaných lékařů a má k dispozici informace o tom, zda si jednotliví lékaři naplánovali své služby na daný měsíc. Dále má možnost nahlížet do předběžně vytvořeného rozvrhu na dané období a provádět manuální úpravy služeb všech lékařů.

Administrátor může vytvářet nové skupiny a přiřazovat do nich lékaře, přičemž každá skupina je omezena maximálně na dva členy. Dále má pravomoc přesouvat lékaře mezi jednotlivými skupinami. Systém však zároveň zajišťuje automatickou kontrolu konzistence dat. Při editaci rozvrhů, úpravě skupin, nastavování pravidel či přesouvání lékařů mezi skupinami systém administrátora upozorňuje na možné konflikty, čímž minimalizuje riziko narušení systémové logiky a architektury.

7.1.2 Přívětivé UX/UI

Nemocniční systém musí zohledňovat velké množství uživatelů, včetně zaměstnanců nemocnice, lékařů a dalších. V ten moment přichází řešení uživatelského dojmu a uživatelského rozhraní, tedy User Experience a User Interface. (6) UX není na první pohled viditelný, jedná se o návrh služby způsobem, aby splnil uživatelovy požadavky, snaží se uživatele nepřehltit spousty zbytečnými informacemi. UI aplikace je na první pohled viditelný, jedná se o návrh prvků, interakcí a animací v rámci celé aplikace, pokud uživatel klikne na tlačítko tak hned má zpětnou vazbu o tom, že se něco stalo.

UX/UI musí být informativní, jednoduché a snadno použitelné. Předpokládá se, že ne každý člověk v nemocnici je profesionál v zacházení s technologiemi a potřebuje řádné školení, aby mohl používat složité funkce softwaru. Díky uživatelsky přívětivému UX/UI může každý používat software efektivně i účinně a využít z něj to nejlepší.

7.1.3 Dodržování předpisů a zabezpečení dat

Příspěvek (24) pojednává o jednom z hlavních požadavků, který musí systém splňovat. Tím je ochrana všech dat, která by při úniku mohli narušit zabezpečení systému, jako jsou osobní nebo přihlašovací údaje. Nemocniční data mohou zahrnovat nemocniční záznamy, které by se daly zneužít. Systémová data provádějí ukládání, výměny a mazání dat, tedy je potřeba je chránit před škodlivými aktivitami, aby se zamezilo zneužití nebo ztrátě dat. Rovněž je potřeba dodržovat právní předpisy a zákoník práce (21) v rámci plánování jednotlivých služeb pro pracovníky nemocnice.

7.2 Uživatelské rozhraní

Tato kapitola se zaměřuje na klíčové prvky uživatelského rozhraní aplikace, které hrají zásadní roli v její celkové použitelnosti a přívětivosti. Nejprve jsou představeny drátové modely, které slouží jako vizuální návrh základní struktury rozhraní a usnadňují plánování jednotlivých funkcionalit. Následně jsou popsány principy uživatelského dojmu (UX), jež byly při návrhu aplikace zohledněny s cílem zajistit intuitivní ovládání a pozitivní uživatelskou zkušenost.

Kapitola se rovněž věnuje vizuálním aspektům aplikace, včetně grafického designu, barevného schématu a typografie, které přispívají k celkové konzistenci a estetice systému. Cílem této části je představit proces návrhu uživatelského rozhraní a demonstrovat, jak byly jednotlivé prvky optimalizovány pro dosažení efektivního a uživatelsky přívětivého prostředí.

7.2.1 Drátové modely

Wireframing neboli vytváření drátových modelů je způsob, jak navrhnout webové služby na strukturální úrovni. Jak uvádí webový příspěvek, (32) drátový model se běžně používá k rozvržení obsahu a funkcí na stránce, které zohledňují potřeby

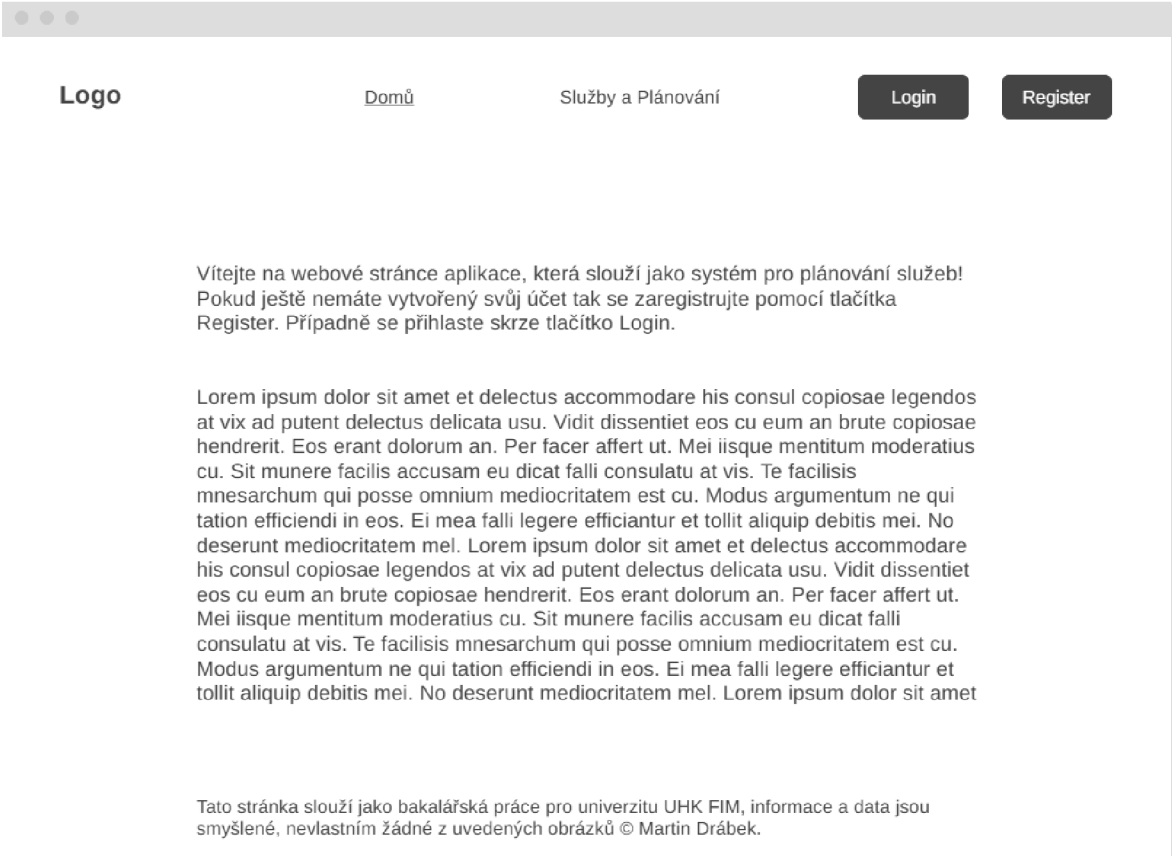
uživatelů a jejich cesty. Používají se na počátku procesu vývoje k vytvoření základní struktury stránky předtím, než se přidá vizuální design a obsah.

Drátový model je rozvržení webové stránky, které ukazuje, jaké prvky rozhraní budou na klíčových stránkách. Jedná se o důležitou součást procesu celkového návrhu. Lze je také použít k vytvoření globální a sekundární navigace, aby se zajistilo, zdali terminologie a struktura použitá pro stránky splňuje očekávání uživatelů.

Využití drátových modelů by mělo nastat již v rané fázi projektu, aby se získal souhlas uživatelů a klientů s rozvržením klíčových stránek a navigací. To projektovému týmu, konkrétně designérům, dodá jistotu při dalším postupu. Zároveň ušetří značný čas a peníze ve fázi testování a úprav v pozdější fázi projektu.

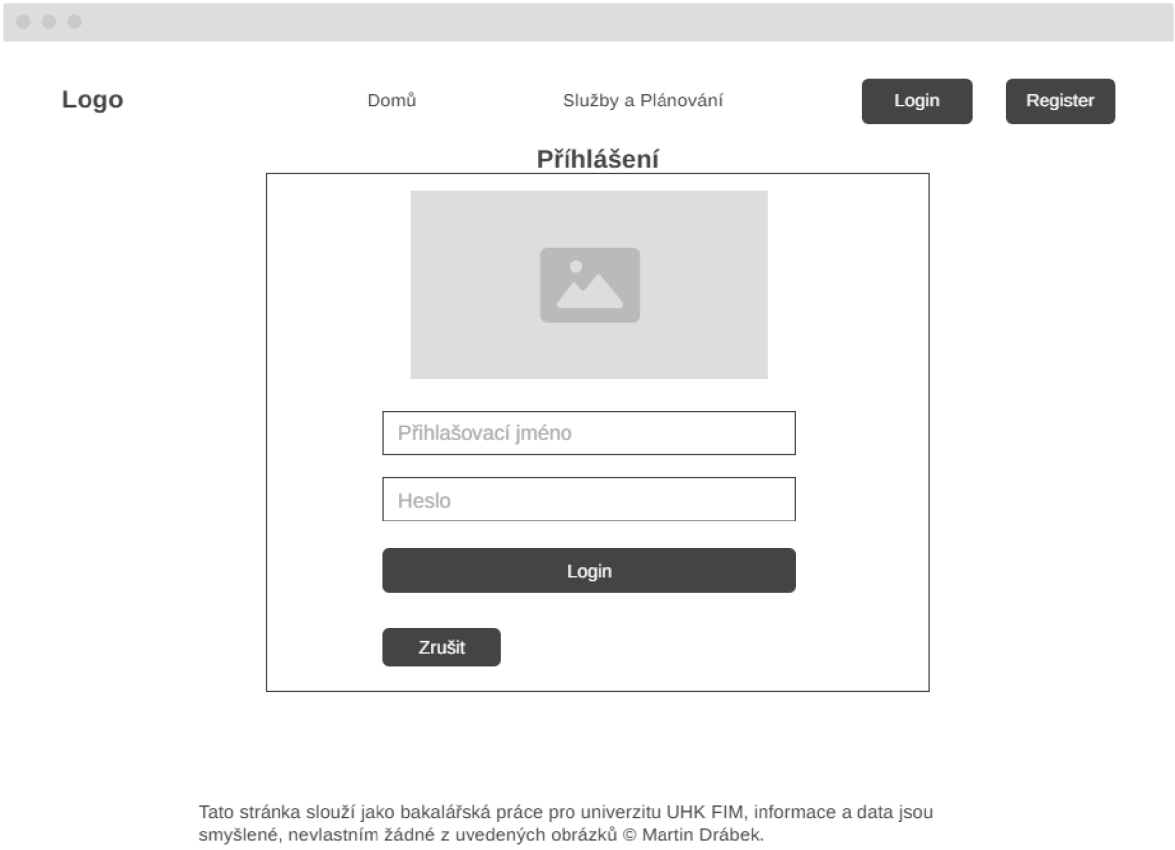
V rámci této práce bylo vytvořeno a popsáno několik drátových modelů pro klíčové části webové aplikace:

První drátový model (viz. Obrázek 3) ukazuje základní pohled webové stránky aplikace, pokud uživatel není přihlášen a nachází se v části 'Domů', obsahuje navigaci, tlačítka pro přihlášení nebo registraci a úvodní text.



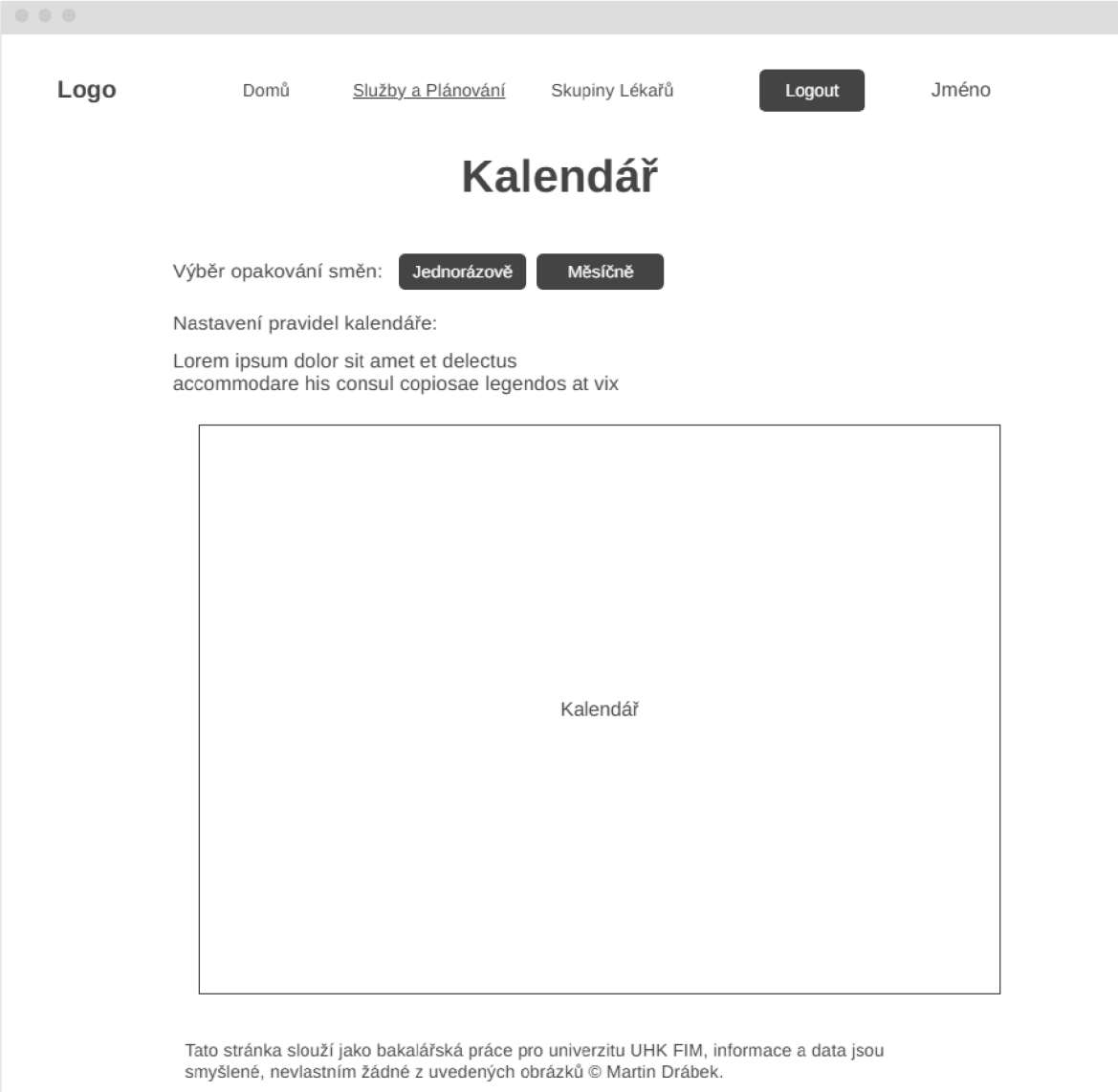
Obrázek 3: Drátový model úvodu
Zdroj: autor

Druhý drátový model (viz. Obrázek 4) znázorňuje stránku pro přihlášení uživatele, která se objeví po stisknutí tlačítka ‘Login‘. Obsahuje formulář pro vyplnění přihlašovacího jména a hesla, tlačítkem ‘Login‘ v okně formuláře se uživatel přihlásí, nebo může tento proces vyrušit stisknutím tlačítka ‘Zrušit‘.



Obrázek 4: Drátový model přihlášení
Zdroj: autor

Třetí drátový model (viz. Obrázek 5) vyobrazuje hlavní segment celé webové aplikace systému. Jedná se o kalendář, který se nachází v části 'Služby a Plánování', do kterého přihlášený uživatel může zadávat svoje směny. Kalendář také obsahuje tlačítka pro výběr opakování směn a nastavení celkových pravidel kalendáře, včetně informací relevantních pro uživatele.



Obrázek 5: Drátový model kalendáře
Zdroj: autor

Čtvrtý drátový model (viz. Obrázek 6) zobrazuje segment webové aplikace v části 'Skupiny Lékařů'. Zde se nachází skupiny registrovaných lékařů, které jsou pro běžné přihlášené uživatele dostupné pouze k prohlížení. Nicméně pro admin uživatele je možné tyto skupiny editovat, tedy přemisťovat uživatele mezi skupinami, vytvářet nové skupiny nebo i mazat vybrané skupiny. Nově registrovaní uživatelé se zobrazují

ve skupině ‘Nezařazení‘ a všichni přiřazení uživatelé dané skupiny spolu sdílí rozvrh pro plánování svých služeb.

Obrázek 6: Drátový model skupin
Zdroj: autor

7.2.2 Principy uživatelského dojmu

Návrh uživatelského prostředí je proces vytváření produktů a služeb, které řeší konkrétní problém uživatele a zároveň zajišťuje, aby se navrhované řešení snadno a příjemně používalo. Podle článku (33) existuje několik univerzálních principů, které se dají využít pro vylepšení uživatelského dojmu:

- Mezi první princip patří orientace na uživatele, která spočívá ve vytváření produktů a služeb, které řeší problémy uživatelů. Na první místo se kladou potřeby uživatele a rozhoduje se na základě toho, co o něm je známo a co od produktu chce.
- Druhý z principů je konzistence, tedy aby návrhy byly konzistentní, pokud jde o jejich vzhled a fungování na všech stránkách a prvcích v rámci stejné aplikace. Měla by existovat konzistence ve vizuálních prvcích, jako jsou tlačítka a odkazy.

- Třetí z principů je hierarchie, která určuje, jak uživatel prochází stránku a jak je tento proces snadný nebo složitý. Zohledňuje se, jak jsou jednotlivé prvky na stránce rozmístěny. Důležitější prvky lze zdůraznit jejich umístěním v horní části stránky, použitím většího písma nebo různých barev, které jim pomohou vyniknout.
- Čtvrtým principem je kontext, který zohledňuje okolnosti, za kterých bude produkt používán, a to, jak mohou určité faktory ovlivnit uživatelskou zkušenost. Prohlížení z mobilního telefonu je nyní hlavním způsobem, jakým většina uživatelů přistupuje k webovým stránkám, takže webové stránky musí být optimalizovány pro tuto formu prohlížení webu, aby vše fungovalo a zobrazovalo se správně.
- Pátý princip je poskytnutí uživateli správnou míru kontroly nad tím, jak s produktem může pracovat. Pomáhat uživatelovi opravit nebo zvrátit chyby, aniž by se tím narušilo celé uživatelské prostředí. Kontrolu a svobodu uživatele lze zabudovat pomocí tlačítka 'Cancel' a jasného označení alternativních akcí a cest, které může uživatel zvolit, pokud chce určitou akci vrátit zpět.

7.2.3 Grafický styl aplikace

Podle příspěvku (34) je vizuální stránka nedílnou součástí procesu tvorby webových stránek. Je potřeba zvážit, jak spolupracují barvy, fonty a multimédia, aby se vytvořil atraktivní vzhled, který odpovídá danému produktu. Vizuální design je často to první, čeho si návštěvník při kontaktu se stránkou všimne.

Atraktivní, profesionální a poutavé uspořádání webových stránek vytváří pozitivní dojem a podněcuje lidi k prozkoumání jeho obsahu. K tomu patří jedinečná barevná paleta, konzistentní typografie a vlastní obrázky, které vytvářejí rozpoznatelnou identitu celého produktu a činí stránky zapamatovatelnými.

Grafický styl této webové aplikace je navržen s důrazem na jednoduchost, přehlednost a profesionální vzhled. Barevná paleta je založena především na odstínech zelené, modré, černé a bílé, což přispívá k vizuální harmonii a usnadňuje orientaci uživatele v rozhraní.

Typografie aplikace využívá písmo z rodiny sans-serif, konkrétně Arial, Helvetica, sans-serif, které zajišťuje dobrou čitelnost na různých zařízeních a displejích. Uživatelské

rozhraní obsahuje prvky s minimalistickým designem, kde jsou dominantní jasné kontrasty mezi textem a pozadím. Tlačítka a interaktivní prvky využívají barevné zvýraznění pro lepší navigaci a intuitivní ovládání. Celkový vizuální styl tak podporuje moderní a efektivní uživatelskou zkušenost.

7.3 Výběr technologií

Tato kapitola se věnuje výběru technologií a prvků použitých při vývoji aplikace. Nejprve je popsána celková struktura vyvíjené webové aplikace, poté jsou detailně představeny hlavní komponenty frontendu a backendu, použitá databáze a zvolený editor zdrojového kódu. Cílem této části je objasnit důvody volby konkrétních technologií a jejich přínos pro výsledný systém.

7.3.1 Webová aplikace

Na základě zadání a stanovených cílů této práce byl zvolen vývoj webové aplikace pro systém plánování služeb. Webové aplikace mají tu výhodu, že je uživatelé mohou používat přímo v internetovém prohlížeči, aniž by bylo nutné stahovat a instalovat software na jejich zařízení. Díky tomu je aplikace dostupná odkudkoliv, kdykoliv a na jakémkoliv zařízení, nezávisle na operačním systému. Zároveň probíhá automatická aktualizace na nejnovější verzi bez nutnosti zásahu uživatele. Aplikace je navržena tak, aby byla plně responzivní, což zajišťuje její bezproblémové používání nejen na počítačích, ale i na menších zařízeních, jako jsou mobilní telefony a tablety.

7.3.2 Zdůvodnění výběru technologií

Při vývoji webové aplikace Systému plánování služeb pro nemocnice byly zvoleny technologie, které umožňují efektivní a přehlednou implementaci s důrazem na jednoduchost, rozšiřitelnost a moderní přístup k vývoji webových aplikací.

Pro tvorbu uživatelského rozhraní byly použity standardní technologie webového vývoje – HTML, CSS a JavaScript. Tyto technologie umožňují snadnou tvorbu strukturovaného obsahu, jeho vizuální stylizaci a interaktivitu. Jako hlavní framework byl zvolen Vue.js, a to především díky jeho jednoduchosti, reaktivnímu přístupu a snadné integraci do projektu. Vue.js poskytuje přehlednou komponentovou architekturu a efektivní správu stavu aplikace, což výrazně usnadňuje vývoj i budoucí rozšiřování systému.

Pro serverovou část aplikace byl vybrán Node.js, který díky asynchronnímu modelu založenému na událostech umožňuje rychlé a škálovatelné zpracování požadavků. Backend komunikuje prostřednictvím RESTful API, což zajišťuje flexibilní správu dat a efektivní propojení s frontendem. Pro autentizaci uživatelů byl implementován JWT (JSON Web Token), který poskytuje bezpečné a efektivní ověřování identity uživatelů bez nutnosti udržování serverových relací.

Jako databázový systém byla zvolena MongoDB, především s ohledem na předchozí zkušenosti s touto technologií. MongoDB je dokumentově orientovaná databáze, která umožňuje snadnou práci s daty bez nutnosti pokročilé znalosti SQL. Je flexibilní, dobře škálovatelná a vhodná pro správu dynamických datových struktur, což odpovídá požadavkům systému na plánování služeb v nemocnicích.

Pro vývoj aplikace byl využit editor Visual Studio Code, který je populární pro svou širokou podporu rozšíření a nástrojů usnadňujících vývoj. Nabízí integrovanou podporu pro JavaScript, Vue.js i backendové technologie, což přispívá k efektivnímu vývoji a údržbě projektu.

Výběr těchto technologií byl motivován snahou o vytvoření přehledného, snadno udržovatelného a výkonného systému, který poskytne dostatečnou flexibilitu pro implementaci potřebných funkcionalit i případné budoucí rozšíření aplikace.

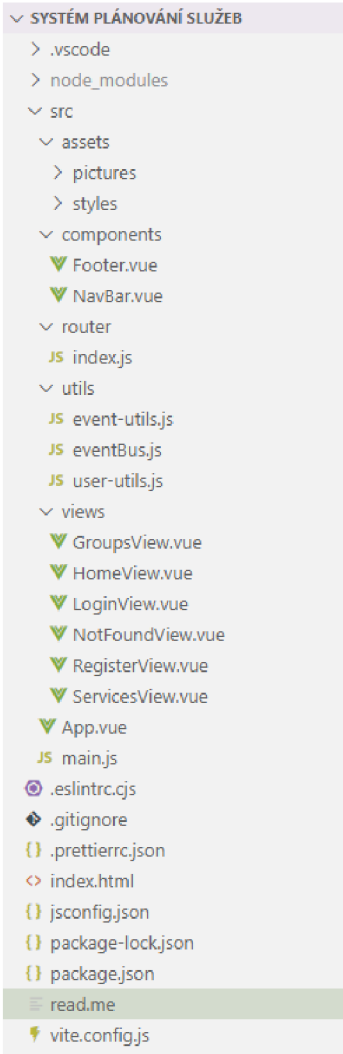
8 Implementace webové aplikace systému

Tato kapitola se věnuje vývoji frontendu a backendu webové aplikace systému plánování služeb. Následující podkapitoly popisují jednotlivá řešení a funkcionality, včetně implementace klíčových prvků v rámci celé aplikace. Dále je zde uvedeno zobrazení samostatných pohledů aplikace a jejich role v uživatelském prostředí.

8.1 Organizace projektu

Nejdříve je potřeba popsat organizaci balíčků a souborů. Vývoj projektu, tedy nová webová aplikace systému, byl rozčleněn do několika podstatných složek. Nejdříve bude popsána struktura projektu frontend části (viz. Obrázek 7) a následně struktura projektu backend části (viz. Obrázek 8).

8.1.1 Frontend struktura



Obrázek 7: Struktura frontend projektu
Zdroj: autor

Frontendová část systému je implementována pomocí frameworku Vue.js. Struktura projektu je navržena tak, aby byla přehledná, modulární a snadno rozšiřitelná. Kořenový adresář projektu obsahuje základní soubory potřebné pro běh aplikace, včetně konfiguračních souborů a definice závislostí. A také se zde nachází konfigurační soubory pro ESLint (35) a Prettier (36), které slouží ke kontrole a formátování kódu podle zvolených standardů. Mezi hlavní soubory v tomto adresáři patří:

- package.json – obsahuje seznam závislostí a konfigurační informace pro správu projektu, včetně skriptů pro spuštění aplikace.
- vite.config.js – konfigurační soubor pro Vite, který je použit jako nástroj pro rychlou kompilaci a optimalizaci kódu.

- index.html – hlavní HTML soubor, do kterého je Vue aplikace vkládána

Složka src , která představuje hlavní zdrojový kód aplikace, obsahuje všechny klíčové soubory potřebné pro její fungování. Tato složka je rozdělena do několika podadresářů, které zajišťují přehlednost a usnadňují správu kódu.

Složka assets obsahuje statické soubory používané aplikací. Jsou zde dvě hlavní podsložky: pictures, která obsahuje obrázky využívané v uživatelském rozhraní a styles, která slouží k ukládání CSS stylů sdílených mezi jednotlivými komponentami.

Složka components obsahuje jednotlivé Vue komponenty, které jsou znovu využívány napříč aplikací. Mezi hlavní komponenty patří soubor Footer.vue, který je odpovědný za zobrazení patičky stránky, a Navbar.vue, což je navigační panel umožňující přechod mezi jednotlivými stránkami aplikace. Tyto komponenty jsou následně importovány a používány v jednotlivých pohledech aplikace.

Ve složce router se nachází soubor index.js, který definuje směrování v aplikaci. Tento soubor obsahuje konfiguraci jednotlivých cest a propojuje je s odpovídajícími pohledy ve složce views. Routing je implementován pomocí knihovny Vue Router (37), která umožňuje navigaci mezi stránkami bez nutnosti jejich úplného znovunačtení.

Složka utils obsahuje pomocné funkce a moduly, které usnadňují práci s daty a událostmi v aplikaci. Mezi hlavní soubory v této složce patří:

- event-utils.js - poskytuje funkce pro práci s událostmi v kalendáři.
- eventBus.js - implementuje jednoduchý Event Bus, který umožňuje komunikaci mezi různými komponentami aplikace. (38)
- user-utils.js – obsahuje uživatelské utility, například třídu pro načítání obsahu podle role uživatele.

Složka views obsahuje soubory odpovídající různým stránkám aplikace. Každý .vue soubor zde představuje jednu stránku, ke které lze přistupovat pomocí směrování skrze navigační panel. Mezi hlavní pohledy patří:

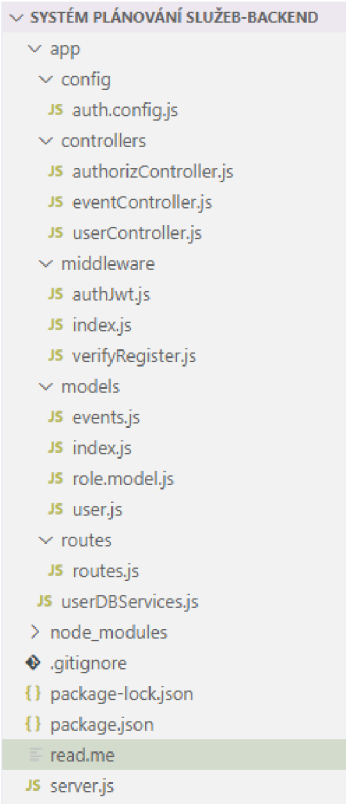
- HomeView.vue – úvodní stránka aplikace.
- LoginView.vue – stránka pro přihlášení uživatele.
- RegisterView.vue – registrační stránka pro nové uživatele.

- ServicesView.vue – stránka kalendáře, který slouží přihlášenému uživateli k zobrazování, vytváření a úpravě jeho událostí v rámci jednotlivých měsíců.
- GroupsView.vue – slouží k zobrazení a správě skupin uživatelů v rámci systému.
- NotFoundView.vue – stránka zobrazující chybovou hlášku v případě, že uživatel zadá neexistující URL.

Soubor App.vue (39) slouží jako hlavní komponenta Vue aplikace. Obsahuje základní strukturu, do které se vkládají jednotlivé podstránky na základě aktivního směrování. A posledním souborem je main.js což je hlavní vstupní bod Vue aplikace. Zde probíhá inicializace Vue instance, připojení routeru a další globální konfigurace.

8.1.2 Backend struktura

Backendová část aplikace je implementována v jazyce JavaScript s využitím Node.js a frameworku Express.js. Hlavním úkolem této části systému je spravovat databázi, zajišťovat autentizaci a autorizaci uživatelů a poskytovat API rozhraní pro komunikaci s frontendovou částí. Pro zajištění přehlednosti a modularity je kód rozdělen do několika logických celků.



Obrázek 8: Stuktura backend projektu
Zdroj: autor

V kořenovým adresáři backendové části projektu se nacházejí klíčové soubory potřebné pro běh serveru:

- server.js – hlavní vstupní bod aplikace, který inicializuje Express server, nastavuje middleware a propojuje jednotlivé části systému.
- package.json – soubor obsahující konfiguraci projektu, seznam závislostí a skripty pro spuštění serveru.
- .gitignore – soubor definující soubory a složky, které by neměly být verzovány v repozitáři.

V rámci adresáře app jsou umístěny klíčové části backendového systému. Tento adresář je rozdělen do několika podadresářů, které reprezentují jednotlivé vrstvy aplikace. Složka config, obsahuje soubor pro konfiguraci tajného klíče pro generování JWT tokenů (17). Dále se zde nachází soubor userDBServices.js, který obsahuje funkce pro práci s databází uživatelů, jako je vyhledávání, aktualizace a mazání údajů.

Složka controllers obsahuje soubory odpovědné za řízení jednotlivých funkcí backendu. Kontrolery zpracovávají požadavky přijaté z frontendové části, provádějí operace nad daty a vracejí odpovědi klientovi. Mezi hlavní kontrolery patří:

- authorizController.js – spravuje proces autorizace a řízení přístupových oprávnění uživatelů.
- eventController.js - řídí správu kalendářních událostí v systému, například jejich vytváření, aktualizaci nebo mazání.
- userController.js - zajišťuje správu uživatelů, včetně registrace, přihlášení a aktualizace jejich údajů.

Složka middleware obsahuje speciální funkce, které se vykonávají mezi přijetím požadavku a jeho zpracováním v kontroleru. Middleware (40) umožňuje oddělení opakujících se operací, jako je ověřování uživatelů nebo validace vstupních dat. Tento projekt obsahuje následující middleware souborů:

- authJwt.js - implementuje ověřování uživatelů pomocí JWT tokenů a zajišťuje, že každý požadavek je autentizován platným tokenem.

- verifyRegister.js - ověřuje údaje při registraci uživatele, například kontroluje, zda e-mail již není registrován.
- index.js – slouží jako centrální bod pro export middleware funkcí, což umožňuje jejich jednodušší import do jiných částí aplikace.

Složka models obsahuje definice databázových modelů. Tyto modely určují, jak jsou data ukládána v databázi a jaké vlastnosti jednotlivé entity mají. V projektu jsou definovány následující modely:

- event.js - model událostí, obsahuje strukturu a vlastnosti jednotlivých plánovaných událostí.
- user.js - model uživatele, definuje uživatelské údaje, jako je jméno, e-mail a heslo.
- role.model.js - model rolí uživatelů, který rozlišuje různé úrovně oprávnění
- index.js – centrální soubor, který slouží k exportu modelů a jejich registraci v systému.

Složka routes obsahuje definice všech dostupných API endpointů, které umožňují frontendové aplikaci komunikovat s backendem. Hlavním souborem je routes.js, který obsahuje definice cest pro jednotlivé části systému.

8.2 Implementace frontendu

Tato kapitola se zaměřuje na konkrétní implementaci frontendu systému plánování služeb pro nemocnice. Frontend představuje uživatelské rozhraní, které umožňuje zdravotnickému personálu efektivní správu rozvrhů, plánování služeb a práci s kalendářem.

Hlavní funkcionalitou frontendu je zobrazení a správa kalendáře s plánovanými službami. Pro implementaci této části aplikace byla využita knihovna FullCalendar (41), která umožňuje interaktivní zobrazení událostí, jejich přidávání, editaci a mazání. Systém musí zajistit, že uživatelé mají přehled o svých naplánovaných směnách a že se jednotlivé události nepřekrývají.

Aplikace je rozdělena do modulárních částí, které zahrnují komponenty, routování, správu stavů a uživatelské služby. Každá část má jasně definovanou odpovědnost, což umožňuje snadnou rozšiřitelnost a úpravy v budoucnu.

V následujících podkapitolách budou podrobně rozebrány jednotlivé aspekty implementace frontendu, jako jsou konfigurace projektu, práce s komponentami, správa kalendáře, automatizace plánování a komunikace s backendem. Dále budou popsány klíčové funkce, mezi které patří správa uživatelských účtů a ochrana přístupu k jednotlivým částem aplikace

8.2.1 Konfigurace projektu

Pro implementaci frontendu byl zvolen framework Vue.js. Projekt byl inicializován pomocí příkazu: ‘`npm create vue@latest`‘ Po spuštění tohoto příkazu následuje jednoduchá instalace, během které je možné vybrat volitelné funkce. Pro tento projekt bylo zvoleno přidání Vue Routeru, který umožňuje vytváření jednostránkových aplikací (SPA), a ESLint, jenž slouží ke kontrole kvality kódu v JavaScriptu.

ESLint (35) pomáhá odhalit syntaktické chyby, problémy s formátováním, porušení stylu kódu a potenciální chyby. Díky němu je kód konzistentní a snadno udržovatelný, což přispívá k lepší čitelnosti v rámci vývoje aplikace.

Následně je důležité nainstalovat všechny závislosti použité v rámci této aplikace, pomocí příkazu: ‘`npm install`‘. Použité závislosti a jejich stručný popis:

- "@fullcalendar/core" - Hlavní knihovna FullCalendar (41), která poskytuje základní funkcionalitu kalendáře.
- "@fullcalendar/daygrid" - Modul pro zobrazení událostí v denním a měsíčním rastru.
- "@fullcalendar/interaction" - Umožňuje interaktivní prvky, jako je přetahování nebo výběr událostí.
- "@fullcalendar/rrule" - Rozšíření umožňující práci s opakujícími se událostmi pomocí pravidel RRule (42).
- "@fullcalendar/vue3" - Integrace FullCalendar s Vue 3 pro snadné použití v komponentách.
- "axios" - Knihovna pro HTTP požadavky (43), využívaná pro komunikaci mezi frontendem a backendem.
- "rrule" – Knihovna (42) umožňující definování opakujících se událostí podle standardu Recurrence Rule (RRule).

- "toastr" - Knihovna (44) pro zobrazování notifikací, jako úspěšné uložení, upozornění, chyba.
- "uuid" - Generátor unikátních identifikátorů (45) pro označování událostí.
- "vue" - Hlavní knihovna Vue.js 3 (39) pro tvorbu frontendových aplikací.
- "vue-router" - Oficiální knihovna pro směrování ve Vue.js aplikacích pro přechody mezi stránkami.
- "vuedraggable" – Knihovna (46) pro přetahování a změnu pořadí prvků.

Samotný příkaz pro spuštění běhu aplikace je: ‘`npm run dev`‘. Po spuštění je aplikace dostupná lokálně na adrese: ‘`http://localhost:3000/`‘.

8.2.2 Komunikace mezi komponentami

V rámci této aplikace je komunikace mezi komponentami řešena několika způsoby. Níže jsou popsány hlavní přístupy, které jsou v projektu využívány, a jejich konkrétní implementace.

Navigace mezi jednotlivými částmi aplikace je realizována pomocí Vue Routeru, který umožňuje definovat různé cesty aplikace a zajišťuje přechody mezi stránkami bez nutnosti jejich znovunačtení. Jak znázorňuje následující Programový kód 1, tak se zde definují cesty pro stránky aplikace a používá se createWebHistory, což umožňuje lepší navigaci bez # v URL. Pro neexistující cesty je implementováno ošetření ‘`:catchAll(.*)`‘, které přesměruje uživatele na stránku 404. V komponentách se pak využívá <RouterLink> k navigaci:

```
<RouterLink to="/sluzby">Služby a Plánování</RouterLink>
```

```
import { createRouter, createWebHistory } from 'vue-router'
import HomeView from '@/views/HomeView.vue'
import ServicesView from '@/views/ServicesView.vue'
import GroupsView from '@/views/GroupsView.vue'
import NotFoundView from '@/views/NotFoundView.vue'
import RegisterView from '@/views/RegisterView.vue'

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    { path: '/', name: 'home', component: HomeView},
    { path: '/sluzby', name: 'services', component: ServicesView},
    { path: '/skupiny', name: 'groups', component: GroupsView},
    { path: '/:catchAll(.*)', name: 'not-found', component: NotFoundView},
    { path: '/register', name: 'register', component: RegisterView}
  ]
})

export default router
```

Programový kód 1: Definice cest v routeru
Zdroj: autor

Vzhledem k tomu, že některé komponenty potřebují reagovat na změny přihlášeného uživatele, je použit Event Bus, (38) který umožňuje komponentám komunikovat mezi sebou i bez přímého propojení. Tento mechanismus je implementován v souboru eventBus.js, kde je definován samotný Event Bus objekt. Tento objekt umožňuje komponentám naslouchat událostem nebo přímo vyvolávat určité události:

```
const EventBus = new EventTarget()
export default EventBus
```

V rámci této aplikace se Event Bus využívá po úspěšném přihlášení uživatele, aby se ověřil a aktualizoval stav v NavBaru, například pro zobrazení uživatelského jména přihlášeného uživatele.

NavBar komponenta obsahuje metodu `‘checkLoginStatus()‘`, která ověřuje, zda je uživatel přihlášen na základě uloženého tokenu v localStorage, který získá po přihlášení uživatele do aplikace. Pokud token existuje, posílá žádost do backendu

k ověření, jestliže je token platný, nastaví stav `‘isLoggedIn = true‘` a zobrazí jméno přihlášeného uživatele. Pokud ověření selže uživatel je odhlášen.

Použití Event Bus v NavBaru je následovné:

```
created() { EventBus.addEventListener('user-logged-in',
this.checkLoginStatus) },
beforeUnmount() { EventBus.removeEventListener('user-logged-in',
this.checkLoginStatus) }
```

Pokud dojde k události `'user-logged-in'`, komponenta NavBar aktualizuje zobrazení přihlášeného uživatele. Naslouchání se odpojí před zničením komponenty, aby se předešlo únikům paměti. Navigační panel rovněž zobrazuje různé prvky na základě stavu přihlášení. Pokud je uživatel přihlášen, zobrazí se tlačítko Logout a jeho uživatelské jméno. V případě, že přihlášen není, jsou zobrazena tlačítka Login a Register.

8.2.3 Autentizace uživatelů

Následující podkapitoly zahrnují registraci nových uživatelů a přihlašování existujících účtů. Tento proces je klíčový pro zabezpečený přístup k aplikaci a zajišťuje, že uživatelé mohou spravovat své služby a skupiny podle svých oprávnění. Implementace autentizace využívá bezpečné metody ověřování, které chrání citlivé údaje a zajišťují správné přiřazení uživatelských rolí v systému.

8.2.3.1 Registrace uživatele

Registrace uživatele probíhá prostřednictvím komponenty RegisterView.vue, která poskytuje formulář pro zadání přihlašovacích údajů. Proces registrace se skládá z několika kroků:

1. Vyplnění registračního formuláře – uživatel zadá uživatelské jméno, e-mail, heslo a potvrzení hesla.
2. Ověření hesel na straně klienta – aplikace zkontroluje, zda zadané heslo a jeho potvrzení souhlasí.
3. Odeslání registračních údajů na server – pokud hesla souhlasí, data jsou odeslána prostřednictvím HTTP POST požadavku na server.

4. Zpracování odpovědi serveru – pokud je registrace úspěšná, uživatel je přesměrován na domovskou stránku a zobrazí se potvrzení o úspěšné registraci. V opačném případě se zobrazí varování s chybovou zprávou.

K zobrazování varovných zpráv nebo hlášení o úspěchu se využívá knihovna Toastr (44), která informuje uživatele o výsledku operací pomocí notifikací.

8.2.3.2 Přihlášení uživatele

Komponenta LoginView.vue slouží k ověření uživatele a jeho přihlášení do systému. Proces přihlášení pomocí metody `‘loginData()‘` (viz. Programový kód 2) zahrnuje následující kroky:

1. Vyplnění přihlašovacích údajů – uživatel zadá své přihlašovací jméno a heslo do formuláře.
2. Odeslání přihlašovacích údajů na server – po stisknutí tlačítka Login se data odesílají na server prostřednictvím HTTP POST požadavku.
3. Ověření uživatele na backendu – server zkontroluje správnost zadaných údajů.
4. Uložení přístupového tokenu a údajů do localStorage – pokud je přihlášení úspěšné, do localStorage se uloží přístupový JWT token, uživatelské jméno a ID uživatele. Tyto údaje jsou následně využívány k identifikaci uživatele při dalších operacích.
5. Aktualizace stavu aplikace – pomocí EventBusu je vyvolána událost ‘user-logged-in‘, která umožní ostatním komponentám aktualizovat informace o přihlášeném uživateli.

6. Přesměrování na domovskou stránku – pokud přihlášení proběhlo úspěšně, uživatel je přesměrován na hlavní stránku a zobrazí se potvrzovací notifikace.

```
async loginData() {
  try {
    const { data } = await axios.post('http://localhost:3001/login', this.user)
    if (data.status === true && this.user.username) {
      const groupsBackup = localStorage.getItem('groups')
      localStorage.clear()
      if (groupsBackup) {
        localStorage.setItem('groups', groupsBackup)
      }
      localStorage.setItem('x-access-token', data.token)
      localStorage.setItem('username', data.username)
      localStorage.setItem('userId', data.userId)
      this.closeModal()
      EventBus.dispatchEvent(new Event('user-logged-in'))
      router.push('/')
      window.toastr.success('Byli jste úspěšně přihlášeni')
    } else {
      window.toastr.warning('Zadané údaje nesedí')
    }
  } catch (err) {
    window.toastr.error('Chyba, zkuste to prosím znovu', 'Error')
  }
},
```

Programový kód 2: Metoda pro přihlášení do systému
Zdroj: autor

8.2.4 Správa kalendáře a událostí

Tato kapitola se zaměřuje na správu kalendáře a událostí pomocí knihovny FullCalendar, (41) která umožňuje zobrazování, vytváření, úpravu a mazání událostí. Kalendář je přizpůsoben potřebám aplikace a integruje se s backendem pro načítání a ukládání dat.

8.2.4.1 Nastavení FullCalendar

Kalendář je konfigurován pomocí objektu calendarOptions, který definuje základní chování a vizuální podobu komponenty:

- Použité pluginy – dayGridPlugin, interactionPlugin a rrulePlugin umožňují zobrazení měsíčního a týdenního rozvrhu, práci s opakujícími se událostmi a interakce s uživatelem.

- Nastavení rozhraní – kalendář obsahuje přizpůsobenou horní lištu, kde jsou k dispozici tlačítka pro změnu zobrazení mezi měsíčním nebo týdenním pohledem, přechod mezi dny a návrat na aktuální datum.
- Jazyková lokalizace – rozhraní kalendáře je lokalizováno do češtiny a první den v týdnu je nastaven na pondělí.
- Responzivita – v závislosti na šířce obrazovky se kalendář přizpůsobuje, například na menších zařízeních se výchozí zobrazení mění z měsíčního na týdenní pro lepší práci s kalendářem.
- Možnosti interakce – uživatelé mohou přidávat nové události, upravovat jejich pozici, ale nemohou měnit jejich délku. Překrývání událostí je zakázáno.
- Načítání a zobrazení událostí – data o událostech jsou načítána z backendu a uložena v poli `'events'`. Události se dynamicky přizpůsobují pomocí přizpůsobení stylu buněk, například jejich obarvením.

8.2.4.2 Správa událostí v kalendáři

Následující podkapitola se zaměřuje na manipulaci s událostmi, jejich přidávání, mazání, úpravu a načítání z backendu. Tato funkcionalita je klíčová pro správný chod systému plánování služeb, protože umožňuje uživatelům efektivně organizovat svůj rozvrh a sdílet data s backendovým serverem.

Přidávání událostí

Proces přidání nové události do kalendáře začíná výběrem datumu, na které chce uživatel zapsat svou službu. Uživatel může označit maximálně jedno datum najednou. Po výběru se automaticky nastaví výchozí název události. Každá vytvořená událost obsahuje unikátní identifikátor, název, čas začátku a konce, ID uživatele pro určení vlastnictví a parametr rrule, který je při běžné události nastaven na hodnotu null.

Systém ověřuje, zda je uživatel přihlášen a má přiřazené ID. Pokud uživatel zvolí možnost měsíčního opakování směn, vytvoří se opakovaná událost pomocí knihovny RRule (42), která vygeneruje opakování na každý měsíc po dobu jednoho roku. Událost je přidána do kalendáře a současně odeslána na backend pomocí POST požadavku. V případě chyby při odesílání na server je událost z kalendáře odstraněna. (viz. Programový kód 3)

```
let eventObject = {
    id: createEventId(),
    title,
    start: new Date(selectInfo.start),
    end: adjustedEnd,
    userId,
    repeated
  }

  if (repeated) {
    if (!eventObject.rrule) {
      eventObject.rrule = {
        freq: RRule.MONTHLY,
        dtstart: eventObject.start,
        until: new Date(
          new Date(eventObject.start).setFullYear(
            new Date(eventObject.start).getFullYear() + 1
          )
        ),
        bymonthday: [eventObject.start.getDate()]
      }
    }
  }

  const newEvent = calendarApi.addEvent(eventObject)
  this.calendarOptions.events.push(eventObject)
  this.updateDatesWithEvents(eventObject)

  try {
    await axios.post('http://localhost:3001/api/events', {
      id: newEvent.id,
      title: newEvent.title,
      start: eventObject.start.toISOString(),
      end: eventObject.end.toISOString(),
      userId,
      repeated: eventObject.repeated,
      rrule: eventObject.rrule || null
    })
    console.log('Událost byla přidána a poslána na server')
  } catch (error) {
    console.error('Nepodařilo se uložit událost na server', error)
    newEvent.remove()
    this.calendarOptions.events = this.calendarOptions.events.filter(
      (event) => event.id !== eventObject.id
    )
  }
}
```

Programový kód 3: Metoda pro vytváření událostí
Zdroj: autor

Mazání událostí

Mazání událostí umožňuje uživatelům odstranit konkrétní události z kalendáře, a zároveň zajistit synchronizaci se serverem (viz. Programový kód 4). Tento proces probíhá nejdřív tak, že pokud uživatel klikne na událost, kterou chce odstranit, zobrazí se potvrzovací dialog, aby uživatel mohl operaci potvrdit nebo zrušit.

Pokud uživatel akci potvrdí, událost se nejprve odstraní z kalendáře na straně klienta. Lokálně se aktualizuje seznam událostí a provede se odebrání příslušného datumu z označených dnů s událostmi. Následně se odešle požadavek na backend, který provede smazání události z databáze a dojde k načtení uživatelových aktuálních událostí. V případě chyby při mazání na serveru se data opět načtou z backendu, aby se obnovil aktuální stav.

```
if (confirm(`Jste si jistí, že chcete smazat tuto událost? '${clickInfo.event.title}'`)) {
  const eventId = clickInfo.event.id
  clickInfo.event.remove()
  this.calendarOptions.events = this.calendarOptions.events.filter(
    (event) => event.id !== eventId
  )
  this.removeDatesFromEvents(clickInfo.event)

  try {
    await axios.delete(`http://localhost:3001/api/events/${eventId}`)
    if (this.selectedUserId) {
      await this.loadEventsFromBackend(this.selectedUserId)
    } else {
      await this.loadEventsFromBackend()
    }
    console.log('Událost byla odstraněna a změna byla poslána na server')
  } catch (error) {
    console.error('Nepodařilo se odstranit událost uloženou na serveru', error)
    this.loadEventsFromBackend()
  }
}
```

Programový kód 4: Metoda pro mazání událostí
Zdroj: autor

Úprava událostí

Úprava událostí umožňuje uživatelům změnit datum již existujících událostí, přičemž je zajištěna synchronizace s backendem (viz. Programový kód 5). Proces úpravy probíhá tak, že pokud uživatel chce přesunout svoji událost na jiný den, jednoduše na tuto událost klikne a přesune jí na jiné datum. Po provedení změny se odešle požadavek na server, který aktualizuje odpovídající záznam v databázi.

Pokud aktualizace proběhne úspěšně, systém provede úpravu i v lokálním seznamu událostí, přičemž se nejprve odstraní staré datum z označených dnů a následně se aktualizovaná událost znovu zapíše. V případě chyby při komunikaci se serverem se změna vrátí do původního stavu, aby nedošlo k nesrovnalostem mezi klientskou a serverovou verzí dat.

```
try {
  await axios.put(`http://localhost:3001/api/events/${changeInfo.event.id}`, {
    id: changeInfo.event.id,
    title: changeInfo.event.title,
    start: changeInfo.event.start,
    end: changeInfo.event.end
  })
  console.log('Událost byla úspěšně aktualizována a poslána na server')

  const updatedEvent = this.calendarOptions.events.find(
    (event) => event.id === changeInfo.event.id
  )
  this.removeDatesFromEvents(updatedEvent)
  if (updatedEvent) {
    updatedEvent.start = changeInfo.event.start
    updatedEvent.end = changeInfo.event.end
  }
  this.updateDatesWithEvents(updatedEvent)
  this.handleEventMouseLeave()
} catch (error) {
  console.log('Update události se nepodařil', error)
  changeInfo.revert()
}
```

Programový kód 5: Metoda pro změnu událostí
Zdroj: autor

Načítání událostí z backendu

Tato metoda zajišťuje získání a zpracování událostí z backendu a jejich následné zobrazení v kalendáři (viz. Programový kód 6). Metoda provádí asynchronní požadavek na backendový server k získání seznamu všech událostí. Server vrací odpověď obsahující pole událostí. Pokud odpověď není validní, zobrazí se chybová hláška.

Nejprve se vymažou předchozí data o termínech obsahujících události, každá událost je zpracována a přizpůsobena pro zobrazení v kalendáři. Pokud je událost přiřazena aktuálně přihlášenému uživateli, je přidána přímo s identifikací `'userId'`. Pokud patří jinému uživateli v rámci stejné skupiny, je označena speciální třídou a nastavena jako needitovatelná. Po úspěšném načtení a zpracování dat jsou události přiřazeny do kalendáře.

```
try {
  let apiUrl = 'http://localhost:3001/api/events/all'
  const response = await axios.get(apiUrl)

  if (!response.data || !response.data.events) {
    console.error('Neplatná odpověď od serveru:', response.data)
    return
  }

  this.datesWithEvents = []

  const transformedEvents = response.data.events.map((event) => {
    this.updateDatesWithEvents(event)

    if (event.userId === effectiveUserId) {
      return {
        ...event,
        extendedProps: {
          userId: event.userId
        },
      }
    } else {
      return {
        ...event,
        classNames: ['group-member-event'],
        editable: false
      }
    }
  })
  this.calendarOptions.events = transformedEvents
  console.log('Události byli načteny ze serveru')
} catch (error) {
  console.error('Nepodařilo se načíst události ze serveru', error)
}
```

Programový kód 6: Metoda pro načtení událostí z backendu
Zdroj: autor

8.2.4.3 Pravidla a uživatelské preference rozvrhu

Správa rozvrhu služeb podléhá několika pravidlům, která zajišťují spravedlivé rozdělení směn a efektivní plánování. Kromě systémových omezení má uživatel možnost nastavit si vlastní preference ovlivňující vytváření událostí.

Pravidlo uzávěrky

Prvním hlavním pravidlem rozvrhu je, že události na daný měsíc lze zadávat pouze do uzávěrky, což je 15. den předchozího měsíce. Po tomto datu již nelze měnit či přidávat nové události pro daný měsíc ani předešlé.

Kontroluje se, zda je možné vytvořit novou událost nebo s danými událostmi nějak manipulovat na základě aktuálního datumu a stanovené uzávěrky (viz. Programový kód 7). Události v aktuálním měsíci lze plánovat pouze do 15. dne předchozího měsíce. Po uzávěrce již není možné vytvářet nové události ani upravovat stávající v daném

měsíci. Tedy pro následující měsíc lze přidávat a upravovat události pouze do 15. dne aktuálního měsíce.

Tato pravidla jsou implementována pomocí kontroly nad výběrem datumu při vytváření nové události nebo jejich úpravou či mazání. Systém nejprve určí aktuální datum a na základě toho nastaví hranice platnosti plánování. Pokud se uživatel pokusí přidat událost mimo povolené období, je operace zrušena a zobrazí se varovná zpráva.

```
const { start } = selectInfo
const today = new Date()
const cutoffDate = new Date(today.getFullYear(), today.getMonth(), 15)
const firstDayOfCurrentMonth = new Date(today.getFullYear(), today.getMonth(), 1)
const lastDayOfCurrentMonth = new Date(today.getFullYear(), today.getMonth() + 1, 0, 23, 59, 59
)

const isForPastOrCurrentMonth =
  start < firstDayOfCurrentMonth ||
  (start >= firstDayOfCurrentMonth && start <= lastDayOfCurrentMonth && today < cutoffDate) ||
  (start >= firstDayOfCurrentMonth && start <= lastDayOfCurrentMonth && today > cutoffDate)

if (isForPastOrCurrentMonth) {
  showToastAtCursorWarningAddPrev()
  selectInfo.view.calendar.unselect()
  return
}

const firstDayOfNextMonth = new Date(today.getFullYear(), today.getMonth() + 1, 1)
const lastDayOfNextMonth = new Date(today.getFullYear(), today.getMonth() + 2, 0)

const isForNextMonth = start >= firstDayOfNextMonth && start <= lastDayOfNextMonth
if (isForNextMonth && today > cutoffDate) {
  showToastAtCursorWarningAddAfter()
  selectInfo.view.calendar.unselect()
  return
}
```

Programový kód 7: Pravidlo uzávěrky
Zdroj: autor

Omezení překrývání a zákaz sousedních událostí

V rámci správy událostí kalendáře bylo nutné implementovat mechanismus, který zabraňuje překrývání událostí a zároveň umožňuje uživatelům nastavit preferenci zákazu dvou po sobě jdoucích událostí (viz. Programový kód 8).

Systém při každé operaci s událostí ověřuje, zda její časový rozsah nekoliduje s jinými událostmi ve stejné uživatelské skupině. Pokud by došlo k překrytí, operace je zrušena a uživatel je upozorněn. Toto pravidlo zajišťuje, že služby se nepřekrývají a rozvrh zůstává přehledný.

Uživatelé mají také možnost aktivovat preferenci, která jim zakazuje plánovat dvě události ve dvou po sobě jdoucích dnech. Pokud je tato možnost zapnutá, systém kontroluje, zda nový termín není bezprostředně před nebo za již existující událostí daného uživatele. Pokud ano, vytvoření události je zablokováno a uživatel je upozorněn.

```
const hasOverlap = this.calendarOptions.events.some((event) => {
  const eventStart = new Date(event.start)
  const eventEnd = new Date(event.end)
  const eventUserId = event.extendedProps?.userId || null
  const currentUserId = userId

  const isOverlapping = start < eventEnd && end > eventStart

  let isAdjacentBlocked = false
  if (this.adjacent) {
    if (eventUserId === currentUserId && currentUserId !== null) {
      const prevDay = new Date(start)
      prevDay.setDate(prevDay.getDate() - 1)

      const nextDay = new Date(start)
      nextDay.setDate(nextDay.getDate() + 1)

      const isAdjacent =
        eventStart.getTime() === prevDay.getTime() ||
        eventStart.getTime() === nextDay.getTime()

      if (isAdjacent) {
        isAdjacentBlocked = true
      }
    }
  }
  return isOverlapping || isAdjacentBlocked
})

if (hasOverlap) {
  showToastAtCursorOverlap()
  selectInfo.view.calendar.unselect()
  return
}
```

Programový kód 8: Omezení překrývání a sousedních událostí
Zdroj: autor

Omezení počtu víkendových dnů a svátků

Uživatelé mají možnost aktivovat preferenci, která omezuje počet víkendových směn a svátků v jejich rozvrhu na maximálně dva dny za jeden měsíc. Toto pravidlo slouží k rovnoměrnému rozdělení méně populárních služeb mezi všechny členy týmu a zajišťuje férové plánování (viz. Programový kód 9).

Při každé operaci s událostí systém ověřuje, zda vybraný den spadá do víkendu nebo je státním svátkem. Následně se provede kontrola již existujících událostí v daném měsíci a spočítá se počet víkendových a svátečních směn daného uživatele. Pokud je tato preference aktivována a nový termín by překročil limit dvou víkendových nebo svátečních směn v měsíci, vytvoření události je zablokováno a uživatel je o této skutečnosti upozorněn.

```
const currentMonth = start.getMonth()
const currentYear = start.getFullYear()
let weekendHolidayEventsCount = 0
const currentUserId = userId
const newEventDate = start.toISOString().split('T')[0]
const newEventDay = start.getDay()

const holidayBaseDates = getHolidays()

this.calendarOptions.events.forEach((event) => {
  const eventStart = new Date(event.start)
  const eventUserId = event.extendedProps?.userId || null
  const eventDateStr = eventStart.toISOString().split('T')[0]
  const eventDay = eventStart.getDay()

  if (
    eventUserId === currentUserId &&
    eventStart.getMonth() === currentMonth &&
    eventStart.getFullYear() === currentYear &&
    (eventDay === 0 || eventDay === 6 || holidayBaseDates.has(eventDateStr))
  ) {
    weekendHolidayEventsCount++
  }
})

if (
  this.holiday &&
  (newEventDay === 0 || newEventDay === 6 || holidayBaseDates.has(newEventDate))
) {
  weekendHolidayEventsCount++

  if (weekendHolidayEventsCount > 2) {
    showToastAtCursorHoliday()
    selectInfo.view.calendar.unselect()
    return
  }
}
```

Programový kód 9: Omezení počtu víkendů a svátků
Zdroj: autor

8.2.4.4 Správa uživatelů a autentizace

Tato podkapitola se zaměřuje na metody sloužící k získávání uživatelských dat, ověřování identity uživatele a určování jeho oprávnění v systému. Aplikace rozlišuje mezi běžnými uživateli a administrátory, čímž umožňuje řízení přístupových práv a

funkcionalit dostupných jednotlivým uživatelům. Pro správné fungování tohoto systému slouží dvě klíčové metody.

Jednoduchá metoda ‘fetchUsers()‘ slouží k získání seznamu všech uživatelů uložených v databázi. Požadavek je odeslán na backend pomocí ‘axios.get()‘, přičemž vrácená data jsou uložena do proměnné ‘users‘. Tyto informace jsou následně využívány například při přiřazování událostí jednotlivým uživatelům nebo při zobrazení nápovědy u jednotlivých událostí v kalendáři.

Dále metoda ‘mounted()‘ je volána okamžitě po načtení stránky a zajišťuje ověření přihlášení uživatele a jeho případného administrátorského oprávnění. Nejprve se z localStorage získá přístupový token a uživatelské jméno. Pokud je token k dispozici, aplikace odešle požadavek na backend, kde získá seznam administrátorů. Následně se kontroluje, zda je přihlášený uživatel mezi nimi, a pokud ano, nastaví se proměnná ‘isAdmin‘ na true.

8.2.5 Automatizace plánování

Automatizace plánování v kalendářním systému umožňuje efektivní rozvrhování služeb zdravotnického personálu bez nutnosti manuálního zadávání každé události zvlášť. Tento proces využívá předem definovaná pravidla a uživatelské preference k optimalizaci rozpisu směn tak, aby byly dodrženy stanovené požadavky, jako je minimální počet směn za měsíc, omezení sousedních směn či preference volných dnů.

Hlavním prvkem automatizace je metoda ‘autoFillEvents()‘, která slouží k automatickému generování událostí v případě, že uživatel nemá v kalendáři dostatečný počet naplánovaných služeb. Tato metoda zajišťuje, že bude vždy splněn minimální požadovaný počet událostí v daném měsíci (viz. Programový kód 10).

Po spuštění metoda nejprve načte identifikátor aktuálního uživatele a časové rozmezí pro generování událostí. Následně získá seznam všech existujících událostí uživatele z backendu a filtruje je podle aktuálního měsíce. Na základě výsledku určí, kolik událostí je ještě potřeba doplnit. Poté začne postupně procházet jednotlivé dny v měsíci a vytvářet nové události až do dosažení požadovaného počtu. Každá nová událost obsahuje základní informace, jako je unikátní identifikátor, název, čas začátku a konce a uživatelské ID. Po vygenerování jsou události odeslány na backend a kalendář se

automaticky aktualizuje. Pokud je dosažen maximální limit generace, systém uživatele informuje.

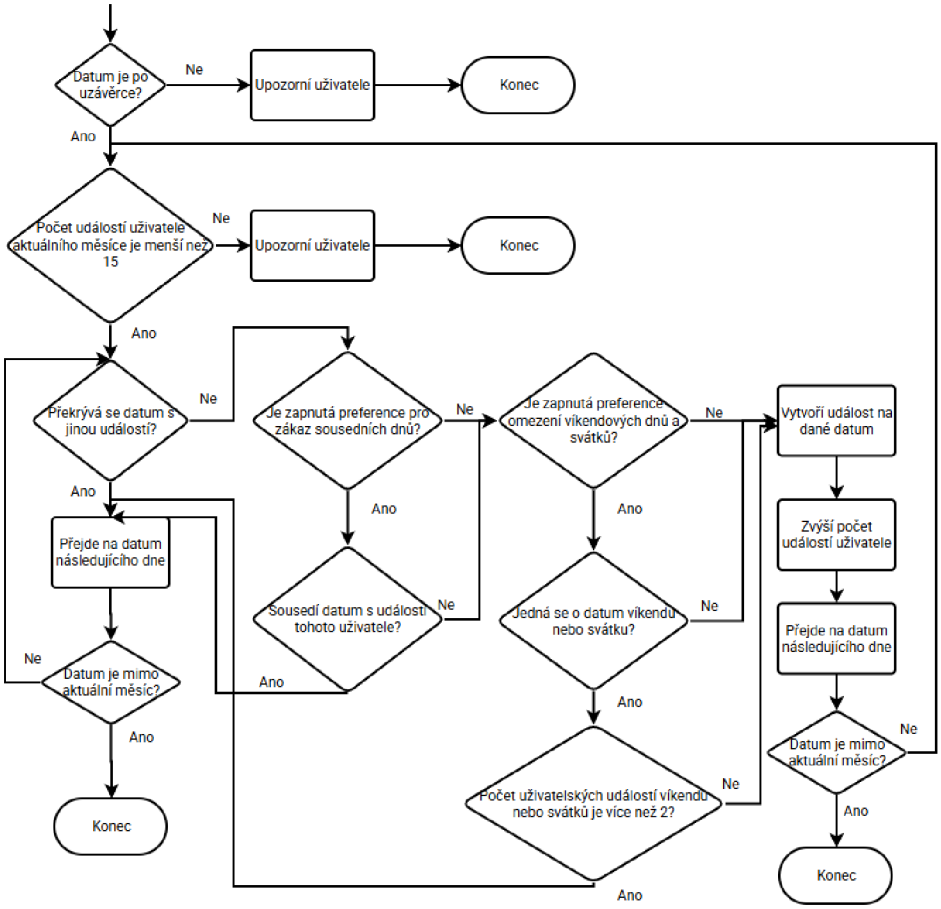
Pro znázornění byl vytvořen vývojový diagram, který popisuje hlavní procesy algoritmu během vytváření událostí při využití automatizovaného plánování (viz. Obrázek 9 a Obrázek 10).

Automatizovaný systém rovněž zohledňuje všechna pravidla a omezení definovaná v předchozí kapitole „Pravidla a uživatelské preference rozvrhu“. To zahrnuje například prevenci překrývání událostí, tedy zamezení, aby měli službu zároveň dva lékaři ze stejné skupiny, respektování preferencí ohledně volných dnů a omezení sousedních směn. Pro zjednodušení odpovídající kód pouze znázorňuje základní algoritmus pro generaci událostí. Výsledkem je inteligentní plánovací mechanismus, který usnadňuje správu rozvrhu a minimalizuje nutnost ručních zásahů.

```
async autoFillEvents() {
  const userId = localStorage.getItem('userId')
  const firstDayOfMonth = new Date(localStorage.getItem('firstViewedDate'))
  const lastDayOfMonth = new Date(localStorage.getItem('lastViewedDate'))

  try {
    const response = await axios.get(`http://localhost:3001/api/events/?userId=${userId}`)
    const allEvents = Array.isArray(response.data) ? response.data : response.data.events

    const userEvents = allEvents.filter((event) => {
      const belongstToUser = event.userId === userId
      const eventDate = new Date(event.start)
      return belongstToUser && eventDate >= firstDayOfMonth && eventDate <= lastDayOfMonth
    })

    const missingEventsCount = 15 - userEvents.length
    if (missingEventsCount <= 0) {
      showToastAtCursorWarningAutoFillFull2()
      return
    }

    let newEvents = []
    let dayIterator = new Date(firstDayOfMonth)

    while (newEvents.length < missingEventsCount && dayIterator <= lastDayOfMonth) {
      dayIterator.setHours(0, 0, 0, 0)

      const startDate = new Date(dayIterator)
      const endDate = new Date(dayIterator.setHours(23, 59, 59, 0))

          newEvents.push({
            id: createEventId(),
            title: 'Aut. služba',
            start: startDate,
            end: endDate,
            userId: userId,
            repeated: false,
            rrule: null
          })
      dayIterator.setDate(dayIterator.getDate() + 1)
    }

    if (newEvents.length === 0) {
      showToastAtCursorWarningAutoFillFull()
    }
```

Programový kód 10: Algoritmus automatizace událostí
Zdroj: autor

Obrázek 10: Vývojový diagram automatizace plánování 1
Zdroj: autor



Obrázek 9: Vývojový diagram automatizace plánování 2
Zdroj: autor

8.2.6 Řešení skupin

Správa skupin umožňuje efektivnější organizaci uživatelů a jejich rozvrhů. Skupiny slouží k seskupování jednotlivých uživatelů a oddělování jejich rozvrhů. Každá skupina může obsahovat až 2 uživatele, kteří následně sdílí stejný rozvrh pro dané měsíce. Tedy pouze jeden uživatel stejné skupiny si může zapsat konkrétní den.

Tato podkapitola se zaměřuje na implementaci funkcionalit souvisejících se skupinami, včetně jejich definice a načítání dat z backendu. Dále bude popsán proces vytváření, mazání skupin a správa jejich uživatelů. Součástí budou také kontrolní mechanismy, které zajistí, že skupiny jsou správně nakonfigurované.

8.2.6.1 Definice skupin

V této části jsou skupiny definovány a je popsáno seskupování jejich uživatelů do předem definovaných kategorií.

Metoda `‘created()‘` je volána při vytvoření komponenty a zajišťuje načtení existujících skupin z localStorage a získání uživatelů z backendu pomocí metody `‘fetchUsers()‘` kde jsou následně uživatelé načteni do proměnné `‘users‘`. Pokud skupiny v úložišti nejsou, inicializuje se výchozí sada skupin, která zahrnuje skupinu "Nezařazení", kde jsou shromažďováni uživatelé, kteří nejsou přiřazeni k žádné skupině a zároveň jsou načteny tři předdefinované prázdné skupiny.

Pomocná metoda `‘getUsersByGroup(group)‘` vrací seznam uživatelů přiřazených k dané skupině. Filtruje uživatele podle jejich atributu `‘group‘` a umožňuje zobrazit správné členy v jednotlivých skupinách.

8.2.6.2 Správa skupin

Správa skupin je klíčovou funkcionalitou, která umožňuje administrátorovi dynamicky vytvářet a mazat jednotlivé skupiny (viz. Programový kód 11). Pokud je skupina odstraněna a obsahuje uživatele, jsou automaticky přesunuti do výchozí skupiny "Nezařazení".

Vytváření nové skupiny umožňuje metoda `‘createGroup()‘` proces probíhá v několika krocích. Nejprve se zkontrolují všechna existující ID skupin, algoritmus

následně hledá nejnižší volné číslo v posloupnosti ID skupin, aby se zajistilo, že skupiny mají vždy souvislou číselnou řadu. Jakmile je nalezeno volné ID, vytvoří se nový objekt reprezentující skupinu s názvem "Skupina X", kde X odpovídá danému ID. Nová skupina se přidá do seznamu skupin, který je následně seřazen podle ID, a nakonec se aktualizovaná struktura uloží do localStorage, aby byla zachována.

Mazání existující skupiny řeší metoda `'deleteGroup(groupId)'` kde při smazání se provádějí jednotlivé kroky. Nejdřív se zjistí, kteří uživatelé jsou aktuálně přiřazeni ke skupině, která má být odstraněna. Každému uživateli je změněna hodnota atributu `'group'` na 0, což označuje nezařazené uživatele. Aktualizace se provádí nejen lokálně, ale změna je zároveň odeslána na backend pomocí metody `'updateUserGroup()'`, která bude popsána v rámci následující kapitoly. Skupina je tedy odstraněna ze seznamu, změna je uložena do localStorage a administrátor je informován o úspěšném odstranění.

```
createGroup() {
  const existingIds = this.groups
    .filter((group) => group.id !== '0')
    .map((group) => parseInt(group.id))

  let newGroupId = 1
  while (existingIds.includes(newGroupId)) {
    newGroupId++
  }

  const newGroup = { id: newGroupId.toString(), name: `Skupina ${newGroupId}` }
  this.groups.push(newGroup)

  this.groups.sort((a, b) => parseInt(a.id) - parseInt(b.id))
  localStorage.setItem('groups', JSON.stringify(this.groups))
  window.toastr.success('Nová skupina byla úspěšně vytvořena!')
},
async deleteGroup(groupId) {
  try {
    const usersToMove = this.users.filter((user) => user.group === groupId)

    for (const user of usersToMove) {
      user.group = '0'
      await this.updateUserGroup(user._id, '0')
    }
    this.groups = this.groups.filter((group) => group.id !== groupId)
    localStorage.setItem('groups', JSON.stringify(this.groups))
    window.toastr.success(`Úspěšně jste smazali skupinu ${groupId}!`)
  } catch (error) {
    window.toastr.error('Naskytla se chyba během mazání skupin a aktualizace jejich uživatelů')
    console.error('Naskytla se chyba během mazání skupin a aktualizace jejich uživatelů', error)
  }
},
```

Programový kód 11: Metody pro správu skupin
Zdroj: autor

8.2.6.3 Manipulace s uživateli ve skupinách

Tato podkapitola se zaměřuje na metody umožňující administrátorům spravovat členství uživatelů ve skupinách. Hlavní funkcionalitou je přesouvání uživatelů mezi skupinami a aktualizace této změny na backendu. Přesouvání probíhá pomocí drag-and-drop mechanismu a je omezeno pravidly.

Metoda `‘updateUserGroup(userId, newGroup)‘` zajišťuje komunikaci se serverem při změně skupiny uživatele. Využívá HTTP požadavek PUT, který aktualizuje atribut skupiny konkrétního uživatele na serveru. Po úspěšné aktualizaci se v konzoli zobrazí potvrzení o přesunutí uživatele do nové skupiny, jinak se vypíše varování, pokud odpověď serveru neobsahuje očekávaná data.

8.2.6.4 Kontrola skupin

Tato podkapitola popisuje metody, které zajišťují kontrolu správného zařazení uživatelů do skupin a ověřují možné kolize při jejich přesunu (viz. Programový kód 12). Hlavním cílem je zajistit, aby uživatelé nebyli trvale nezařazeni a aby při přesunu do skupiny nedocházelo ke konfliktům v plánování událostí.

Metoda `‘checkGroup()‘` ověřuje, zda je přihlášený uživatel zařazen ve skupině "Nezařazení". Pokud ano a není administrátorem, zobrazí se mu upozornění, aby byl informován a vyčkal k zařazení do skupiny administrátorem. Pokud je tedy přihlášený uživatel administrátor, metoda navíc kontroluje, zda se v systému nacházejí uživatelé bez přiřazené skupiny. Pokud ano, zobrazí adminovi varování, aby mohl situaci napravit a zařadit uživatele do odpovídajících skupin.

Dalším pravidlem je kontrola kolizí událostí při přesunu uživatele mezi skupinami. Je nutné zkontrolovat, zda v rámci této skupiny nedochází ke kolizi událostí jednotlivých uživatelů. Tento problém řeší následující proces. V rámci metody `‘handleDrop(newGroup)‘` se nejprve získá seznam všech naplánovaných událostí přesouvaného uživatele. Následně se získají události všech uživatelů, kteří jsou již členy cílové skupiny a ověří se, zda některá z událostí přesouvaného uživatele koliduje s událostí jiného člena skupiny ve stejný den. Pokud se zjistí konflikt, zobrazí se adminovi potvrzovací dialog s varováním o kolizi, kde admin může přesun zrušit nebo

potvrdit s tím, že by musel případně tyto události upravit a informovat jednotlivé uživatele.

```
const response = await axios.get(
  `http://localhost:3001/api/events/?userId=${this.draggedUser._id}`
)

const userEvents = response.data.events || []

const groupUserIds = usersInGroup.map((user) => user.id || user._id)
const groupResponse = await axios.get(`http://localhost:3001/api/events/all`)
const allEvents = groupResponse.data.events || []

const groupEvents = allEvents.filter((event) => groupUserIds.includes(event.userId))
const hasConflict = userEvents.some((userEvent) =>
  groupEvents.some(
    (groupEvent) =>
      new Date(userEvent.start).toISOString().slice(0, 10) ===
      new Date(groupEvent.start).toISOString().slice(0, 10)
  )
)

if (hasConflict) {
  if (
    confirm(
      'Při přesunu uživatele do této skupiny dojde ke kolizi vypsaných služeb\nOpravdu chcete pokračovat?'
    )
  ) {
    this.draggedUser.group = groupId
    await this.updateUserGroup(this.draggedUser.id || this.draggedUser._id, groupId)
    this.draggedUser = null
  }
```

Programový kód 12: Pravidlo pro kontrolu konfliktu událostí
Zdroj: autor

8.3 Implementace backendu

Pro zajištění funkcionality systému plánování služeb pro nemocnice je nezbytné vytvořit robustní backendovou část, která bude zajišťovat správu uživatelů, skupin, plánování událostí a jejich synchronizaci s frontendovou aplikací. Backend slouží jako centrální část systému, která zpracovává požadavky od klientské aplikace, spravuje databázi a zajišťuje bezpečnostní aspekty aplikace. Cílem této kapitoly je tedy představit implementaci serverové části systému, která umožňuje efektivní správu nemocničního plánování a zajišťuje bezproblémovou komunikaci mezi uživateli a aplikací.

8.3.1 Architektura backendu

Backend aplikace je navržen jako serverová služba, která poskytuje rozhraní pro manipulaci s uživatelskými daty, správu událostí v kalendáři a ověřování přístupu na základě uživatelských rolí. Klientská aplikace komunikuje s backendem

prostřednictvím REST API (16), přičemž důraz je kladen na bezpečnost, efektivní práci s databází a škálovatelnost řešení.

V rámci implementace backendu byly použity následující technologie a knihovny:

- CORS – umožňuje bezpečnou komunikaci mezi frontendem a backendem, zajišťuje správné nastavení oprávnění pro přístup k API (47).
- simple-encryptor – knihovna pro šifrování citlivých dat, využívaná pro bezpečné uložení určitých uživatelských údajů (48).
- jsonwebtoken (JWT) - slouží k autentizaci uživatelů prostřednictvím tokenů, čímž zajišťuje bezpečný přístup k chráněným částem systému (17).
- rrule – knihovna pro práci s opakujícími se událostmi, která umožňuje efektivní generování a správu cyklických záznamů v kalendáři (42).
- mongoose – knihovna modelování objektových dat pro MongoDB, která usnadňuje práci s databázovými modely a zajišťuje validaci dat (49).

8.3.2 Návrh databázových modelů

Databázové modely definují strukturu a vztahy mezi daty v systému. Backend aplikace využívá MongoDB (18) ve spojení s Mongoose (49), což umožňuje efektivní správu dat pomocí schémat a zajišťuje validaci a referenční integritu. V rámci systému jsou definovány tři hlavní modely:

8.3.2.1 Model uživatele

Model uživatele uchovává informace o jednotlivých zaměstnancích v systému, jejich přihlašovací údaje, role a preference pro plánování směn. Model se využívá k autentizaci uživatelů, správě jejich rolí a skupinového rozdělení. Každý uživatel je definován následujícími atributy:

- username (String, povinné) – uživatelské jméno.
- email (String, povinné) – e-mailová adresa uživatele.
- password (String, povinné) – heslo uživatele uložené v hashované podobě.
- roles (ObjectId, ref: "Role") – odkaz na roli uživatele v systému.
- group (String, povinné) – identifikátor skupiny, ke které uživatel patří.

- preference_holiday (Boolean, povinné) – preference uživatele pro zapisování maximálního počtu svátků nebo víkendů za měsíc.
- preference_adjacent (Boolean, povinné) – preference uživatele pro zákaz zapisování dvou po sobě jdoucích dnů.

8.3.2.2 Model událostí

Model událostí ukládá informace o jednotlivých směnách v kalendáři, včetně možnosti opakování. Používá se k plánování a evidenci směn a pomocí pravidel rrule (42) lze definovat opakující se směny. Každá událost je definována následujícími atributy:

- id (String) – unikátní identifikátor událostí, generovaný pomocí uuid (45).
- title (String) – název události.
- start (Date) – datum a čas začátků směny.
- end (Date) – datum a čas konce směny.
- repeated (Boolean) – označuje, zda se událost opakuje.
- userId (String) – identifikátor uživatele, kterému je směna přiřazena.
- rrule (Object) – objekt specifikující pravidla opakování událostí.

8.3.2.3 Model rolí

Model rolí definuje uživatelské oprávnění v rámci systému. Role se využívají k řízení přístupu k různým funkcím systému. Například pouze uživatelé s rolí admin mohou spravovat skupiny nebo plánovat směny ostatním zaměstnancům. Struktura modelu je definována jedním atributem:

- name (String) – název role

8.3.3 Operace s databází

Databázová vrstva backendu slouží ke správě uživatelů, jejich přihlášení a autorizaci v systému. Implementace využívá MongoDB Atlas (50), cloudovou verzi NoSQL databáze MongoDB, která umožňuje škálovatelnost a snadnou integraci s backendovým serverem.

8.3.3.1 Připojení k databázi a inicializace

Pro připojení k databázi je využit knihovní balíček Mongoose, který usnadňuje práci s MongoDB v Node.js. Připojení k databázi probíhá asynchronně, aby bylo možné efektivně zachytit případné chyby a zajistit stabilitu aplikace.

Po úspěšném připojení je spuštěna funkce `'initial()'`, která zajistí inicializaci základních dat v databázi. Kontroluje, zda jsou v databázi definovány základní role. Pokud nejsou nalezeny, jsou automaticky vytvořeny role admin a user, a zároveň je administrátorovi přidělen výchozí účet.

8.3.3.2 Databázové operace

Backend poskytuje několik klíčových operací nad databází, zejména pro správu uživatelů tedy jejich registraci a přihlášení.

Registrace uživatele probíhá pomocí metody `'createUserDBService'` (viz. Programový kód 13), tato metoda vytvoří nový objekt uživatele a naplní jeho atributy a pomocí knihovny simple-encryptor zahashuje heslo. Následně nastaví výchozí skupinu a uživatelské preference, přiřadí uživateli roli, pokud uživatel specifikuje roli, vyhledá jí v databázi, jinak mu přiřadí základní roli user, a nakonec uloží uživatele do databáze.

```
module.exports.createUserDBService = async (userDetails) => {
    try {
        var userModelData = new userModel();

        userModelData.username = userDetails.username;
        userModelData.email = userDetails.email;
        userModelData.password = userDetails.password;
        var encrypted = encryptor.encrypt(userDetails.password);
        userModelData.password = encrypted;
        userModelData.group = "0";
        userModelData.preference_holiday = false;
        userModelData.preference_adjacent = false;

        if (userDetails.roles) {
            const roles = await Role.find({ name: { $in: userDetails.roles } });
            userModelData.roles = roles.map((role) => role._id);
        } else {
            const role = await Role.findOne({ name: "user" });
            userModelData.roles = [role._id];
        }

        await userModelData.save();
        return true;
    } catch (error) {
        return false;
```

Programový kód 13: Metoda pro registraci uživatele
Zdroj: autor

Pro přihlášení uživatele je třeba ověřit, zda zadané údaje odpovídají záznamu v databázi. Proces přihlášení probíhá v několika krocích (viz. Programový kód 14).

Prvně se na základě uživatelského jména vyhledá odpovídající záznam v databázi, následně je heslo uložené v databázi dešifrováno a porovnáno se zadaným heslem. Pokud jsou údaje správné, je uživateli vygenerován JWT token pro autorizaci dalších požadavků v systému. Pokud ověření proběhne úspěšně, uživatel obdrží potvrzení a svůj autentizační token.

```
module.exports.loginUserDBService = async (employeeDetails) => {
    try {
        const user = await userModel
            .findOne({
                username: employeeDetails.username,
            })
            .populate("roles", "-__v");

        if (user) {
            const decrypted = encryptor.decrypt(user.password);
            if (decrypted === employeeDetails.password) {
                const token = jwt.sign({ id: user._id }, config.secret, {
                    algorithm: "HS256",
                    expiresIn: 86400,
                });
                return {
                    status: true,
                    msg: "Uživatel úspěšně ověřen!",
                    token: token,
                    user: { id: user._id, username: user.username, email: user.email },
                };
            } else {
                return { status: false, msg: "Ověření uživatele selhalo" };
            }
        } else {
            return { status: false, msg: "Uživatel nenalezen" };
        }
    } catch (error) {
        return { status: false, msg: "Neplatné údaje, došlo k chybě" };
    }
};
```

Programový kód 14: Metoda pro přihlášení uživatele
Zdroj: autor

8.3.4 Kontrolery

Kontrolery slouží jako prostředník mezi klientskou aplikací a databází. Zpracovávají HTTP požadavky, volají odpovídající služby a vrací odpovědi klientovi. Tato kapitola popisuje tři hlavní kontrolery systému.

8.3.4.1 Kontroler správy uživatelů

UserController je zodpovědný za operace související s uživateli, jako je registrace, přihlášení, získávání informací o uživatelích a úprava jejich nastavení. Klíčovou součást backendové logiky tvoří metody, které zajišťují interakci mezi uživateli a systémem. Každý uživatel má možnost nastavit si vlastní preference, které jsou ukládány v databázovém modelu User a lze je kdykoli načíst nebo aktualizovat. Správa preferencí zahrnuje operace jejich načtení a následné aktualizace.

Následující dvě metody slouží k získávání uživatelů z databáze, pro jejich správu je důležité umožnit přístup k informacím o uživatelích. První metodou je získání všech uživatelů a druhá získává pouze uživatele podle skupiny.

Metoda `‘fetchAllUsers‘` vrací seznam všech uživatelů uložených v databázi. Používá k tomu Mongoose metodu `‘.find()‘`, která vyhledá všechny záznamy v kolekci `‘User‘` a vrátí je jako JSON odpověď.

Metoda `‘fetchUser‘` umožňuje načíst všechny uživatele patřící do určité skupiny na základě `‘groupId‘`.`‘User.find({ group: groupId })‘` vrátí všechny uživatele, kteří mají atribut `‘group‘` rovný `‘groupId‘` a následně nalezení uživatelé se odešlou zpět klientovi.

8.3.4.2 Kontroler správy událostí

EventController zajišťuje správu událostí v systému. Obsahuje metody pro uložení nové události, úpravu existující události, smazání události a získání všech událostí uživatele.

Metoda `‘saveNewEvent‘` umožňuje uživateli uložit novou událost do databáze. Události mohou být jednorázové nebo opakované (viz. Programový kód 15). Prvně metoda získá potřebná data, převede datumy na Date objekty, uloží novou událost pomocí `‘event.save()‘` a API nakonec vrátí vytvořenou událost.

```
var saveNewEvent = async (req, res) => {
    try {
        const { id, userId, title, start, end, repeated, rrule } = req.body;

        const startDate = new Date(start);
        const endDate = new Date(end);

        const event = new Event({
            id,
            title,
            start: startDate,
            end: endDate,
            repeated,
            userId,
            rrule: rrule || null,
        });
        await event.save();
        res.send(event);
    } catch (error) {
        res.send(error);
    }
};
```

Programový kód 15: Metoda pro uložení události do databáze
Zdroj: autor

Další dvě metody slouží k změně a smazání událostí v rámci databáze (viz. Programový kód 16). Metoda changeEvent dovoluje uživateli změnit detaily již existující události a pomocí metody deleteEvent je možné smazat konkrétní událost.

```
var deleteEvent = async (req, res) => {
    try {
        await Event.findOneAndDelete({ id: req.params.id });
        res.send("Event byl úspěšně smazán");
    } catch (error) {
        res.send(error);
    }
};

var changeEvent = async (req, res) => {
    try {
        const event = await Event.findOneAndUpdate(
            { id: req.params.id },
            req.body,
            { new: true },
        );
        res.send(event);
    } catch (error) {
        res.send(error);
    }
};
```

Programový kód 16: Metody pro smazání a změnu událostí
Zdroj: autor

Metoda `'getAllEvents'` umožňuje uživateli zobrazit všechny události v rámci jeho uživatelské skupiny (viz. Programový kód 17). Nejprve proběhne ověření, zda bylo předáno `'userId'` a zda daný uživatel skutečně existuje. Následně metoda získá všechny uživatele patřící do stejné skupiny a načte události přiřazené těmto uživatelům. V případě, že událost obsahuje pravidlo rrule, jsou její výskyty vygenerovány na období jednoho roku dopředu. Na závěr API vrací seznam všech událostí.

```
var getAllEvents = async (req, res) => {
    try {
        const { userId } = req.query;
        const currentUser = await User.findById(userId).exec();
        if (!currentUser) {
            return res.send({ message: "Uživatel nenalezen" });
        }
        const usersInGroup = await User.find({ group: currentUser.group }).exec();
        const userIdsInGroup = usersInGroup.map((user) => user._id);

        const events = await Event.find({ userId: { $in: userIdsInGroup } }).exec();

        const expandedEvents = events.flatMap((event) => {
            if (event.rrule && event.start && event.end) {
                const rule = new RRule({
                    ...event.rrule,
                    dtstart: new Date(event.start),
                });

                const startDate = new Date(event.start);
                const endDate = new Date(
                    new Date().setFullYear(new Date().getFullYear() + 1),
                );
                const occurrences = rule.between(startDate, endDate, true);

                return occurrences.map((date) => ({
                    ...event.toObject(),
                    start: date.toISOString(),
                    end: new Date(date.getTime() + 24 * 60 * 60 * 1000).toISOString(),
                }));
            } else if (event.start && event.end) {
                return {
                    ...event.toObject(),
                    start: new Date(event.start).toISOString(),
                    end: new Date(event.end).toISOString(),
                };
            } else {
                console.warn(
                    `Event with id ${event.id} has missing dates and was skipped.`,
                );
                return [];
            }
        });
        res.send({ events: expandedEvents });
    } catch (error) {
        console.error("Error fetching events:", error);
        res.send(error);
    }
```

Programový kód 17: Metoda pro získání všech událostí z databáze
Zdroj: autor

8.3.4.3 Kontroler autorizace

AuthorizController zajišťuje řízení přístupu k různým částem aplikace na základě oprávnění uživatele. Obsahuje tři metody, které určují, kdo má přístup k určitým funkcionalitám systému.

První dvě metody `‘allAccess‘` a `‘userBoard‘` slouží k rozlišení, zda je uživatel přihlášen. V případě nepřihlášeného uživatele je vrácena odpověď metodou `‘allAccess‘`, zatímco pro přihlášené uživatele je využita metoda `‘userBoard‘`, která vrací specifická data dostupná pouze po přihlášení.

Hlavní metodou kontroleru je `‘adminBoard‘`, která ověřuje, zda má uživatel roli administrátora. Nejprve probíhá kontrola, zda role administrátora existuje v databázi. Pokud ano, metoda vyhledá všechny uživatele s touto rolí a vrací jejich seznam jako odpověď prostřednictvím API.

8.3.5 Middleware pro autentizaci a validaci

Tato podkapitola backendu se zabývá middlewarem, který slouží jako mezivrstva požadavků uživatele a backendovou logikou. Následující část backendu obsahuje dva klíčové middleware soubory autentizace a verifikace.

8.3.5.1 Middleware autentizace

Middleware autentizace slouží k ověřování platnosti tokenu při přístupu k chráněným částem aplikace. Metoda `‘verifyToken‘` zajišťuje, že je uživatelský token platný. Nejprve je token získán z HTTP hlavičky požadavku. V případě, že token není poskytnut, je uživateli vrácena odpověď s odpovídající chybovou zprávou.

Pokud je token přítomen, ověřuje se jeho platnost pomocí metody `‘jwt.verify()‘`, která jej porovná s tajným klíčem uloženým v `‘config.secret‘`. V případě úspěšného ověření middleware předá uživatelské ID do objektu požadavku a zavolá funkci next(), která předá upravené objekty dalšímu middlewaru.

8.3.5.2 Middleware validace

Middleware validace slouží k ověřování uživatelských údajů při registraci, s cílem zabránit duplicitním záznamům a přiřazení neexistujících rolí (viz. Programový kód

18). Následující middleware funkce se se spouštějí před zpracováním požadavků v kontrolerech a zajišťují bezpečnost a správnost vstupních dat. Proces validace zahrnuje metody, které ověřují, zda uživatelské jméno nebo e-mail již nejsou registrovány, a zda role, které mají být uživateli přiřazeny, skutečně existují.

Metoda `‘checkDuplicateUsernameOrEmail‘` vyhledává v databázi uživatele se stejným uživatelským jménem, jaké bylo poskytnuto klientem. Pokud takový uživatel existuje, vrací middleware odpověď s informací, že zadané jméno je již obsazené. Stejným způsobem je kontrolován i zadaný e-mail. V případě, že jsou oba údaje volné, je řízení předáno další funkci prostřednictvím `‘next()‘`.

Metoda `‘checkRolesExisted‘` ověřuje, zda požadované role existují. Pokud pole `‘roles‘` není v požadavku přítomné, je kontrola přeskočena. Každá role se porovná s předem definovaným polem `‘ROLES‘`. Pokud některá z rolí neexistuje, vrací middleware chybovou zprávu. V opačném případě je požadavek předán dál voláním `‘next()‘`.

```
checkDuplicateUsernameOrEmail = async (req, res, next) => {
    try {
        const userByUsername = await User.findOne({
            username: req.body.username,
        }).exec();
        if (userByUsername) {
            return res.send({
                status: false,
                message: "Uživatelské jméno je již zabrané!",
            });
        }

        const userByEmail = await User.findOne({ email: req.body.email }).exec();
        if (userByEmail) {
            return res.send({ status: false, message: "E-mail je již zabrán!" });
        }

        next();
    } catch (err) {
        res.send({ message: err.message });
    }
};

checkRolesExisted = (req, res, next) => {
    if (req.body.roles) {
        for (let i = 0; i < req.body.roles.length; i++) {
            if (!ROLES.includes(req.body.roles[i])) {
                return res.send({
                    message: `Role ${req.body.roles[i]} neexistuje!`,
                });
            }
        }
    }

    next();
};
```

Programový kód 18: Metody pro verifikaci údajů a rolí
Zdroj: autor

8.3.6 Směrování požadavků

Tato kapitola popisuje klíčovou část backendové aplikace, jejímž úkolem je správné směrování požadavků od klienta na příslušné kontrolery, které následně provádějí požadované operace. Směrování je rozděleno do několika hlavních skupin rout, přičemž každá skupina se zaměřuje na specifickou oblast funkcionality systému.

8.3.6.1 Autentizace a autorizace uživatelů

Následující routery se starají o správu uživatelských účtů, přihlašování a ověřování přístupu. Registrace nového uživatele probíhá prostřednictvím požadavku na adresu `'/register'`, který nejprve ověřuje, zda zadané uživatelské jméno či e-mail již neexistují, následně kontroluje existenci požadovaných rolí a poté provede samotnou registraci. Přihlášení se realizuje pomocí požadavku na `'/login'`, kde kontroler ověřuje zadané přihlašovací údaje a v případě úspěchu vrací přístupový token. K ověření platnosti tokenu slouží požadavek na `'/auth'`, který zajišťuje, že uživatel má oprávnění k provádění požadovaných operací.

```
router.post("/register", [
    verifyRegister.checkDuplicateUsernameOrEmail,
    verifyRegister.checkRolesExisted,
    userController.registerUserController,
]);
router.post("/login", userController.loginUserController);
router.post("/auth", authJwt.verifyToken);
```

8.3.6.2 Zpracování obsahu podle role

Další skupina směrování se zaměřuje na zobrazování obsahu na základě uživatelských rolí. Přístupný obsah pro všechny uživatele, včetně nepřihlášených, zajišťuje `'/all'`, který vrací obecné informace. Pro přihlášené uživatele slouží požadavek `'/user'`, jenž poskytuje specifický obsah přizpůsobený jejich oprávněním. Administrátorský přístup je realizován prostřednictvím endpointu `'/admin'`, který vrací seznam všech administrátorů v systému a zpřístupňuje obsah určený pouze pro tuto roli.

```
router.post("/all", controller.allAccess);
```

```
router.post("/user", controller.userBoard);
router.post("/admin", controller.adminBoard);
```

8.3.6.3 CRUD operace pro události kalendáře

Tato podkapitola popisuje skupinu směrování, která slouží ke správě událostí v kalendáři. Nové události lze přidávat prostřednictvím ‘/api/events‘, který umožňuje uložení jedné události do databáze, nebo pomocí ‘/api/events/array‘, jenž umožňuje hromadné přidání více událostí najednou. Pro získání událostí slouží ‘/api/events‘, který vrací seznam událostí přihlášeného uživatele a jeho skupiny. Pokud je potřeba načíst všechny události bez ohledu na konkrétního uživatele, lze využít ‘/api/events/all‘. Správa událostí dále zahrnuje možnost jejich mazání prostřednictvím ‘/api/events/:id‘ a úpravu existujících událostí pomocí téhož endpointu.

```
router.post("/api/events", eventController.saveNewEvent);
router.post("/api/events/array",
eventController.saveNewEventArray);
router.get("/api/events", eventController.getAllEvents);
router.get("/api/events/all", eventController.getAllEventsPure);
router.delete("/api/events/:id", eventController.deleteEvent);
router.put("/api/events/:id", eventController.changeEvent);
```

8.3.6.4 Správa skupin a uživatelů

Tato část směrování se zaměřuje na správu uživatelů, jejich zařazení do skupin a správu uživatelských preferencí. Požadavek na endpoint ‘/api/users‘ vrací seznam všech uživatelů v systému, zatímco ‘/api/users/group/:groupId‘ umožňuje načíst uživatele patřící do konkrétní skupiny. V případě potřeby změny skupiny, do níž je uživatel přiřazen, lze využít ‘/api/users/:id‘, který umožňuje aktualizaci této informace. Preference konkrétního uživatele lze načíst prostřednictvím požadavku na ‘/api/user/preferences/:userId‘. Pokud uživatel potřebuje své preference upravit, může je aktualizovat pomocí stejného endpointu.

```
router.get("/api/users", userController.fetchAllUsers);
router.get("/api/users/group/:groupId", userController.fetchUser);
router.put("/api/users/:id", userController.updateGroup);
router.get("/api/user/preferences/:userId",
userController.getPreference);
router.put("/api/user/preferences/:userId",
userController.updatePreference);
```

8.4 Testování aplikace

Testování aplikace představuje klíčovou fázi vývojového procesu, jejímž cílem je odhalení chyb a ověření správné funkčnosti systému. Tato kapitola popisuje metody testování, které byly uplatněny při vývoji systému pro plánování nemocničních služeb. Mezi hlavní používané přístupy v rámci této práce patřily:

- Ruční testování,
- Odchytávání chyb pomocí konstrukce try-catch,
- Využití nástroje console.log().

Ruční testování bylo prováděno průběžně v průběhu celého vývoje aplikace. Testovány byly jednotlivé funkcionality, jako například přihlašování uživatele, správa událostí v kalendáři, změna preferencí a ověřování přístupových oprávnění. Testování probíhalo převážně formou simulace běžného používání aplikace a následným ověřením, zda jednotlivé akce probíhají dle očekávání. Důraz byl kladen na správné načítání a zobrazování dat, možnost jejich úprav a správnou synchronizaci s backendem.

V rámci vývoje byly v klíčových částech aplikace implementovány mechanismy pro zachytávání chyb prostřednictvím konstrukce `‘try-catch‘`. Tento přístup umožňuje zpracovat chyby, které mohou vzniknout při asynchronních operacích, jako je například komunikace se serverem. V případě výskytu chyby je uživateli zobrazena odpovídající chybová zpráva, čímž se předchází neočekávanému selhání aplikace.

Dalším důležitým nástrojem používaným při testování a ladění aplikace bylo využití funkce `‘console.log()‘`. Tento jednoduchý, avšak efektivní nástroj sloužil ke sledování hodnot proměnných, výstupů funkcí nebo odpovědí ze serveru během vývoje.

9 Shrnutí a diskuse výsledků

Cílem bylo vytvořit aplikaci, která umožní efektivní rozvrhování služeb s ohledem na organizační požadavky i individuální preference zaměstnanců. Výsledkem je funkční webová aplikace, která poskytuje uživatelsky přívětivé prostředí a nabízí základní funkcionality pro správu plánování směn.

Výsledky práce odpovídají teoretickým předpokladům a potvrzují, že automatizované plánování směn může významně usnadnit procesy ve zdravotnických zařízeních. Analýza dostupných plánovacích systémů ukázala, že většina stávajících řešení je buď příliš složitá na implementaci, nebo nenabízí dostatečnou flexibilitu pro přizpůsobení konkrétním potřebám nemocnic. Navržený systém tak představuje alternativu, která umožňuje individuální konfiguraci a snadnou rozšiřitelnost.

Použité technologie a metodologie odpovídají současným trendům ve vývoji webových aplikací. Například využití *FullCalendar* pro vizualizaci směn a implementace moderních backendových frameworků umožnily vytvoření systému, který je efektivní a zároveň uživatelsky přívětivý. Výsledky naznačují, že i relativně jednoduché softwarové řešení může přinést výrazné zlepšení oproti manuálnímu plánování nebo zastaralým systémům.

V průběhu práce se objevily některé výzvy, které ovlivnily výslednou podobu aplikace. Jedním z klíčových aspektů bylo zajištění správného fungování plánování při současném dodržení pravidel pro rozvrhování směn. Vývoj byl ovlivněn také nutností optimalizace uživatelského rozhraní, aby bylo co nejintuitivnější pro administrátory i samotné zdravotnické pracovníky.

Dalším důležitým faktorem byla integrace bezpečnostních prvků, zejména v oblasti autentizace a správy uživatelských oprávnění. Ukázalo se, že implementace robustního systému oprávnění je zásadní pro ochranu citlivých dat, což bylo zohledněno v návrhu a implementaci middleware pro autorizaci uživatelů.

Celkově lze říct, že dosažené výsledky potvrzují význam moderních technologií při plánování služeb ve zdravotnictví. Navržený systém splňuje definované cíle a představuje základ pro další rozvoj v oblasti plánování směn.

10 Závěry a doporučení

Tato bakalářská práce se zabývala návrhem a implementací webové aplikace systému pro plánování služeb v nemocnicích. Cílem bylo vytvořit efektivní, uživatelsky přívětivý a adaptivní nástroj, který umožní optimalizovat rozvrhování směn zdravotnického personálu na základě zadaných pravidel a individuálních preferencí.

V teoretické části práce byly analyzovány základní principy informačních systémů, problematika plánování směn a algoritmické přístupy k řešení *Nurse Rostering Problem* (NRP). Byly popsány také existující plánovací softwary a jejich využití v praxi. Na základě této analýzy byly definovány klíčové požadavky na nový systém.

Praktická část se zaměřila na návrh a implementaci webové aplikace pro plánování služeb. Byly popsány použité technologie, včetně backendové a frontendové části systému, a implementace klíčových funkcionalit.

Výsledkem této práce je funkční aplikace, která umožňuje efektivní plánování směn lékařům. Systém nabízí flexibilitu při vytváření rozvrhů, přičemž zohledňuje nejen organizační požadavky, ale i individuální preference uživatelů. Aplikace může sloužit jako základ pro další rozvoj, například rozšíření o pokročilejší algoritmy optimalizace nebo integraci s nemocničními informačními systémy.

Přestože implementovaný systém splňuje stanovené cíle, je zde prostor pro další vylepšení. Možné rozšíření by mohlo zahrnovat pokročilejší metody automatického generování rozvrhů, vylepšenou vizualizaci dat či integraci s existujícími nemocničními systémy. Další vývoj by se mohl zaměřit také na zlepšení uživatelského rozhraní a testování v reálném provozu.

11 Seznam použité literatury

1. **Král, Jaroslav.** *Informační systémy.* Praha : Science, 1998. 80-860-8300-4.

2. **MAYANK, Solanki.** What is Systems Design – Learn System Design. *GeeksforGeeks | a computer science portal for geeks.* [Online] 18. 6 2024. https://www.geeksforgeeks.org/what-is-system-design-learn-system-design/.

3. **Brooks Jr., Frederick.** *Design of Design, The: Essays from a Computer Scientist.* Addison-Wesley Professional, 2010. 0201362988.

4. **GeeksforGeeks.** What is Single Page Application? [Online] 25. 11 2024. https://www.geeksforgeeks.org/what-is-single-page-application/.

5. **Hewlett Packard Enterprise.** Front-End Applications. [Online] 26. 12 2024. https://www.hpe.com/cz/en/what-is/front-end-applications.html.

6. **UX/UI design.** Co je UX/UI design (webů a aplikací)? [Online] 24. 11 2024. https://www.cojeuxui.cz/.

7. **Mozilla Corporation.** HTML: HyperText Markup Language. *MDN Web Docs.* [Online] 26. 12 2024. https://developer.mozilla.org/en-US/docs/Web/HTML.

8. **Mozilla Corporation.** CSS: Cascading Style Sheets. [Online] 26. 12 2024. https://developer.mozilla.org/en-US/docs/Web/CSS.

9. **Mozilla Corporation.** JavaScript. *MDN Web Docs.* [Online] 27. 12 2024. https://developer.mozilla.org/en-US/docs/Web/JavaScript.

10. **Mozilla Corporation.** What is JavaScript? *MDN Web Docs.* [Online] 27. 12 2024. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/What_is_JavaScript.

11. **Vue.js.** Introduction. [Online] 27. 12 2024. https://vuejs.org/guide/introduction.html#what-is-vue.

12. **GeeksforGeeks .** What is Ajax ? [Online] 29. 12 2024. https://www.geeksforgeeks.org/what-is-ajax/.

13. **GeeksforGeeks.** Frontend vs Backend. [Online] 29. 12 2024. https://www.geeksforgeeks.org/frontend-vs-backend/.

14. **Node.js.** About Node.js. [Online] 29. 12 2024. https://nodejs.org/en/about.

15. **GeeksforGeeks.** Express.js Tutorial. [Online] 31. 12 2024. https://www.geeksforgeeks.org/express-js/.

16. **J. Bigelow, Stephen a S. Gillis, Alexander.** RESTful API. *TechTarget.* [Online] 31. 12 2024. https://www.techtarget.com/searchapparchitecture/definition/RESTful-API.

17. **JWT.** Introduction to JSON Web Tokens. [Online] 31. 12 2024. https://jwt.io/introduction.

18. **GeeksforGeeks.** MongoDB – Working and Features. [Online] 8. 1 2025. https://www.geeksforgeeks.org/what-is-mongodb-working-and-features/.

19. **Ó Tuama, Daragh.** Visual Studio Code (VS Code): What It Is and Why Developers Love It. *Code Institute.* [Online] 9. 1 2025. https://codeinstitute.net/global/blog/what-is-vs-code/.

20. **Pinedo, L. Michael.** *Scheduling Theory, Algorithms, and Systems.* Springer Cham, 2022. 978-3-031-05920-9.

21. **AION CS.** Zákony pro lidi. *Zákon č. 262/2006 Sb. zákoník práce.* [Online] 5. 2 2025. https://www.zakonyprolidi.cz/cs/2006-262.

22. **CAMERON, John.** A best practice approach to transitioning from manual to automatic scheduling. *Manufacturing & Logistics IT Magazine.* [Online] 22. 6 2024. https://www.logisticsit.com/articles/2016/07/06/a-best-practice-approach-to-transitioning-from-manual-to-automatic-scheduling/.

23. **Ceschia Sara, Luca Di Gaspero, Mazzaracchio Vincenzo, Policante Giuseppe, Schaerf Andrea.** Solving a real-world nurse rostering problem by Simulated Annealing. *Operations Research for Health Care.* [Online] 24. 6 2024. https://www.sciencedirect.com/science/article/pii/S2211692323000024. 2211-6923.

24. **Scopex, The Complete Enterprise Solution.** What are the requirements of a Hospital management system? [Online] 24. 11 2024. https://scopex.in/what-are-the-requirements-of-a-hospital-management-system/.

25. **Caner Taşkın, Z.** Optimization vs. heuristics: Which is the right approach for your business? *ICRON TECHNOLOGIES.* [Online] 24. 6 2024. https://www.icrontech.com/resources/blogs/optimization-vs-heuristics-which-is-the-right-approach-for-your-business.

26. **Kumar, Atul.** Genetic Algorithms. *GeeksforGeeks | a computer science portal for geeks.* [Online] https://www.geeksforgeeks.org/genetic-algorithms/.

27. **Matrickx.** Why Matrickx. [Online] 19. 6 2024. https://www.matrickx.com/en/why-matrickx.

28. **UKG.** Employee Scheduling software. [Online] 19. 6 2024. https://www.ukg.com/solutions/employee-scheduling-software.

29. **TimeClock Plus, LLC (TCP Software).** Dynamic employee scheduling Humanity. [Online] 21. 6 2024. https://tcpsoftware.com/products/humanity/.
30. **MakeShift.** MakeShift People First Scheduling. [Online] 15. 7 2024. https://www.makeshift.ca/.
31. **QGenda.** QGenda Hospital & Health Systems. [Online] 23. 11 2024. https://www.qgenda.com/hospital-health-systems/.
32. **Experience UX.** What is wireframing? [Online] 6. 2 2025. https://www.experienceux.co.uk/faqs/what-is-wireframing/.
33. **Stevens, Emily.** 7 fundamental user experience (UX) design principles all designers should know. *UX Deisgn Institute.* [Online] 6. 2 2025. https://www.uxdesigninstitute.com/blog/ux-design-principles/.
34. **Webflow.** 9 steps for creating a website visual design in the web design process. [Online] 7. 2 2025. https://webflow.com/blog/the-web-design-process-creating-the-visual-design.
35. **Npmjs.** ESLint. [Online] 24. 2 2025. https://www.npmjs.com/package/eslint.
36. **Npmjs.** Prettier. [Online] 24. 2 2025. https://www.npmjs.com/package/prettier.
37. **Vue.js.** Routing. [Online] 24. 2 2025. https://vuejs.org/guide/scaling-up/routing.
38. **Lotanna, Nwose.** Using event bus in Vue.js to pass data between components. *LogRocket.* [Online] 24. 2 2025. https://blog.logrocket.com/using-event-bus-vue-js-pass-data-between-components/.
39. **Vue.js.** Creating a Vue Application. [Online] 24. 2 2025. https://vuejs.org/guide/essentials/application.html.
40. **GeeksforGeeks.** Middlewres in Next.js. [Online] 24. 2 2025. https://www.geeksforgeeks.org/middlewares-in-next-js/.
41. **FullCalendar.** The Most Popular JavaScript Calendar. [Online] 25. 2 2025. https://fullcalendar.io/.
42. **FullCalendar.** RRule Plugin. [Online] 25. 2 2025. https://fullcalendar.io/docs/rrule-plugin.
43. **Axios.** What is Axios? [Online] 25. 2 2025. https://axios-http.com/docs/intro.
44. **CodeSeven GitHub.** Toastr. [Online] 25. 2 2025. https://codeseven.github.io/toastr/.
45. **Npmjs.** Uuid. [Online] 25. 2 2025. https://www.npmjs.com/package/uuid?activeTab=readme.

46. **Npmjs.** VueDraggable. [Online] 25. 2 2025. https://www.npmjs.com/package/vue-draggable.

47. **Mozilla Corporation.** Cross-Origin Resource Sharing (CORS). *MDN Web Docs.* [Online] 3. 3 2025. https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

48. **Npmjs.** Simple-encryptor. [Online] 3. 3 2025. https://www.npmjs.com/package/simple-encryptor#simple-encryptor.

49. **Mongoose.** Getting Started. [Online] 3. 3 2025. https://mongoosejs.com/docs/index.html.

50. **MongoDB.** Atlas Database. [Online] 4. 3 2025. https://www.mongodb.com/products/platform/atlas-database.

51. **W3Schools.** AJAX Introduction. [Online] 29. 12 2024. https://www.w3schools.com/js/js_ajax_intro.asp.

12 Přílohy

Frontend a backend části aplikace

https://gitfront.io/r/Karambicak/tQHLMZAc1snk/System-planovani-sluzeb-frontend/

https://gitfront.io/r/Karambicak/Mk8TxfEDgr52/System-planovani-sluzeb-backend/

13 Zadání práce z IS (eVŠKP)

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Martin Drábek**
Studium:	I24A0195
Studijní program:	B0613A140033 Aplikovaná informatika
Studijní obor:	Softwarové inženýrství
Název bakalářské práce:	**Systém plánování služeb pro nemocnice**
Název bakalářské práce AJ:	Service scheduling system for hospitals

Cíl, metody, literatura, předpoklady:

Cíl práce:

Cílem práce je návrh a implementace aplikace, která zvládne rozvrhnout služby lékařů podle zadaných parametrů a preferencí tak, aby byla dodržena všechna pravidla. V teoretické části je popsán obecný návrh informačního systému. Celkové definování problematiky plánování služeb, omezení, správa a řešení rozvrhu. Problém rozpisu služeb, algoritmy které jsou používány k jeho řešení a přehled dostupného plánovacího softwaru na trhu. V praktické části práce jsou vymezeny požadavky na navrhovaný nemocniční systém. Výběr technologie a popis využitého softwaru pro tvorbu aplikace, zvolené frameworky a nástroje pro backend, frontend a databázi. Celkové představení a návrh webové aplikace systému. Implementace v rámci jednotlivých částí práce a závěr.

Osnova:

1. Úvod
2. Návrh informačního systému
3. Plánování služeb
4. Požadavky na systém
5. Výběr technologií
6. Návrh webové aplikace systému
7. Implementace webové aplikace systému
8. Shrnutí
9. Závěr

Zadávající pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	Ing. Tomáš Nacházel, Ph.D.
Datum zadání závěrečné práce:	14.10.2024