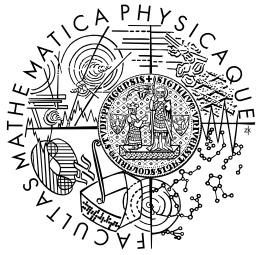


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Marie Korandová

Aplikace pro strážce Národního parku Šumava

Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. Mgr. Martin Nečaský, Ph.D.

Studijní program: Informatika

Praha 2025

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Ráda bych poděkovala svému vedoucímu doc. Mgr. Martinovi Nečaskému, Ph.D., za trpělivost a rady poskytnuté během odborného vedení práce.
Děkuji Jaroslavu Korandovi, Dis., a Ing. Vítu Chladovi za ochotu a poskytnutí námětu na práci.

Název práce: Aplikace pro strážce Národního parku Šumava

Autor: Marie Korandová

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. Mgr. Martin Nečaský, Ph.D., Katedra softwarového inženýrství

Abstrakt: Stráž přírody hraje klíčovou roli pro ochranu národních parků. Strážci pravidelně procházejí určené trasy, kde dohlíží na dodržování pravidel chování návštěvníků a poskytují veřejnosti informační servis. Jejich služba je plánována vedoucími strážních obvodů tak, aby byly trasy kontrolovány v požadované frekvenci, došlo k efektivnímu využití dopravních prostředků a strážci měli pestrou práci, která odpovídá jejich časovým možnostem. Cílem této bakalářské práce bylo navrhnout, implementovat a otestovat aplikaci na plánování služby na základě konzultací s vedoucími strážců Národního parku Šumava. Aplikace umožňuje autorizovaným strážcům a jejich vedoucím zobrazit a vytvořit plán služby. Vedoucí mohou navíc spravovat informace o plánovaných objektech a generovat plán tras, jehož sestavení je navrženo jako řešení problému s omezujícími podmínkami. Výsledkem je single-page webová aplikace využívající knihovnu React, která zobrazuje a modifikuje data načtená z ASP.NET Core backendu.

Klíčová slova: webová aplikace, plánování, národní park

Title: Aplication for rangers of the Šumava National park

Author: Marie Korandová

Department: Department of Software Engineering

Supervisor: doc. Mgr. Martin Nečaský, Ph.D., Department of Software Engineering

Abstract: Rangers play a vital role in protecting national parks. They patrol assigned routes to ensure that visitors follow park rules and to provide guidance to the general public. Their plans are created by the head of the district to ensure adequate individual route coverage, efficient use of all vehicles, and a balanced workload that fits each ranger's availability. The goal of this thesis was to design, implement, and test a planning application based on consultations with supervisors of the Šumava National Park rangers. The application allows authorized rangers and their district heads to view and create service plans. In addition, head of districts can manage data about entities participating in planning and generate a weekly route plan, that is implemented as a constrained satisfaction problem. The result is a single-page web application using the React library, which displays and modifies data retrieved from a backend built on the ASP.NET Core framework.

Keywords: web application, planning, national park

Obsah

Úvod **7**

1 Analýza požadavků **9**
1.1 Uživatelé . . . 9
1.2 Definice . . . 9
1.3 Požadavky . . . 11
1.3.1 Funkční požadavky . . . 11
1.3.2 Kvalitativní požadavky . . . 13
1.4 Use Cases . . . 13

2 Návrh **20**
2.1 Návrh uživatelského rozhraní . . . 20
2.1.1 Přehled stránek . . . 20
2.1.2 Plánování . . . 21
2.1.3 Správa zdrojů . . . 26
2.2 Návrh architektury . . . 28
2.2.1 Frontend . . . 29
2.2.2 Backend . . . 30
2.2.3 Databáze . . . 32
2.3 Spravování dat plánování . . . 32
2.3.1 Rozvrh strážce . . . 32
2.3.2 Provider Dat Plánování . . . 32
2.4 Algoritmus generování plánu tras . . . 33
2.4.1 Contraint Satisfaction Problem . . . 34
2.4.2 Modelování problému . . . 34
2.4.3 Návrh algoritmu . . . 35
2.4.4 Optimalizace . . . 37

3 Implementace **39**
3.1 Frontend . . . 39
3.1.1 Směrování stránek . . . 40
3.1.2 Poskytování kontextu . . . 40
3.1.3 Komunikace s webovým API - Services . . . 44
3.1.4 Stránky . . . 44
3.2 Backend . . . 46
3.2.1 Kontrolery . . . 46
3.2.2 Autentikace a autorizace . . . 50
3.2.3 Hubs . . . 51
3.2.4 Modul generování plánu tras . . . 51
3.2.5 Mapování modelu na databáze . . . 55
3.2.6 Repositories . . . 55
3.3 Spuštění . . . 56

4 Testování **58**
4.1 Frontend 58
4.2 Backend 58
4.3 Uživatelské testování 59
4.3.1 Testování funkcionalit pro strážce 59
4.3.2 Testování funkcionalit pro vedoucího obvodu 61

Závěr **63**

Literatura **64**

Seznam obrázků **65**

Úvod

Posláním Národního parku Šumava [1], stejně jako všech národních parků, je dlouhodobá ochrana biodiverzity a zachování přirozeného ekosystému na chráněném území. Národní parky také umožňují realizaci výzkumných projektů a vzdělávání veřejnosti.[1] K plnění těchto cílů přispívá stráž přírody, která dohlíží na dodržování nastavených pravidel chování návštěvníků národního parku, poskytuje veřejnosti informační servis a monitoruje území spolu se stavem infrastruktury.

Strážci Národního parku Šumava jsou rozděleni mezi dvě pracoviště – Pracoviště západ: Strážní obvod Modrava a pracoviště jih: Strážní obvod Stožec. Obvody řídí vedoucí strážních obvodů, kteří plánují a kontrolují činnosti profesionálních i dobrovolných strážců a starají se o efektivní využití materiálních zdrojů obvodu. Základní náplní práce strážce je nasazení do terénu, tedy průchod nebo kontrola určitých tras. Trasy mají různé priority, které určují minimální frekvenci strážení.

Plánování služby

Aby byla práce strážců efektivní, musí být služba všech strážců s předstihem naplánována podle určitých kritérií. Plánování služby v obvodech probíhá v týdenních intervalech následovně:

- Strážci dají vedoucímu informaci o svých časových možnostech v podobě plánované docházky, případně o preferencích na trasy. Pokud strážce informaci nedá, vyplní jeho docházku vedoucí podle potřeb obvodu.
- Vedoucí strážního obvodu plánovanou docházku zkontroluje a přiřadí dostupným strážcům nějakou množinu tras a dopravních prostředků na každý plánovaný den.
- Po uzavření plánování vedoucím do něj strážce již nesmí sám zasahovat. Všechny další změny vyvolané nečekanou událostí schvaluje a provádí vedoucí strážního obvodu.

Ideálně vytvořený plán služby naplňuje prioritu tras a rozděluje trasy mezi strážce rovnoměrně, aby byla jejich práce pestrá a každý měl možnost projít nejatraktivnější trasy.

Aktuální řešení situace

Aktuálně probíhá plánování služby v obvodech v měsíčních excelových tabulkách na úložišti Google. Tento způsob sice splňuje minimální požadavky, ale není to dlouhodobě udržitelné řešení. Míra autorizace nedostačuje požadavkům – není možné uzavření plánování a zamknutí vůči změnám určitých uživatelů. Práce s daty napříč měsíci či obvody je zdlouhavá, protože nejsou spravována jednotně.

[1] `https://www.npsumava.cz/`

Cíle práce

Cílem této bakalářské práce je navrhnout, implementovat a otestovat aplikaci na plánování služby strážců, což zahrnuje:

C1 Rozhovory se strážci za účelem získání hlubšího pochopení domény a definování požadavků na aplikaci.

C2 Návrh architektury systému, uživatelského rozhraní a struktury databáze.

C3 Implementace jednotlivých funkcionalit a získávání průběžné zpětné vazby od strážců.

C4 Provedení jednotkového, integračního i uživatelského testování aplikace.

1 Analýza požadavků

Nejprve představíme uživatele aplikace, včetně jejich rolí a souvisejících pravomocí. Poté definujeme základní pojmy a popíšeme jejich vztahy v rámci plánování služby. Nakonec na základě rozhovorů se strážci specifikujeme funkční a kvalitativní požadavky a stanovíme use cases (případy užití), které popisují interakce uživatelů se systémem.

1.1 Uživatelé

Aplikace je určena pouze pro interní použití a přístup k ní mají pouze ověření uživatelé s jasně danými rolemi, které odpovídají jejich pracovním pozicím. Podle přidělené role mají uživatelé různé možnosti interakce se systémem.

Strážce je zaměstnanec národního parku, který má za úkol chránit přírodní území národního parku. Řídí se podle plánu služby a může se podílet na jeho vytvoření plánováním docházky a tras.

Vedoucí strážního obvodu (dále jen vedoucí obvodu) je strážce, který vytváří plán služby a spravuje zdroje v obvodu.

Vedoucí oddělení stráže a ochrany přírody je zaměstnanec národního parku, který má přehled o práci strážců ze všech obvodů, má přístup k datům, ale nezasahuje do nich.

1.2 Definice

Struktura domény plánování služby a vztahy mezi níže definovanými entitami jsou reprezentovány v doménovém modelu. (viz Obrázek 1.1)

Obvod je území národního parku spravované vedoucím strážního obvodu. Pracují v něm určití strážci, patří do něj trasy a má k dispozici vlastní dopravní prostředky.

Strážce prochází přidělené trasy a používá dopravní prostředky. Profesionální strážce má přístup do aplikace a aktivně se podílí na plánování služby. Dobrovolný strážce není uživatelem, trasy mu po domluvě přiděluje vedoucí obvodu.

Trasa je část území obvodu, která může být přidělena strážci na daný den. Má danou prioritu, která určuje potřebnou frekvenci projití - denní, týdenní, čtrnáctidenní nebo měsíční.

Kontrolní bod může být přidělen trase. Jedná se o konkrétní místo a čas, na které je strážce, který danou trasu prochází, povinný se dostavit. Kontrolní body jsou vedeny pro zajištění bezpečnosti strážců.

Dopravní prostředek patří do určitého obvodu a je využíván strážci. Mezi typy dopravních prostředků patří například kola, elektrická kola, auta nebo čtyřkolky. Přidělování dopravních prostředků strážcům má na starost výhradně vedoucí obvodů.

Zdroj obvodu může být strážce, trasa nebo dopravní prostředek.

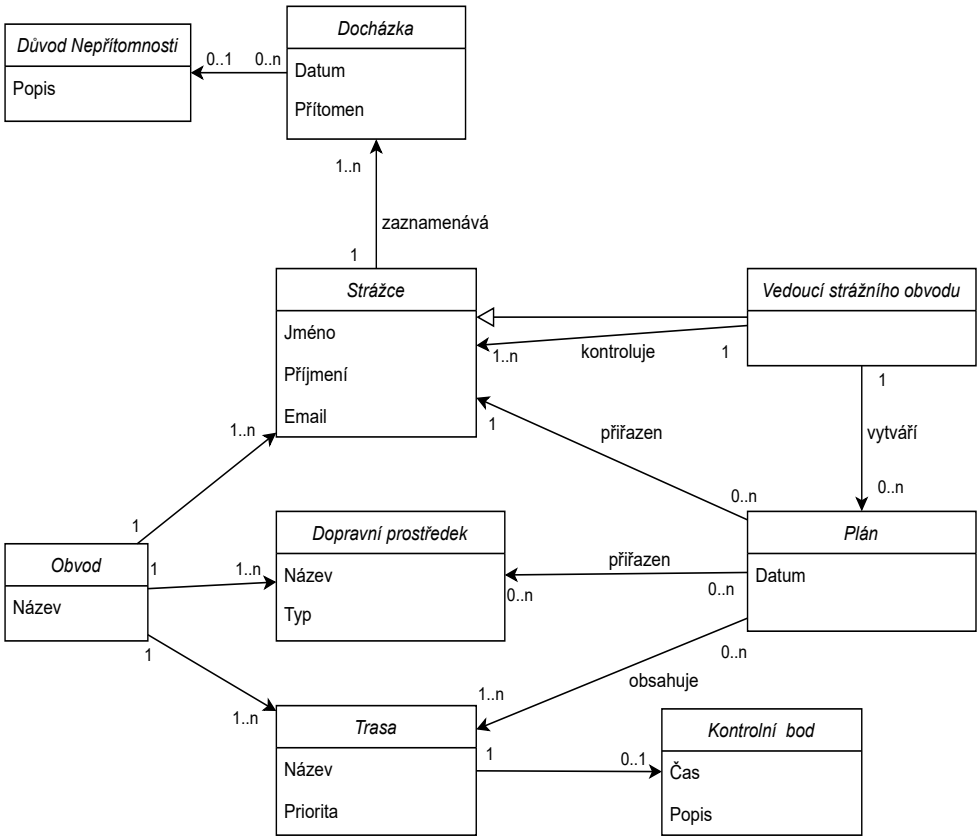
Docházka je informace, zda strážce má v daný den službu a důvod nepřítomnosti, pokud nemá.

Plán strážce obsahuje trasy a dopravní prostředky přidělené konkrétnímu strážci na konkrétní den. Pokud v daný den nemá strážce plánovanou docházku, je plán prázdný.

Rozvrh strážce je spojení plánu a docházky do jedné entity za účelem jednodušší práce s daty.

Plán tras jsou přidělené trasy strážcům na dané dny.

Plán služby je souhrnný název pro docházku, plán tras a přidělené dopravní prostředky na dny nějaké množiny strážců.



Obrázek 1.1 Doménový model

1.3 Požadavky

V této části se zaměříme na specifikaci vlastností a funkcionalit, které by měla aplikace uživatelům poskytovat. Požadavky lze rozdělit na funkční a kvalitativní. Funkční požadavky určují, jaké služby a funkce má systém uživatelům nabízet, a jak by se měl systém chovat v určitých situacích. Naproti tomu kvalitativní (jinak také nefunkční) požadavky nepopisují, co bude systém poskytovat, ale jakým způsobem. Zaměřují se na vlastnosti systému, jako jsou bezpečnost, výkon, použitelnost nebo časová dostupnost.

1.3.1 Funkční požadavky

Funkční požadavky jsou vyjádřeny ve formě *user stories*, které se řídí předlohou *"Jako <uživatel> chci <funkcionalita>, aby/protože <důvody/výhody>"*. Tento přístup umožňuje popsat potřeby jednotlivých uživatelů a poskytuje hlubší pochopení jejich očekávání a motivací. Jsou strukturované do okruhů a každý požadavek má přidělen identifikátor ve formátu *F<číslo požadavku>*. Požadavky s identifikátorem *VF<číslo požadavku>* byly při rozhovorech se strážci označeny za méně důležité.

Správa zdrojů

F1 Jako vedoucí obvodu chci mít možnost přidat, upravit a smazat trasu z kolekce, protože informace tras se mohou měnit a data musí být aktuální pro automatizované generování i pro zobrazování plánu tras.

F2 Jako vedoucí obvodu chci mít možnost určit a změnit, na které trase je kontrolní bod a v jakém čase probíhá kontrola, protože požadavky na kontrolní body se mohou změnit.

F3 Jako vedoucí obvodu chci mít možnost přidat, smazat nebo upravit dopravní prostředek, protože aktuální počet nebo jejich druh se může změnit.

F4 Jako vedoucí obvodu chci mít možnost přidávat, smazat nebo upravit strážce z mého obvodu, aby se v plánu služeb změny promítali.

Zobrazení plánu služby

F5 Jako strážce si chci zobrazit plán služby, abych měl přehled.

F6 Jako strážce chci, aby se na mobilní obrazovce zobrazoval plán služby na jeden den, s možností omezení na zobrazení pouze strážců, kteří jsou ten den v práci, protože je pro mě důležitá přehlednost a čitelnost.

F7 Jako strážce chci, aby se na počítačové obrazovce zobrazoval plán služby na 14 dní, protože to je rozsah plánování do budoucna a na velké obrazovce chci mít širší přehled.

F8 Jako strážce chci, aby i na počítačové obrazovce byla možnost zobrazit si plán služby na jeden vybraný den s možností omezení na zobrazení pouze strážců, kteří jsou v ten den v práci, abych rychle získal přehled o právě pracujících strážcích.

F9 Jako vedoucí oddělení stráže a ochrany přírody chci mít možnost vybrat si obvod a zobrazit jeho informace, abych si mohl zobrazit plán služby daného obvodu a měl všeobecný přehled.

Vytváření plánu služby

F10 Jako strážce si chci naplánovat, které dny budu sloužit, aby mi vedoucí mohl naplánovat trasy a plán odpovídal mým potřebám.

F11 Jako strážce chci přidat důvod nepřítomnosti, pokud v daný den nemám v plánu sloužit, protože potřebuji schválení od vedoucího.

F12 Jako vedoucí obvodu chci změnit naplánovanou docházku strážců, aby docházka odpovídala mým požadavkům a změnám.

F13 Jako strážce chci podat návrh na moje plánované trasy, protože mohu mít preference a chci je předat vedoucímu.

F14 Jako vedoucí obvodu chci plán služby kdykoliv upravit, protože může nastat neočekávaná situace a plán musí být aktuální.

F15 Jako vedoucí obvodu chci mít možnost přiřadit nějakému strážci na den nějaký dopravní prostředek, protože jich je omezené množství a musím je rozdělovat podle potřeby.

F16 Jako vedoucí obvodu chci mít možnost zamknout vytvořený plán služby, aby nemohly nastat změny v plánu bez mého vědomí.

F17 Jako vedoucí obvodu chci generovat týdenní plán tras, který odpovídá docházce strážců, prioritám tras a spravedlivému rozdělení tras mezi strážce v rámci měsíce, protože strážci musejí být s rozdělením tras spokojení a požadavky na četnost průchodů tras v rámci priorit naplněny.

Statistiky tras

VF18 Jako vedoucí obvodu chci zobrazit vizualizaci (graf) - kolikrát byla určitá trasa navštívena jakými strážci v rámci určitého období, protože chci vědět, jestli je trasa plánovaná pro strážce přiměřeně rovnoměrně.

VF19 Jako vedoucí obvodu chci zobrazit vizualizaci (graf) počtu průchodů tras v rámci určitého období, protože chci vědět frekvenci procházení tras.

VF20 Jako vedoucí obvodu chci zobrazit vizualizaci (graf) - kolikrát prošel jaké trasy určitý strážce v rámci určitého období, protože chci vědět, jaké trasy jsou pro určitého strážce plánované nejvíce a nejméně často.

Poznámka ke dni

VF21 Jako strážce chci mít možnost ohodnotit den ve formě poznámky, protože chci poznamenat události, které se staly a předat zajímavé informace vedoucímu obvodu a ostatním strážcům.

VF22 Jako strážce chci vidět denní poznámky ostatních strážců, abych se dozvěděl potřebné informace a mohl na ně reagovat.

1.3.2 Kvalitativní požadavky

Kvalitativní požadavky definují vlastnosti, které by měl systém mít, aby splnil uživatelské očekávání a byl schopný dodat potřebnou funkcionalitu. Každý požadavek má přidělen identifikátor ve formátu *N<číslo požadavku>*.

N1 Mobilní uživatelské rozhraní musí být intuitivní, s dostatečně velkým textem, aby se dalo ovládat s minimálním zaučením i pro netechnicky založené lidi.

N2 Systém musí zobrazovat provedené změny v reálném čase pro všechny aktivní klienty bez nutnosti obnovení stránky.

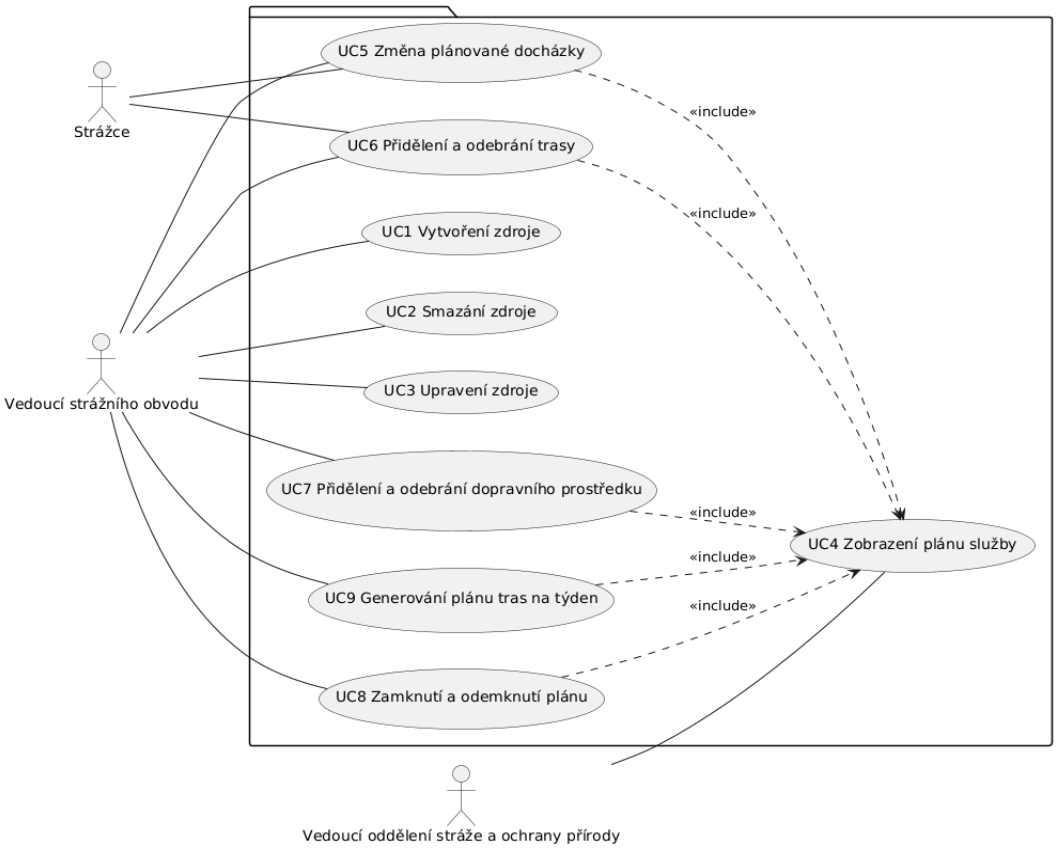
1.4 Use Cases

V této části popíšeme *use cases*, které ilustrují, jak uživatel interaguje se systémem za účelem dosažení potřebného výsledku. Každý se skládá z předpokladů, normálního toku a někdy i alternativního toku. Předpoklady vymezují podmínky pro spuštění dané akce (například role uživatele nebo systémové podmínky). Normální tok po krocích popisuje ideální průběh interakce uživatele a systému a alternativní tok zohledňuje chybové situace nebo krajní případy, a popisuje, jak by měl systém v takových situacích reagovat.

Každý má přidělen identifikátor ve tvaru *UC<číslo use case>* a úzce souvisí s množinou funkčních požadavků (viz Tabulka 1.1). Závislosti mezi *use cases* a jejich vztahy s uživateli jsou znázorněny v *use case diagramu* (viz Obrázek 1.2).

Use Case	Funkční požadavky
UC1 Vytvoření zdroje UC2 Smazání zdroje UC3 Upravení zdroje	F1, F2, F3, F4
UC4 Zobrazení plánu služby	F5, F6, F7, F8, F9
UC5 Změna plánované docházky	F10, F11, F12
UC6 Přidělení a odebrání trasy	F13, F14
UC7 Přidělení a odebrání dopravního prostředku	F15
UC8 Zamknutí a odemknutí plánu	F16
UC9 Generování plánu tras na týden	F17

Tabulka 1.1 Mapování funkčních požadavků na use cases



Obrázek 1.2 Use case diagram

UC1 Vytvoření zdroje

Předpoklady

- Uživatel je přihlášen s rolí vedoucího obvodu.

Normální tok

1. Uživatel se v menu přesměruje na stránku "Správa zdrojů".
2. Uživatel klikne tlačítko "Vytvořit"u požadovaného druhu zdroje.
3. Systém zobrazí formulář na vytvoření nového zdroje.
4. Uživatel vyplní formulář s atributy zdroje a klikne "Uložit".
5. Systém provede validaci zadaných dat, přidá zdroj do seznamu, uloží do databáze a pošle upozornění ostatním klientům o změně v reálném čase.

Alternativní tok

- Validace dat v kroku 5 neproběhne úspěšně, systém oznámí chybovou hlášku a uživatel pokračuje v kroku 4.

UC2 Smazání zdroje

Předpoklady

- Uživatel je přihlášen s rolí vedoucího obvodu.

Normální tok

1. Uživatel se v menu přesměruje na stránku "Správa zdrojů".
2. Uživatel najde v seznamu zdroj, který chce smazat, a klikne u něj na tlačítko "Smazat".
3. Systém zobrazí varovnou hlášku o trvalosti smazání.
4. Uživatel klikne na tlačítko "OK".
5. Systém vymaže zdroj ze seznamu a z databáze, a pošle upozornění ostatním klientům o změně v reálném čase.

UC3 Upravení zdroje

Předpoklady

- Uživatel je přihlášen s rolí vedoucího obvodu.

Normální tok

1. Uživatel se v menu přesměruje na stránku "Správa zdrojů".
2. Uživatel najde v seznamu zdroj, který chce upravit, a klikne u něj na tlačítko "Upravit".
3. Systém zobrazí formulář na upravení zdroje.
4. Uživatel vyplní formulář s atributy zdroje a klikne "Uložit".
5. Systém provede validaci zadaných dat, upraví atributy zdroje v seznamu, uloží změny do databáze a pošle upozornění ostatním klientům o změně v reálném čase.

Alternativní tok

- Validace dat v kroku 5 neproběhne úspěšně, systém oznámí chybovou hlášku a uživatel pokračuje v kroku 4.

UC4 Zobrazení plánu služby

Předpoklady

- Uživatel je přihlášen a má jednu ze tří definovaných rolí.
- Uživatel se nachází na stránce "Plánování".

Normální tok

1. Pokud má uživatel roli vedoucího stráže a ochrany přírody, vybere si v menu obvod z rozbalovacího seznamu.
2. Systém zobrazí plán služby.
    - Pokud je uživatel na počítačové obrazovce, systém zobrazí tabulku s následujícími pravidly:
        - Sloupce představují dny a tabulka je omezena na dva týdny.
        - Sloupce, které odpovídají dnům o víkendu jsou barevně rozlišeny.
        - Řádky představují strážce v obvodu.
        - Rozsah tabulky je možné posunout o týden dopředu a dozadu šipkami.
        - Počáteční rozsah je určen aktuálním datem tak, aby se nacházel v prvním týdnu rozsahu.
        - Buňky v tabulce představují plán daného strážce na daný den.
    - Pokud je uživatel na mobilní obrazovce, systém zobrazí denní plán pro dnešní datum se stejnou charakteristikou jako pod krokem 3.
3. Pokud je uživatel na počítačové obrazovce, může kliknout na tlačítko "Denní plán"a systém zobrazí denní plán strážců, charakterizovaný následovně:
    - Je to seznam plánu strážců na vybraný den.

- Seznam se dá omezit na strážce, který v ten den pracují, stisknutím přepínacího tlačítka.
- Zobrazovaný den se dá změnit vybráním jiného dne ze stejného týdne.
- Vybraný týden lze posunout dopředu a dozadu šipkami.
- V týdnu je zvýrazněn právě vybraný den.
- V týdnu je zvýrazněn dnešní den.
- V týdnu je barevně odlišen víkend.

UC5 Změna plánované docházky

Předpoklady

- Uživatel splňuje jednu z následujících podmínek:
  1. Má roli vedoucího obvodu.
  2. Má roli strážce a docházka, kterou chce změnit, patří jemu, není zamčená vedoucím obvodu a není starší než aktuální datum.
- Uživatel má zobrazený plán služby (UC4).

Normální tok

1. Systém zobrazí každý konkrétní plán strážce tak, že:
   - Obsahuje informaci o docházce strážce daný den (zda má strážce naplánovanou službu).
   - Pokud strážce nemá naplánovanou službu, obsahuje informaci o důvodu nepřítomnosti, která lze změnit výběrem z rozbalovacího seznamu.
   - Pokud strážce má naplánovanou službu, zobrazí se případný list naplánovaných tras a dopravních prostředků.
2. Uživatel v konkrétním plánu klikne na tlačítko reprezentující docházku.
3. Systém změní tlačítko docházky na opačný stav, zobrazí plán tak, aby odpovídal popisu v kroku 1, uloží změnu do databáze a pošle upozornění ostatním klientům o změně v reálném čase.

UC6 Přidělení a odebrání trasy

Předpoklady

- Strážce, kterému se přiděluje nebo odebírá trasa, má v daný den plánovanou službu.
- Uživatel splňuje jednu z následujících podmínek:
  1. Má roli vedoucího obvodu.
  2. Má roli strážce a trasu přiděluje nebo odebírá ve vlastním plánu, který není zamčený vedoucím obvodu a není starší než aktuální datum.
- Uživatel má zobrazený plán služby (UC4).

Normální tok

1. Uživatel v konkrétním plánu klikne na tlačítko s ikonou tužky, což reprezentuje úpravu plánu.
2. Systém zobrazí u každé již naplánované trasy tlačítko k odstranění a rozbalovací seznam všech tras na přidání.
3. Uživatel odebere trasu z plánu.
   (a) Uživatel klikne na tlačítko k odebrání trasy.
   (b) Systém odstraní trasu ze seznamu, uloží změnu do databáze a pošle upozornění ostatním klientům o změně v reálném čase.
4. Uživatel přidá trasu do plánu.
   (a) Uživatel rozbalí seznam všech tras na přidání a jednu vybere kliknutím.
   (b) Systém přidá trasu do seznamu, resetuje rozbalovací seznam, uloží změnu do databáze a pošle upozornění ostatním klientům o změně v reálném čase.
5. Uživatel ukončí editaci plánu stisknutím tlačítka "x".

Alternativní tok

- Pokud se uživatel snaží přidat trasu, která již v daném plánu je, nic se nezmění.

UC7 Přidělení a odebrání dopravního prostředku

Přidělení a odebrání dopravního prostředku má stejný průběh jako přidělení a odebrání trasy, s jediným rozdílem a to, že uživatel musí být přihlášený s rolí vedoucího obvodu.

UC8 Zamknutí a odemknutí plánu

Předpoklady

- Uživatel je přihlášen s rolí vedoucího obvodu.
- Uživatel má zobrazený plán služby (UC4) na počítačovém rozhraní.

Normální tok

1. Uživatel v plánu u konkrétního dne klikne na tlačítko s ikonou zámku.
2. Systém aktualizuje ikonu zámku na opačný stav, uloží změnu do databáze a pošle upozornění ostatním klientům o změně v reálném čase.

UC9 Generování plánu tras na týden

Předpoklady

- Uživatel je přihlášen s rolí vedoucího obvodu.
- Uživatel má zobrazený plán služby (UC4) na počítačovém rozhraní.

Normální tok

1. Uživatel klikne na tlačítko "Generovat".
2. Systém vygeneruje plán tras na druhý týden v zobrazovaném rozsahu a zobrazí ho.
   (a) První se systém pokusí vygenerovat plán tras tak, aby zahrnoval manuálně naplánované trasy.
   (b) Pokud a) není úspěšné, pokusí se vygenerovat plán tras bez zahrnutí manuálně naplánovaných tras.
3. Uživatel potvrdí použití vygenerovaného plánu tlačítkem "Uložit".
4. Systém zobrazí plán služeb, uloží změny do databáze a pošle upozornění ostatním klientům o změně v reálném čase.

Alternativní tok

- V kroku 2, pokud systémem vygenerovaný plán tras nezahrnuje manuálně naplánované trasy, je systémem spolu s plánem zobrazeno i varování.
- V kroku 2, pokud generování není úspěšné, je zobrazena chybová hláška a tlačítko "Zpět", které uživatele vrátí na plán služby.
- V kroku 3 uživatel zruší změny kliknutím tlačítka "Zahodit"a systém vygenerovaný plán neuloží a vrátí uživatele na plán služeb.

2 Návrh

V této kapitole se nejprve zaměříme na návrh uživatelského rozhraní, který vychází z definovaných požadavků a případů užití. Následně navrhneme architekturu systému využitím C4 modelu, včetně použitých technologií.

2.1 Návrh uživatelského rozhraní

Při návrhu uživatelského rozhraní vycházíme z dříve zmíněných *use cases* (1.4) a požadavků na aplikaci. Klíčové principy, kterými se budeme řídit jsou následující:

- **Role-based access control (RBAC)** – dostupnost jednotlivých funkcionalit se liší podle role uživatele
- **Responzivní design** – uživatelské rozhraní se přizpůsobuje velikosti obrazovky. Důraz je kladen na snadno ovladatelné a přehledné mobilní rozhraní (podle požadavku N1).

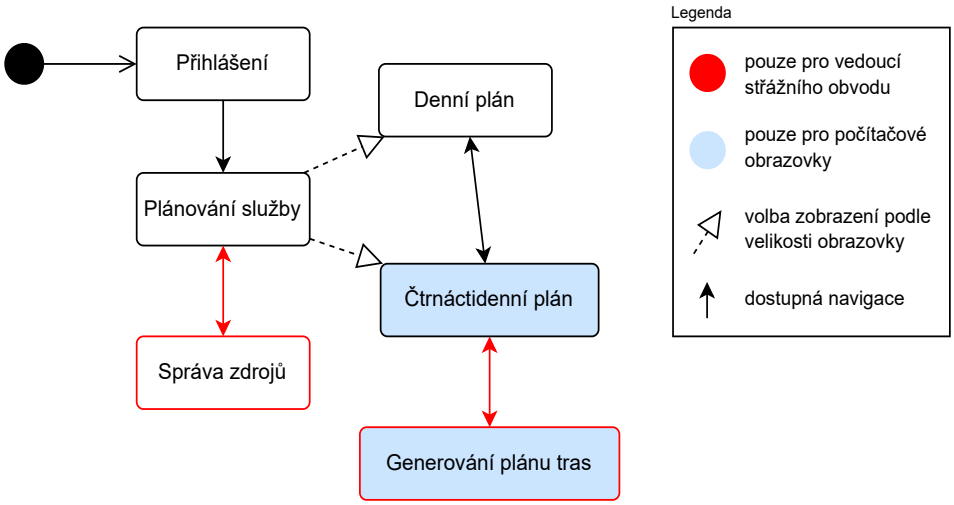
Wireframy uživatelského rozhraní byly navrženy pomocí nástroje Figma[1] a ostatní obrázky v softwaru draw.io[2].

2.1.1 Přehled stránek

Po přihlášení má uživatel v závislosti na své roli přístup ke dvěma hlavním stránkám, které jsou dostupné z menu (viz Obrázek 2.1):

- **Plánování** – zobrazuje plán služby a umožňuje v něm provádět změny. Zobrazení se přizpůsobuje velikosti obrazovky:
  - Na mobilních zařízeních (maximální šířka obrazovky 560 px) je k dispozici pouze denní plán.
  - Na větších obrazovkách lze přepínat mezi denním plánem a dvoutýdenním plánem služby v tabulkovém formátu, vedoucí strážního obvodu může také generovat plán tras a zamykat/odemykat plán.
- **Správa zdrojů** – dostupná pouze pro uživatele s rolí vedoucího strážního obvodu. Umožňuje vytváření, mazání a úpravu zdrojů.

[1] `https://www.figma.com/`
[2] `https://app.diagrams.net/`



Obrázek 2.1 Navigační schéma

2.1.2 Plánování

Stránka Plánování je první stránkou zobrazenou po přihlášení.

Plán strážce

Základní jednotkou zobrazení v denním i dvoutýdenním plánu je komponenta plánu strážce. Tato komponenta slouží k reprezentaci přítomnosti konkrétního strážce v konkrétní den, včetně jeho plánovaných tras, dopravních prostředků, důvodu nepřítomnosti a možnostech úpravy.

Komponenta se zobrazuje rozdílně podle stavu:

- **Strážce nepracuje** - je zobrazena negativní docházka "X"a důvod nepřítomnosti (viz Obrázek 2.2). Pokud je uživatel oprávněný k úpravě docházky, může stisknutím negativní docházky přejít do stavu "Strážce pracuje"nebo změnit důvod nepřítomnosti vybráním z rozbalovacího seznamu.
- **Strážce pracuje** – je zobrazena kladná docházka a seznam plánovaných tras a dopravních prostředků (viz Obrázek 2.3). Trasy jsou barevně rozlišeny podle priority (2.10). Pokud je uživatel oprávněný plán upravovat, může kliknutím na kladnou docházku přejít do stavu "Strážce nepracuje". Oprávněnému uživatelovi se také zobrazí ikona tužky, po jejímž kliknutí přejde plán do editačního režimu.
- **Editační režim** - je zobrazena kladná docházka a seznam plánovaných tras a dopravních prostředků stejně jako ve stavu "Strážce pracuje". Navíc je zobrazen rozbalovací seznam tras pro přidání nové trasy a trasy je možno z plánu smazat stisknutím tlačítka "X". Pro vedoucí strážních obvodů je zobrazena stejná funkcionalita i pro dopravní prostředky (viz Obrázek 2.4).

V denním plánu se komponenty zobrazují v seznamu pod výběrem zobrazovaného dne a součástí komponent jsou i jména strážců. V dvoutýdenním plánu

jsou zobrazeny ve formátu tabulky – každá buňka reprezentuje jednoho strážce a jeden den, a jména strážců jsou vynechána.

Denní plán

Na mobilních zařízeních je uživateli k dispozici pouze denní plán (viz. Obrázek 2.5), který zobrazuje plán služby na jeden konkrétní den. Uživatel může kliknutím vybrat den z týdenního rozsahu. Týdenní rozsah lze posouvat pomocí šipek. Vybraný den je zvýrazněný zakroužkováním, červeným - v případě, že se zároveň jedná o dnešní datum, šedivým - pokud se jedná o víkend, jinak černým.

Pod přehledem dnů se nachází seznam plánů strážců na daný den. Po přepnutí tlačítka v pravém horním rohu na aktivní dojde k filtrování seznamu na pouze plány, které jsou ve stavu "Strážce pracuje".

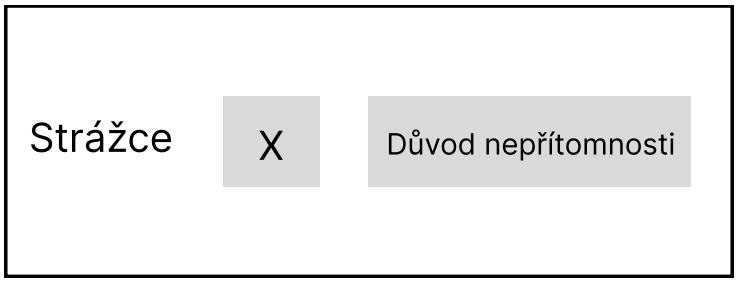
Dvoutýdenní plán

Na obrazovkách širších než 560 px je uživateli zobrazován plán služby ve formátu dvoutýdenního plánu. (viz Obrázek 2.6). Zobrazovaný rozsah lze posunout o týden dopředu nebo dozadu stisknutím šipek. Uživatel má možnost přepnout na denní plán stisknutím tlačítka "Denní plán". Vedoucímu strážního obvodu se zobrazuje také tlačítko "Generovat", kterým může generovat plán tras na druhý týden v zobrazovaném rozsahu.

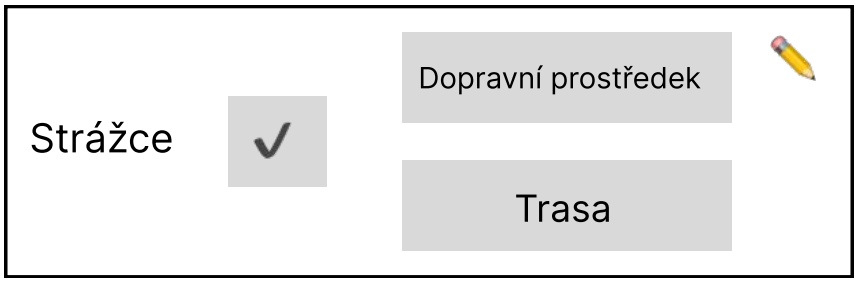
Generování plánu tras

Po stisknutí tlačítka "Generovat"v dvoutýdenním plánu se uživateli zobrazí vygenerovaný plán tras na daný týden (viz Obrázek 2.7). Podobně jako v dvoutýdenním plánu se jedná o tabulku s dny a strážci, buňky ale obsahují pouze trasy (viz Obrázek 2.8), v případě, že danému strážci byla daný den nějaká naplánovaná, nebo "X", pokud nebyla (viz Obrázek 2.9). Uživatel může vygenerovaný plán tras uložit do systému stisknutím tlačítka "Uložit"nebo zahodit pomocí tlačítka "Zahodit", a je přesměrovaný zpět na stránku plánování.

V případě, že generování nebylo úspěšné, je uživateli místo vygenerovaného plánu zobrazena chybová hláška a tlačítko "Zpět".

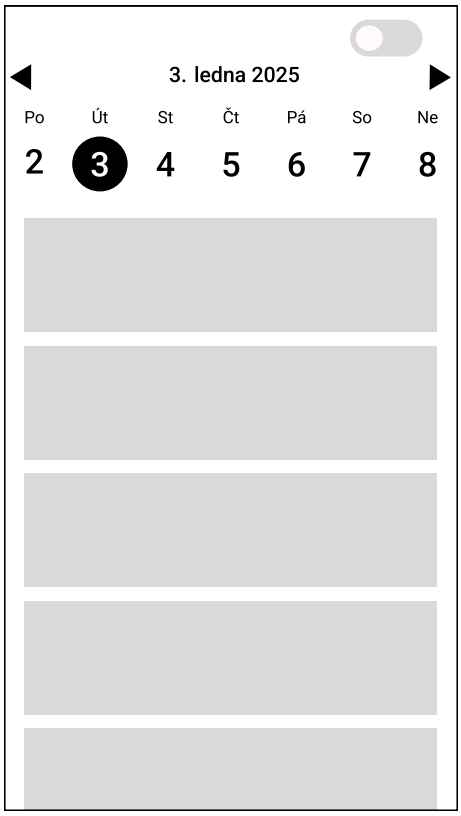


Obrázek 2.2 Wireframe komponenty plánu strážce v případě, že nepracuje

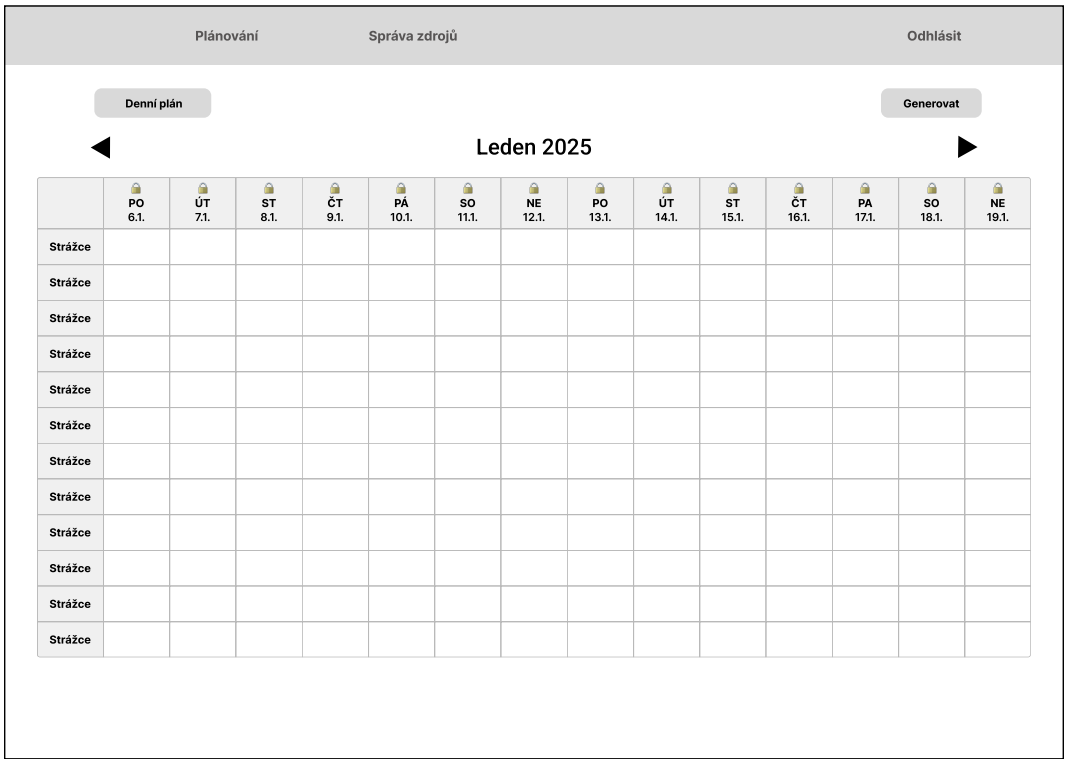


Obrázek 2.3 Wireframe komponenty plánu strážce v případě, že pracuje

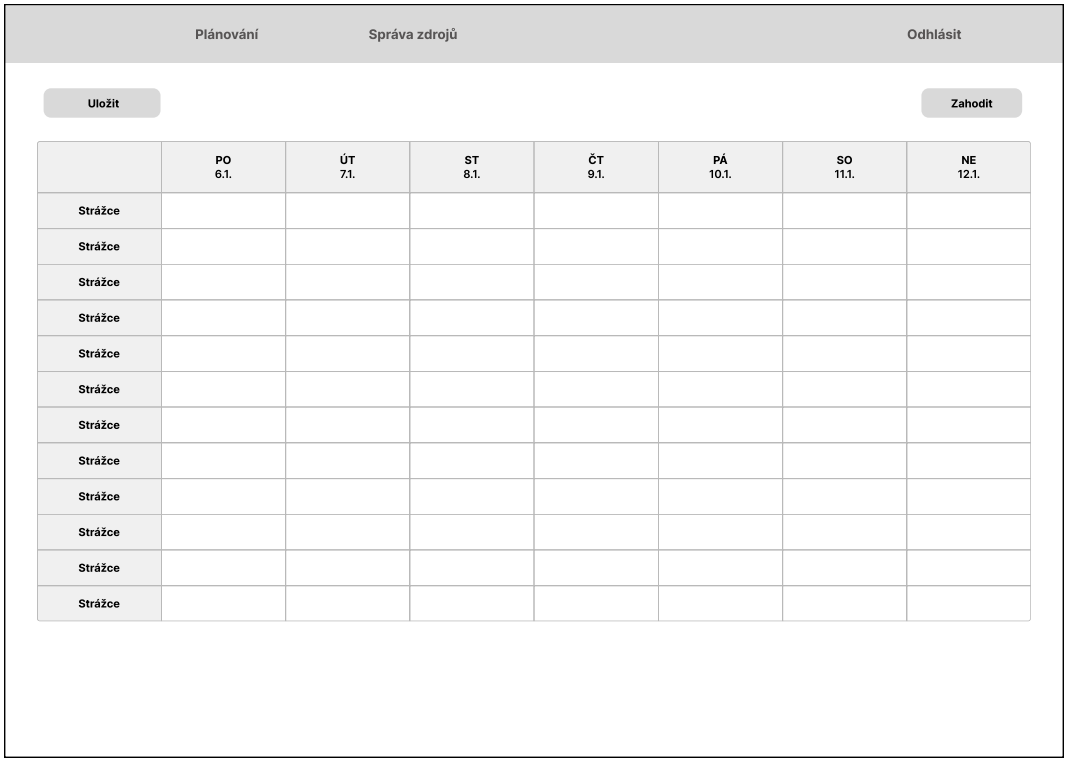
Obrázek 2.4 Wireframe komponenty plánu strážce, který je editován



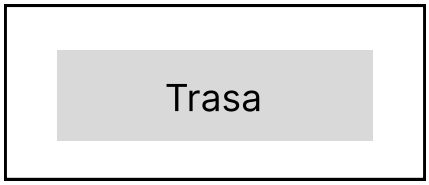
Obrázek 2.5 Wireframe denního plánu pro mobilní obrazovku



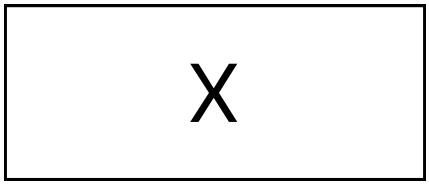
Obrázek 2.6 Wireframe dvoutýdenního plánu



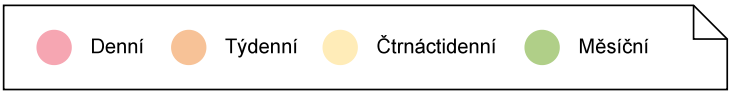
Obrázek 2.7 Wireframe vygenerovaného plánu tras



Obrázek 2.8 Wireframe obsahu plánu tras s vygenerovanou trasou



Obrázek 2.9 Wireframe obsahu plánu tras bez vygenerované trasy



Obrázek 2.10 Barevné rozlišení priorit tras

2.1.3 Správa zdrojů

Vedoucí strážního obvodu mají přes menu přístup ke stránce správy zdrojů (viz Obrázek 2.11). Stránka je rozdělena na tři části - trasy, dopravní prostředky a strážce - a každá část obsahuje list komponent příslušných zdrojů. (viz Obrázek 2.12). Uživatel může zdroj vymazat stisknutím tlačítka "Smazat", nebo přejít k upravování zdroje tlačítkem "Upravit"(viz Obrázek 2.13). Pro vytvoření nového zdroje je nad každou částí k dispozici tlačítko "Vytvořit", kterým se do seznamu přidá nový zdroj v editovaném stavu.

Trasy

Při editaci má komponenta zdroje navíc tlačítko "Přidat kontrolu"nebo "Odebrat kontrolu", kterým může uživatel přidávat a odebírat kontrolní bod trase. Komponenta zdroje u tras obsahuje následující atributy:

- Název - je v komponentě zdroje použit jako `Jméno` při zobrazování. Při editaci musí uživatel zadat neprázdný text.
- Priorita - při editaci vybírá uživatel hodnotu z rozbalovacího seznamu, který obsahuje priority tras - Měsíční, Čtrnáctidenní, Týdenní a Denní.
- Kontrola v čase - je zobrazen jen, když trasa má přidanou kontrolu. Při editaci zadává uživatel novou hodnotu ve formátu času - HH:MM.
- Místo kontroly - je zobrazen jen, když má trasa přidanou kontrolu. Při editaci musí uživatel zadat neprázdný text.

Dopravní prostředky

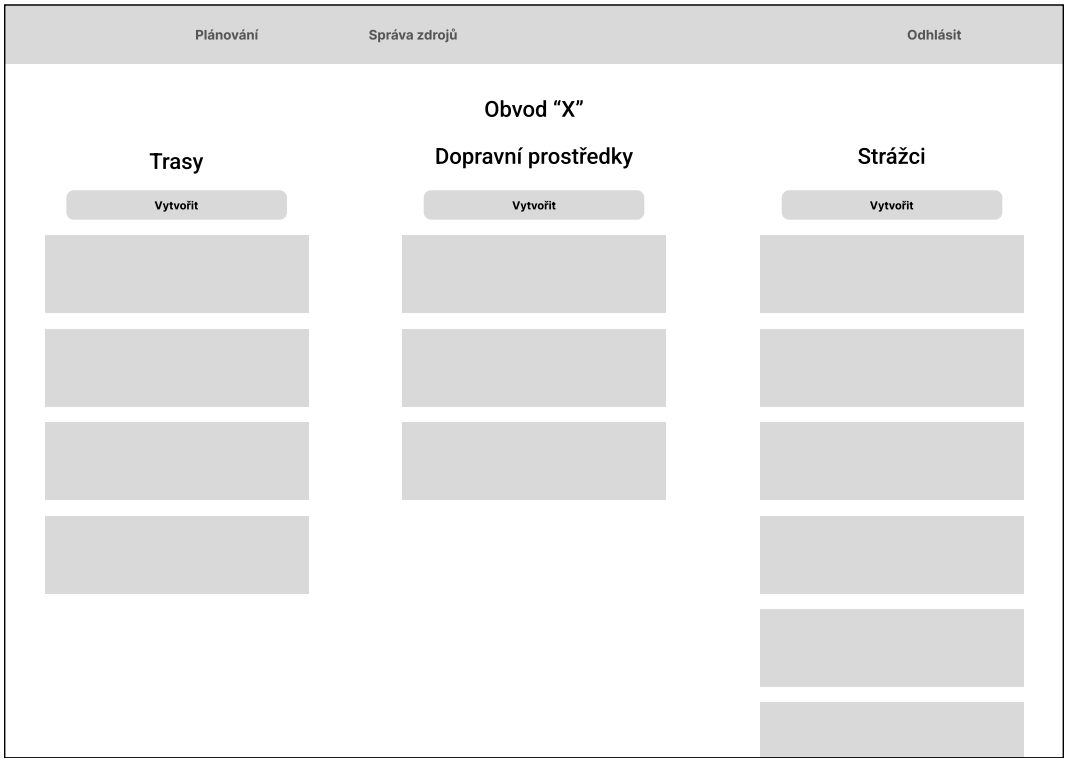
Komponenta zdroje u dopravních prostředků obsahuje následující atributy:

- Název - je v komponentě zdroje použit jako `Jméno` při zobrazování. Při editaci musí uživatel zadat neprázdný text.
- Typ - při editaci musí uživatel zadat neprázdný text.

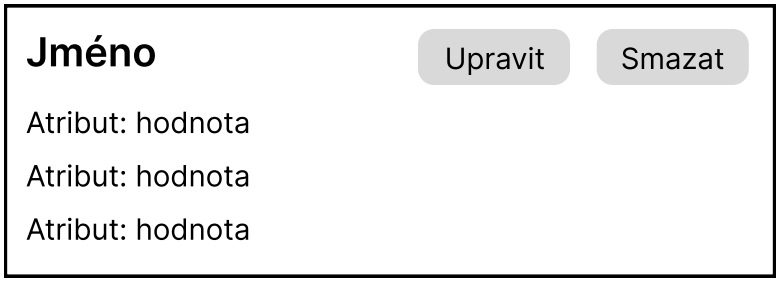
Strážce

Komponenta zdroje u strážců obsahuje následující atributy:

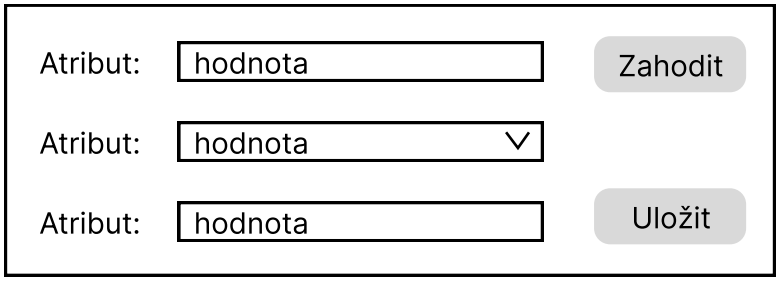
- Jméno - je spolu s atributem Příjmení v komponentě zdroje použit jako `Jméno` při zobrazování. Při editaci musí uživatel zadat neprázdný text.
- Příjmení - je spolu s atributem Jméno v komponentě zdroje použit jako `Jméno` při zobrazování. Při editaci musí uživatel zadat neprázdný text.
- Email - při editaci musí uživatel zadat validní emailovou adresu.



Obrázek 2.11 Wireframe stránky Správa zdrojů



Obrázek 2.12 Wireframe komponenty zdroje



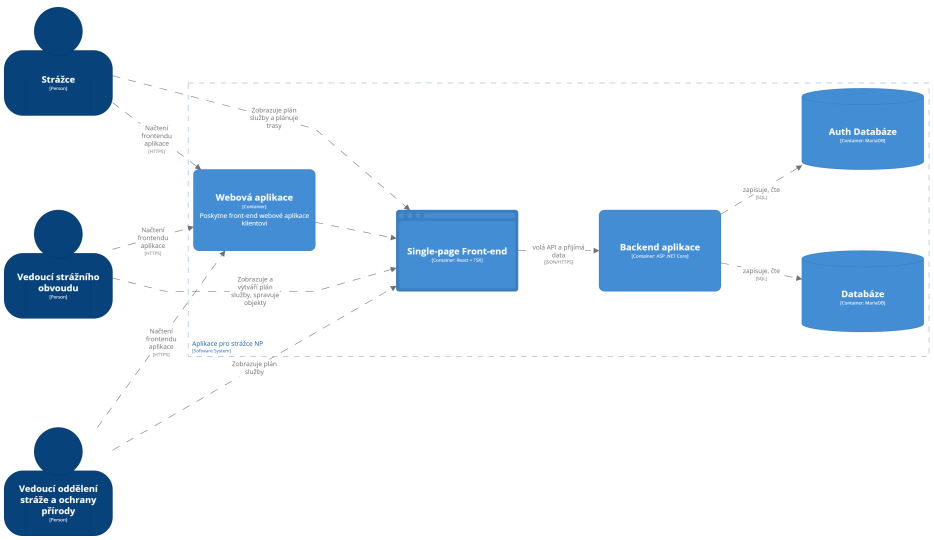
Obrázek 2.13 Wireframe komponenty upravovaného zdroje

2.2 Návrh architektury

Architekturu představíme pomocí **C4 modelu** [2], který definuje sadu hierarchických diagramů[3] popisujících systém na různých úrovních abstrakce:

- Level 1: System **C**ontext Diagram - zobrazuje softwarový systém jako celek a jeho vztahy k uživatelům a externím systémům. Protože žádné takové systémy nevyužíváme a vztahy uživatelů k aplikaci jsme již definovali (1.1), tak tento typ diagramu vynecháme.
- Level 2: **C**ontainer Diagram - rozděluje systém na kontejnery, což jsou samostatně spustitelné nebo nasaditelné jednotky, které vykonávají kód nebo ukládají data (např. aplikace, databáze, servery).
- Level 3: **C**omponent Diagram - je přiblížení kontejneru, který je poskládaný z komponent - skupin souvisejících funkcionalit pod jasně definovaným rozhraním.
- Level 4: **C**ode Diagram - konkrétní třídy, funkce nebo jiné kusy kódu, ze kterých se skládají komponenty. Tak vysoká úroveň detailu je pro naše účely redundantní, a proto tento typ diagramu nevyužijeme.

Navržená architektura systému je znázorněna v *Container diagramu*(viz Obrázek 2.14). Systém se skládá z *single-page* webové aplikace, backendu a dvou databází, detailnímu popisu kontejnerů se budeme věnovat níže. Tento přístup vychází z třívrstvého stylu architektury, kde se frontend stará o prezentační a část business logiky, backend zajišťuje business a perzistentní logiku a data jsou uložena v databázích.



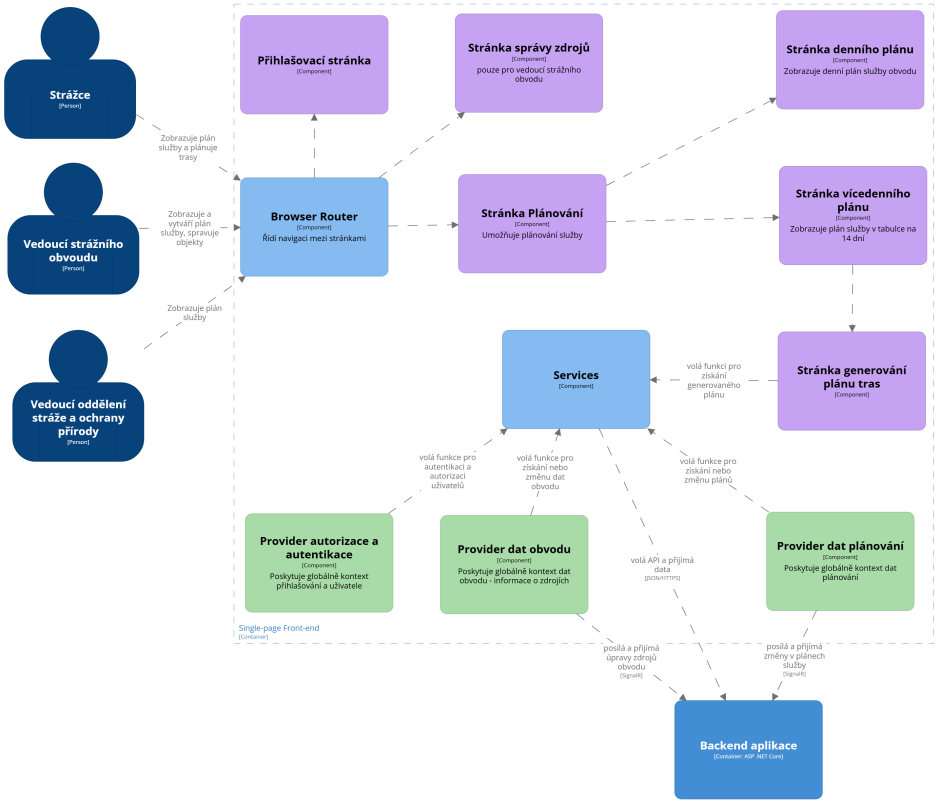
Obrázek 2.14 C4 diagram Level 2: Container diagram

[3]Pro vytvoření diagramů byl použit online nástroj Strukturizr přístupný na `https://structurizr.com/`.

Webová aplikace poskytne klientovi frontend, který využívá knihovnu **React s Typescriptem**. *React*[4] jsme zvolili pro jeho flexibilitu, přehlednou dokumentaci a nižší náročnost na osvojení oproti frameworkům jako *Angular*, což umožňuje rychlejší vývoj. Frontend zobrazuje a pracuje s daty, které jsou načtené z backendu **ASP.NET Core** voláním API kontrolerů přes protokol HTTPS, přičemž data jsou přenášena ve formátu JSON. Aby byla data aktuální pro všechny klienty podle N2, je backend využit také na posílání informací o provedených změnách pomocí *SignalR* knihovny. Backend čte a zapisuje data do dvou databází **MariaDB** pomocí dotazů v jazyce SQL - jedna databáze obsahuje aplikační data (zdroje, docházka, plánování) a druhá je určena pro autentikační data o identitách a rolích uživatelů.

2.2.1 Frontend

Frontend webové aplikace poskytuje autorizovaným uživatelům funkcionality plánování a spravování zdrojů. V *component diagramu* frontendu (viz Obrázek 2.15) jsou zeleně rozlišeni poskytovatelé dat, kteří uchovávají a předávají data všem stránkám, ty jsou zvýrazněny fialově. Tyto vztahy byly z diagramu vynechány za účelem větší přehlednosti.



Obrázek 2.15 C4 diagram Level 3: Component diagram frontendu

[4] `https://react.dev/learn`

Strukturu a odpovědnosti stránek jsme již pokryli v návrhu uživatelského rozhraní (2.1.1), dalšími komponentami jsou:

- **Browser router**[5] (součást knihovny React Router) - je odpovědný za navigaci mezi stránkami.
- **Services** - jsou odpovědné za komunikaci s backendem voláním API.
- **Provider Dat Plánování** - je odpovědný za centrální spravování dat plánování služby (docházka a plány strážců) v rámci vybraného a právě zobrazovaného čtrnáctidenního rozsahu. Tyto plány poskytuje v rámci kontextu stránce plánování. Při změně zobrazovaného rozsahu je odpovědný za získání nových plánů pomocí Services a pokud je provedena v plánech změna, notifikuje o tom pomocí Hub Plánů ostatní klienty.
- **Provider Dat Obvodu** - je odpovědný za centrální spravování zdrojů obvodu (tras, dopravních prostředků, strážců). Tyto data poskytuje v rámci kontextu stránce plánování a stránce správy zdrojů. Pokud je provedena v zdrojích změna, notifikuje o změně ostatní klienty použitím Hub Obvodu na backendu.
- **Provider Autentizace a autorizace** - spravuje stav přihlášení a ověřuje roli uživatele, tento stav v rámci kontextu poskytuje všem stránkám. Volá funkce z Services spojené s přihlašováním a odhlašováním.

2.2.2 Backend

Backend zpracovává požadavky frontendu pomocí kontrolerů[6]. V *component diagramu* backendu je za účelem přehlednosti většina kontrolerů zastoupena komponentou *placeholder* (viz Obrázek 2.16). Kontrolery:

- Kontrolery tras, strážců a dopravních prostředků - obsahují primárně pouze HTTP metody pro základní CRUD operace (vytváření, čtení, úpravy a mazání). Výjimkou je kontroler strážců, který také obsahuje metodu na získání strážce, který je přiřazený k aktuálně přihlášenému uživateli.
- Kontroler Plánů - umožňuje generovat plán tras, spravovat přiřazení tras a dopravních prostředků a získávat plánovací data v zadaném časovém rozmezí.
- Kontroler Docházky - slouží k updatování stavu docházky konkrétního strážce na konkrétní den.
- Kontroler Zámků - zamknutí a odemknutí dne v určitém obvodu a získání všech zamknutých dní.
- Kontroler Obvodu - poskytuje metody na získání informací o obvodech.
- Kontroler Uživatelů - přihlašování a odhlašování uživatelů, registraci nových uživatelů a správu rolí.

[5] `https://reactrouter.com/6.30.0/router-components/browser-router`
[6] `https://learn.microsoft.com/cs-cz/aspnet/core/web-api/`

Pro oddělení business logiky od přístupu k datům je použit *Repository Pattern.*[3, kapitola 21]

Repositories zajišťují přístup k databázi pomocí *Entity Framework Core*[7], který mapuje databázové tabulky na objekty aplikace. *General Repository* je obecná šablona, která implementuje rozhraní pro získávání, ukládání, mazání a aktualizaci objektů v databázi. Je využit pro správu všech objektů kromě plánů a docházky, jejich *repositories* implementují stejné rozhraní, ale mají specifické metody pro získávání dat.

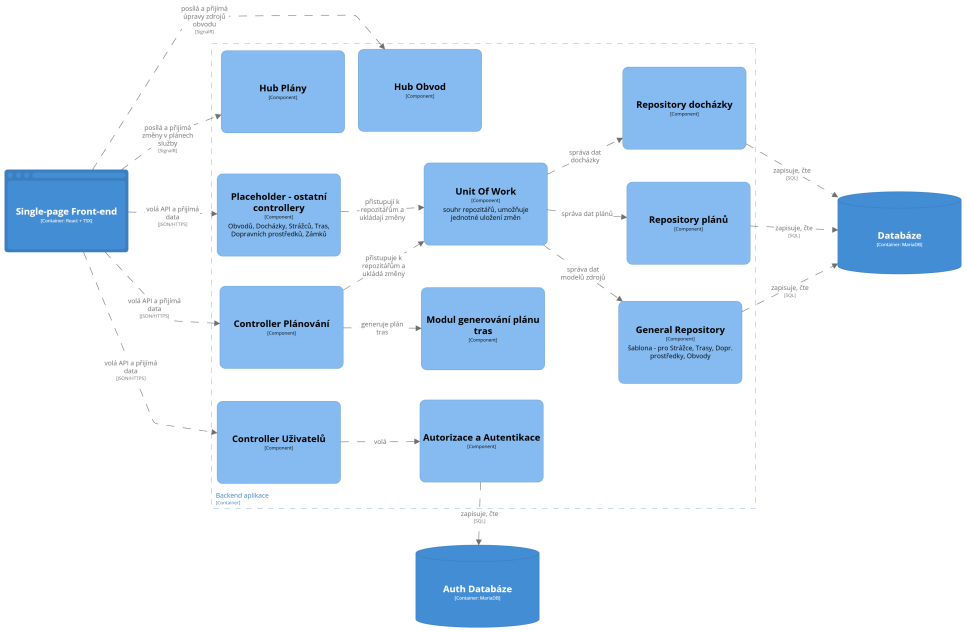
Unit of work je třída, která kontrolerům poskytuje přístup k jednotlivým *repositories* a zajišťuje, že všechny pracují se stejnou reprezentací databáze, čímž udržuje konzistenci dat. Umožňuje uložit všechny změny provedené v reprezentaci databáze do reálné databáze v jednom atomickém kroku.

Huby jsou implementované pomocí knihovny *SignalR* a slouží k posílání informací o provedených změnách a tím udržují plán služby a informace o objektech aktuální napříč všemi klienty. Umožňují klientům přidat se do skupiny podle obvodu. Díky skupinám jsou notifikace o změnách posílány pouze klientům, kterých se změny týkají.

- Hub Plány - slouží k posílání notifikací o změnách v plánu služby.
- Hub Obvod - slouží k posílání notifikací o změnách ve zdrojích skupině klientů. Notifikační metody jsou čtyři - pro posílání změn o trasách, dopravních prostředcích, strážcích a zámcích.

Autentikace a autorizace je řešena pomocí *ASP.NET Core Identity*. Autentikační služba zajišťuje přihlašování, odhlašování, registraci a získání informací o uživateli. Autorizační služba pak pracuje s přiřazováním a ověřováním rolí.

Modul generování plánu tras je odpovědný za sestavení nového plánu tras podle požadavku F17. Upřesnění problému a detailní návrh algoritmu je představen v samostatné sekci - 2.4.



Obrázek 2.16 C4 diagram Level 2: Component diagram backendu

[7] `https://learn.microsoft.com/cs-cz/ef/core/`

2.2.3 Databáze

Pro ukládání dat systém využívá dvě samostatné relační databáze **MariaDB**, což je systém postaven na *MySQL* a využívá jazyk SQL *(Structured Query Language)* pro manipulaci s daty.

Jedna databáze slouží pro uložení doménových dat aplikace (například plánů, docházky, zdrojů), zatímco druhá uchovává údaje o uživatelských identitách a rolích. Toto oddělení vychází z principu *separation of concerns*, kdy je každý subsystém zodpovědný za jasně vymezenou oblast. Data týkající se autentizace a autorizace mají odlišný charakter než aplikační data, a jejich oddělením zvyšujeme modularitu systému. Zároveň tento přístup výrazně usnadňuje případnou budoucí výměnu mechanismu autentizace a autorizace – například za externí poskytovatele, aniž by bylo nutné zasahovat do struktury hlavní aplikační databáze.

Struktura aplikační databáze odpovídá navrženému doménovému modelu (viz Obrázek 1.1) s několika drobnými odchylkami. Například pro implementaci zamykání plánů byla přidána pomocná tabulka pro ukládání zámků, kde každý záznam představuje konkrétní zamknutý den a obvod.

Entity mají přiděleny unikátní identifikátor, pro většinu se jedná o generované číslo. Výjimkou jsou entity plánů a docházky, které jsou určovány kombinací data a identifikátoru obvodu.

2.3 Spravování dat plánování

Tato sekce má za úkol navrhnout způsob spravování dat plánování. Nejprve popíšeme rozdíl spravování dat mezi backendem a frontendem, a představíme novou pomocnou entitu *Rozvrh strážce*. Poté navrhneme, jak uchovává a načítá plánovací data komponenta Provider Dat Plánování.

2.3.1 Rozvrh strážce

Data plánů a data docházky (1.2) jsou z doménového pohledu a jejich budoucího použití oddělené entity. Například z dat plánů by bylo možné vytvářet statistiky pro průchody tras a z docházky získat počet zbývající dovolené. Na frontendu naší aplikace ale zobrazujeme docházku a plány v přesně stejných situacích a jejich oddělení přináší zbytečnou složitost.

Na backendu a v databázi budeme s daty pracovat odděleně, zabráníme tak vzniku závislosti na způsobu, jak data používá frontend. Poskytneme ale HTTP metodu v kontroleru plánů, která bude vracet spojené data plánů a docházky pro nějaký časový úsek. Tuto novou entitu nazveme *Rozvrh strážce*.

Na frontendu budeme přijímat plánovací data jako *Rozvrhy strážců* a stejným způsobem je také poskytujeme stránkám prostřednictvím globálního kontextu. Pro updatování dat do backendu budeme rozvrhy dělit zpět na plány a docházku.

2.3.2 Provider Dat Plánování

Pro Provider dat plánování musíme navrhnout, kdy budeme rozvrhy načítat a na jaký časový úsek. Plán služby je zobrazován na čtrnáct dní nebo na den v týdenním rozsahu, posouvání rozsahu je v obou případech o týden.

Při návrhu jsme zvážili tři možnosti spravování rozvrhů:

- Spravování **dvoutýdenního rozsahu** - při inicializaci jsou načteny dva týdny z backendu, poskytovány jsou jako jedna proměnná, ale interně se týdny spravují zvlášť. Při posunu rozsahu uživatelem o týden vpřed nebo dozadu budeme načítat pouze jeden nový týden, druhý získáme z interního uložení.
- Spravování **čtyřtýdenního rozsahu** - při inicializace jsou načteny čtyři týdny, které se také spravují interně zvlášť. Při změně rozsahu není třeba čekat na nově načítaný týden, protože aktuálně zobrazované týdny jsou již součástí interních dat.
- Složitější způsob s využitím **cache** - týdny jsou načítány dynamicky podle potřeby a při překročení kapacity cache jsou nejdéle nepoužívané rozvrhy automaticky odstraněny.

Rozhodli jsme se pro spravování dvoutýdenního rozsahu kvůli jednoduchosti přístupu. Frekvence posouvání zobrazovaného rozsahu ze strany uživatele je nízká, a tedy nepředstavuje významný problém.

2.4 Algoritmus generování plánu tras

Podle požadavku F17 musí naše aplikace být schopná sestavit nový plán tras na vybraný týden podle následujících kritérii:

A) **Docházka** - každému strážci, který má na určitý den zaznamenanou kladnou docházku, musí být naplánována nějaká trasa. Zároveň žádná trasa není naplánována strážci, který daný den nepracuje. Toto kritérium je **povinné** a bez jeho naplnění není řešení validní.

B) **Priority tras** - vytvořený plán tras naplňuje priority tras.

   - Denní priorita - trasa je naplánována každý den v týdnu.
   - Týdenní priorita - trasa je naplánována alespoň jednou v týdnu.
   - Dvoutýdenní priorita - pokud nebyla trasa naplánována minulý týden, musí být v tomto týdnu naplánována přesně jednou. Jinak maximálně jednou.
   - Měsíční priorita - pokud nebyla trasa naplánována v minulých třech týdnech, musí být přesně jednou naplánována v tomto týdnu. Jinak maximálně jednou.

   Toto kritérium je také **povinné**.

C) **Preference na trasy** - strážci mají možnost si manuálně plánovat trasy na určité dny, které by měli být součástí vytvořeného plánu tras. Toto kritérium je **preferované**, ale pokud neexistuje řešení, jsou manuálně naplánované trasy opomenuty.

D) **Rovnoměrné rozložení tras** - trasy by měli být mezi strážce rovnoměrně rozděleny. Toto kritérium je **optimalizační**, tedy neovlivňuje validitu řešení.

Upřesnění

Pro zjednodušení problému, zavedeme dodatečná upřesňující pravidla:

- Každému strážci může být při generování plánu přiřazena maximálně jedna trasa.
- Pokud má podle C) strážce manuálně naplánované více než jednu trasu, je toto naplánování považováno za fixní. Pokud vygenerování validního plánu závisí na zahození této instance, pak generování nebude úspěšné.
- Dvou různým strážcům nemůže být při generování naplánována stejná trasa na jeden den.
- Pokud mají dva různí strážci manuálně naplánovanou stejnou trasu na jeden den, je toto naplánování považováno za fixní.
- Rovnoměrné rozložení tras bere ohled na minulé plány v rozsahu jednoho měsíce.

2.4.1 Contraint Satisfaction Problem

Constraint Satisfation Problem (CSP) se skládá z:

- konečné množiny **proměnných** (*variables*),
- jejich **domén** (*domains*), což je konečná množina hodnot, které mohou proměnné nabývat,
- a konečné množiny **podmínek** (*constraints*), které omezují hodnoty, které mohou proměnné současně nabývat. [4]

Řešení CSP je přiřazení hodnot všem proměnným tak, aby byly všechny podmínky splněny. Na náš problém generování plánu tras se můžeme dívat jako na CSP, kde hledáme přiřazení tras strážcům na dny, které splňuje naše kritéria.

2.4.2 Modelování problému

Problém modelujeme definováním proměnných, jejich domén a podmínek. Můžeme modelovat dvěma způsoby:

1. **Přiřazování tras** - vychází z přirozeného chování domény, kde strážcům přiřazujeme na určitý den trasy.

   Proměnné jsou dvojice [den, strážce].
   Domény jsou trasy.

   Použitím pouze proměnných, kdy strážci v daný den pracují, jednoduše splníme kritérium A). Kritérium priorit tras B) se řeší složitěji. Pro každou trasu, kterou je nutné naplánovat (alespoň jednou, maximálně jednou nebo přesně jednou), musí být přidána podmínka, která omezuje všechny proměnné. Například pro všechny proměnné $v_1, v_2, \ldots, v_n$ a trasu t, která musí být naplánována alespoň jednou, by byla přidána tato podmínka: $\bigvee_{i=1}^{n}(v_i = t)$. Výjimku tvoří trasy s denní prioritou, u kterých se podmínka vztahuje pouze na proměnné daného dne.

   Kritérium C) se snadno naplní omezením domény dané proměnné pouze na manuálně naplánovanou trasu. Poslední kritérium D) můžeme řešit seřazením

domény jednotlivých proměnných podle míry vhodnosti přiřazení dané trasy danému strážci na daný den, například na základě toho, jak často již danou trasu měli naplánovanou v porovnání s ostatními.

Tento přístup přesně vystihuje náš problém, ale kvůli velkým doménám a vysokému počtu podmínek, z nichž většina musí být definována globálně, vede k vysoké složitosti řešení.

2. **Přiřazování strážců** - vychází z alternativního pohledu na problém, kdy si představíme, že každá trasa má pro každý den v týdnu potenciál být naplánovaná nějakému strážci.

   Proměnné jsou dvojice [den, trasa].
   Domény jsou strážci spolu s prázdnou hodnotou.

   Přiřazením prázdné hodnoty reprezentujeme, že trasa nebyla v den naplánována. Kritérium docházky A) se snadno naplní omezením domén proměnných pouze na strážce, kteří v daný den pracují. Pro splnění kritéria priorit tras B) vytvoříme sadu omezujících podmínek. Pro každou trasu, kterou je nutné naplánovat (alespoň jednou, maximálně jednou nebo přesně jednou), přidáme podmínku, která omezuje pouze proměnné s danou trasou. Například pro proměnné $v_1, v_2, \ldots, v_n$ se společnou trasou t, která musí být naplánována alespoň jednou, by byla přidána následující podmínka: $\bigvee_{i=1}^{n}(v_i \neq \emptyset)$. Každá proměnná v_1, která se vztahuje k trase s denní prioritou, by měla unární podmínku[8] $v_1 \neq \emptyset$. Kritéria preferencí na trasy C) a rovnoměrného rozdělení tras D) můžeme řešit stejně jako v prvním způsobu modelování, pouze s převrácením strážců a tras.

   Tento přístup sice definuje více proměnných, ze kterých se většině přiřadí prázdná hodnota, ale proměnné mají menší domény a omezující podmínky jsou více lokalizované.

Z uvedených dvou způsobů volíme model **přiřazování strážců**, který sice není tak intuitivní jako model přiřazování tras, ale snižuje složitost řešení.

2.4.3 Návrh algoritmu

Pro nalezení řešení CSP existuje primitivní algoritmus *generate and test*, který prohledá všechny možné kombinace přiřazení hodnot proměnným a testuje je, dokud nenajde platné řešení. V našem případě použijeme *backtracking* algoritmus, který postupně rozšiřuje částečná řešení výběrem hodnot pro proměnné a vrací se zpět, pokud dojde k porušení podmínek (viz Algoritmus 1). [4]

Zpracování dat

Před spuštěním samotného algoritmu musíme nejdříve zpracovat data:

- Rozdělit trasy do typových skupin podle jejich požadovaného počtu naplánování v plánovaném týdnu - závisí na prioritách a minulých plánech podle B) (jednou, alespoň jednou, maximálně jednou).
- Seřadit pro každou trasu strážce podle míry vhodnosti - podle procentuálního zastoupení dané trasy v minulých plánech strážce.

[8] Unární podmínka omezuje pouze jednu proměnnou.

Sestavení CSP vlastností

Poté můžeme připravit proměnné, domény a podmínky našeho CSP:

- **Proměnné** - pouze dvojice [den,trasa], kde trasa v daný den již není manuálně naplánována. Jsou seřazeny podle třech kritérii - podle dne, v rámci dne podle typové skupiny a naposled náhodně.
- **Domény** - do domény proměnné [den, trasa] patří pouze strážci, kteří pracují a nemají daný den nic manuálně naplánováno. Tyto hodnoty jsou pak seřazeny podle míry vhodnosti.
- **Podmínky** - jsou sestavené jak pro množiny proměnných se stejnou trasou, pro naplnění typových skupin, tak pro i pro proměnné se stejným dnem, aby nebyl přiřazen jeden strážce dvakrát v rámci jednoho dne.

Při generování plánu tras se pokusíme najít řešení se zahrnutím manuálně naplánovaných tras, když takové řešení nebude existovat, zopakujeme proces přípravy dat a CSP vlastností bez nich (s výjimkou fixních plánů definovaných v doplňujících pravidlech) a znovu spustíme *backtracking* algoritmus.

Algoritmus 1 *Backtracking* algoritmus, řeší CSP pro proměnné V, jejich domény D a podmínky C. Vrací první nalezené řešení nebo *null*.

```
1: function BackTrack(solution)
2:     if |V| = |solution| then
3:         return solution                                  ▷ našli jsme řešení
4:     end if
5:     variable ← min{v ∈ V | v ∉ solution}      ▷ první proměnná bez hodnoty
6:     for domain in D[variable] do
7:         solution[variable] ← domain
8:         if ∀c ∈ C[variable], solution splňuje c then     ▷ kontrola podmínek
9:             result ← BackTrack(solution)
10:            if result ≠ null then
11:                return result
12:            end if
13:        end if
14:        solution ← solution \ {variable}                 ▷ vrácení zpět
15:    end for
16:    return null                                          ▷ řešení neexistuje
17: end function
```

Pozorování

Vzhledem k tomu, že se strážci seřadí podle vhodnosti pouze jednou před spuštěním algoritmu, dochází pro trasy k opakovanému vybírání stejných strážců. Například pokud je vyhodnoceno, že strážce S je nejvhodnější pro trasu T, která musí být naplánována každý den, pak se mu trasa T přiřadí pokaždé, když pracuje. To jde silně proti kritériu rozložení tras D) a je nutná optimalizace algoritmu.

2.4.4 Optimalizace

Představíme si dvě hlavní optimalizace, po jejichž zavedení došlo k napravení hlavního problému a dosažení lepšího rozdělení tras mezi strážce. Viz Algoritmus 2 pro pseudokód výsledného *backtracking* algoritmu po přidání optimalizací.

Algoritmus 2 *Backtracking* algoritmus po zavedení optimalizací.

```
1: function BackTrack(solution)
2:     if |V| = |solution| then
3:         return solution                                  ▷ našli jsme řešení
4:     end if
5:     variable ← ChooseVariable(solution)                  ▷ výběr proměnné
6:     for domain in GetValidDomain(variable) do            ▷ výběr hodnoty
7:         solution[variable] ← domain
8:         if ∀c ∈ C[variable], solution splňuje c then
9:             UpdateDomains(variable, domain) ▷ aktualizace hodnot domén
10:            result ← BackTrack(solution)
11:            if result ≠ null then
12:                return result
13:            end if
14:            RevertDomains(variable, domain)   ▷ obnova původního stavu
15:        end if
16:        solution ← solution \ variable                   ▷ vrácení zpět
17:    end for
18:    return null                                          ▷ řešení neexistuje
19: end function
```

Dynamické domény

V prvním návrhu jsme pro každou proměnnou určili fixní doménu před spuštěním algoritmu. Jako optimalizaci zavedeme funkci `GetValidDomain`, která nám vrátí novou hodnotu z domény pro proměnnou. Abychom viděli změny, musíme domény všech proměnných průběžně aktualizovat v závislosti na přiřazovaných hodnotách. Tento postup je známý jako *pruning* a je součástí zajišťování lokální konzistence podmínek. [5]

Nechť je proměnné v přiřazena hodnota d, pak ze všech domén proměnných, které s v sdílí den, d odstraníme, protože podle podmínek nesmí být strážce přiřazen dvakrát v jednom dni. U všech domén proměnných, které s v sdílí trasu, přesuneme d na konec domény, aby se strážce stal nevhodným pro opakované přiřazení. Pokud však přiřazení, které způsobilo změnu domén neproběhne úspěšně a dojde k vracení, je nutné vrátit domény do původního stavu. Tyto funkcionality jsou v pseudokódu reprezentovány funkcemi `UpdateDomains` a `RevertDomains`. Můžeme si všimnout, že pokaždé když domény updatujeme, děláme to pro stejné skupiny proměnných - se společnou trasou nebo dnem. Spravování jednotlivých domén pro všechny proměnné je zbytečné a neefektivní. Místo toho budeme spravovat domény pro dny a pro trasy zvlášť, a voláním `GetValidDomain` dojde k jejich spojení a výpočtu nejvhodnější hodnoty.

Dynamické řazení proměnných

Také řazení proměnných bylo v prvním návrhu jasně dané před spuštěním algoritmu a v průběhu se již neměnilo. Teď když máme zavedené dynamické domény, můžeme na jejich změny reagovat i detailnějším řazením proměnných a výběrem následující. Tato funkcionalita je v pseudokódu představena jako `ChooseVariable`.

3 Implementace

Z návrhu architektury je již jasné, že se systém skládá z dvou oddělených částí: frontendu (single-page webové aplikace v Reactu) a backendu (ASP.NET Core). Pro jejich vývoj jsme zvolili Visual Studio, které poskytuje šablonu *React and ASP.NET Core* pro Typescript, což přesně vyhovuje našemu návrhu. V této kapitole popíšeme strukturu a implementaci našeho řešení, spolu s vysvětlením implementačních detailů složitějších komponent pro frontend i backend. Nakonec poskytneme návod, jak aplikaci sestavit a spustit.

3.1 Frontend

Zdrojové kódy *React* frontendu se nachází ve složce `./app.client`. Pro lepší přehled je přiložena vizualizace její zjednodušené struktury (viz 1).

React aplikace jsou postavené z komponent[1], což jsou části uživatelského rozhraní s vlastní logikou a vzhledem, které do sebe můžeme vnořovat. Komponenty mohou využívat speciální funkce nazývané *hooks*, které poskytují přístup ke globálnímu kontextu, spravování stavu nebo jinou funcionalitu. React poskytuje sadu předdefinovaných - například `useState`, `useContext` nebo `useEffect`.

Program 1 Zjednodušená struktura složek frontendu

```
app.client
 \---src
     +---Components
     |   +---Menu
     |   +---Planner
     |   |   +---DailyPlanner
     |   |   +---GeneratedPreview
     |   |   +---PlanRecord
     |   |   +---PlansForDay
     |   |   +---PlanTable
     |   \---Sources
     |       +---Rangers
     |       +---Routes
     |       +---Vehicles
     +---Hooks
     +---Pages
     +---Providers
     |   +---Authentication
     |   \---DataProviders
     +---Services
     |---Util
     \---App.tsx
```

[1]Pozor, nejedná se o stejné komponenty jako v C4 modelu. (2.2)

3.1.1 Směrování stránek

Pro směrování mezi stránkami používáme `BrowserRouter`, který je součástí knihovny React Router. Uvnitř `BrowserRouter` jsou definované cesty na stránky, obalené *providery* dat pro spravování globálních kontextů (viz Program 2). Cesty na stránky jsou čtyři: `/prihlasit`, `/planovani`, `/sprava` a `/generovani/:date`. Domovská stránka není v aktuálním stavu nutná, takže je cesta `"/"` využita na obalení stránek komponentou `Menu`.

Všechny stránky kromě přihlašovací vyžadují autentizaci uživatele, a proto jsou navíc obaleny v komponentě `AuthRoute`, která podle kontextu autorizace a autentizace renderuje své *child* komponenty nebo naviguje na přihlašovací stránku. `AuthRoute` také umožňuje specifikovat list rolí, pro které jsou stránky přístupné, a autentizovaný uživatel bez povolené role je přesměrován na plánovací stránku.

Program 2 Směrování mezi stránkami v App.tsx. (Dvě tečky pouze znázorňují zkrácení názvů.)

```
<DistrictDataProvider> <AuthProvider> <SchedulesProvider>
<Routes>
    <Route path="/prihlasit" element={<Login />}/>
    <Route element={<AuthRoute />}>
        <Route path="/" element={<Menu />}>
            <Route path="/planovani" element={<Planner />} />
            <Route element={<AuthRoute roles={["A..", "H.."]} />}>
                <Route path="/sprava" element={<S.. />} />
                <Route path="/generovani/:date" element={<G.. />}/>
            </Route>
        </Route>
    </Route>
</Routes>
</SchedulesProvider> </AuthProvider> </DistrictDataProvider>
```

3.1.2 Poskytování kontextu

Pro udržování globálního stavu aplikace a jeho zpřístupnění komponentám napříč stromovou strukturou využíváme funkcionalitu Reactu zvanou *Context*[2]. Poskytování globálně dostupných dat se provádí ve třech krocích (viz Program 3):

1. Definice typu a vytvoření kontextu - nejprve definujeme typ dat, která bude kontext poskytovat (`TContextType`), a poté vytvoříme samotný kontext pomocí funkce `createContext()`.

2. Využití kontextu - komponenty volají `useContext(TContext)` pro získání přístupu k datům. V našem případě jsme definovali pomocné *hooky* s jasnějšími jmény pro snadné používání (viz Program 4).

3. Poskytnutí kontextu - vytvoření `TProvider`, který data spravuje a obaluje stromovou strukturu využitím `TContext.Provider`, čímž data poskytuje svým *child* komponentám.

[2] `https://react.dev/learn/passing-data-deeply-with-context`

Program 3 Obecná struktura poskytovaní kontextu

```
export const TContext
    = createContext<TContextType>({} as TContextType);

export const TProvider
    = ({ children }: { children: ReactNode }): JSX.Element => {
    .
    .
    return (
        <TContext.Provider value={value}>
            {children}
        </TContext.Provider>
    );
};
```

Program 4 Pomocný *hook* na využití kontextu

```
export default function useT() {
    return useContext(TContext);
}
```

Provider Dat Obvodu

Provider dat obvodu (`DistrictDataProvider`) poskytuje komponentám kontext typu `DistrictContextType` (viz Program 5). Data obvodu jsou populována voláním funkce `assignDistrict`, která také inicializuje připojení k SignalR Hub `/districtHub` využitím `HubConnection` pro přijímání notifikací o změnách dat od ostatních klientů. Po odhlášení jsou data obvodu zahozena voláním funkce `clearDistrict`. Součástí kontextu jsou také funkce na správu zdrojů (přidání, odebrání a aktualizace), a zamykání a odemykání dnů. Tyto funkce volají fetch funkce definované v `Services` pro propagaci změn na backend a využívají definovanou `HubConnection` pro posílání upozornění o provedených změnách.

Pro použití kontextu obvodu volají komponenty definovaný *hook* `useDistrict`.

Provider Autentizace a Autorizace

Provider autentizace a autorizace (`AuthProvider`) poskytuje komponentám kontext typu `AuthContextType` (viz Program 6). Poskytovaný kontext obsahuje informace o přihlášeném uživateli a jednoduché funkce pro přihlášení, odhlášení a ověření role uživatele. Při prvním načtení se `AuthProvider` pokusí získat přihlášeného uživatele voláním fetch funkce `GetCurrentUser` ze složky `Services`. Pokud získá přihlášeného uživatele, který patří do obvodu, zavolá funkci `assignDistrict` použitím *hooku* `useDistrict`. Takto zajistíme, že uživatel zůstane přihlášený při obnovení stránky. Funkci `assignDistrict` volá i při úspěšném přihlášení. Při odhlašování je volána funkce `clearDistrict`.

Pro použití kontextu autentizace a autorizace je definován *hook* `useAuth`.

Program 5 Typ poskytovaného kontextu obvodu. Kód byl mírně komprimován sloučením funkcí stejného typu.

```
interface DistrictContextType {
    // data obvodu
    district: District|undefined,  routes: Route[],
    vehicles: Vehicle[],  rangers: Ranger[],
    locks: Locked[],

    // zmena obvodu
    assignDistrict: (districtId: number) => void,
    clearDistrict: () => void,

    // sprava zdroju
    addRoute, removeRoute, changeRoute: (route: Route) => void,
    addVehicle, removeVehicle, change...: (vehicle: Vehicle) => void,
    addRanger, removeRanger, changeRanger: (ranger: Ranger) => void,

    // zamykani planu
    addLock, removeLock: (date: string) => void,

    loading: boolean,
    error: Error| null,
}
```

Provider Dat Plánování

Provider dat plánování (`ScheduleDataProvider`) poskytuje komponentám kontext typu `ScheduleContextType` (viz Program 7) a jeho návrhem jsme se zabývali v podkapitole o spravování dat plánů (2.3). Rozvrhy strážců se načítají nebo čistí vždy při změně přiřazeného obvodu, který je získán pomocí *hooku* `useDistrict`. Při změně obvodu se také inicializuje připojení k SignalR Hubu `/rangerScheduleHub` pomocí `HubConnection`, přes které jsou přijímány notifikace o změnách v plánech a docházce. Speciální notifikace `Reload` vyžaduje znovunačtení celých rozvrhů z backendu. Kontext dále obsahuje funkce pro správu zobrazovaného rozsahu - `resetSchedules`, `weekForward`, `weekBack` a `changeDate`. Funkce `triggerReload` je volána po uložení vygenerovaného týdenního plánu tras a posílá všem ostatním klientům notifikaci `Reload`. Součástí kontextu jsou také funkce pro změnu docházky, přidání a odebrání tras a dopravních prostředků, které jsou využívány komponenty `PlanRecord`. Tyto funkce volají fetch funkce definované v `Services` pro propagaci změn na backend, využívají definovanou `HubConnection` pro posílání upozornění o provedených změnách a aktualizují lokální stav. `ScheduleDataProvider` spravuje tři pole s rozvrhy strážců:

- jedno pole zahrnující rozvrhy pro celé dvoutýdenní období (poskytované kontextem),
- a dvě další pole – pro první a druhý týden zvlášť – určené pro interní správu.

Všechna pole musí zůstat aktuální, sloučené pole je používáno komponentami pro zobrazování a na konzistenci interních se spoléhá logika pro posouvání rozsahu zobrazovaného plánu. Všechny změny první promítneme do hromadného pole, které následně znovu rozdělíme podle týdnů a uložíme jako novou interní strukturu.

Pro použití kontextu obvodu volají komponenty definovaný *hook* `useSchedule`.

Program 6 Typ poskytovaného kontextu autentizace a autorizace

```
interface AuthContextType {
    user: User | undefined,
    loading: boolean,
    error: Error | null,
    signin: (email: string, password: string) => void,
    signout: () => void,
    hasRole: (role: string) => boolean
 }
```

Program 7 Typ poskytovaného kontextu plánování. Kód byl mírně komprimován sloučením funkcí stejného typu a zkrácením názvů.

```
interface ScheduleContextType {
    // sprava rozvrhu strazcu a zmena rozsahu
    schedules: RangerSchedule[],
    dateRange: { start: Date, end: Date },
    resetSchedules: () => void,
    weekForward: () => void,
    weekBack: () => void,
    changeDate: (date: Date) => void,
    triggerReload: ()=> void,

    // zmena dochazky
    updateWorking:
          (date: string, r: Ranger, w: boolean) => void,
    updateReasonOfAbsence:
          (date: string, r: Ranger, r: ReasonOfAbsence) => void,

    //plánovani dopravnich prostredku
    addPlannedVehicle, removePlannedVehicle:
          (date: string, rId: number, vehicleId: number) => void,

    // planovani tras
    addPlannedRoute, removePlannedRoute:
          (date: string, rId: number, routeId: number) => void,

    loading: boolean,
    error: Error|null,
}
```

3.1.3 Komunikace s webovým API - Services

Ve složce `.\app.client\Services` jsou funkce pro komunikaci s API, rozdělené do složek podle odpovídajících kontrolerů. Při neúspěšné odpovědi vyhazují chybu se zprávou převzatou z odpovědi backendu.

3.1.4 Stránky

Navazujeme na návrh uživatelského rozhraní (2.1) a máme tři komponenty představující stěžejní stránky - `Login`, `Planner` a `SourceManagement`.

Jednoduchá komponenta `Login` zobrazuje formulář pro přihlášení a volá funkci `signIn` získanou použitím *hooku* `useAuth`.

Plánování

Základní stavební jednotkou stránky plánování je komponenta `PlanRecord`, která odpovídá navržené jednotce *Plán strážce* uživatelského rozhraní (2.1.2). Jako parametry přijímá konkrétní rozvrh strážce, informaci zda zobrazovat jména strážců a povolovat přechod do režimu editace (viz Program 8). Využívá všechny námi definované kontexty:

1. `DistrictContext` na získání informací o zobrazovaných trasách a dopravních prostředcích,
2. `AuthContext` pro umožnění spravování přidaných dopravních prostředků pouze vedoucímu obvodu využitím funkce `hasRole`,
3. a `ScheduleContext`. pomocí kterého volá funkce na změnu docházky a plánování dopravních prostředků i tras.

Program 8 Předpis komponenty `PlanRecord`. Názvy typů byli zkráceny, RS představuje typ `RangerSchedule` a b je `boolean`.

```
const PlanRecord:
    React.FC<{ schedule: RS, includeRangerName: b, isEditable: b }>
= ({ schedule, includeRangerName, isEditable }) : JSX.Element
```

Pomocná komponenta, která zobrazuje list komponent `PlanRecord` na konkrétní den, se nazývá `PlansForDay`. Jako parametry přijímá den a informaci, zda filtrovat rozvrhy pouze na strážce, kteří pracují (viz Program 9). Využívá kontext hook `useDistrict` pro získání `schedules`, které filtruje podle vybraného dne, případně i podle pracujících. Pokud se mají zobrazit i nepracující, poskytne prázdné rozvrhy pro strážce, kteří žádný nemají. Určuje také, které `PlanRecord` komponenty mají povolen režim editace, podle role uživatele, zámků a vlastnictví.

Program 9 Předpis komponenty `PlansForDay`.

```
const PlansForDay:
    React.FC<{ date: Date, onlyWorking: boolean}>
    = ({ date, onlyWorking }) : JSX.Element
```

Zobrazení denního plánu (2.1.2) zprostředkovává komponenta `DailyPlanner`, která udržuje a zobrazuje stav vybraného dne, umožňuje zapínat a vypínat filtrování na pracující strážce a tyto parametry posílá komponentě `PlansForDay`. Zobrazuje šipky pro posouvání týdenního rozsahu a při jejich aktivaci volá funkce `weekForward` nebo `weekBack` získaných voláním *hooku* `useSchedule`.

Zobrazení dvoutýdenního plánu (2.1.2) zprostředkovává `PlanTable` komponenta, která vytváří tabulku buněk `PlanRecord` komponent. Využívá všechny námi definované kontexty:

1. `AuthContext` pro zobrazování zámků pouze vedoucím obvodů, a také pro určování, které `PlanRecord` komponenty mají povolen režim editace,

2. `DistrictContext` pro získání strážců, zámků a funkcí spojených se zámky.

3. a `ScheduleContext` pro získání `schedules`, které pro úplnost doplňuje prázdnými a předává jednotlivým `PlanRecord` komponentám. Tlačítka šipek volají funkce pro posouvánní rozsahu. Zobrazované nadpisy sloupců - datumy - se aktualizují podle měnícího se rozsahu `dateRange`.

Hlavní komponenta stránky plánování - `Planner`, má za úkol podle šířky obrazovky a stavu tlačítka na přepínání zobrazit `DailyPlanner` nebo `PlanTable`. Pokud je obrazovka mobilní, což zjistí pomocí *hooku* `useMediaQuery`, zobrazuje pouze komponentu `DailyPlanner`. Když je obrazovka větší, poskytuje tlačítko na přepínání mezi `DailyPlanner` a `PlanTable`. Při zobrazení dvoutýdenního plánu je k dispozici pro vedoucí obvodu také tlačítko na generování plánů, které pomocí funkce `navigate` z *hooku* `useNavigate` směruje na adresu `/generovani/:date` zobrazující komponentu `GeneratedPreview`. Druhá část cesty `:date` je datum druhého pondělí v rozsahu ve tvaru YYYY-MM-DD.

Komponenta `GeneratedPreview` slouží k zobrazení náhledu vygenerovaného plánu tras pro aktuální obvod a datum, které je získáno z URL použitím *hooku* `useParams`. Při inicializaci se zobrazí zpráva o načítání, zatímco volá funkci `fetchGeneratedRoutePlan` ze složky `Services`. Úspěšně získaný plán tras je zobrazen v tabulce a při jeho případném uložení jsou volány funkce `UpdatePlans` a `Reload` skrz `ScheduleContext`. Při neúspěchu je zobrazena chybová zpráva a navigační tlačítko zpět na `/planovani`.

Správa Zdrojů

Hlavní komponenta stránky správy zdrojů - `SourceManagement` - se dělí na tři *child* komponenty - `RoutesManger`, `VehicleManager` a `RangerManager`. Každý *TManager* využívá *hook* `useDistrict` pro získání zdrojů *T* a zobrazuje list komponent *TItem*, spolu s tlačítkem pro vytvoření nového zdroje. *TItem* odpovídá navržené komponentě zdroje uživatelského rozhraní (2.1.3), při přechodu do editačního režimu zobrazí komponentu *TForm*, což je jednoduchý formulář s konkrétními atributy zdroje. Komponenta *TForm* je využita i z *TManager* pro vytvoření nového zdroje.

3.2 Backend

Zdrojové kódy *ASP.NET Core* backendu se nachází ve složce `.\App.Server`. Pro lepší přehled je přiložena vizualizace její zjednodušené struktury (viz 10).

Program 10 Zjednodušená struktura složek backendu

```
App.Server
 +---Controllers
 +---CSP
 +---DTOs
 +---Models
 |   +---AppData
 |   \---Identity
 +---Repositories
 |   \---Interfaces
 +---Services
 |   +---Authentication
 |   +---Authorization
 |   \---Hubs
 +---Util
 \---Program.cs
```

Vstupním bodem aplikace je soubor `Program.cs`, kde probíhá inicializace pomocí *ASP.NET Core* třídy `WebApplicationBuilder`. Jsou zde nakonfigurované a do *dependency injection* kontejneru přidané služby, aby mohly být k dispozici napříč aplikací[3]. Mezi takové služby patří `UnitOfWork`, `RoutePlanGenerator`, interní služby pro autentizace a autorizaci a `DbContexty` pro mapování databází.

Při inicializaci je také ověřeno, že jsou v databázi definované uživatelské role, a že je přítomen uživatel s administrátorskou rolí. Nakonec aplikace zpřístupní *endpointy* kontrolerů a hubů voláním funkcí `MapControllers` a `MapHub`.

3.2.1 Kontrolery

Kontrolery zpracovávají HTTP požadavky frontendu. Každý kontroler má atribut `[ApiController]` a dědí od třídy `ControllerBase`[4]. Atribut `[Route]` definuje cestu, na které jsou HTTP metody kontroleru přístupné. Většina kontrolerů mají *dependency-injected* službu `IUnitOfWork` pro přístup k jednotlivým *repositories*. Výjimkou je `UserController`, který využívá služby spojené s autorizací a autentikací.

Atribut `[Authorize]` omezuje přístup jen pro uživatele určitých rolí. Pokud je atribut použit bez uvedení rolí, je přístup povolen všem přihlášeným uživatelům. Metody vždy vrací objekt typu `ActionResult`, který specifikuje stavový kód a případnou návratovou entitu. Obecná struktura je představena v Programu 11.

Data Transfer Objekty a Validace

Data Transfer Object (DTO) je datová struktura sloužící k přenosu informací mezi systémy. DTO používáme na posílání dat mezi frontendem a backendem, pro

[3] `https://learn.microsoft.com/en-us/aspnet/core/fundamentals`
[4] `https://learn.microsoft.com/cs-cz/aspnet/core/web-api/`

definování jasného rozhraní komunikace a zabránění odhalení interních datových struktur frontendu. Jsou umístěné ve složce `.\App.Server\DTOs`. Využitím atributů z balíčku *Data Annotations* provádíme jejich základní validaci, jako označení vlastností za povinné a omezení rozsahu hodnot nebo formátu (viz Program 12). Protože jsou naše kontrolery označeny atributem `[ApiController]`, provádí se validace automaticky.

Program 11 Zobecněný příklad struktury tříd kontrolerů a jejich metod

```
[ApiController]
[Route("api/[controller]")]
public class TController(IUnitOfWork unitOfWork) : ControllerBase
{
    private readonly IUnitOfWork _unitOfWork = unitOfWork;

    [Authorize(Roles = "Admin,HeadOfDistrict")]
    [HttpPost("create")]
    public async Task<ActionResult<TDto>> Create(TDto tDto)
    {...}
}
```

Program 12 Ukázka části *Data Tranfer Objectu* pro entitu strážce

```
public class RangerDto
{   .......
    [Required]
    public string LastName { get; set; } = string.Empty;

    [Required]
    [EmailAddress]
    public string Email { get; set; } = string.Empty;

    [Required]
    [Range(0, int.MaxValue)]
    public int DistrictId { get; set; }
}
```

Kontroler Docházky

Kontroler Docházky (`AttendenceController`) poskytuje HTTP *endpoint* pro aktualizaci docházky.

`PUT api/Attendence/update`
: Aktualizuje nebo vytvoří novou docházku, pokud její záznam neexistuje. Jako parametr přijímá v těle požadavku `AttendenceDto`.

Kontroler Obvodu

Kontroler Obvodu (`DistrictController`) poskytuje HTTP *endpointy* pro získání obvodů s prefixem `api/District`.

`GET /get-all`
: Vrací všechny obvody jako list `DistrictDto`.

`GET /{districtId}`
: Vrací obvod s identifikačním číslem `districtId` jako `DistrictDto`.

Kontroler Zámků

Kontroler Zámků (`LockController`) poskytuje následující HTTP *endpointy* s prefixem `api/Lock`:

`POST /lock/{districtId}/{date}`
: Uzamkne den `date` v obvodu s identifikačním číslem `districtId`.

`DELETE /unlock/{districtId}/{date}`
: Odemkne den `date` v obvodu s identifikačním číslem `districtId`.

`GET /locks/{districtId}`
: Vrací list zámků jako `LockDto` v obvodu s identifikačním číslem `districtId`.

Kontroler Plánů

Kontroler Plánů (`PlanController`) poskytuje HTTP *endpointy* s prefixem `api/Plan`:

`PUT /add-route/{date}/{rangerId}?routeId={routeId}`
: Přidá strážci s identifikačním číslem`rangerId` do plánu na den `date` trasu s identifikačním číslem`routeId`. Pokud plán neexistuje, vytvoří nový.

`PUT /remove-route/{date}/{rangerId}?routeId={routeId}`
: Odebere strážci s identifikačním číslem`rangerId` z plánu na den `date` trasu s identifikačním číslem`routeId`. Pokud je plán po odebrání prázdý, smaže ho.

`PUT /add-vehicle/{date}/{rangerId}?vehicleId={vehicleId}`
: Přidá strážci s identifikačním číslem `rangerId` do plánu na den `date` dopravní prostředek s identifikačním číslem `vehicleId`. Pokud plán neexistuje, vytvoří nový.

`PUT /remove-vehicle/{date}/{rangerId}?vehicleId={vehicleId}`
: Odebere strážci s identifikačním číslem `rangerId` do plánu na den `date` dopravní prostředek s identifikačním číslem `vehicleId`. Pokud je plán po odebrání prázdý, vymaže ho.

`POST /update-all`
: Přijímá list `PlanDto`, které aktualizuje nebo vytváří, pokud neexistují.

`POST /update`
: Přijímá `PlanDto` a aktualizuje nebo vytváří ho, pokud neexistuje.

`GET /generate/{districtId}/{startDate}`
: Získá data potřebná ke generování plánu tras pro obvod s identifikačním číslem `districtId` na týden začínající datem `startDate`, se kterými volá funkci `Generate` na `IRoutePlanGenerator` rozhraní a vrací její výsledek jako `GenerateResultDto`.

`GET /by-dates/{districtId}/{startDate}/{endDate}`
: Spojuje data docházky a plánů v obvodu s identifikačním číslem `districtId` v rozmezí dat `startDate` a `endDate` do listu `RangerScheduleDto`, který vrací.

Kontroler Strážců

Kontroler Strážců (`RangerController`) poskytuje následující HTTP *endpointy* s prefixem `api/Ranger`:

`GET /`
: Vrací strážce právě přihlášeného uživatele jako `RangerDto`.

`GET /in-district/{districtId}`
: Vrací list všech strážců v obvodu s identifikačním číslem `districtId` jako list `RangerDto`.

`POST /create`
: Vytváří a vrací nového strážce, kterého přijímá v těle požadavku ve formátu `RangerDto`.

`DELETE /delete/{rangerId}`
: Odstraní strážce s identifikačním číslem `rangerId`.

`POST /update`
: Aktualizuje a vrací strážce, kterého přijímá v těle požadavku ve formátu `RangerDto`.

Kontroler Tras

Kontroler Tras (`RouteController`) poskytuje následující HTTP *endpointy* s prefixem `api/Route`:

`GET /in-district/{districtId}`
: Vrací list všech tras v obvodu s identifikačním číslem `districtId` jako list `RouteDto`.

`POST /create`
: Vytváří a vrací novou trasu, kterou přijímá v těle požadavku ve formátu `RouteDto`.

`DELETE /delete/{routeId}`
: Odstraní trasu s identifikačním číslem `routeId`.

`POST /update`
: Aktualizuje a vrací trasu, kterou přijímá v těle požadavku ve formátu `RouteDto`.

Kontroler Dopravních prostředků

Kontroler Dopravních prostředků (`VehicleController`) poskytuje HTTP *endpointy* s prefixem `api/Vehicle`:

`GET /in-district/{districtId}`
: Vrací list všech dopravních prostředků v obvodu s identifikačním číslem `districtId` jako list `VehicleDto`.

`POST /create`
: Vytváří a vrací nový dopravní prostředek, který přijímá v těle požadavku ve formátu `VehicleDto`.

`DELETE /delete/{vehicleId}`
: Odstraní dopravní prostředek s identifikačním číslem `vehicleId`.

`POST /update`
: Aktualizuje a vrací dopravní prostředek, který přijímá v těle požadavku ve formátu `VehicleDto`.

Kontroler Uživatelů

Kontroler Uživatelů (`UserController`) poskytuje HTTP *endpointy* s prefixem `api/User`. Má `dependency-injected` služby `IAppAuthenticationService` a `IAppAuthorizationService`, které mu poskytují logiku správy uživatelů a přihlašování.

`GET /`
: Vrací právě přihlášeného uživatele jako `UserDto`.

`PUT /register-user`
: Registruje nového uživatele spojeného se strážcem, kterého přijímá v těle požadavku ve formátu `RangerDto`.

`PUT /assign-role/{userEmail}/{role}`
: Přiřazuje uživateli s emailem `userEmail` roli `role`.

`POST /signin`
: V těle požadavku přijímá `SignInRequest` a pokud je přihlášení úspěšné, vrací uživatele jako `UserDto`.

`POST /signout`
: Odhlásí přihlášeného uživatele.

3.2.2 Autentikace a autorizace

Pro definování funkcí potřebných na autentizaci a autorizaci uživatelů jsou zavedena dvě rozhraní - `IAppAuthenticationService` a `IAppAuthorizationService`, které oddělují aplikaci od implementačních detailů způsobu správy uživatelů a ověřování rolí. Aktuálně jsou rozhraní implementované využitím `ASP.NET Core Identity` a jejich služeb - `SignInManager` a `UserManager`. Díky zavedení rozhraní je možné relativně snadno přejít na jiný způsob autentizace a autorizace.

3.2.3 Hubs

Pro posílání notifikací v reálném čase máme dva *SignalR* Huby.

DistrictHub

`DistrictHub` posílá notifikace o provedených změnách zdrojů obvodu. Při nastavení připojení na Hub se klient první přidá do skupiny podle obvodu voláním funkce `AddToDistrictGroup(int districtId)`. Notifikační metody jsou čtyři - pro posílání změn o trasách, dopravních prostředcích, strážcích a zámcích. Všechny mají stejnou strukturu, která je v zobecněné verzi ilustrována v programu 13.

Program 13 Zobecněná notifikační metoda Hubu obvodu

```
// operation can be TUpdated, TAdded, TDeleted
public async Task SendTNotification(string operation,
                                    int districtId,
                                    TDto t)
{
    await Clients.OthersInGroup(districtId.ToString())
                 .SendAsync(operation, t);
}
```

RangerScheduleHub

`RangerSheduleHub` slouží k posílání notifikacích o provedených změnách v plánovacích datech. Stejně jako v `DistrictHubu` se klient při inicializaci připojení přidá do skupiny podle obvodu voláním funkce `AddToPlanGroup(int districtId)`. RangerScheduleHub má celkem tři notifikační metody:

1. `TriggerReload(int districtId)`
   - slouží k vynucování znovunačtení všech plánovacích dat.

2. `UpdatePlan(int districtId, PlanDto plan)`
   - posílá notifikaci o změně konkrétního plánu.

3. `UpdateAttendence(int districtId, AttendenceDto attendence)`
   - posílá notifikaci o změně konkrétní docházky.

3.2.4 Modul generování plánu tras

Při implementaci sestavování plánu tras se řídíme dříve provedeným návrhem (2.4). Zdrojové kódy jsou umístěné ve složce `.\App.Server\CSP`. Stěžejní třídou modulu je `RoutePlanGenerator`, který koordinuje celkový průběh a volá funkce pomocných tříd.

DataProcessor

`DataProcessor` je pomocná statická třída poskytující sadu funkcí pro zpracování dat do užitečného formátu.

Metoda `GetRouteDistributions` (viz Program 14) počítá procentuální rozložení tras pro jednotlivé strážce na základě zadaných plánů. Výsledky jsou vyjádřeny procentuálně, aby byly porovnatelné pro strážce s různým celkovým počtem prošlých tras. Výstupem je slovník `Dictionary<int, RouteDistribution>`, kde klíčem je identifikátor strážce a hodnotou objekt typu `RouteDistribution`. Ten představuje další slovník `Dictionary<int, int>`, kde klíčem je identifikátor trasy a hodnota odpovídá procentuálnímu zastoupení trasy v plánech strážce vynásobené deseti (např. hodnota 500 odpovídá 50%).

Program 14 Předpis metody `DataProcessor.GetRouteDistribution` pro získání rozložení tras strážců

```
public static Dictionary<int, RouteDistribution>
                GetRouteDistributions(List<PlanDto> plans,
                                      List<RangerDto> rangers,
                                      List<RouteDto> routes);
```

Metoda `GetBestRangersForRoutes` seřadí strážce pro každou trasu od nejvhodnějších po nejméně vhodné na základě jejich procentuálního rozložení tras (`routeDistribution`). Pokud má některý strážce danou trasu manuálně předplánovanou, je přesunut na konec seznamu.

Program 15 Předpis metody `DataProcessor.GetBestRangersForRoutes` pro získání nejvhodnějších strážců pro trasy

```
public static Dictionary<int, Rangers> GetBestRangersForRoutes(
    Dictionary<int, RouteDistribution> routeDistributions,
    List<RouteDto> routes,
    List<PlanDto> preexistingPlans);
```

Dalšími jednoduchými metadami třídy DataProcessor jsou:

- `getWorkingRangers` - vrací seznam pracujících strážců na jednotlivé dny, kterým můžeme naplánovat trasu.
- `getFixedPlans` - z manuálně naplánovaných plánů vrací fixní, které nemůžeme přeplánovat.
- `convertAssignmentToPlan` - převádí vytvořené přiřazení hodnot proměnným na plány.

RouteDeterminer

`RouteDeterminer` je pomocná třída, která pro trasy určuje s jakou frekvencí by měly být naplánovány, aby byla splněna jejich priorita v kontextu jejich minulého a manuálního naplánovaní. Je využita v třídě `VariableManager` při tvorbě proměnných. Poskytuje metody definované rozhraním `IRouteDeterminer` (viz Program 16).

Program 16 Interface `IRouteDeterminer`

```
public interface IRouteTypeDeterminer
{
    public RouteType DetermineRouteType(RouteDto route);
    public bool IsPreplanned(int routeId, int dayNumber);
}
```

VariableManager

`VariableManager` je třída na tvorbu proměnných. Má metodu `GetVariables`, která vytvoří proměnnou pro každou trasu na každý den v týdenním rozsahu a vrací pouze proměnné, kterým by mohla být přiřazena hodnota. Pro proměnné máme definovanou třídu `Variable` (viz Program 17).

Program 17 `Variable` drží informace o konkrétní proměnné.

```
public class Variable
{
    public int RouteId { get; set; }
    public int DaysFromStart { get; set; }
    public RouteType RouteType { get; set; }
    public int VariableId { get; set; } // used in solver
}
```

Contraints

Všechny podmínky dědí od abstraktní generické třídy `Constraint`, která obsahuje metodu `isSatisfied` pro určení, zda aktuální přiřazení hodnot splňuje logiku podmínky pro danou sadu proměnných (viz Program 18). Máme celkem čtyři druhy podmínek - `DifferentValueConstraint`, `FilledAttendenceConstraint`, `RequiredConstraint` a `SomeAssignedConstraint`. V Našem finálním návrhu používáme pouze podmínky typu `SomeAssignedConstraint`, protože ostatní jsou propagované v logice vytváření řešení.

Program 18 `Constraint` je abstraktní generická třída, od které dědí všechny typy podmínek.

```
public abstract class Constraint<TVariable, TDomain>
    (TVariable[] variables) where TVariable : notnull
{
    public TVariable[] Variables { get; } = variables;
    public abstract bool IsSatisfied(
            Dictionary<TVariable, TDomain> assigned
    );
}
```

Solver

`Solver` je třída která implementuje `backtracking` algoritmus navržený v pseudokódu 2.4.4. Konstruktor přijímá seznam proměnných, dva druhy domén a podmínky (viz Program 19). Jediná veřejná metoda `Solve` spouští rekurzivní funkci se samotným algoritmem a vrací výsledné přiřazení hodnot proměnným nebo `null`, pokud řešení neexistuje.

Program 19 Předpis konstruktoru třídy `Solver`, která řeší náš CSP.

```
public Solver(List<Variable> variables,
           Dictionary<int, List<int>> domainsForDays,
           Dictionary<int, List<int>> domainsForRoutes,
           Dictionary<TVariableId,
                List<Constraint<TVariableId,TDomain>>> constraints);
```

RoutePlanGenerator

`RoutePlanGenerator` implementuje rozhraní `IRoutePlanGenerator` (viz Program 20), definované pro usnadnění testování a přidání do *dependency injection* kontejneru. Funkce `Generate` vytváří nový plán tras v následujících krocích:

1. Určení fixních plánů voláním metody `DataProcessor.GetFixedPlans`.
2. Pokus o sestavení plánu, který vyhovuje manuálně naplánovaným trasám voláním privátní metody `TrySolving`:
   (a) Získání prvního druhu domén, který odpovídá nejvhodnějším strážcům pro trasy, voláním `GetBestRangersForRoutes`.
   (b) Získání druhého druhu domén, který odpovídá pracujícím strážcům na jednotlivé dny, voláním `GetWorkingRangers`.
   (c) Získání proměnných voláním `GetVariables`.
   (d) Kontrola, zda je dost pracujících strážců na naplnění priority tras, a vrácení neúspěchu, pokud není.
   (e) Sestavení podmínek podle typů proměnných.
   (f) Spuštění `Solveru` s doménami, proměnnými a podmínkami.
3. Pokud byl první pokus úspěšný, vrátíme výsledek.
4. Pokud nebyl úspěšný, proběhne druhý pokus, kde se zachovají pouze fixní plány.
5. Vrácení výsledku druhého pokusu.

Program 20 Interface IRoutePlanGenerator, který definuje metodu Generate

```
public interface IRoutePlanGenerator
{
    GenerateResultDto Generate(
        List<PlanDto> previousPlans,
        List<PlanDto> preexistingPlans,
        List<AttendenceDto> attendences,
        List<RouteDto> routes,
        List<RangerDto> rangers,
        DateOnly startDate
        );
}
```

3.2.5 Mapování modelu na databáze

Na mapování databázových tabulek na objekty aplikace byl použit Entity Framework Core. Strukturu aplikační databáze reprezentuje třída `PlannerNPContext`, která dědí od `DbContext` a definuje jednotlivé tabulky pomocí `DbSet` pro jednotlivé modely. Strukturu AuthDatabáze mapujeme využitím třídy `IdentityDbContext`. Nastavení vazeb mezi tabulkami, datových typů nebo omezení probíhá pomocí anotací přímo v modelech. Zdrojové kódy modelů a databázové kontexty jsou umístěné ve složce `.\App.Server\Model`.

3.2.6 Repositories

Všechny *repositories* implementují generické rozhraní `IGenericRepository` (viz Program 21), které definuje základní metody pro získání, přidání, úpravu nebo odstranění entit. Je možné získat list entit podle určitého filtru voláním metody Get. Rozhraní je implementováno generickou třídou `GenericRepository`, která je využita pro všechny entity kromě plánů a docházky. `PlanRepository` a `AttendenceRepository` od této třídy dědí, ale mají předefinovanou funkci `GetById`, aby bylo možné získávat plány a docházku podle data a identifikačního čísla strážce. *Repositories* všech entit jsou poskytovány v `UnitOfWork`, která obsahuje metodu `SaveAsync` na jednotné uložení změn do databáze.

Program 21 Generický interface IGenericRepository, který definuje základní operace pro práci s daty typu `T`.

```
public interface IGenericRepository<T> where T : class
{
    Task<IEnumerable<T>> Get(Expression<Func<T, bool>>? filter,
                             string includeProperties);
    Task<T?> GetById(params object[] id);
    void Add(T entity);
    void Update(T entityToUpdate);
    void Delete(T entityToDelete);
}
```

3.3 Spuštění

Aplikace byla vyvíjena ve Visual Studiu a testována na Windows.

Požadavky

1. *.NET SDK* (verze 8.0)[5]
2. *MariaDB* (verze 11.x)[6]
3. *Node.js* (verze 20.11) a *npm*[7]

Konfigurace databáze

V souboru `.\App.Server\appsettings.json` je potřeba aktualizovat sekce `ConnectionStrings` a `SeedAdminUser`.

- V `ConnectionStrings` upravit přihlašovací údaje pro MariaDB server.
- V `SeedAdminUser` upravit údaje admina zahrnující:
  - Email – musí být validní emailová adresa
  - Heslo – musí odpovídat konvencím pro hesla (jedno velké, malé písmeno, číslo a znak).

Pro vytvoření databáze je nutné spustit následující příkazy:
V *Package Manager Console* (Visual Studio):

```
> Update-Database -context PlannerNP
> Update-Database -context ApplicationIdentity
```

Alternativně v terminálu s nainstalovaným EF[8]:

```
> dotnet ef database update --context PlannerNP
> dotnet ef database update --context ApplicationIdentity
```

Spuštění při vývoji

Po spuštění backendu se automaticky připojí frontendová aplikace a otevře se Vite terminál s odkazem na localhost adresu, na které bude aplikace k dispozici. Spuštění backendu ze složky `.\App.Server`:

```
> dotnet run
```

[5] `https://dotnet.microsoft.com/en-us/download`
[6] `https://mariadb.org/download/`
[7] `https://nodejs.org/en/download`
[8] `https://www.nuget.org/packages/dotnet-ef`

Sestavení a nasazení

První je potřeba sestavit frontendovou část spuštěním následujícího příkazu ve složce `.\app.client`:

```
> npm run build
```

Tím se ve složce `.\app.client\dist` vytvoří statické soubory frontendové aplikace, které zkopírujeme na backend do složky `.\App.Server\wwwroot`.

Poté spustíme *release build* aplikace ze složky `.\App.Server`:

```
> dotnet publish -c Release -o ./publish
```

Tím se do složky `.\App.Server\publish` umístí aplikace připravená na nasazení. Alternativně je možné frontendovou část aplikace nasadit odděleně.[9]

[9] `https://vite.dev/guide/static-deploy.html`

4 Testování

V této kapitole představíme způsob automatizovaného testování frontendu i backendu a popíšeme výsledky uživatelského testování.

4.1 Frontend

Pro testování *React* frontendové aplikace byl zvolen testovací framework *Vitest*[1] spolu s balíčky knihovny *React Testing Library*[2]. Jednotlivé testy jsou umístěné uvnitř projektu u testovaných funkcí nebo komponent ve složkách `__tests__`. Mohou být spuštěny z adresáře `.\app.client` pomocí příkazu:

```
> npm run test
```

Testy vztahující se k jedné komponentě nebo funkci jsou seskupené do jednoho `describe` bloku a vyjádřené pomocí výrazu `it` (viz Program 22). Výraz `it` byl zvolen namísto aliasu `test` pro lepší čitelnost. Při testování funkcí ověřujeme správné výstupy nebo správná volání backendu. U komponent, které jsou vizuální součástí uživatelského rozhraní, testujeme navíc přítomnost nebo nepřítomnost jednotlivých prvků v DOM (*Document Object Model*) a správné reakce na uživatelské akce. Jelikož funkcionalita většiny komponent silně závisí na datech získaných z jednotlivých kontextů, vytváříme jejich mocky využitím `vi.mock()` a `vi.fn()`. Toto nám umožňuje testovat reakce na různá data z kontextů a testovat funkcionalitu odděleně.

Celkem bylo implementováno *124* testů, které pokrývají klíčové chování hlavních komponent a funkcí komunikujících s backendem.

Program 22 Struktura testů frontendu

```
describe('Component', () => {
    it('does/should/renders/calls X', () => {
        ...
    });
    ...
})
```

4.2 Backend

Pro testování *ASP.NET Core* backendu byl zvolen nástroj *xUnit*[3]. Všechny testy jsou umístěné v odděleném projektu ve složce `.\Tests`, ze které mohou být spuštěny pomocí příkazu:

```
> dotnet test
```

[1] `https://vitest.dev/`
[2] `https://testing-library.com/`
[3] `https://xunit.net/`

Pro psaní testů byl použit *Arrange-Act-Assert Pattern*, který rozděluje každý test do třech oddělených částí: příprava vstupů (*arrange*), spuštění testované funkce (*act*) a ověření výsledků (*assert*). Celkem bylo implementováno *194 unit* a integračních testů.

Unit testy pokrývají hlavní business logiku, která se nachází v metodách kontrolerů a modulu generování plánu tras. Integrační testy ověřují správné volání HTTP metod kontrolerů s využitím *WebApplicationFactory*[4], která spouští aplikaci a nahrazuje reálné za *in-memory* databáze s testovacími daty. Posílání HTTP požadavků může být ručně testováno při spuštění ve vývojovém režimu s reálnou databází využitím nástroje *Swagger UI*, který poskytuje webové rozhraní s voláním všech metod.

4.3 Uživatelské testování

Cílem uživatelského testování je ověřit použitelnost uživatelského rozhraní, získat zpětnou vazbu a otestovat funkčnost aplikace. Před začátkem testování jsme aplikaci naplnili testovacími daty pro jeden obvod s podobným počtem strážců, tras a dopravních prostředků jako by bylo při reálném používání. Manuálně jsme voláním HTTP metod registrovali uživatelské účty pro roli strážce i vedoucího obvodu.

4.3.1 Testování funkcionalit pro strážce

Pro otestování aplikace pro roli strážce byla využita kombinace řízených testů (udávání úkolů) a volného průběhu, kdy uživatelé navigovali funkce bez externí pomoci. Celkem se zúčastnilo sedm uživatelů z okruhu strážců a vedoucích, ale i uživatelů bez znalosti domény.

Formát testování

Byla vytvořena produkční verze aplikace (*realease build*) a spuštěna na lokální síti. Pomocí nástroje *ngrok*[5] byla generována dočasná veřejná URL adresa, jejichž požadavky byly směrovány na aplikaci běžící lokálně. Tato dočasná adresa byla poskytnuta uživatelům pro spuštění na vlastních zařízeních. Tímto způsobem jsme také nepřímo otestovali provádění změn v reálném čase.

Jednotlivé uživatelské testy měli následující průběh:

- Uživateli byla poskytnuta adresa aplikace a testovací přihlašovací údaje s rolí strážce.
- Uživatel dostal prostor na zorientování se ve funkcích aplikace bez externí pomoci.
- Uživateli byly zadány úkoly, které simulovaly základní reálné využívání aplikace.
- Uživatel vyplnil dotazník, kde ohodnotil složitost jednotlivých úkolů, celkové vlastnosti aplikace a přidal poznámky nebo očekávání ohledně dalšího vývoje aplikace.

[4] `https://learn.microsoft.com/en-us/aspnet/core/test/integration-tests`
[5] `https://ngrok.com/`

Dotazník

Dotazník je strukturovaný do tří částí. V první části jsou otázky mířené na upřesnění typu uživatele, v druhé je hodnocení obtížnosti splnění jednotlivých úkolů a poslední část hodnotí aplikaci jako celek. Na konci dotazníku mohl uživatel předat svoje myšlenky formou poznámky.

1. Část první
   (a) Pracujete v národním parku? Pokud ano, jaká je Vaše pozice?
   (b) Na jakém zařízení jste testovali aplikaci?
2. Část druhá
   Ú1 Představte si, že v den `X` chcete jít na trasu `T`, zapište tuto informaci do aplikace. Jak obtížné bylo splnit tento úkol?
   1 = velmi obtížné, 10 = jednoduché.
   Ú2 Představte si, že v den `X` plánujete mít dovolenou, zapište tuto informaci do aplikace. Jak obtížné bylo splnit tento úkol?
   1 = velmi obtížné, 10 = jednoduché.
   Ú3 Zkuste zjistit, jaký strážce se dnes nachází na trase `T`. Jak obtížné bylo splnit tento úkol?
   1 = velmi obtížné, 10 = jednoduché.
   Ú4 Podívejte se, jakou máte zítra naplánovanou trasu. Jak obtížné bylo splnit tento úkol?
   1 = velmi obtížné, 10 = jednoduché.
3. Část třetí
   (a) Jsou texty a popisky v aplikaci čitelné?
   1 = velmi nečitelné, 10 = velmi čitelné.
   (b) Jak hodnotíte celkovou přehlednost aplikace (rozmístění prvků, navigování)?
   1 = málo přehledná, 10 = velmi přehledná.
   (c) Jak intuitivní bylo celkové ovládání aplikace?
   1 = velmi neintuitivní, 10 = velmi intuitivní.

Výsledky testování

Výsledky z dotazníků jsou uspořádané do tabulky 4.1. Celkem se testování zúčastnilo sedm uživatelů - tři strážci (`S1-S3`), jeden vedoucí obvodu (`VO`), jeden vedoucí strážní služby (`VS`) a dva uživatelé bez znalosti domény (`N1-N2`). Strážci svoji docházku a trasy většinou plánují z mobilních zařízení, a proto byli uživatelé spíše směrováni testovat na nich. Dva uživatelé, kteří testovali na počítačovém zařízení jsou označeni příponou `-P`.

Během testování se většina uživatelů dokázala v aplikaci snadno zorientovat a plnit úkoly bez potřeby nápověd. I ti, kteří si napoprvé nebyli jistí, nakonec pochopili fungování aplikace a bylo pro ně její ovládání jednoduché. Na konci

dotazníku měli uživatelé možnost přidat poznámku a vyjádřit, jaké další funkcionality by v aplikaci uvítali. Uživatelé **VO** a **N1** zmínili, že by v aplikaci ocenili alternativní navigování mezi dny, a to možnost výběru data z kalendáře nebo tlačítka navigující na dnešní den. Uživatel **S3** uvedl, že by ocenil možnost přepnutí na širší přehled i na mobilní obrazovce - například na celý týden.

Uživatel	Ú1	Ú2	Ú3	Ú4	Čitelnost	Přehlednost	Intuitivnost
VO	10	10	10	10	7	8	10
VS-P	9	10	9	10	8	7	9
S1	10	10	10	10	10	10	10
S2	10	10	9	10	9	10	10
S3	10	10	10	10	9	10	10
N1	10	7	10	10	10	10	6
N2-P	9	10	10	10	9	9	9
Průměr	9.71	9.57	9.71	10	8.86	9.14	9.14

Tabulka 4.1 Výsledky uživatelského testování funkcionalit pro strážce.

4.3.2 Testování funkcionalit pro vedoucího obvodu

Testování pro uživatele v roli vedoucího obvodu bylo během vývoje prováděno pomocí konzultací s vedoucím strážního obvodu Stožec (`VO`) a vedoucím strážní služby a ochrany přírody (`VS`). Během setkání byl prezentován průběžný stav aplikace a dostupné funkcionality. Při závěrečném testování byl použit stejný přístup jako v sekci 4.3.1, kdy vedoucí používali aplikaci na vlastních zařízeních a hodnotili její použitelnost a funkčnost v dotazníku.

Dotazník

Dotazník byl strukturován stejným způsobem jako při testování funkcionalit pro strážce (viz 4.3.1). Výjimkou je druhá část, kde byli zadávány úkoly specifické pro používání aplikace vedoucím:

Ú1 Upravte prioritu trasy `T`, aby byla procházena každý den. Jak obtížné bylo splnit tento úkol?
1 = velmi obtížný, 10 = jednoduchý.

Ú2 Přidejte do kolekce nový dopravní prostředek. Jak obtížné bylo splnit tento úkol?
1 = velmi obtížný, 10 = jednoduchý.

Ú3 Vytvořte kompletní plán na jeden celý týden. Jak obtížné bylo splnit tento úkol?
1 = velmi obtížný, 10 = jednoduchý.

Ú4 Uzamkněte plán pro úpravy od strážců. Jak obtížné bylo splnit tento úkol?
1 = velmi obtížný, 10 = jednoduchý.

Výsledky testování

Výsledky z dotazníků jsou uspořádané do tabulky 4.2. Protože velká množina funkcionalit pro vedoucí obvodů nejsou dostupná pro mobilní obrazovky, bylo testování prováděno pouze na počítačových zařízeních. Při plnění úkolu **Ú4** narazil uživatel **VO** na chybu, kde se na jeho zařízení neměnily ikony pro zamykání a odemykání dnů. Ikony jsou řešené *Unicode* znaky a na jiných testovacích zařízeních se problém neprojevil. Jako dočasné řešení byly k zámkům přidány textové nápovědy a v další iteraci vývoje dojde k návrhu nové reprezentace bez použití *Unicode* znaků. Uživatelé se shodli, že aplikace je snadno ovladatelná, ale určité ovládací prvky by mohly být větší či jinak zvýrazněné. Zmíněné byly větší ikony zámků a barevně odlišené tlačítko přepnutí na denní plán.

Uživatel	Ú1	Ú2	Ú3	Ú4	Čitelnost	Přehlednost	Intuitivnost
VO	10	10	6	5	8	8	7
VS	9	9	8	9	8	9	10
Průměr	9.5	9.5	7	7	8	8.5	8.5

Tabulka 4.2 Výsledky uživatelského testování vedoucími.

Závěr

Cílem bakalářské práce bylo navrhnout, implementovat a otestovat aplikaci na plánování služby strážců národního parku. Nejprve jsme na základě rozhovorů se strážci specifikovali funkční a nefunkční požadavky a sestavili případy užití týkající se správy zdrojů, zobrazovaní plánu služby a jeho vytváření. Pro aplikaci jsme navrhli uživatelské rozhraní a architekturu jako *single-page* webovou aplikaci využívající knihovnu *React*. Ta získává a upravuje data voláním API kontrolerů backendu *ASP.NET Core*. Pro ukládání dat byly představeny dvě MariaDB databáze. Navrhli jsme také algoritmus na generování plánu tras (F17) jako řešení problému s omezujícími podmínkami (CSP). Popsali jsme implementaci navrženého řešení a využité technologie. Nakonec probíhalo automatizované a uživatelské testování aplikace.

Pro nasazení aplikace do produkčního prostředí strážcům Národního parku Šumava je potřeba další vývoj. Ten zahrnuje nahrazení interních služeb poskytujících autorizaci a autentizaci za služby využívající LDAP (*Lightweight Directory Access Protocol*) organizace. Dále je v reakci na výsledky uživatelského testování potřeba přidat alternativní způsob navigování mezi dny v denním plánu (4.3.1) a navrhnout ikony pro reprezentaci zámků, pro napravení odhalené chyby (4.3.2).

Mezi další možnosti rozšíření, které mohou být součástí budoucího vývoje, patří zavedení podpory pro volitelné požadavky VF18–VF22, které byly v této práci zpracovány pouze ve fázi analýzy, a také optimalizování a rozšíření algoritmu pro generování plánu tras.

Literatura

1. ČESKO. *§ 15 zákona č. 114/1992 Sb., České národní rady o ochraně přírody a krajiny* [online]. [cit. 2025-01-22]. Dostupné z: `https://www.e-sbirka.cz/sb/1992/114/2025-03-01#par_15`.
2. Brown, Simon. *The C4 model for visualising software architecture* [online]. [cit. 2025-04-15]. Dostupné z: `https://c4model.com/`.
3. Millett, Scott; Tune, Nick. *Patterns, Principles, and Practices of Domain-Driven Design*. Indianapolis, Indiana: Wrox, a Wiley brand, 2015. Wrox programmer to programmer. ISBN 978-1-118-71470-6.
4. Barták, Roman. *On-line Guide to Contraint Programming* [online]. Prague, 1998 [cit. 2025-04-20]. Dostupné z: `https://kti.mff.cuni.cz/~bartak/constraints/`.
5. Rossi, Francesca; Van Beek, Peter; Walsh, Toby. Constraint programming. *Foundations of Artificial Intelligence*. 2008, roč. 3, s. 181–211.

Seznam obrázků

1.1 Doménový model 10
1.2 Use case diagram 14

2.1 Navigační schéma 21
2.2 Wireframe komponenty plánu strážce v případě, že nepracuje . . . 23
2.3 Wireframe komponenty plánu strážce v případě, že pracuje 23
2.4 Wireframe komponenty plánu strážce, který je editován 23
2.5 Wireframe denního plánu pro mobilní obrazovku 24
2.6 Wireframe dvoutýdenního plánu 24
2.7 Wireframe vygenerovaného plánu tras 25
2.8 Wireframe obsahu plánu tras s vygenerovanou trasou 25
2.9 Wireframe obsahu plánu tras bez vygenerované trasy 25
2.10 Barevné rozlišení priorit tras 25
2.11 Wireframe stránky Správa zdrojů 27
2.12 Wireframe komponenty zdroje 27
2.13 Wireframe komponenty upravovaného zdroje 27
2.14 C4 diagram Level 2: Container diagram 28
2.15 C4 diagram Level 3: Component diagram frontendu 29
2.16 C4 diagram Level 2: Component diagram backendu 31