ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ
KATEDRA ELEKTRONIKY A INFORMAČNÍCH TECHNOLOGIÍ

BAKALÁŘSKÁ PRÁCE

Řízení kolejiště pomocí webové aplikace

Kryštof Trkovský **2021**

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta elektrotechnická

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Kryštof TRKOVSKÝ**
Osobní číslo:	**E18B0037P**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a telekomunikace**
Téma práce:	**Řízení kolejiště pomocí webové aplikace**
Zadávající katedra:	**Katedra elektroniky a informačních technologií**

Zásady pro vypracování

Navrhněte a realizujte řídící aplikace pro katedrální kolejiště.

1. Aplikace bude založená na webových technologiích
2. Zvolené řešení bude respektovat stávající nízko-úrovňovou řídící strukturu kolejiště

Diskutujte vhodnost zvolených technologií, příp. navrhněte srovnejte různá řešení, pokud je to možné.

Rozsah bakalářské práce: **30 – 40 stran**
Rozsah grafických prací: **podle doporučení vedoucího**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. YANK, Kevin a Cameron ADAMS. Začínáme s JavaScriptem: názorný průvodce tvorbou WWW stránek. Brno: Zoner Press, 2008. Encyklopedie webdesignera. ISBN 978-80-86815-94-7.
2. MACRAE, Callum. Vue.js Up & Running. O'Reilly Media, 2018. ISBN 978-1-491-99724-6.

Vedoucí bakalářské práce: **Ing. Petr Weissar, Ph.D.**
Katedra elektroniky a informačních technologií

Datum zadání bakalářské práce: **9. října 2020**
Termín odevzdání bakalářské práce: **27. května 2021**

L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Doc. Ing. Jiří Hammerbauer, Ph.D.
vedoucí katedry

V Plzni dne 9. října 2020

Abstrakt

V práci je řešena realizace webové aplikace, jejíž účel je dálkové ovládání modelového kolejiště. Aplikace poskytuje kompaktní řešení pro sledování provozu, definici jízdních řádů a přímé řízení kolejiště. Vybrané technologie sledují současné trendy při vývoji obdobných dynamických aplikací.

Projekt se dělí na dvě části - klient a server. Obě části byly realizovány v programovacím jazyku Typescript. Klient využívá framework Vue.js, server je založen na frameworku Koa.js. Uchování dat zajišťuje SQL databáze MariaDB. Pro komunikaci mezi serverem a klientem je využito REST API a knihovny Socket.io.

Výsledkem je aplikace, která funguje na serveru univerzity a v budoucnu bude využita pro kontrolu provozu kolejiště. Možnosti aplikace odpovídají současnému stavu nízkoúrovňového řídícího software.

Klíčová slova

webová aplikace SPA, modelové kolejiště, Vue.js, Koa.js, Socket.io

Abstract

The thesis is focused on implementation of a web application. The purpose is remote control of model railway. The application provides compact solution for traffic monitoring, timetable definition and direct control of the track. Selected technologies follows current trend in development of similar web applications.

The project is divided into two parts - client and server. Both parts were implemented in Typescript language. Client uses the Vue.js framework, server is based on Koa.js framework. Data storage is provided by SQL database MariaDB.

The result is an application running on the university server. In future will be used to control operation of model railway. The capabilities of the application corresponds to the current state of low-level control software.

Keywords

web application SPA, model railway, Vue.js, Koa.js, Socket.io

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této bakalářské práce.

Dále prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

V Plzni dne 26. května 2021

Kryštof Trkovský

. . . . . .

Podpis

Obsah

Seznam obrázků **viii**

Seznam tabulek **ix**

Seznam symbolů a zkratek **x**

1 Úvod **1**

2 Stávající softwarové řešení **2**

3 Výběr knihoven a programovacího jazyku **3**
3.1 Klient . . . 3
3.1.1 Programovací jazyk . . . 3
3.1.2 Framework . . . 3
3.1.3 Komponenty . . . 4
3.2 Server . . . 4
3.2.1 Programovací jazyk . . . 4
3.2.2 Framework . . . 5
3.3 Komunikace . . . 5
3.4 Ostatní . . . 6

4 Vývojové nástroje **7**
4.1 NPM . . . 7
4.2 TypeScript . . . 7
4.3 ESLint . . . 8
4.4 SASS . . . 8
4.5 Webpack . . . 8
4.6 Vue CLI . . . 9
4.7 Docker . . . 9

5 Realizace webové aplikace **11**
5.1 Struktura aplikace . . . 11
5.2 Server . . . 12

5.3 Klient . . . 13
5.4 Uživatelské role . . . 13
5.5 Popis API, sdílené typy . . . 14
5.6 Klient . . . 16
5.6.1 Service . . . 16
5.6.1.1 API . . . 16
5.6.1.2 Socket . . . 16
5.6.2 Router . . . 17
5.6.3 Store . . . 17
5.6.4 Výjimky . . . 17
5.6.5 Stránky . . . 18
5.6.5.1 Úvodní stránka . . . 18
5.6.5.2 O kolejišti . . . 18
5.6.5.3 Zdroje . . . 19
5.6.5.4 Jízdní řády . . . 20
5.6.5.5 Administrace - uživatelé . . . 21
5.6.5.6 Administrace - jízdní řády . . . 21
5.6.5.7 Administrace - chyby . . . 24
5.6.5.8 Administrace - hardware . . . 25
5.6.5.9 Uživatel - přihlášení, registrace . . . 26
5.6.5.10 Uživatel - nastavení . . . 27
5.6.5.11 Chybová stránka 404 . . . 28
5.6.6 Komponenty . . . 29
5.6.6.1 Mapa - základ . . . 29
5.6.6.2 Mapa - úvodní stránka . . . 29
5.6.6.3 Mapa - výběr trasy . . . 29
5.6.6.4 Jízdy . . . 30
5.6.6.5 Jízdní řády . . . 31
5.6.6.6 Plánování . . . 32
5.6.6.7 Pozadí . . . 32
5.6.6.8 Tabulka chyb . . . 32
5.6.6.9 Menu - jízdní řády . . . 33
5.6.6.10 Menu - vlaky . . . 33
5.6.6.11 Detail vlaku . . . 33
5.6.7 Grafický vzhled . . . 33
5.7 Server . . . 34
5.7.1 Databáze a data . . . 34
5.7.1.1 Migrace a seed . . . 34
5.7.2 Autentizace a autorizace uživatele . . . 35
5.7.3 Controller . . . 35

5.7.4 Hardware . . . 35
5.7.5 Service . . . 36
5.7.6 Model . . . 36
5.7.7 Konfigurace . . . 36
5.7.8 Výjimky a HTTP stavové kódy . . . 36
5.7.9 Cron . . . 36
5.8 Produkční prostředí . . . 37

6 Možnosti zlepšení a další vývoj **39**

7 Závěr **40**

Reference, použitá literatura **41**

Přílohy **43**

A Příkazy a konfigurace **43**
A.1 Nastavení projektu pro vývoj . . . 43
A.1.1 S dockerem . . . 43
A.1.2 Bez dockeru . . . 44
A.1.3 Vývoj . . . 44
A.1.4 Vývoj server . . . 45
A.1.5 Vývoj klient . . . 45
A.1.6 Test produkčního nastavení docker . . . 45
A.1.7 Nastavení produkčního serveru . . . 46
A.1.8 Konfigurace . . . 46

Seznam obrázků

4.1 Příklad kódu SASS 8
4.2 Příklad kódu CSS 8
4.3 Princip slučování kódu nástrojem webpack. Převzato z [21]. 9
4.4 Zkrácený Dockerfile pro vývoj Node.js aplikace 10

5.1 Adresářová struktura projektu 12
5.2 Diagram knihoven server 12
5.3 Diagram knihoven klient 13
5.4 Příklad nastavení práv pro uživatelskou roli editor 14
5.5 Úvodní stránka. 18
5.6 Strana o kolejišti. 18
5.7 Strana se zdroji. 19
5.8 Strana s následujícími jízdami. 20
5.9 Strana s nastavením uživatelů. 21
5.10 Strana s nastavením jízd. 22
5.11 Strana s nastavením jízdních řádů. 22
5.12 Strana s nastavením plánování. 22
5.13 Průvodce přidáním - jízda. 23
5.14 Průvodce přidáním - jízdní řád. 23
5.15 Průvodce přidáním - plán. 24
5.16 Strana s zobrazením chyb. 24
5.17 Strana s nastavením hardware. 25
5.18 Strana s nastavením zpoždění dojezdu vlaků. 26
5.19 Strana s nastavením pozice vlaků. 26
5.20 Strana s přihlášením uživatele. 27
5.21 Strana s registrací uživatele. 27
5.22 Strana s nastavením uživatele. 28
5.23 Strana s chybou 404. 28
5.24 Komponenta s nastavením jedné jízdy. 30
5.25 Strana s nastavením jednoho jízdního řádu. 31
5.26 Komponenta s nastavením jednoho plánu - opakující se. 32
5.27 Komponenta s nastavením jednoho plánu - neopakující se. 32

5.28 Detail vlaku 33
5.29 Diagram vazeb mezi databázovými tabulkami. 34
5.30 Diagram produkčního prostředí 37

Seznam tabulek

2.1 Přehled paketů přijímaných programem TCP Server Train TT 2

5.1 Přehled adres REST API 15

Seznam symbolů a zkratek

API	Application Programming Interface - Rozhraní pro programování aplikací
CAN	Controller Area Network - Sériová datová sběrnice
CMS	Content Management System - Systém pro správu obsahu
CSS	Cascading Style Sheets - Kaskádové styly
CSV	Comma Separated Values - Hodnoty oddělené čárkami
DOM	Document Object Model - Objektový model dokumentu
ES	ECMA Script
HTML	Hypertext Markup Language - Značkovací jazyk pro tvorbu stránek s hypertextovými odkazy
HTTP	Hypertext Transfer Protocol - Protokol pro přenos hypertextových dokumentů
HTTPS	Hypertext Transfer Protocol Secure - Zabezpečený protokol pro přenos hypertextových dokumentů
HW	Hardware
IP	Internet Protocol - Internetový protokol
JS	JavaScript
JSON	JavaScript Object Notation - JavaScriptový objektový zápis
JWT	JSON Web Token
MVC	Model View Controller
NPM	Node Package Manager - Systém pro zprávu balíčků Node.js
ORM	Object Relational Mapping - Objektově relační mapování
PC	Personal Computer - Osobní počítač
REST	Representational State Transfer
SPA	Single Page Application - Jednostránková aplikace
SVG	Scalable Vector Graphics - Škálovatelná vektorová grafika
SW	Software
TCP	Transmission Control Protocol - Protokol pro řízení přenosu
TLS	Transport Layer Security - Zabezpečená přenosová vrstva
TS	TypeScript
USB	Universal Serial Bus - Univerzální sériová sběrnice

1

Úvod

Práce se zabývá návrhem vhodných technologií pro realizaci webové aplikace a následně samotnou realizací. Cílem aplikace je možnost vzdáleného ovládání kolejiště, které je umístěné na katedře elektroniky a informačních technologií.

Hlavními vlastnostmi aplikace jsou: zobrazení pozic vlaků na mapě, plánování odjezdů vlaků, definice posloupností s jednotlivými odjezdy, které je možné spouštět v libovolnou dobu a přímé ovládání prvků kolejiště. Aplikace umožňuje správu uživatelů a jejich rolí, kdokoli může sledovat aktuální dění na kolejišti a po přihlášení je možné podle přiřazených práv s kolejištěm operovat. Dalším účelem aplikace je reprezentace dlouhodobého projektu kolejiště, ať už před veřejností, tak před studenty, kteří mohou dále pracovat na rozvoji.

Aplikace spolupracuje s již hotovým softwarem, který komunikuje s řídícími jednotkami kolejiště a zajišťuje základní provoz. Webové technologie poskytují mnoho výhod oproti současnému řešení. Například neustálá dostupnost sledování kolejiště ve webovém prohlížeči jak na PC, tak na mobilním telefonu a tedy možnost řízení téměř kdekoli a kdykoli.

V současné době existují produkty pro ovládání kolejiště ve formě hardware nebo software. Obojí je omezeno na prostor u kolejiště. Podobné řešení využívající webových technologií nebylo dohledáno.

V práci jsou popsány důvody výběru jednotlivých knihoven a principy fungování částí webové aplikace, krátce jsou zmíněny možnosti produkčního nasazení. V práci nejsou popsány příklady používání jednotlivých knihoven, ty je možné snadno nalézt v odkazech na dokumentaci, nebo přímo v kódu aplikace.

2

Stávající softwarové řešení

Současné řešení je spuštěné na stolním počítači u kolejiště a přes USB/CAN převodník komunikuje s řídícími jednotkami. Skládá se ze samostatných celků: Visual Debug Control Train TT, Timetable Control Train TT a TCP Server Train TT. Komunikace mezi nimi probíhá pomocí TCP paketů.

TCP Server Train TT přijímá TCP pakety a překládá je na instrukce pro řídící jednotky. Zároveň udržuje provoz kolejiště a hlídá základní kolize jako např. odeslání více instrukcí jednomu vlaku. V současné době jsou implementovány tyto funkce:

Packet	Parametr	Výstup	Funkce
train_move	název, rychlost, směr		rozjetí vlaku
train_function	název, světla		rozsvícení světel
train_move_to_place	název, rychlost, směr, cíl, čas		vlak na místo
unit_instruction	instrukce		pro jednotky
unit_info		instrukce	pro jednotky
occupancy_section		úsek=hodnota	obsazení sekce
unknown			výpis do konzole

Tabulka 2.1: Přehled paketů přijímaných programem TCP Server Train TT

Jednotlivé instrukce pro jednotky jsou: zapnutí zdroje, vypnutí zdroje, restart H můstku, restart mikroprocesoru, prodleva odesílaných proudů.

Visual Debug Control slouží k přímému ovládání vlaků a sledování obsazenosti kolejiště. Tato funkce bude nahrazena webovou aplikací, ale může nalézt své uplatnění při odlaďování nového hardware, protože je dostupná na PC přímo u kolejiště.

Timetable Control Train TT poskytuje jednoduchou funkcionalitu plánování odjezdů vlaků. Jízdní řád je definován CSV souborem. Jeho úpravy nemusí být bez pročtení dokumentace zcela jednoduché, a proto funkce tohoto programu bude nahrazena webovou aplikací.

Softwarová vrstva a provedení kolejiště jsou detailněji popsány v bakalářské práci „Řídící software pro modelové kolejiště“ [1].

3

Výběr knihoven a programovacího jazyku

Tato kapitola se zabývá popisem procesu výběru použitých knihoven a jejich porovnáním. Může se zdát, že projekt obsahuje velké množství kódu třetí strany, ale faktem zůstává, že bez již postaveného základu, který je mnohokrát ověřen v praxi, by bylo téměř nemožné vytvořit aplikaci s podobnými vlastnostmi.

Účel těchto knihoven je obvykle zjednodušit konstrukce programovacího jazyka, řešení opakujících se úkolů a ošetření běžných bezpečnostních problémů.

Pro zvolené prostředí vzniklo nepřeberné množství knihoven, balíčků a frameworků. Výběr je ovlivněn osobní zkušeností a předpokladem, že nejpoužívanější balíčky jsou nejlépe ověřené a vhodné pro nasazení. Popularita je srovnána ve veřejných github repozitářích a na stránkách NPM [2].

Všechny použité knihovny jsou open source.

3.1 Klient

3.1.1 Programovací jazyk

Výběr pro klientskou stranu není příliš náročný. Jediné v současnosti použitelné přístupy jsou JS - JavaScript a WebAssembly. WebAssembly je relativně nová technologie, která by v tomto případě nyní nepřinášela velký benefit, spíše problémy s kompatibilitou [3]. Takže jako základní jazyk byl zvolen JS. Ovšem samotný kód byl psaný v jazyku TS - TypeScript a následně přeložen do JS z důvodů popsaných v kapitole 4.

3.1.2 Framework

Historicky nejznámější front-end knihovnou je jQuery. Modernější volbou jsou dnes progresivní frameworky jako Angular, React a Vue.

Pro tento projekt byla volba progresivního frameworku zřejmá z důvodu, že web vyžaduje intuitivní konfiguraci a okamžité reakce na změny přicházející z hardware. Těchto vlastností by

bylo možné dosáhnout i použitím jednodušších nástrojů, jako jQuery, ale výsledný kód by byl těžko čitelný a do budoucna složitě rozšiřitelný [4].

Knihovna jQuery se dá popsat spíše jako nadstavba nad JS. Zpřehledňuje manipulaci s HTML dokumentem, podporuje základní animace a obsahuje nástroje pro AJAX komunikaci. Použití je vhodné spíše pro menší weby, které nevyžadují velké množství dynamických prvků [5].

Principy zmíněné trojice progresivních frameworků jsou podobné. Umožňují kód oddělit od DOMu a tím zpřehlednit kód. Vývoj v nich je založen na rozdělení částí webu do menších částí - komponent, které mezi sebou komunikují. Obvykle jsou využity pro tvorbu SPA, což je i tento případ.

Volbu pro konkrétní projekt ovlivňují spíše osobní preference a předchozí zkušenosti. Z práce „Srovnání progresivních frameworků Vue.js, React a Angular“ [6] vychází pro nové projekty nejlépe Vue.js díky kvalitě dokumentace, efektivnosti a čitelnosti kódu.

3.1.3 Komponenty

Knihovny znovupoužitelných komponent poskytují běžně používané prvky jako jsou vyskakovací okna, notifikace, nástroje pro rozvržení stránky a další. Zároveň zajišťují, aby byl celkový grafický design jednotný a uživatelsky přívětivý. Za nevýhodu využití takové knihovny lze považovat to, že všechny vytvořené weby se mohou podobat a nejsou zcela unikátní. Tvorba graficky unikátního webu, ale v tomto případě není primárním cílem.

Často využívaný projekt se jmenuje Bootstrap[1]. Pro potřeby Vue.js vznikla knihovna komponent BootstrapVue[2]. V době psaní práce vyšlo Vue.js ve své třetí verzi, které BootstrapVue nepodporuje. Proto se omezil výběr pouze na Element Plus[3] a Quasar[4].

Obě knihovny vychází z designového jazyku vyvinutého společností Google - material design. Design je použit ve většině produktů Googlu, což je výhodné pro uživatele, kteří jsou na něj zvyklí. Po vydání Vue.js 3.0 se nejrychleji adaptoval Quasar, a proto byl zvolen.

3.2 Server

3.2.1 Programovací jazyk

Na straně serveru můžeme vybírat z větší nabídky, než na straně klienta. V dnešní době jsou populární jazyky jako PHP, Python, JavaScript, C#, Java, Go, C++ a další.

V případě tohoto webu, není předpokládána příliš velká návštěvnost, proto výkon není hlavní rozhodující faktor, takže nejsou vyžadována řešení jako C++, nebo Go. Zároveň pro

[1] https://getbootstrap.com
[2] https://bootstrap-vue.org
[3] https://element-plus.org
[4] https://quasar.dev

běh aplikace se předpokládá hosting v rámci univerzity, kde není problém nainstalovat jakékoli běhové prostředí, takže není nutné omezovat se na nejrozšířenější PHP.

S programovacím jazykem přichází celý „ekosystém“ balíčků, z toho důvodu je velmi výhodné použít stejný programovací jazyk na obou stranách. Balíčky jsou distribuovány správcem balíčků NPM. Navíc díky TS je možné opakovaně využít definované typy pro přenos dat. Z těchto důvodů padla volba na Node.js [7] [8].

3.2.2 Framework

Node.js frameworky lze dělit na kompletní MVC - pro vývoj backend i frontend, dále MVC pouze pro backend a určené na tvorbu REST API. Jistě nejpopulárnější volba je Express, dále je zajímavé kompletní řešení Sails, nebo Koa.

Express lze použít jednoduše a zároveň s využitím balíčků dobře škálovat. Nejvíce vhodný je pro vývoj REST API, který nevyžaduje pokročilou strukturu. Jeho architektura je vcelku blízká nativnímu vývoji v Node.js. Architekturu aplikace nechává na vývojáři [9].

Koa má velmi podobné vlastnosti jako Express. Komunita není tak rozsáhlá, proto nedisponuje takovým rozsahem balíčků jako Express, ale nakonec byl zvolen právě tento framework z důvodu, že je modernější a podporuje ES2015 async funkce, což velmi zpřehledňuje kód [10].

Sails využívá MVC architektury a integruje ORM pro propojení s databází, tím je Sails předurčen pro použití na velmi rozsáhlých projektech [11]. Pro tento projet by byl příliš rozsáhlý a pravděpodobně by velká část frameworku zůstala nevyužita [12].

3.3 Komunikace

Běžné HTTPS spojení je vhodné pro většinu komunikace, jako například přenos textového obsahu, autentizace uživatelů. Dále může být výhodné tím, že lze na serveru nasadit cache pro odpovědi a tím snížit zatížení serveru, nebo gzip a tím snížit datový tok. HTTPS ale není výhodné pro navazování velkého množství spojení a pro přenos často měnících se dat. Proto je potřeba využít další technologie [13]. V úvahu přichází samotná implementace WebSocket, knihovna Socket.io a WebRtc.

WebRtc je určené převážné pro přenos zvuku a videa [14]. Nyní není využito a je uvedeno z důvodu, že v budoucnu je možné přidat funkcionalitu sledování kolejiště kamerou a přenos z kamer vláčků.

Socket.io není implementace WebSocket, ale může ho pro přenos dat využívat. Oproti samotné implementaci WebSocket přináší podporu jmenných prostorů - multiplexing, větší spolehlivost, automatické připojení po přerušení komunikace, nebo systém zaslání odpovědi acknowledgments. Podporuje i zabezpečené připojení přes TLS. Většina těchto vlastností bude potřeba, proto byl zvolen Socket.io [15].

3.4 Ostatní

Další z použitých knihoven je date-fns. JS nemá zrovna bohatou základnu funkcí, co se týče práce s datumy a časy. Date-fns přidává funkcionalitu převodu do českého tvaru a velké množství funkcí pro počítání s časem [16]. Zároveň pracuje se standardním objektem Date, proto zachovává kompatibilitu. Knihovna je použita v části serveru i klienta.

Pro rozšíření základní funkcionality je na straně klienta použita knihovna Lodash. Obsahuje funkce pro zjednodušení práce s JS [17]. V aplikaci jsou použity funkce remove - odebírání prvků z pole a clone - kopie objektů.

4

Vývojové nástroje

Pro kvalitní vývoj se v prostředí JS nelze obejít bez dodatečných nástrojů sloužících k podpoře vývoje nebo sestavení aplikace.

4.1 NPM

NPM je správce balíčků pro JS. Dělí se na tři části - webová stránka, nástroj příkazové řádky a registr.

Na webové stránce je možné procházet a objevovat jednotlivé balíčky. Pomocí příkazové řádky se instalují a aktualizují balíčky. Registr je databáze všech balíčků [2].

Základní konfigurační soubor se jmenuje package.json. Obsahuje vyžadované verze všech balíčků. Na základě tohoto souboru se generuje package-lock.json, který obsahuje konkrétní verze nainstalovaných balíčků a informace o všech závislostech mezi nimi. Soubory jsou umístěny ve složce node_modules.

4.2 TypeScript

TS je nástroj vyvinutý společností Microsoft pro přidání statické typové kontroly k JS. Jakýkoliv kód napsaný v TS je zároveň validní kód JS. Z toho vyplývá, že TS je rozšíření JS [18]. Hlavní výhody, které TS přináší jsou:

- Překlad do jakékoli verze ES, takže je možné používat nové verze ES dříve, než jsou dostupné v prohlížeči.
- Napovídání a automatické doplňování kódu ve vývojovém prostředí.
- Automatické odhalování chyb, které souvisejí s typy.
- Možnost sdílení typů a kódu mezi serverem a klientem.
- Vede programátora ke kvalitnějšímu kódu.

Faktem je, že vývoj v TS může být kvůli nutné definici typů zdlouhavější, než v samotném JS a napovídání např. ve vývojovém prostředí Visual Studio Code funguje i pro JS. Samozřejmě ne tak kvalitně. Proto je vhodnější spíše pro projekty s důrazem na kvalitu výsledného kódu. Pro kontrolu kvality kódu je vhodný nástroj ESLint.

4.3 ESLint

Slouží pro statickou analýzu kódu. Umí automaticky opravovat běžné problémy a překlepy. Zároveň je možné ho nastavit tak, aby automaticky vynucoval formátování kódu [19]. V tomto projektu je využit styl se jménem standard. Používají se ještě styly airbnb a google. Styl standard například nevyužívá středníky, koncové čárky, a definuje mezery mezi konstrukcemi kódu. Je nasazen na straně serveru i klienta.

4.4 SASS

Preprocesor pro CSS. Podporuje dva druhy syntaxe sass a scss. SASS považuji za vhodnější, protože využívá úspornější syntaxi.

SASS rozšiřuje CSS o proměnné, vnořování prvků, znovupoužitelné úseky kódu, funkce, matematické operátory a další.

```
$color: #ccc
body
  color: $color
```

Obrázek 4.1: Příklad kódu SASS

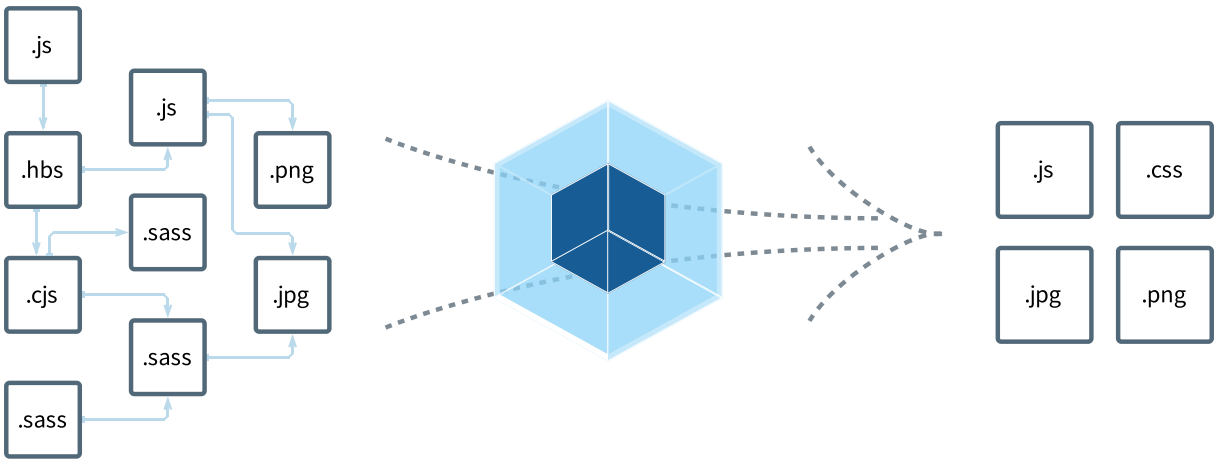
```
body {
  color: #ccc;
}
```

Obrázek 4.2: Příklad kódu CSS

Přes to, že existují překladače SASS fungující v prohlížeči, je vhodnější provést překlad ještě před odesláním kódu do prohlížeče [20]. Toto zajišťuje Webpack.

4.5 Webpack

Hlavní účel nástroje je sloučit kód ze všech použitých modulů a minimalizovat ho. Podporuje automatické spouštění překladu SASS do CSS, TS do JS, nebo minimalizaci obrázků. Místo vkládání scriptů do stránky pomocí HTML tagu script, se využívá ECMAScript modulů. Po překladu vytvoří graf závislostí a sloučí všechny importované moduly. Výsledkem může být .js, .css soubor a složka s obrázky. Ve všech případech není vhodné vytvořit jeden velký soubor a to primárně z důvodu rychlosti načítání stránky. Proto je ve client aplikaci využita funkce „chunks“, která rozděluje stránky na samostatné soubory a ty jsou načítány až v případě potřeby [21].



Obrázek 4.3: Princip slučování kódu nástrojem webpack. Převzato z [21].

Díky Webpack DevServeru a funkci „hot reload“ je možné téměř ihned po uložení napsaného kódu vidět výsledek v prohlížeče na druhé obrazovce, což je velmi příjemné, oproti neustálému obnovování stránky po změně kódu.

Na straně klienta je použití webpacku téměř nezbytné. Tím, že je veškerý kód minimalizován a sloučen, se výrazně snižuje doba načítání stránky.

Na server stačí přenášet pouze sloučené soubory bez všech závislostí ve složce node_modules.

4.6 Vue CLI

Vue CLI je nástrojová knihovna složená nejen z výše zmíněných nástrojů. Nástroje jsou univerzálně nakonfigurované pro potřeby projektu Vue.js. Umožňuje vytvořit rychle strukturu projektu díky průvodci, kde je možné zvolit si požadované funkce.

Skládá se z více NPM balíčků. @vue/cli je potřeba nainstalovat globálně v rámci běhového prostředí. @vue/cli-service je potřeba nainstalovat lokálně k projektu [22]. Jako běhové prostředí lze výhodně využít Docker.

4.7 Docker

Řadí se do mírně odlišné kategorie nástrojů. Jedná se o zapouzdřené prostředí v rámci operačního systému. Zapouzdřená prostředí se nazývají kontejner. Každý kontejner je založen na šabloně, která se nazývá obraz [23].

Pro popis obrazu se používá tzv. Dockerfile, což je soubor, který obvykle vychází z některého již vytvořeného obrazu a obsahuje instalační instrukce. Obrazy, ze kterých je možné dědit, jsou umístěné na docker hubu[1].

[1] https://hub.docker.com

```
1 FROM node:16-buster
2
3 WORKDIR /usr/src/app
4 RUN npm i -g nodemon
5 USER node
6 CMD npm run serve
7 EXPOSE 8080
```

Obrázek 4.4: Zkrácený Dockerfile pro vývoj Node.js aplikace

Kontejnery mohou být sdružovány do služeb, které mezi sebou komunikují. Tyto služby jsou popsány souborem docker-compose.yml a spouštěny příkazem docker-compose up.

Výhoda takového přístupu je to, že pokud je potřeba pracovat na více projektech na jednom PC, tak není nutné mít nainstalované všechny verze potřebných nástrojů na samotném počítači. Dále je pro každého nového vývojáře výrazně jednodušší spustit projekt založený na technologii Docker, díky kterému nemusí řešit nekompatibilitu verzí software nainstalovaného na jeho PC a software vyžadovaný aplikací.

Docker se běžně používá i pro produkční nasazení, toho ale v tomto případě není využito.

5

Realizace webové aplikace

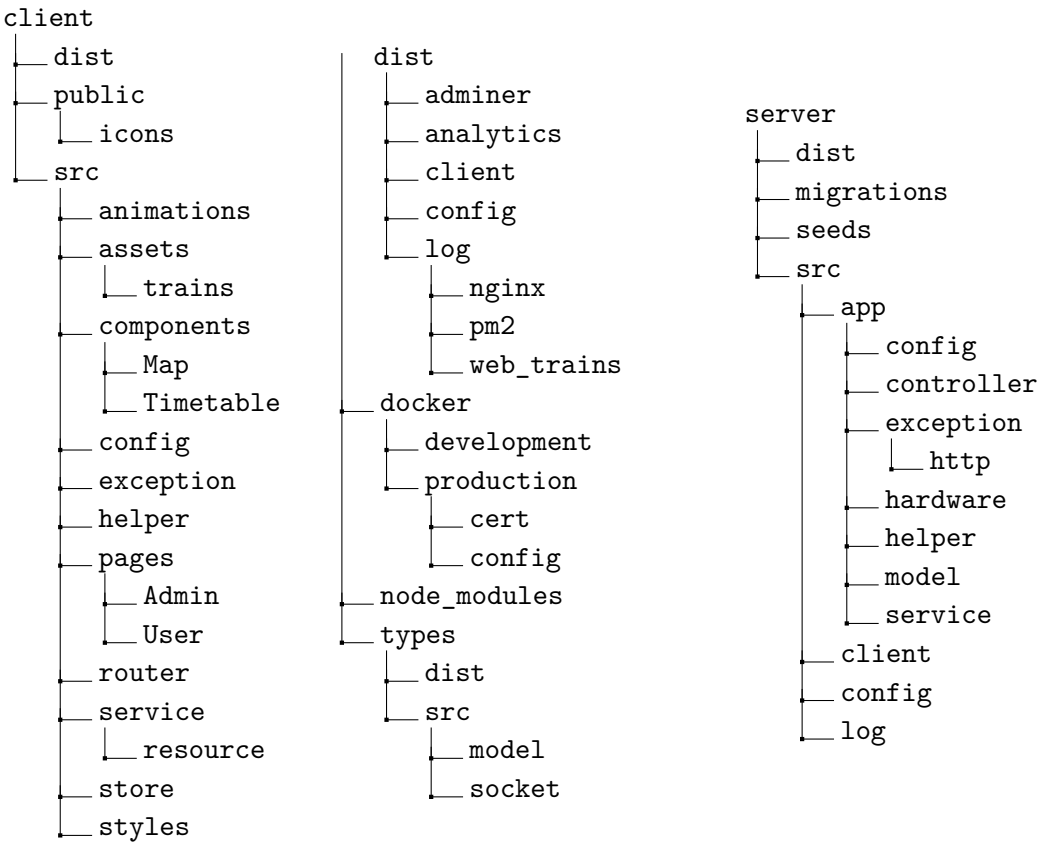
5.1 Struktura aplikace

Projekt se skládá ze server aplikace ve složce *server*, klient aplikace ve složce *client* a knihovny sdílených typů ve složce *types*.

Každá část obsahuje svojí složku *dist*. Při spuštění sestavení celé aplikace se z těchto částí překopíruje do složky *dist* v kořenu. Ve složce *docker* se nachází konfigurační soubory dockeru.

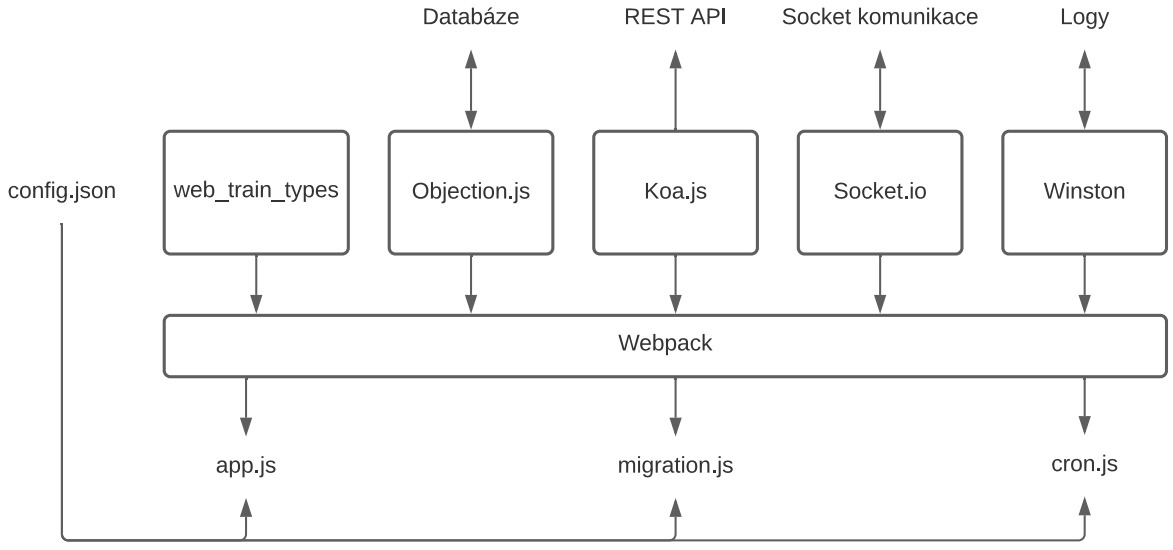
Složka *docker* je dále dělena na *development* a *production*. Obě obsahují soubor *docker-compose.example.yml*, který je doporučenou konfigurací proto, že jsou běžné úpravy pro potřeby konkrétního vývojáře, je nutné, aby byly přejmenovány a zkopírovány, tak jak je popsáno v příloze A. Nejdůležitější soubory jsou Dockerfile s konfigurací obrazů. Složka production obsahuje ještě konfiguraci web serveru Nginx.

Části server, klient a typy sdílejí některé balíčky, proto je využito funkcionality systému NPM „workspaces“, díky tomu jednotlivé části mohou instalovat externí závislosti do stejné složky *node_modules*.



Obrázek 5.1: Adresářová struktura projektu

5.2 Server



Obrázek 5.2: Diagram knihoven server

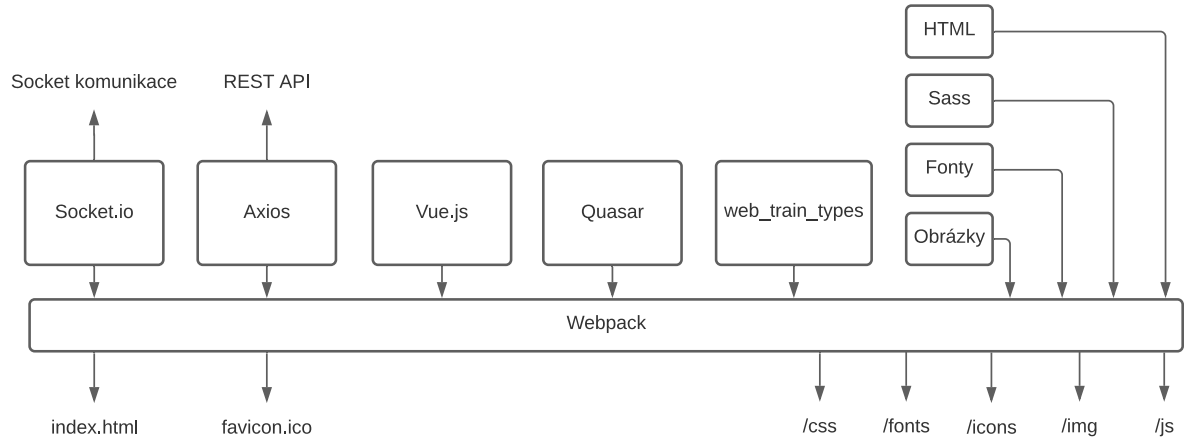
Server obsahuje složky *dist*, *migrations*, *seeds*, *src* a konfigurační soubory knihoven a nástrojů. Složka *dist* obsahuje sestavenou část aplikace. Obsah složek *migrations* a *seeds* je popsán v kapitole 5.7.1. Složka *src* obsahuje TS kódy. Dělí se na složky *app*, *client*, *config*, *log*

a obsahuje hlavní soubory serveru - *cron.js*, *app.js* a *migration.js*.

Migration.js slouží ke spouštění databázových migrací popsaných v kapitole 5.7.1. App.js je hlavní webová aplikace. Cron.js komunikuje s řídící aplikací kolejiště.

Složka *client* obsahuje soubor s varováním, že klient aplikace běží na portu 8080. Varování se zobrazí v případě, že je aplikace ve vývojovém režimu. Složka *config* obsahuje příklady konfiguračních souborů pro aplikaci a pro správce Node.js aplikací PM2. Nástroj PM2 je popsán v kapitole 5.8. Složka *log* obsahuje všechny chybové zprávy ze serveru a klienta. Struktura složky *app* je popsána v kapitole 5.7.

5.3 Klient



Obrázek 5.3: Diagram knihoven klient

Klient obsahuje složky *dist*, *public* a *src*. Složka *public* obsahuje statické soubory jako například ikonky a základní HTML dokument.

Obsah složky src je popsán v kapitole 5.6.

5.4 Uživatelské role

Správa všech uživatelů a jejich možností je implementována v souboru *types/src/RolesManager.ts*. Současné pojetí je univerzální a nyní není využito všech jeho možností.

Uživatelé mohou nabývat tří rolí: návštěvník (visitor), uživatel (user), editor (editor) a administrátor (admin). Každý uživatel může mít přidělenou akci (action) na nějakou funkcionalitu (resource). Akce jsou: vidět (see) a upravovat (crud). Funkcionality jsou: administrace uživatelů (userAdmin), administrace jízdních řádů (timetable), úprava CMS (cms), tabulky s chybami (error), administrace HW (hardwareAdmin).

Uživatel má nyní nastaveno právo na administraci jízdních řádu, tím i přímé řízení provozu. Editor navíc může upravovat CMS stránku a upravovat nastavení HW kolejiště. Administrátor

navíc může vidět tabulky s chybami a mazat uživatele, nebo jim měnit práva.

V aplikaci se může registrovat kdokoli. Novému uživateli je přidělena role návštěvník, který má stejná práva jako nepřihlášený uživatel. Administrátor následně může roli libovolně povýšit. Tímto způsobem je zajištěno, že si každý nový uživatel může sám zvolit jméno a heslo.

Definice jednotlivých rolí je zapsána v konstruktoru třídy RolesManager v poli *roles* a například nastavení pro editora vypadá takto:

```
this.roles = {
    [Role.editor]: {
      [Resource.timetable]: [Action.see, Action.crud],
      [Resource.cms]: [Action.see, Action.crud],
      [Resource.hardwareAdmin]: [Action.see, Action.crud]
    }
}
```

Obrázek 5.4: Příklad nastavení práv pro uživatelskou roli editor

5.5 Popis API, sdílené typy

Pro komunikaci mezi klientem je využito dvou způsobů REST API a Socket.io.

Každá API adresa je popsaná typy, které jsou sdílené mezi klientem a serverm v *types/src/model*. V tabulce je zobrazeno, jaké HTTP metody lze na adrese zpracovat a formát žádosti. Formát adresy obsahuje povinné parametry za „:“.

Metoda GET s parametrem vrací jeden záznam. Bez parametru vrací pole všech záznamů. Metoda POST je určena pro přidání nové entity. Metody PUT a PATCH jsou určeny pro úpravu záznamu, liší se tím, že PATCH může měnit pouze část záznamu. PUT mění vždy celý záznam. DEL slouží k odstranění záznamu [24].

Adresa *cms* se váže na úpravu stránky o kolejišti. *Delay* k nastavení zpoždění dojezdu vlaků. *Error* k výčtu chyb v administraci a odesílání chyb z klienta na server. *Journey* k zobrazení a editaci jízd. *Planner* k plánování. *Station* a *train* obsahují pouze získání všech záznamů z databáze. *Timetable* k jízdním řádům. *Register* k registraci uživatele. *Signin* k přihlášení uživatele, navrací JWT. *Me* pro zobrazení a správu o právě přihlášeném účtě. *User* k administraci uživatelů. *Test* na jakýkoliv požadavek odpoví „API works“.

Adresa	Metoda	Žádost
/cms/:id	GET	
/cms/:id	PATCH	`text`
/delay	GET	
/delay	PUT	`trainId, stationId, delay`
/error:type	GET	
/error:type:count:offset	GET	
/error	POST	`data`
/journey:id	GET	
/journey	GET	
/journey	POST	`stationFromId, stationToId, trainId, direction, section`
/journey:id	PATCH	`stationFromId, stationToId, trainId, direction, section`
/journey:id	DEL	
/planner:id	GET	
/planner	GET	
/planner/range/from/to	GET	
/planner	POST	`time, day, month, timetableId`
/planner	POST	`once, timetableId`
/planner:id	PATCH	`time, day, month, once, timetableId`
/planner:id	DEL	
/station	GET	
/timetable:id	GET	
/timetable	GET	
/timetable	POST	`name, journey`
/timetable:id	PATCH	`name, journey`
/timetable:id	DEL	
/train	GET	
/register	POST	`name, password`
/signin	POST	`name, password`
/me	GET	
/me	PATCH	`name`
/me	DEL	
/user	GET	
/user:id	PATCH	`name, role`
/user:id	DEL	
/test	vše	

Tabulka 5.1: Přehled adres REST API

Komunikace přes Socket.io probíhá ve třech kanálech - adminHardware, adminMap

a publicMap.

Nastavení hardware v administraci je připojeno na jmenný prostor adminHardware. Spouštění jízdních řádů a jízd bez ohledu na plán je připojeno na jmenný prostor adminMap. Oba jmenné prostory jsou dostupné pouze pro uživatele s příslušnými právy. PublicMap slouží pro posílání pozic vlaků všem připojeným uživatelům.

V souboru *types/src/socket/Packets.ts* jsou definovány typy pro přenos informací. Struktura je podobná jako v řídící aplikaci, která je popsaná v kapitole 2.

5.6 Klient

5.6.1 Service

Service jsou třídy sloužící pro komunikaci s aplikací server.

5.6.1.1 API

Základem komunikace je třída ApiService. Zajišťuje vytvoření instance knihovny Axios[1] a nastavení odchytávání případných chyb - odpovědi s jiným kódem než 200. Axios je knihovna pro odesílání HTTP požadavků. Dále třída definuje funkce *get*, *post*, *patch*, *put*, *del* a funkce pro nastavení a mazání autentizačního JWT v instanci Axios. Pokud má Axios JWT nastaven, tak ho odesílá při každém požadavku.

Všechny třídy, které komunikují přes REST API, mají jako závislost v konstruktoru ApiService. Třída UserService navíc vyžaduje instanci třídy StorageService, tato třída zajišťuje přístup k localStorage v prohlížeči pro ukládání a mazání JWT, když se uživatel přihlásí nebo odhlásí.

5.6.1.2 Socket

Vytvoření instance Socket.io není tak jednoduché, jako v případě Axios. Socket.io totiž po inicializaci vytvoří kanál, přes který probíhá následující komunikace. Ovšem není žádoucí vytvořit všechny komunikační kanály hned po načtení stránky. Na stránku může vstoupit nepřihlášený uživatel, který nemá práva odesílat všechny požadavky. V případě, že se přihlásí je nutné reinicializovat všechny Socket.io připojení.

Základní třída se jmenuje SocketService. Ta vytváří instanci třídy Manager z knihovny Socket.io a nastavuje zachytávání chyb připojení, nebo chyb přicházejících z aplikace server.

Všechny třídy, které používají komunikaci Socket.io, musí dědit od BaseSocketService, která redefinuje funkce *on*, *emit* a definuje funkci *connect*. Funkce *connect* se volá v případě přístupu na stránku, kde je využívána komunikace Socket.io.

[1] https://github.com/axios/axios

5.6.2 Router

Pro tvorbu různých stránek v SPA je nutné využít knihovny Vue Router[2]. Jeho nastavení je definováno v souboru *client/src/router/router.ts*. Pro každou stránku je potřeba nastavit cestu - český název zobrazený v prohlížeči, jméno - na které se odkazuje v kódu a vue komponentu.

Pro kontrolu oprávnění přístupu uživatelů je využito funkcionality „beforeEach“. Před každým přístupem na stránku se zkontroluje, jestli má současný uživatel oprávnění na ni přistupovat a v případě, že nemá, je přesměrován na přihlašovací stránku. Nastavení oprávnění probíhá podobně jako v kapitole 5.4.

5.6.3 Store

Vue.js je založeno na uzavřených komponentách, které můžou měnit pouze svůj stav a neměli by být závislé na stavu jiných komponent. Tento princip je nutné v některých případech porušit. Hlavní důvod je obvykle uložení stavu uživatele - jeho jména, JWT, atd. Doporučená knihovna pro tento účel se jmenuje Vuex, ale její použití s TS je velmi nekomfortní. Proto byla dohledána knihovna Pinia, která je vyvinuta jedním z autorů Vuex. Je jednodušší na použití a lépe spolupracuje s Vue.js verze 3 [25].

V této aplikaci je Pinia použita pro změny stavu uživatele a pro uchování seznamu vlaků na kolejišti. Protože tento seznam je použit v komponentě Map a TrainDrawer, bylo by zbytečné ho načítat vícekrát.

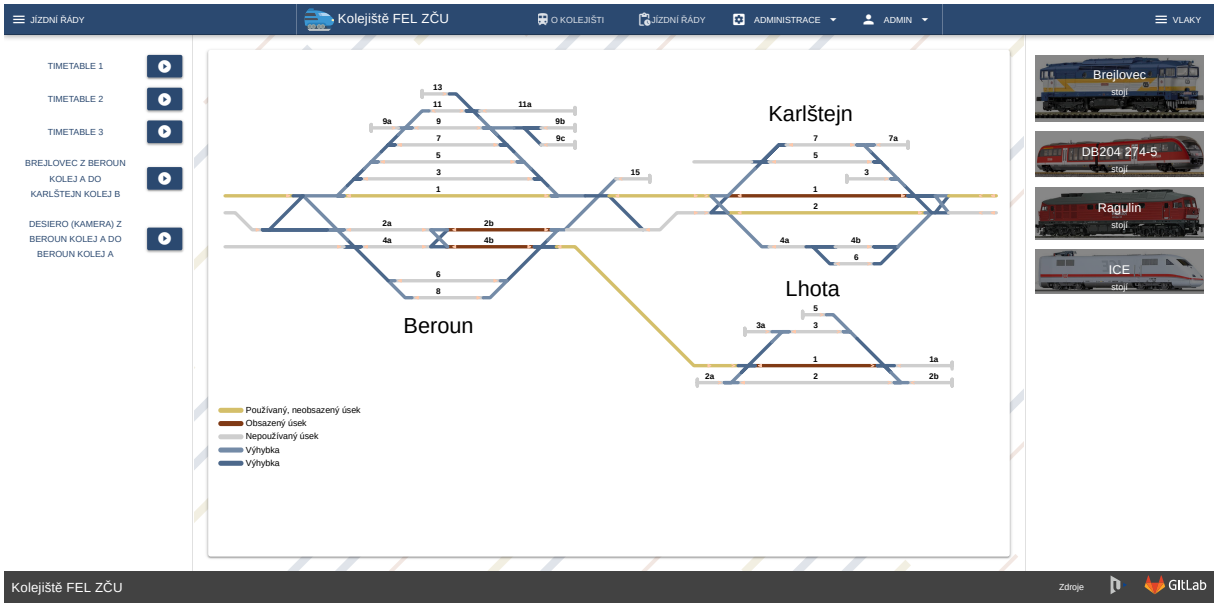
5.6.4 Výjimky

Zachytávání chyb aplikace - chyb v kódu je zasíláno na server, kde jsou uloženy pro případné budoucí zpracování. K tomu je využita knihovna log4javascript[3].

[2] https://router.vuejs.org
[3] http://log4javascript.org

5.6.5 Stránky

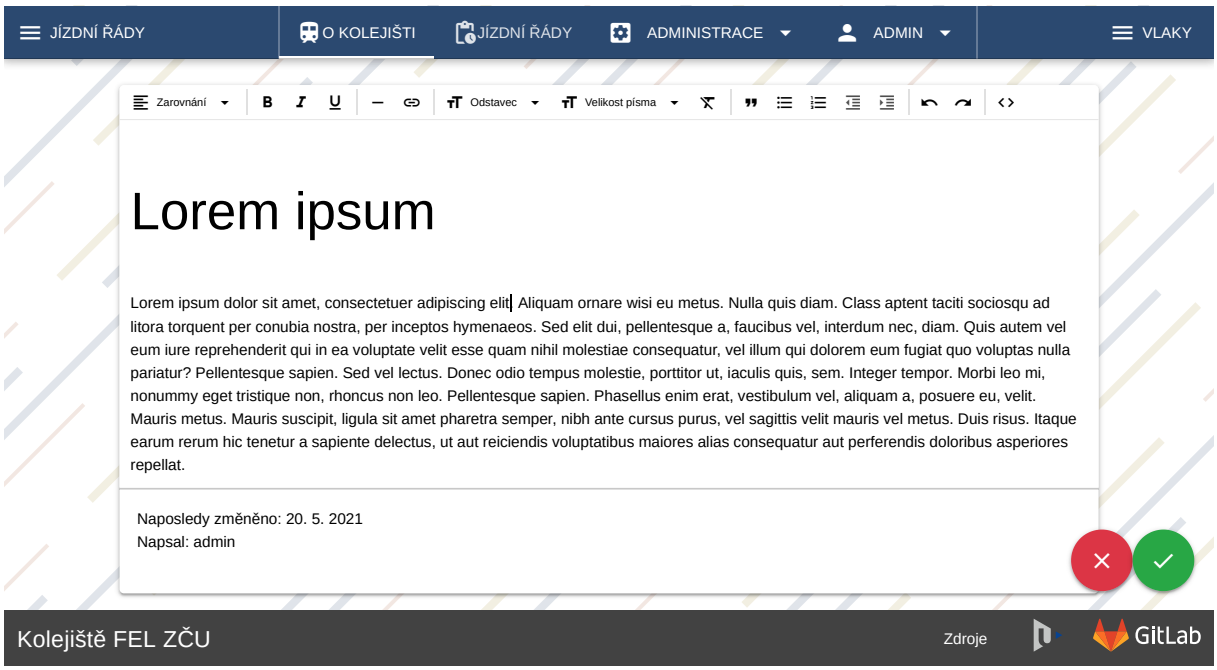
5.6.5.1 Úvodní stránka



Obrázek 5.5: Úvodní stránka.

Při zobrazení úvodní stránky se automaticky vysune pravé menu popsané v kapitole 5.6.6.9. Dále se skládá z komponenty „mapa“ popsané v kapitole 5.6.6.2.

5.6.5.2 O kolejišti

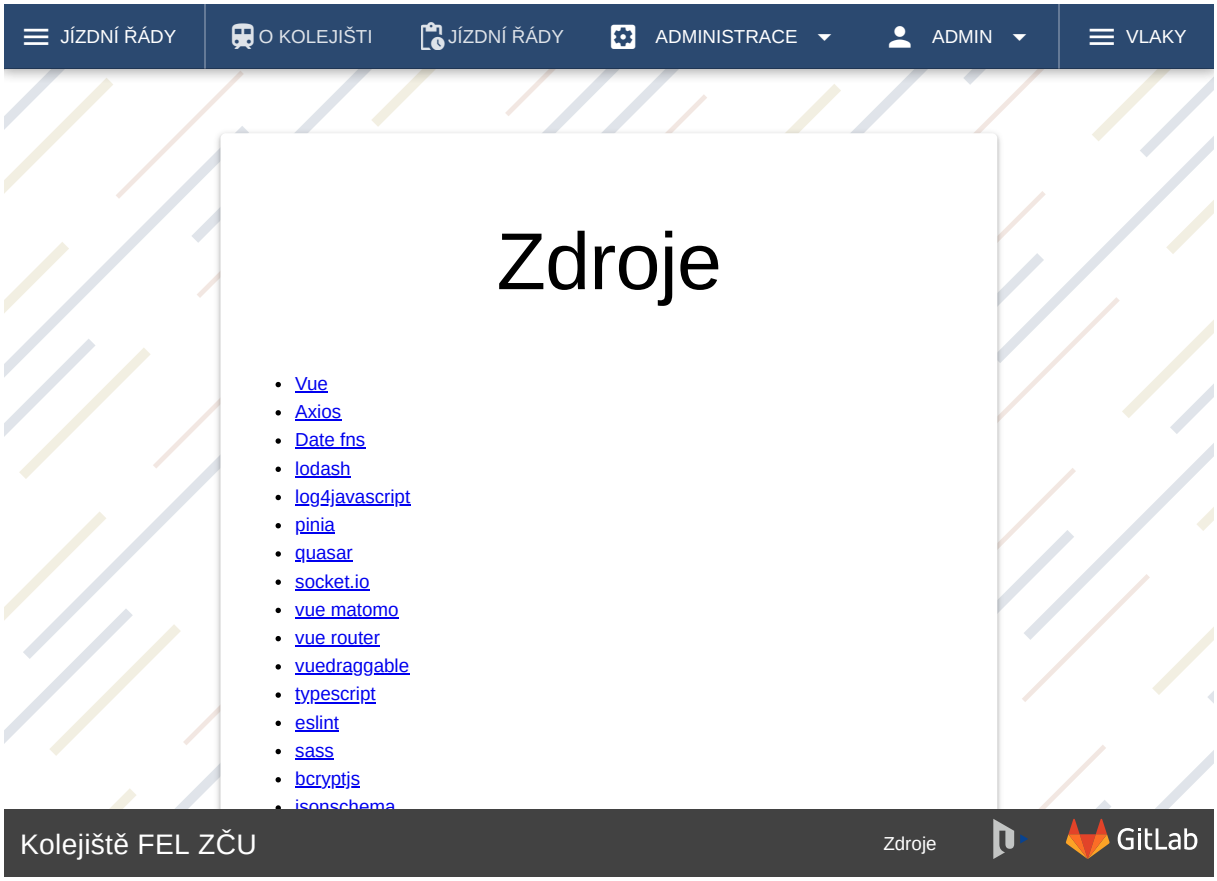


Obrázek 5.6: Strana o kolejišti.

Stránka CMS obsahuje text o kolejišti a informaci o poslední změně. V případě, že je přihlášen uživatel, který má právo stránku editovat, je v pravém dolním rohu zobrazena ikona tužky.

Po kliknutí na ikonku se stránka přepne do režimu úprav. Ten je viditelný na obrázku 5.6. Editor je komponenta z knihovny Quasar. Umožňuje základní editaci textu.

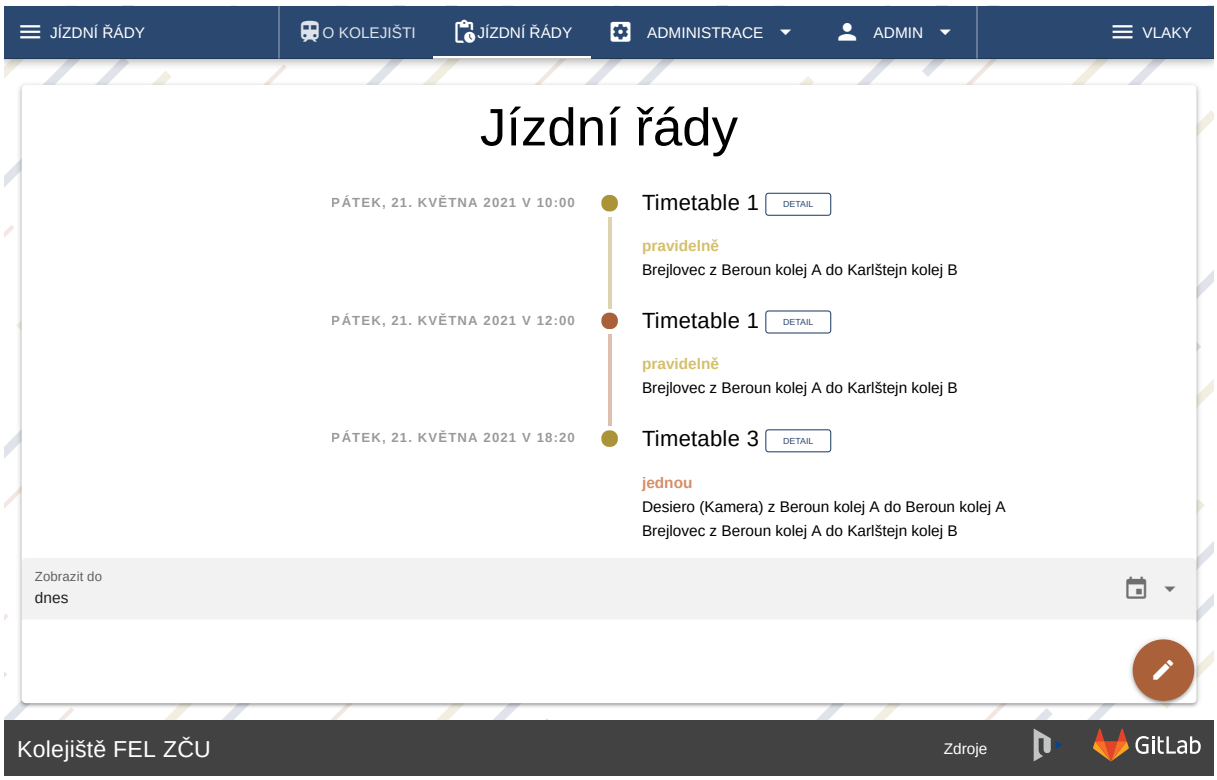
5.6.5.3 Zdroje



Obrázek 5.7: Strana se zdroji.

Na stránce zdroje jsou zobrazeny odkazy na knihovny využité při vývoji aplikace.

5.6.5.4 Jízdní řády



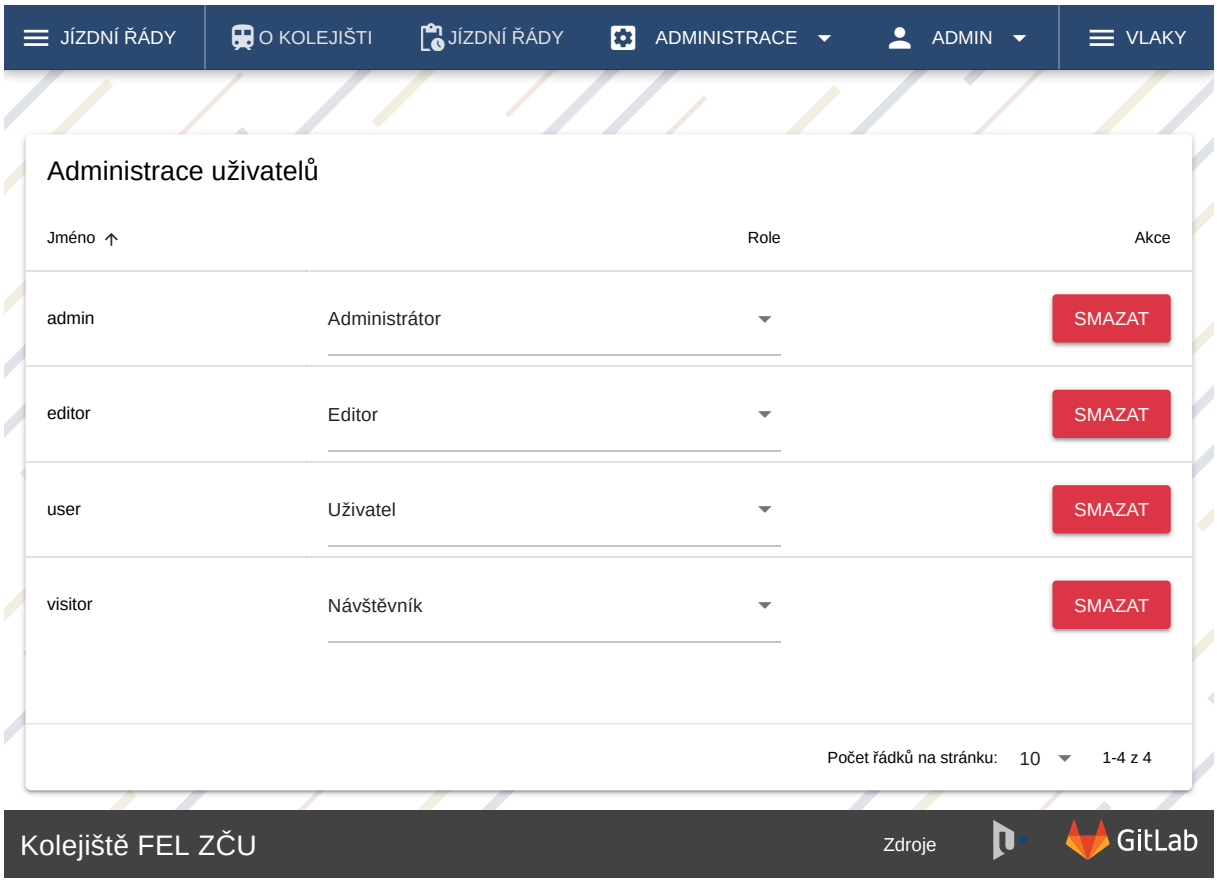
Obrázek 5.8: Strana s následujícími jízdami.

Na stránce jízdní řády jsou zobrazeny všechny následující jízdy v zadaném časovém intervalu. Je možné vybrat dnes, zítra, týden, měsíc, nebo libovolný interval kliknutím na ikonku kalendáře.

V případě, že je přihlášen uživatel s právem editovat jízdní řády, je v pravém spodním rohu zobrazen odkaz na administraci s ikonou tužky.

V seznamu jízd je viditelný datum odjezdu, jméno jízdního řádu a seznam všech jízd ve formátu: vlak z A do B. Při kliknutí na detail se zobrazí příslušný plán (komponenta je popsaná v kapitole 5.6.6.6). Plán z tohoto místa nelze upravovat.

5.6.5.5 Administrace - uživatelé



Obrázek 5.9: Strana s nastavením uživatelů.

V administraci uživatelů je možné jednotlivé uživatele mazat a měnit jejich uživatelské role. Po kliknutí na jméno je možné provést jeho úpravu. Všechny změny jsou provedeny okamžitě a jsou ohlášeny notifikací.

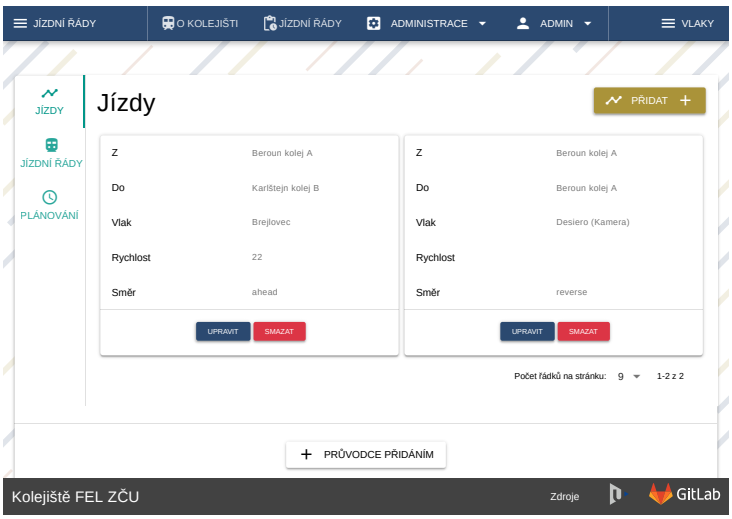
5.6.5.6 Administrace - jízdní řády

Administrace jízdních řádů se dělí na tři části - jízdy, jízdní řády a plánování.

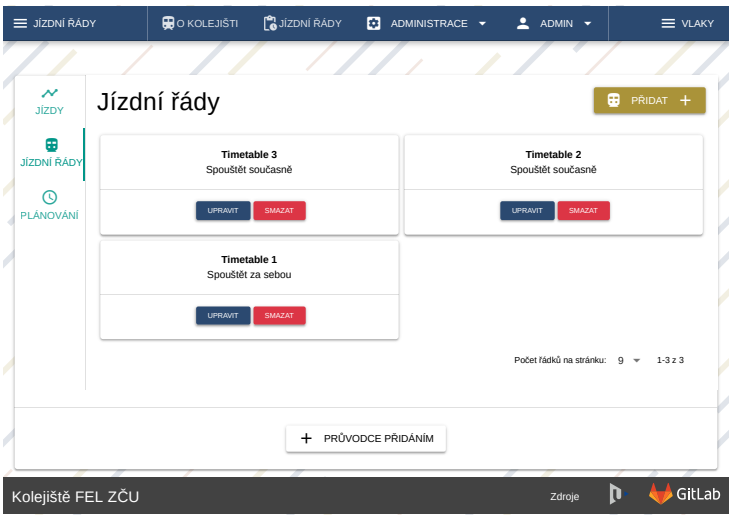
Administrace seznamu jízd je na obrázku 5.10. Jízda je z hlediska aplikace základní celek, který se váže k jednomu vlaku a nastavuje jeho trasu v libovolném počtu bodů - od, přes, do. Úprava jízdy je popsána v kapitole 5.6.6.4.

Administrace seznamu jízdních řádu je na obrázku 5.11. Jízdní řád je seznam více jízd. Zvolené jízdy lze spouštět buď současně, nebo za sebou. Úprava jízdního řádu je popsána v kapitole 5.6.6.5.

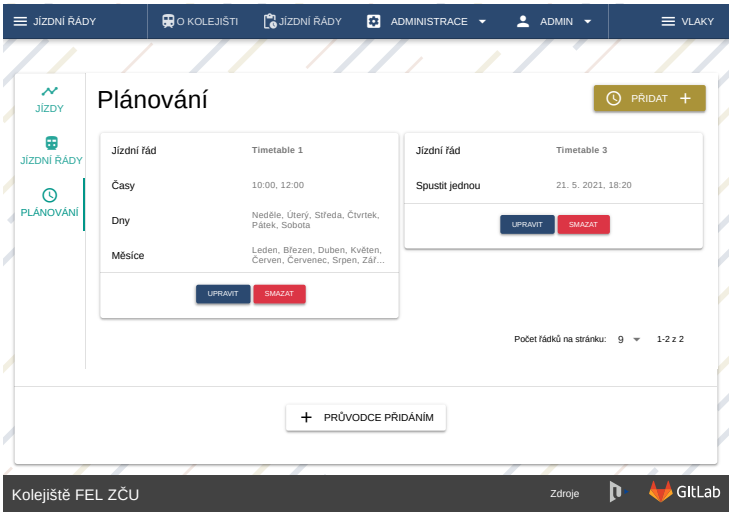
Administrace seznamu plánů je na obrázku 5.12. Plán se dělí na dva druhy - pravidelný a jednorázový. Pravidelný znamená, že se nastaví v jaké dny, měsíce a časy se má periodicky spouštět. Jednorázový znamená, že se nastaví jeden datum a čas, kdy bude spuštěn. Úprava plánu je popsána v kapitole 5.6.6.6.



Obrázek 5.10: Strana s nastavením jízd.

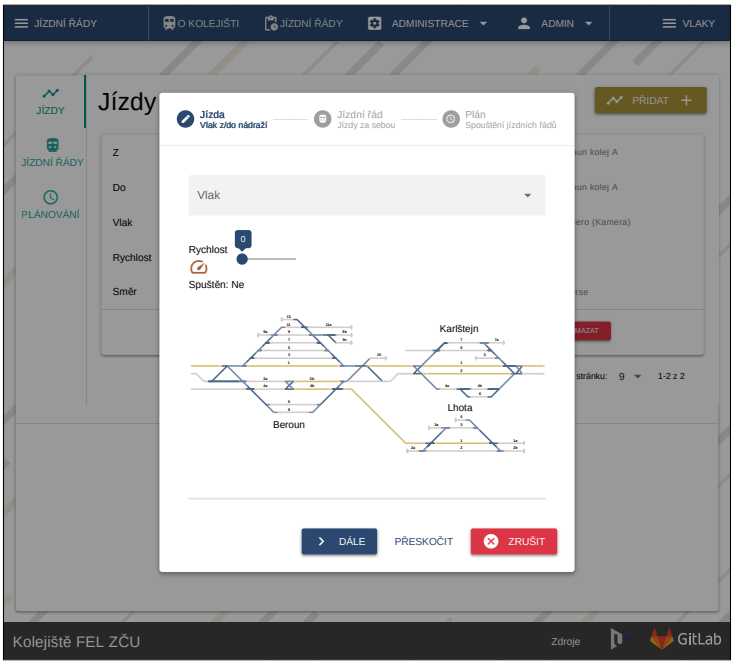


Obrázek 5.11: Strana s nastavením jízdních řádů.

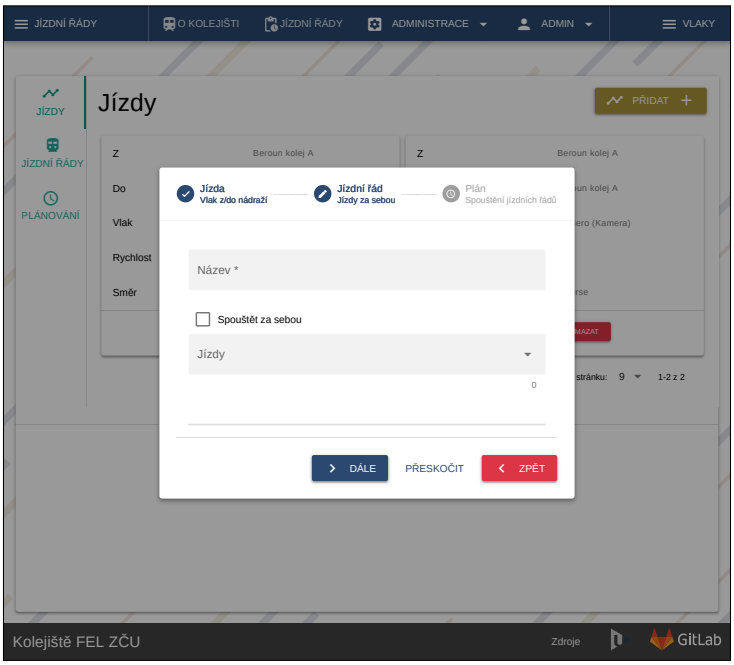


Obrázek 5.12: Strana s nastavením plánování.

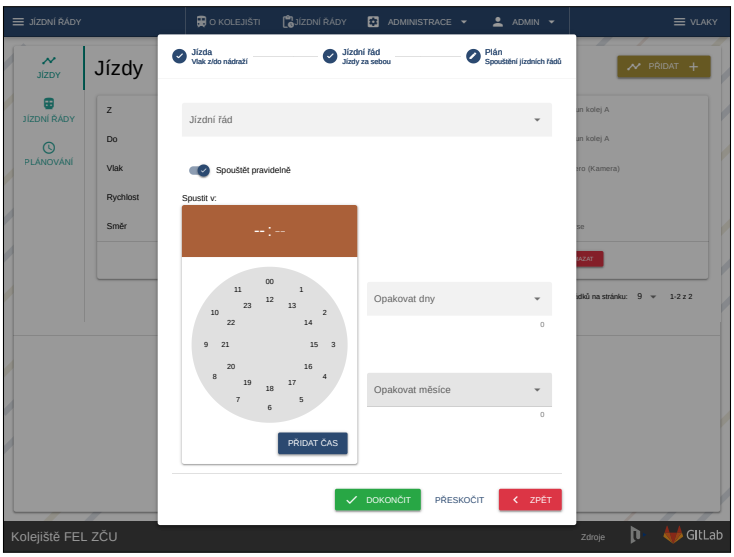
Administrace je obohacena o průvodce přidáním všech celků. Tento proces se skládá z přidání jízdy na obrázku 5.13, jízdního řádu obrázku 5.14 a plánu obrázku 5.15. Kliknutím na tlačítko „další“ se změny uloží do databáze. Pokud se uživatel vrátí o krok v průvodci zpět jsou znovu načtena uložená data. Z toho vyplývá, že průvodcem je možné přidat pouze jednu jízdu/jízdní řád/plán.



Obrázek 5.13: Průvodce přidáním - jízda.

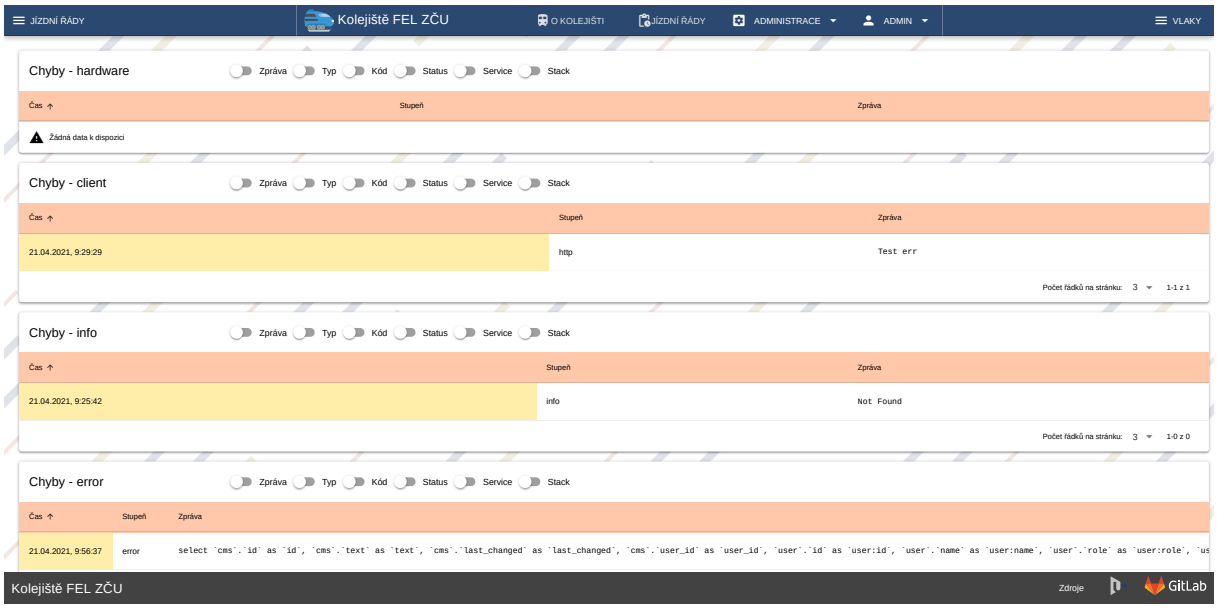


Obrázek 5.14: Průvodce přidáním - jízdní řád.



Obrázek 5.15: Průvodce přidáním - plán.

5.6.5.7 Administrace - chyby



Obrázek 5.16: Strana s zobrazením chyb.

V administraci je možné zobrazit všechny chyby vzniklé v aplikaci. Dělí se na chyby hardware, client, info a error. Samotná tabulka je komponenta popsaná v kapitole 5.6.6.8.

Chyby hardware se týkají připojení k řídící aplikaci a v budoucnu například zjištěných kolizí nižšími vrstvami.

Chyby client jsou chyby vzniklé v prohlížeči, zaslané a uložené na serveru.

Chyby info jsou informační hlášení, které nutně neznamenají chybný stav.

Chyby error jsou fatální selhání vzniklé například selháním databáze, nebo chybou v kódu ze strany aplikace server.

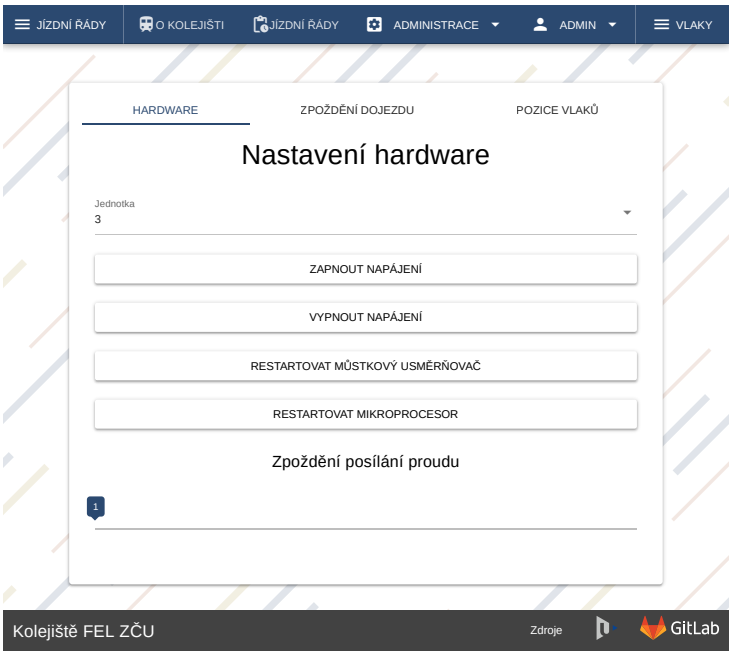
5.6.5.8 Administrace - hardware

Administrace hardware se dělí na tři části. Nastavení řídících jednotek na obrázku 5.17, nastavení zpoždění dojezdu vlaků na obrázku 5.18 a nastavení současné pozice vlaků na obrázku 5.19.

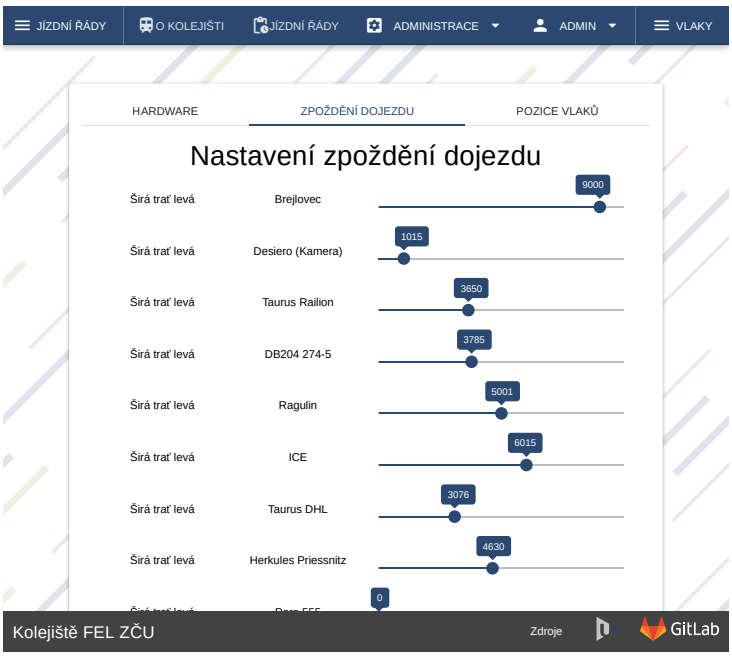
V nastavení řídících jednotek lze zvolit řídící jednotku - v současné době pouze 3. a 4. Zapnout, vypnout napájení, restartovat můstkový usměrňovač, nebo mikroprocesor. Dále lze nastavit zpoždění odesílání změřených proudů.

V nastavení zpoždění dojezdu jsou vypsány všechny kombinace úseků kolejiště a vlaků. Dojezd do požadovaného úseku je registrován na jeho začátku, proto je potřeba, aby vlak ještě nějakou dobu pokračoval. Toto nastavení se může měnit například z důvodu osazení jiného množství vagonů.

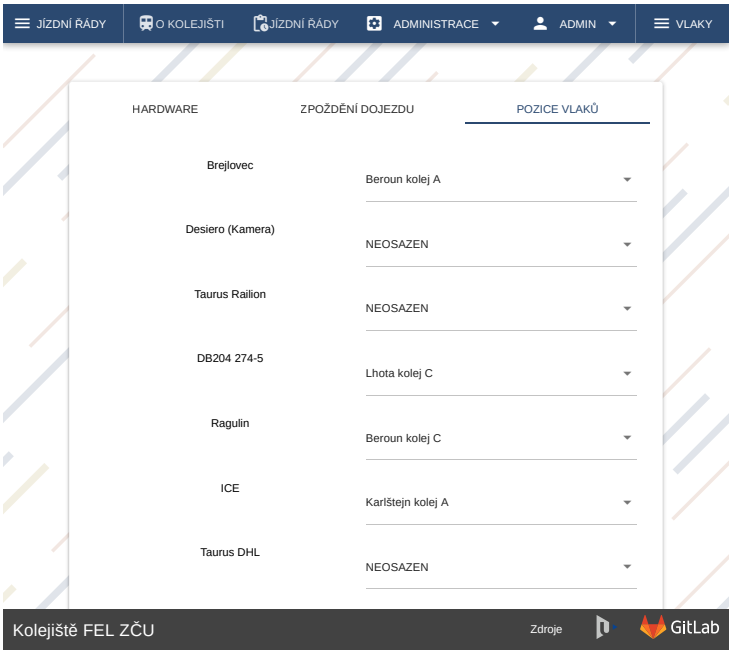
V nastavení pozice vlaků lze měnit současné pozice vlaků, případně nastavit, že je vlak neosazen.



Obrázek 5.17: Strana s nastavením hardware.



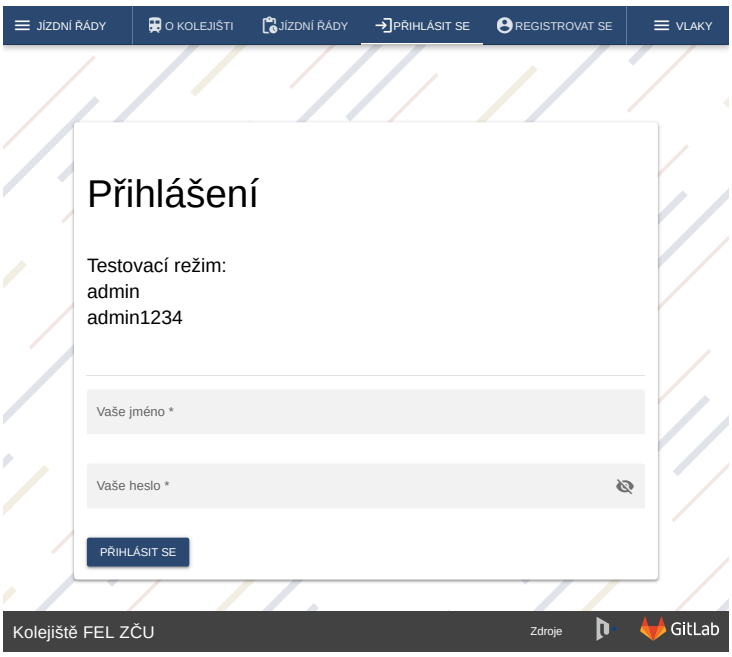
Obrázek 5.18: Strana s nastavením zpoždění dojezdu vlaků.



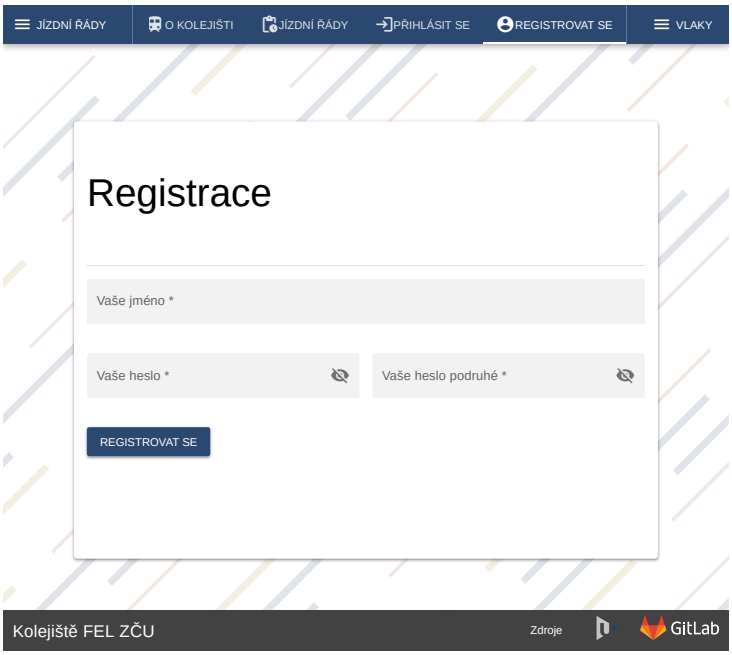
Obrázek 5.19: Strana s nastavením pozice vlaků.

5.6.5.9 Uživatel - přihlášení, registrace

Stránky přihlášení jsou zcela běžné. Vyžadují zadání jména a hesla. V případě registrace je nutné zadat heslo dvakrát. Po registraci uživatel není přihlášen a přesměrován na stranu přihlášení, aby si mohl vyzkoušet své nově zvolené přihlašovací údaje. Heslo je možné zobrazit, nebo skrýt kliknutím na ikonu přeškrtnutého oka.



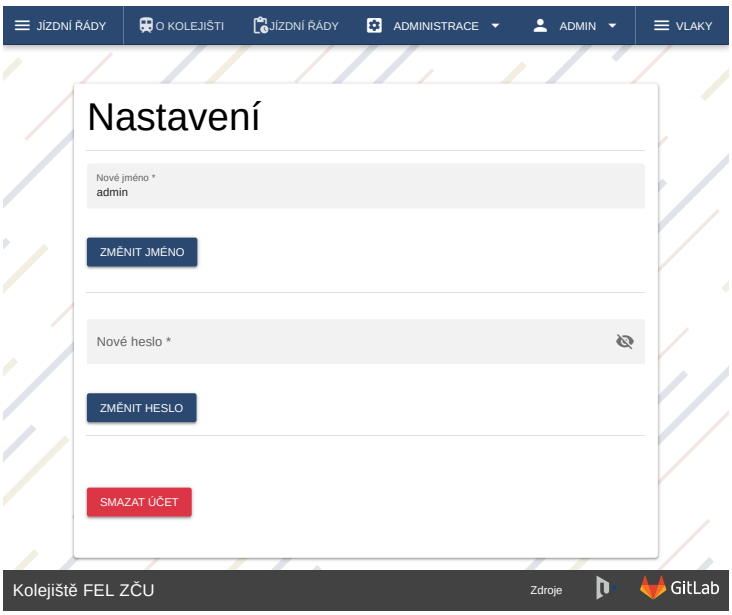
Obrázek 5.20: Strana s přihlášením uživatele.



Obrázek 5.21: Strana s registrací uživatele.

5.6.5.10 Uživatel - nastavení

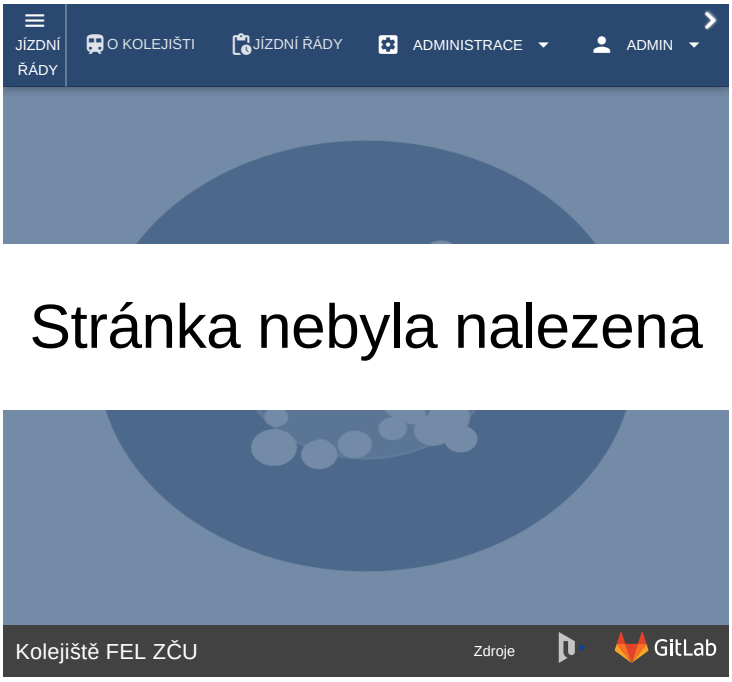
Stránka nastavení uživatele je dostupná všem přihlášeným uživatelům. Je na ní možné měnit své uživatelské jméno, heslo a případně smazat účet.



Obrázek 5.22: Strana s nastavením uživatele.

5.6.5.11 Chybová stránka 404

Chybová stránka se zobrazí v případě, že uživatel zadá do prohlížeče adresu, kterou aplikace nezná. Pozadí stránky 404 je inspirován kódem[4], který byl přepsán do TS.



Obrázek 5.23: Strana s chybou 404.

[4]https://codepen.io/juliangarnier/pen/gmOwJX

5.6.6 Komponenty

5.6.6.1 Mapa - základ

Z komponenty Map vychází dvě komponenty - na úvodní stránce a při výběru trasy. Obsahuje SVG s diagramem kolejiště. Úseky kolejiště jsou barevně rozlišeny na používaný, ale neobsazený úsek, obsazený úsek, nepoužívaný úsek a výhybka. Komponenta obarvuje nepoužívané úseky. Používané úseky mají k související cestě ve struktuře SVG přiřazeno id koleje z databáze.

5.6.6.2 Mapa - úvodní stránka

Na úvodní stránce je zobrazen diagram s aktuálním obsazením kolejí. Komponenta přijímá každých pět sekund informaci o pozici osazených vlaků. Formát této zprávy je pole naplněné objekty s vlastností t - id vlaku a p - id sekce.

Po najetí kurzorem nad úsek je zobrazen informace o obsazení. Pokud je úsek obsazen vlakem, tak po kliknutí je zobrazen detail ve formě komponenty popsané v kapitole 5.6.6.11.

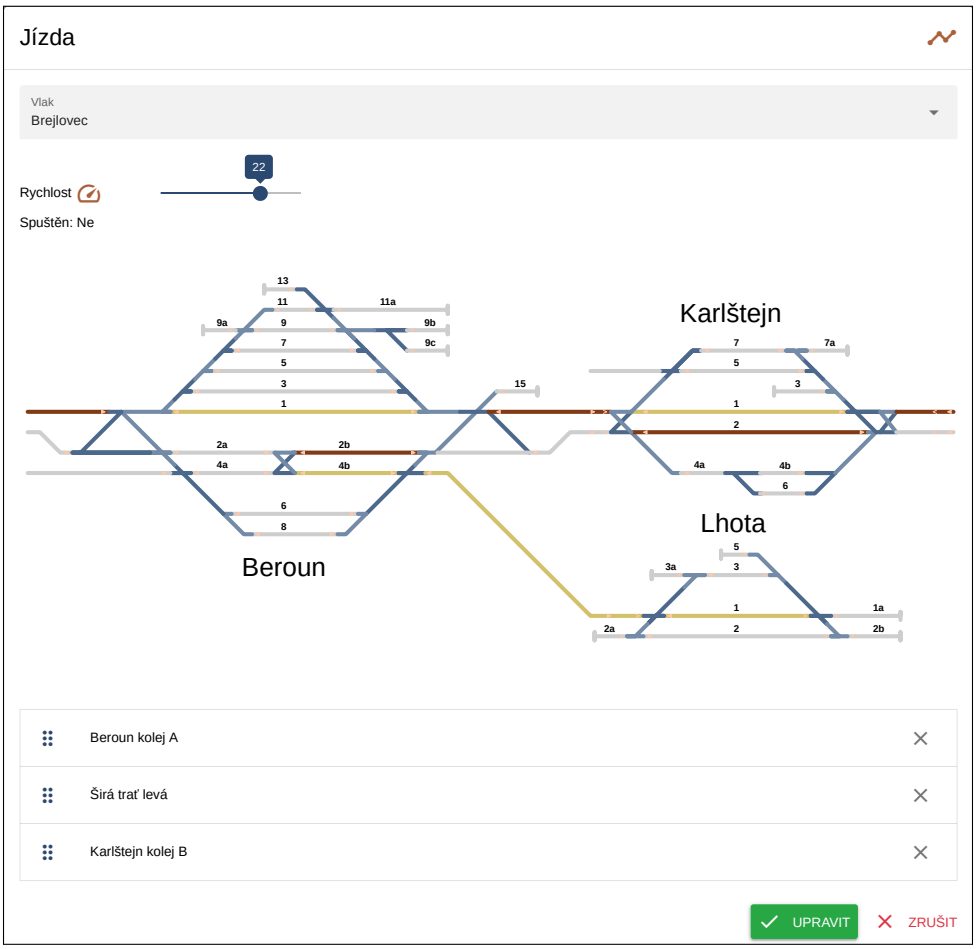
Vzhledem k tomu, že seznam vlaků je držen v globálním úložišti, které je popsané v kapitole 5.6.3, je potřeba po přidání vlaku v administraci obnovit stránku s touto komponentou.

5.6.6.3 Mapa - výběr trasy

Tato komponenta je určena pro výběr trasy vlaku. Hlídá, aby bylo možné vybrat sekci pouze jednou a vybrané sekce umisťuje do seznamu ve kterém je možné položky prohazovat. Seznam je realizovaný knihovnou Vue.Draggable[5].

[5]https://github.com/SortableJS/Vue.Draggable

5.6.6.4 Jízdy

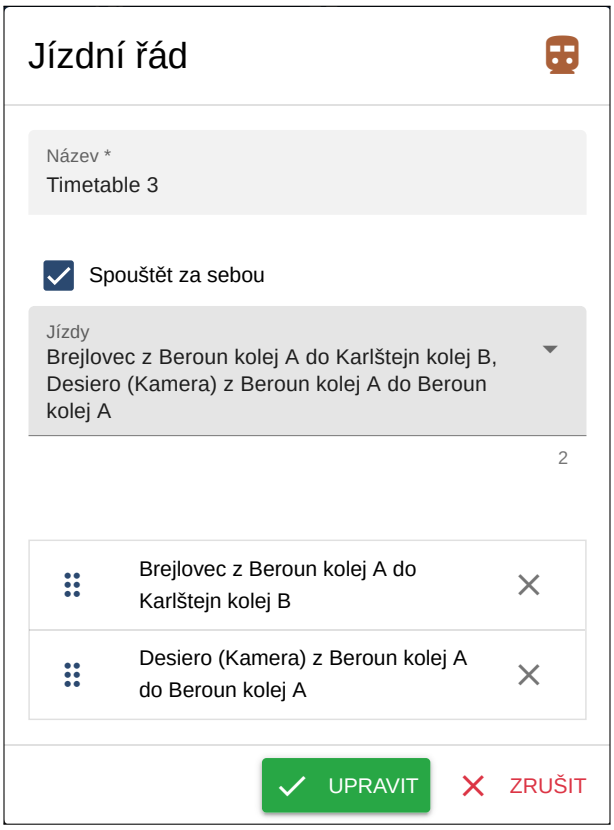


Obrázek 5.24: Komponenta s nastavením jedné jízdy.

Komponenta je zobrazena po kliknutí na přidání nové jízdy, při její úpravě, při zobrazení detailu z levého menu a v průvodci přidáním. Tyto zobrazení obsahují stejné prvky, v případě editace a detailu se načtou původní údaje a v případě detailu se navíc znemožní údaje upravovat.

Obsaženo je výběrové pole na jeden vlak, nastavení rychlosti, informace jestli je jízda zrovna spuštěna a komponenta mapa - výběr trasy.

5.6.6.5 Jízdní řády



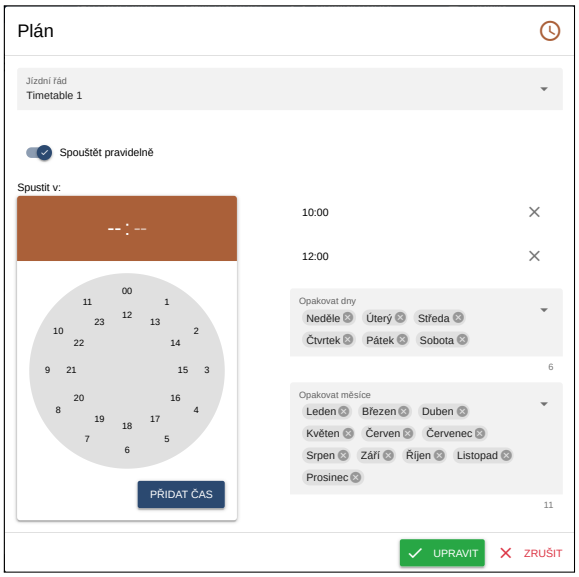
Obrázek 5.25: Strana s nastavením jednoho jízdního řádu.

Chování komponenty je stejné jako komponenty jízdy. Obsahuje textové pole názvu jízdního řádu, pole s výběrem více jízd a zaškrtávací pole, které určuje, jestli jízdy budou spuštěny za sebou nebo současně.

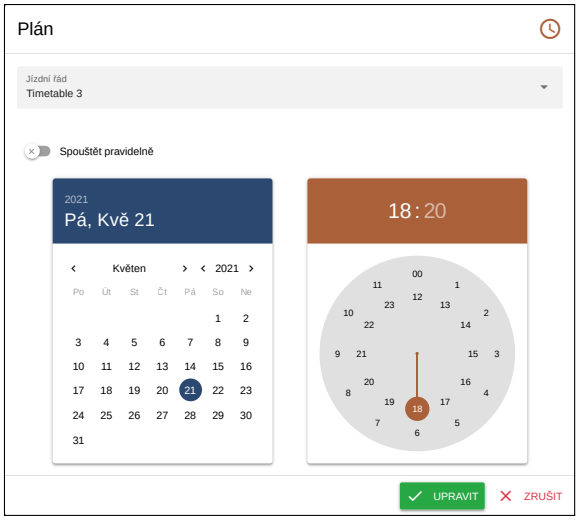
Vybrané jízdy z pole jsou umístěny do seznamu, ve kterém je možné jednotlivé položky prohazovat. Seznam je realizován knihovnou[6]. Jeho účel je určení pořadí jízdy v režimu „spuštění za sebou“.

[6]https://github.com/SortableJS/Vue.Draggable

5.6.6.6 Plánování



Obrázek 5.26: Komponenta s nastavením jednoho plánu - opakující se.



Obrázek 5.27: Komponenta s nastavením jednoho plánu - neopakující se.

Komponenta plánování je zobrazena v administraci jízdních řádů při přidání nového plánu, jeho úpravě a v průvodci přidáním. Detail plánu je možné zobrazit na stránce jízdní řády. Chování je obdobné jako u předcházejících komponent.

Komponenta má dva režimy zobrazení, které je ovlivněno přepínačem „spouštět pravidelně“. V obou režimech je přítomné výběrové pole pro jeden jízdní řád.

V režimu pravidelného spouštění lze nastavit dny, měsíce a časy, ve které se bude plán spouštět. Pro přidání času je nutné navolit čas na ciferníku a kliknout na tlačítko přidat čas. Je možné přidat libovolné množství časů.

V režimu jednorázového spuštění lze na ciferníku navolit čas a v kalendáři datum.

5.6.6.7 Pozadí

Komponenta pozadí obsahuje SVG obrázek a je vložena do všech stránek. Komponenta podporuje zapnutí animace. Pozadí stránky bylo inspirováno touto animací[7].

5.6.6.8 Tabulka chyb

Komponenta je vidět na obrázku 5.16. Chybová hlášení obsahují velké množství informací, proto jsou po načtení zobrazena pouze pole stupeň a zpráva. Libovolně lze zobrazit další pole. Každé hlášení nemusí obsahovat všechny pole.

[7] https://codepen.io/Sepion/pen/ZQJyeq

5.6.6.9 Menu - jízdní řády

Menu je vidět v levé části obrázku 5.5. Je možné ho zobrazit kdekoliv v aplikaci kliknutím do levého horního rohu nebo na mobilním zařízení tažením prstu z levé strany.

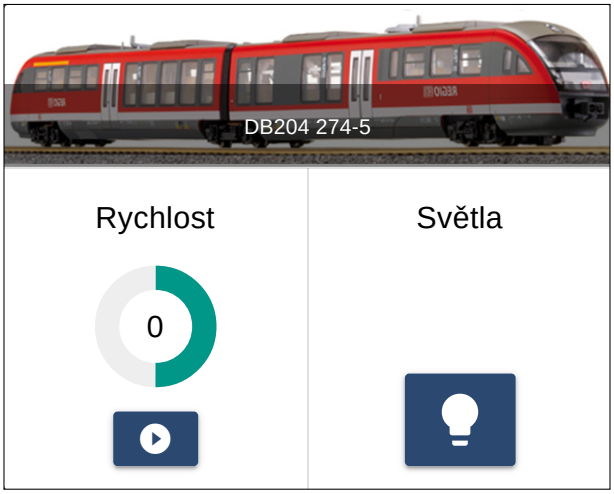
Zobrazuje všechny nastavené jízdy a jízdní řády. Po kliknutí je zobrazen příslušný detail. Pokud má uživatel právo, tak má zobrazené i tlačítko s ikonou šipky. Tímto tlačítkem je možné jízdu nebo jízdní řád spustit bez ohledu na plán.

5.6.6.10 Menu - vlaky

Menu je vidět v pravé části obrázku 5.5. Co se týká zobrazení má stejné vlastnosti jako menu - jízdní řády.

Zobrazuje všechny osazené vlaky. Po kliknutí na řádek se zobrazí konkrétní detail popsaný v kapitole 5.6.6.11. Pod jménem je zobrazena informace jakou rychlostí se vlak pohybuje, případně jestli stojí.

5.6.6.11 Detail vlaku



Obrázek 5.28: Detail vlaku

V detailu vlaku je zobrazen název vlaku a jeho fotografie v případě že je nastavena. Pokud se vlak pohybuje, tak je přednastavena jeho rychlost. V případě, že uživatel má příslušná práva, může rozsvítit světla, nebo spustit vlak bez ohledu na plány.

5.6.7 Grafický vzhled

Při tvorbě vzhledu stránky bylo využito několik otevřených nástrojů a příkladů. Barevná paleta stránky byla vygenerována nástrojem Paletton[8]. Obrázek v konzoli prohlížeče pochází ze stránek ASCII ART[9].

[8] https://paletton.com

[9] https://asciiart.website/index.php?art=transportation/trains

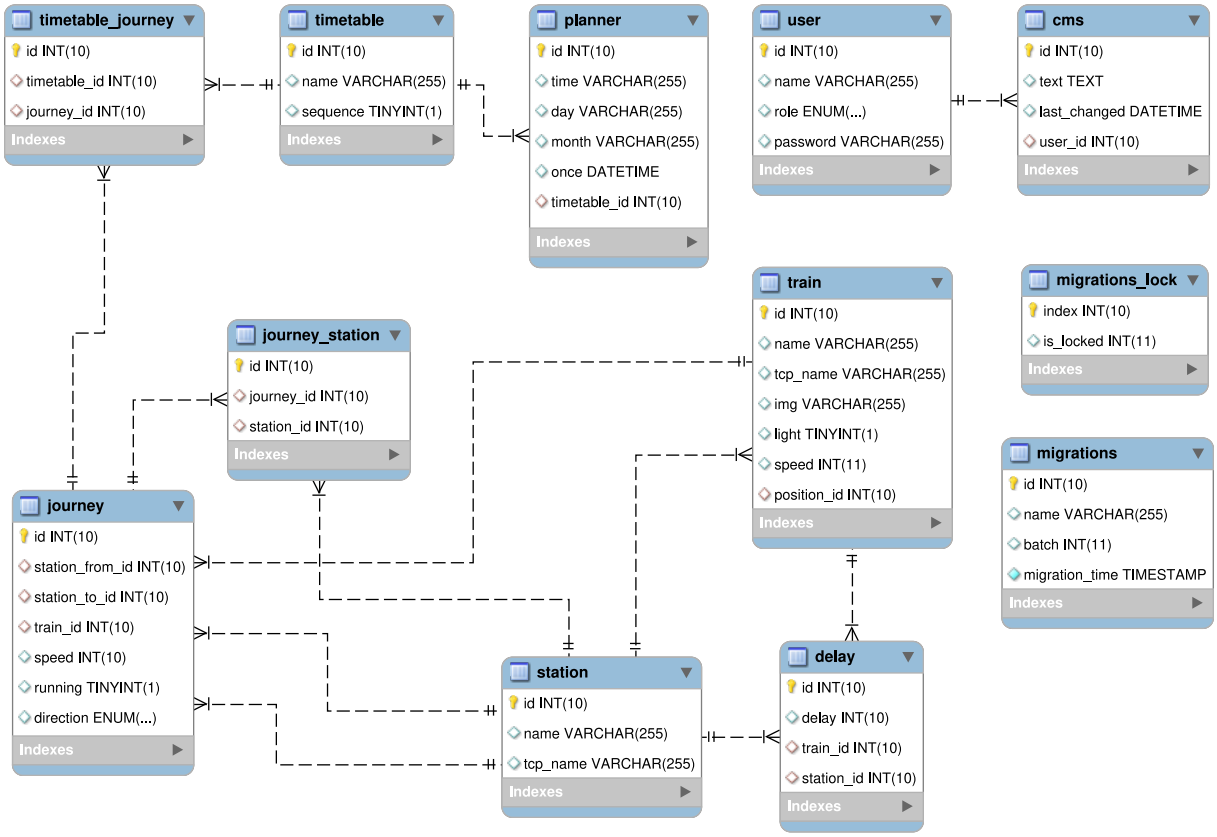
5.7 Server

Každá z částí má ve složce *server/src* svůj spouštěč. Ty slouží k předání nastavení, závislostí a vytvoření a spuštění tříd.

5.7.1 Databáze a data

Jako vrstva mezi databází a aplikací byla zvolena knihovna Objection.js. Nejedná se o celkové ORM, protože pracuje pouze s jazykem SQL, neumí automaticky generovat migrace a definice modelu [26]. Což u projektu této velikosti není omezující. Objection.js je postaven na knihovně Knex.js[10]. Knex.js je nástroj pro sestavování SQL dotazů, tvorbu migrací a dat pro plnění databáze.

Jako spojení s databází MariaDB a Node.js aplikace je využito knihovny mysql2[11].



Obrázek 5.29: Diagram vazeb mezi databázovými tabulkami.

5.7.1.1 Migrace a seed

Migrace jsou důležité při vývoji aplikace, protože umožňují automaticky naplnit databázi tabulkami a vazbami mezi nimi. Není proto nutné vše naklikávat ve správci databáze. Zároveň v případě chyby v datech je snadné celou databázi smazat a vytvořit ji znovu.

[10] http://knexjs.org
[11] https://github.com/sidorares/node-mysql2

Když je aplikace v provozu delší dobu a je naplněná daty, tak není příliš vhodné měnit její strukturu bez využití migrací, protože by například špatným kliknutím mohlo dojít ke ztrátě dat. Migrace lze otestovat ve vývojovém prostředí a až následně spustit v produkčním. To zaručuje bezchybné provedení změn ve struktuře databáze.

Pro vývoj je nutné naplnit databázi daty. Pro to slouží funkce „seeds“. V souboru se nadefinují data a spuštěním se provede jejich nahrání. Zároveň při vývoji jde o nástroj, kterým lze otestovat bezchybnost vazeb mezi tabulkami.

5.7.2 Autentizace a autorizace uživatele

Identita uživatelů je ověřována pomocí JWT. Princip ověření je takový, že uživatel po odeslání přihlašovacích údajů na server (jméno, heslo) dostane jako odpověď textový řetězec, který obsahuje hlavičku, informace o uživateli a podpis. Informace o uživateli jsou JSON zašifrované pouze pomocí Base64, takže nesmí obsahovat žádné citlivé údaje. Čas expirace nelze zfalšovat, protože by nebylo možné ověřit podpis [27].

Tento řetězec je uložen v localStorage prohlížeče a je odeslán při každém požadavku na server, který z data expirace pozná, jestli je požadavek validní. V případě, že není, tak odpoví kódem 401 Unauthorized.

K vytváření a ověření JWT je využita knihovna[12].

Při registraci uživatele je jeho heslo uloženo v databázi jako otisk vytvořený knihovnou bcrypt[13]. Heslo, kterým se uživatel přihlašuje, je vždy ověřováno oproti tomuto otisku. Takže samotné heslo není nikde uloženo.

5.7.3 Controller

Třídy se jménem končící Controller se starají o komunikaci REST API. Jsou vytvořeny spouštěčem ApiBootstrap. Každý controller si vytváří instanci jedné, nebo více tříd service.

Dědí buď od třídy BaseController nebo BaseAuthController. BaseController přidává metody související s autentizací uživatele a nastavuje data současného uživatele.

5.7.4 Hardware

Třídy, kterým jméno končí na Hardware, zajišťují komunikaci s klientem přes Socket.io a předávají instrukce řídící aplikaci přes TCP/IP. Některé třídy si vytváří instance tříd service pro přístup do databáze. Spouštěč HwClientBootstrap vytváří třídu HardwareTcp, která uchovává objekt s připojením k řídící aplikaci. Spouštěč SocketBootstrap vytváří třídy hardware.

Všechny třídy hardware dědí od BaseHardware nebo BaseAuthHardware. BaseAuthHardware přidává metodu pro kontrolu, jestli daný uživatel má práva k přístupu do jmenného prostoru. K tomu je využito funkcionality Socket.io se jménem „middleware“.

[12] https://github.com/auth0/node-jsonwebtoken

[13] https://github.com/dcodeIO/bcrypt.js

5.7.5 Service

Service jsou třídy, které komunikují s Objection.js modely. Slouží pro získávání dat z databáze a jejich převedení do formy vhodné pro další zpracování, nebo jejich ukládání. Zároveň obsahují logiku autentizace uživatelů ve třídě UserService, nebo čtení souborů s logy ve třídě ErrorService.

5.7.6 Model

Model jsou třídy, které definují strukturu databáze, vazby mezi tabulkami pro automatické načítání knihovnou Objection.js a validační pravidla pro vyváření těchto tříd.

5.7.7 Konfigurace

Konfigurace je detailně popsána v příloze A. Konfigurační soubor je validován knihovnou[14].

5.7.8 Výjimky a HTTP stavové kódy

Pokud je v jakékoliv části aplikace potřeba odeslat do prohlížeče chybový kód, je možné vyhodit jakoukoli výjimku, která dědí od třídy Exception. Jsou implementovány běžné kódy 4xx a 500.

Všechny chyby v aplikaci jsou směřovány do souboru *server/src/app/exception/Handler.ts*, kde je rozhodnuto, jestli má být chyba uložena do logu, odeslána k uživateli, nebo obojí. Chyby databáze jsou přeloženy na chybové kódy HTTP. Ukládání logů zajišťuje knihovna Winston[15].

V části cron a Hardware se využívá přímo ukládání chyb knihovnou Winston.

5.7.9 Cron

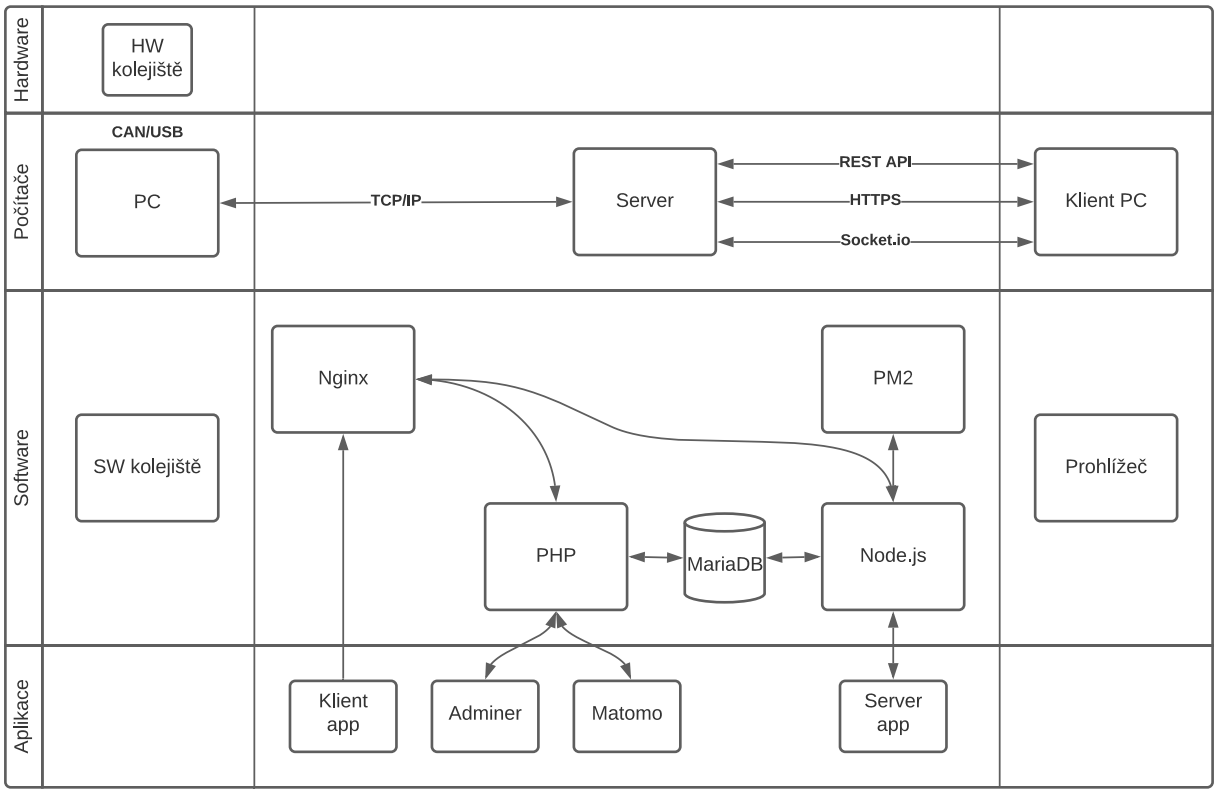
Cron je stále běžící script. Po spuštění si vytvoří spojení s řídící aplikací a posílá každou minutu dotaz do databáze ve které kontroluje tabulku plánů, jestli není čas spustit jízdní řád. Dále kontroluje, jestli nebyl spuštěn jízdní řád uživatelem. Pokud ano, tak předá instrukce řídící aplikaci.

Cron přijímá stavy obsazených úseků z řídící aplikace a ukládá je do databáze.

[14] https://json-schema.org

[15] https://github.com/winstonjs/winston

5.8 Produkční prostředí



Obrázek 5.30: Diagram produkčního prostředí

Na obrázku 5.30 jsou zobrazené vazby mezi jednotlivými bloky. Ve vrstvě hardware jsou obsaženy řídící jednotky kolejiště, které přes USB/CAN komunikují s PC u kolejiště. To komunikuje se serverem přes protokol TCP/IP a s tímto serverem komunikují uživatelé po internetu s využitím HTTPS, přes REST API a pomocí WebSocket (většina uživatelů) přes Socket.io.

Na serveru je nainstalováno Node.js pro běh webové aplikace. Stav serveru a webové aplikace je monitorován pomocí NPM balíčku PM2.

PM2 poskytuje webové rozhraní, kde je viditelné např. současné zatížení serveru, nebo využití paměti. Zároveň hlídá běh skriptů cron.js a app.js a v případě pádu se je pokusí restartovat.

Dále je na serveru nainstalováno PHP kvůli běhu aplikací Adminer[16] a Matomo[17].

Adminer slouží k zobrazení MariaDB databáze a případnému provádění změn za běhu aplikace.

Matomo je software, který poskytuje možnost sledování chování uživatelů, což je vhodné z důvodu budoucí možné optimalizace designu aplikace a také ke sledování počtu uživatelů, kteří ji používají. Jako propojení Vue.js aplikace a Matomo je využita knihovna VueMatomo[18].

[16] https://www.adminer.org
[17] https://matomo.org
[18] https://github.com/AmazingDreams/vue-matomo

Pro přenášení obsahu uživatelům je nasazen Nginx[19]. Jeho doporučená konfigurace je ve složce *docker/production/config*. Konfigurace nyní obsahuje přesměrování všech HTTP požadavků na HTTPS. Komunikace Socket.io probíhá přes TLS. Předání požadavků na adrese */analytics* aplikaci Matomo a požadavků na adrese */adminer* aplikaci Adminer. Certifikát je obnovován pomocí služby certbot[20].

Na adresách */api* a */socket* je nastavena proxy pro předání požadavků běžící aplikaci server a adresa / přímo posílá soubory aplikace klient. Sice je možné využít pro servírování obsahu klienta i Node.js server, ale Nginx je o něco málo efektivnější a musí být přítomný hlavně z důvodu běhu PHP aplikací.

[19]https://www.nginx.com
[20]https://certbot.eff.org

6

Možnosti zlepšení a další vývoj

Po dokončení vývoje hardware kolejiště bude potřeba upravit jak řídící, tak webovou aplikaci a rozšířit je o možnost ovládat a zobrazovat více úseků kolejiště. Dále je potřeba přidat zobrazení nastavení výhybek a semaforů. Tyto dvě vrstvy nyní s výhybkami a semafory nijak nepracují. Do budoucnosti je potřebné vytvořit bezpečnostní vrstvu.

Úkol bezpečnostní vrstvy by mělo být hlídání provozu na kolejišti. To znamená, že pokud běží nastavené jízdní řády a někdo spustí lokomotivu, aplikace musí zjistit kolizi a info odeslat uživateli aplikace. Tato SW vrstva by dále měla umět nastavovat výhybky a semafory, podle zadaného příkazu, kam a přes jaké úseky se má vlak pohybovat.

Co se týká webové aplikace by bylo vhodné naprogramovat kontrolu kolizí nastavených jízdních řádů. Nyní lze kontrolu provést manuálně na stránce jízdní řády.

Jistě by bylo efektní zprovoznit kameru a přenášet živý obraz do webové aplikace, některé vlaky rovněž disponují kamerou.

Nyní není implementována funkce spouštění jízdních řádů za sebou v cronu aplikace. Dále není zcela dokončeno přijímání obsazených úseků z řídícího SW. Tyto funkcionality budou doplněny po zprovoznění kolejiště.

Co se týká mapy bylo by realizovatelné model kolejiště přiblížit realitě a místo pouhého zobrazování obsazených úseků animovat pohyb vlaků na kolejišti. Toto by bylo možné dostat až do úrovně 3D modelu kolejiště s využitím knihovny three.js[1].

Další možnost vylepšení se nabízí v optimalizaci programu cron.js, místo periodického dotazování se databáze, by bylo možné zapracovat na plánování. U takového řešení je ale potřeba řešit jakoukoli změnu jízdního řádu. Podobnou optimalizaci by bylo možné udělat i při odesílání pozic vlaků do prohlížeče. Kdy místo periodického zasílaní pozic by se odesílali pouze změny. Vzhledem k využití, by ale tyto optimalizace nejspíše nepřinesli měřitelné zrychlení.

[1] https://threejs.org

7

Závěr

V rámci této práce bylo navrženo více přístupů a technologií. Byly vybrány vhodné technologie pro vývoj webové aplikace s definovanými vlastnostmi. Aplikace byla implementována s využitím jazyka Typescript a otevřených knihoven. Základní stavební prvky části klient jsou progresivní knihovna Vue.js a knihovna komponent Quasar. Server byl naprogramován s pomocí knihovny Koa.js. Napojení na databázovou vrstvu realizuje knihovna Objection.js. Části mezi sebou komunikují s pomocí REST API a knihovny Socket.io.

Dále bylo navrženo nastavení prostředí pro běh aplikace na serveru. K tomu je využit Nginx, který zajišťuje komunikaci přes protokol TLS. K ukládání dat byla zvolena databáze MariaDB.

V době tvorby této práce (květen 2021) nebylo kolejiště provozuschopné, proto byla funkce ověřena s pomocí řídícího software, který nebyl připojen ke kolejišti.

V elektronické formě jsou přiloženy soubory se všemi kódy. Ty jsou také dostupné online na platformě Gitlab[1]. Aplikace je spuštěna na serveru univerzity[2].

[1] https://gitlab.com/krystof.trko/web_trains

[2] https://sulis52.zcu.cz

Reference, použitá literatura

1. LAPUNÍK, Vojtěch. *Řídící software pro modelové kolejiště*. Plzeň, 2019. Bak. pr. Západočeská univerzita v Plzni, Fakulta elektrotechnická. Vedoucí práce Ing. Petr Weissar PH.D.
2. *NPM* [online] [cit. 2021-05-01]. Dostupné z: `https://docs.npmjs.com/about-npm`.
3. ZLATKOV, Alexander. *How JavaScript works: A comparison with WebAssembly + why in certain cases it's better to use it over JavaScript* [online] [cit. 2021-05-01]. Dostupné z: `https://blog.sessionstack.com/how-javascript-works-a-comparison-with-webassembly-why-in-certain-cases-its-better-to-use-it-d80945172d79`.
4. MACRAE, Callum. *Vue.js: up and running: building accessible and performant web apps*. Sebastopol: O'Reilly, 2018. ISBN 9781491997246.
5. *jQuery* [online] [cit. 2021-05-01]. Dostupné z: `https://jquery.com`.
6. VOVCHUK, Yuriy. *Srovnání progresivních frameworků Vue.js, React a Angular*. Praha, 2019. Bak. pr. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce Ing. Filip Vencovský PH.D.
7. *node.js* [online] [cit. 2021-05-01]. Dostupné z: `https://nodejs.org/en/about`.
8. *Node.js vs PHP* [online] [cit. 2021-05-01]. Dostupné z: `https://www.simform.com/nodejs-vs-php`.
9. *Express* [online] [cit. 2021-05-01]. Dostupné z: `https://expressjs.com`.
10. *koa* [online] [cit. 2021-05-01]. Dostupné z: `https://koajs.com`.
11. *Sails* [online] [cit. 2021-05-01]. Dostupné z: `https://sailsjs.com`.
12. KANERIYA, Tejas. *12 Best Nodejs Frameworks for Web Apps in 2021* [online] [cit. 2021-05-01]. Dostupné z: `https://www.simform.com/best-nodejs-frameworks`.
13. CARNAHAN, Jeff. *When to use a HTTP call instead of a WebSocket (or HTTP 2.0)* [online] [cit. 2021-05-01]. Dostupné z: `https://blogs.windows.com/windowsdeveloper/2016/03/14/when-to-use-a-http-call-instead-of-a-websocket-or-http-2-0`.
14. *Web RTC* [online] [cit. 2021-05-01]. Dostupné z: `https://webrtc.org`.

15. *socket.io* [online] [cit. 2021-05-01]. Dostupné z: https://socket.io/docs/v4.

16. *Date fns* [online] [cit. 2021-05-01]. Dostupné z: https://date-fns.org.

17. *lodash* [online] [cit. 2021-05-01]. Dostupné z: https://lodash.com.

18. MICROSOFT. *typescript* [online] [cit. 2021-05-01]. Dostupné z: https://www.typescriptlang.org.

19. *eslint* [online] [cit. 2021-05-01]. Dostupné z: https://eslint.org.

20. *sass* [online] [cit. 2021-05-01]. Dostupné z: https://sass-lang.com/guide.

21. *webpack* [online] [cit. 2021-05-01]. Dostupné z: https://webpack.js.org/guides/getting-started.

22. *Vue CLI* [online] [cit. 2021-05-01]. Dostupné z: https://cli.vuejs.org/guide.

23. *Docker* [online] [cit. 2021-05-01]. Dostupné z: https://docs.docker.com/get-started.

24. FREDRICH, Todd. *Rest API Tutorial* [online] [cit. 2021-05-01]. Dostupné z: https://www.restapitutorial.com/lessons/whatisrest.html.

25. MOROTE, Eduardo San Martin. *pinia* [online] [cit. 2021-05-01]. Dostupné z: https://github.com/posva/pinia.

26. *objection* [online] [cit. 2021-05-01]. Dostupné z: https://vincit.github.io/objection.js.

27. *JWT* [online] [cit. 2021-05-01]. Dostupné z: https://jwt.io/introduction.

Příloha A

Příkazy a konfigurace

Všechny příkazy jsou uvedeny také ve zdrojových kódech v souboru README.md.

A.1 Nastavení projektu pro vývoj

Zkopírování ukázkového konfiguračního souboru:

```
cp ./server/src/config/config.example.json ./server/src/config/config.json
```

A.1.1 S dockerem

Zkopírování docker-compose.yml:

```
cp ./docker/development/docker-compose.example.yml ./docker-compose.yml
```

Pro použití lokálního ssh klíče:

```
eval `ssh-agent`
ssh-add ~/.ssh/id_rsa
```

Build kontejnerů a instalace npm balíčků:

```
docker-compose build
docker-compose run trains_webserver bash -c "npm i"
```

Spuštění kontejnerů:

```
docker-compose up
```

Připojení se do konzole vývojového kontejneru:

```
docker exec -it trains_webserver bash
```

K použití matomo je potřeba přidat v matomo kontejneru do souboru /var/www/config/config.ini:

```
trusted_hosts[] = "localhost:8088"
trusted_hosts[] = "trains_matomo:80"
proxy_uri_header = 1
proxy_client_headers[] = HTTP_X_FORWARDED_FOR
proxy_host_headers[] = HTTP_X_FORWARDED_HOST
```

K automatickému spuštění kontejnerů, připojení do něj a nastavení SSH klíče lze použít:

```
./up.sh
```

Aplikace klienta je spuštěna na portu 8080.
Aplikace serveru je spuštěna na portu 8081 nebo localhost:8080/api.
Adminer je na portu 8089 nebo localhost:8080/adminer přihlášení je:

- server: trains_db
- username: web_trains
- password: test1234
- db: web_trains

Matotmo je na portu 8088 nebo localhost:8080/analytics

A.1.2 Bez dockeru

Instalace globálních a lokálních npm balíčků:

```
npm i -g npm
npm i -g nodemon @vue/cli@^5.0.0-beta.0
npm i -g @quasar/icongenie --unsafe-perm
npm install
npm run serve
```

Spuštění server aplikace a Webpack dev serveru:

```
npm run serve
```

Aplikace klienta je spuštěna na portu 8080.
Aplikace serveru je spuštěna na portu 8081 nebo localhost:8080/api.
V případě potřeby je nutné nainstalovat matomo a adminer ručně.

A.1.3 Vývoj

Kompilace a minifikace, sloučení obou aplikací:

```
npm run build
```

Kontrola kódu:

```
npm run lint
```

A.1.4 Vývoj server

Nová db migrace:

```
npx knex migrate:make name
```

Spuštění poslední migrace:

```
npx knex migrate:latest
```

Nahrání ukázkových dat do db:

```
npx knex seed:run
```

A.1.5 Vývoj klient

Vygenerování všech ikon:

```
icongenie generate -m spa -i src/assets/icon-train-blue.png
```

A.1.6 Test produkčního nastavení docker

Vygenerování sebou podepsaného certifikátu:

```
openssl req -x509 -out web_trains.crt -keyout web_trains.key \
        -newkey rsa:2048 -nodes -sha256 \
        -subj '/CN=localhost' -extensions EXT -config <( \
        printf "[dn]\nCN=localhost\n[req]\ndistinguished_name = \
        dn\n[EXT]\nsubjectAltName=DNS:localhost\nkeyUsage\
        =digitalSignature\nextendedKeyUsage=serverAuth")
```

Kopie docker-compose.yml a spuštění kontejneru:
Pozor ve složce dist musí být sestavené soubory.

```
cp ./docker/production/docker-compose.example.yml ./docker-compose.prod.yml
docker-compose -f docker-compose.prod.yml up
```

Připojení se k docker kontejneru:

```
docker exec -it trains_prod_webserver bash
```

Ke spuštění kontejneru a připojení do něj je možné využít:

```
./up.sh prod
```

Aplikace běží na portu 80
Matomo: localhost/analytics
Adminer: localhost/adminer

A.1.7 Nastavení produkčního serveru

Vytvoření přístupů do db s následujícími údaji.
Pro aplikaci:

- db: web_rains
- user: web_trains
- password: test1234

Pro matomo:

- db: matomo
- user: matomo
- password: test1234

```
mysql_secure_installation

mysql

CREATE USER 'web_trains'@'localhost' IDENTIFIED BY 'test1234';
CREATE DATABASE web_trains;
GRANT ALL PRIVILEGES ON web_trains . * TO 'web_trains'@'localhost';
CREATE USER 'matomo'@'localhost' IDENTIFIED BY 'test1234';
CREATE DATABASE matomo;
GRANT ALL PRIVILEGES ON matomo . * TO 'matomo'@'localhost';
FLUSH PRIVILEGES;
```

Monitorování spuštěných aplikací:

```
pm2 monit
```

Obnova https certifikátu pomocí certbot https://certbot.eff.org/

A.1.8 Konfigurace

Umístění config.json záleží na prostředí:

- Produkce: /config
- Vývoj: /server/app/config

```
"db": {
        "host": "trains_db",
        "user": "web_trains",
        "password": "test1234",
        "database": "web_trains"
},
"app": {
        "port": 8081,
        "listen": "0.0.0.0",
        "secret": "oijfjo45",
        "mode": "production"
},
"hardware": {
        "port": 8083,
        "host": "false"
}
```

db:

- host: adresa databáze
- user: uživatel databáze
- password: heslo databáze
- database: jméno databáze

app:

- port: port na kterém běží server aplikace: api a socket
- listen: adresa pro server
- secret: používaný pro generování JWT
- mode
  - production: logy jsou ukládany do souborů
  - development: logy jsou vypisovány do konzole
  - env: rozhodnutí podle prostředí

hardware:

- port: port řídící aplikace
- host: adresa řídící aplikace, pro zakázní nastavit false