FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	ETCS - RBC I
Student:	Matěj Gorgol
Vedoucí:	Ing. Jiří Chludil
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

ETCS (European Train Control System) je jednotný celoevropský vlakový zabezpečovací systém. Cílem této práce je pro simulátor ETCS vytvořit modul RBC (Radio Block Center).

1. Analyzujte dostupnou dokumentaci k ETCS (ERTMS/ETCS subset-026) s ohledem na fungování RBC.
2. Na základě konzultací se zadavateli z Fakulty dopravní definujte funkční a nefunkční požadavky vybraných funkcionalit RBC.
3. Pomocí metod softwarového inženýrství navrhněte vybrané funkcionality RBC.
 - Prototyp RBC musí být funkční na vzdáleném serveru s operačním systémem Linux.
 - Pro RBC navrhněte prototyp lektorského PC.
4. Pomocí vhodných testů funkčnost prototypu řádně otestujte.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 3. února 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

ETCS-RBC I

Matěj Gorgol

Katedra softwarového inženýrství
Vedoucí práce: Jiří Chludil

12. května 2022

Poděkování

V první řadě bych chtěl poděkovat vedoucímu mé práce, Ing. Jiřímu Chludilovi za cenné rady, přátelský přístup a možnost zúčastnit se tohoto projektu. Dále bych chtěl poděkovat svým přátelům, kteří mě i v nejtěžších chvílích dokázali přivést na jiné myšlenky, rozveselit a namotivovat. Na závěr bych chtěl nejvíce poděkovat své mámě, protože bez její obrovské podpory bych se k psaní této práce nedostal.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 12. května 2022 . . . . . .

Odkaz na tuto práci

Gorgol, Matěj. *ETCS-RBC I.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Práce se zaměřuje na dokončení modulu RBC (Radio Block Centre) a vytvoření REST API Lektorského PC (ovládací prvek simulace) pro projekt vlakového simulátoru se systémem ETCS. Nejprve se práce zabývá představením systému ETCS a jeho vybraných částí, poté se věnuje analýze stávajícího stavu projektu a vymezení nových funkčních a nefunkčních požadavků modulu RBC a Lektorského PC. Dále se v práci nachází návrh Lektorského PC, jehož součástí je návrh architektury a samotného API, představení použitých technologií a porovnání knihoven pro tvorbu REST API v C++. Na to navazuje kapitola realizace, ve které se nachází shrnutí implementace Lektorského PC, testování Lektorského PC, programátorská a instalační příručka.

Klíčová slova systém ETCS, RBC, vlaková doprava, REST API, C++, Crow

Abstract

The thesis focusses on finalizing the RBC (Radio Block Centre) module and creating a REST API for a simulation control module (Lecturer PC) of the ETCS equipped train simulator project. First, the thesis introduces the ETCS system and its most important parts, and then focusses on analyzing the current state of the project. Afterwards it specifies the functional and non-functional requirements for the Lecturer PC and RBC module. Additionally, there is a design of the Lecturer PC, which includes design of the architecture and the REST API, introduction of the used technologies and comparison of C++ REST API libraries. Last the thesis includes realization chapter which contains summary of Lecturer PC implementation, testing and installation and programming guides.

Keywords ETCS system, RBC, train transport, REST API, C++, Crow

Obsah

Seznam obrázků

Úvod

V současné době se v EU, a tedy i v ČR začíná zavádět systém ETCS. ETCS je jednotný evropský vlakový zabezpečovací systém, který by měl nahradit a sjednotit současné vlakové zabezpečovací systémy v celé EU. Zavedení tohoto systému v ČR se plánuje na rok 2025 pro hlavní koridory a v planu je vybavit všechny tratě tímto systémem do roku 2040 [1].

Vybavení tratí a vozidel je ale pouze částí přechodu na tento nový systém. Je potřeba i řádně přeškolit strojvedoucí, aby uměli s tímto novým systémem pracovat. Součástí tohoto přeškolení je i trénink na simulátoru ETCS. Na jednom takovém simulátoru spolupracují FD a FIT ČVUT. Tento simulátor by měl sloužit pro výuku studentů, školení strojvedoucích, propagaci a v neposlední řadě výzkum. Simulátor zahrnuje klíčové prvky ETCS s realistickým chováním a provádí simulace jízdy vlaku.

Tato bakalářská práce navazuje na projekt studentů vytvořený během předmětů *Softwarový týmový projekt 1* a *Softwarový týmový projekt 2*. Výsledkem byl rozpracovaný simulátor, jehož jednotlivé moduly byly v různém stavu dokončenosti. Modul RBC, na kterém jsem pracoval, zvládal načítat data o trati z databáze a komunikaci s modulem EVC, ale zatím nezvládal výpočet a sestavení paketu s povolením k jízdě (tzv. *Movement Authority*).

V této práci se tedy zabývám dokončením modulu RBC a návrhem, implementací a testováním REST API Lektorského PC, které by mělo sloužit jako ovládací prvek pro ostatní moduly a celou simulaci. Společně s touto prací vznikají na téma ETCS další 4 bakalářské práce. K modulu RBC a klienstské části Lektorského PC vzniká bakalářská práce Jiřího Sternwalda [2]. K modulu EVC vznikají bakalářské práce od Aliny Krasnenkové [3] a Darii Roshchupkinové [4] a k modulu DMI displeje ji píše Jan Stejskal [5].

Téma jsem si vybral, protože jsem chtěl na tomto projektu pokračovat a myslím si, že výsledný produkt bude mít nějaké reálné využití a nepřímo snad i pozitivní dopad na bezpečnost v železniční dopravě.

Cíle práce

Tato práce má 2 hlavní cíle. Prvním cílem je dokončení modulu RBC, tak aby modul správně počítal délky povolení k jízdě a mohl tedy být využit v simulaci systému ETCS.

Druhým cílem je návrh, realizace a otestování REST API Lektorského PC, které by po dokončení klientské části, které se věnuje Jiří Sternwald ve své bakalářské práci, sloužilo k ovládání simulace systému ETCS.

V kapitole Analýza bude stručně představen systém ETCS a jeho části. Dále se zde rozebere aktuální stav projektu simulátoru ETCS a modulu RBC. Poté se vydefinují funkční a nefunkční požadavky pro modul RBC a Lektorské PC, které se dohodli na konzultacích se zadavateli z Fakulty dopravní ČVUT.

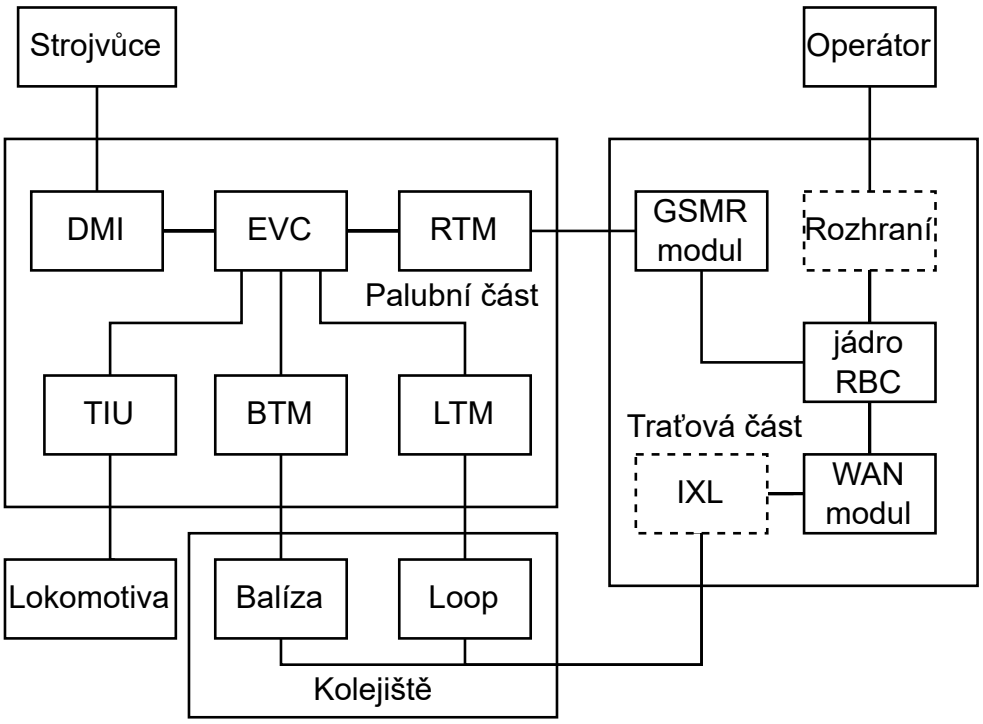
Kapitola Návrh bude sloužit na představení metod, které byly zvoleny pro vývoj Lektorského PC a technologie, které k tomu byly použity. Dále se zde představí jeho struktura, koncové body a operace nad nimi. Na konci kapitoly se nachází návrhy paketů, které by měly sloužit ke komunikaci mezi moduly.

Obsahem kapitoly Realizace bude shrnutí implementované části a popis testování. Zde se také bude nacházet programátorská a instalační příručka, která bude sloužit k přiblížení vnitřního fungování Lektorského PC, respektive jeho instalaci a zprovoznění.

Kapitola 1

Analýza

V této kapitole představím zabezpečovací systém ETCS a jeho části se kterými se v této práci pracuje. Poté se budu věnovat analýze současného stavu projektu ETCS a modulu RBC. Na konci kapitoly vydefinuji funkční a nefunkční požadavky modulu RBC a Lektorského PC.



Obrázek 1.1: Jednotlivé části ETCS a jejich vzájemná komunikace [6]

1.1 Systém ETCS

> *„Je to jednotný celoevropský zabezpečovací systém, který bude zajišťovat jednotnou evropskou železnici a vyšší bezpečnost provozu na ní. Zajistí také, aby vlaky dopravců mohly volně, bez problémů přejíždět z jedné země do druhé.“* [7]

Systém by měl nahradit velké množství nespolupracujících a často zastaralých traťových zabezpečovacích systému evropských zemí. Bez jeho implementace se neobejde budování vysokorychlostních tratí, a zároveň má jeho přítomnost pozitivní dopad na bezpečnost a provozní rychlosti i na klasických tratích [7].

1.1.1 Úrovně ETCS

Systém ETCS je tvořen více částmi, jejich kombinace a úroveň zapojení umožňují dosažení několika úrovní zabezpečení. Definování těchto úrovní umožní výběr vhodného stupně zabezpečení pro trať, ať už na základě vybavovací strategie, dostupné traťové infrastruktury, nebo požadovaného výkonu. Úrovně označujeme písmenem L (z anglického názvu *level*).

- **ETCS L0**
  Vlak vybavený ETCS provozovaný na trati, která není vybavena žádným systémem řízení vlaku. Sytém hlídá pouze maximální rychlost vlaku.

- **ETCS L1**
  Vlak vybavený ETCS provozovaný na trati vybavené návěstidly a balízami. Ty slouží k přenosu informací o trati a povolení k jízdě. Od této úrovně dál už je dostupný režim plného dohledu systému ETCS (*Full Supervision*).

- **ETCS L2**
  Vlak vybavený ETCS provozovaný na trati vybavené balízami a pod kontrolou RBC. Kontrola pozice a integrity vlaku probíhá v traťové části. Odpadá nutnost návěstidel. Touto úrovní se v ČR zabezpečují hlavní koridory.

- **ETCS L3**
  Podobná jako L2, ale kontrola pozice a celistvosti vlaku probíhá v palubní části. Odpadá nutnost klasických prostředků pro detekci vlaku. V této úrovni se místo pevných traťových oddílů pracuje s tzv. „pohyblivými oddíly“, které jsou určeny parametry vlaku a tratě. Tato úroveň není kompatibilní s ostatními úrovněmi, tedy na úseku s touto úrovní zabezpečení lze provozovat pouze vlaky v režimu ETCS L3. [8][9][10]

1.1.2 Části ETCS

V této sekci popisuji hlavní úkoly modulů, které jsou součástí projektu simulátoru ETCS. Systém ETCS dělíme na 2 části:

Palubní část

- **EVC**
  Modul EVC (European Vital Computer) je centrálním bodem celého systému. Přijímá a vyhodnocuje data přijatá z ostatních modulů a vypočítává brzdné křivky. Má také za úkol případně regulovat rychlost vlaku. V našem projektu je tento modul rozdělený na 2 projekty: výpočet brzdné křivky a EVC. [8]

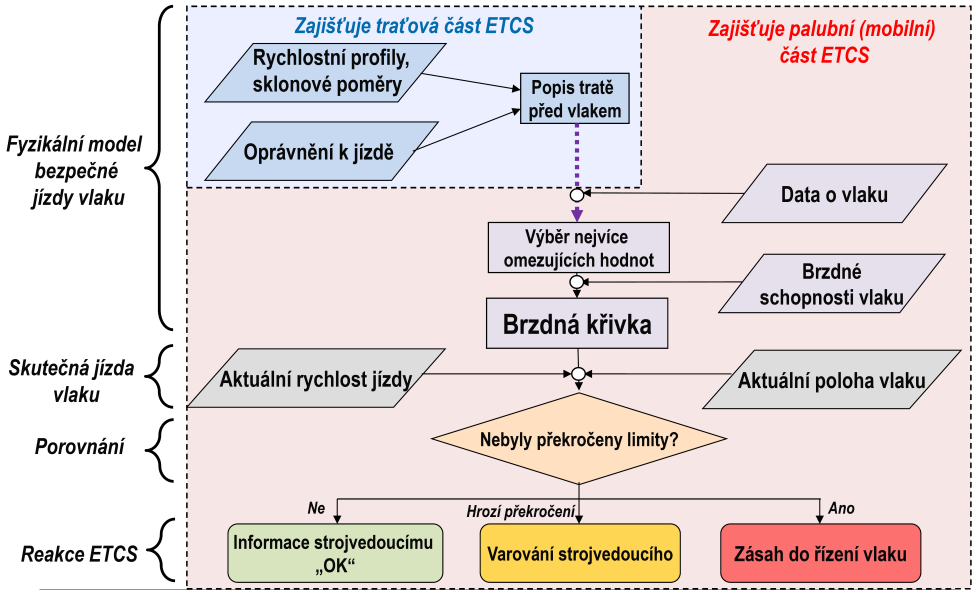
- **JRU**
  Modul JRU (Juridical Recording Unit) je součástí EVC a funguje jako černá skříňka pro vlaky. Zaznamenává posledních několik úkonů strojvedoucího, stavů návěstidel a informací o vlaku. Tento modul vyžaduje vysokou fyzickou odolnost pro případ mimořádné události. [8]

- **DMI**
  Modul DMI (Driver Machine Interface), dříve také MMI (Man Machine Interface), je standardizované uživatelské rozhraní, které slouží k ovládání mobilní části ETCS a kde se strojvedoucímu zobrazují informace od EVC. Mezi zobrazované informace patří mimo jiné například současná a maximální povolená rychlost vlaku, aktuální režim ETCS, délka povolení k jízdě a průběh rychlostního profilu. [8]

Traťová část

- **RBC**
  Modul RBC (Radio Block Center) slouží ke generování povolení k vjezdu tzv. *Movement Authority*. Tyto povolení počítá na základě dat o poloze a stavu vlaku a informací o trati. Tyto povolení jsou ve formě zpráv, které se předávají EVC ve vlaku. Na tento modul se budu zaměřovat ve své práci. [8]

- **Eurobalíza**
  Někdy též jen balíza, je primární informační zdroj pro vlak. Při průjezdu vlaku nad balízou bezdrátově přenáší informace o trati a zároveň balíza slouží jako referenční bod při určování polohy. Balízy se na trati většinou vyskytují ve skupinách, takzvaných *Balise Groups*. [8]



Obrázek 1.2: Blokové schéma systému ETCS [7]

Lektorské PC

Lektorské PC není standardní součástí ETCS, ale v našem projektu slouží jako hlavní ovládací a dozorčí prvek simulace tohoto systému. Je pomocí něj možné spouštět, zastavit a restartovat simulaci, dále slouží k výběru simulované trati a scénáře jízdy. V budoucnu z něj bude možné sledovat informace o stavu jednotlivých modulů ETCS, jejich vzájemnou komunikaci a informace o vlaku. Návrh, implementace a testování REST API Lektorského PC je obsaženo v této práce.

1.2 Stav projektu

1.2.1 Stav ETCS

Tato bakalářská práce navazuje na projekt studentů, vytvořený během předmětů *Softwarový týmový projekt 1* a *Softwarový týmový projekt 2*. Vzhledem k rozdílné náročnosti jednotlivých modulů a faktu, že někteří studenti využili možnost na projektu pracovat během letních prázdnin, byly jednotlivé části projektu v rozdílném stavu. Některé části projektu byly relativně funkční, zatímco jiné ještě nešlo ani pořádně spustit. Celkově tedy nebyl projekt funkční.

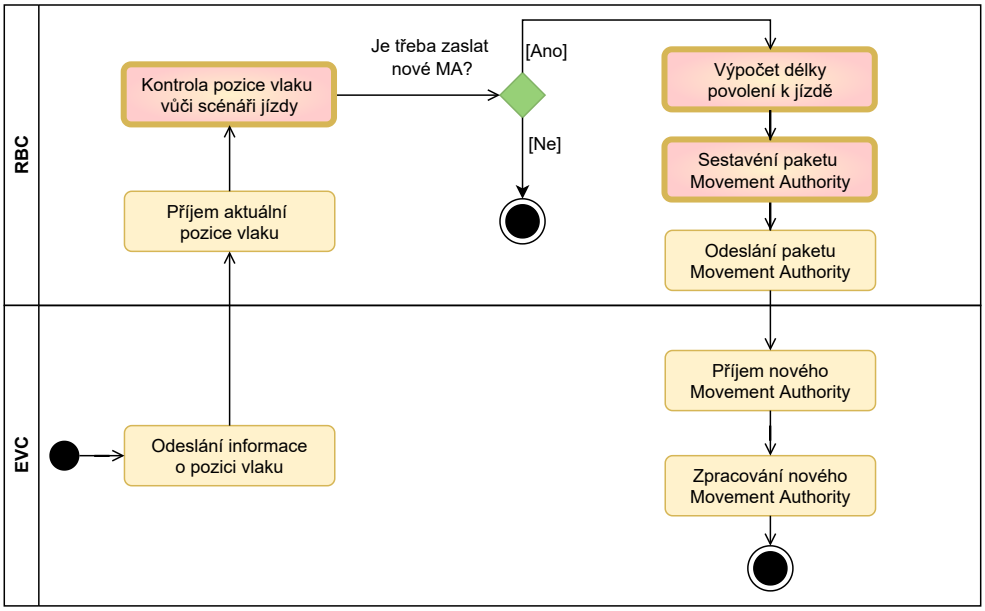
1.2.2 Stav RBC

Modul RBC, na jehož vývoji jsem se podílel, byl jeden z nedodělaných modulů. Na konci předmětů *Softwarový týmový projekt 1* a *Softwarový týmový projekt 2* se modul dokázal připojit k databázi, načíst z ní data o trati a úspěšně odesílat a přijímat zprávy z EVC pomocí MQTT brokeru. Pořád ale nebyla implementovaná snad ta nejdůležitější část, a to vypočítávání a vydávání povolení k jízdě (tzv. *Movement Authority*). Tedy z aktivity diagramu 1.3 byl implementován pouze:

- Příjem aktuální pozice vlaku
- Odeslání paketu Movement Authority

A chybělo doimplementovat:

- Kontrola pozice vlaku vůči scénáři jízdy
- Výpočet délky povolení k jízdě
- Sestavení paketu Movement Authority



Obrázek 1.3: Diagram aktivity RBC – vydávání povolení ke vjezdu

1.3 Funkční a nefunkční požadavky

Na základě konzultací se zadavateli z FD ČVUT se určily tyto funkční a nefunkční požadavky pro modul RBC a Lektorské PC:

1.3.1 Funkční požadavky RBC

F1 Komunikace s EVC
Systém bude obsahovat modul komunikace, který bude odesílat a přijímat pakety definované v subsetu 026 7. Ke komunikaci bude využívat síťový protokol MQTT.

F2 Management vnitřních struktur trati
Systém bude obsahovat databázový modul, který bude umožňovat načítání informací o infrastruktuře trati z databáze. Data budou obsahovat například pozice balízových skupin, výškové gradienty a rychlostní omezení.

F3 Komunikace s Lektorským PC
Systém bude obsahovat modul komunikace s Lektorským PC, který bude umět odesílat informace o aktuální poloze vlaku a přijímat a zpracovávat příkazy, které přímo ovlivňují povolení k jízdě. Pro komunikaci bude využívat síťový protokol MQTT.

F4 Načítání informací z konfiguračních souborů
Systém bude schopen načítat aktuální informace o dostupnosti trati z konfiguračních souborů, které mu budou dodány Lektorským PC. Na základě polohy vlaku a informací v konfiguračních souborech bude systém informován o novém dostupném úseku. Tato funkcionalita simuluje postupné uvolňování trati, tak jak k němu dochází v reálném prostředí. Konfigurační soubory jsou ve formátu JSON.

F5 Vydávání povolení k jízdě (Movement Authority)
Systém bude schopen na základě informací o dostupnosti trati a o pozici vlaku počítat vzdálenosti a generovat povolení k jízdě. Každý paket s povolením k jízdě bude obsahovat informace o výškovém gradientu a rychlostních omezeních platných na danou vzdálenost, tak jak je definováno v subsetech 026-3 a 026-7 [8].

F6 Výběr scénáře jízdy pro simulaci
Systém bude schopen na základě zprávy z Lektorského PC nahrát přijatý scénář jízdy a podle něj simulovat vydávání povolení ke vjezdu.

F7 Výběr trati pro simulaci
Systém se bude schopen na základě zprávy z Lektorského PC připojit k požadované databázi a nahrát vybranou trať do svých vnitřních struktur a s tou pak nadále pracovat.

F8 Samostatný běh
Systém bude po spuštění schopen samostatně (tj. bez nutnosti dalšího zásahu operátora) obsluhovat vlaky.

F9 Zobrazení aktuálního stavu systému
Systém bude vypisovat stručné informace o stavu vygenerovaných dat a přijatých a odeslaných paketech do příkazové řádky. Zde se rovněž budou vypisovat chybové hlášky z funkčních požadavků F10 a F11.

F10 Řešení neúspěšného připojení k EVC
Systém bude uživatele informovat o neúspěšném pokusu připojení k EVC. V případě neúspěšného pokusu o připojení se bude s rostoucím časovým odstupem pokoušet o znovupřipojení k EVC. Po dvacátém neúspěšném pokusu o připojení systém vypíše chybové hlášení a ukončí se.

F11 Řešení neúspěšného připojení k databázi
Systém bude uživatele informovat o neúspěšném pokusu připojení k databázi. V případě neúspěšného pokusu o připojení se bude s rostoucím časovým odstupem pokoušet o znovupřipojení k databázi. Po dvacátém neúspěšném pokusu o připojení systém vypíše chybové hlášení a ukončí se.

F12 Validace dat
Systém bude ověřovat a zpracovávat všechny přijaté a odchozí pakety, na základě jejich definice v subsetu 026-7 [8]. Při zjištění chyby v paketu vypíše chybovou hlášku do konzole a v případě odchozího paketu se pokusí o nové sestavení a následné odeslaní paketu.

1.3.2 Nefunkční požadavky RBC

N1 Jazyk implementace
Systém bude implementován v jazyce C++ verze 14.

N2 Databáze popisu infrastruktury
Databáze bude využívat software MySQL verze 1.1.9.

N3 Kompatibilita systému
Systém poběží na vzdáleném virtuálním počítači s operačním systémem Debian 10.

N4 Verze subsetů
Implementace bude podporovat více verzí subsetů ETCS, konkrétně verze: 2.3.0d, 3.4.0, 3.6.0.

N5 Formát scénářů jízdy
Scénáře jízdy budou definované pomocí datového formátu JSON. Pro

práci s daným formátem se bude využívat knihovna Nlohmann-json3 verze 3.5.0.

N6 Komunikace s moduly
Ke komunikaci s ostatními moduly ETCS bude využit síťový protokol MQTT a knihovna Mosquitto verze 1.5.7.

1.3.3 Funkční požadavky Lektorského PC

Pro Lektorské PC máme 3 skupiny funkčních požadavků:

Minimální funkční požadavky – je nutné je splnit, aby systém dával smysl

Optimální funkční požadavky – požadavky, které by měl systém plnit

Další rozšíření funkčních požadavků – vhodné budoucí rozšíření

Minimální funkční požadavky

F1 Komunikace s moduly ETCS
Systém bude obsahovat modul komunikace, který umožní komunikaci s ostatními moduly projektu ETCS. Komunikovat bude pomocí odesílání a přijímání paketů, které jsou definované v kapitole 2.5.1. Ke komunikaci bude využívat síťový protokol MQTT.

F2 Zaslání signálu START
Systém bude umět na vyžádání vyslat signál, který spustí celou simulaci. Tento signál je definovaný paketem 2.1.

F3 Zaslání signálu STOP
Systém bude umět na vyžádání vyslat signál, který zastaví celou simulaci. Tento signál je definovaný paketem 2.1.

F4 Zaslání signálu RESTART
Systém bude umět na vyžádání vyslat signál, který restartuje celou simulaci. Tento signál je definovaný paketem 2.1.

F5 Volba tratě
Systém umožní na začátku simulace zvolit na které trati se bude simulace odehrávat. Informaci o zvolené trati odešle do RBC pomocí paketu 2.6.

F6 Volba scénáře jízdy
Systém umožní na začátku simulace zvolit požadovaný scénář a tento scénář odešle do RBC pomocí paketu 2.8.

F7 Volba počáteční konfigurace
Systém umožní na začátku simulace nastavit počáteční konfiguraci modulů a národní traťové hodnoty. Tyto informace pošle do EVC pomocí paketu 2.10.

Optimální funkční požadavky

F8 Přidání nového scénáře jízdy
Systém umožní přidání nového scénáře jízdy do úložiště Lektorského PC skrze REST API.

F9 Úprava scénáře jízdy
Systém umožní úpravu stávajícího scénáře jízdy skrze REST API.

F10 Odstranění scénáře jízdy
Systém umožní odstranění uloženého scénáře jízdy z úložiště Lektorského PC skrze REST API.

F11 Přidání nové tratě
Systém umožní přidání nové tratě do úložiště Lektorského PC skrze REST API.

F12 Úprava tratě
Systém umožní úpravu stávající tratě skrze REST API.

F13 Odstranění tratě
Systém umožní odstranění uložené tratě z úložiště Lektorského PC skrze REST API.

F14 Přidání nové počáteční konfigurace
Systém umožní přidání nové počáteční konfigurace do úložiště Lektorského PC skrze REST API.

F15 Úprava počáteční konfigurace
Systém umožní úpravu stávající počáteční konfigurace skrze REST API.

F16 Odstranění počáteční konfigurace
Systém umožní odstranění uložené počáteční konfigurace z úložiště Lektorského PC skrze REST API.

Další rozšíření funkčních požadavků

F17 Odebírání informací o poloze vlaku
Systém bude od simulátoru vlaku skrze MQTT odebírat informace o současné poloze vlaku. Poloha bude přijímaná jako absolutní vzdálenost od začátku tratě. Lektorské PC si informaci o poslední poloze vlaku bude udržovat a ta bude pro klientskou část dostupná skrze REST API.

F18 Procentuální výpočet průběhu scénáře
Systém bude schopen z dat o trati a informacích o současné pozice vlaku přibližně spočítat procentuální dokončenost probíhajícího scénáře jízdy.

F19 Kontrola stavu modulů
Systém bude kontrolovat, že jsou všechny moduly stále aktivní. Všechny moduly musí v intervalu 10 vteřin posílat paket, kterým potvrdí, že je modul aktivní. Pokud Lektorské PC neobdrží ve stanoveném intervalu zprávu, že modul je aktivní, změní se v Lektorském PC danému modulu stav na neodpovídající (*Non-Responding*). Pokud během následujícího intervalu Lektorské PC opět neobdrží od modulu ve stavu *Non-Responding* paket s informací, že modul je aktivní, nastaví jeho stav na vypnutý (*Off*). Při obdržení paketu, kterým modul potvrdí, že je aktivní, se stav modulu přepne na zapnutý (*On*), pokud tomu tak již nebylo.

F20 Hlášení výpadku MQTT brokeru
Systém při zjištění výpadku MQTT brokeru upozorní uživatele na jeho výpadek.

F21 Kontrola překročení rychlostního limitu
Systém bude kontrolovat překročení rychlostního limitu při jízdě mimo režim úplného dozoru (*Full Supervision*). Při zjištění překročení rychlosti nastaví příslušnou proměnnou v paketu s informacemi o vlaku.

F22 Volba verze subsetů
Systém umožní na začátku simulace zvolit, ve které verzi subsetů proběhne simulace. Na výběr bude mezi verzemi: 2.3.0.d, 3.4.0, 3.6.0.

F23 Ovládání stavu návěstidel
Systém umožní na měnit stav návěstidel na trati. Informaci o změně stavu se bude posílat přes MQTT do modulu vizualizace.

F24 Kontrola desynchronizace modulů
Systém bude kontrolovat, že jednotlivé moduly, které hlásí svůj stav (F15), nejsou příliš desynchronizované. Kontrola bude prováděna na základě rozdílu v časech které moduly posílají. Pokud bude rozdíl mezi moduly nezanedbatelný, systém na tuto skutečnost upozorní.

F25 Validace uložených souborů
Systém bude při každém pokusu o úpravu, anebo přidání souborů bude jeho obsah zkontrolován, jestli je jeho obsah validní.

1.3.4 Nefunkční požadavky Lektorského PC

N1 Jazyk implementace
Systém bude implementován v jazyce C++ verze 17.

N2 Kompatibilita systému
Systém poběží na vzdáleném virtuálním počítači s operačním systémem Debian 10.

N3 Knihovna REST API

Pro tvorbu REST API bude použita C++ knihovna CrowCpp verze *v1.0+3 Hotfix*. Formát těla přijímaných a odesílaných HTTP zpráv bude JSON.

N4 Komunikace s moduly

Ke komunikaci s ostatními moduly ETCS bude využit síťový protokol MQTT a knihovna Mosquitto verze 1.5.7.

N5 Formát paketů

Pakety posílané skrze protokol MQTT broker budou ve formátu JSON. K převodu do formátu JSON se bude využívat knihovna Nlohmann-json3 verze 3.5.0. [11]

Kapitola 2

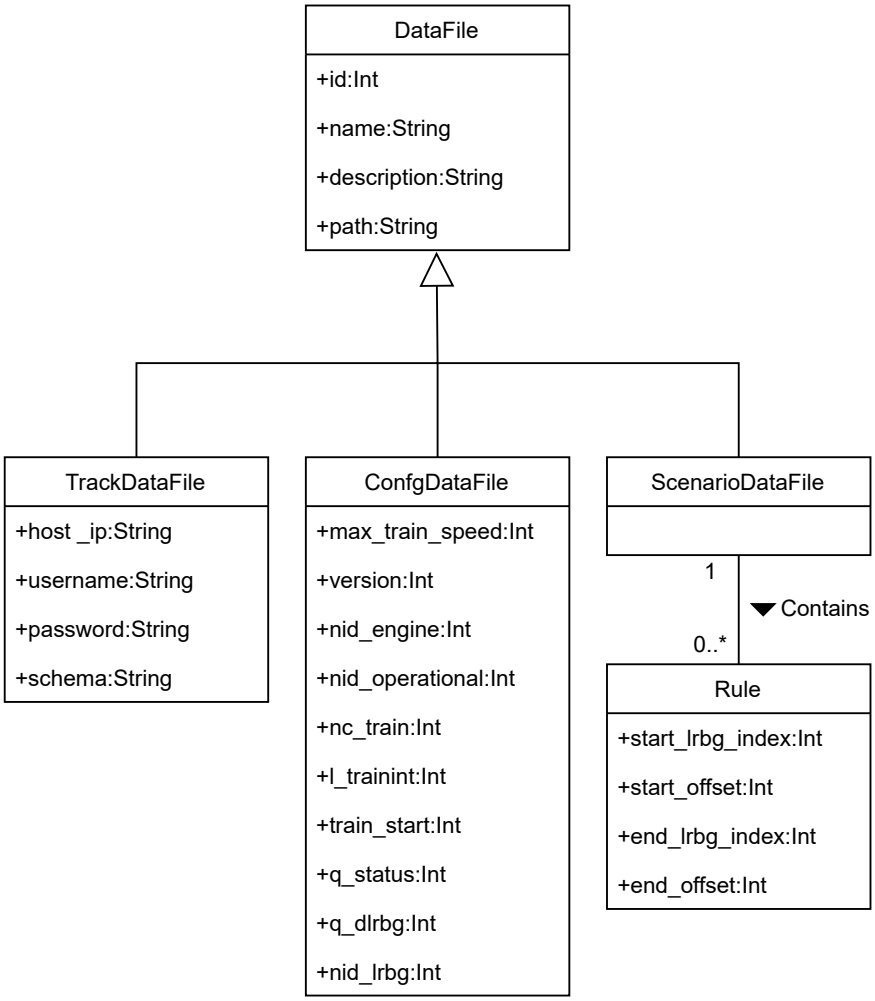
Návrh

V této kapitole se budu věnovat návrhu REST API Lektorského PC, který bude implementovat požadavky z kapitoly 1.3.3 a návrhy paketů pro komunikaci mezi moduly.

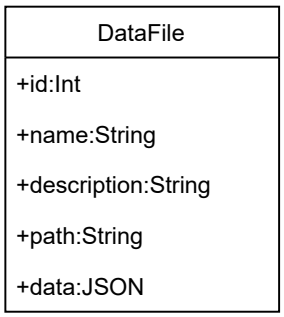
2.1 Doménový model

> *„Doménový model slouží k definování domény aplikace, tedy jednotlivých entit, jejich atributů a vztahů mezi entitami. Je nezávislý na platformně, tzn. není závislý na konkretním programovacím jazyce, architektuře aplikace, nebo použitých technologiích. Zpravidla bývá zachycen formou UML diagramu. Diagram slouží k pochopení a popisu základních datových struktur v aplikaci a slouží jako podklad pro další návrhy či implementace. Z doménového návrhu lze jednoduše vytvořit databázový model a také je využíván k definici základních tříd, pokud je implementace prováděna v objektové orientovaném programovacím paradigmatu.“* [12]

V případě Lektorského PC nastala situace, kdy se veškerá data, která rozlišují jednotlivé *DataFile* soubory vyskytují v JSON formátu a samotné Lektorské PC s nimi nepracuje. Tato skutečnost by se neměla v budoucnu měnit a lze tedy místo několika tříd mít pouze jednu třídu, která si bude udržovat informace, které jsou pro Lektorské PC užitečné (*id, path, name a description*) a instanci třídy `json`, ve které bude uložen zbytek dat. V našem případě se jedná o třídu `json` knihovny *nlohmann/json*, která umožňuje excelentní práci s daty uloženými v této třídě [11], takže ani o tuto funkcionalitu Lektorské PC nepřichází.



Obrázek 2.1: UML diagram doménového modelu

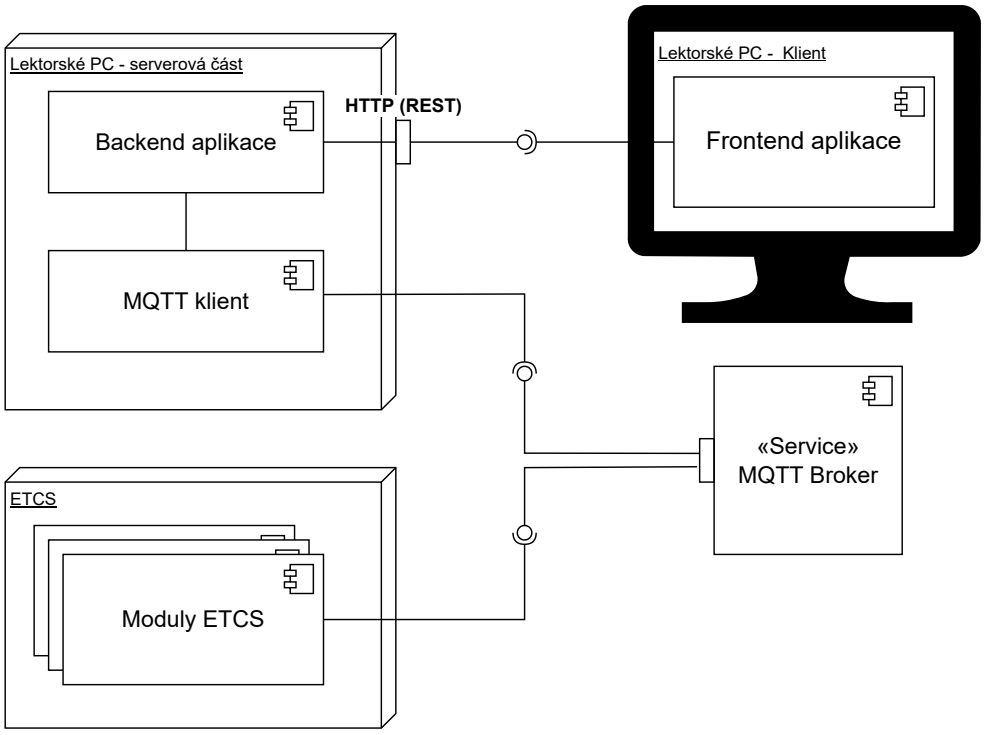


Obrázek 2.2: UML diagram použitého doménového modelu

2.2 Architektura

Lektorské PC je rozděleno na 2 hlavní části – serverovou část, která poběží na vzdáleném serveru a klientskou část, kterou se zabývá Jiří Sternwald ve své bakalářské práci [2]. Klientská část bude poskytovat uživatelsky přívětivé ovládání serverové části a bude se starat o případné zobrazování informací ze serverové části. Síťová architektura tedy bude klient-server, kde většinu funkcionality bude obstarávat serverová část, která se bude starat o práci s daty, soubory a o komunikaci s ostatními moduly.

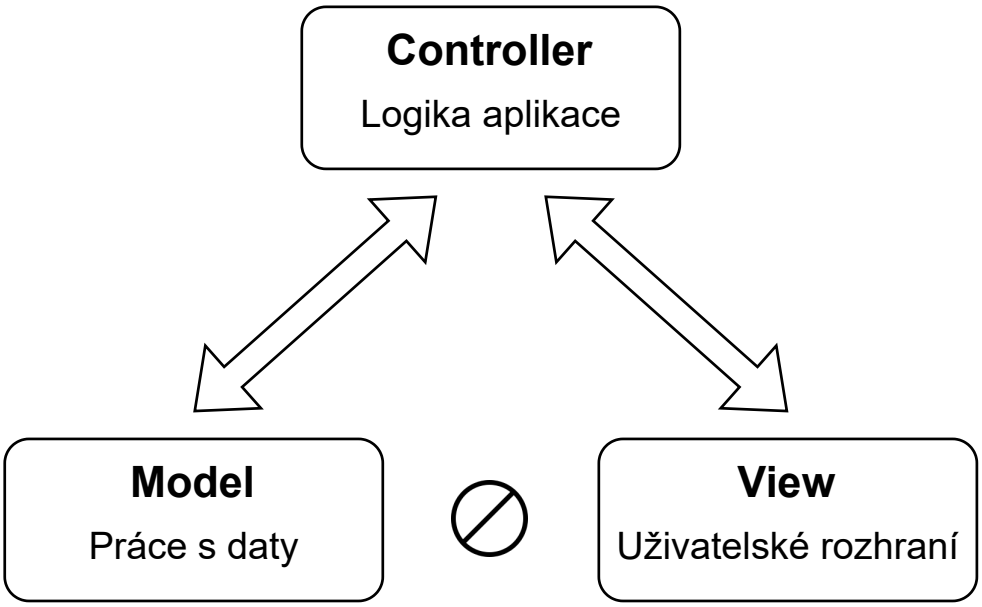
Mezi hlavní výhody této architektury patří například velmi snadné rozšíření na jiné platformy. Jelikož hlavní logická část aplikace (server) zůstává neměnná a společná pro všechny klientské části. V budoucnu by tedy stačilo vytvořit pouze novou klientskou část, která bude server ovládat a zobrazovat data. Mělo by tedy být relativně snadné rozšířit možnost ovládání Lektorského PC třeba na mobilní zařízení.



Obrázek 2.3: Diagram architektury aplikace

2.2.1 Serverová část

Při tvorbě serverové části jsem se inspiroval návrhovým vzorem MVC (Model-View-Controller). [13] Část *Controller* má v mé aplikaci na starost obsluhu REST API (tedy přijímání požadavků z klientské části, jejich zpracování a odeslání odpovědi) a komunikaci s ostatními moduly ETCS. Část *Model* se stará o načítání, ukládání a další práci s daty. Poslední částí *View* jsem se v případě Lektorského nezabýval, protože ta má na starost především zobrazování dat a tím se zabývá klientská část aplikace.



Obrázek 2.4: Diagram komunikace MVC v Lektorském PC

2.3 Technologie

Vzhledem k tomu, že na projektu budou velice pravděpodobně pokračovat studenti v předmětech *Softwarový týmový projekt 1* a *Softwarový týmový projekt 2* a vzhledem k tomu, že celý projekt ETCS je psán v jazyce C++, jsem se rozhodl pro Lektorské PC taky zvolit C++.

Je to jazyk, se kterým se setkal každý student na FIT ČVUT a nemělo by tedy být příliš náročné navázat na mou práci a serverovou část Lektorského PC dál rozvíjet.

Při hledání vhodné knihovny pro tvorbu REST API v C++ jsem narazil na srovnání těch nejpoužívanějších knihoven [14]. Srovnávaly se zde knihovny:

- C++ REST SDK
- Crow
- HttpLib
- Pistache
- Restbed
- Restinio

Na jedné z konzultací mi bylo doporučena knihovna *Pistachio*, která ani v tomto srovnání nevycházela špatně, musel jsem ji ale zavrhnout kvůli chybějící, nebo momentálně nedostupné dokumentaci. *C++ REST SDK* jsem zavrhl kvůli tomu, že se dále nerozvíjí a sami tvůrci na nové projekty doporučují využití jiné knihovny [15]. Knihovnu *Restbed* jsem vyloučil kvůli AGPL licenci [16], která by mohla v budoucnu představovat problém pro projekt. Z knihoven *Crow, HttpLib* a *Restinio* jsem nakonec vybral knihovnu *Crow*, respektive *CrowCpp*, která na *Crow* navazuje a dále ji rozvíjí. [17] Na *CrowCpp* mě zaujala jednoduchá instalace, velká komunitní podpora a přívětivé webové stránky s ukázkami, návody a dokumentací [18].

2.4 API

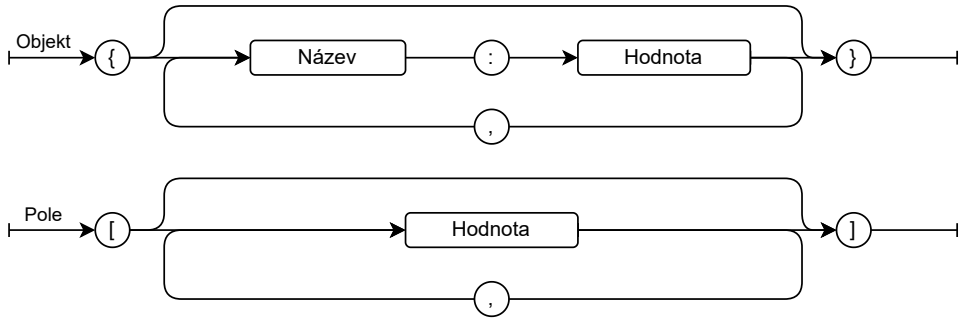
Jelikož API zajišťuje komunikaci mezi hlavními částmi aplikace, je návrh správného API dle mého názoru jedna z nejdůležitějších částí tvorby aplikace. Špatně navržené API může zapříčinit nepoužitelnost aplikace v budoucnu, a tedy nutnost aplikaci předělat.

Pro možnost jednoduchého budoucího rozšíření API a zkušenosti s tvorbou v této architektuře, jsem se rozhodl pro architektonický vzor REST a jako formát pro datovou komunikaci jsem zvolil datový formát JSON.

2.4.1 JSON

> „*JSON (JavaScript Object Notation) je odlehčený formát pro výměnu dat. Je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojově. Je založen na podmnožině Programovacího jazyka JavaScript, Standard ECMA-262 3rd Edition - December 1999. JSON je textový, na jazyce zcela nezávislý formát, využívající však konvence dobře známé programátorům jazyků rodiny C (C, C++, C#, Java, JavaScript, Perl, Python a dalších). Díky tomu je JSON pro výměnu dat opravdu ideálním jazykem.*“ [19]

JSON jsem zvolil kvůli jeho platformní nezávislosti, vysoké čitelnosti člověkem a díky tomu, že se již nyní v tomto formátu ukládají některá data, se kterými bude Lektorské PC pracovat. JSON má 2 datové struktury: pole hodnot a pár *název:hodnota*. Hodnotou může být string, číslo, boolean, null, objekt, nebo pole.



Obrázek 2.5: Grafické znázornění formátu JSON

2.4.2 REST API

> *„REST API je specifický, velmi oblíbený druh API. Vyniká zejména svou jednoduchostí a čitelností. API zajišťuje funkci „rozhraní“ pro předávání dat mezi dvěma či více aplikacemi. REST k tomuto obecnému mechanizmu dodává sadu doporučení a omezení, které když API dodržuje, tak se může nazývat REST API. Příkladem může být „bezstavovost“, při které by klientská aplikace měla vždy posílat veškeré nezbytné informace pro zpracování požadavku. REST API obsahuje dílčí endpointy, které jsou dostupné na definovaných URL adresách, které provolává klientská aplikace přes HTTP protokol. Pro tělo požadavku se u nás nejčastěji používá komunikační formát JSON.“* [20]

REST jsem zvolil kvůli jeho snadné možnosti úprav, což se u tohoto projektu hodí, jelikož v plánu je ho dále rozvíjet. Dalším důvodem, proč jsem zvolil architekturu REST oproti RPC [21], je má zkušenost s tvorbou REST API. Vytvořit kvalitní REST API jsem si již vyzkoušel v předmětu *Technologie Java* a mohl jsem tedy zužitkovat své znalosti a zkušenosti.

REST je datově orientovaný a jeho základem REST jsou zdroje. Za zdroj považujeme nějakou informaci, kterou identifikujeme pomocí URI. Na těchto zdrojích poté provádíme takzvané CRUD operace:

- **Create** Vytvoření nového zdroje. Využíváme HTTP metodu POST, nebo PUT.
- **Read** Získaní aktuálního stavu (dat) zdroje. HTTP metoda GET.
- **Update** Nastavení nového stavu (dat) zdroje. HTTP metoda PUT.
- **Delete** Odstranění zdroje. Slouží k tomu HTTP metoda DELETE.

2.4.3 HTTP

Stav, respektive data zdroje posíláme pomocí internetového protokolu HTTP. HTTP funguje na principu požadavku–odpovědi (*Request–Response*) [22].
V těle těchto zpráv se nachází data, v hlavičce jsou metadata, která tyto data popisují a dále specifikují zprávu. Na prvním řádku těchto zpráv se nachází informace o typu požadavku, případně typu odpovědi. Nejdůležitější pro požadavky na tomto řádku jsou především metody, které určují, co se má s daty v těle zprávy udělat. Mezi 4 základní metody patří:

- **GET**
  Tato metoda žádá zdroj o jeho stav (data).
- **POST**
  Tato metoda vytváří nový zdroj z dat obsažených v těle zprávy.
- **PUT**
  Tato metoda žádá zdroj o úpravu (náhradu) jeho stávajících dat, daty v těle zprávy. Pokud tento zdroj neexistuje vytvoří jej.
- **DELETE**
  Tato metoda slouží k odstranění zdroje.
  [23]

Pro odpovědi jsou na prvním řádku zase nejdůležitější stavové kódy, které nám pomáhají kategorizovat odpověď. Jsou to přesně definované tříciferné kódy, které spravuje organizace ICANN. Nejdůležitější číslovka je ta první, podle které řadíme odpovědi do 5 kategorií:

- **1xx - Informational**
  Informační kategorie. Server přijal naši žádost a pokračuje v jeho zpracování.
- **2xx - Success**
  Tato kategorie nás informuje, že požadavek byl přijat a zpracován v pořádku. Toto je očekávaná odpověď na většinu požadavků.
- **3xx - Redirection**
  Tato kategorie nás informuje, že pro dokončení požadavku je nutná nějaká akce z naší strany.

- **4xx - Client Error**
  Chybová kategorie, která nás informuje, že náš požadavek je špatný a nemohl být proveden.

- **5xx - Server Error**
  Chybová kategorie, která nás informuje, že i když náš požadavek je validní server ho nebyl schopen dokončit. Nastala tedy chyba někde na straně serveru. [24]

2.4.4 Návrh API

Zde se nachází můj návrh API pro Lektorské PC. Vždy je uvedena URI zdroje, metody, které jsou na tomto zdroji povoleny a krátký popis toho co volání této metody dělá.

/api/v1/start
GET Spuštění simulace

/api/v1/stop
GET Zastavení simulace

/api/v1/restart
GET Restartování simulace

/api/v1/tracks
GET Získání všech přístupových souborů k tratím
POST Vytvoření nového přístupového souboru tratě

/api/v1/tracks/{id}
GET Získání informací o trati se zvoleným id
PUT Úprava dat přístupového souboru tratě
DELETE Smazání přístupového souboru tratě

/api/v1/tracks/select/{id}
GET Výběr přístupového souboru pro simulaci

/api/v1/scenarios
GET Získání všech scénářů jízdy
POST Vytvoření nového scénáře jízdy

/api/v1/scenarios/{id}
GET Získání informací o scénáři se zvoleným id
PUT Úprava dat scénáře jízdy
DELETE Smazání scénáře jízdy

/api/v1/scenarios/select/{id}
GET Výběr scénáře jízdy pro simulaci

/api/v1/configs
GET Získání všech konfiguračních souborů
POST Vytvoření nového konfiguračního souboru

/api/v1/configs/{id}
GET Výběr konfiguračního souboru
PUT Úprava dat konfiguračního souboru
DELETE Smazání konfiguračního souboru

2.5 Pakety a soubory

Jelikož Lektorské PC nevyužívá k ukládání dat žádnou databázi, ale ukládá si potřebné soubory lokálně do souborů typu JSON, vznikla tím možnost si potřebné soubory ukládat v takovém formátu, v jakém se jako pakety budou posílat do ostatních modulů, respektive do klientské části Lektorského PC.

Níže se nachází návrhy těchto paketů/souborů. Pakety, které si Lektorské PC ukládá jsou ty, kterými Lektorské PC komunikuje s klientskou částí a jsou odesílány přímo v takovém tvaru, v jakém jsou uloženy. Mezi tyto pakety patří paket popisující trať a databázi, kde je trať uložena 2.6, paket obsahující scénář jízdy 2.8 a paket s počáteční konfigurací modulů ETCS simulace 2.10.

Pakety, které si Lektorské PC neukládá, jsou pakety, kterými Lektorské PC komunikuje s ostatními moduly. Tyto pakety jsou buďto už součástí uložených paketů a Lektorské PC si potřebnou část dokáže z uložených paketů vybrat, nebo se jedná o velice jednoduché pakety, jejichž obsah se často mění a bylo by zbytečné si je ukládat (například paket 2.1).

2.5.1 MQTT pakety

Tyto pakety se nikde neukládají a jsou využívány ke komunikaci s ostatními moduly ETCS skrze MQTT broker.

```
{
  "simulation": "start/stop/restart"
}
```

Paket 2.1: Spuštění/Zastavení/Restartování simulace – `MQTT`

```
{
  "module": "rbc/evc/dmi/train/...",
  "status": "running/stopped"
}
```

Paket 2.2: Očekávaná odpověď modulů na paket 2.1 – `MQTT`

```
{
  "host"     :"database IP address",
  "username" :"database credentials",
  "password" :"database credentials",
  "schema"   :"database schema"
}
```

Paket 2.3: Popis připojení k databázi s tratí – `MQTT`

```
{
  "rules": [
    {
      "start_lrbg_index" : "Index of the starting LRBG",
      "start_offset"     : "Offset from the startin LRBG",
      "end_lrbg_index"   : "Index of the ending LRBG",
      "end_offset"       : "Offset from the ending LRBG"
    },
    ...
  ]
}
```

Paket 2.4: Scénář jízdy – `MQTT`

```
{
  "max_train_speed": "<int> Max train speed in km/h",
  "version"        : "<int> Version of the ETCS language",
  "nid_engine"     : "<int> Onboard ETCS identity",
  "nid_operational": "<int> Train running number",
  "nc_train"       : "<int> International train category",
  "l_traininit"    : "<int> Train length in meters",
  "train_start"    : "<int> Strating position in abs. dist.",
  "q_status"       : "<int> LRBG validity",
  "q_dlrbg"        : "<int> LRBG direction",
  "d_lrbg"         : "<int> LRBG distance",
  "nid_lrbg"       : "<int> LRBG information"
}
```

Paket 2.5: Počáteční konfigurace – `MQTT`

2.5.2 Traťové pakety

Soubory do kterých se pakety popisující trať ukládají se pojmenovávají stylem `trackX.json`, kde `X` je nahrazeno za `id` této tratě.

```
{
  "id"          : 0,
  "name"        :"Name of the track",
  "description" :"Description of the track",
  "dtbconfig"   :{
    "host"        :"database IP address",
    "username"    :"database credentials",
    "password"    :"database credentials",
    "schema"      :"database schema"
  }
}
```

Paket 2.6: Popis tratě – `REST API`

```
{
  "tracks":[
    {
      "id"          : 0,
      "name"        :"Name of the first track",
      "description" :"Description of the first track",
      "dtbconfig"   :{
        "host"        :"database IP address",
        "username"    :"database credentials",
        "password"    :"database credentials",
        "schema"      :"database schema"
      }
    },
    {
      "id"          : 1,
      "name"        :"Name of the second track",
      "description" :"Description of the second track",
      "dtbconfig"   :{...}
    },
    ...
  ]
}
```

Paket 2.7: Seznam všech tratí – `REST API`

2.5.3 Pakety scénáře jízdy

Soubory do kterých se pakety popisující scénář simulace jízdy ukládají se pojmenovávají stylem `scenarioX.json`, kde `X` je nahrazeno za `id` tohoto scénáře.

```
{
  "id"          : 0,
  "name"        :"Name of the scenario",
  "description" :"Description of the scenario",
  "scenario"    :{
    "rules":[
      {
        "start_lrbg_index"  :"Index of the starting LRBG",
        "start_offset"      :"Offset from the startin LRBG",
        "end_lrbg_index"    :"Index of the ending LRBG",
        "end_offset"        :"Offset from the ending LRBG"
      },
      ...
    ]
  }
}
```

Paket 2.8: Popis scénáře jízdy – `REST API`

```
{
  "scenarios":[
    {
      "id"          : 0,
      "name"        :"Name of the first scenario",
      "description" :"Description of the first scenario",
      "scenario":{
        "rules":[...]
      }
    },
    {
      "id"          : 1,
      "name"        :"Name of the second scenario",
      "description" :"Description of the second scenario",
      "scenario":{...}
    },
    ...
  ]
}
```

Paket 2.9: Seznam všech scénářů jízd – `REST API`

2.5.4 Pakety počáteční konfigurace

Soubory do kterých se ukládají pakety s počáteční konfigurací se pojmenovávají stylem `configX.json`, kde `X` je nahrazeno za `id` této konfigurace.

```
{
  "id"          : 0,
  "name"        :"Name of the configuration",
  "description" :"Description of the configuration",
  "config":{
    "max_train_speed":"<int> Max train speed in km/h",
    "version"        :"<int> Version of the ETCS language",
    "nid_engine"     :"<int> Onboard ETCS identity",
    "nid_operational":"<int> Train running number",
    "nc_train"       :"<int> International train category",
    "l_traininit"    :"<int> Train length in meters",
    "train_start"    :"<int> Strating position in absolute distance",
    "q_status"       :"<int> LRBG validity",
    "q_dlrbg"        :"<int> LRBG direction",
    "d_lrbg"         :"<int> LRBG distance",
    "nid_lrbg"       :"<int> LRBG information"
  }
}
```

Paket 2.10: Výběr počáteční konfigurace – `REST API`

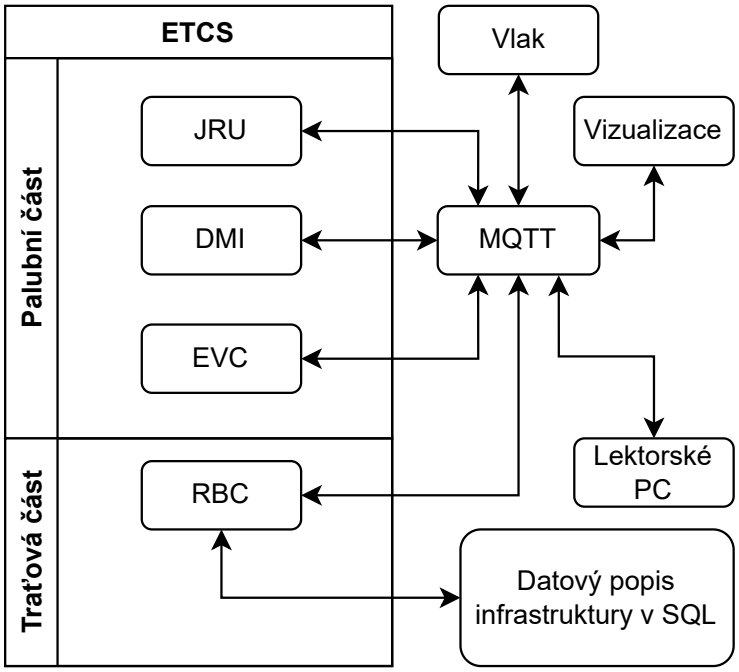
```
{
  "configs": [
    {
      "id"          :  0,
      "name"        : "Name of the first configuration",
      "description" : "Description of the first configuration",
      "config": {...}
    },
    {
      "id"          :  1,
      "name"        : "Name of the second configuration",
      "description" : "Description of the second configuration",
      "config": {...}
    },
    ...
  ]
}
```

Paket 2.11: Seznam všech počátečních konfigurací – `REST API`

Kapitola 3

Realizace

V této kapitole se věnuji implementaci Lektorského PC z kapitoly 2. Popisuji zde co všechno se mi podařilo implementovat, jaké nedostatky má současná implementace a jaké jsou vhodné rozšíření 3.1. Jelikož ze zkušenosti vím, jak těžké může být se zorientovat v cizím projektu, sepsal jsem zde také instalační 3.2 a programátorskou 3.3 příručku. V instalační příručce popíšu jak Lektorské PC a ostatní podpůrné programy zprovoznit a programátorská příručka by zase měla pomoct s orientací v projektu. Poté se věnuji testování Lektorského PC, přesněji jeho MQTT a REST API části 3.4.



Obrázek 3.1: Jednotlivé části projektu a jejich vzájemná komunikace

3.1 Implementace

V rámci mé bakalářské práce se mi podařilo implementovat veškeré minimální a optimální funkční požadavky, tak jak jsou vydefinované v kapitole 1.3.3. REST API Lektorského PC tedy zpracovává HTTP požadavky a na jejich základě informuje ostatní moduly ETCS skrze MQTT broker, nebo spravuje uložená data. Po přijetí a zpracování těchto požadavků odpovídá API klientské části správným stavovým kódem a případně i požadovaným tělem zprávy, tak jak je definované v sekci 2.4.4.

V tuto chvíli tedy Lektorské PC dokáže informovat ostatní moduly o změně stavu simulace, odeslat data s vybranou tratí, počáteční konfigurací a scénářem jízdy, spravovat úložiště těchto dat a korektně odpovídat na HTTP požadavky.

Nedostatky a další rozšíření

Hlavním nedostatkem aktuálního prototypu Lektorského PC, je absence implementace přijímání zpráv z MQTT brokeru. Tato funkcionalita, by mohla být využita například pro vylepšení aktuální implementace ovládání stavu simulace.

V aktuální podobě je odpovědí na tento požadavek pouze HTTP stavový kód, který nás informuje o tom, zdali se povedlo náš požadavek odeslat do MQTT brokeru. Vylepšená podoba odpovědi by mohla k tomuto stavovému kódu připojit i informaci, zdali moduly tuto zprávu úspěšně přijaly a jsou opravdu v požadovaném stavu. O tom by moduly informovaly Lektorské PC například pomocí paketu 2.2.

Vhodným rozšířením by také byla implementace CI/CD [25] a dalších funkčních požadavků ze sekce: Další rozšíření funkčních požadavků.

3.2 Instalační příručka

3.2.1 Instalace potřebných knihoven

Lektorské PC jsem vyvíjel na systému Windows 10 s WSL2 – Ubuntu 20.04. Popíšu tedy zde jak Lektorské PC zprovoznit na této platformě, ale postup by měl být stejný pro všechny systémy založené na Debianu. Jak už bylo zmíněno v kapitole 2.3, pro tvorbu REST API jsem zvolil knihovnu *CppCrow* a to konkrétně verzi *v1.0+3 Hotfix* , která byla v době implementace nejnovější vydaná verze. Instalace této knihovny je pro všechny operační systémy popsaná na webových stránkách `https://crowcpp.org/getting_started/setup/linux/`.

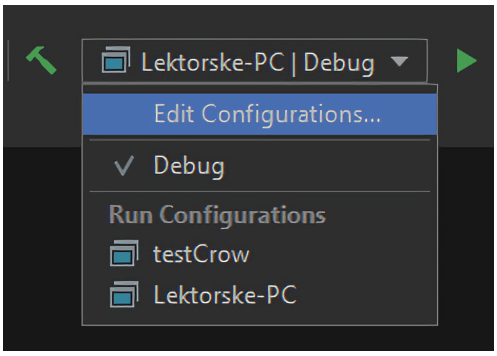
Pro operační systémy založené na Debianu se instalace skládá ze stažení vhodné verze `crow-vXXX.deb` souboru z GitHub repozitáře na adrese: `https://github.com/CrowCpp/Crow/releases` a nainstalování ho pomocí příkazu:

`sudo dpkg -i crow-vXXX.deb`. Knihovna CrowCpp vyžaduje pro správnou funkčnost knihovnu Boost verze alespoň 1.64 a knihovnu *phtread*, ty lze nainstalovat pomocí příkazů: `sudo apt-get install libboost-all-dev` a `sudo apt-get install libpthread-stubs0-dev`. Lektorské PC využívá k práci s JSON soubory knihovnu *Nlohmann-json3*, tu lze nainstalovat pomocí příkazu: `sudo apt-get install --yes nlohmann-json3-dev`.

3.2.2 Spuštění programu

Pro vývoj a spouštění programu jsem využíval IDE CLion od společnost JetBrains [26]. Používaný `CMakeList.txt` je součástí GitLab repozitáře. Jako alternativu jsem napsal jednoduchý skript, kterým lze program zkompilovat a spustit bez nutnosti otevírat IDE. Tento skript se pouští příkazem `./buildScript.sh`.

POZOR! V případě využití systému Windows a WSL2 je třeba v nastavení konfigurace nastavit `Working directory` na místo na disku, kde se projekt nachází. Toho docílíme kliknutím vpravo nahoře na konfiguraci→`Edit configuration` a vybráním správného místa na disku v kolonce `Working directory`.



Obrázek 3.2: Nastavení konfigurace v IDE CLion

3.3 Programátorská příručka

Vstupním bodem programu je `main.cpp`, zde se nejdříve nainicializují všechny knihovny, připojí *Route* makra a následně se spustí API. Nastavení portů, adres a ostatních konfiguračních proměnných se nachází v souboru `config.hpp`. Všechny *Route* makra 3.3 se nachází ve složce `Routes/` a jejich účelem je definování cest API a jejich povolených metod.

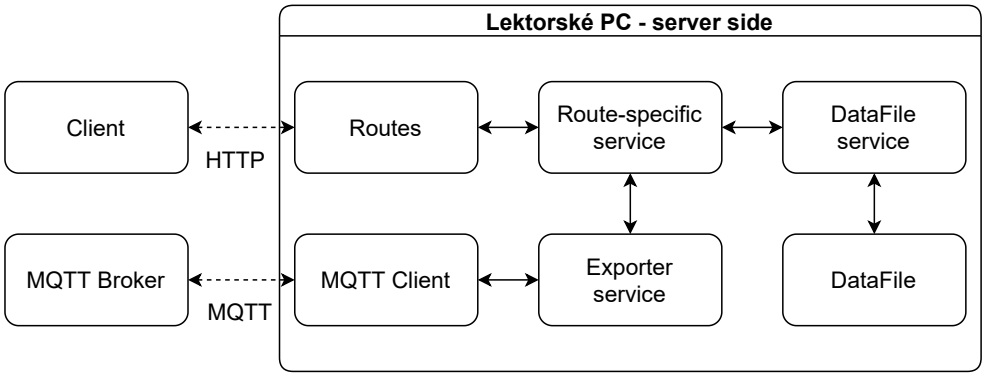
Tyto makra poté volají `Service` dané cesty, které se stará o zpracování požadavku a odeslání odpovědi. Tyto *„Route-services“* využívají k získání dat z uložených souborů `DataFileService`, která má na starost správu těchto

souborů a veškeré operaci s nimi. K sestavení odpovědí na HTTP požadavky a k odesílání zpráv ostatním modulům ETCS skrze MQTT broker využívají „`Route-services`“ třídu `ExporterService`.

```
CROW_ROUTE(app, "/api/v1/tracks/<int>").methods(
        crow::HTTPMethod::GET,
        crow::HTTPMethod::PUT,
        crow::HTTPMethod::DELETE
)
        ([&ts](const crow::request &req, int id) {
            if (req.method == crow::HTTPMethod::PUT) {
                return ts.updateTrack(id, req.body);
            } else if (req.method == crow::HTTPMethod::DELETE)
                return ts.deleteTrack(id);
            else
                return ts.getTrackByID(id, false);
        });
```

Obrázek 3.3: Ukázka `CROW_ROUTE` makra

Třída `ExporterService` umí sestavit HTTP odpověď s JSON obsahem a správně nastavenou HTTP hlavičkou. Tato třída využívá třídu `MqttClient` k odesílání zpráv do ostatních ETCS modulů a její poslední implementovanou funkcionalitou je spojení položek do JSON pole pomocí metody `mergeJSONs`.



Obrázek 3.4: Komunikace mezi vnitřními částmi Lektorského PC

3.4 Testování

Při testování jsem kladl důraz zejména na testování aplikace jako funkčního celku. Věnoval jsem se tedy především testování jeho výstupů. Jelikož ale nebyla dostupná funkční klientská část Lektorského PC a ani ETCS moduly nebyly připraveny pro testování komunikace, musel jsem obě tyto části nasimulovat. Pro simulaci klientské části jsem využil program Postman [27] a pro simulaci ETCS modulů jsem využil balíček `mosquitto-clients` [28].

3.4.1 Testování API

Jak jsem již zmínil výše, k simulování klientské části jsem využil program Postman. Ten nabízí přehledné uživatelské rozhraní, kterým se dá Lektorské PC ovládat a testovat.

Pro příkazy, které vyžadují přítomnost těla HTTP žádosti, nabízí Postman vložení tohoto těla v mnoha formátech. Pro manuální testování a ovládání Lektorského PC jsem využíval možnost přidání JSON těla, tak jak je demonstrováno na obrázku 3.6.

Aplikace Postman také nabízí možnost automatizace testování API. Tato možnost automaticky kontroluje HTTP odpověď serveru, včetně obsahu hlavičky a těla odpovědi. Této vlastnosti jsem využil a otestoval takto všechny minimální a optimální funkční požadavky z kapitoly 1.3.3. Tyto testy jsem exportoval a vložil k zdrojovému kódu Lektorského PC na ETCS GitLab.

3.4.2 Testování MQTT části

Kvůli nedodělaným částem ostatních modulů, jsem byl nucen simulovat přijímání MQTT paketů z Lektorského PC. K tomu jsem využil již dříve zmíněný balíček `mosquitto-clients` a veřejný on-line MQTT broker na webu `https://test.mosquitto.org/`, konkrétně port 1883.

Jelikož paketů, které se přes MQTT posílají je relativně málo (pouze 4 typy) a jsou přesně definované, zvolil jsem manuální kontrolu těchto zpráv. Testování se skládalo z několika částí:

- **Příprava Lektorského PC**
  Tato část se skládala z nastavení konfiguračních proměnných v souboru `config.hpp` na testovací hodnoty:

  ```
  #define BROKER_ADDRESS "test.mosquitto.org"
  #define MQTT_PORT 1883
  ```

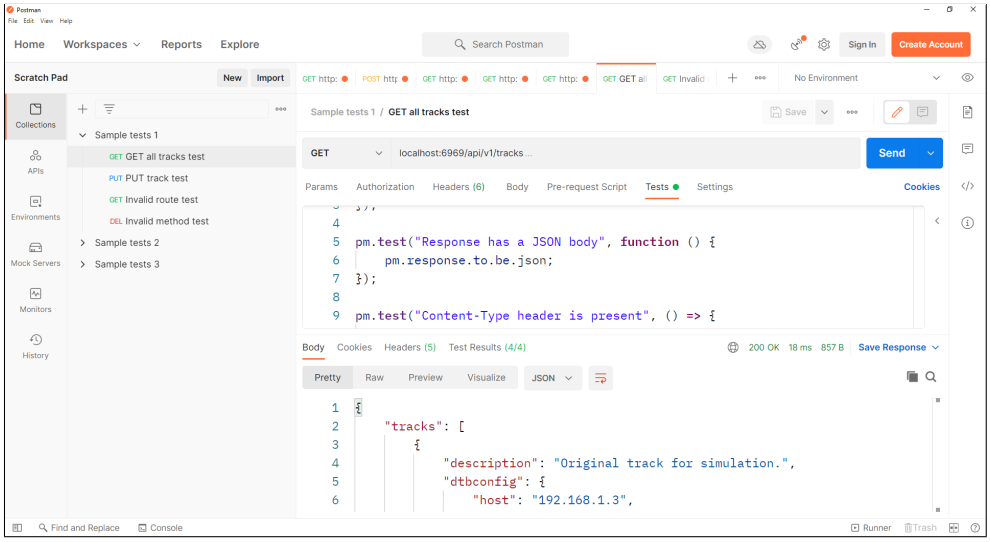
- **Spuštění mosquitto klienta**
  MQTT klient na příjem zpráv se pouští příkazem `mosquitto_sub`, na odesílání zpráv se používá příkaz `mosquitto_pub`. Celý příkaz, který jsem využíval pro testování byl:

  ```
  mosquitto_sub -h test.mosquitto.org -t "lpc/etcs/#"
  ```

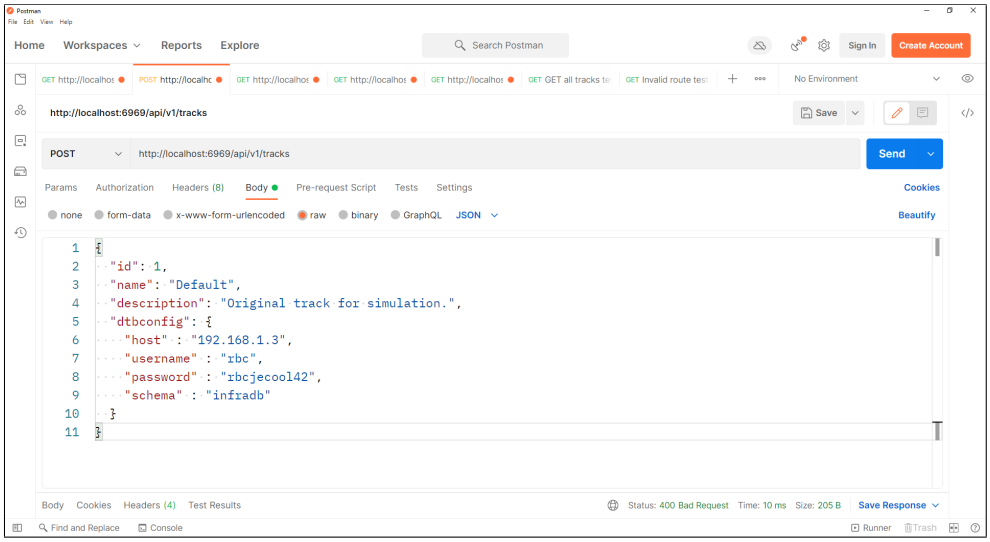
- **Odesílání zpráv skrze aplikaci Postman**
  V aplikaci Postman to znamenalo nastavení typu požadavku na GET a zavolání vhodné URI. Mezi URI, které způsobí odeslání zprávy přes MQTT se řadí například: `/api/v1/start` a `/api/v1/tracks/select/1` Při lokálním testování na portu 6969 tedy celá URI vypadala následovně:

  ```
  http://localhost:6969/api/v1/tracks/select/1
  ```

  V okně se spuštěným `mosquitto_sub` by se po odeslání tohoto požadavku měla objevit zpráva odpovídající paketu 2.3.



Obrázek 3.5: Uživatelské prostředí aplikace Postman [27]



Obrázek 3.6: Ukázka požadavku s JSON tělem v aplikaci Postman [27]

Závěr

V práci jsem se věnoval analýze projektu ETCS, návrhem, implementací a testováním nového modulu pro tento projekt a dokončením starého modulu RBC.

Na začátku práce v kapitole Analýza jsem se zaměřil na představení systému ETCS, jeho částí a analýzu současného stavu projektu ETCS a modulu RBC. Po konzultacích se zadavateli z Fakulty dopravní ČVUT jsem zde vymezil funkční a nefunkční požadavky pro modul RBC a Lektorské PC.

V kapitole Návrh jsem představil a popsal architekturu, kterou jsem zvolil pro implementaci Lektorského PC a technologie které jsem k tomu využil. Dále jsem zde popsal koncové body a operaci nad nimi které Lektorské PC zvládá. Na konci kapitoly jsem popsal pakety, které jsou využity pro komunikaci mezi moduly a Lektorským PC a pro komunikaci s klientskou části Lektorského PC

Kapitola Realizace se věnovala popisu implementace vybraných částí Lektorského PC. Dále jsem zde sepsal programátorskou příručku, která slouží k seznámení s vnitřním fungováním Lektorského PC a zároveň slouží jako návod na případné další rozšiřování funkcionalit Lektorského PC. Poté jsem zde také sepsal instalační příručku, ve které jsem popsal, jak zprovoznit a ovládat Lektorské PC. Na závěr jsem se věnoval testovaní Lektorského PC, konkrétně jeho výstupů.

Hlavní cíle práce považuji za splněné, modul RBC je zprovozněn a posílá povolení k jízdě, taktéž REST API Lektorského PC splňuje veškeré minimální a optimální funkční požadavky a je připraveno na další rozšíření.

Bibliografie

1. MINISTERSTVO DOPRAVY. *Plán moderního zabezpečení české železnice* [online]. 2021 [cit. 2022-04-26]. Dostupné z: `https://zdopravy.cz/wp-content/uploads/2021/08/etcsplan.pdf`.
2. STERNWALD, Jiří. *ETCS - RBC II*. Praha, 2023. Bakalářská práce. ČVUT v Praze, Fakulta informačních technologií.
3. KRASNENKOVA, Alina. *ETCS - Dynamic speed monitoring module for EVC*. Praha, 2022. Bakalářská práce. ČVUT v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství.
4. ROSHCHUPKINA, Daria. *ETCS - Modul pro komunikaci mezi EVC a RBC*. Praha, 2022. Bakalářská práce. ČVUT v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství.
5. STEJSKAL, Jan. *ETCS – DMI displej*. Praha, 2022. Bakalářská práce. ČVUT v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství.
6. FLAMMINI, Francesco; MARRONE, Stefano; IACONO, Mauro; MAZZOCCA, Nicola; VITTORINI, Valeria. A Multiformalism Modular Approach to ERTMS/ETCS Failure Modelling. *International Journal of Reliability, Quality and Safety Engineering*. 2014, roč. 21, s. 1450001–4. Dostupné z DOI: `10.1142/S0218539314500016`.
7. SPRÁVA ŽELEZNIC. *Co je ETCS* [online]. 2022 [cit. 2022-04-30]. Dostupné z: `https://www.spravazeleznic.cz/stavby-zakazky/modernizace/etcs/co-je-etcs`.
8. EUROPEAN UNION AGENCY FOR RAILWAYS. *SUBSET-026-2 3.6.0 System Requirements Specification Chapter 2 Basic System Description* [online]. 2016 [cit. 2022-05-04]. Dostupné z: `https://www.era.europa.eu/content/set-specifications-3-etcs-b3-r2-gsm-r-b1_en`.

9. STANLEY, P. *ETCS for Engineers*. Eurailpress, 2011. ISBN 9783777104164.
10. MACHÁČEK, Miloslav. *Vědeckotechnický sborník ČD*. Instalace ETCS v českých podmínkách. Praha: Generální ředitelství Českých drah, 2012. Č. 33. ISSN 1214-9047.
11. LOHMANN, Niels. *JSON for Modern C++* [online]. 2022 [cit. 2022-05-05]. Dostupné z: `https://github.com/nlohmann/json#json-as-first-class-data-type`.
12. GROFEK, Tomáš. *Aplikace pro chovatele psů*. Praha, 2022. Diplomová práce. ČVUT v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství.
13. DOHERTY, Erin. *MVC Architecture in 5 minutes: a tutorial for beginners* [online]. 2020 [cit. 2022-05-08]. Dostupné z: `https://www.educative.io/blog/mvc-tutorial`.
14. GUTEKSAN. *REST-CPP-benchmark* [online]. 2022 [cit. 2022-04-23]. Dostupné z: `https://github.com/guteksan/REST-CPP-benchmark`.
15. MICROSOFT. *Cpprestsdk* [online]. 2022 [cit. 2022-04-24]. Dostupné z: `https://github.com/microsoft/cpprestsdk#readme`.
16. FREE SOFTWARE FOUNDATION, Inc. *GNU Affero General Public License Version 3* [online]. 2007 [cit. 2022-05-08]. Dostupné z: `https://www.gnu.org/licenses/agpl-3.0.en.html`.
17. CROWCPP. *CrowCpp/Crow* [online]. 2022 [cit. 2022-04-24]. Dostupné z: `https://github.com/CrowCpp/Crow`.
18. CROWCPP. *CrowCpp* [online]. 2022 [cit. 2022-04-26]. Dostupné z: `https://crowcpp.org/guides/app/`.
19. *Úvod do JSON* [online]. [B.r.] [cit. 2022-04-23]. Dostupné z: `https://www.json.org/json-cz.html`.
20. DAMI. *REST API* [online]. [B.r.] [cit. 2022-04-24]. Dostupné z: `https://www.damidev.com/slovnik/rest-api`.
21. CORPORATION, IBM. *Remote Procedure Call* [online]. 2022 [cit. 2022-05-11]. Dostupné z: `https://www.ibm.com/docs/en/aix/7.1?topic=concepts-remote-procedure-call`.
22. CORPORATION, IBM. *The HTTP protocol* [online]. 2021 [cit. 2022-05-08]. Dostupné z: `https://www.ibm.com/docs/en/cics-ts/5.2?topic=concepts-http-protocol`.
23. MFUJI09; J9T; AICHBAUER; CAMILLE347; FSCHOLZ; OBIORA22; GAVDADDY27; MIKE-LANG; TEOLI. *HTTP request methods* [online]. 2019 [cit. 2022-05-05]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods`.

24. IANA. *Hypertext Transfer Protocol (HTTP) Status Code Registry* [online]. 2018 [cit. 2022-04-26]. Dostupné z: `https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml`.

25. INC., GitLab. *GitLab CI/CD | GitLab*. 2022. Dostupné také z: `https://docs.gitlab.com/ee/ci/`.

26. JETBRAINS S.R.O. *CLion* [soft.]. 2022. Ver. 2022.1 [cit. 2022-04-15]. Dostupné z: `https://www.jetbrains.com/clion/`.

27. POSTMAN, INC. *Postman* [soft.]. 2022. Ver. 9.16.0 [cit. 2022-04-15]. Dostupné z: `https://www.postman.com/downloads/`.

28. ECLIPSE FOUNDATION, INC. *mosquitto-clients* [soft.]. 2022. 1.5.7-1+deb10u1 [cit. 2022-05-11]. Dostupné z: `https://packages.debian.org/buster/mosquitto-clients`.

Příloha A

Seznam použitých zkratek

API application programming interface

CRUD Create Read Update Delete

ČR Česká Republika

ČVUT České vysoké učení technické v Praze

DMI Driver Machine Interface

ETCS European Train Control System

EVC European Vital Computer

FD Fakulta dopravní

FIT Fakulta informačních technologií

HTTP Hypertext Transfer Protocol

ICANN Internet Corporation for Assigned Names and Numbers

IDE integrated development environment

IP Internet Protocol

JRU Juridical Recording Unit

JSON JavaScript Object Notation

MA Movement Authority

MMI Man Machine Interface

MVC Model View Controller

RBC Radio Block Centre

REST Representational State Transfer

URI Uniform Resource Identifier

Příloha B

Obsah přiloženého CD

- `src.zip` ..... zdrojová forma práce ve formátu LaTeX
- `thesis.pdf` ..... text práce ve formátu PDF