Středoškolská technika 2016

Setkání a prezentace prací středoškolských studentů na ČVUT

IoT: ŘÍZENÍ MODELOVÉ ŽELEZNICE

Jakub Topič

Střední průmyslová škola elektrotechnická
V Úžlabině 320, Praha 9

Poděkování

Rád bych poděkoval vedoucí své práce Ing. Ivě Tomáškové za její ochotu a cenné rady, které mi během realizace této maturitní práce poskytla. Dále bych chtěl poděkovat svému strýci Dr. Ing. Karlu Nevšímalovi za to, že jsem mohl na jeho modelové železnici implementovat svůj řídicí systém. Poděkování také patří autorům svobodného softwaru a hardwaru, na kterém tato práce stojí.

Abstrakt

Tato práce se zabývá návrhem a implementací systému pro řízení analogového modelu železnice. Součástí tohoto systému je webová aplikace, která v reálném čase vizualizuje celkový stav kolejiště a pozici všech vlaků. Webová aplikace umožňuje vzdálené interaktivní ovládání návěstidel a výhybek, ale také poskytuje možnost vlastního skriptování a spouštění sekvencí povelů.

Důraz je mimo jiné kladen na analýzu a zdůvodnění výběru technologií a návrh způsobu komunikace.

Klíčová slova: Internet věcí, Arduino, WebSocket, JavaScript, Node.js, HTML5

Abstract

This graduation work is focused on design and implementation of the analog model railway control system. Part of this system is a web application that visualizes the overall situation of the railway and the location of all trains in real time. The web application enables remote interactive control of signals and turnouts and also provides the ability of scripting and executing custom sequences of commands.

Emphasis is laid on the analysis justifying decisions made for the choice of technology and design of communication methods.

Keywords: Internet of Things, Arduino, WebSocket, JavaScript, Node.js, HTML5

Obsah

Úvod

„The Internet of Things has the potential to change the world, just as the Internet did. Maybe even more so. “

[5, Kevin Ashton]

Internet věcí je nově vznikající fenomén, o němž poprvé hovořil Kevin Ashton v roce 1999 a který umožňuje propojení fyzických předmětů, vozidel, budov nebo měst přes celosvětovou komunikační síť Internet nebo obecně přes komunikační sítě. Tím vznikají nové možnosti např.: zpracování dat z velkého množství environmentálních senzorů, vzdálené ovládání různých objektů (klimatizace, topení, …), budování nových infrastruktur ve městech (chytrá města).

Trh v současnosti nabízí různé nositelné technologie a širokou škálu zařízení pro chytré domácnosti (dálkově spínané žárovky, zásuvky apod.). Mým záměrem však bylo vytvořit zařízení či systém, který bude zcela jedinečný. Při rozhovoru se strýcem Karlem Nevšímalem mě napadlo, že by se právě jeho analogová modelová železnice mohla stát předmětem mého zájmu. Po vzájemné dohodě jsem se rozhodl vyvinout komplexní systém, jehož základní funkcí bude vzdálené řízení železnice přes Internet.

V jednotlivých částech protokolu postupně popíši veškeré použité technologie, realizaci celého projektu a nakonec prostředí mnou vytvořené webové aplikace.

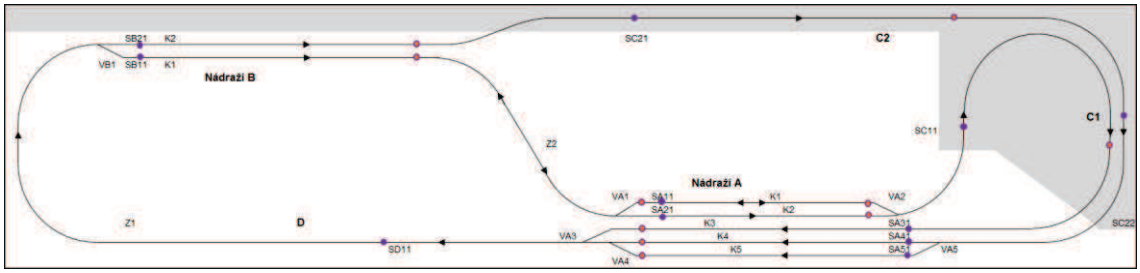
1. KAPITOLA

Specifikace implementace

V této kapitole popisuji cíle a požadavky práce, které jsem si na začátku vývoje stanovil.

1.1 Výchozí stav

Modelovou železnici si strýc Karel po asi 30 letech uložení v krabicích zprovoznil na ploše 430×110 cm. Jelikož se ukázalo, že technika z té doby funguje, zavrhl modernizaci ovládacích prvků či dokonce digitalizaci, zejména z finančních důvodů. Upravil pouze spínací koleje a byla použita moderní relé k vybavování výhybek, návěstidel a přestavníků. Tím se zamezilo jiskření na kontaktech spínacích kolejí a velkým proudům na kolejích. Další úpravu, kterou Karel vnesl do zapojení prvků železnice, je usměrnění vybavovacího napětí 16V diodovým můstkem. Toto napětí ovládá výhybky, přestavníky a mechanická návěstidla. Usměrnění umožňuje využít diodovou logiku, kdy různé spínací koleje mohou ovládat jeden prvek bez vlivu na další prvky, které jsou řízeny danou spínací kolejí. Tak je umožněno povely sdružovat a minimalizovat. Např. povel *VOLNO* pro určitou kolej vyvolá v témže okamžiku přestavení návěsti na zelenou, přestavení výjezdových výhybek z nádraží a může ještě přestavit i vjezdovou výhybku na tuto kolej. Povely jsou vlastně minimalizovány na *VOLNO* pro každou kolej, kde se diodová logika postará o to, aby výhybky byly nastavené správně. Poté jsou zde povely *STŮJ*, které nastaví návěst na červenou pro souběžné koleje v daném nádraží či tunelu a povely pro samostatné výhybky, které řeší vjezd na příslušnou kolej do nádraží. Povel *STŮJ* si vlaky spouštějí samy po opuštění nádraží při vjezdu na příslušnou spínací kolej.



Obrázek 1 – Nákres topologie modelové železnice

Obsluha se tedy omezuje na nastavování vjezdových výhybek a vybavování vlaků. Povely jsou vyvedeny na tlačítka – klávesnici modelové železnice. Zároveň jsou přivedeny

na konektor CANON, kde přizemněním daného povelu se tento vybaví. Na panel jsou vyvedeny i reléové kontakty spínacích kolejí. Tím je umožněno vytvářet „programy“ pouhým propojením pinů konektoru, kdy např. při příjezdu vlaku do nádraží vybaví tento druhý vlak na druhém nádraží. Pomocí takovýchto konektorů s definovanými propojkami lze sestavit několik programů (každý program = samostatný konektor), jejichž limitací je však časová souslednost jedoucích vlaků a spínací koleje, které přitom přejíždějí. Rovněž chybí jakékoliv časové zpoždění, při přejezdu spínací koleje se povel ihned vybaví.

Analogová modelová železnice, pro kterou jsem vyvíjel tento řídicí systém, již disponovala řídicím panelem a jednoduchou diodovou logikou. Ta zajišťuje, že se při vybavení vlaku z dané koleje zároveň nastaví i příslušné výjezdové výhybky z nádraží a zároveň i vjezdové výhybky na tutéž kolej. Tyto akce proto nemusí má řídicí jednotka provádět.

Vstupy všech důležitých povelů *VOLNO* pro jednotlivé koleje v nádražích (viz K*x* v tabulce 1), pro jednotlivé koleje v tunelech (C*x*) a pro nastavování stavu výhybek (VK*x*) jsou vyvedeny do 37pinového CANON konektoru na řídicím panelu. Do konektoru jsou vyvedeny i spínací koleje (S*x*), které při projíždění vlaku uzemňují daný pin konektoru. Pro nastavení návěsti *stůj* na všech návěstidlech v daném nádraží slouží piny označené *STŮJ A*, *STŮJ B* a *STŮJ C*. K vybavení povelu dojde přivedením logické nuly (GND) na příslušný pin konektoru a řídicí jednotka musí zajistit, aby tento vybavovací impuls trval určitou dobu (v řádu stovek milisekund).

Tabulka 1 – Schéma CANON konektoru

19 1
37 20

Pin	Funkce
1	GND
2	VK5
3	K1/2
4	K4/5
5	K1 (B)
6	K2 (B)
7	VK1 (B)
8	VK2 (B)
9	K1
10	K2
11	K3
12	K4
13	K5
14	K1zpět
15	C1
16	C2
17	STŮJ C
18	Spínač
19	Spínač

Pin	Funkce
20	SD11
21	SA51
22	VK1
23	SA41
24	VK2
25	SA31
26	Spínač2
27	SA21
28	Spínač2
29	SA11
30	VK4
31	SC22
32	SC21
33	SC11
34	SB21
35	SB11
36	STŮJ A
37	STŮJ B

Mezi požadavky řídicího systému, které jsem si zvolil, patří:

- Řídicí systém umožní vzdálené ovládání a monitorování kolejiště pomocí webové aplikace. Další funkcí bude možnost vytvářet vlastní skripty, tedy posloupnosti jednoduchých příkazů, pomocí kterých může řídicí jednotka automatizovaně ovládat kolejiště.

- Řídicí jednotka vyhodnocuje impulsy ze spínacích kolejí, které identifikují vjezd vlaku do dané koleje. Obsazenost úseků si program udržuje na základě vjezdu vlaku na danou kolej a na základě vypravení vlaku z dané koleje.

- Řídicí jednotka bude mít implementovaný bezpečnostní systém, který zamezí jakékoliv kolizi vlaků. Na základě tabulky obsazených úseků bude mít tedy jednotka možnost zakázat určité povely a povolit jen ty bezpečné.

- Uživatel bude mít možnost povolit či zakázat tzv. střídání vlaků, jehož důsledkem je to, že se řídicí jednotka snaží udržet nákladní vlaky na vnějším okruhu a osobní vlaky na vnitřním.

- I v případě nedostupného připojení k serveru by měla být řídicí jednotka schopna spouštět skripty ovládání kolejiště.

- Díky tomu, že má program informace o tom, které koleje jsou obsazené a které volné, může si náhodně vybrat, jaký vlak vypraví do volného úseku. To lze implementovat jako jeden z příkazů skriptu. Stále ale bude platit volba, zda se mají či nemají kombinovat osobní s nákladními vlaky.

- Řídicí jednotka si též musí pamatovat poslední pozici vlaků. Tlačítkem *DEFAULT* si nastaví obvyklou pozici vlaků pro případ, že bylo dříve použito manuální řízení.

- Řídicí jednotka by měla disponovat displejem, který by umožňoval volbu skriptu ze seznamu a sledování běhu skriptu.

- Skriptování by také mělo do řízení kolejiště přinést zavedení možnosti časové prodlevy pro získání větší modelovosti.

- Při zapnutí řídicí jednotky do napájení je třeba zajistit, aby za žádných okolností nedošlo k vyslání povelu *VOLNO* na jakoukoliv kolej.

2. KAPITOLA

Zvolené technologie

V této kapitole popisuji veškeré fyzické komponenty a softwarové technologie, které jsem v projektu použil.

2.1 Mikrokontrolér pro řídicí jednotku

Pro základ řídicí jednotky bylo potřeba vybrat vhodnou mikrokontrolérovou desku, která by vyhovovala všem požadavkům pro implementaci. Nejdůležitější vlastností byl dostatečný počet GPIO pinů, pomocí kterých lze číst a generovat impulsy pro ovládání modelové železnice. Bez použití posuvných registrů nebo podobného způsobu rozšíření počtu vstupů a výstupů, kterého jsem se chtěl vyvarovat, bylo potřeba celkem **19 výstupních digitálních pinů**, z toho 10 určeno pro vybavování kolejí, 6 pro nastavování stavu výhybek a 3 pro povel *STŮJ* a dále **9 vstupních digitálních pinů** pro čtení stavů spínacích kolejí. Další GPIO piny bylo potřeba rezervovat pro připojení ostatních periferií.

Mezi dalšími potřebnými vlastnostmi byla spolehlivá konektivita k Internetu a dostatek flash paměti pro program a operační paměti pro běh programu. Jako nejvhodnější z dostupných vývojových kitů se mi jevilo Arduino MEGA v kombinaci s rozšiřujícím Ethernet shieldem, který slouží jako síťové rozhraní. Zmíněný Ethernet shield je založen na ethernetovém čipu Wiznet W5100, který poskytuje TCP/IP stack a umožňuje až čtyři souběžná spojení. Výhodou shieldu je také přítomný slot na microSD kartu, která je přístupná díky dodávané knihovně podporující souborové systémy FAT16 a FAT32.

Arduino[1] je open-sourcová prototypovací platforma s místem vzniku v Itálii. Platforma se skládá jednak z hardwaru čítajícího množství mikrokontrolérových desek, jednak ze softwaru v podobě grafického vývojového prostředí a programovacího jazyka s C/C++ syntaxí založeného na jazyce Wiring.

Arduino Mega je mikrokontrolérová deska založená na 8bitovém mikročipu Atmel ATmega2560 s Harvardskou architekturou a instrukční sadou typu RISC. Mikrokontrolér

[1] <https://www.arduino.cc/>

pracuje na frekvenci 16 MHz a pro kód a data programu slouží jeho vnitřní flash paměť o velikosti 256 kB. Při běhu programu jsou statická data, zásobník a halda uloženy ve volatilní paměti SRAM o velikosti 8 kB. Pro dlouhodobé uchování dat je k dispozici 4kB nevolatilní EEPROM paměť, kterou jsem však nevyužil a k tomuto účelu používám externí SD kartu o mnohem větší kapacitě. Deska disponuje 54 digitálními I/O piny a 16 analogovými vstupními piny.

2.2 Server

K hostování webové aplikace by bylo možné použít webhosting podporující běh node.js nebo si pronajmout virtuální privátní server (VPS). Pro nasazení projektu jsem však použil vlastní malý dedikovaný server, který provozuji v našem bytě v obýváku a používám jej k různým účelům, mimo jiné k provozování vlastních projektů.

2.3 Technologie na straně serveru

2.3.1 GNU/Linux

Linux nebo linuxová distribuce je označení pro operační systém založený na Linuxovém jádru. Jednou z nejznámějších linuxových distribucí je Ubuntu, kterou jsem i já použil jako operační systém mého serveru. Ubuntu je vyvíjeno komunitou za podpory společnosti Canonical pro osobní počítače, notebooky a servery (bez grafického prostředí). Jeho název pochází ze zuluštiny (jazyk používaný Zuly v jižní Africe) a překládá se jako lidskost nebo se jako jeho význam uvádí *„jsem tím, čím jsem, díky lidem kolem mne“*. Verze Ubuntu vycházejí každých 6 měsíců a každé dva roky je zveřejněna LTS verze. Na mém serveru je nyní nasazena serverová edice Ubuntu ve verzi 14.04 LTS, tedy ve verzi s prodlouženou podporou. Ta trvá 5 let.

2.3.2 Node.js

Node.js je multiplatformní open-source běhové prostředí pro vývoj škálovatelných webových či síťových aplikací v JavaScriptu, které jsou určeny pro běh na straně serveru. Node.js používá událostmi řízený (event-driven) model. Díky asynchronnímu zpracování úloh tedy nedochází k blokování celého programu během I/O operací, což má za důsledek minimalizaci využití prostředků a maximalizaci výkonu. Běhové prostředí interpretuje JavaScript pomocí enginu V8, vyvinutého společností Google pro webový prohlížeč *Google Chrome* a *Chromium*.

2.3.2.1 Použité moduly

Funkci knihoven plní v Node.js tzv. moduly, které obsluhují souborový systém, síťové služby, kryptografické funkce a umožňují využít další služby operačního systému. Zde je výčet modulů, které jsem v projektu použil:

- `crypto` – Tento modul poskytuje kryptografické funkce openSSL, které zahrnují výpočty hashů, autentifikaci pomocí HMAC, šifrování a další. Ve své aplikaci tento modul používám pro generování UUID.
- `fs` – Implementace přístupu k souborovému systému pomocí standardních POSIXových funkcí.
- `net` – Modul, který umožňuje vytvořit síťové klienty i servery komunikující pomocí protokolu TCP.
- `ws` – Tento modul umožňuje snadnou implementaci protokolu WebSocket. Jedná se o jednu z nejrychlejších WebSocket knihoven pro node.js.

2.3.3 PM2

Process Manager 2 je správce procesů sloužící pro produkční nasazení Node.js aplikací. PM2 umožňuje neustálý dlouhodobý běh aplikací a v případě pádu aplikace nebo restartu serveru dokáže běh obnovit a tím částečně zamezit možným výpadkům. Node.js pracuje standardně pouze v jednom vlákně, tudíž nedokáže využít schopnosti vícejádrových systémů, PM2 ovšem dokáže zátěž rozložit mezi více instancí jedné aplikace bez nutnosti přizpůsobení kódu aplikace. Další výhodou je monitorování využití systémových prostředků (operační paměť a čas na procesoru) a výpis logů jednotlivých procesů v reálném čase.

Kód 2.1: Ukázka vypsání seznamu všech procesů spuštěných pomocí PM2.

```
$ pm2 status
┌─────────────┬────┬──────┬───────┬────────┬─────────┬────────┬─────────────┬──────────┐
│ App name    │ id │ mode │ pid   │ status │ restart │ uptime │ memory      │ watching │
├─────────────┼────┼──────┼───────┼────────┼─────────┼────────┼─────────────┼──────────┤
│ zeleznice   │ 0  │ fork │ 12743 │ online │ 0       │ 37D    │ 22.328 MB   │ disabled │
└─────────────┴────┴──────┴───────┴────────┴─────────┴────────┴─────────────┴──────────┘
```

2.3.4 Apache2

Při použití Node.js není v žádném případě nezbytně nutné použít oddělený webový server, ale vzhledem k tomu, že je veškerý zdrojový kód klientské části webové aplikace statický, rozhodl jsem se pro vyřizování žádostí o tyto statické soubory použít jiný webový server. To má za výhodu i lepší možnost nastavení kešování a lepší zabezpečení TLS zakázáním slabých šifer, povolením HSTS apod.

Často se v kombinaci s node.js používá vysoce výkonný, svobodný, multiplatformní webový server nginx. Na svém serveru jsem však měl již nasazený webový server Apache, který jsem použil i pro tento projekt. V budoucnu možná zvážím přechod na nginx, ale nyní je pro mě Apache kvůli svým výhodám při poskytování dynamického obsahu v jiných mých projektech výhodnější.

Apache HTTP Server je open-source webový server vyvíjený v jazyce C a C++ pro množství platforem (mj. GNU/Linux, FreeBSD, Solaris, OS X, Windows). Jedná se o celosvětově nejpoužívanější webový server, což dokládá průzkum serveru Netcraft ze září roku 2015, kdy byl Apache nasazen na 34.96 % serverů.

Apache poskytuje podporu rozsáhlého množství funkcí a programovacích jazyků (PHP, Python, Perl nebo Ruby) a také podporu kryptografického protokolu TLS pro šifrovanou komunikaci. Pro snížení velikosti přenášených dat používá kompresi gzip.

2.4 Technologie ve webové aplikaci

2.4.1 HTML5

Hypertext Markup Language je standardní značkovací jazyk používaný pro tvorbu webových stránek a webových aplikací. Ty je možné navzájem propojit hypertextovými odkazy. Jazyk je vyvíjen hlavní standardizační organizací pro WWW konsorciem W3C a organizací WHATWG.

2.4.2 CSS3

CSS3 (kaskádové styly) je jazyk pro popis stylu dokumentů napsaných ve značkovacích jazycích HTML, XHTML, SVG atd. Vývoj CSS je zajišťován konsorciem W3C. Cílem jazyka CSS3 je oddělení struktury a obsahu dokumentu od jeho stylu (vzhledu). Oddělení zabraňuje přebytečným redundancím, protože stejné vlastnosti a styly může sdílet více různých elementů, tak i celé dokumenty.

2.4.3 JavaScript

JavaScript je vysokoúrovňový, dynamický, interpretovaný programovací jazyk používaný zejména ve webových aplikacích. Jedná se o multiparadigmatický jazyk, který podporuje objektově orientované, imperativní a funkcionální programovací styly. Vzhledem k tomu, že žádné paradigma neumožňuje nejefektivnější řešení všech úloh, nabízí JavaScript programátorům vždy pro daný úkol volbu nejvhodnějšího programovacího stylu.

2.4.4 WebSocket

WebSocket[2] je protokol navržený zejména pro použití ve webových aplikacích, který umožňuje vytvořit full-duplexní spojení mezi klientem a serverem, takže mohou obě strany současně přijímat a odesílat data. Díky trvalému spojení může server odeslat klientu data, aniž by jím byla vyžádána. To je velký rozdíl oproti v současnosti běžnému návrhu interaktivních webových aplikací, kdy se za použití například technologie AJAX nebo AJAJ musí klientská strana v pravidelných intervalech dotazovat serveru, zda nedošlo k nějakým změnám.

Protokol využívá transportního protokolu TCP (standardně na portu 80, či 443 při použití zabezpečené varianty) a jeho handshake[3] při navazování spojení se podobá činnosti protokolu HTTP s tím rozdílem, že žádost klienta s metodou *GET* navíc obsahuje hlavičku *Upgrade: websocket*, hlavičku obsahující verzi protokolu *a* hlavičku, která zabraňuje možnému opětovnému odeslání dat z předchozího spojení v případě, že je mezi klientem a serverem cachovací proxy[4]. Tato skutečnost umožňuje serveru obsluhovat zároveň HTTP a WebSocket spojení na stejném portu. Jakmile je spojení navázáno, přepne se komunikace na obousměrný binární protokol, který již HTTP protokolu neodpovídá.

---

[2] <https://tools.ietf.org/html/rfc6455>

[3] Handshake, neboli „potřesení rukou“, je vyjednávání dvou entit, které má za účel nastavit parametry komunikačního kanálu ještě před zahájením normální komunikace.

[4] Cachovací proxy server je prostředník mezi klientem a serverem, jehož úkolem je urychlování vyřizování žádostí načtením odpovědi uložené z předchozí žádosti odeslané tím samým nebo jiným klientem.

Ve své aplikaci jsem implicitně použil šifrovanou variantu WebSocket spojení za použití kryptografického protokolu TLS. Webový server je z hlediska bezpečnosti nakonfigurován tak, aby akceptoval pouze TLS protokoly a neumožnil spojení přes zastaralé a zranitelné SSL. Šifrování spojení také zvyšuje šanci úspěšného spojení v prostředích obsahující transparentní nebo jiné HTTP proxy servery.

Protokol WebSocket je implementován ve všech moderních webových prohlížečích a dle statistik podílu využití dostupných na portálu StatCounter[5] jej podporují webové prohlížeče používané 90 % uživatelů.

Kód 2.2: Jednoduchá ukázka použití protokolu WebSocket v JavaScriptu. Po navázání spojení pošle klient serveru zprávu „Hello World! from client“ a server mu následně odpoví zprávou „Hello World! from server“.

```
// Node.js server s použitím knihovny „ws“:
var WebSocketServer = require('ws').Server;
var wss = new WebSocketServer({ port: 80 });

wss.on('connection', function (ws) {
    ws.on('message', function (message) {
        console.log('WebSocket received: ' + message);
        ws.send('Hello World! from server');
    });
});

// Klient
<script type="text/javascript">
    var socket = new WebSocket('ws://localhost');
    socket.onopen = function () {
        socket.send('Hello World! from client');
    };

    socket.onmessage = function (message) {
        console.log('WebSocket received: ' + message.data);
    };

    socket.onerror = function (error) {
        console.log('WebSocket error: ' + error);
    };
</script>
```

[5] <http://gs.statcounter.com/>

2.5 Použité nástroje

2.5.1 Atom

Při vývoji tohoto projektu jsem pro psaní veškerého zdrojového kódu používal svobodný textový editor Atom[6]. Atom pochází z dílny vývojářů GitHubu[7], jejichž cílem bylo vytvořit textový editor, který bude možné přizpůsobovat pomocí JavaScriptu, HTML a CSS. Již v základu podporuje Atom množství programovacích jazyků a jeho možnosti lze rozšířit stovkami komunitních balíčků[8].

2.5.2 JSHint

JSHint je open-source nástroj pro analýzu zdrojového kódu napsaném v jazyce JavaScript.

2.5.3 Arduino IDE

Ze začátku jsem pro vývoj programu pro Arduino používal open-source vývojové prostředí Arduino IDE[9] napsané v Javě. Podle mého názoru je však spíše vhodné pro menší projekty a nenabízí tolik možností jako většina vývojových prostředí pro jiné platformy. Proto jsem brzy přešel na PlatformIO, což je prostředí, které umožňuje vývoj pro velké množství platforem (mj. Arduino), založené na výše zmíněném textovém editoru Atom.

2.5.4 Inkscape

Až na výjimky se v mém projektu nachází pouze vektorová grafika. Pro její tvorbu jsem použil svobodný multiplatformní grafický vektorový editor Inkscape[10], který jako svůj nativní formát používá SVG verze 1.1 v kombinaci s CSS3. Implementace obou standardů je dosud neúplná, ale nástroj jako takový posloužil výborně a s jeho používáním mám pouze pozitivní zkušenosti.

---

[6] <https://atom.io/>

[7] GitHub je webová služba pro hostování repositářů verzovacího systému Git. <https://github.com/>

[8] <https://atom.io/packages>

[9] <https://www.arduino.cc/en/Main/Software>

[10] <https://inkscape.org>

2.5.5 Git

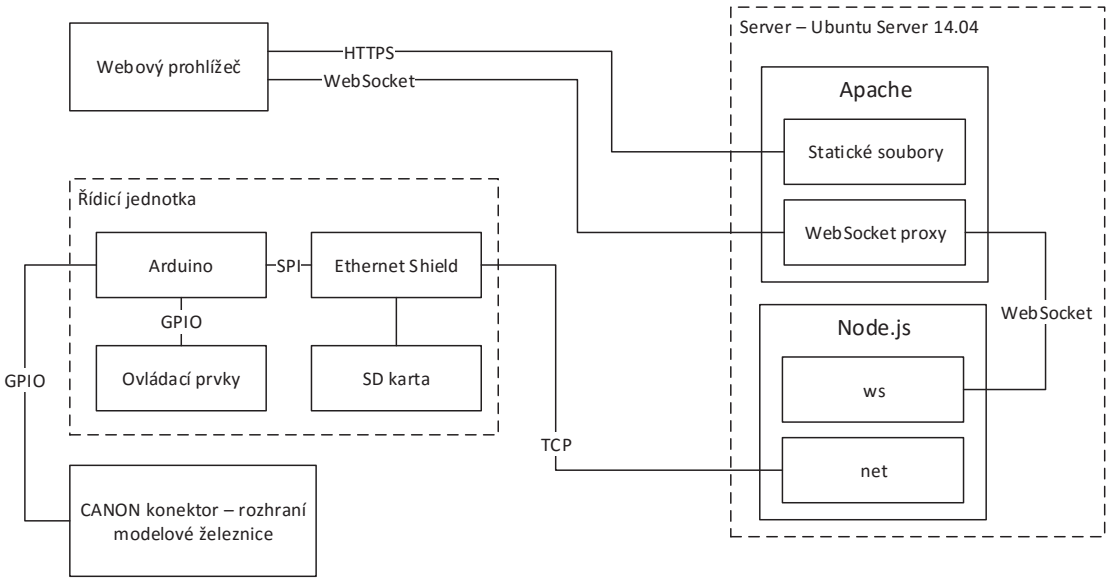
Přestože jsem na projektu pracoval sám a nebylo tedy nutností použít nástroj pro správu verzí, rozhodl jsem se použít distribuovaný systém správy verzí Git[11]. Git mi umožnil lepší přehled nad vývojem projektu a také se například vracet k předchozím verzím ve chvílích, kdy jsem se ve vývoji vydal špatným směrem.

[11] <https://git-scm.com/>

3. KAPITOLA

Řešení problematiky

V této kapitole je popsána implementace a realizace celého řídicího systému. Systém se skládá z řídicí jednotky, serveru a webové aplikace.



Obrázek 2 – Diagram řídicího systému

3.1 Řídicí jednotka

Jednou z hlavních funkcí řídicí jednotky je poskytnout rozhraní k ovládání modelové železnice. Řídicí jednotka je zároveň jádrem celého řízení a je na ní implementovaná veškerá logika. Jednotka má tedy celkový přehled o stavu modelové železnice, který oznamuje serveru. Ze serveru naopak dostává příkazy, stahuje si z něj aktuální konfiguraci a uživatelské skripty.

3.1.1 Zapojení

Jelikož železnice pracuje s různými napětími a hlavně se zde vyskytuje mnoho elektromagnetických prvků (elektromotory lokomotiv, elektromagnety návěstidel, výhybek a přestavníků), bylo zvoleno optické oddělení řízení železnice pomocí optočlenů PC844 (KP30400A), které jsou doplněny výkonovým tranzistorem a ochrannou diodou. Tím se

vyhneme případnému průniku přepětí či rušivých špiček do elektrického obvodu mikrokontroléru nebo počítače, k němu připojeného. Cena této vyšší bezpečnosti se jeví jako přijatelná.

V následující tabulce je přehled všech GPIO pinů Arduina s popisky znázorňujícími, co je na daný pin připojeno. Nevyužité piny jsou označeny *NC*.

Tabulka 2 – Zapojení GPIO pinů Arduina

Pin	Připojeno	Pin	Připojeno	Pin	Připojeno	Pin	Připojeno
A0	NC	2	BUTTON6	20	NC - SDA	38	C1
A1	NC	3	LED2	21	NC - SCL	39	K1
A2	NC	4	SS - SD card	22	VK1	40	C2
A3	NC	5	LED1	23	SA11	41	K2
A4	NC	6	LCD Enable	24	VK5	42	STŮJ C
A5	NC	7	LCD RS	25	SA21	43	K3
A6	NC	8	LCD D7	26	VK2	44	STŮJ A
A7	NC	9	LCD D6	27	SB21	45	K4
A8	BUTTON2	10	SS - Ethernet	28	VK1B	46	SC11
A9	BUTTON1	11	LCD D5	29	SA51	47	SB11
A10	NC	12	LCD D4	30	VK2B	48	STŮJ B
A11	NC	13	NC - LED	31	SA41	49	SC21
A12	NC	14	LED6	32	VK4	50	MISO
A13	NC	15	LED5	33	SA31	51	MOSI
A14	NC	16	LED4	34	K5	52	SCK
A15	NC	17	LED3	35	K1B	53	SS - HW
0	RX	18	BUTTON4	36	K1zpět		
1	TX	19	BUTTON3	37	K2B		

3.1.2 Konstrukční řešení

Jako hlavní konstrukční prvek byla zvolena plastová krabička KP28 (Z-37), do které jsem řídicí jednotku zabudoval. Řídicí jednotka se skládá z mikrokontrolérové desky Arduino Mega, LCD displeje a dvou desek plošných spojů (viz obrázky 13 a 14). Jeden z těchto plošných spojů slouží jako rozhraní mezi modelovou železnicí a mikrokontrolérem. Je z něj tedy vyveden kabel s CANON konektorem, který slouží pro připojení ke stávajícímu ovládacímu panelu modelové železnice. Druhý plošný spoj obsahuje indikační LED diody a tlačítka a je uchycen k horní části konstrukčního boxu, v němž jsem pro tlačítka i diody vyvrtal otvory. LCD displej a obě desky jsou s Arduinem propojeny pomocí plochých vícežilových kabelů. Všechny konektory jsou umístěny na pravé boční straně řídicí jednotky.

Plošné spoje jsem navrhnul v programu Sprint Layout a vyrobil jsem je fotocestou, tj. osvitem desky plošného spoje a vyleptáním v roztoku chloridu železitého. Schémata a další podklady pro řídicí jednotku jsou obsaženy v příloze.

3.1.3 Fáze programu

Následuje zjednodušený popis činností, které program řídicí jednotky po spuštění provádí.

3.1.3.1 Inicializace

1. Prvním krokem je nastavení znakové rychlosti sériové linky (9600 baudů) používané pro komunikaci s počítačem během diagnostiky a testování.

2. Následuje inicializace LCD displeje, během které se do paměti řadiče displeje nahraje 8 vlastních znaků, kterými doplňuji již přítomnou znakovou sadu anglické abecedy. Na první řádek displeje se vypíše text `Rizeni modelove zeleznice` a na druhý řádek aktuálně prováděná činnost či stav.

3. Dalším krokem je spuštění DHCP klientu a automatická konfigurace síťového rozhraní (Ethernet shield). Během tohoto kroku je na druhém řádku displeje vypsán text `> setting net`. Jakmile je síťové rozhraní připraveno pro komunikaci, vypíše se na displej stav `> network set` a po 500ms prodlevě `> connecting`. V tu chvíli se program snaží navázat TCP spojení se serverem za předpokladu, že v lokální síti správně funguje překlad doménových jmen.

4. V případě úspěšného pokusu o navázání spojení se na displeji vypíše text `> connected` a rozsvítí se kontrolní LED dioda s popiskem „online“. V opačném případě se na displeji objeví `> conn. failed`, ale nový pokus o připojení se prozatím nezahájí.

5. Následuje inicializace SD karty, na které je uložena veškerá konfigurace a také poslední stav kolejiště, doprovázená textovým popiskem `> SD init`.

6. Nakonec proběhne konfigurace GPIO a vyšlou se 3 impulzy pro nastavení výchozího stavu výhybek.

3.1.3.2 Hlavní cyklus

Řídicí jednotka musí obsluhovat současně několik činností, které se ale nesmí navzájem blokovat. Programovací jazyk Arduina však standardně nepodporuje asynchronní či paralelní zpracování úloh, a proto jsem musel tyto činnosti rozložit na co nejmenší části, které se postupně za sebou provádějí v nekonečné smyčce. Alternativním řešením by mohlo být spouštění těchto činností na základě přerušení z časovače, ale praxe ukázala, že je původní řešení plně dostačující.

1. Z interního bufferu síťového rozhraní se načte jeden příchozí byte (interpretovaný jako znak) a uloží se do bufferu pro příchozí zprávu, pro který je rezervováno 120 bytů v paměti SRAM.

2. Pokud se jedná o znak EOL a tudíž konec zprávy, předá se obsah bufferu metodě obsahující implementaci mého komunikačního protokolu. Tato metoda zprávu zpracuje a vykoná.

3. Poté se v případě, že v tuto chvíli běží skript ovládání kolejiště a příchozí buffer síťového rozhraní je prázdný (není přijímána žádná zpráva), načte a provede následující příkaz z aktuálně běžícího skriptu.

4. Další činností je skenování všech výstupů ze spínacích kolejí. Pokud je čtení vyhodnoceno jako průjezd vlaku (impuls musí být dostatečně dlouhý, aby se se zamezilo chybě), je číslo daného GPIO pinu předáno metodě, která je zodpovědná za udržování tabulky obsazených úseků. Pokud je vše v pořádku, nastaví se přiřazením identifikačního čísla vlaku z předchozího úseku (podle definice návaznosti úseků) navazující úsek jako obsazený. Předchozí úsek, z kterého vlak vyjel, se nastaví jako volný. Udržování tabulky úseků nicméně není to jediné, co tato metoda obstarává. Na základě této tabulky totiž také rozhoduje, které povely je bezpečné provést v případě vypravování vlaků.

5. Kontrola stisknutí dvou tlačítek, na která nevystačily vstupy externího přerušení. (kurzorové šipky)

6. Následuje překreslení displeje, pokud je to vyžádáno. Děje se tak po změně stavu běhu skriptu či po stisku nějakého tlačítka sloužícího pro pohyb v menu.

7. Na konci cyklu probíhá ověření připojení k serveru popsané v kapitole 3.4.2. Pokud spojení vypršelo nebo bylo při inicializaci řídicí jednotky napoprvé přeskočeno, pokusí se v této fázi řídicí jednotka spojení znovu navázat. Ale pouze za předpokladu, že není v pohybu žádný vlak. Sledování obsazených úseků je totiž kritická, časově citlivá operace, a jelikož navazování TCP spojení (stejně jako každá jiná I/O operace) blokuje provádění dalšího kódu a v některých případech může připojování k serveru trvat delší dobu (až v řádu jednotek sekund), mohlo by se stát, že se během toho nezaznamená průjezd jedoucích vlaků. To by způsobilo chybné údaje v tabulce obsazených úseků, která by musela být resetována.

8. Cyklus se znovu opakuje.

3.1.4 Persistence dat

Jako úložiště pro dlouhodobé uchování dat jsem se rozhodl použít microSD kartu místo EEPROM paměti přímo v mikrokontroléru. Hlavními důvody byla nízká kapacita paměti (pouze 4 kB) a krátká životnost omezená počtem zapisovacích cyklů. SD karta nabízí více prostoru pro data a díky dlouhé životnosti může řídicí jednotka zapisovat data kdykoliv, kdy je to potřeba. Pokud tedy dojde k náhlé ztrátě napájení, lze všechna data bez jakýchkoliv problémů obnovit.

Mezi data ukládaná na SD kartu patří konfigurace systému, tabulka obsazených úseků, aktuální stav výhybek a stav běhu programu. Na paměťovou kartu se také ukládají všechny skripty ovládání kolejiště.

3.1.5 Ovládací prvky

Pro interakci uživatele nabízí řídicí jednotka pouze omezené možnosti. Jedná se o 6 mikrospínačů, 6 indikačních LED diod (5 zelených, 1 červená) a podsvícený alfanumerický LCD displej s 2×16 znaky.

3.1.5.1 Indikační LED

- Online – Indikuje stav spojení se serverem. Pokud dioda nesvítí, řídicí jednotka se pokouší o připojení k serveru. Jakmile dojde k úspěšnému navázání spojení, dioda se rozsvítí. Pokud se nedaří navázat spojení již delší dobu (asi 30 sekund), přejde jednotka do offline režimu, který je indikován velmi slabým rozsvícením diody generováním PWM pulzu. Pro opětovný pokus o připojení je třeba řídicí jednotku resetovat.

- Dojezd – Tato dioda se běžně rozsvěcí při běžícím skriptu v momentě, kdy se čeká na dojezd minimálně jednoho vlaku. Dioda zhasne, jakmile všechny vlaky, na které se čekalo, stojí.

- Střídání – Dioda svítí, pokud je zapnuto střídání vlaků. Tedy pokud smějí nákladní vlaky jezdit i na vnitřní okruh a osobní na vnější.

- Impuls – Dioda svítí v okamžiku, kdy je generován vybavovací impuls alespoň na jeden z výstupů.

- Čtení – Indikuje čtení ze spínacích kolejí.

- Chyba – Dioda určená ke hlášení chyb. Při generování upozornění z tabulky 6 dioda jen krátce červeně blikne. V případě, že se jedná o kritickou chybu (např. chyba inicializace SD karty), se dioda trvale rozsvítí a program se zastaví. Poté je nutné závadu odstranit a resetovat řídicí jednotku.

3.1.5.2 Mikrospínače

- První dva mikrospínače slouží jako kurzorové šipky pro pohyb v menu vykreslovaném na LCD displeji.

- START/STOP – Tlačítko slouží pro potvrzování volby na displeji a ke spouštění či zastavení běhu uživatelského skriptu.

- PAUSE – Slouží pro pozastavení či pokračování v běhu uživatelského skriptu. Stav běhu je vykreslován na LCD displeji.

- RESET – Slouží pro resetování mikrokontroléru.

- DEFAULT – Po stisku tohoto tlačítka se aktuální informace o obsazených úsecích přepíší výchozím stavem. Funkce je užitečná zejména v případech, kdy je nutné resetovat stav kolejiště.

3.1.5.3 LCD displej

LCD displej je jediná zobrazovací jednotka mé řídicí jednotky. Jeho hlavní účel je zobrazování stavu inicializace, možnost výběru skriptu z menu a sledování jeho běhu.

Obrázek 3 – Výchozí menu řídicí jednotky

Po inicializaci řídicí jednotky se na displeji zobrazí očíslovaný seznam uživatelem vytvořených skriptů, které jsou momentálně uloženy na SD kartě. Pro pohyb v menu slouží tlačítka s kurzorovými šipkami a pro usnadnění orientace se na displeji vykresluje rolovací lišta. Pro spuštění skriptu slouží tlačítko *START*. Po jeho stisku se zobrazí dotaz na počet cyklů (opakování), který lze zvolit kurzorovými šipkami a opět potvrdit tlačítkem *START*. Maximální počet opakování, jenž lze zvolit, je 255. Pokud uživatel vybere 0, bude se skript neustále opakovat bez omezení.

Při běhu skriptu je na prvním řádku displeje vypsán název skriptu a ikonkou je znázorněn jeho stav (► – skript běží, ▮▮ – skript je pozastaven). Na druhém řádku se v intervalu 2 sekund střídají tyto informace: doba v běhu skriptu v minutách, počet uplynulých cyklů z celkového počtu cyklů skriptu a počet zpracovaných příkazů z celkového počtu příkazů skriptu. Po skončení skriptu se opět zobrazí výchozí menu.

Obrázek 4 – Informace o běhu skriptu

3.2 Webová aplikace

3.2.1 Serverová část

Serverová část aplikace je vyvinuta v Node.js a slouží hlavně jako rozhraní a mezipaměť mezi klientskou webovou aplikací a řídicí jednotkou. Serverová aplikace obsahuje správu

připojených relací, úložiště pro konfiguraci systému a pro všechny uživatelem vytvořené skripty. Sama však neobsahuje žádnou řídicí logiku, ani žádné jiné podpůrné funkce, které by mohla řídicí jednotka během ovládání kolejiště využít. Funkce serveru jsou podrobněji popsány na návrhu komunikačního protokolu v kapitole 3.4.

3.2.2 Klientská část

Klientská část webové aplikace již slouží k samotnému interaktivnímu ovládání kolejiště a monitorování jeho stavu. Návrh prostředí webové aplikace je popsán v kapitole 4.

3.2.2.1 Vykreslování stavu kolejiště

Zásadním rozhodnutím při návrhu webové aplikace bylo, jakou technologii zvolit pro vykreslování stavu kolejiště. Již od začátku jsem počítal se dvěma možnostmi: SVG a canvas. Zatímco SVG je značkovací jazyk pro popis vektorové grafiky, canvas je HTML5 element, který umožňuje dynamické vykreslování dvojrozměrných tvarů a bitmapové grafiky. Obě technologie by splnily svůj účel. Po zvážení všech kladů a záporů jsem se nakonec rozhodl pro SVG. Co se týče výkonu, byl by canvas zajisté rychlejší, ale to by se projevilo pouze za určitých okolností. Pokud by bylo potřeba vykreslovat například tisíce objektů se složitými animacemi, přiklonil bych se rozhodně ke canvasu. Ale vzhledem k tomu, že celkový počet objektů byl ve výsledku maximálně několik set a objekty nebylo potřeba složitě animovat, rozdíl v rychlosti vykreslování by byl neznatelný. Výběr SVG v tomto případě přinesl rychlejší možnost vývoje, protože jsou objekty SVG plně přístupné přes DOM API jazyka JavaScript. Navíc je SVG nezávislé na rozlišení, takže se může přizpůsobit velikosti obrazovky.

3.3 Skriptovací jazyk

Pro možnost vytváření vlastních skriptů pro ovládání kolejiště bylo potřeba navrhnout jednoduchý jazyk, který bude řídicí jednotkou nějakým způsobem interpretován. Navržený jazyk obsahuje celkem sedm příkazů. Jednotlivé příkazy se píší na samostatné řádky a mají povinné parametry, které jsou od příkazu odděleny mezerou a podléhají následujícím pravidlům. Každý skript může obsahovat maximálně 255 příkazů, což je pro běžné užití více než dostatečné. Zdrojový kód se před uložením překládá do binární podoby.

3.3.1 Možné příkazy

- `volno` – Příkaz vypraví vlak z koleje uvedené v parametru. Parametrem může být označení kterékoliv koleje v nádražích (viz K*x* v tabulce 1) nebo v tunelech (C*x*). Příkaz se stejně jako v případě dvou následujících interpretuje jako impulz na daný pin a parametr příkazu je překladačem zaměněn za číslo daného pinu podle tabulky 1.

- `výhybka` – Přestaví příslušnou výhybku na kolej uvedenou v parametru. Možné parametry: *K1*, *K2*, *K1B*, *K2B*, *K4*, *K5*.

- `stůj` – Příkaz `stůj` nastaví návěst na červenou pro souběžné koleje v nádraží či tunelu. Akceptovaným parametrem je označení nádraží/tunelu znakem *A*, *B* nebo *C*.

- `prodleva` – Pozastaví na určitou dobu běh skriptu. Parametrem je celé číslo v intervalu <0;255>, které udává dobu čekání v sekundách.

- `dojezd` – Pozastaví běh skriptu, dokud postupně nedorazí signály z libovolných dojezdových spínacích kolejí. Parametr určuje, na kolik různých signálů se má počkat. Vzhledem k velikosti kolejiště je interval parametru omezen na <1;2>. Čeká se tedy vždy na dojezd buď jednoho, nebo dvou vlaků.

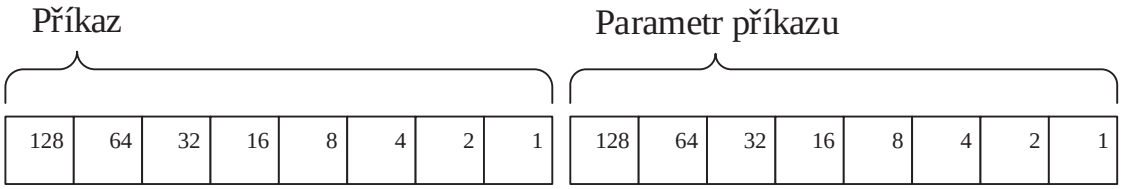
- `střídání` – Povolí či zakáže střídání vlaků. Parametrem je text *povoleno* nebo *zakázáno*.

- `náhodně` – Slouží k náhodnému vypravení vlaku z množiny kolejí dané parametrem. Tím může být jedna z těchto definovaných množin kolejí: *vše*, *osobní*, *nákladní*, *K1K2*, *K4KB*, *K3K4K5*, *nádraží_B*, *tunel*. Zpracování příkazu probíhá tak, že se z uvedené množiny kolejí vyřadí všechny, které není bezpečné provést z toho důvodu, že je navazující úsek obsazený. Pak se podle generátoru pseudonáhodných čísel[12] vybere jedna z těchto kolejí. Když vznikne situace, že má vlak více možností úseků, do kterých smí vjet, tak ještě řídicí jednotka náhodně přestaví výhybky.

[12] Jako seed je použit šum z některého odpojeného vstupního analogového pinu.

3.3.2 Překlad

Pro co nejsnazší zpracování skriptů řídicí jednotkou probíhá ještě před nahráním skriptu do její paměti překlad zdrojového kódu do binární podoby. Tím dojde ke snížení paměťové náročnosti při interpretaci. O překlad se stará serverová aplikace. Každý příkaz zabírá v binární podobě 2 byty, přičemž první (vyšší) obsahuje číslo příkazu (viz tabulka 3) a druhý obsahuje samotný parametr. Vzhledem k tomu, že je počet příkazů v jednom skriptu omezen na 256, velikost přeloženého skriptu může být maximálně 512 B.



Obrázek 5 – Binární podoba příkazu

Kód 3.3: Ukázka kódu kratšího skriptu

```
prodleva 3
volno K3
volno K2B
dojezd 2
prodleva 7
náhodně K1K2
dojezd 1
prodleva 7
náhodně K4K5
dojezd 1
prodleva 7
volno C2
prodleva 2
volno C1
dojezd 2
prodleva 7
volno K1B
dojezd 1
prodleva 5
```

Kód 3.4: Hexadecimální přepis přeložené podoby

```
0103 002b 0025 0202 0107 0403 0201 0107
0404 0201 0107 0028 0102 0026 0202 0107
0023 0201 0105
```

Tabulka 3 – Překlad příkazů

Příkaz	**Hodnota**
volno	0_{DEC}
výhybka	0_{DEC}
stůj	0_{DEC}
prodleva	1_{DEC}
dojezd	2_{DEC}
střídání	3_{DEC}
náhodně	4_{DEC}

3.4 Komunikační protokol

Jedním z aspektů této práce byl návrh vhodného způsobu komunikace mezi řídicí jednotkou a webovou aplikací. Kvůli tomu, že se řídicí jednotka může nacházet v jiné síti, než odkud se uživatel snaží připojit, a předpokládá se, že tato síť neumožňuje, aby byla řídicí

jednotka přímo dostupná ze sítě Internet, vložil jsem mezi řídicí jednotku a webovou aplikaci veřejně dostupný server. Webovou aplikaci tedy nebude hostovat řídicí jednotka, ale tento server, se kterým bude řídicí jednotka i webová aplikace komunikovat.

Vzhledem k povaze projektu bylo potřeba zajistit oboustrannou komunikaci klientů a serveru v reálnem čase. V případě webové aplikace jsem využil možnosti aplikačního protokolu Websocket v zabezpečené variantě. Řídicí jednotka se serverem komunikuje čistě na transportní vrstvě přes trvalý TCP socket. V obou případech se data přenáší v textové podobě (kódování UTF-8 v případě webové aplikace, ASCII v případě řídicí jednotky) ve formě JavaScriptového objektového zápisu JSON. Každý takový objekt představuje samostatnou zprávu. Význam zprávy určuje hodnota atributu *type* a všechny další parametry jsou vnořeny do atributu *data*. Jelikož se serverová aplikace chová jako prostředník mezi webovou aplikací a řídicí jednotkou, jsou zprávy některých typů bez jakékoliv změny pouze předány z jednoho komunikačního kanálu na druhý.

K identifikaci klientů, zejména kvůli zabezpečení, slouží v případě webové aplikace atribut *uuid*, který obsahuje 128bitový UUID identifikátor relace. Oproti tomu se řídicí jednotka identifikuje 128bitovým otiskem, který je statický a ve zprávě je obsažen v atributu *secret v* kanonickém tvaru.

3.4.1 Komunikace mezi webovou aplikací a serverem

V této části popisuji veškeré typy zpráv mého komunikačního protokolu mezi webovou aplikací a serverem. Zprávy jsou popsány na konkrétních příkladech v pořadí, ve kterém mohou obvykle nastat. Následující ukázky zpráv obsahují pro lepší čitelnost odřádkování i odsazení atributů, v reálném nasazení se však žádné nadbytečné netisknutelné znaky neposílají.

3.4.1.1 Autentifikace uživatele

Webová aplikace ihned po navázání spojení se serverem odesílá zprávu typu `init` se svým UUID. V tomto případě byla aplikace spuštěna poprvé, takže je UUID nulové.

Kód 3.5: Zpráva typu `init` ve směru klient → server

```
{
  "type": "init",
  "data": { },
  "uuid": "00000000-0000-0000-0000-000000000000"
}
```

V případě, že je přijaté UUID nulové nebo neznámé, zaznamená server informaci o nové relaci a vygeneruje nový identifikátor, který pošle klientu v následující odpovědi. Od této chvíle se bude klient identifikovat novým UUID, které se uloží v objektu HTML5 Local Storage.

Kód 3.6: Zpráva typu `uuid` ve směru server → klient

```
{
  "type": "uuid",
  "data": {
    "value": "35a9112e-7038-0faf-3f7b-f4b5a82e50e5"
  }
}
```

Poté, co uživatel zadá své přihlašovací jméno a heslo, jsou přihlašovací údaje odeslány serveru ve zprávě typu `authenticate`. Dokud nedojde k ověření uživatele, server všechny ostatní zprávy ze strany klientu ignoruje.

Kód 3.7: Zpráva typu `authenticate` ve směru klient → server

```
{
  "type": "authenticate",
  "data": {
    "user": "test",
    "password": "test"
  },
  "uuid": "35a9112e-7038-0faf-3f7b-f4b5a82e50e5"
}
```

Pokud se uživatelské jméno a heslo shoduje s databází, je tato relace schválena a server odešle klientu zprávu typu `initialization`, čímž dojde k počátečnímu nastavení webové aplikace. Tato zpráva obsahuje veškeré aktuální informace popisující stav kolejiště, jako je obsazení úseků (pole *sections*), stav výhybek (pole *turnouts*), seznam všech uživatelem vytvořených skriptů s jejich názvy a popisky (pole *programs*), informace o aktuálně běžícím skriptu (objekt *program*), všechny hodnoty nastavení (objekt *settings*) a nakonec informaci, zda je momentálně připojena řídicí jednotka (hodnota *connected*).

Kód 3.8: Zpráva typu `initialization` ve směru server → klient


```
{
  "type": "initialization",
  "data": {
    "sections": [ 0, 1, 2, 3, 4, 2, 0, 0, 0, 0, 0, 0, 5 ],
    "turnouts": [ false, false, false ],
    "programs": [
      {
        "file": "Program 1",
        "label": "prog001",
        "name": "Automatické řízení 6 vlaků, využito K1zpět"
      }
    ],
    "program": {
      "endAfterCycle": false,
      "label": "",
      "mix": false,
      "name": "",
      "remainingCycles": 0,
      "state": 0
    },
    "settings": {
      "defaultOnly": false,
      "randomOnly": false,
      "defaultSections": [ 0, 1, 2, 3, 4, 6, 0, 0, 0, 0, 0, 0, 5 ],
      "restrictedSections": [
        [ 0, 2, 0 ],
        [ 0, 2, 0 ],
        [ 0, 2, 0 ],
        [ 1, 0, 0 ],
        [ 0, 0, 1 ],
        [ 0, 0, 0 ],
      ]
    },
    "connected": false
  }
}
```


Jakmile se některá informace z předchozí zprávy změní, informuje server webovou aplikaci vhodnou zprávou, která obsahuje pouze změněná data. Na následujících příkladech podrobněji popíši tyto zprávy.

3.4.1.2 Aktualizace dat

Pro aktualizaci seznamu uživatelem vytvořených skriptů posílá server zprávu typu `programs-list`. Ta obsahuje pole objektů s názvy a popisky všech skriptů a také název souboru, ve kterém je daný skript na serveru uložen.

Kód 3.9: Zpráva typu `programs-list` ve směru server → klient

```
{
  "type": "programs-list",
  "data": {
    "programs": [
      {
        "file": "prog001",
        "name": "Program 1",
        "label": "Automatické řízení 6 vlaků, využito K1zpět"
      }, {
        "file": "prog002",
        "name": "Program 2",
        "label": "Automatické řízení 4-6 vlaků, C2 využito k zastavování"
      },
    ]
  }
}
```

Pokud se jakkoli změní stav běhu skriptu, posílá server webovým klientům zprávu typu `program-status`. Tato zpráva obsahuje název a popisek běžícího skriptu, stav jeho běhu (viz tabulka 4), počet zbývajících cyklů (hodnota *remainingCycles*) a informaci, zda má být běh skript ukončen po skončení aktuálního cyklu nehledě na skutečný počet zbývajích cyklů (hodnota *endAfterCycle*). Hodnota *mix* dále určuje, zda je povoleno střídání vlaků.

Tabulka 4 – Stavy běhu skriptu

Stav	**Význam stavu**
0	Žádný skript není spuštěn
1	Běh skriptu je pozastaven
2	Skript právě běží

Kód 3.10: Zpráva typu `program-status` ve směru řídicí jednotka → server → webový klient

```
{
  "type": "program-status",
  "data": {
    "state": 2,
    "mix": false,
    "remainingCycles": 1,
    "endAfterCycle": false,
    "name": "Program 2",
    "label": "Automatické řízení 4-6 vlaků, C2 využito k zastavování"
  }
}
```

O změnách stavu úseků v důsledku pohybu vlaků po kolejišti informuje server webové klienty zprávou typu `sections-table`. Tato zpráva obsahuje pole všech úseků kolejiště,

ve kterém hodnoty jednotlivých položek obsahují identifikační číslo vlaku (viz tabulka 5), který se v tomto úseku nachází. Hodnota 0 symbolizuje volný úsek.

Tabulka 5 – Označení jednotlivých vlaků

Vlak	**Název**
0	Volný úsek
1	Motoráček
2	Rychlík
3	Rychlík patrový
4	Nákladní 1 (rychlý)
5	Nákladní 2 (parní)
6	Nákladní 3 (diesel)

Kód 3.11: Zpráva typu `sections-table` ve směru řídicí jednotka → server → webový klient

```
{
  "type": "section-table",
  "data": {
    "table": [ 0, 1, 2, 3, 4, 2, 0, 0, 0, 0, 0, 0, 5 ]
  }
}
```

Jakmile dojde k přestavení jedné z výhybek, posílá server zprávu typu `turnout-table`, která obsahuje pole s aktuálními stavy všech výhybek, které řídicí jednotka může ovládat. Výhybky jsou v pořadí *VA1*, *VA5*, *VB1*. Každá výhybka má pouze dva stavy, proto byl užit datový typ Boolean.

Kód 3.12: Zpráva typu `turnout-table` ve směru řídicí jednotka → server → webový klient

```
{
  "type": "turnout-table",
  "data": {
    "table": [ false, false, false ],
  }
}
```

O připojení nebo ztrátě komunikace s řídicí jednotkou informuje server webové klienty zprávou typu `unit-status`. Zda je jednotka připojena nebo odpojena, určuje hodnota *connected*. V případě, že je řídicí jednotka odpojena, je znemožněno manuální řízení kolejiště a další funkce, které vyžadují připojenou jednotku. Uživatel má ale i nadále možnost vytvářet a upravovat řídicí skripty a měnit nastavení. Všechny provedené změny se nahrají do paměti řídicí jednotky v momentě jejího připojení.

Kód 3.13: Zpráva typu `unit-status` ve směru server → klient

```
{
  "type": "unit-status",
  "data": {
    "connected": false
  }
}
```

3.4.1.3 Interakce uživatele

Prostředí webové aplikace nabízí pro každé návěstidlo na plánku kolejiště ovládací prvek, který slouží k vyslání povelu *volno* a tudíž k vypravení vlaku z dané koleje. To je ze strany webové aplikace zajištěno posláním zprávy typu `impulse`, která obsahuje parametr *pin*, jež určuje číslo GPIO pinu, na který řídicí jednotka vyšle impuls, podle zapojení znázorněného v tabulce 2 (viz *Kx* ve sloupci *připojeno*). Dalším parametrem je délka trvání impulsu v milisekundách (hodnota *duration*), který je dán konfigurací aplikace.

Tato zpráva je kromě vyslání povelu k vypravení vlaku používána také pro přestavování výhybek. V tomto případě se opět pošle zpráva s číslem pinu podle tabulky 2 (viz *VKx*).

Následující příklad zprávy je řídicí jednotkou interpretován jako pokus o vyslání 200 milisekundového impulsu na digitální pin č. 34, což v případě úspěšného vyhodnocení zapříčiní vypravení vlaku z koleje *K5*.

Kód 3.14: Zpráva typu `impulse` ve směru webový klient → server → řídicí jednotka

```
{
  "type": "impulse",
  "data": {
    "pin": 34,
    "duration": 200
  },
  "uuid": "35a9112e-7038-0faf-3f7b-f4b5a82e50e5"
}
```

O tom, zda byl impuls skutečně vyslán, je klient informován zprávou typu `report`, která se odesílá při každé změně logické hodnoty některého digitálního výstupního pinu. Zpráva obsahuje číslo pinu a jeho aktuální stav (*state*). To dává možnost webové aplikaci znázornit skutečný průběh impulsu.

Kód 3.15: Zpráva typu `report` ve směru řídicí jednotka → server → webový klient

```
{
  "type": "report",
  "data": {
    "pin": 34,
    "state": true
  }
}
```

Jakékoliv chyby a další upozornění jsou webovému klientu předávány ve zprávě typu `notice`, která jednak obsahuje číselný kód upozornění v parametru *code* a v určitých případech také obsahuje nepovinný parametr *cause*, který v závislosti na významu upozornění obsahuje dodatečnou informaci. Tou je například v případě upozornění s kódem 1 číslo obsazeného úseku. Kód 1 informuje o tom, že nelze vypravit určený vlak, protože je následující úsek obsazen. Významy kódů upozornění jsou popsány v tabulce 6, podbarvené kódy jsou generovány serverem, ostatní generuje řídicí jednotka.

Kód 3.16: Zpráva typu `notice` ve směru řídicí jednotka → server → webový klient

```
{
  "type": "notice",
  "data": {
    "code": 1,
    "cause": 34
  }
}
```

Tabulka 6 – Významy jednotlivých upozornění

Kód	**Význam**
0	Nelze vypravit vlak, kolej je prázdná.
1	Povel není bezpečné provést, úsek je obsazen.
2	Nelze spustit skript, soubor neexistuje.
3	Soubor, který obsahuje skript, nelze otevřít.
4	Není spuštěný žádný skript.
5	Nelze spustit skript, úseky nejsou ve výchozím stavu.
6	Řídicí jednotka byla připojena nebo odpojena (závisí na data.cause).
7	Objevila se neznámá chyba.
8	Kolej je blokována na základě nastavení uživatelem.
9	Spojení se serverem bylo navázáno.
10	Nastavení bylo uloženo.
11	Kolej je pro tento vlak zakázaná
12	Bylo zadáno nesprávné jméno nebo heslo.

Celý systém si za normálních podmínek udržuje informace o obsazení jednotlivých úseků kolejiště. V případě, že bylo užito ruční ovládání pomocí původního ovládacího panelu, je třeba resetovat stav kolejiště. K tomu slouží následující zpráva typu `default-sections`, odesílaná webovým klientem bez jakýchkoliv parametrů, která informuje systém, že se vlaky nachází ve výchozím postavení definovaném v uživatelském nastavení. Funkce je také užitečná během ladění.

Kód 3.17: Zpráva typu `default-sections` ve směru webový klient → server → řídicí jednotka

```
{
  "type": "default-sections",
  "data": { },
  "uuid": "35a9112e-7038-0faf-3f7b-f4b5a82e50e5"
}
```

Pokud uživatel provede změny v uživatelském nastavení a uloží tyto změny, odešlou se všechny hodnoty nastavení na server ve zprávě `update-settings`. První parametr *defaultOnly* určuje, zda je možné uživatelský skript spustit i v případě, že nejsou vlaky ve výchozím stavu. Dále se ve zprávě nachází pole *defaultSections* s výchozí obsazením úseků a *restrictedSections*, které pro každý vlak určuje zakázané koleje. O tom, zda má systém hlídat zakázané koleje pouze při náhodném provozu, rozhoduje parametr *randomOnly*.

Kód 3.18: Zpráva typu `update-settings` ve směru klient → server

```
{
  "type": "update-settings",
  "data": {
    "defaultOnly": false,
    "randomOnly": false,
    "defaultSections": [ 0, 1, 2, 3, 4, 6, 0, 0, 0, 0, 0, 0, 5 ],
    "restrictedSections": [
      [ 0, 2, 0 ],
      [ 0, 2, 0 ],
      [ 0, 2, 0 ],
      [ 1, 0, 0 ],
      [ 0, 0, 1 ],
      [ 0, 0, 0 ],
    ]
  }
}
```

3.4.1.4 Vytváření a editace skriptů

Pro získání zdrojového kódu skriptu, o který webová aplikace žádá v případě, že chce uživatel upravit některý skript ovládání kolejiště, se použije zpráva typu `get-source` s parametrem *file* obsahujícím identifikátor daného skriptu.

Kód 3.19: Zpráva typu `get-source` ve směru klient → server

```
{
  "type": "get-source",
  "data": {
    "file": ""
  },
  "uuid": "35a9112e-7038-0faf-3f7b-f4b5a82e50e5"
}
```

Pro odpověď použije server zprávu typu `source-code`, která v parametru *source* obsahuje zdrojový kód skriptu a pro kontrolu je zde znovu parametr file.

Kód 3.20: Zpráva typu `source-code` ve směru server → klient

```
{
  "type": "source-code",
  "data": {
    "source": ""
  }
}
```

Pokud chce uživatel odstranit nějaký skript, odešle serveru webová aplikace po potvrzení této akce zprávu typu `remove-program` opět s parametrem *file*. V případě úspěchu odesílá server všem klientům zprávu `programs-list` viz Kód 3.9.

Kód 3.21: Zpráva typu `remove-program` ve směru klient → server

```
{
  "type": "remove-program",
  "data": {
    "file": ""
  },
  "uuid": "35a9112e-7038-0faf-3f7b-f4b5a82e50e5"
}
```

Pro odeslání upraveného zdrojového kódu skriptu má webová aplikace k dispozici zprávu typu `update-program`. Parametr *file* opět obsahuje textový identifikátor skriptu nebo slovo „new“ v případě, že se jedná o nově vytvořený skript. Následují parametry *name* a *label* obsahující název a popisek skriptu a konečně parametr *code*, ve kterém je obsažen

nový zdrojový kód skriptu. Jelikož je tato zpráva používána při ukládání skriptu i pouhém ověřování zdrojového kódu, je zde navíc přítomen parametr *verify*. Pokud je jeho hodnota *true*, je zdrojový kód pouze ověřován a neukládá se.

Kód 3.22: Zpráva typu `update-program` ve směru klient → server

```
{
  "type": "update-program",
  "data": {
    "file": "",
    "name": "",
    "label": "",
    "verify": false,
    "code": ""
  },
  "uuid": "35a9112e-7038-0faf-3f7b-f4b5a82e50e5"
}
```

Po přijetí zprávy `update-program` posílá server webové aplikaci zprávu typu `source-verified`, která obsahuje výsledek překladu zdrojového kódu. Výsledek překladu, který se má zobrazit uživateli, je obsažen v parametru *text* a k barevnému vyjádření významu se použije barva v parametru *color*. Pro kontrolu je opět uveden identifikátor skriptu v parametru *file* a o tom, zda byl skript uložen, informuje parametr *saved*.

Kód 3.23: Zpráva typu `source-verified` ve směru server → klient

```
{
  "type": "source-verified",
  "data": {
    "result": {
      "text": "",
      "color": "",
      "file": "",
      "saved": false
    }
  }
}
```

3.4.1.5 Řízení běhu skriptu

Pro spuštění běhu jednoho z vytvořených skriptů odesílá klient zprávu typu `start-program`, která obsahuje identifikátor skriptu (parametr *file*), kterým je název souboru, do kterého jej serverová aplikace zkompilovala, a počet cyklů (parametr *cycles*), tj. počet opakování běhu skriptu, který uživatel zadal před spuštěním.

Kód 3.24: Zpráva typu `start-program` ve směru webový klient → server → řídicí jednotka

```
{
  "type": "start-program",
  "data": {
    "file": "",
    "cycles": 0
  },
  "uuid": "35a9112e-7038-0faf-3f7b-f4b5a82e50e5"
}
```

Pokud chce uživatel nějakým způsobem změnit běh skriptu (například jej pozastavit), je klientem odeslána serveru zpráva typu `program-operation`, která parametrem *operation* určuje akci, která má být provedena (viz tabulka 7).

Tabulka 7 – Změna běhu skriptu

Operace	Význam
pause	Pozastavit běh skriptu.
resume	Pokračovat v běhu skriptu.
stop	Ukončit běh skriptu.
end-after-cycle	Ukončit běh skriptu po skončení aktuálního cyklu. O tom, zda se má tato vlastnost povolit či zakázat, rozhoduje parametr *value*.
mix	Na základě parametru *value* povolí či zakáže střídání vláken.

Kód 3.25: Zpráva typu `program-operation` ve směru web. klient → server → řídicí jednotka

```
{
  "type": "program-operation",
  "data": {
    "operation": "pause"
  },
  "uuid": "35a9112e-7038-0faf-3f7b-f4b5a82e50e5"
}
```

3.4.2 Komunikace mezi řídicí jednotkou a serverem

Komunikace mezi řídicí jednotkou a serverem je postavena na protokolu TCP, což je spolehlivý, spojovaný protokol transportní vrstvy sloužící pro přenos toku bytů. Protokol zaručuje, že budou všechny byty přijaty ve správném pořadí. Jednotlivé zprávy, posílané mezi serverem a řídicí jednotkou, jsou kódovány pomocí znakové sady ASCII a každá zpráva je ukončena znakem EOL (nový řádek – konkrétně znak LF po vzoru unixových systémů).

Pro udržení spojení posílá řídicí jednotka serveru každých 60 sekund zprávu `check-connection` a server jí stejnou zprávou odpovídá. Pokud déle než 65 sekund[13] neproběhne žádná komunikace, považuje server řídicí jednotku za odpojenou. Řídicí jednotka se v tomto případě pokouší navázat znovu spojení. Tento pokus opakuje maximálně 5×, a pak přechází do offline režimu. Pokud není spojení ukončeno standardně oběma stranami příznakem FIN dle RFC793[14] a dojde k jeho přerušení, dozví se o tom obě strany až ve chvíli, kdy se jedna z nich pokusí odeslat nějakou zprávu nebo dojde po 65 sekundách od poslední zprávy k vypršení spojení.

Kód 3.26: Obousměrná zpráva typu `check-connection`

```
{
  "type": "check-connection",
  "data": { }
}
```

Díky snaze o zachování co největší konzistence protokolu je několik typů zpráv používaných v komunikaci mezi řídicí jednotkou a serverem zcela shodných se zprávami používanými v komunikaci mezi serverem a webovou aplikací s tím rozdílem, že zprávy obsahují navíc parametr *secret*, kterým se řídicí jednotka identifikuje serveru. Tato kapitola se bude na ně proto odkazovat, popřípadě doplní další informace.

3.4.2.1 Inicializace spojení

Po navázání TCP spojení mezi řídicí jednotkou a serverem posílá server zprávu typu `init-request`, jíž žádá nově připojeného klienta o aktuální data, která se od posledního spojení mohla změnit. Jedná se o tabulku obsazených úseků kolejiště, aktuální stav výhybek a stav běhu skriptu. K předání těchto dat použije řídicí jednotka totožné zprávy

[13] Limit pro přijetí odpovědi je stanoven na 5 sekund. Dlouhý limit zde není z důvodu vysoké latence komunikace (ta bývá obvykle maximálně v řádu desítek milisekund), ale z toho důvodu, že řídicí jednotka neobsahuje žádný obvod reálného času. Měření času tedy zcela závisí na přesnosti oscilátoru, který určuje frekvenci procesoru. Arduino místo přesnějšího krystalu používá jako oscilátor keramický resonátor, jehož frekvence (16 Mhz) má chabou toleranci a také je náchylný k výkyvům pracovní teploty. To vše může mít za následek nepřesnost měření času.

[14] <https://tools.ietf.org/html/rfc793>

popsané výše v kapitole 3.4.1. Jedná se o zprávy typu `section-table` (Kód 3.11), `turnout-table` (Kód 3.12) a `program-status` (Kód 3.10). Zmíněné zprávy odesílá řídicí jednotka serveru kdykoliv, kdy dojde ke změně dat, ke kterým se daná zpráva vztahuje.

Kód 3.27: Zpráva typu `init-request` ve směru server → klient

```
{
  "type": "init-request",
  "data": { }
}
```

3.4.2.2 Aktualizace skriptů a konfigurace

Dalším krokem po odeslání aktuálních dat serveru je aktualizace uživatelského nastavení a skriptů ovládání kolejiště, které se nachází v úložišti řídicí jednotky a může být již zastaralé. Nejprve řídicí jednotka odešle zprávu typu `repository` s parametrem *timestamp*, který obsahuje čas poslední změny těchto dat v unixovém formátu.

Kód 3.28: Zpráva typu `repository` ve směru klient → server

```
{
  "type": "repository",
  "data": {
    "timestamp": 1456749906
  },
  "secret": "973c3f4f01b805767c5f48d51ea5c135"
}
```

Server zprávu přijme a porovná časovou známku s časem poslední změny konfigurace. Pokud se časy neshodují, odešle řídicí jednotce zpět zprávu typu `repository` bez žádných parametrů, čímž dává najevo pokyn ke stažení aktuálních dat. V opačném případě server nic neodesílá.

Zde jsem se začal potýkat s hardwarovým omezením mikrokontroléru v řídicí jednotce, který nemá dostatek paměti SRAM pro zpracování JSON objektu tak velkého, aby obsahoval celou konfiguraci a uživatelské skripty. S rostoucí složitostí objektu se totiž zvyšuje i paměťová náročnost. Nejprve mě napadlo řešení, že by si řídicí jednotka stahovala konfigurační soubory z webového serveru pomocí protokolu HTTP. To by ale znamenalo nutnost alespoň jednoduché implementace tohoto protokolu do programu řídicí jednotky, která by pak navíc musela pro stažení každého souboru vytvořit další nové TCP spojení se serverem. Proto jsem se rozhodl pro aktualizaci skriptů a konfigurace pozměnit chování mého protokolu a také způsob zpracování dat.

Jakmile řídicí jednotka dostane zprávu typu `repository`[15], připraví se na přenos souborů dat. K tomuto účelu poskytuje protokol zprávu `get` s parametrem *file*, ve kterém řídicí jednotka specifikuje, o jaký soubor žádá. Prvním z nich je soubor confix.txt, který obsahuje veškerou konfiguraci systému a také seznam všech uživatelem vytvořených skriptů. Poté, co server tuto zprávu přijme, začne odesílat obsah daného souboru a řídicí jednotka si všechna přicházející data začne ihned ukládat na SD kartu. Přenos je ukončen dvěma byty s hodnotami 255. Je vyloučeno, že by se stejný řetězec nalézal v přenášených datech, protože je konfigurace kódovaná v ASCII a binární forma skriptů z hlediska svého návrhu nemůže obsahovat dva za sebou následující byty s hodnotami 255.

Na základě zpracované konfigurace si pak řídicí jednotka stejným způsobem stáhne ze serveru všechny aktuální skripty ovládání kolejiště a dále pokračuje v normálním běhu.

Kód 3.29: Zpráva typu `get` ve směru klient → server

```
{
  "type": "get",
  "data": {
    "file": ""
  },
  "secret": "973c3f4f01b805767c5f48d51ea5c135"
}
```

3.4.2.3 Řízení běhu skriptu

K řízení běhu skriptu se používají totožné zprávy ve směru server → klient popsané v kapitole 3.4.1.5. Jedná se o zprávy „start-program“ (Kód 3.24) a `program-operation` (Kód 3.25).

3.4.2.4 Interakce s železnicí

Pro interakci s železnicí jsou na straně řídicí jednotky implementovány zprávy `impulse` (Kód 3.15) a `report` (Kód 3.16) popsané v kapitole 3.4.1. Pro hlášení chyb a dalších událostí generuje řídicí jednotka zprávy typu `notice` (Kód 3.16). Též je zde implementována zpráva typu `default-sections` (Kód 3.17).

[15] Nemusí to být nutně v kroku bezprostředně po inicializaci. Server odesílá zprávu `repository` kdykoliv uživatel změní nastavení, vytvoří, smaže či pozmění některý skript ovládání kolejiště.

4. KAPITOLA

Prostředí webové aplikace

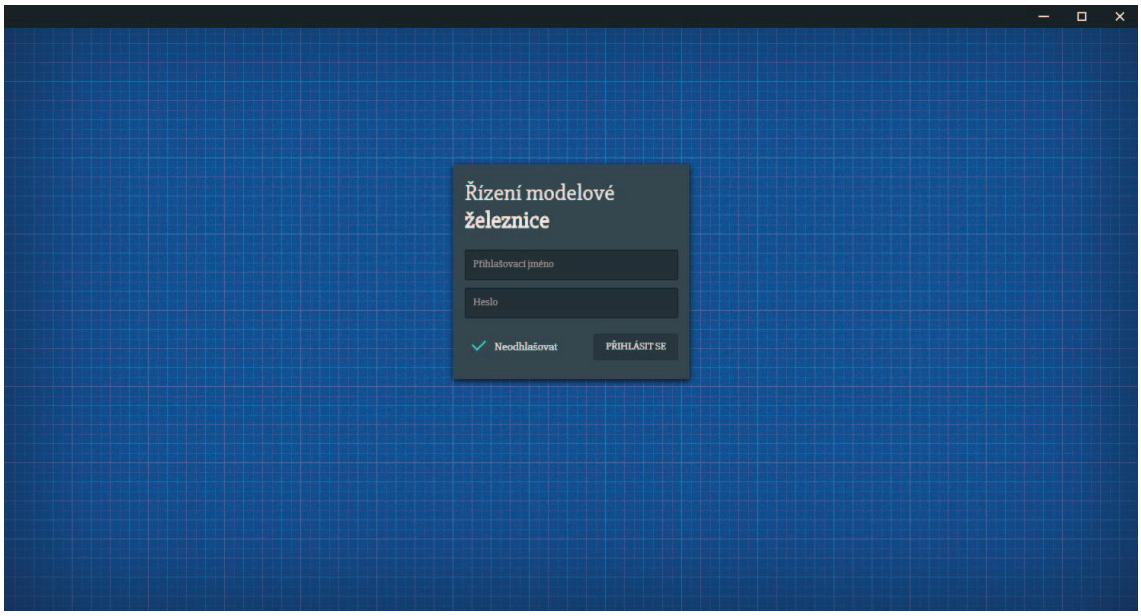
V této kapitole se zabývám popisem návrhu prostředí webové aplikace. Celkový design aplikace jsem se snažil sladit s jejím účelem a také se vztahem k technice. Nechal jsem se tedy inspirovat tzv. blueprintem, což je forma reprodukce technických nákresů, která v minulosti našla využití i pro kopírování výkresů při výrobě železničních lokomotiv. Vzhledem k povaze ovládání kolejiště jsem příliš neuvažoval o přizpůsobení webové aplikace mobilním telefonům, ale spíše jsem její využití zaměřil na stolní počítače, notebooky a tablety. Možnost používání webové aplikace na mobilním telefonu se však nevylučuje.

Webová aplikace je nasazena na adrese https://zeleznice.jakubtopic.cz/ a pro účely testování se lze do ní přihlásit pod uživatelským jménem „test“ a totožným heslem. Její možnosti jsou však značně omezeny kvůli pravděpodobně odpojené řídicí jednotce. Komunikační protokol řídicí jednotky lze ale otestovat například připojením telnetového klientu na port 3010, identifikace řídicí jednotky pomocí tajného klíče je momentálně vypnutá.

4.1 Přihlášení uživatele

Po prvním otevření webové aplikace je třeba se přihlásit přiděleným uživatelským jménem a heslem, k čemuž slouží formulář uprostřed stránky. Ten kromě polí pro zadání uživatelských údajů obsahuje též zaškrtávací políčko „Neodhlašovat“, které v případě zaškrtnutí zapříčiní uložení vygenerovaného UUID identifikátoru relace (viz komunikační protokol) do úložiště HTML5 Local Storage. Uložené UUID relace se v tomto případě použije i při příštích návštěvách a je platné po dobu 30 dnů, nebo dokud se uživatel z aplikace neodhlásí. Úložiště Local Storage jsem zvolil z toho důvodu, že se na rozdíl od běžně používaných souborů Cookies jeho data neodesílají v hlavičkách každé HTTP žádosti, což by bylo zbytečné z toho důvodu, že veškerá komunikace mé aplikace se serverem probíhá přes spojení protokolem Websocket.

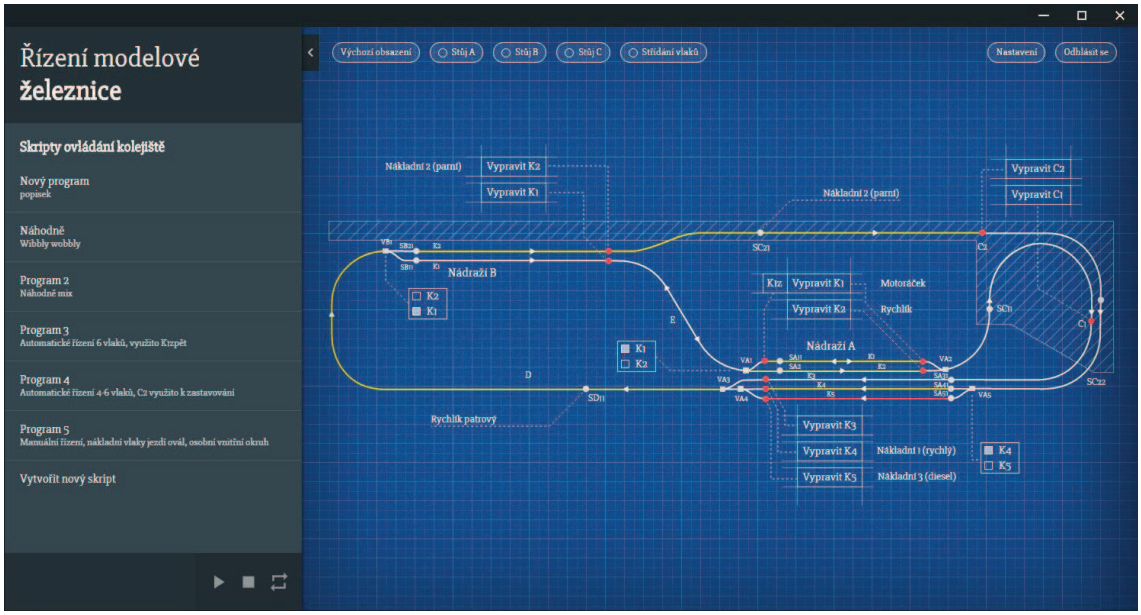
Pokud se nepodaří navázat spojení se serverem, tak zůstane tlačítko pro přihlášení zšedlé a neklikatelné a aplikace se pokouší znovu navázat spojení každých 10 sekund.



Obrázek 6 – Přihlašovací obrazovka webové aplikace

4.2 Prostředí aplikace

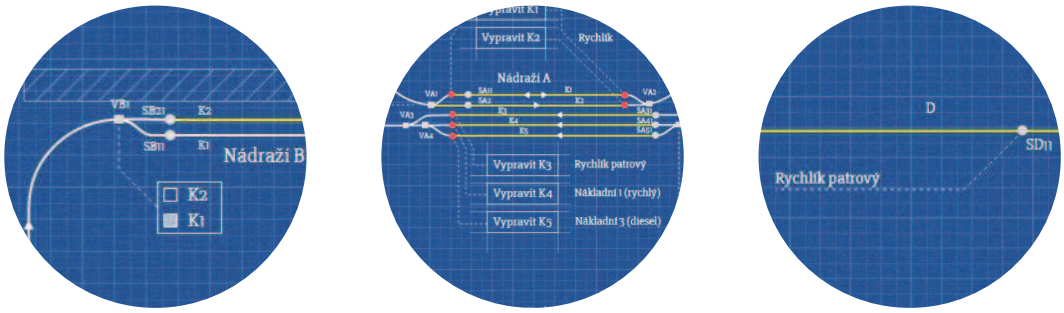
Po přihlášení se okamžitě zobrazí hlavní část prostředí aplikace, které je navrženo velmi intuitivně. Důležitou část aplikace tvoří ve stylu zmíněného blueprintu živý nákres kolejiště, který v reálném čase vizualizuje obsazené úseky, přestavování výhybek a změny signálů návěstidel. Nalevo se pak nachází skrývatelný boční panel, který obsahuje seznam všech uživatelem vytvořených skriptů a aktuální stav běžícího skriptu.



Obrázek 7 – Prostředí webové aplikace

Momentálně obsazené úseky kolejiště jsou na živém nákresu zvýrazněny žlutou barvou, úseky vyřazené z provozu jsou pak vykresleny červeně. Pokud se v úseku momentálně nachází vlak, je k němu vykreslena vodící čára s popiskem obsahující název tohoto vlaku. Výjimku tvoří koleje v nádražích a v tunelech, kde jsou tyto popisky umístěny vedle ovládacích prvků návěstidel.

U všech ovladatelných výhybek jsou k jejich přestavování vykresleny ovládací prvky s vodícími čarami k daným výhybkám na nákresu. Ovládací prvek každé výhybky je de facto blok s dvěma tlačítky, jejichž texty obsahují označení koleje, na kterou se výhybka po kliknutí na dané tlačítko přestaví. Aktuální stav výhybky je znázorněn vyplněným čtverečkem u jednoho z tlačítek a tato vizualizace je vázána na reálnou odezvu řídicí jednotky.



Obrázek 8 – Prvky nákresu kolejiště ve webové aplikaci

Obdobným způsobem je řešeno ovládání návěstidel. Ta jsou vykreslena jako body, jejichž barva vyjadřuje aktuální návěst (červená – *STŮJ*, zelená – *VOLNO*). S každým ovladatelným návěstidlem je vodící čarou spojeno tlačítko, které slouží k vyslání povelu *volno* na dané návěstidlo a tudíž k vypravení vlaku z příslušné koleje.

Nad živým nákresem kolejiště se pak nachází dvě skupiny tlačítek. Skupina nalevo obsahuje tlačítka s dalšími povely pro řídicí jednotku. Jedná se o tlačítko „Výchozí obsazení“, které resetuje stav kolejiště, tři tlačítka pro nastavení návěsti *STŮJ* na všech návěstidlech v *Nádraží A*, *Nádraží B* a v tunelech a nakonec tlačítko pro povolení či zakázání střídání vlaků. V pravém horním rohu jsou umístěna tlačítka s možnostmi pro odhlášení uživatele nebo otevření nastavení.

4.3 Nastavení

Po kliknutí na tlačítko „Nastavení“ v pravém horním rohu se zpoza pravé hrany aplikace vysune panel s nastavením, které umožnuje nastavit:

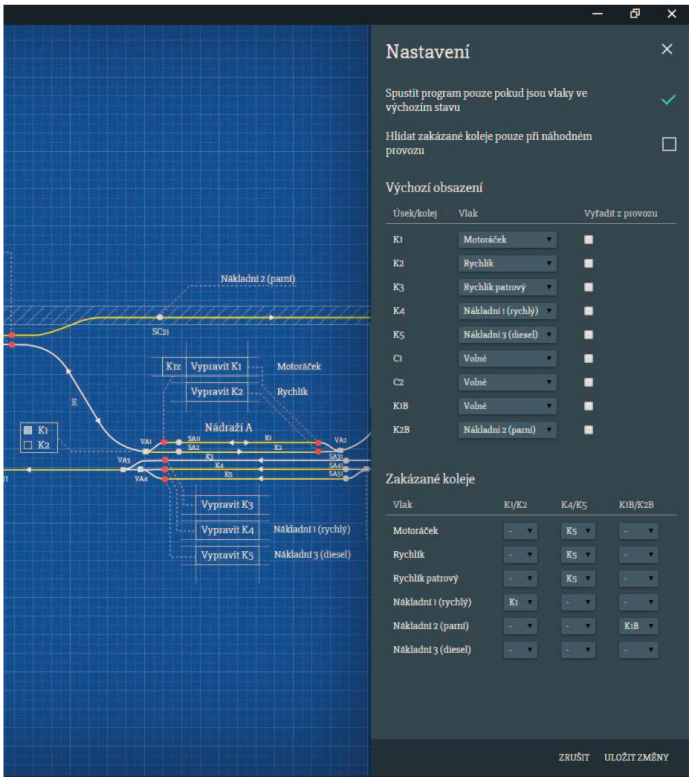
- Zda je možné spustit skripty ovládání kolejiště i v případě, že nejsou vlaky ve výchozím stavu.

- Zda se mají hlídat zakázané koleje pouze v případech, kdy se z uživatelského skriptu zpracovává příkaz „náhodně“ a řídicí jednotka se pseudonáhodně rozhoduje, který vlak pustí do volného úseku. Pokud je tato volba vypnutá, budou se zakázané koleje hlídat i během manuálního ovládání.

- Výchozí obsazení úseků – tabulka, která obsahuje seznam všech úseků končících návěstidlem a pro každý úsek umožňuje určit, který vlak v něm ve výchozím stavu stojí. Do výchozího stavu se kolejiště nastaví kliknutím na tlačítko „Výchozí obsazení“. Úseky je též možné vyřadit z provozu a řídicí jednotka je poté vnímá jako obsazené, ale zároveň z nich nelze vypravit vlak. To je užitečné zejména tehdy, kdy dojde k poruše některé lokomotivy, které poté nic nebrání zůstat stát na kolejišti – na koleji vyřazené z provozu. Ostatně i stojící vlak je hezký.

- Zakázané koleje – tabulka, která umožňuje pro každý vlak určit jednu kolej z dvojic *K1* a *K2*, *K4* a *K5*, *K1B* a *K2B*, na kterou daný vlak nebude smět vjet. Tuto funkci jsem implementoval hlavně kvůli navození reálného dojmu tím, že se při správném nastavení nestane, že by například nákladní vlak vjel na první kolej v nádraží nebo naopak osobní vlak na poslední kolej.

Uživatel má možnost provedené změny zrušit či uložit kliknutím na jedno z tlačítek dole na panelu nastavení. V případě uložení se aktuální nastavení nahraje na server a následně do řídicí jednotky. Tlačítko zrušit však vrátí nastavení do původní podoby a panel skryje. Panel lze skrýt i kliknutím na ikonku křížku v pravém horním rohu nebo kliknutím na plochu s nákresem kolejiště. V tom případě ale nedojde ke ztrátě provedených změn.



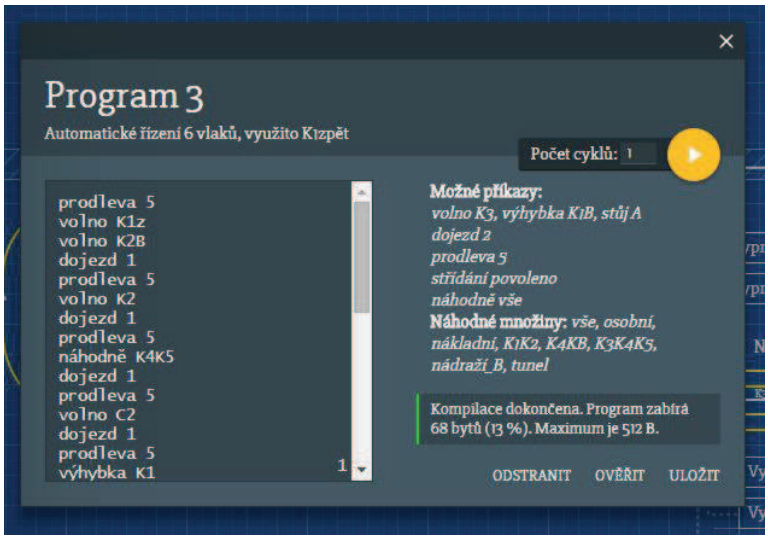
Obrázek 9 – Nastavení systému

4.4 Skripty ovládání kolejiště

Po kliknutí na jednu z položek seznamu skriptů ovládání kolejiště v levém panelu se zobrazí posouvatelné okno editoru skriptu. To slouží jako jednoduchý textový editor pro úpravu zdrojového kódu skriptu, který umožňuje i změnu jeho názvu a popisku a navíc jej umožňuje spustit s předem určeným počtem opakování (počet cyklů). V pravém dolním rohu se nacházejí tlačítka pro odstranění skriptu, ověření kódu a uložení.

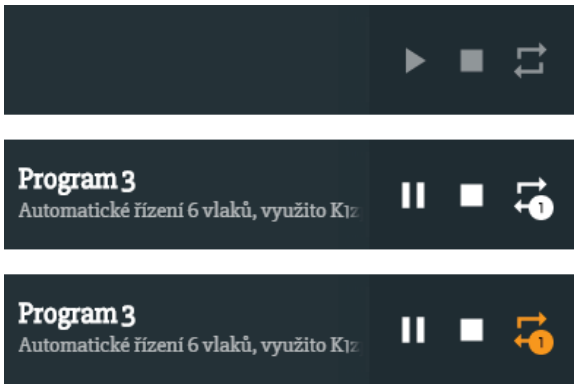
Obě tlačítka „Ověřit“ i „Uložit“ slouží k nahrání zdrojového kódu na server s tím rozdílem, že se zdrojový kód v případě kliknutí na „Ověřit“ neuloží a nenahraje se do řídicí jednotky, ale pouze proběhne pokus o jeho překlad a tudíž ověření správnosti syntaxe. V případě chybné syntaxe v kódu se zobrazí zpráva informující o chybě a jejím výskytu. Pokud je kód v pořádku, výsledkem je informace o počtu bytů, který program zabírá. Během úpravy skriptu jsou skryty ovládací prvky pro spuštění běhu. Uživatel by totiž mohl nabýt dojmu, že spouští kód, který právě vidí, ale realita by mohla být úplně jiná.

Pro vytvoření úplně nového skriptu slouží tlačítko „Vytvořit nový skript“ na konci seznamu v levém panelu.



Obrázek 10 – Okno editoru skriptu

O stavu běhu skriptu informuje malý proužek umístěný dole na levém panelu aplikace. Pokud žádný skript neběží, je prázdný a ovládací prvky jsou zšedlé. Jakmile ale uživatel spustí nějaký skript, zobrazí se zde název a popisek skriptu a uživatel má možnost použitím přítomných ovládacích prvků pozastavit či pokračovat v běhu skriptu, nebo běh úplně zastavit. Třetí tlačítko zobrazuje počet zbývajících cyklů a slouží pro pokyn, že se má běh skriptu ukončit po skončení aktuálního cyklu. Pokud je tato volba aktivní, je tlačítko zbarveno oranžově. Opětovným stisknutím tlačítka lze volbu deaktivovat.



Obrázek 11 – Ovládání běhu skriptu

5. KAPITOLA

Dokumentace

5.1 Dokumentace webové aplikace

K dokumentaci JavaScriptového kódu, tj. K serverové a klientské části webové aplikace, jsem použil nástroj JSDoc Toolkit ve verzi 2.4.0. Tento nástroj slouží k automatickému generování dokumentace ve formě provázaných HTML dokumentů na základě značek v komentářích zdrojového kódu JavaScriptu.

Syntaxe značek, které JSDoc používá, je velmi podobná struktuře komentářů nástroje JavaDoc, který slouží k dokumentaci kódu psaném v jazyce Java. Liší se však tím, že je přizpůsobená specifickému chování JavaScriptu[16].

Obrázek 12 – Ukázka dokumentace vygenerované nástrojem JSDoc Toolkit

[16] Např. dynamické typování proměnných

Zde uvádím některé z více používaných JSDoc značek:

@constructor – Označí funkci jako konstruktor.

@deprecated – Označí funkci jako zastaralou.

@param – Formální parametr funkce. Před název parametru je vhodné do složených závorek napsat jeho datový typ. Ten totiž nelze z kódu nijak určit.

@return, **@returns** – Návratová hodnota funkce. Stejně jako u parametrů funkce je zde vhodné určit návratový typ funkce a hodnotu opatřit komentářem.

@private – Značí, že se jedná o soukromý člen.

@exception, **@throws** – Dokumentuje výjimku, která by mohla být funkcí vržená.

@this – Určuje, k čemu se v kontextu funkce vztahuje klíčové slovo *this*.

@event – Dokumentuje událost.

@fires, **@emits** – Popisuje události, které by mohla funkce spustit.

@listens – Definuje seznam událostí, kterým následující funkce naslouchá.

@author – Jméno vývojáře.

@version – Číslo verze položky.

Kód 5.30: Příklad užití JSDoc k dokumentaci konstruktoru.

```
/**
 * Enables dragabble functionality on script editor windows
 *
 * @constructor
 * @memberof window
 * @param {string} handle - ID of draggable window‘s handle
 * @param {string} window - ID of draggable window
 * @this {Draggable}
 */
function Draggable(handle, window) {
  ...
}
```

5.2 Dokumentace programu pro mikrokontrolér

Program pro Arduino v řídicí jednotce jsem okomentoval obdobně jako JavaScriptový kód. Pomocí nástroje Doxygen, který se obvykle používá pro generování dokumentace k C++, pro něj lze vygenerovat podobnou webovou dokumentaci.

Kód 5.31: Příklad dokumentace metody

```
/**
 * Auxiliary method for the resolving of restricted/disabled sections for the
 * particular trains
 *
 * @param  rnd     Indicates whether the calling is a part of a random selection
 * @param  trainID ID of train staying on the certain section
 * @param  col     Column in the table of the restricted tracks/sections
 * @param  track   Destination track
 * @return         True if not restricted/disabled
 */
bool restrictedRes(bool rnd, uint8_t trainID, uint8_t col, uint8_t track) {
  ...
}
```

Na rozdíl od dokumentace JavaScriptového kódu pomocí JSDoc zde není nutné explicitně psát datové typy formálních parametrů, protože je generátor dokumentace určí ze zdrojového kódu.

6. KAPITOLA

Ověření funkčnosti

Vzhledem k tomu, že jsem neměl k modelové železnici neomezený přístup, musel jsem se během vývoje často spoléhat na samotnou řídicí jednotku. Výrazně mi při testování funkčnosti pomohla diagnostická hlášení mého programu pro mikrokontrolér posílaná sériovým portem, indikační LED diody na výstupech řídicí jednotky (viz Obrázek 20), které jsem pro indikaci generovaných impulzů přidal přímo na desku s optočleny, a mikrospínače na vstupech, kterými jsem simuloval průjezd vlaku spínací kolejí. I tak bylo ale nutné provést před finálním nasazením mnoho testů celého systému s připojenou modelovou železnicí.

6.1 Statická analýza zdrojového kódu

Během vývoje webové aplikace byla průběžně prováděna statická analýza JavaScriptového zdrojového kódu za pomoci open-sourcového nástroje JSHint[17], který jsem si nainstaloval přímo do textového editoru. Tento nástroj mi pomohl odhalit některé chyby, jako je například deklarace nevyužitých proměnných či přebytečných metod, a zvýšit přehlednost kódu doporučením některých konvencí. JSHint dokáže mimo jiné určit i cyklomatickou složitost kódu.

[17] <http://jshint.com/about>

Závěr

Závěrem své práce bych chtěl zhodnotit přínos projektu. Jsem spokojen s funkčností systému řízení, který splňuje všechny mé požadavky, jež jsem si na začátku vývoje stanovil. Webová aplikace v reálném čase plynule vizualizuje celkový stav kolejiště (stav výhybek, návěstidel, obsazenost úseků) a umožňuje jeho manuální řízení přes Internet.

K dispozici je i jednoduchý editor, který slouží pro tvorbu skriptů, což jsou sekvence jednoduše navržených příkazů pro ovládání kolejiště, jež lze opakovaně spouštět a tím železnici automatizovat. Mistrovským kouskem je v tomto ohledu příkaz „náhodně“, při jehož použití si sama řídicí jednotka může náhodně vybrat, jaký vlak pustí do jakého volného úseku. Železnice se tedy může chovat naprosto nepředvídatelně při zachování maximální bezpečnosti. To je umožněno tím, že řídicí jednotka ví, které koleje jsou obsazené a které volné a i při manuálním ovládání dokáže zabránit jakékoliv kolizi vlaků.

Díky tomu, že je režie veškeré logiky implementovaná na řídicí jednotce, je do jisté míry možné používat samotnou řídicí jednotku i s odpojeným serverem bez použití webové aplikace. V tomto případě však řídicí jednotka umožňuje pouze běh předem vytvořených skriptů ovládání kolejiště.

Projekt v tuto chvíli považuji za dokončený, ale do budoucna by bylo možné se například zaměřit na větší modulárnost a tím vytvořit univerzální řídicí systém pro jakoukoliv analogovou modelovou železnici. Také by bylo možné implementovat algoritmy pro nalezení nejkratší cesty v grafu ve spojení s metodou prohledávání do hloubky, což by mohlo umožnit automaticky přemístit vlaky po kolejišti z jakéhokoliv stavu do daného výchozího stavu.

Práce pro mě byla velkým osobním přínosem, neboť jsem nabyl cenných zkušeností s nově vznikající platformou Node.js, s tvorbou realtimových webových aplikací, bezpečnou konfigurací webového serveru, návrhem síťové komunikace a v neposlední řadě jsem si rozšířil znalosti programování. Realizace projektu mě velice bavila a v každé fázi bylo zábavné pozorovat, jak železnice reaguje. Práce na tomto projektu byla pro mě velmi zajímavá, a proto mám do budoucna v plánu se Internetu věcí i nadále věnovat a pokusit se vytvořit vlastní všeobecnou platformu pro toto nově vznikající odvětví.

Seznam použité literatury

[1] TEIXEIRA, Pedro Dennis. Professional node.js: building javascript based scalable software. 1st ed. Indianapolis, IN: Wiley Pub., Inc., 2012. ISBN 1118185463.

[2] Node.js Foundation. Node.js v5.8.0 Documentation. [online]. [cit. 2016-02-12]. Dostupné z: http://nodejs.org/api/all.html

[3] CASTRO, Elizabeth a Bruce HYSLOP. HTML5 a CSS3: názorný průvodce tvorbou WWW stránek. 1. vyd. Brno: Computer Press, 2012. ISBN 978-80-251-3733-8.

[4] FLANAGAN, David. JavaScript: the definitive guide. Fifth edition. Sebastopol: O´Reilly, 2006. ISBN 0-596-10199-6.

[5] ASHTON, Kevin. That 'Internet of Things' Thing. In: RFID Journa [online]. 2009 [cit. 2016-02-01]. Dostupné z: http://www.rfidjournal.com/articles/view?4986

[6] Arduino Language Reference [online]. [cit. 2016-02-12]. Dostupné z: http://arduino.cc/en/Reference/HomePage

[7] C Library Reference [online]. [cit. 2016-02-12]. Dostupné z: http://www.cplusplus.com/reference/clibrary/

[8] Ubuntu Wiki [online]. [cit. 2016-02-12]. Dostupné z: http://wiki.ubuntu.cz/

[9] Official Ubuntu Documentation [online]. [cit. 2016-02-12]. Dostupné z: https://help.ubuntu.com/

[10] I. Fette, Google Inc., A. Melnikov a Isode Ltd. The WebSocket Protocol [online]. 2011 [cit. 2016-02-12]. Dostupné z: https://tools.ietf.org/html/rfc6455

[11] TRANSMISSION CONTROL PROTOCOL [online]. 1981 [cit. 2016-02-12]. Dostupné z: https://tools.ietf.org/html/rfc793

Seznam obrázků

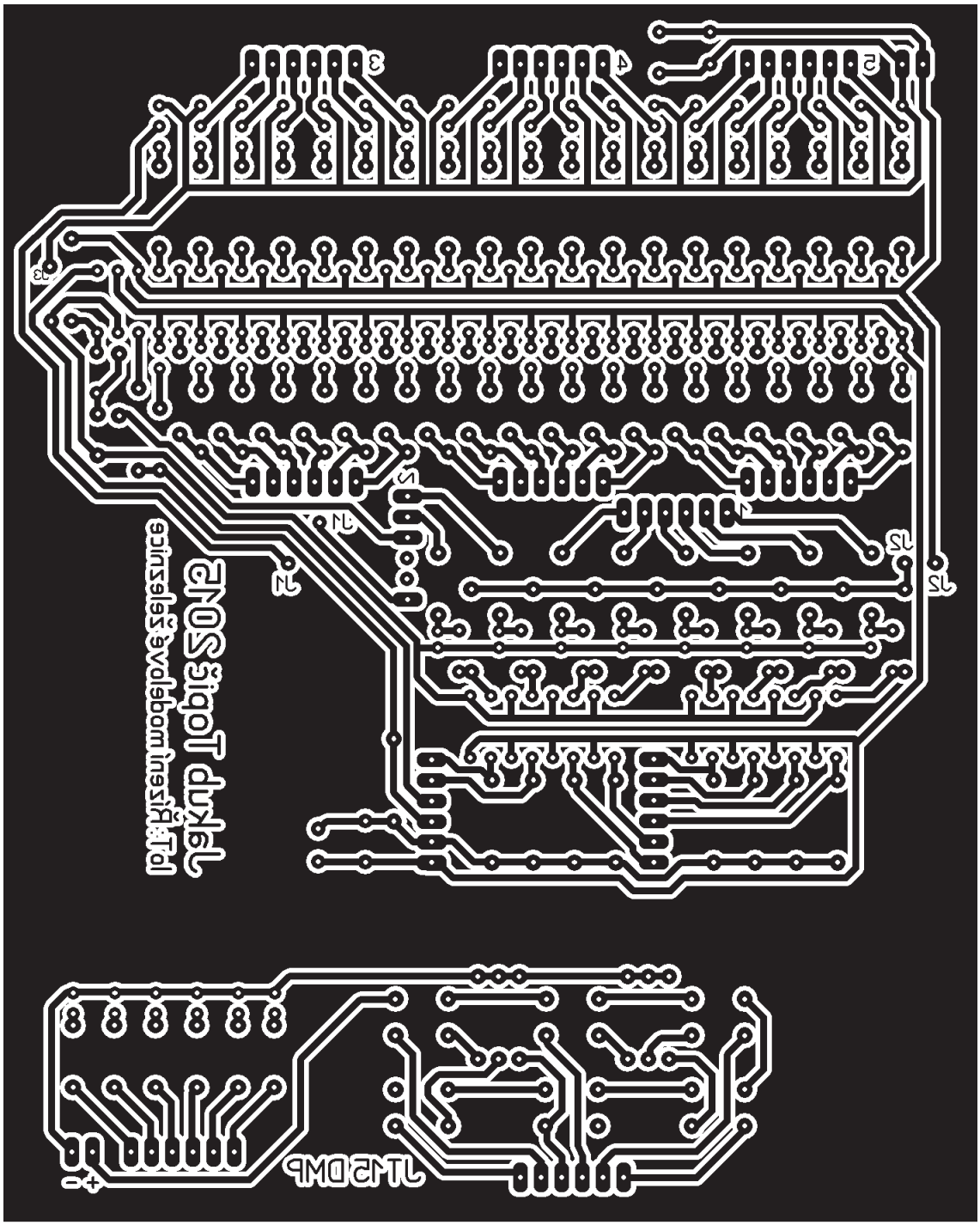
Seznam tabulek

Seznam použitých zkratek

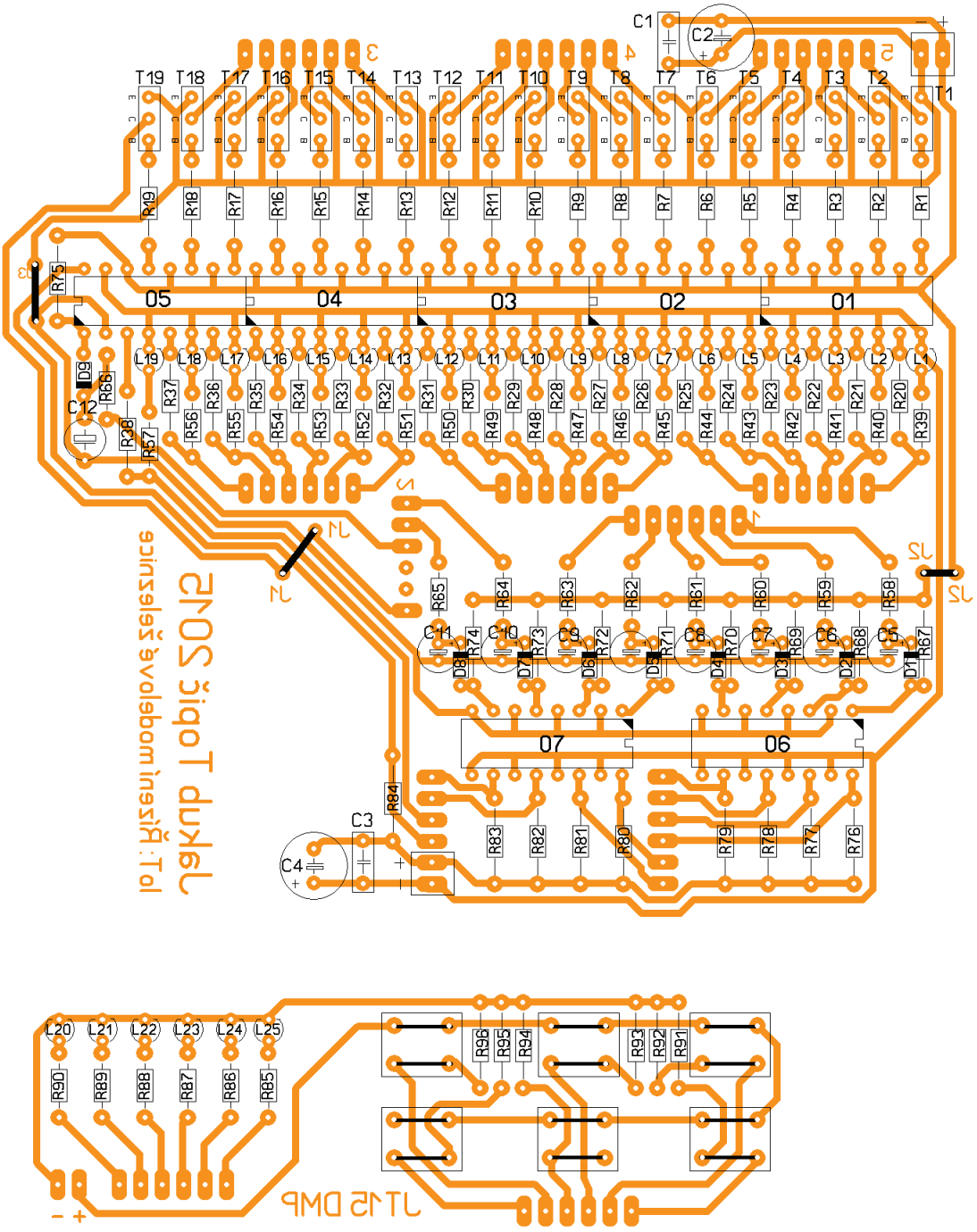
AJAJ	Asynchronous JavaScript and JSON
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BSD	Berkeley Software Distribution
CSS	Cascading Style Sheets
DHCP	Dynamic Host Configuration Protocol
DOM	Document Object Model
EEPROM	Electrically Erasable Programmable Read-Only Memory
EOL	End of Line
FAT	File Allocation Table
GND	Ground
GNU	GNU's Not Unix!
GPIO	General-Purpose Input/Output
HSTS	HTTP Strict Transport Security
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
JS	JavaScript
JSON	Javascript Object Notation
LCD	Liquid-Crystal Display
LED	Light-Emitting Diode
LTS	Long-Term Support
MISO	Master In Slave Out
MOSI	Master Out Slave In
NC	Not Connected
OS	Operating System

PWM	Pulse Width Modulation
RFC	Request For Comments
RX	Reception
SCL	Serial Clock Line
SS	Slave Select
SD	Secure Digital
SDA	Serial Data Line
SRAM	Static Random Access Memory
SSL	Secure Sockets Layer
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TX	Transmission
UTF	Unicode Transformation Format
UUID	Universally Unique Identifier
VPS	Virtual Private Server
W3C	World Wide Web Consortium
WHATWG	Web Hypertext Application Technology Working Group
WWW	World Wide Web
XHTML	Extensible Hypertext Markup Language

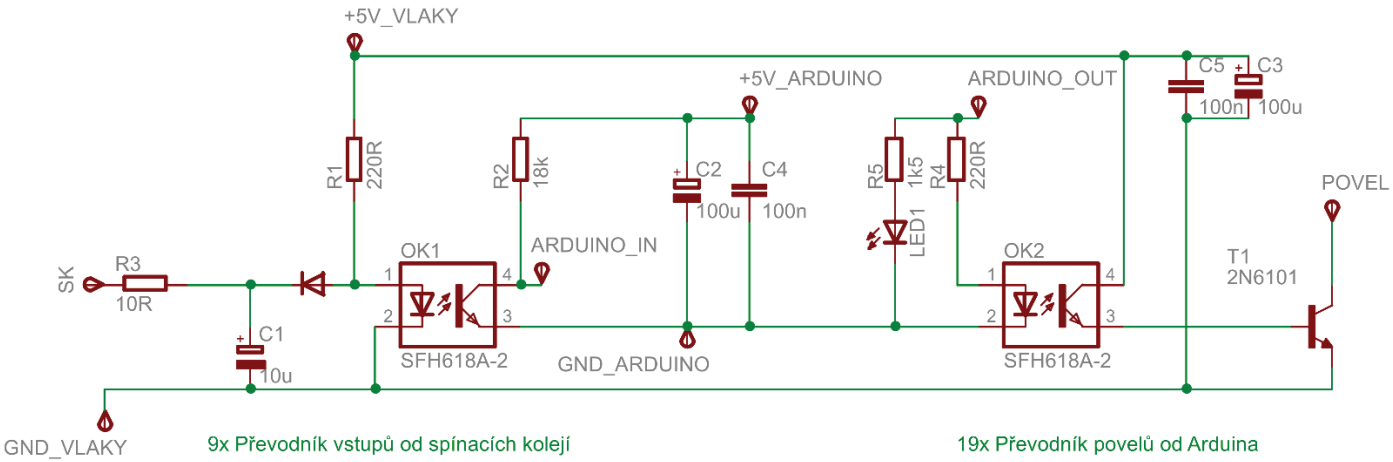
Přílohy



Obrázek 13 – Návrh desek plošných spojů (Řídicí jednotka)



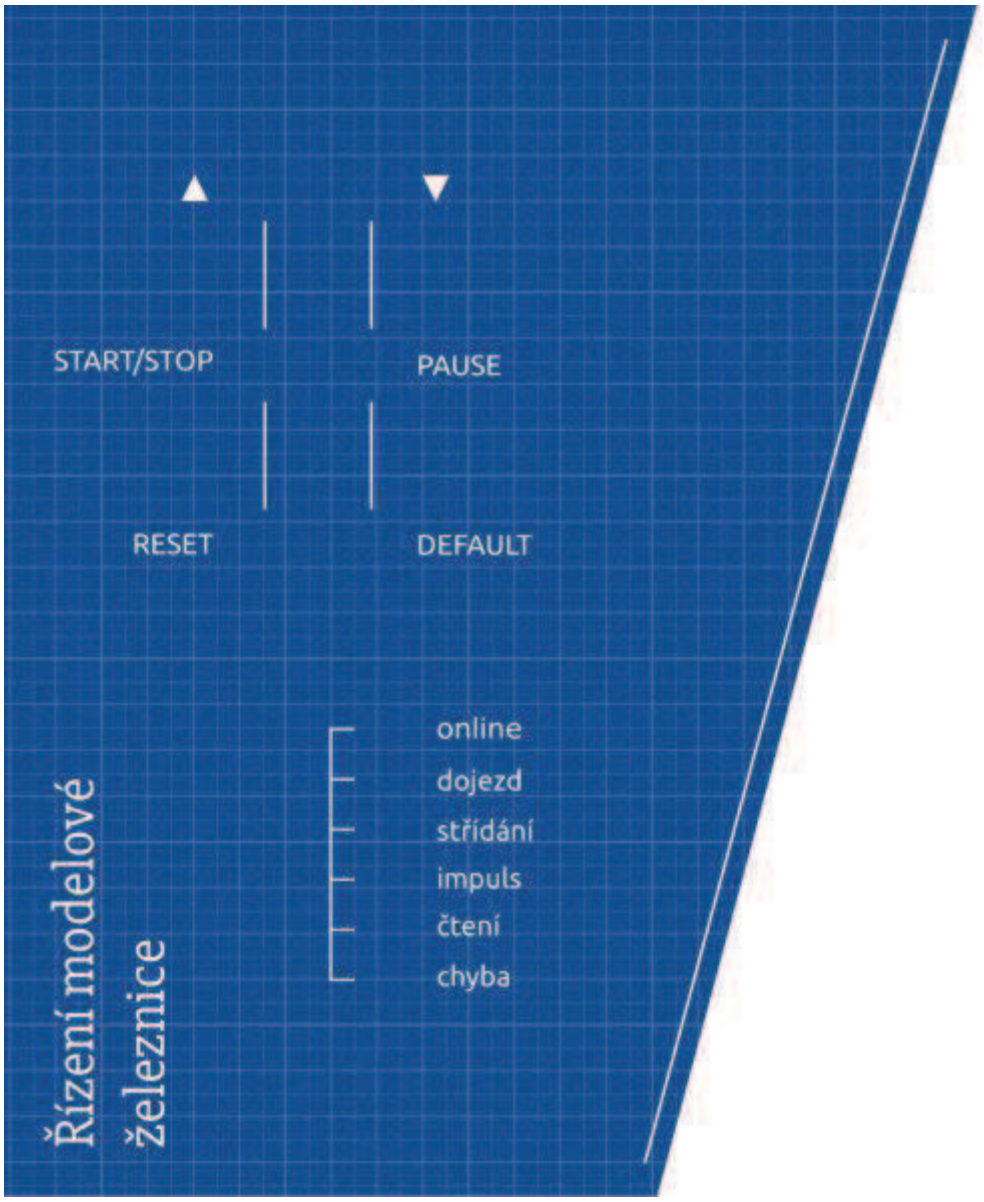
Obrázek 14 – Osazovací plán (Řídicí jednotka)



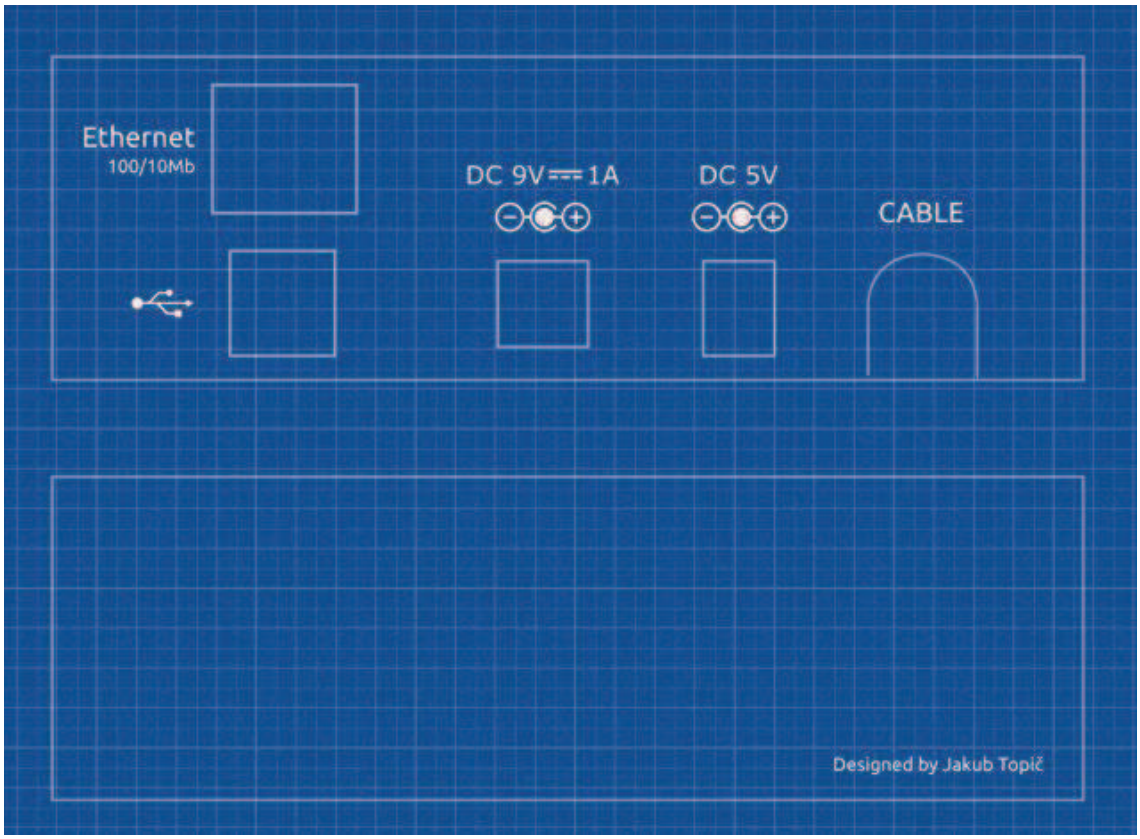
Obrázek 15 – Zjednodušené schéma rozhraní mezi GPIO mikrokontroléru a konektorem řídicího panelu

Tabulka 8 – Rozpiska součástek (Řídicí jednotka)

Pol.	Název součástky	Označení	Hodnota	Kusů	Poznámka
1	Kondenzátor	C1,C3	100u	2	Keramický
2	Kondenzátor	C2,C4	100n	2	Elektrolytický
3	Kondenzátor	C5-C12	10u	8	Elektrolytický
4	Dioda	D1-D9	1N4148	9	Dioda 75V/0,2A, DO35
5	LED	L1-L19, L25	LED 3MM RED 120/60° L-3N4SRD	20	LED 3mm červená, 60°
6	LED	L20-L24	LED 3MM CYL GREEN 7/130°	5	3mm zelená, 130°
7	Optočlen	O1-O7	PC844	7	4 x optočlen s tranzistorem
8	Rezistor	R1-R19, R58-66	RU 10R	19	RU 10R 0207 0,25W 5%
9	Rezistor	R20-R38, R67-75, R85-R90	RU 220R	34	RU 220R 0207 0,25W 5%
10	Rezistor	R39-R57	RM 1k5	19	RM 1k5 0204 0,4W 1% HI-TANO
11	Rezistor	R76-84	RU 18k	18	RU 18K 0207 0,25W 5%
12	Rezistor	R91-96	RU 10k	6	RU 10K 0207 0,25W 5%
13	Tranzistor	T1-T19	BD237 TO126	19	Bipolární NPN tranzistor
14	Mikrospínač	TL1-TL6	TC-0108-T	6	Mikrospínač do DPS
15	LCD	LCD	MC1602E-SBL/H	1	Alfanumerický LCD displej s řadičem HD44780
16	Napájecí konektor		DS-214B	1	Napájecí souosý konektor, 6,3mm

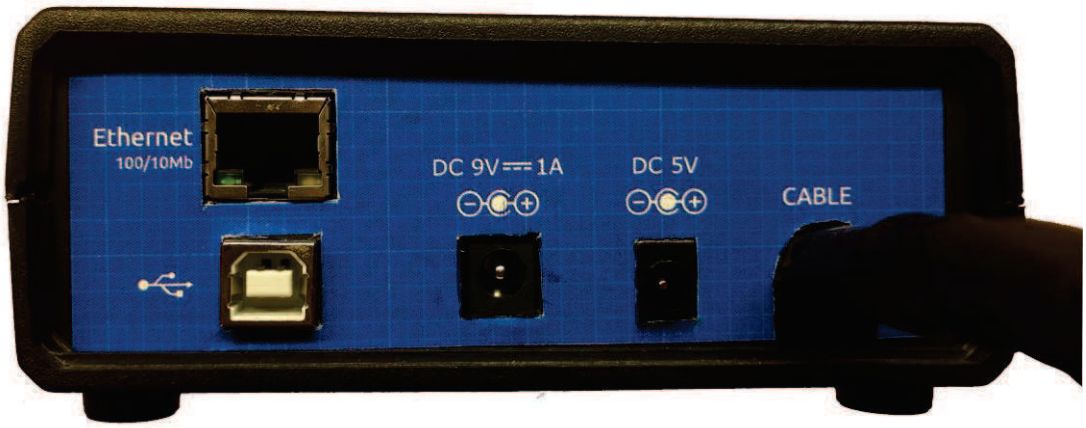


Obrázek 16 – Potisk řídicí jednotky (Horní strana)

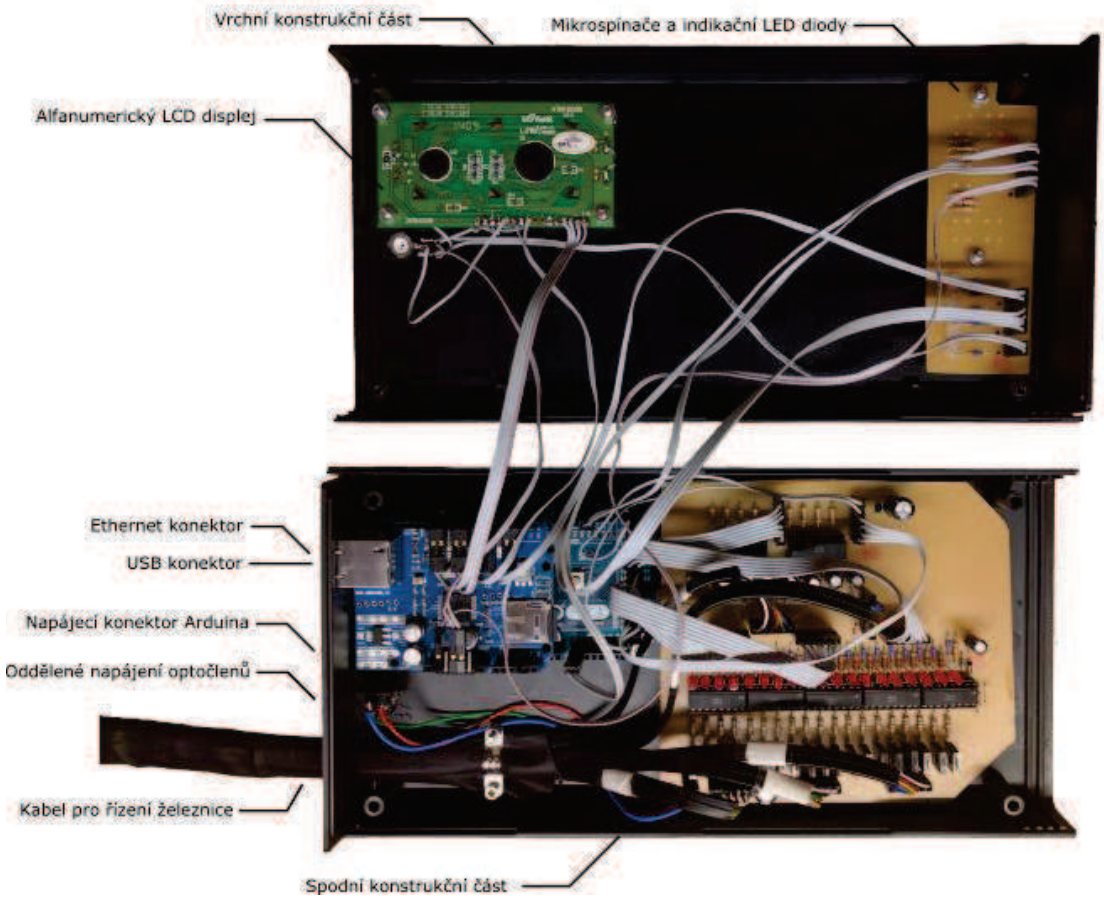


Obrázek 17 – Potisk řídicí jednotky (Boční strany)

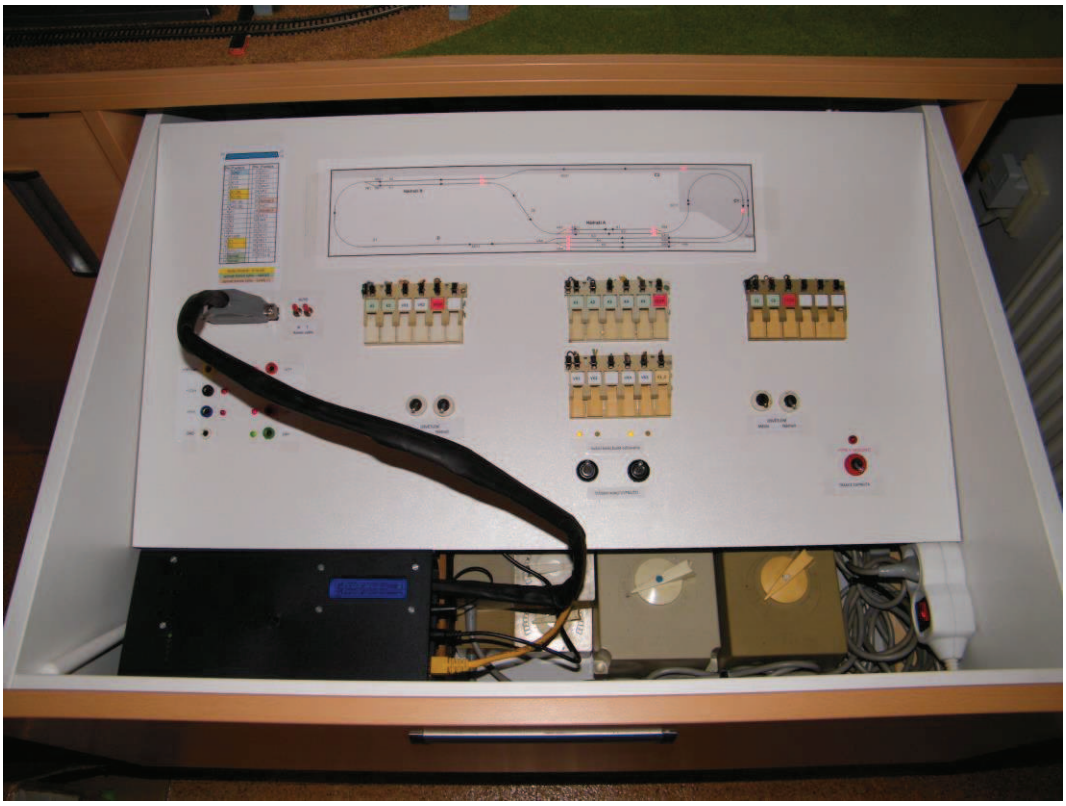
Obrázek 18 – Řídicí jednotka



Obrázek 19 – Boční strana řídicí jednotky



Obrázek 20 – Konstrukce řídicí jednotky



Obrázek 21 – Řídicí jednotka (bez potisku) připojená k původnímu ovládacímu panelu modelové železnice

Obrázek 22 – Pohled na celou modelovou železnic