VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Vizualizace jízdy elektromobilu s výpočtem spotřeby

Visualization of Electric Car Journeys with Consumption Calculation

Bc. Tomáš Kapusňák

Diplomová práce

Vedoucí práce: Ing. David Seidl, Ph.D.

Ostrava, 2025

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/24/135159
vsboee2403E7A3

Zadání diplomové práce

Student: **Bc. Tomáš Kapusňák**

Studijní program: N0613A140034 Informatika

Téma: Vizualizace jízdy elektromobilu s výpočtem spotřeby
Visualization of Electric Car Journeys with Consumption Calculation

Jazyk vypracování: čeština

Zásady pro vypracování:

Použijte vybraný programovací jazyk, framework a knihovny k vytvoření komplexní webové aplikace, která bude schopna získávat, filtrovat a zobrazovat data o jednotlivých jízdách elektromobilů na interaktivní mapě. Aplikace by měla být rozdělena do tří hlavních částí: databázová vrstva, serverová logika a klientská část. Klientská část by měla obsahovat interaktivní vizualizační komponentu, která umožní zobrazovat detailní informace o trasách pomocí grafů.

Prozkoumejte možnosti integrace simulačních výpočtů, které mohou být implementovány buď jako samostatná komponenta nebo prostřednictvím externího API. Navrhněte algoritmus, který na základě fyzikálních zákonů popisujících pohyb elektromobilu, dokáže simulovat jízdu podle zadaných parametrů.

1. Popis a zdůvodnění zvolení programovacího jazyka, frameworku a knihoven.
2. Popište formát dat využívaných při výpočtu spotřeby.
3. Vytvořte a popište fyzikální model schopný počítat spotřebu elektromobilu.
4. Vytvořte webovou aplikaci sloužící k vizualizaci jízd a vypočtené spotřeby.
5. Proveďte testování modelu vůči testovacím scénářům.
6. Srovnejte dosažené výsledky s podobnými aplikacemi.

Seznam doporučené odborné literatury:

[1] JIMENEZ BERMEJO, David; HERNANDEZ, Sara; FRAILE-ARDANUY, Jesus; SERRANO ROMERO, Javier; POZO, Rubén; ALVAREZ, Federico. Modelling the Effect of Driving Events on Electrical Vehicle Energy Consumption Using Inertial Sensors in Smartphones. Energies. 2018-02, roč. 11, s. 412. Dostupné z doi: 10.3390/en11020412.
[2] FOTOUHI, Abbas; AUGER, Daniel; PROPP, Karsten; LONGO, Stefano. Simulation for pre-diction of vehicle efficiency, performance, range and lifetime: A review of current techniques and their applicability to current and future testing standards. In: 2014-11. Dostupné z doi: 10.1049/cp.2014.0959.
[3] GENIKOMSAKIS, Konstantinos N.; MITRENTSIS, Georgios. A computationally efficient si-mulation model for estimating energy consumption of electric vehicles in the context of route planning applications. Transportation Research Part D: Transport and Environment. 2017, roč. 50, s. 98–118. issn 1361-9209. Dostupné z doi: https://doi.org/10.1016/j.trd.2016.10.014.

17. listopadu 2172/15
708 00 Ostrava-Poruba
Česká republika

spojovatelka: +420 597 321 111
epodatelna: epodatelna@vsb.cz
ID datové schránky: d3kj88v

IČ: 61989100
DIČ: CZ61989100

email: studijni.fei@vsb.cz
www.fei.vsb.cz

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Seidl, Ph.D.**

Datum zadání: 01.09.2024

Datum odevzdání: 30.04.2025

Garant studijního programu: prof. RNDr. Václav Snášel, CSc.

V IS EDISON zadáno: 07.11.2024 09:15:04

Abstrakt

Tato diplomová práce se zabývá návrhem a vývojem webové aplikace sloužící k interaktivnímu plánování trasy s vizualizací výsledné spotřeby energie a dalších parametrů trasy. Součástí práce je vytvoření fyzikálního modelu určeného k výpočtu spotřeby energie během jízdy elektromobilů, který zohledňuje fyzikální vlivy, jako jsou aerodynamický odpor, valivý odpor, hmotnost vozidla či topografie trasy. Práce dále zdůvodňuje zvolení využitého programovacího jazyka, frameworku a použitých knihoven. Součástí je také detailní popis vstupních dat využívaných při výpočtu spotřeby, jejich struktura a způsob získání. Výstupem práce je funkční webová aplikace doplněná o API rozhraní umožňující efektivní využití fyzikálního modelu. Model byl testován na předem definovaných testovacích scénářích a výsledky byly porovnány s obdobnými existujícími řešeními pro ověření jeho přesnosti a praktické využitelnosti. Práce se rovněž zaměřuje na možnosti budoucího rozšíření modelu o další fyzikální vlivy a vylepšení uživatelského rozhraní pro širší využitelnost výsledné aplikace.

Klíčová slova

Spotřeba energie; Elektromobilita; GPS; Fyzikální model

Abstract

This diploma thesis focuses on the design and development of a web application for interactive route planning, including the visualization of energy consumption and other route parameters. A key part of the work is the creation of a physical model intended for calculating energy consumption during electric vehicle operation. The model takes into account various physical factors, such as aerodynamic drag, rolling resistance, vehicle weight, and route topography. The thesis also provides justification for the choice of programming language, framework, and libraries used. It includes a detailed description of the input data formats utilized in the consumption calculations, their structure, and methods of collection. The outcome of the work is a fully functional web application, supplemented by an API interface that allows efficient use of the developed physical model. The model was tested using predefined test scenarios, and the results were compared with those of existing solutions to verify its accuracy and practical applicability. The thesis also explores potential future extensions of the physical influences considered by the model and possible improvements to the application's user interface for broader usability.

Keywords

Energy consumption; Electromobility; GPS; Physical model

Poděkování

Děkuji mému vedoucímu diplomové práce, panu Ing. Davidu Seidlovi Ph.D., za jeho odborné vedení, podnětné rady, nápady, připomínky a čas věnovaný konzultacím při vzniku této práce a tvorbě veškerých testovacích jízd.

Obsah

Seznam použitých zkratek a symbolů

GPS	–	Global Positioning System
API	–	Application Programming Interface
ABRP	–	A Better Routeplanner
UNECE	–	United Nations Economic Commission for Europe
WLTP	–	Worldwide Harmonized Light-Duty Vehicles Test Procedure
UX	–	User Experience
HTML	–	Hypertext Markup Language
CSS	–	Cascading Style Sheets
PHP	–	Hypertext Preprocessor
PWA	–	Progressive Web Application
NPM	–	Node Package Manager
GIS	–	Geographic Information System
SQL	–	Structured query language
MVCC	–	Multiversion concurrency control
OBD	–	On-board diagnostics
JWT	–	JSON Web Token
JSON	–	JavaScript Object Notation
ORM	–	Objektově Relační Mapování
ID	–	Identification
CSV	–	Comma Separated Value
SOC	–	State Of Charge
SOH	–	State Of Health

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Plánování jízd je běžnou součástí každodenního života většiny z nás. Ať už se jedná o cestu do zaměstnání, výlet mimo město nebo dovolenou, vždy se rozhodujeme, jakou trasu zvolíme a jakým způsobem pojedeme. U konvenčních vozidel se spalovacím motorem je tento proces zpravidla jednoduchý – objem palivové nádrže a dostupnost čerpacích stanic umožňuje poměrně flexibilní pohyb bez nutnosti složitého plánování a to i v rámci cest po Evropě. Oproti tomu elektromobily představují zcela nový pohled, neboť jejich dojezd je omezen kapacitou baterie, a doba dobíjení je momentálně výrazně delší než čas na tankování.

S rostoucím podílem elektromobilů na evropském i světovém trhu se však tato problematika stává stále aktuálnější. Podle dat zveřejněných na portálu acea.auto [1] bylo v roce 2023 v Evropské unii zaregistrováno více než 10,5 milionu nových vozidel, z čehož elektromobily tvořily 22,7 % podle údajů eea.europa.eu [2]. Oproti předchozímu roku se jedná o nárůst o více než 37 %. To ukazuje výrazný růstový trend. Vzhledem k tomuto faktu lze očekávat, že elektromobily budou v budoucnu hrát stále významnější roli v individuální dopravě a tím i vzroste potřeba nástrojů, které uživatelům pomohou efektivněji plánovat jejich cesty a předcházet situacím spojeným s nedostatečným dojezdem.

Cílem této diplomové práce je navrhnout a implementovat nástroj, který na základě definovaných vstupních parametrů (trasa, vozidlo, atd.) dokáže odhadnout spotřebu elektrické energie a vizualizovat průběh celé simulované jízdy. Klíčovým prvkem tohoto nástroje je fyzikální model, jenž vychází z principů klasické mechaniky a bere v úvahu faktory, jako jsou valivý odpor, odpor vzduchu, sklon trasy, zrychlení či účinnost jednotlivých komponent elektromobilu. Model je navržen tak, aby z dostupných dat (např. GPS souřadnic, nadmořské výšky, rychlosti) dokázal v reálném čase vypočítat energetickou náročnost úseku trasy a následně celkové spotřebované množství elektrické energie.

Součástí této práce je také návrh a realizace webové aplikace, která umožní uživateli interaktivně plánovat jízdy, zadávat potřebné parametry a přehledně zobrazit výstupy simulace v podobě grafu. Aplikace bude navržena s důrazem na uživatelskou přívětivost, responsivitu a škálovatelnost. Bude rozdělena do tří hlavních částí – klientská (frontend) část, zajišťující interakci s uživatelem, serverová

(backend) část, realizující logiku systému a výpočty, a databázová část, sloužící k ukládání dat o vozidlech nebo také o uživatelích a jejich preferencích.

Významnou součástí práce je i popis datového formátu, který bude použit pro reprezentaci jednotlivých tras a vozidel, včetně způsobu získávání těchto dat – ať už z reálných jízd, nebo pomocí externích navigačních API. Fyzikální model bude podroben testování na sadě reálných dat získaných z elektromobilů prostřednictvím aplikací Leaf Spy Pro [3] a Sensor Logger [4]. Získané výsledky budou porovnány s reálnou spotřebou vozidel a s výsledky jedné z již existujících aplikací, které slouží k plánování tras elektromobilů, jako je například A Better Route Planner [5] nebo EVNavigation [6].

Výsledkem této diplomové práce bude funkční, modulární a přesný systém, který bude možné dále rozvíjet a nasadit do reálného provozu. Práce tak může přispět nejen k praktickému využití v oblasti elektromobility, ale zároveň otevírá cestu pro další výzkum v oblasti simulace jízdních profilů a optimalizace spotřeby elektrické energie.

Kapitola 2

Současný stav poznání

Problematice simulace spotřeby elektromobilů se věnovala celá řada výzkumných i vývojových týmů, přičemž výsledky těchto snah byly různě úspěšné z hlediska přesnosti i praktické využitelnosti. Cílem této kapitoly je představit vybrané přístupy a metody, které se zaměřily na modelování energetické spotřeby elektromobilů a dosáhly pozoruhodných výsledků.

Existuje více přístupů k řešení této problematiky – od jednoduchých statických modelů vycházejících z historických dat, přes fyzikálně založené simulace zahrnující podrobné modely odporových sil, až po pokročilé metody využívající strojové učení a analýzu rozsáhlých datových sad.

V následujících podkapitolách je uveden výběr odborných článků, které prezentují inovativní přístupy k modelování spotřeby. Následně budou představeny dostupné softwarové nástroje a aplikace, které nabízejí podobnou funkcionalitu jako výsledná aplikace vytvořená v rámci této práce.

2.1 Rešerše literatury

V odborné literatuře lze nalézt řadu článků a studií, které se zabývají problematikou modelování spotřeby energie u elektromobilů. V této části je uveden výběr relevantních publikací, které představují zajímavé přístupy k řešení této problematiky.

Modelling the Effect of Driving Events on Electrical Vehicle Energy Consumption Using Inertial Sensors in Smartphones

Autoři: David Jimenez Bermejo, Sara Hernandez, Jesus Fraile-Ardanuy, Javier Serrano Romero, Rubén Pozo, Federico Alvarez

David Jimenez Bermejo se spoluautory v publikaci [7] představili přístup pro modelování spotřeby energie elektromobilů založený na jízdních událostech. Pro jejich zaznamenávání využili inerciální senzory pohybu dostupné v chytrých telefonech. Autoři kladli důraz na skutečnost, že úkony,

jako je akcelerace, brzdění nebo prudké zatáčení, mají významný vliv na energetické požadavky vozidla. Pomocí hlubokých neuronových sítí vytvořili model, který analyzoval více než 22 000 reálných jízd široké skupiny řidičů. Z dosažených výsledků vyplývá, že zohlednění agresivních jízdních událostí vedlo k více než 50% snížení chyby predikce spotřeby energie. Tento přístup ukazuje potenciál využití mobilních zařízení pro efektivní sledování jízdních stylů a tím i spotřeby energie.

Simulation for Prediction of Vehicle Efficiency, Performance, Range and Lifetime: A Review of Current Techniques and Their Applicability to Current and Future Testing Standards

Autoři: Abbas Fotouhi, Daniel J. Auger, Karsten Propp, Stefano Longo

Autoři ve své publikaci [8] představují rozsáhlý přehled simulačních technik používaných pro odhad účinnosti, výkonu, dojezdu a životnosti baterie moderních elektrických a hybridních automobilů. Rozlišují mezi přímými a nepřímými modelovacími přístupy a hodnotí jejich vhodnost v kontextu aktuálních i budoucích standardů, jako jsou UNECE nebo WLTP. Studie klade důraz na význam modelové věrnosti pro spolehlivou predikci dojezdu a navrhuje nové způsoby hodnocení degradace baterií v reálném i laboratorním prostředí. Výsledky této studie mohou přispět ke standardizaci simulačních metod a zlepšení řízení životního cyklu baterií.

A computationally efficient simulation model for estimating energy consumption of electric vehicles in the context of route planning applications

Autoři: Konstantinos N. Genikomsakis, Georgios Mitrentsis

Autoři v publikaci [9] představili výpočetně nenáročný model pro simulaci spotřeby energie elektromobilů, zaměřený na potřeby plánování tras. Model využívá obecný fyzikální popis pohybu vozidla a detailní specifikace parametrů klíčových komponent elektromobilu, přičemž převádí požadavky trakčního výkonu na potřebný výkon baterie. Autoři se zaměřili také na dynamické faktory, včetně schopnosti rekuperace a zohlednění zatížení motoru během náročných úseků trati. Pro validaci modelu byl využit simulační nástroj FASTSim [10]. Predikce modelu dosahovala průměrné absolutní chyby menší než 45 Wh. Výpočetní náročnost modelu byla v řádu desítek milisekund, což potvrzuje jeho vhodnost pro aplikace pracující v reálném čase. Autoři rovněž prezentovali simulační výsledky pro různé jízdní profily se sklonem vozovky mezi -6 % a 6 %, čímž demonstrují robustnost modelu v různých podmínkách.

Power-based electric vehicle energy consumption model: Model development and validation

Autoři: Chiara Fiori, Kyoungho Ahn, Hesham A. Rakha

Autoři ve své publikaci [11] popsali vývoj výpočetně nenáročného, ale přesného modelu pro okamžitý odhad spotřeby energie elektromobilů. Navržený model je založen na přímém výpočtu spotřeby energie podle výkonnostního profilu vozidla a je schopen rozlišovat různé pohyby vozidla, jako je akcelerace, jízda konstantní rychlostí, zpomalování nebo rekuperace. Model následně aplikuje konkrétní koeficienty účinnosti a zohledňuje faktory, jako je povaha trasy, odpor vzduchu, hmotnost vozidla a účinnost převodové soustavy. Jedním z jeho hlavních přínosů je schopnost přesně predikovat spotřebu energie bez nutnosti využívat komplexní simulační nástroje nebo přímý přístup k fyzikálním parametrům jednotlivých komponent vozidla. Validace modelu probíhala na základě reálných dat z několika vozidel, včetně modelů jako je Tesla Model S a Nissan Leaf. Model dosahoval průměrné chyby nižší než 10 %, což je srovnatelné s pokročilými simulačními nástroji. Autoři kladli důraz na využitelnost modelu v aplikacích, jako je online plánování tras nebo řízení flotil elektromobilů.

2.2 Dostupné řešení

V této části budou představeny dostupné aplikace, jež uživatelům umožňují plánovat trasu s ohledem na spotřebu elektromobilu. Tyto nástroje kombinují různé metody výpočtu spotřeby a často nabízejí interaktivní rozhraní, které zohledňuje konkrétní podmínky jízdy, specifikace vozidla či aktuální infrastrukturu nabíjecích stanic. Cílem této části je přiblížit nejzajímavější dostupná řešení, která mohou sloužit jako inspirace i jako srovnávací základ pro výslednou aplikaci této práce.

A Better Routeplanner

A Better Routeplanner [5] (ABRP) je pokročilá webová aplikace, která umožňuje efektivní plánování trasy pro elektrická vozidla. Uživatelé mohou specifikovat konkrétní model vozidla, počáteční kapacitu baterie, teplotu v kabině a řadu dalších parametrů, včetně počátku trasy, cíle a průjezdních bodů. Na základě těchto údajů aplikace generuje optimální trasu s ohledem na odhadovaný čas jízdy a potřebné zastávky na dobití. ABRP disponuje i navigačním režimem, díky němuž lze naplánovanou trasu využít přímo během jízdy.

EVNavigation

EVNavigation [6] je další aplikace zaměřená na plánování tras pro elektromobily. Umožňuje uživatelům zvolit konkrétní model vozidla, zadat počáteční a cílovou destinaci, nastavit průjezdní body a další parametry. Výhodou aplikace je integrace dat o dostupnosti nabíjecích stanic v reálném

čase. Podobně jako ABRP, i EVNavigation disponuje navigačním režimem a proto je vhodná i pro každodenní použití.

Kapitola 3

Volba technologii

Volba vhodných technologií pro vývoj aplikace je klíčovým krokem, který zásadním způsobem ovlivňuje nejen kvalitu výsledného produktu, ale také rychlost vývoje, jeho udržovatelnost a uživatelský komfort. Správně zvolená kombinace nástrojů a přístupů umožňuje efektivní realizaci požadovaných funkcí a zároveň usnadňuje případné budoucí rozšíření nebo údržbu.

V této kapitole budou rozebrány jednotlivé technologie, které byly v průběhu návrhu aplikace zvažovány. U každé části systému je uvedeno porovnání dostupných možností a následně zdůvodnění finálního rozhodnutí, proč byla zvolena právě tato technologie.

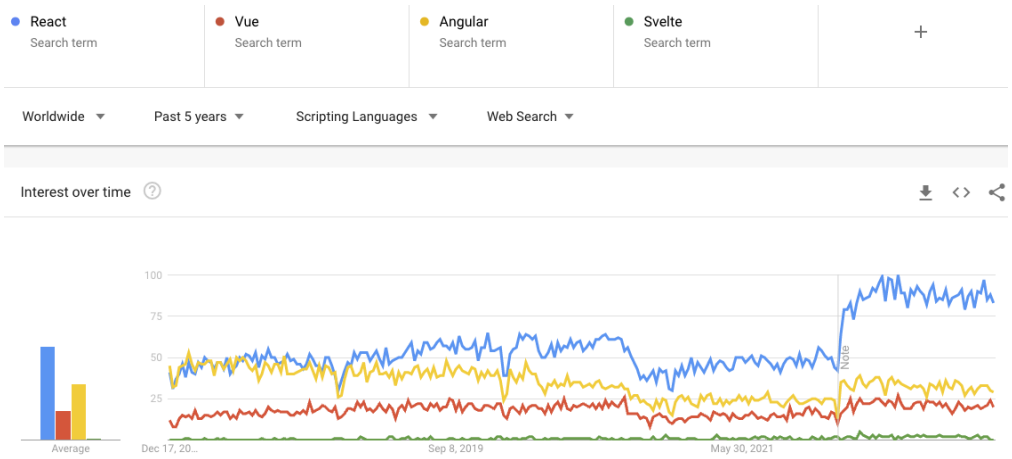
3.1 Uživatelské rozhraní

Prvním krokem při volbě uživatelského rozhraní je rozhodnutí, zda se bude jednat o webovou, desktopovou či mobilní aplikaci, případně jejich kombinaci. V rámci této práce bylo zvoleno vytvoření webové aplikace.

Webové aplikace jsou tvořeny především pomocí jazyka HTML, který určuje jejich strukturu a jednotlivé prvky, a stylovacího jazyka CSS, jenž definuje vzhledové vlastnosti. Moderní webové aplikace však často vyžadují i zpracování dat na pozadí, aby bylo možné dynamicky měnit zobrazovaný obsah. K tomuto účelu lze využít různé technologie – od běžně používaného PHP [12] nebo JavaScriptu, až po pokročilejší nástroje jako například knihovnu Flask [13] v jazyce Python [14].

Webová řešení patří v současnosti k nejrozšířenějším formám uživatelského rozhraní, a to především díky své univerzálnosti, snadné dostupnosti a poměrně jednoduchému vývoji. Samozřejmě s sebou nesou i jisté nevýhody, například nižší efektivitu využívání výpočetních zdrojů a tím pádem potenciálně pomalejší běh oproti nativním aplikacím. Otázkou však zůstává, zda tato nevýhoda v konkrétním případě skutečně představuje podstatný problém.

Díky vývojovým knihovnám, jako je React [15] nebo Vue.js [16], je dnes možné vytvářet pokročilé webové aplikace, které se chováním přibližují klasickým desktopovým programům. To přináší řadu



Obrázek 3.1: Popularita webových vývojových platforem dle Google Trends [20]

výhod jak pro vývojáře (rychlejší vývoj, přehlednější kód), tak pro uživatele (plynulé prostředí, dynamické generování obsahu). Zároveň lze pomocí technologií jako PWA snadno rozšířit aplikaci i do mobilního prostředí.

3.1.1 Zvolena technologie

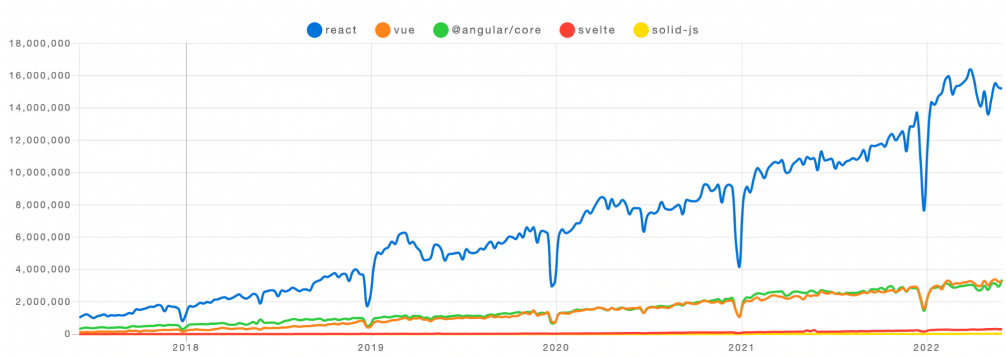
Již na počátku vývoje bylo rozhodnuto, že aplikace bude vytvořena jako jednostránková webová aplikace (tzv. SPA – Single Page Application) za využití vývojového prostředí založeného na jazyce JavaScript. Cílem bylo navrhnout řešení, které bude uživatelsky přívětivé, plynulé a zároveň snadno udržovatelné z pohledu vývoje.

Za tímto účelem byla provedena analýza nejčastěji používaných vývojových platforem s cílem nalézt nejvhodnější variantu. Do užšího výběru se dostaly tři technologie: React, Vue.js a Angular [17]. Kritéria výběru zahrnovala výkonnost, jednoduchost implementace a oblibu v rámci vývojářské komunity. Přehled popularity jednotlivých platforem je uveden na obrázcích 3.1 a 3.2, kde lze sledovat trendy dle vyhledávání na Google [18] a počtu stažení prostřednictvím balíčkovacího systému NPM [19].

Na základě provedené analýzy byla jako finální technologie zvolena vývojová platforma Vue.js. Přestože se nejedná o nejpoužívanější technologii z hlediska celkové popularity, její výhody v podobě jednoduchosti vývoje, rychlosti zvládnutí základů, široké dostupnosti knihoven a vysoké výkonnosti z ní činily nejvhodnější volbu pro účely této práce.

Vue.js

Vývojová platforma Vue.js vznikla v roce 2013 díky vývojáři Evanovi You, jehož cílem bylo vytvořit jednoduchý nástroj inspirovaný některými funkcemi z frameworku Angular. Aktuálně je Vue.js k dispozici ve verzi 3.5, přičemž verze 3.0, uvedená v roce 2020, přinesla zásadní změny



Obrázek 3.2: Popularita webových vývojových platforem dle počtu stažení přes NPM [20]

oproti předchozí řadě 2.X. Nejvýznamnější novinkou byl přechod od tzv. Options API k modernějšímu Composition API, což umožnilo vytvářet přehlednější a flexibilnější kód. Mezi další důležité změny patří podpora TypeScriptu [21], zlepšená výkonnost a reaktivní systém práce s komponentami.

Jednotlivé komponenty ve Vue.js se typicky skládají ze tří částí: `template`, `script setup` a `style`. Poslední z nich je volitelná. Část `template` definuje vzhled komponenty, tedy její strukturu a prvky, které obsahuje, včetně přiřazení CSS tříd a callback funkcí. Příklad této části lze vidět ve výpisu 3.1.

```
<template>
  <div>
    <component :is="getHomePageComponent"/>
  </div>
  <div class="text-container">
    <h3 v-html="$t('main.introduction')"></h3>
  </div>
</template>
```

Výpis 3.1: Template zobrazující strukturu domovské stránky

Druhou povinnou částí komponenty je `script setup`. Tato forma zápisu plně nahrazuje tradiční konstrukci `export default` (která je však stále podporována), a umožňuje kompaktnější a přehlednější zápis logiky komponenty. Využívá tzv. kompoziční API, které umožňuje snazší práci s reaktivními proměnnými, funkcemi a přístupem k datům mezi komponentami.

Mezi důležité nástroje, které lze v této části využít, patří například `Provide` a `Inject`. Tyto funkce slouží ke sdílení dat mezi komponentami – `Provide` definuje hodnotu v rodičovské komponentě, zatímco `Inject` umožňuje její načtení v potomkovi. Tento princip je vhodný např. pro globální nastavení nebo přístup ke sdíleným službám bez nutnosti předávat data skrze každou úroveň hierarchie komponent.

Dalším důležitým prvkem je `Emits`, který slouží ke komunikaci opačným směrem – tedy předávání událostí z potomka zpět rodiči. Typickým příkladem může být potvrzovací dialog pro smazání, kde po potvrzení akce potomkem rodič vykoná skutečné odstranění dat. Podobně významné jsou i `Props`, které definují, jaké hodnoty komponenta očekává z vnějšku. Příklad struktury `script setup` lze vidět ve výpisu B.1.

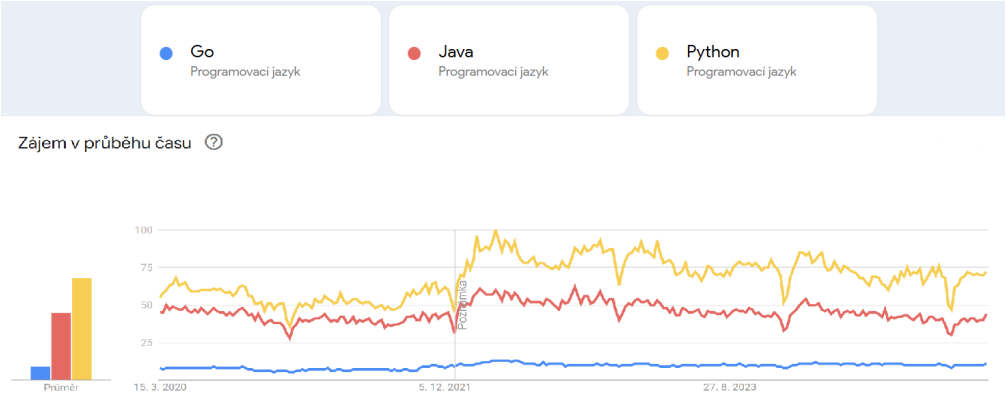
Třetí částí komponenty je `style`, která slouží k definici vzhledu komponenty. Pomocí direktivy `scoped` lze zajistit, že definovaný styl bude aplikován výhradně na danou komponentu, čímž se zabrání nechtěným zásahům do stylů ostatních částí aplikace. Ukázku této části lze vidět ve výpisu 3.2.

```
<style scoped>
.graph-card {
  width: 100%;
  margin-top: 20px;
}

.chart-container {
  position: relative;
  height: 400px;
  width: 100%;
}

v-card-text {
  padding: 0;
  height: 100%;
}

v-card-title {
  padding: 16px;
}
</style>
```

Výpis 3.2: Ukázka nastavení stylu pro konkrétní template

3.2 API server

Při volbě technologie pro implementaci API serveru přicházely v úvahu běžně používané programovací jazyky, jako například Java [22], .NET [23] nebo Python [14], případně méně tradiční varianty,



Obrázek 3.3: Zájem o Go, Javu a Python v průběhu času dle Google Trends [32]

jako jsou Rust [24], Ruby [25] nebo Go [26]. Během rozhodování bylo nutné zohlednit několik klíčových faktorů – mezi ně patřila rychlost zpracování požadavků, efektivita výpočtů, kvalita a dostupnost knihoven, ale také podpora cílových operačních systémů.

Finální výběr byl zúžen na tři nejvhodnější kandidáty: Java s vývojovým prostředím Spring Boot [27], Go s prostředím Gin [28] a Python s prostředím FastAPI [29] . Na obrázku 3.3 je možné vidět přibližný zájem o tyto technologie, kde dominuje Python následovaný Javou. Jazyk Go se dle těchto dat jeví jako nejméně populární z trojice, nicméně pro tento účel není popularita zásadním kritériem – důležitější jsou výkonnostní parametry daných prostředí.

Porovnání výkonu těchto řešení bylo provedeno například na webu travisloung.com [30], kde v roce 2022 zvítězil Spring Boot, následovaný Ginem. O rok později byla provedena obdobná analýza na webu medium.com [31], přičemž Gin dosahoval výsledků velmi blízkých Spring Bootu. S ohledem na tato srovnání a předchozí zkušenosti s jazykem Go padla finální volba právě na něj.

Go Lang

Go vznikl v roce 2007 díky společnosti Google s cílem zefektivnit vývoj v době vícejádrových systémů a síťových aplikací. Cílem bylo reagovat na slabiny jazyků běžně používaných v Googlu, ale zároveň zachovat jejich výhody. Mezi inspirace patřilo například statické typování a výkonnost z jazyka C, či čitelnost a jednoduchost známá z Pythonu. První veřejné oznámení proběhlo v roce 2009 a stabilní verze 1.0 byla uvolněna v roce 2012. Zajímavostí je, že Go dodnes běží na architektuře této první stabilní verze, což zajišťuje zpětnou kompatibilitu.

Go nabízí několik klíčových vlastností, které přispívají k jeho popularitě, zejména v oblasti vývoje webových serverů a distribuovaných systémů. Mezi nejvýznamnější patří gorutiny, princip bezpečného paralelismu a efektivní správa závislostí.

Gorutiny představují lehké jednotky souběžného běhu, které jsou spravovány běhovým prostředím Go. Oproti klasickým vláknům mají výrazně menší režii a umožňují efektivní paraleli-

zaci i ve velkém měřítku. Gorutiny běží ve stejném adresním prostoru, což vyžaduje správu přístupu ke sdílené paměti. K tomu slouží balíček `sync`, který nabízí synchronizační primitiva jako mutexy. Doporučenější a běžnější přístup však představují kanály, které umožňují předávat data mezi gorutinami a zároveň zajišťují jejich synchronizaci. Kanál se zablokuje, dokud obě strany nejsou připravené ke komunikaci, což eliminuje nutnost explicitního zamykání.

Bezpečný paralelismus v Go vychází z filozofie: „nesdílejte paměť prostřednictvím komunikace, ale sdílejte paměť pomocí komunikace“ [33]. To znamená, že vývojáři jsou vedeni k tomu, aby mezi gorutinami předávali data pomocí kanálů namísto přímé manipulace se sdílenou pamětí. Tento přístup výrazně snižuje riziko typických paralelních chyb, jako je souběh nebo deadlocky.

Správa závislostí v Go je řešena prostřednictvím systému Go Modules, aktivovaného pomocí příkazu `go mod`. Na rozdíl od jazyků jako Java nebo Python, kde je potřeba použít externí nástroje (např. Maven, pip), je tento systém plně integrován. Veškeré závislosti jsou spravovány v souboru `go.mod`, který obsahuje jak přímé, tak i tranzitivní závislosti, automaticky zjištěné při sestavení projektu.

Za zmínku stojí také rozsáhlá standardní knihovna, která pokrývá širokou škálu oblastí. Například balíček `crypto` nabízí nástroje pro práci s kryptografií, zatímco `gzip` nebo `zlib` umožňují kompresi a dekompresi souborů. Kromě toho standardní knihovna obsahuje nástroje pro práci se sítí, formáty JSON, HTTP servery i klienty, šablonovací systémy a mnoho dalšího, což výrazně snižuje potřebu externích knihoven.

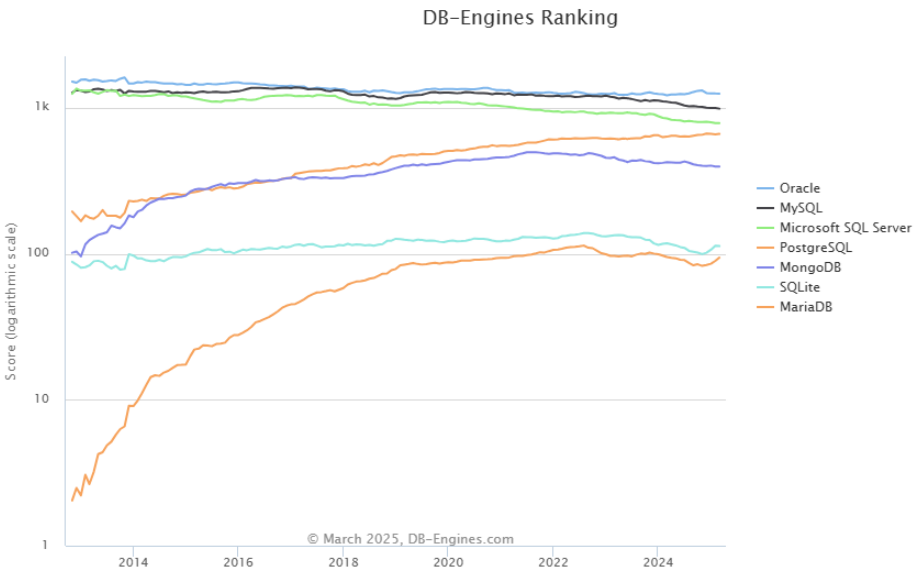
3.3 Databázové systémy

Při výběru databázového systému nebylo kladeno velké množství specifických požadavků. Hlavními kritérii byly dostupnost bez nutnosti licencování a podpora prostorových dat. Na základě popularity databázových systémů, kterou lze vidět na obrázku 3.4, byl výběr zúžen na Microsoft SQL Server [34], Oracle [35], MySQL [36] a PostgreSQL [37]. Po aplikování prvního kritéria zůstaly pouze dvě možnosti, a to MySQL a PostgreSQL. Finálním rozhodujícím faktorem byly tedy předchozí zkušenosti, přičemž nejlépe vyhovovalo řešení PostgreSQL, a to jak z hlediska kvality, tak i uživatelského komfortu při práci s tímto systémem.

PostgreSQL

PostgreSQL patří mezi nejrozšířenější open-source relační databázové systémy. Jeho vývoj započal v roce 1989 jako pokračování projektu Ingres. Mezi jeho hlavní výhody patří důsledná podpora standardu SQL a vysoká míra rozšiřitelnosti, která umožňuje přidávání nových datových typů, operátorů či funkcí.

Významnou vlastností PostgreSQL je implementace konceptu *Multi-Version Concurrency Control* (MVCC), který umožňuje souběžný přístup více uživatelů k datům bez potřeby blokování ope-



Obrázek 3.4: Popularita Databázových systému dle db-engines.com [38]

rací čtení a zápisu. Díky tomu se PostgreSQL hodí pro systémy s vysokými nároky na dostupnost a konzistenci dat.

Pro svou stabilitu, škálovatelnost a širokou komunitní podporu je PostgreSQL oblíbenou volbou například v oblasti financí, zdravotnictví nebo geografických informačních systémů. Podpora rozšíření jako PostGIS navíc výrazně rozšiřuje možnosti práce s prostorovými daty.

3.4 Verzovací systém

V závěrečné fázi návrhu bylo nutné rozhodnout, který verzovací systém bude využíván. Mezi hlavní kandidáty patřily platformy GitHub [39], Bitbucket [40] a GitLab [41]. Volba nakonec padla na GitLab, a to i díky tomu, že jej hostuje také univerzita.

Tato volba přináší řadu výhod, zejména v oblasti správy projektu a organizace práce v týmu. GitLab umožňuje detailní nastavení rolí a oprávnění jednotlivých členů, což usnadňuje řízení přístupu k repozitáři. Další výhodou je integrovaná wiki, která je součástí každého projektu. Díky ní je možné efektivně sdílet dokumentaci a usnadnit přenos znalostí mezi členy týmu, což přispívá k rychlejšímu zapojení nových členů.

Kapitola 4

Fyzikální model

Tato kapitola se zaměřuje na popis fyzikálního modelu navrženého v rámci této práce, který slouží jako klíčový prvek pro odhad předpokládané spotřeby elektromobilu. Model vychází z fyzikálních zákonitostí popisujících pohyb vozidla a zohledňuje hlavní faktory ovlivňující spotřebu.

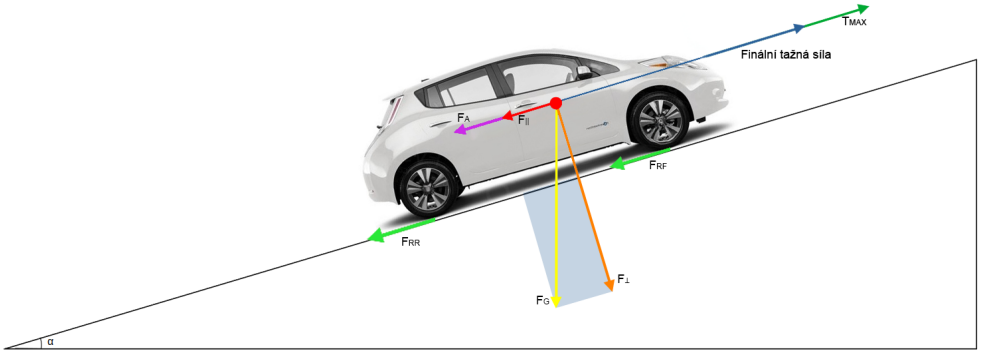
Jeho cílem je nabídnout dostatečně přesný odhad spotřeby ve výpočetně nenáročné podobě, vhodný pro použití v reálném čase. Na rozdíl od čistě statických metod či přístupů založených na strojovém učení poskytuje navržené řešení transparentní výpočty, a tedy i snadnější možnosti ladění a rozšiřování.

V následujících kapitolách bude popsán počáteční vývoj, včetně první verze modelu, dále aktuální implementace s detailním popisem využívaných fyzikálních jevů, následně jeho reprezentace v kódu a závěrem budou nastíněny možné směry dalšího rozvoje.

4.1 Historie vývoje fyzikálního modelu

Tento fyzikální model původně vycházel z výpočtového modelu spotřeby elektrických vlaků [42]. V první fázi vývoje byla zkoumána možnost adaptace tohoto existujícího řešení s cílem vytvořit efektivnější variantu, která by byla vhodná pro prostředí elektromobilů. Tento přístup byl však poměrně brzy opuštěn, protože ačkoliv může být pohyb vlaků a aut na první pohled podobný, rozdíly mezi nimi jsou značné. Jedním z příkladů je limitace u vlaků daná přenosem energie mezi trakčním ústrojím a kolejemi. Tento jev sice u elektromobilů do určité míry také existuje, nicméně má výrazně menší význam.

Na základě těchto zjištění byl vytvořen nový model, představující první pokus o realistickou simulaci spotřeby elektromobilu. Vývoj této verze spočíval v testování modelu vůči předem definovaným scénářům – tedy bez simulace konkrétní trasy, pouze se snahou dosáhnout spotřeby, která by odpovídala zaznamenaným hodnotám. Postupně byly do modelu přidávány další fyzikální vlivy, které se při jízdě uplatňují. Mezi ně patří například paralelní složka gravitační síly ($F_{\perp}$), aerodynamický odpor (F_A) a valivý odpor (F_R), jejichž působení je popsáno rovnicemi 4.1, 4.2 a 4.3.



Obrázek 4.1: Zobrazení působících sil v první verzi modelu

Aerodynamický odpor je dán součinitelem odporu prostředí, hustotou vzduchu, čelní plochou vozidla a kvadrátem rychlosti. Valivý odpor zohledňuje tlak v pneumatikách i rychlost vozidla, zatímco paralelní i normálová složka gravitační síly vycházejí z hmotnosti vozidla, gravitačního zrychlení a sklonu vozovky. Přesné vztahy jsou uvedeny níže. Proti těmto silám působí tažná síla, vznikající jako součin normálové síly a koeficientu tření. Přehled všech působících sil ve druhé verzi modelu znázorňuje obrázek 4.1. Tyto výpočty byly získány ze stránek mvwautotechniek.nl [43] a engineeringtoolbox.com [44].

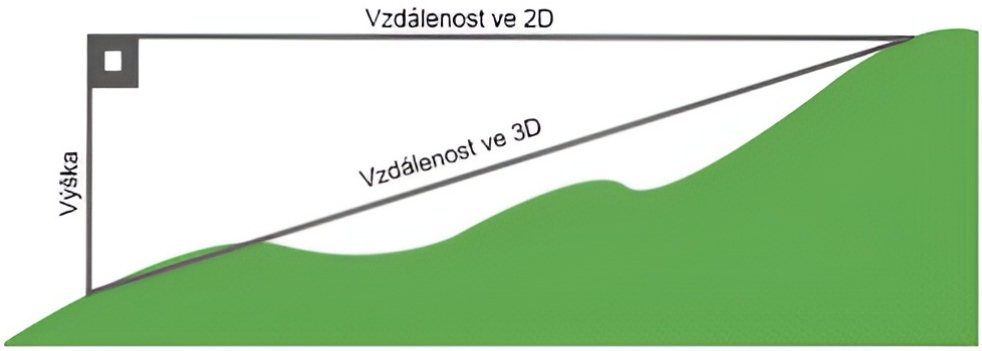
$$F_A = 0.5 \cdot C_d \cdot \rho \cdot A_f \cdot v^2 \quad [\mathrm{N}] \tag{4.1}$$

$$F_R = (0.005 + (1/p_t)) \cdot (0.01 + 0.0095 \cdot (v/100)^2) \cdot F_{||} \quad [\mathrm{N}] \tag{4.2}$$

$$F_\perp = m \cdot g \cdot \cos(\alpha) \quad [\mathrm{N}] \tag{4.3}$$

$$F_{||} = m \cdot g \cdot sin(\alpha) \quad [\mathrm{N}] \tag{4.4}$$

Tato verze modelu vykazovala poměrně uspokojivé výsledky při simulaci kratších tras, na nichž byla testována. Výsledky porovnání modelu s reálnými daty jsou uvedeny v tabulce C.1. Během testování delších tras však začaly vznikat výrazné odchylky od skutečných hodnot. Na základě těchto poznatků byl zahájen vývoj druhé verze modelu, která je využita v rámci této práce.



Obrázek 4.2: Vzdálenost svahu

4.2 Aktuální verze fyzikálního modelu

Aktuální verze modelu se výrazně liší od počátečních pokusů založených na simulaci jízdy vlaků, které sloužily jako prvotní předloha, a později byly nahrazeny první verzí modelu. Momentální verze pracuje s širší řadou fyzikálních vlivů, než tomu bylo dříve. Základ této verze poskytla kniha Electric Vehicle Technology od William B. Ribbens [45]. Model očekává jako vstup data s GPS body trati a momentální rychlosti v nich. Následně dochází k přepočtu vzdálenosti mezi dvěma body. Za tímto účelem byl využit výpočet 4.5, jenž na základě poloměru Země (R), zeměpisné šířky (ϕ) a zeměpisné délky (λ) vypočítá vzdálenost (d) mezi těmito body. Jelikož tento výpočet představuje vzdálenost pouze v rovině, je nutné vypočítat vzdálenost ve svahu. Určení této vzdálenosti není nikterak složité, lze ji získat skrze Pythagorovu větu. Přesněji díky využití rozdílu nadmořských výšek a již vypočítané vzdálenosti v rovině. Tento výpočet lze zapsat vzorcem 4.6, a také zobrazit v grafické podobě, jež lze vidět na obrázku 4.2. Přesný popis jednotlivých složek výpočtu je zobrazen v seznamu níže.

- ϕ - představuje zeměpisnou šířku
- λ - představuje zeměpisnou délku
- d_{2D} - představuje výslednou vzdálenost v rovině (**km**)
- d_{3D} - reprezentuje výslednou vzdálenost ve svahu (**km**)

$$\begin{aligned} a &= (\sin(\Delta\phi))^2 + \cos(\phi_n) \cdot \cos(\phi_{n+1}) \cdot (\sin(\Delta\lambda))^2 \\ c &= 2 \cdot \arctan\left(\sqrt{\frac{1-a}{a}}\right) \\ d_{2D} &= 6371.0 \cdot c \end{aligned} \tag{4.5}$$

$$d_{3D} = \sqrt{d_{2D}^2 + \Delta h^2} \tag{4.6}$$

4.2.1 Principy využité v modelu

Aktuální verze fyzikálního modelu pracuje s využitím několika zákonů, které působí v průběhu jízdy. Simulací těchto veličin dostáváme sílu, kterou vozidlo musí vytvořit. Jejím převodem na výkon a zohledněním efektivity jednotlivých komponent vozidla získáme energii, kterou vozidlo spotřebuje. Níže nalezneme jednotlivé složky, které model zohledňuje.

Trakční síla

Trakční síla představuje celkovou hnací sílu, jež musí vozidlo vyvinout k překonání všech odporů působících proti pohybu. V kontextu elektromobilu jde o sílu, kterou zajišťuje elektromotor za účelem dosažení požadovaného zrychlení a udržení pohybu mezi dvěma body.

V rámci této práce je trakční síla chápána jako součet čtyř základních složek, které musí motor překonat, aby vozidlo mohlo dosáhnout požadované rychlosti a překonat vzdálenost mezi dvěma body. Jedná se o výslednou sílu, jež se dále převádí na výkon potřebný k určení celkové spotřebované energie. Konkrétně jde o valivý odpor (F_{rr}), odpor stoupání (F_{hc}), aerodynamický odpor (F_a) a setrvačnou sílu (F_i). Veškeré veličiny použité v tomto výpočtu jsou uváděny v základních jednotkách soustavy SI. Tento vztah lze vyjádřit rovnicí 4.7.

$$F_t = F_{rr} + F_{hc} + F_a + F_i \quad [\mathrm{N}] \tag{4.7}$$

Setrvačná síla

Setrvačná síla vyplývá z fyzikální vlastnosti těles, podle níž objekt setrvává ve svém stavu klidu nebo rovnoměrného přímočarého pohybu, dokud na něj nepůsobí vnější síla. V kontextu vozidla tato síla určuje míru potřebné energie k dosažení nebo udržení určité rychlosti.

V rámci tohoto modelu setrvačná síla vyjadřuje, zda vozidlo zrychluje, zpomaluje nebo se pohybuje konstantní rychlostí. Pro výpočet je zapotřebí znát okamžité zrychlení (a) v ms^{-2}, hmotnost pasažérů (M_p) v kilogramech, hmotnost vozidla (M_a), také v kilogramech, a korekční faktor (C_{io}), který zohledňuje vliv rotačních hmot, jako jsou například kola vozidla. Hodnota korekčního faktoru byla stanovena na základě doporučení v publikaci Modelling the Effect of Driving Events on Electrical Vehicle Energy Consumption Using Inertial Sensors in Smartphones [46] na 0,05.

$$F_i = C_{io} \cdot (M_{auto} + M_{pasažéři}) \cdot a \quad [\mathrm{N}] \tag{4.8}$$

Odpor stoupání

Odpor stoupání představuje složku gravitační síly, která působí ve směru sklonu terénu, a výrazně ovlivňuje energetické nároky na pohyb vozidla. Při jízdě do kopce působí proti směru pohybu, čímž zvyšuje potřebný výkon motoru. Naopak při jízdě ze svahu působí ve směru jízdy a napomáhá pohybu vozidla, což může vést ke snížení spotřeby nebo i rekuperaci energie.

V rámci tohoto modelu je odpor stoupání počítán dle rovnice 4.9. Výsledná síla je určena jako součin gravitačního zrychlení (g), celkové hmotnosti vozidla včetně posádky a nákladu ($M_{auto} + M_{pasažéři}$) a sinu sklonu svahu ($\sin(\alpha)$). Úhel sklonu α je stanoven jako poměr výškového rozdílu mezi dvěma body ($\Delta elev$) a jejich prostorové vzdálenosti ($dist_{3D}$), jak vyjadřuje rovnice 4.10. Veškeré veličiny jsou uváděny v základních jednotkách soustavy SI.

$$F_{hc} = (M_{auto} + M_{pasažéři}) \cdot g \cdot \sin(\alpha) \quad [\mathrm{N}] \tag{4.9}$$

$$\sin(\alpha) = \frac{\Delta elev}{dist_{3D}} \tag{4.10}$$

Valivý odpor

Valivý odpor je síla, která působí proti pohybu tělesa valícího se po povrchu. Je způsoben především nepružnými deformacemi v místě kontaktu, například pneumatiky a vozovky. V praxi to znamená, že při zatížení dochází ke ztrátám energie, která se nevrací při uvolnění deformace, a je tedy nutná dodatečná energie k udržení pohybu.

V rámci tohoto modelu je valivý odpor generován interakcí pneumatik vozidla s vozovkou. Výpočet vychází z rovnice 4.11, kde se využívá celková hmotnost vozidla včetně posádky ($M_{auto} + M_{pasažéři}$), gravitační zrychlení (g), rychlost vozidla (v), úhel sklonu terénu (α) a koeficient valivého odporu (c_{rr}).

Koeficient c_{rr} je v tomto modelu aproximován vztahem 4.12, který zohledňuje závislost na rychlosti vozidla. Tento přístup byl převzat z publikace Modelling the Effect of Driving Events on Electrical Vehicle Energy Consumption Using Inertial Sensors in Smartphones [46]. Veškeré veličiny jsou uváděny v základních jednotkách soustavy SI.

$$F_{rr} = sign(v) \cdot (M_{auto} + M_{pasažéři}) \cdot g \cdot \cos(\alpha) \cdot c_{rr} \quad [\mathrm{N}] \tag{4.11}$$

$$c_{rr} = 0.01 \cdot \left(1 + \frac{v}{100}\right) \tag{4.12}$$

Odpor vzduchu

Odpor vzduchu je síla, která působí proti pohybu vozidla v důsledku interakce s okolním vzduchem. Tato síla narůstá s rostoucí rychlostí vozidla a má kvadratickou závislost na jeho rychlosti, čímž se stává jedním z hlavních faktorů ovlivňujících spotřebu energie, zejména ve vyšších rychlostech.

Aerodynamický odpor je ovlivněn několika parametry - především čelní plochou vozidla (A), součinitelem aerodynamického odporu (C_d) a relativní rychlostí mezi vozidlem a prouděním vzduchu. Vztah pro výpočet odporu vzduchu je uveden v rovnici 4.13, kde ρ představuje hustotu vzduchu, v rychlost vozidla a $v_{\text{vítr}}$ rychlost větru ve směru jízdy. Směr výsledné síly je určován znaménkem této relativní rychlosti.

V rámci tohoto modelu je z důvodu nepředvídatelnosti a chybějících dat o směru a intenzitě větru veličina $v_{\text{vítr}}$ zanedbána, a odpor vzduchu je tedy počítán pouze na základě rychlosti vozidla. Hustota vzduchu je pro zjednodušení považována za konstantní s hodnotou $\rho = 1{,}25\ \text{kg/m}^3$. Veškeré veličiny jsou udávány v základních jednotkách soustavy SI.

$$F_{aero} = \text{sign}(v + v_{\text{vítr}}) \cdot \frac{1}{2} \cdot \rho \cdot A \cdot C_d \cdot (v + v_{\text{vítr}})^2 \quad [\text{N}] \tag{4.13}$$

Požadavek na výkon dodávaný z baterie

Jakmile známe trakční výkon motoru, můžeme z něj odvodit okamžitý požadavek na výkon baterie. Tento požadavek zahrnuje jak ztráty způsobené neefektivitou převodového ústrojí a elektromotoru, tak i spotřebu komfortních systémů.

V případě, že je trakční výkon kladný, využíváme výpočet 4.14, kde:

- P_t je trakční výkon motoru,
- $\eta_{\text{převod}}$ účinnost převodové soustavy,
- η_{motor} účinnost elektromotoru,
- P_{aux} výkon pomocných zařízení (např. klimatizace, světla apod.).

$$P_{\text{bat}} = \frac{P_t}{\eta_{\text{převod}} \cdot \eta_{\text{motor}}} + P_{\text{aux}} \quad [\text{W}] \tag{4.14}$$

Pokud však trakční výkon nabývá záporných hodnot (např. během zpomalování), znamená to, že vozidlo může zpětně získávat energii pomocí rekuperace. V takovém případě se využívá vztah 4.15, kde se bere v úvahu i rekuperační koeficient k, který vyjadřuje efektivitu rekuperace v závislosti na rychlosti vozidla:

$$P_{\text{bat}} = k \cdot P_t \cdot \eta_{\text{převod}} \cdot \eta_{\text{motor}} + P_{\text{aux}} \quad [\text{W}] \tag{4.15}$$

Tento koeficient je definován následujícím způsobem:

$$k = \begin{cases} 0.5 \cdot v, & v < 5 \text{ m/s} \\ 0.5 + 0.015 \cdot (v - 5), & v \geq 5 \text{ m/s} \end{cases} \tag{4.16}$$

Díky tomuto rozlišení jsme schopni zohlednit jak energetické ztráty během pohonu, tak i případné zisky při brzdění, a tím přesněji určit celkový energetický tok mezi baterií a pohonným ústrojím.

Výpočet celkové spotřebované energie

Celková spotřeba energie během jízdy je určena součtem okamžitých požadavků na výkon baterie v jednotlivých časových krocích, přičemž každý z nich je násoben délkou časového intervalu Δt.

Do výpočtu je rovněž zahrnuta účinnost baterie η_{bat}, která reprezentuje ztráty vznikající při přeměně a dodávce elektrické energie z baterie do pohonného systému. Výsledný vztah je uveden ve vzorci 4.17.

$$E_{trip} = \frac{1}{\eta_{bat}} \sum_{k=1}^{N} P_{bat}[k] \cdot \Delta t \quad [\text{J}] \tag{4.17}$$

4.3 Fyzikální model v kódu

Navržený model má svou podobu také v kódu výsledné aplikace. Zde je reprezentován dvěma soubory. Konkrétně `Consumption.go` a `Calculations.go`, kde první zmíněný řídí průchod daty trasou a provádí volání jednotlivých výpočtů, které jsou zapsány ve druhém zmíněném souboru.

Prvním úkonem, který musí simulační funkce provádět, je příprava všech proměnných, jež bude simulace využívat pro mezi výpočty, nebo již finální výsledky. Následně dochází k průchodu jednotlivými body trasy. V rámci jednotlivých iterací je proveden výpočet akcelerace, vzdálenosti ve svahu, hodnotě $\sin(\alpha)$ a $\cos(\alpha)$. Následně díky těmto hodnotám dochází k výpočtu trakční síly, z níž se v dalším kroku provádí výpočet okamžitého požadavku na výkon baterie. V případě, že požadavek je záporný, dojde k výpočtu rekuperované energie. Následně se tento požadavek násobí časem, abychom dostali přesnou hodnotu, která byla požadována v tomto časovém intervalu. Na závěr smyčky dojde k uložení mezivýsledků do předem připravených proměnných, a cyklus pokračuje dále. Na konci simulované trasy dojde k výpočtu celkové požadované energie v průběhu jízdy. Tento popsaný princip lze vidět ve formě vývojového diagramu v obrázku A.1.

Druhý zmíněný soubor se skládá z funkcí reprezentujících jednotlivé výpočty, které byly představeny v kapitole 4.2. Příkladem může být výpočet valivého odporu, jehož funkci lze vidět společně s funkcí výpočtu koeficientu valivého odporu ve výpisu 4.1.

```
func rollingResistanceForce(vehicle models.Vehicle, passengersMass float64,
    angleCos, velocity float64) float64 {
   rollingResistanceCoefficient := rollingResistancesCoefficientCalc(velocity)

   sign := controllers.Sign(velocity)

   force := sign * (vehicle.TotalMass + passengersMass) * g * angleCos *
       rollingResistanceCoefficient

   return force
}

func rollingResistancesCoefficientCalc(velocity float64) float64 {
   return 0.01 * (1 + (velocity / 100))
}
```

Výpis 4.1: Kód prezenetující výpočet valivého odporu uvnitř fyzikálního modelu

Dalším zajímavým příkladem může být výpočet hodnot vzdáleností spolu s výpočtem sinu a cosinu úhlů. Ten lze vidět ve výpisu 4.2. Lze si povšimnout, že uvnitř těla se volá funkce Haversine, ta odpovídá popsanému výpočtu vzdálenosti v rovině. Její reprezentaci v kódu lze vidět na výpisu 4.3. Jelikož se v případě funkce Haversine počítá s hodnotami v kilometrech, je nutné výslednou vzdálenost převést na metry, aby tato hodnota odpovídala požadavkům ostatních funkcí.

```
func GetDistanceValues(currentPoint, nextPoint models.Point) (float64, float64,
    float64, float64) {
   distance2D := controllers.Haversine(currentPoint, nextPoint)
   distance3D := math.Sqrt(math.Pow(distance2D, 2) + math.Pow(nextPoint.Elevation-
       currentPoint.Elevation, 2))
   elevationDelta := nextPoint.Elevation - currentPoint.Elevation
   angleCos := 1.0
   angleSin := 0.0
   if distance3D != 0 {
      angleCos = distance2D / distance3D
      angleSin = elevationDelta / distance3D
   }

   return distance2D, distance3D, angleCos, angleSin
```

```
}
```

Výpis 4.2: Kód prezenetující výpočet vzdáleností a úhlů pro fyzikální model

```
func Haversine(point1, point2 Model.Point) float64 {

   R := 6371.0
   dlat := degreeToRads(point2.Latitude) - degreeToRads(point1.Latitude)
   dlon := degreeToRads(point2.Longitude) - degreeToRads(point1.Longitude)

   a := math.Pow(math.Sin(dlat/2), 2) + math.Cos(degreeToRads(point1.Latitude))*
      math.Cos(degreeToRads(point2.Latitude))*math.Pow(math.Sin(dlon/2), 2)
   c := 2 * math.Atan2(math.Sqrt(a), math.Sqrt(1-a))

   distance := R * c * 1000

   return distance
}
```

Výpis 4.3: Kód prezenetující výpočet vzdálenosti na ploše uvnitř fyzikálního modelu

4.4 Možné rozšíření modelu

Aktuální verze modelu nezohledňuje veškeré faktory ovlivňující spotřebu energie. Mezi další vlivy, které by bylo možné do modelu zahrnout, patří například venkovní teplota, jež ovlivňuje jak tepelný management baterie, tak i vytápění nebo chlazení interiéru vozidla.

Zajímavým rozšířením by mohlo být také zahrnutí odporu vznikajícího při průjezdu zatáčkou. Ten může být významný především při dynamickém stylu jízdy, kdy řidič využívá limitních schopností vozidla v zatáčkách. V běžném provozu je však tento vliv zanedbatelný, a proto nebyl v současné verzi modelu uvažován.

Kapitola 5

Formát dat

Základním předpokladem pro správné fungování simulačního modelu jsou vstupní data reprezentující průběh trasy. Jejich přesnost je zásadní pro korektní určení průběhu jízdy, a tím i pro výpočet výsledné spotřeby energie. Tato kapitola bude popisovat strukturu a podobu dat požadovaných modelem, přibližovat způsob jejich získávání během počátečního testování a také postupy využívané pro kompletní simulaci jízdy.

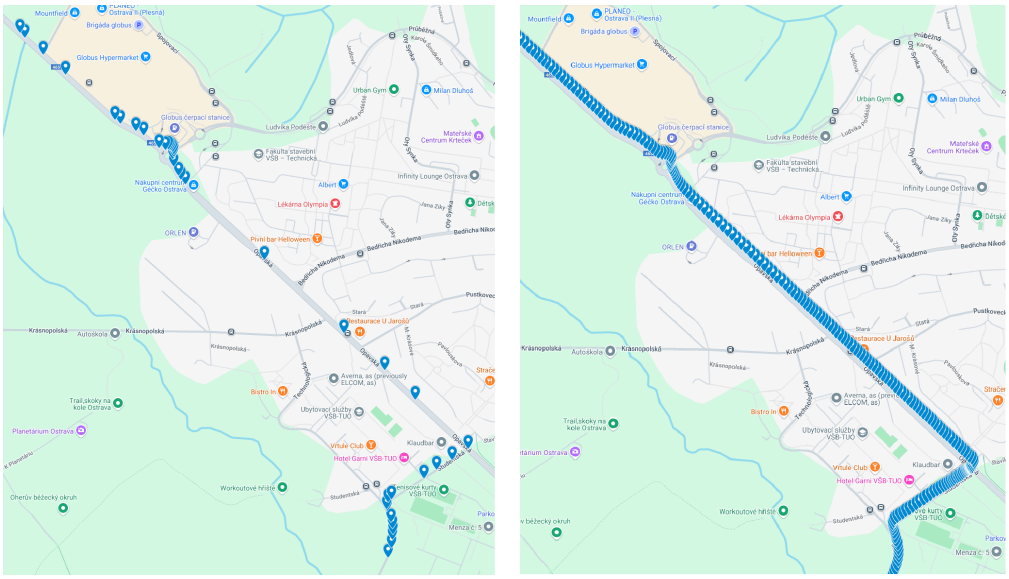
5.1 Požadovaná data modelem

Jak již bylo zmíněno, kvalita vstupních dat je klíčovým faktorem pro správnou funkčnost modelu. Kromě samotného záznamu trasy je důležitá i forma, ve které jsou data do modelu vkládána. Model nepracuje pouze s trasovými daty, ale vyžaduje i parametry vozidla. Bez těchto údajů by nebylo možné přesně spočítat energetické ztráty vozidla, a tedy ani výslednou spotřebu.

5.1.1 Požadovaná data trasy

Přesný popis trasy, kterou vozidlo absolvuje, patří mezi klíčové aspekty simulačního modelu. K dosažení potřebné přesnosti se využívají GPS souřadnice průjezdních bodů, jejich nadmořská výška a buď okamžitá rychlost, nebo čas průjezdu daným bodem. I když není nezbytné zaznamenávat trasu v extrémně krátkých intervalech, vyšší hustota bodů obvykle zlepšuje výslednou přesnost výpočtu. Efektivním přístupem je zaznamenat trasu detailně zejména v místech, kde dochází ke změně směru nebo rychlosti. Rozdíl mezi detailním a efektivním záznamem lze vidět na obrázku 5.1.

Simulační model však nekonzumuje surová data, musí tedy být programem zpracována a převedena do formátu, který byl za tímto účelem vytvořen. Trasa je reprezentována strukturou `Trip`, jenž představuje informace spojené s jednou konkrétní jízdou. Parametry této struktury lze vidět ve výpisu 5.1. Trasa je reprezentována názvem, vozidlem, váhou pasažérů a sadou bodů, jenž odpovídají právě zmiňovaným průjezdným bodům na trati. Strukturu reprezentující konkrétní



Obrázek 5.1: Vizualizace efektivního a detailního záznamu trati

bod lze vidět ve výpisu 5.2. Hmotnost pasažérů zahrnuje také hmotnost zavazadel a dalšího nákladu vozidla.

```
type Trip struct {
   TripName      string
   Points        []Point
   Vehicle       Vehicle
   PassengersMass float64
}
```

Výpis 5.1: Datová struktura Trip, reprezentující konkrétní trasu

```
type Point struct {
   Longitude float64
   Latitude float64
   Elevation float64
   Time      time.Time
   Speed     float64
}
```

Výpis 5.2: Datová struktura Point, reprezentující konkrétní polohu na trase

5.1.2 Požadovaná data vozidla

Datová struktura popisující trasu vyžaduje také informace o vozidle. Tyto hodnoty jsou nezbytné pro výpočty prováděné fyzikálním modelem a částečně slouží i pro vizuální reprezentaci simulace. Jedná se o technické parametry, které definují specifické vlastnosti vozidla. Mezi klíčové patří například:

- `BatteryEfficiency` – účinnost baterie,
- `DriveLineEfficiency` – účinnost hnací soustavy,
- `TotalMass` – celková hmotnost vozidla,
- `FrontalArea` – čelní plocha vozidla.

Získání těchto hodnot může být v praxi obtížné, a proto se některé z nich často odhadují. Typickým příkladem je právě čelní plocha, která se obvykle neuvádí v běžné dokumentaci. Pro její přibližný výpočet lze použít vzorec 5.1, jenž využívá výšku a šířku vozidla. Tyto hodnoty lze snadno dohledat v technických specifikacích výrobce. Všechny jednotky jsou uváděny v základní soustavě SI.

$$A = 0.85 \cdot \text{šířka} \cdot \text{výška} \quad \left[m^2\right] \tag{5.1}$$

5.2 Zisk dat pro prvotní testy

Pro testování a ověřování funkčnosti modelu bylo nutné získat scénáře se známým průběhem trasy a detailním záznamem spotřeby energie. Prvotní sběr dat probíhal na vozidle *Nissan Leaf ZE0* [47] za využití aplikace *Leaf Spy Pro*, která přes OBD2 adaptér komunikovala přímo s řídicí jednotkou vozidla. Její hlavní výhodou byla schopnost zaznamenávat kapacitu baterie v čase, což z ní činilo jediný dostupný nástroj vhodný pro sledování reálné spotřeby během jízdy.

Pro záznam trasy však nebyla ideální – zápis GPS souřadnic byl nekonzistentní, body byly často zaznamenány mimo reálnou trasu nebo chyběla část jízdy. Výsledkem byl neuspořádaný záznam s body rozptýlenými v okolí skutečné trasy, jak je patrné z obrázku 5.2b.

Z těchto důvodů byla paralelně využívána aplikace *Sensor Logger*, která zaznamenávala polohu ve velmi častých a přesných intervalech, a tím poskytovala detailní a spojitý popis skutečné trasy. Ilustrace takového záznamu je uvedena na obrázku 5.2a. Výsledná testovací data byla vytvořena kombinací obou zdrojů. Přesněji spotřeba energie pocházela z Leaf Spy Pro, zatímco trasa byla převzata ze Sensor Loggeru.

Pro efektivnější záznam byl navíc vytvořen vlastní přístupový bod a databáze, do které aplikace automaticky ukládaly svá data. Tím se zjednodušila správa a zpracování naměřených dat.

(a) mobilní aplikace (b) Řídící jednotka vozu

Obrázek 5.2: Naměřené GPS body během testování

Při testování jiných vozidel, kde nebyla k dispozici možnost přímé komunikace s řídicí jednotkou, byla spotřeba energie získávána z palubního počítače. V těchto případech šlo o agregované údaje s nižší přesností, avšak dostatečné pro validaci trendů simulace.

5.3 Data pro simulování jízdy

Jelikož cílem práce nebylo simulovat pouze trasy, které byly reálně projety vozidlem, bylo nezbytné najít nástroj pro generování ideální trasy včetně potřebných atributů. Jako vhodné řešení se nabízelo využití externího navigačního API.

Mezi zvažované varianty patřila API rozhraní od společností Google [48], TomTom [49] a GraphHopper [50]. Prvotně bylo testováno API od Googlu, které však kvůli přísnému omezení bezplatného provozu nebylo možné dlouhodobě využívat. Následně bylo zkoumáno API od společnosti TomTom, které nabízelo příznivější podmínky z hlediska ceny i funkčnosti. Při podrobnějším testování však vyšlo najevo, že v odpovědích chybí informace o nadmořské výšce, což je jeden z klíčových požadavků modelu.

Finální volba tak padla na API od společnosti GraphHopper. Přestože komerční tarify nejsou nejlevnější, nabízí platforma i bezplatnou variantu, která pro účely této práce plně postačuje.

Kromě dostupnosti nadmořské výšky splňuje API veškeré funkční požadavky a umožňuje přímé testování požadavků prostřednictvím webového rozhraní. Díky tomu se stalo primárním zdrojem tras pro simulované jízdy.

Kapitola 6

Vývoj aplikace

Vývoj aplikace pro využití navrženého simulačního modelu tvořil klíčovou část této práce. Jejím hlavním cílem bylo nabídnout uživatelsky přívětivé rozhraní pro práci s modelem, které umožňuje plánování trasy a simulaci spotřeby energie v průběhu jízdy.

V této kapitole bude popsána architektura aplikace, její jednotlivé komponenty, použité balíčky, knihovny a externí API. Zmíněna budou také úskalí, která bylo nutné během vývoje překonat, a návrhy na možná budoucí rozšíření aplikace.

6.1 Rozdělení aplikace

Aplikace byla navržena tak, aby její jednotlivé části byly logicky členěny podle funkce. Architektura je rozdělena na databázovou, klientskou a serverovou vrstvu.

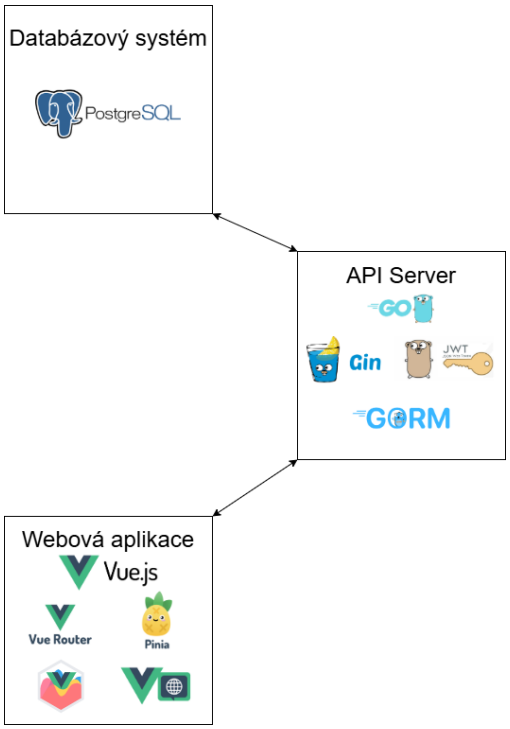
Databázová část slouží k uchovávání stálých informací, které se nemění s každým požadavkem. Patří sem především data o vozidlech, uživatelské účty a jejich preference.

Klientskou část tvoří webová aplikace, která poskytuje uživatelsky přívětivé rozhraní a zajišťuje komunikaci se serverem.

Celkové rozdělení aplikace je znázorněno na obrázku 6.1. V následujícím textu jsou popsány jednotlivé části systému, použité knihovny, balíčky a externí API, která jsou využívána napříč vrstvami.

6.2 Uživatelská část

Uživatelská část je tou nejdůležitější z pohledu klienta, jenž aplikaci využívá. Primárním úkolem této vrstvy je zpracovávat uživatelské požadavky a zobrazovat kontext tomu odpovídající. Důležité tedy bylo vytvořit jednoduché a přehledné uživatelské rozhraní, které nebude působit obtížně na používání.



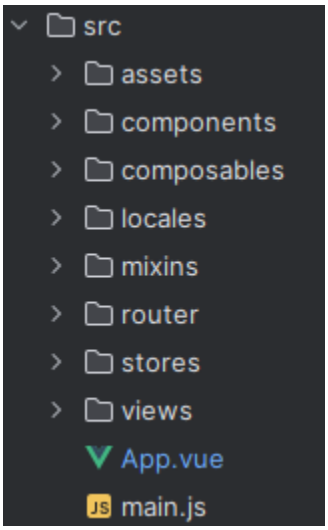
Obrázek 6.1: Architektura aplikace s využitými technologiemi

Uživatelské rozhraní bylo psáno ve vývojovém prostředí VueJS, jehož volba byla zdůvodněna v kapitole 3. Webová aplikace byla rozdělena do doporučované souborové struktury, jež lze vidět na obrázku 6.2.

Jednotlivé složky v rámci aplikační struktury mají své logické významy. Například složka `components` slouží k uchovávání všech komponent, které jsou v systému využívány. Složka mixins má za úkol uchovávat veškerou logiku, jež přímo nesouvisí s komponentami. Příkladem může být prostředník pro komunikaci s API serverem, kde bude definována kompletní logika, a ve zbytku aplikace bude k dispozici pouze jako funkce. Složka `composables` uchovává ty soubory, jejichž logika je opakovaně využívána. Příkladem může být soubor zastřešující autorizační logiku, do které spadá také snaha o obnovení vypršeného tokenu.

Hlavní částí aplikace je App.vue, jenž reprezentuje strukturu celé aplikace. V grafické struktuře souboru jsou veškeré komponenty, které jsou součástí po celou dobu běhu. Strukturu této komponenty lze vidět ve výpisu 6.1. Důležitým prvkem je RouterView, jenž s pomocí balíčku Vue-Router [51] dokáže zobrazovat požadované pohledy v rámci aplikace. Příklad definice routeru lze vidět ve výpisu B.2.

Jednotlivé stránky, jenž se vykreslují, jsou tvořeny uvnitř adresáře Views. Jejich význam spočívá v definování konkrétního pohledu, který chceme uživateli zobrazit. V rámci těchto pohledů se využívají také komponenty, jež jsou definovány ve složce Components. Jejich význam je v definici vlastních prvků systému, které mají funkční logiku a lze je využívat opakovaně. Příkladem může



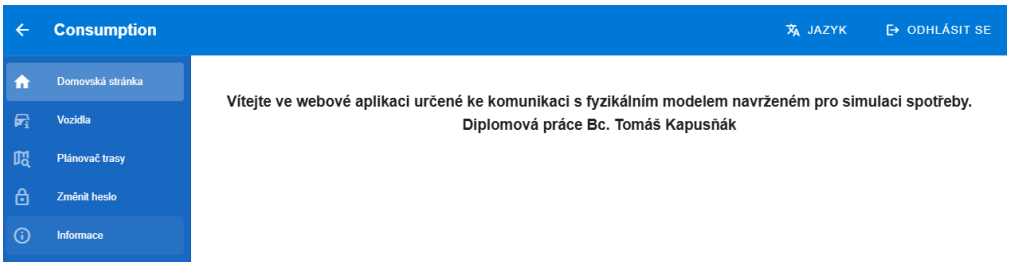
Obrázek 6.2: Souborová struktura webové aplikace

být komponenta Graph displayer, jejíž definici lze vidět ve výpisu B.3. Cílem této komponenty je přijímat libovolná data a zobrazovat je v uživatelsky přívětivém grafu s možností volby zobrazení definovaných hodnot.

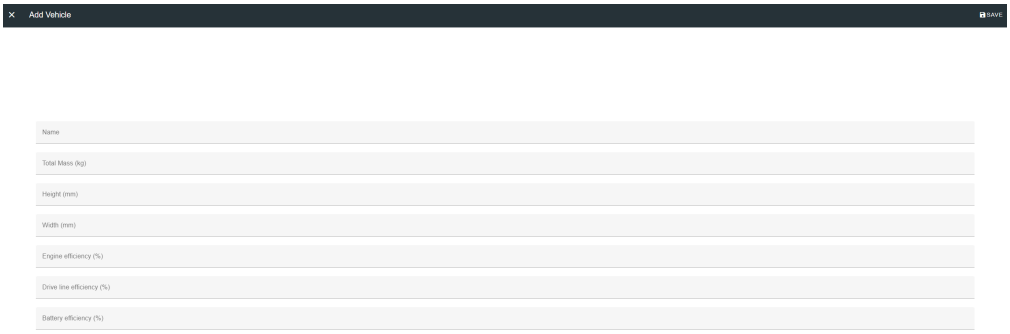
```
<template>
  <v-app>
    <Navbar />
    <Sidebar ref="sidebar" />
    <v-main class="scrollable">
      <transition name="fade" mode="out-in">
        <div>
          <RouterView />
        </div>
      </transition>
    </v-main>
    <Alert ref="alertComponent" />
    <LoginForm ref="loginForm" />
  </v-app>
</template>
```

Výpis 6.1: Definice vzhledu webové aplikace

Webová aplikace se skládá ze tří hlavních stránek - domovská obrazovka, výpis vozidel a plánovač jízdy. Hlavní stránka slouží k uvítání uživatele a k základnímu obeznámení uživatele o smyslu dané webové aplikace. Do budoucna zde mohou být přidány informace o aktualizacích a novinkách.

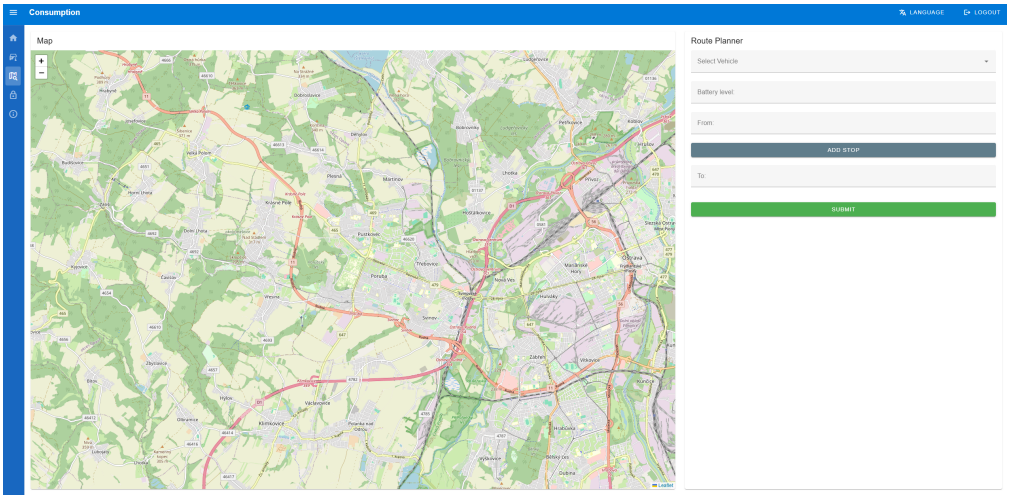


Obrázek 6.3: Hlavní stránka webové aplikace

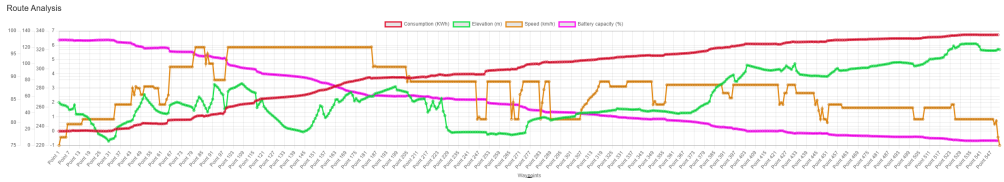


Obrázek 6.4: Formulář pro přidání nového vozidla

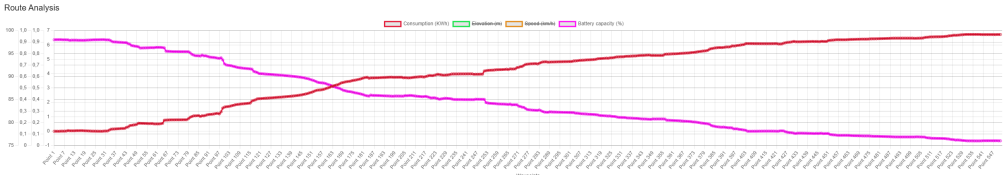
Vzhled hlavní stránky lze vidět na obrázku 6.3. Druhou možností v navigačním panelu je karta vozidla, jež uživatele přesměruje na výpis všech vozidel v systému. Mimo výpis je zde možnost vozidla editovat, mazat nebo přidávat nové. Během editace nebo přidávání nového vozidla se uživateli zobrazí jednoduchý formulář s hodnotami, které jsou nutné pro správné fungování simulačního modelu. Tento formulář lze vidět na obrázku 6.4. Třetí, a tou nejdůležitější možností, je plánovač trasy. Tato část nabízí uživateli jednoduchý přístup k simulačnímu modelu. Jak lze vidět na obrázku 6.5, stránka se skládá z mapy a formuláře. Pro spuštění simulace si musí uživatel vybrat vozidlo ze seznamu, následně zadá procentuální stav baterie před výjezdem a zvolí si jednotlivé body na trati. V rámci volby bodů je uživatel povinen určit startovní a cílový bod, zastávky jsou volitelné. Volbu lokace bodů může uživatel provést dvěma způsoby - volbou konkrétní pozice na mapě, nebo skrze vyhledávání konkrétní adresy skrze geolokační API, jenž je zabudované ve formuláři. Po vyplnění formuláře a jeho odeslání je uživateli zobrazena komponenta grafu, který lze vidět na obrázku 6.6. Zde uživatel zjistí průměrnou hodnotu na dané trati, výškový profil, vývoj kapacity akumulátoru a vývoj spotřebované energie. Uživatel má díky této komponentě možnost volby zobrazení hodnot, jež jsou do grafu vkládány. Tuto volbu lze vidět na obrázku 6.7. Samozřejmostí webové aplikace je také přihlašovací formulář nebo formulář na změnu hesla.



Obrázek 6.5: Ukázka navigační stránky



Obrázek 6.6: Ukázka výsledného grafu simulované jízdy



Obrázek 6.7: Ukázka výsledného grafu simulované jízdy se selekcí dat

6.2.1 Využité balíčky

V rámci vývoje uživatelské části bylo využito několik rozšíření a knihoven, které přispěly ke zvýšení efektivity vývoje a zlepšení uživatelského dojmu. Následující přehled uvádí nejdůležitější z nich.

Vuetify

Jednou z klíčových knihoven je Vuetify [52], která poskytuje sadu předpřipravených komponent pro vývojové prostředí Vue.js. Toto rozšíření výrazně zjednodušilo tvorbu uživatelského rozhraní, neboť eliminovalo potřebu rozsáhlejší ruční návrhářské práce a umožnilo rychle vytvářet moderní, responzivní komponenty. Vuetify vzniklo v roce 2016 jako open-source projekt pod vedením Johna Leidera a je dodnes aktivně rozvíjeno komunitou i samotným autorem. Mezi jeho výhody patří rozsáhlé možnosti konfigurace a vysoká míra přizpůsobení vzhledu.

Vue-router

Knihovna Vue Router [51] slouží jako směrovací mechanismus pro celou aplikaci. Umožňuje definovat jednotlivé pohledy aplikace a přepínat mezi nimi na základě změn URL. Je tedy zodpovědná za dynamické vykreslování komponent podle aktuálně aktivní trasy. Ukázka konfigurace routeru je uvedena ve výpisu B.2.

Pinia

Pro správu stavu aplikace byl využit stavový manažer Pinia [53], který v rámci Vue nahradil starší knihovnu Vuex [54]. Autorem knihovny je Eduardo San Martin Morote, jeden z vývojářů Vue. Mezi hlavní výhody Pinia patří jednodušší API, modularita a lepší podpora TypeScriptu [21]. V této práci Pinia zajišťuje správu dat o přihlášení uživatele, uchovávání JWT tokenů a také volbu jazykové mutace uživatelského rozhraní.

Vue Charts

Pro vizualizaci výsledků simulací byla použita knihovna Vue Charts [55], která staví na populárním balíčku Chart.js [56]. Výběr byl motivován předchozími pozitivními zkušenostmi s tímto nástrojem. Komponenty z Vue Charts umožnily vytvářet přehledné a interaktivní grafy s možností dynamického výběru zobrazovaných dat.

6.2.2 Využívané externí API

Jak bylo zmíněno v předchozích kapitolách, součástí webové aplikace je mapová komponenta propojující uživatele se simulačním modelem. Pro tuto funkcionalitu bylo nutné získávat GPS souřadnice výchozích, cílových a průjezdních bodů.

Za tímto účelem bylo využito geolokační API Nominatim [57], které umožňuje vyhledávání lokalit na základě textového vstupu. Uživatel tak může zadat libovolnou adresu, k níž API vrací odpovídající GPS souřadnice. Ty jsou následně použity pro výpočet trasy v rámci serverové části aplikace.

6.3 API server

Webová aplikace vyžaduje rozhraní, které umožní zpracování uživatelských požadavků a poskytování odpovědí. Za tímto účelem bylo vytvořeno API v jazyce Go [26]. Zdůvodnění volby jazyka lze nalézt v kapitole 3.

U této části systému bylo cílem vytvořit rozhraní, které zpřístupňuje data z databáze, provádět výpočty a také spravovat simulační model. API server byl rozdělen do dvou hlavních částí - databázové a serverové. První zmíněná část se starala o komunikaci s databází, migrace a nahrávání testovacích dat v testovacím režimu aplikace.

Serverová část obnášela veškerou aplikační logiku a byla rozdělena na tři další části. První byly kontroléry, které zpracovávaly a vyhodnocovaly data. V rámci této části lze nalézt také simulační model - ten je však zařazen do svého modulu s názvem ModelV2. Následně se v serverové části nachází modul modelů, jenž je předpisem objektů, které jsou v systému využívány. Některé z těchto modelů odpovídají také databázovým tabulkám. Posledním modulem v serverové části je security, jenž validuje platnost uživatelského JWT tokenu a také zastřešuje vytváření nových tokenů při přihlášení. Adresářová struktura tohoto API serveru lze vidět na obrázku 6.8

6.3.1 Využité knihovny

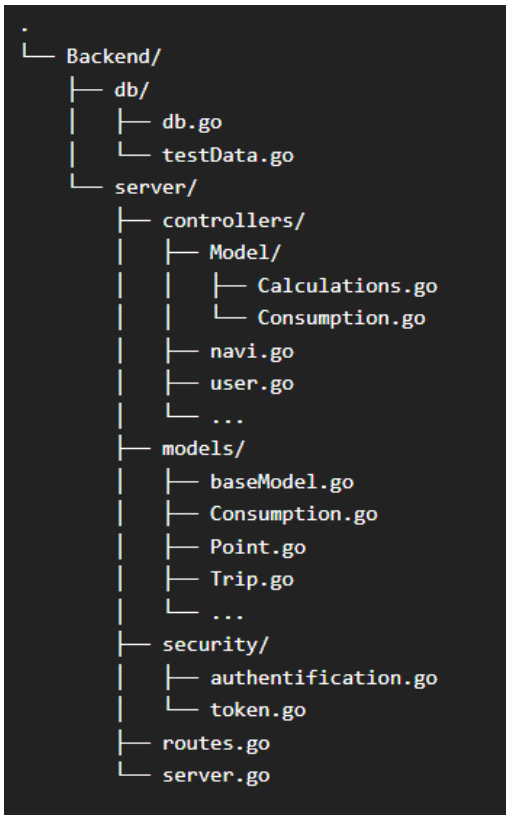
V rámci API serveru bylo nutné vyřešit klíčové části, jako je přístup k databázi nebo zpracování příchozích požadavků. V mnoha případech bylo jednodušší a často efektivnější využít již implementované řešení skrze externí knihovnu. Níže jsou popsány nejdůležitější z využitých balíčků.

Go Gin

Jako webová vývojová platforma byl využit Go Gin [28]. Jedná se o jedno z nejpoužívanějších rozšíření pro tvorbu API serveru. Mezi důvody, proč byla vybrána právě tato knihovna, patří například automatické zpracování a validace JSON dat, možnost seskupování jednotlivých API odkazů nebo její rychlost. Gin vznikl jako výkonnější náhrada za Martini API [58].

Golang JWT

Součástí API serveru je také bezpečnostní část, jež spočívá v generování a validaci JWT tokenů. Za tímto účelem byla využita knihovna Golang JWT [59]. V případě této aplikace je JWT token



Obrázek 6.8: Adresářová struktura API serveru

využíván k uchování uživatelského ID, jenž slouží k ověření identity uživatele, délku platnosti a identifikátor vystavovatele daného tokenu.

Gorm

Gorm [60] je jednou z nejpopulárnějších ORM knihoven pro jazyk Go. Často se udává, že využívání ORM není ideální z důvodu efektivity. Avšak v tomto případě je databáze vcelku jednoduchá a většina dotazů představuje operace čtení, zápisu, úpravy nebo mazání. Gorm přináší jednoduchou možnost migrace tabulek, generování dotazů a jejich následné mapování do objektů. Využití ORM bylo tedy upřednostněno z důvodu urychlení vývoje databázové logiky a zmenšení času stráveného psaním vlastních dotazů a zpracováním výsledků.

6.3.2 Využívané externí API

Za účelem simulace jízdy bez nutné získat GPS body odpovídající trase, aniž by byla předem fyzicky naměřena. K tomuto bylo využito externí navigační API od společnosti GraphHopper [50]. Jejich systém vyžaduje GPS polohu počátku trasy a cíle. Po odeslání požadavku získáme důležité polohy napříč trasou. Tyto pozice často reprezentují oblasti, kde dochází ke změně směru jízdy, zakřivení dráhy nebo změně rychlosti. V případě této aplikace se využívají ony body na trase, nadmořská výška a průměrná rychlost v daných bodech.

6.3.3 Klíčové komponenty

V rámci API serveru existuje řada komponent, bez kterých by aplikace správně nefungovala. Mimo samotný simulační model se jedná například o Uživatelský kontrolér, jenž se stará o přihlášení a obnovování platnosti JWT tokenů. Další část, která stojí za zmínku, je bezpečnostní modul, který se stará jak o autentizaci JWT tokenu, tak o jejich vytváření a validaci. V neposlední řadě stojí za zmínku navigační komponenta, jež slouží jako přístupový bod pro API, který na základě požadavku provede volání simulačního modelu a vyhodnocení výsledných dat. Níže naleznete detailnější popis fungování těchto částí.

Uživatelský kontrolér

Uživatelský kontrolér se stará o přihlášení uživatele i o obnovu platnosti tokenů. Mimo to zastřešuje také změnu uživatelského hesla.

Validace přihlášení je poměrně jednoduchá. V případě této aplikace funguje skrze validaci uloženého haše hesla vůči heslu zadanému ve formuláři. Validace má několik částí. První je kontrola JSON dat v požadavku, kdy pokud neobsahuje pouze konkrétní části, dochází k zamítnutí požadavku s kódem 401 a informací o chybějících datech. Druhou validací je kontrola uživatele vůči databázi. Pokud je tedy formulář korektní, dochází k hledání uživatele vůči zadanému uživatelskému jménu.

Pokud v tomto případě nebude uživatel nalezen, API server vrátí opět kód 401 se zprávou uživatel nenalezen. Třetí varianta, která může nastat, je, že uživatel zadá sice správné uživatelské jméno, ale kontrola haše hesla vrátí chybu. V tomto případě bude uživatel informován zprávou uživatel nenalezen s kódem 401. V případě, že hesla souhlasí, nastane situace vytváření tokenu, jenž může sice havarovat, o čemž bude uživatel informován, ale v případě, že vše proběhne korektně, nastane finální přihlášení uživatele, kdy je JSON vracen s kódem 200, zprávou Uživatel úspěšně přihlášen a daty nesoucími jak přístupový JWT token, tak i obnovovací. Diagram zobrazující fungování této funkcionality lze vidět na obrázku A.2.

Obnova platnosti tokenu je nezbytnou součástí systému. V případě, že by nedocházelo k obnovování přístupového tokenu, a uplynula by doba jeho platnosti, nedostával by již uživatel další odpovědi na své požadavky, a musel by se znovu přihlásit. Tímto způsobem by nastávala situace, kdy uživatel ztratí veškerý postup ve své práci a musí začít znovu. Z tohoto důvodu byla vytvořena logika ve webové aplikaci, jež po uplynutí platnosti přístupového tokenu zašle požadavek na obnovení skrze obnovovací token, který má delší platnost. API server následně ověří vstupní JSON data a provede validaci obnovovacího tokenu. V případě neplatnosti dojde k navrácení kódu 401 a hlášky o neplatnosti tokenu. V opačném případě dochází k znovu vytvoření obou tokenů a navrácení uživateli.

Autentizační komponenta

Autentizační komponenta je důležitým prvkem systému, jenž validuje příchozí přístupový token a nastavuje uživatelské ID pro celý kontext systému. Její princip fungování je vcelku jednoduchý. Nejdříve provede pokus o vytažení tokenu z hlavičky požadavku, kdy v případě nenalezení tokenu vrátí kód 401 a zprávu o chybějícím tokenu. V opačném případě dojde k rozklíčování tokenu a k jeho validaci. V případě špatného tokenu dojde k vrácení kódu 401 a odpovídající zprávě. Finálně, je-li token platný, nastaví se z tokenu správné uživatelské ID, v jehož kontextu bude API server dále provádět operace.

Navigační komponenta

Velice důležitou částí pro fungování simulačního modelu je navigační komponenta. Ta zastřešuje převod GPS souřadnic počátku trasy, průjezdných bodů a cíle na formát požadovaný externím API, jež vrací výslednou trasu, po které má vozidlo jet. Tělo požadavku na navigační API od společnosti GraphHopper lze vidět ve výpisu 6.2. V případě tohoto softwaru se využívají parametry points, které definují počáteční a koncový bod. Dalším parametrem těla požadavku je profile, jenž určuje typ vozidla, se kterým budeme simulovat. Nastavení tohoto parametru má vliv jak na zvolení trasy nebo nastavení rychlostí, tak na to, zda se má trasa vyhýbat zpoplatněným úsekům nebo rychlostním komunikacím. Jelikož simulační model vyžaduje i výškový profil trasy, je nutné nastavit parametr elevation na true, v opačném případě API nevrací informace o nadmořské výšce. Průměrnou rychlost

v jednotlivých bodech získáme skrze details, do nichž byl přidán požadavek na průměrnou rychlost. Finálním parametrem je points_encoded, který při nastavení na true vrací šifrované GPS hodnoty jednotlivých bodů.

```
{
    "points": [
        [
            18.158974,
            49.833007
        ],
        [
            18.494592,
            49.767721
        ]
    ],
    "profile": "car",
    "elevation": true,
    "locale": "cs_CZ",
    "points_encoded_multiplier": 1000000,
    "details": [
        "average_speed"
    ],
    "points_encoded": false
}
```

Výpis 6.2: Tělo požadavku na profil trasy

Jelikož API vrací data ve formě JSON, bylo nutné navrhnout způsob, jak je převést do struktur, které jsou přijatelné pro chod modelu. Příklad výsledného JSON souboru lze vidět ve výpisu B.4. Byla tedy implementována převodní funkce, která tato data transformuje do struktury kompatibilní se simulačním modelem. Následně jsou tato data vrácena a mohou být zaslána do simulačního modelu.

Simulační endpoint

Za účelem vystavení simulačního modelu byl vytvořen API endpoint. Za tímto odkazem se schovává funkce, jež zpracovává příchozí data a navrací výslednou spotřebu, procentuální stav baterie a vývoj stavu baterie. Tato funkce funguje vcelku jednoduše - nejdříve převede JSON na odpovídající struktury. Následně dochází k ověření, zda požadavek obsahuje nějaké mezizastávky nebo nikoli. Dle jejich počtu dochází k volbě cesty, kterou algoritmus půjde. Obecně lze říci, že algoritmus

vygeneruje trasu skrze navigační komponentu, na níž následně spustí simulaci. Výsledkem simulace je pak spotřeba energie v jednotlivých bodech, výsledný procentuální stav baterie a průběh stavu baterie. V případě, že požadavek obsahoval nějaké zastávky, dochází k této simulaci opakovaně, dokud nenarazí na cíl. Veškeré výsledky simulací jsou následně spojeny a vráceny. Vývojový diagram, jenž popisuje tuto funkci, lze vidět na obrázku A.3.

6.4 Databázová část

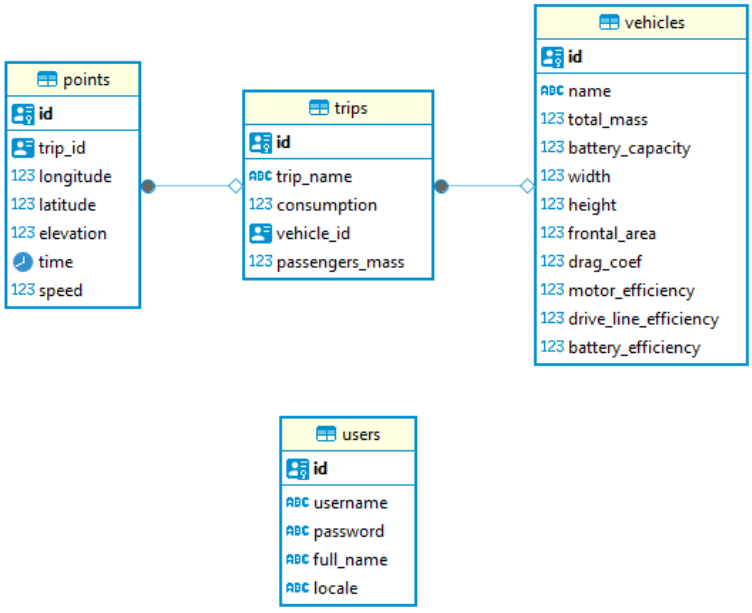
Databázová část byla postavena na technologii PostgreSQL [37]. Databáze se skládá ze čtyř tabulek - users, points, trips a vehicles. Vztahy mezi těmito tabulkami lze vidět v entitně-relačním diagramu na obrázku 6.9.

V aktuální verzi aplikace se již téměř nevyužívají tabulky points a trips. Jejich primární využití bylo v době testování modelu, kdy se data z měření vkládala do těchto tabulek, a následně již nebylo nutné číst tato data ze souborů. Tyto tabulky se tedy mohou zdát již nadbytečné, avšak v rámci budoucích rozšíření budou mít svůj význam.

Webová aplikace tedy momentálně využívá čistě tabulku users a vehicles. Tabulka users obsahuje platné uživatele systému, přesněji jejich uživatelské jméno, haš hesla a uživatelské preference, jež jsou nastaveny v rámci webové aplikace. Tabulka vehicles obsahuje seznam všech vozidel, která byla do systému přidána. Přesněji obsahuje název vozidla a veškeré parametry, jež jsou nezbytné ke správnému fungování simulačního modelu.

6.5 Možné rozšíření

Tato aplikace je v momentální fázi dobrý nástroj sloužící k plánování tras s odhadem spotřeby na základě simulace fyzikálním modelem. Avšak potenciál, který tato aplikace má, je značně rozsáhlejší. Pomocí minimálních úprav lze například přidat správu jízdních řádů pro bateriové autobusy, jež by umožnila odhad spotřeby s možností plánování efektivní trasy. K tomuto účelu lze využít již existující databázové tabulky trips a points, které by reprezentovaly jednotlivé linky. Dalším možným rozšířením je úprava fyzikálního modelu o simulaci vozidel na vodíkový pohon. Tato úprava by vyžadovala simulaci spotřeby vodíku, včetně dobíjení baterie a získání dalších souvisejících parametrů vozidla.



Obrázek 6.9: Entitně relační diagram databáze pro simulační aplikaci

Kapitola 7

Testování aplikace a fyzikálního modelu

Testování funkčnosti fyzikálního modelu a simulační části aplikace představovalo jednu z nejdůležitějších částí této práce. V případě, že by jednotlivé testy nevycházely, muselo by dojít k hlubší analýze a následné optimalizaci modelu, aby se výsledky co nejblíže podobaly realitě.

V rámci této kapitoly bude popsán návrh jednotlivých testů, průběh zaznamenávání dat, vyhodnocování správnosti modelu a porovnání reálné spotřeby. Dále pak porovnání simulace na záznamu z jízdy s kompletní simulací v případě generované trasy.

7.1 Návrh testů

V prvotní fázi vývoje modelu bylo nutné mít dostatečný objem dat, která by mohla sloužit k validaci výsledků. Za tímto účelem bylo vytvořeno několik testovacích jízd, v nichž byly rozdíly v typu trasy, její délce, rychlosti nebo také v samotném vozidle. Mezi testovaná vozidla patří Nissan Leaf ZE0 [47], Renault ZOE [61] a Hyundai Kona Electric [62]. Specifikace zmíněných vozidel lze vidět v tabulce 7.1. Cílem bylo pokrýt co největší množství možných variant. Proto byly testovány trasy s převahou dálnic, ale i okresních cest, trasy výrazně proti svahu i po svahu, či cesty v městském provozu. Většina jízd byla prováděna v teplotách pod nulou. Příklad některých testů lze vidět v seznamu níže.

- Trasa VŠB-TUO budova HARD → Bruntál
  - Start: VŠB-TUO budova HARD
  - Cíl: Bruntál
  - Délka trasy: 53.96 Km
  - Vozidlo: Nissan Leaf ZE0
  - Popis trasy: Cesta z části po dálnici s převážnou části do svahu.
- Trasa Hlučín → Opava-Suché Lazce,

– Start: Hlučín

– Cíl: Opava-Suché Lazce

– Délka trasy: 22.53 Km

– Vozidlo: Nissan Leaf ZE0

– Popis trasy: Krátká okresní jízda přes více obcí.

Tabulka 7.1: Seznam využívaných vozidel a jejich parametry

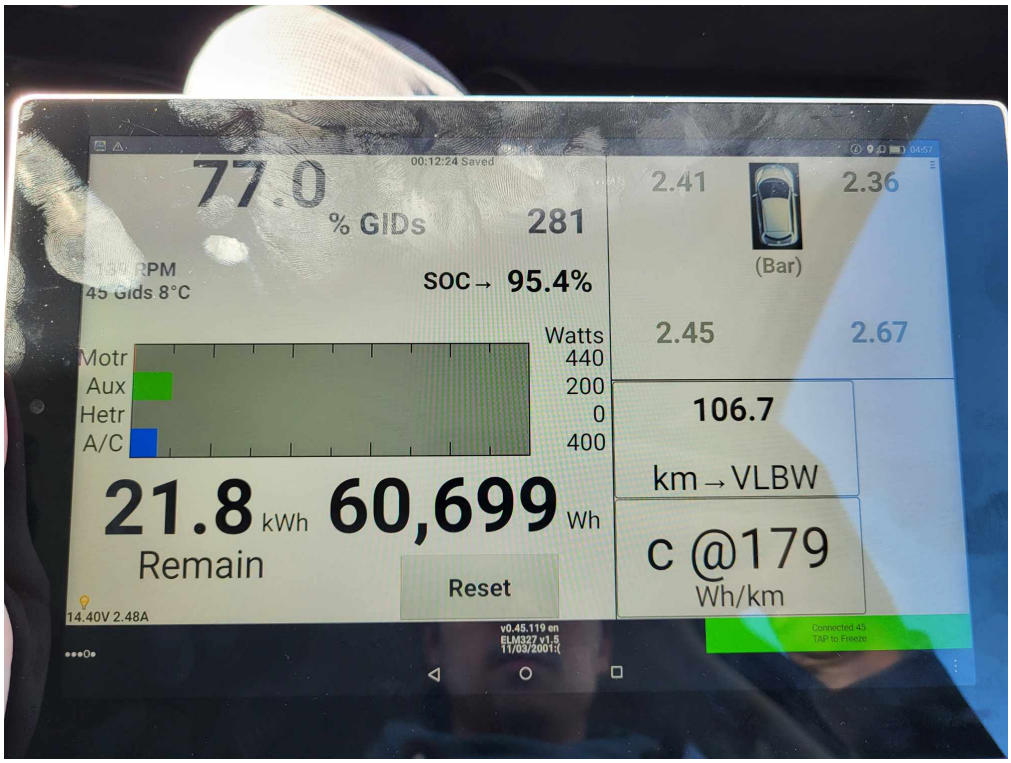
Název modelu	Nissan Leaf ZE0	Hyundai Kona Electric	Renualt ZOE
Rok výroby	2016	2019	2020
Výkon motoru (kW)	80	160	100
Kapacita baterie (kWh)	30	65.4	52
Váha vozidla (kg)	1630	1760	1502
Účinnost baterie (%)	98	98	98
Účinnost motoru (%)	98	98	98
Účinnost hnacího ústrojí (%)	95	95	95
Čelní plocha vozidla (m^2)	2.3	2.41	2.29
Aerodynamický koeficient	0.29	0.29	0.29

7.1.1 Zaznamenávání dat z jízd

Během testování bylo nutné zaznamenávat hodnoty sloužící k určení správnosti simulačního modelu. Za tímto účelem byla využita data z řídící jednotky a záznam polohy GPS z mobilu. V případě Nissan Leaf ZE0 byla využita také aplikace Leaf Spy Pro, jež dokázala uživatelsky přívětivě zpracovávat data skrze OBD2 modul a následně je také zasílat do databáze. Pro záznam polohy mobilním zařízením byla zvolena aplikace Sensor Logger, která je zdarma dostupná pro Android i iOS.

Leaf Spy Pro

Leaf Spy Pro je mobilní aplikace dostupná pro Android i iOS. Slouží ke komunikaci s vozidlem Nissan Leaf ZE0 skrze modul ELM327 OBDII. Tato aplikace byla primárním zdrojem dat o spotřebě energie vozidla v prvotní fázi testování. Zpočátku byla data ukládána v podobě CSV souborů a následně exportována pro analýzu a porovnání výsledků. V průběhu užívání byla také využita možnost automatického exportu do databáze již během jízdy. Toto vylepšení značně zjednodušilo práci s výslednými daty a přineslo možnost efektivnější zálohy těchto dat.



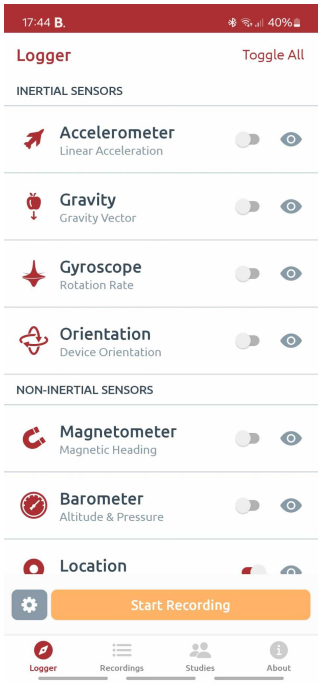
Obrázek 7.1: Hlavní obrazovka aplikace Leaf Spy Pro

Aplikace přináší vcelku příjemné uživatelské prostředí, ve kterém může uživatel sledovat aktuální hodnoty vozidla. Ukázku uživatelského rozhraní lze vidět na obrázku 7.1. Jak je na obrázku patrné, díky aplikaci dostáváme přívětivý pohled na aktuální nabití akumulátoru (SOC) nebo také na aktuální spotřebu některých částí vozidla. Aplikace udává i stav akumulátoru v kWh s ohledem na opotřebení baterie.

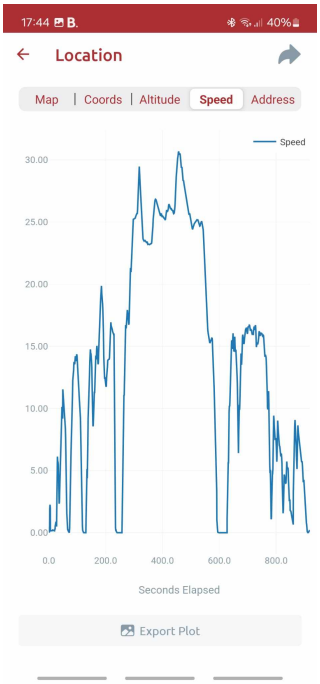
Sensor Logger

Sensor Logger byl nejdůležitější aplikací během provádění reálných testů. Jak již bylo zmíněno, záznam GPS polohy z řídící jednotky nebyl často příliš přesný nebo nebyl k dispozici vůbec. Aplikace nám sice nedává informace z řídící jednotky, a tedy již nezískáváme detailnější informace o spotřebě a stavu vozidla, avšak pro testy bylo důležité dosáhnout primárně totožné celkové spotřeby, jenž bylo možno získat z palubního počítače vozidla.

Sensor Logger poskytuje kompletní záznam senzorů z mobilního zařízení. Jedná se například o senzor polohy, gyroskop, akcelerometr nebo barometr. Pro účely této práce však postačoval záznam polohy skrze GPS modul. Frekvenci zaznamenávání bylo možno konfigurovat skrze menu v aplikaci. Uživatel má možnost volby mezi maximální frekvencí, jednou sekundou, deseti sekundami nebo vlastním časovým rozpětím mezi jednotlivými záznamy. V našem případě byla využita možnost



(a) Výběr senzorů k zaznamenávání v mobilní aplikaci Sensor Logger



(b) Graf rychlosti z mobilní aplikaci Sensor Logger

(c) Zobrazení trasy v mobilní aplikaci Sensor Logger

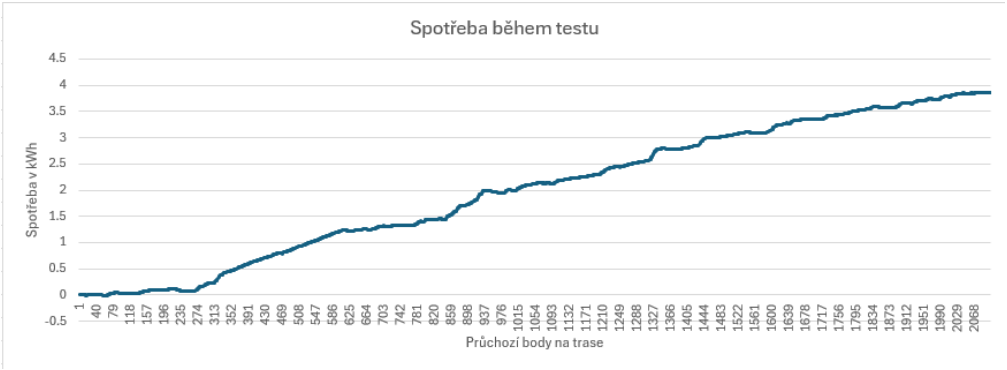
Obrázek 7.2: Obrázky z aplikace Sensor Logger

maximální frekvence záznamů s cílem co možná nejdetailnějšího pokrytí trasy. Vzhled aplikace lze vidět na skupině obrázků 7.2.

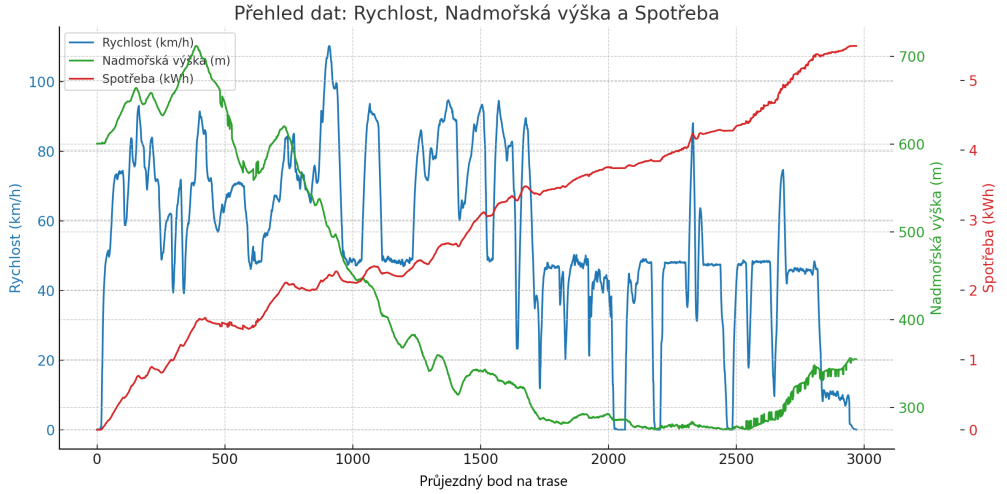
7.2 Vyhodnocení správnosti modelu

V průběhu vývoje fyzikálního modelu bylo nutné provádět validaci úprav a rozšíření. Za tímto účelem byly využívány vytvořené testy, na nichž se prováděla simulace a docházelo k detailnějšímu porovnání výsledků. Testování modelu probíhalo na úrovni API serveru, přímo na komponentě fyzikálního modelu. Nedocházelo tedy ke komunikaci skrze API ani k využívání webové aplikace. Díky tomu bylo jednodušší a rychlejší provádět testování při jednotlivých změnách.

Funkčnost modelu, a tedy jeho správnost, byla vyhodnocována na základě finální spotřeby a také na jejím průběhu vůči charakteru trasy. Příklad výsledného grafu spotřeby lze vidět na obrázku 7.3. Jak si lze na obrázku povšimnout, model validně zobrazuje rostoucí spotřebu v průběhu jízdy, avšak vykazuje také známky rekuperace energie. Pro lepší porovnání lze vidět na obrázku 7.4 vývoj spotřeby v závislosti na rychlosti vozidla a změně nadmořské výšky. Díky tomu lze vidět, že k rekuperaci dochází převážně v případě klesající nadmořské výšky.



Obrázek 7.3: Příklad výsledné spotřeby simulované fyzikálním modelem



Obrázek 7.4: Příklad výsledné spotřeby simulované fyzikálním modelem v porovnání s výškovým profilem a rychlostí

7.3 Porovnání výsledků se simulací

Finálním krokem testování bylo porovnat výslednou spotřebu v reálném provozu, simulací nad reálnými daty (nekompletní simulace) a kompletní simulací s generováním trasy. Pro účely této práce bude do porovnání přidána také aplikace Abetterrouteplanner [5] zkráceně ABRP, jež nabízí možnost generování trasy s odhadem spotřeby. Pro výpočet reálné spotřeby byl využit vzorec 7.1, kde C_{new} označuje kapacitu nové baterie, SOH představuje kondici baterie - přesněji se jedná o procentuální vyjádření aktuální kapacity vůči nové baterii, SOC odpovídá stavu nabití baterie, což SOC_0 představuje procentuální stav na počátku jízdy a SOC_n odpovídá stavu na konci jízdy. Například v případě Nissanu Leaf ZE0, jenž byl využíván během testování, byla původní kapacita 28 kWh a momentální stav SOH je cca 76 %, tedy aktuální maximální kapacita baterie odpovídá cca 21.28 kWh. S touto maximální kapacitou byla tedy určována výsledná reálná spotřeba.

$$Spotřeba = \frac{\frac{C_{new} \cdot SOH}{100} \cdot (SOC_0 - SOC_n)}{100} \quad \text{[kWh]} \tag{7.1}$$

Testovací jízda Hlučín → Opava-Suché Lazce

První testovací trasa, jež byla nasnímána, měla svůj počátek ve Hlučíně na ulici Moravská. Cílovou destinací byly Suché Lazce. Tato trasa byla vedena převážně po okresních komunikacích s četnými obcemi na trase. Tento test byl prováděn pomocí vozidla Nissan Leaf ZE0. Výslednou trasu pořízenou aplikací Sensor Logger lze vidět na obrázku 7.5, trasu, jenž byla generovaná skrze GraphHopper API lze vidět na obrázku 7.6.

Tabulka 7.2: Výsledky spotřeby na trase Hlučín → Opava-Suché Lazce

	Řídící jednotka	**Částečná simulace**	**Kompletní simulace**	**ABRP**	**WLTP**
Spotřeba (kWh)	3.57	3.30	3.25	3.70	3.53
Rozdíl vůči řídící jednotce (%)	0	7.56	8.96	3.64	1.12

Testovací jízda Dětmarovice-Koukolná → Opava-Suché Lazce

Druhou zásadní testovací jízdou byla z Koukolné směr Suché Lazce. Na této trase bylo využito opět vozidlo Nissan Leaf ZE0. Jízda byla z převážně po dálnici a následně po rychlostní komunikaci spojující Ostravu a Opavu. Na trase došlo ke dvěma zastávkám, konkrétně na ulici Ivana Kubice v Kravařích u Hlučína, části Dvořisko, a na ulici Májová v Opavě, části Suché Lazce.

Tabulka 7.3: Výsledky spotřeby na trase Dětmarovice-Koukolná → Opava-Suché Lazce

	Řídící jednotka	**Částečná simulace**	**Kompletní simulace**	**ABRP**	**WLTP**
Spotřeba (kWh)	9.80	9.55	10.40	10.50	9.88
Rozdíl vůči řídící jednotce (%)	0	2.55	6.12	7.14	0.81

Opava-Suché Lazce → Dětmarovice-Koukolná

Tato testovací trasa je téměř totožná s předchozím testem, avšak byla projeta v opačném směru a obsahuje jednu zastávku v areálu VŠB-TUO u budovy CEET. Během tohoto testu bylo využíváno vozidlo Nissan Leaf ZE0 z roku 2015 s původní kapacitou baterie 30 kWh. Tato trasa byla oproti předešlé více z kopce, bylo tedy očekáváno, že spotřeba bude znatelně nižší než v předchozím případě. Výsledky spotřeby lze vidět v tabulce 7.4.

Tabulka 7.4: Výsledky spotřeby na trase Suché Lazce → Dětmarovice-Koukolná

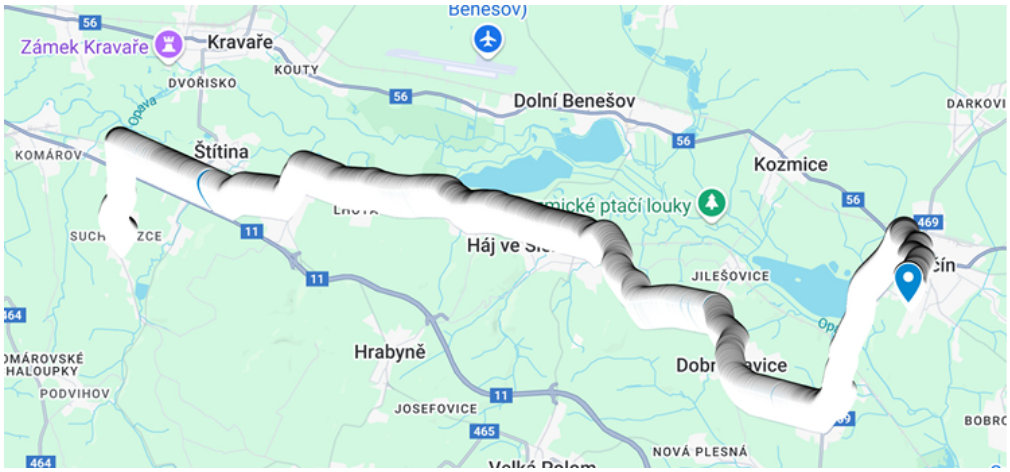
	Řídící jednotka	**Částečná simulace**	**Kompletní simulace**	**ABRP**	**WLTP**
Spotřeba (kWh)	9.03	8.21	8.16	9.8	9.13
Rozdíl vůči řídící jednotce (%)	0	9.08	9.63	8.53	1.11

Testovací jízda VŠB-TUO budova HARD → Opava-Suché Lazce

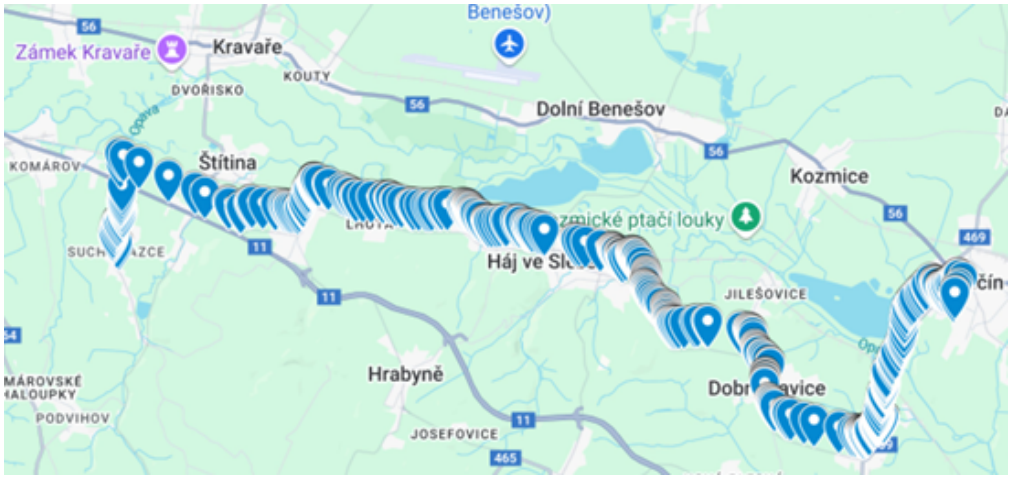
Tato testovací trasa byla prvním testem na jiném vozidle než-li Nissan Leaf ZE0. Jednalo se o Hyundai Kona Electric ve variantě s 65 kWh baterií [62]. Tato trasa měla počátek na parkovišti VŠB-TUO u budovy těžkých laboratoří (tzv. HARD). Cíl byl opět v obci Suché Lazce na Opavsku. Trasa vede po rychlostní komunikaci z Ostravy na Opavu. V případě této trasy nebyly provedeny žádné zastávky. Výsledky této jízdy lze nalézt v tabulce 7.5. Nutno však podotknout, že v tomto případě byla spotřeba vypočítána skrze palubní počítač, který mohl ukazovat částečně nepřesný výsledek. Spotřebu dle palubního počítače lze vidět na obrázku A.4.

Tabulka 7.5: Výsledky spotřeby na trase VŠB-TUO budova HARD → Opava-Suché Lazce

	Řídící jednotka	**Částečná simulace**	**Kompletní simulace**	**ABRP**	**WLTP**
Spotřeba (kWh)	4.18	3.61	3.40	3.10	2.81
Rozdíl vůči řídící jednotce (%)	0	13.64	18.66	25.84	32.76



Obrázek 7.5: Mapa zobrazující zaznamenané body na trase Hlučín → Suché Lazce



Obrázek 7.6: Mapa zobrazující vygenerované body na trase Hlučín → Suché Lazce

Testovací jízda VŠB-TUO budova HARD → Kravaře-Dvořisko

Testovací jízda s počátkem na parkovišti VŠB-TUO u budovy HARD s cílem na ulici Ivana Kubice v Kravařích u Hlučína, části Dvořisko, sloužila k otestování modelu na jiném vozidle než doposud. Zvolen byl elektromobil Renault ZOE R135 [61]. Reálná spotřeba byla určena na základě hodnoty palubního počítače, jež lze vidět na obrázku A.5. Délka této trasy činila 17.7 km. Výsledná reálná spotřeba byla tedy stanovena na 2.90 kWh.

Tabulka 7.6: Výsledky spotřeby na trase VŠB-TUO budova HARD → Kravaře-Dvořisko

	Řídící jednotka	**Částečná simulace**	**Kompletní simulace**	**ABRP**	**WLTP**
Spotřeba (kWh)	2.90	2.72	2.76	2.70	3.06
Rozdíl vůči řídící jednotce (%)	0	6.20	4.83	6.90	5.52

Testovací jízda Bruntál → Opava-Suché Lazce

Zajímavou trasou pro porovnání je Bruntál → Opava-Suché Lazce. Trasa je převážně ze svahu. Lze tedy očekávat, že na trase z Bruntálu bude výrazně nižší spotřeba než v případě opačném. Na této trase byl využit elektromobil Nissan Leaf ZE0 z roku 2016. Tento konkrétní test měl počátek v Bruntále na ulici Opavská, s cílem v obci Suché Lazce na ulici Přerovecká. Trasa vedla skrze Opavu, tedy městským provozem, jinak převážně po okresních cestách. Výsledky spotřeby lze vidět v tabulce 7.7.

Tabulka 7.7: Výsledky spotřeby na trase Bruntál → Opava-Suché Lazce

	Řídící jednotka	**Částečná simulace**	**Kompletní simulace**	**ABRP**	**WLTP**
Spotřeba (kWh)	5.75	5.63	5.39	6.00	8.02
Rozdíl vůči řídící jednotce (%)	0	2.09	6.26	4.35	39.48

Testovací jízda Opava-Suché Lazce → Bruntál

Jak bylo v předešlém testu zmíněno, trasa mezi Bruntálem a obcí Suché Lazce je vcelku zajímavá svou změnou v nadmořské výšce, přesněji Bruntál leží ve vyšší nadmořské výšce než-li Suché Lazce. V tomto testu měla trasa počátek v obci Suché Lazce a cílem byla nabíjecí stanice ČEZ. Kvůli nejlepšímu porovnání výsledné spotřeby s předešlým testem byl opět využit elektromobil Nissan Leaf ZE0. V tomto případě navíc nevedla trasa skrze centrum Opavy, nýbrž jejím obchvatem. Trasa byla převážně po okresních cestách, městský provoz byl až v cílovém Bruntále. Výsledné spotřeby lze vidět v tabulce 7.8.

Tabulka 7.8: Výsledky spotřeby na trase Opava-Suché Lazce → Bruntál

	Řídící jednotka	**Částečná simulace**	**Kompletní simulace**	**ABRP**	**WLTP**
Spotřeba (kWh)	9.37	7.83	8.05	8.60	8.07
Rozdíl vůči řídící jednotce (%)	0	16.44	14.09	8.21	13.87

Testovací jízda Ostrava-Svinov → Hlučín

Trasa z Hornbachu v Ostrava-Svinov do Hlučína je vcelku zajímavá, jelikož je z malé části vedena po ulici Opavská v Ostravě-Porubě a následně pokračuje přes Ostrava-Martinov dále na Děhylov. Trasa je tedy vedena skrze okresní cesty, ale také ze značné části skrze obce. Pro tento test bylo využito vozidlo Nissan Leaf ZE0. Výsledky spotřeby v tomto testu lze vidět v tabulce 7.9.

Tabulka 7.9: Výsledky spotřeby na trase Ostrava-Svinov → Hlučín

	Řídící jednotka	**Částečná simulace**	**Kompletní simulace**	**ABRP**	**WLTP**
Spotřeba (kWh)	1.68	1.73	1.53	2.00	1.91
Rozdíl vůči řídící jednotce (%)	0	2.98	8.93	19.05	13.69

Testovací jízda VŠB-TUO budova HARD → Hornbach Ostrava-Svinov

Test na trase s počátkem na parkovišti VŠB-TUO u budovy HARD a cílem v Hornbachu v Ostravě-Svinově je nejkratší testovanou trasou v rámci této práce. Trasa vedla skrze ulici Hlavní třída v Ostravě-Porubě a následně po ulici Polská a Mongolská. Testovacím elektromobilem byl Nissan Leaf ZE0. Trasa byla o délce cca 5.56 km. Výsledky spotřeby lze vidět v tabulce 7.10.

Tabulka 7.10: Výsledky spotřeby na trase VŠB-TUO budova HARD → Hornbach Ostrava-Svinov

	Řídící jednotka	**Částečná simulace**	**Kompletní simulace**	**ABRP**	**WLTP**
Spotřeba (kWh)	0.84	0.55	0.28	0.7	0.91
Rozdíl vůči řídící jednotce (%)	0	32.52	66.67	16.67	8.33

7.4 Vyhodnocení kvality modelu

Na základě zmíněných testů lze říct, že simulační model je schopen fungovat s odchylkou do 10 procent. Jsou zde výjimečné případy, kdy jsou odchylky znatelně vyšší. Příkladem může být test krátké

trasy VŠB - Hornbach Ostrava-Svinov, kde odchylka dosahovala až 66.67 % v případě kompletní simulace. Nutno však podotknout, že v případě simulace nedochází k náhodnému zpomalování a následnému zrychlování (např. na přechodu nebo semaforu) nebo také že model nepodporuje spotřebu skrze topení, jež má v případě využívaného modelu značný vliv během nízkých teplot. Vliv topení byl také testován a bylo zjištěno, že v prvotní fázi vytápění, tedy když vozidlo i baterie mají venkovní teplotu, spotřebovává cca 2 kWh a následně slábne na cca 0.5 kWh. Další významná odchylka byla zjištěna při testování vozu Hyundai Kona Electric, kde hodnota odchylky dosahovala téměř 19 %. V tomto případě je možné, že palubní počítač vozidla nadhodnocuje spotřebu energie, což vede k výrazně vyšší hodnotě spotřeby, než jaká byla predikována simulací.

Ve výsledku lze tvrdit, že simulační model je schopen pokrýt většinu tras s dostatečnou přesností pro naplánování potřebných zastávek na dobíjení vozidla. Zároveň bylo dokázáno, že navigační API je schopno generovat konkrétní požadovanou trasu uživatele, popřípadě navrhnout optimálnější trasu s ohledem na délku trasy a čas potřebný k jejímu překonání.

7.5 Rozšíření modelu o možnost simulace vodíkových vozidel

V době psaní této práce již probíhá vývoj rozšíření modelu, které umožní simulaci vodíkových vozidel. Jako testovací vozidlo byla zvolena Toyota Mirai druhé generace [63], a to díky její dostupnosti v rámci výzkumného ústavu CEET na VŠB-TUO. Vodíková vozidla se v současnosti velmi podobají čistým elektromobilům, přičemž hlavní rozdíly spočívají v kapacitě baterie a způsobu jejího dobíjení. Cílem tohoto rozšíření tedy není pouze výpočet výsledné spotřebované energie, ale také modelování průběhu nabíjení baterie a spotřeby vodíku. Pro navržení optimálního rozšíření bude nezbytné provést sérii reálných měření, jejichž cílem bude sledovat proces dobíjení akumulátoru a konkrétně určit, při jakých procentech kapacity baterie začíná a končí její dobíjení. Dalším klíčovým bodem bude sledování změn výkonu článků pro dobíjení baterie v závislosti na kapacitě, což ovlivní i spotřebu vodíku. V současnosti existuje první hrubý návrh, ve kterém je dobíjení simulováno konstantním výkonem. Práce na tomto rozšíření tedy pokračuje s cílem co nejpřesněji odhadnout spotřebu vodíku.

Kapitola 8

Závěr

Cílem této diplomové práce bylo navrhnout a realizovat webovou aplikaci, která umožňuje vizualizaci jízdy elektromobilu a zároveň provádí simulaci spotřeby elektrické energie na základě fyzikálního modelu (viz kapitola 4). V rámci řešení bylo dosaženo plné funkcionality aplikace, jež kombinuje přesné výpočty se snadno použitelným uživatelským rozhraním. Výsledkem je komplexní nástroj, který umožňuje plánování trasy a přehled jejích vlastností, jako je výškový profil, očekávaná rychlost nebo odhadovaná spotřeba energie (podrobnosti v kapitole 6).

Fyzikální model vyvinutý v rámci této práce je založen na ověřených principech mechaniky (viz kapitola 4). Model zohledňuje klíčové veličiny, jako je odpor vzduchu, valivý odpor, odpor stoupání i setrvačné síly. Kromě těchto sil je započítána i efektivita jednotlivých komponent elektromobilu – především účinnost motoru, převodového ústrojí a samotné baterie. Výsledná simulace poskytuje uživateli detailní výpočet energetické náročnosti celé trasy a umožňuje predikovat zbytkovou kapacitu baterie po skončení jízdy (viz kapitola 7).

Aplikace byla implementována jako webová platforma, která komunikuje se serverovým rozhraním a databází (viz kapitola 3). Klientská část poskytuje interaktivní rozhraní s možností plánování tras pomocí mapového podkladu, zadání technických parametrů vozidla a zobrazení výsledků simulace ve formě grafů. Serverová logika provádí výpočty na základě zadaných vstupních dat a využívá fyzikální model. Celý systém byl navržen s ohledem na rozšiřitelnost a budoucí možnosti integrace dalších funkcionalit (podrobnosti v kapitole 6).

V průběhu práce byla také podrobně popsána data, která model využívá pro výpočty spotřeby energie, včetně jejich formátu a způsobu získávání (viz kapitola 5). Výsledky testování ukázaly, že simulovaný výpočet spotřeby vykazuje vysokou míru přesnosti. Rozdíly mezi reálnou a vypočítanou spotřebou se v rámci testovacích scénářů pohybovaly v přijatelných mezích, a to i při simulaci delších tras (více v kapitole 7). Model je tak vhodný nejen pro běžné uživatele, ale i pro nasazení v profesionálních aplikacích, například při správě flotily elektromobilů nebo při plánování energeticky optimalizovaných tras (viz kapitola 7).

Do budoucna se nabízí několik směrů rozšíření. Mezi nejzajímavější patří zohlednění vlivu počasí nebo dynamického provozu. Užitečná by mohla být i podpora navigačního režimu nebo export výsledků pro další analýzu. Tato práce přináší nástroj, který podporuje širší adopci elektromobility a přispívá k udržitelnému způsobu dopravy (více v kapitole 6).

Literatura

1. EUROPEAN AUTOMOBILE MANUFACTURERS' ASSOCIATION (ACEA). *New Passenger Car Registrations in the EU*. 2025. Dostupné také z: https://www.acea.auto/figure/new-passenger-car-registrations-in-eu/. Accessed: 2025-04-12.
2. EUROPEAN ENVIRONMENT AGENCY. *New registrations of electric vehicles in Europe*. 2024. Dostupné také z: https://www.eea.europa.eu/en/analysis/indicators/new-registrations-of-electric-vehicles. Accessed: 2025-03-09.
3. TURBO3. *Leaf Spy Pro* [online]. 2024. [cit. 2024-05-20]. Dostupné z: https://play.google.com/store/apps/details?id=com.Turbo3.Leaf_Spy_Pro&hl=cs&gl=US&pli=1.
4. KELVIN, Inc. *Sensor Push* [online]. 2024. [cit. 2024-05-20]. Dostupné z: https://play.google.com/store/apps/details?id=com.kelvin.sensorapp&hl=cs&gl=US.
5. AB, Iternio Planning. *A Better Routeplanner (ABRP)*. 2025. Dostupné také z: https://abetterrouteplanner.com/. Accessed: 2025-03-09.
6. GPS TUNER SYSTEMS KFT. *EV Navigation: EV Range and Smart Route Planner*. 2025. Dostupné také z: https://evnavigation.com/. Accessed: 2025-03-09.
7. BERMEJO, David Jimenez; HERNANDEZ, Sara; FRAILE-ARDANUY, Jesus; ROMERO, Javier Serrano; POZO, Rubén; ALVAREZ, Federico. Modelling the Effect of Driving Events on Electrical Vehicle Energy Consumption Using Inertial Sensors in Smartphones. *Energies*. 2018-02, roč. 11, s. 412. Dostupné z DOI: 10.3390/en11020412.
8. FOTOUHI, Abbas; AUGER, Daniel; PROPP, Karsten; LONGO, Stefano. Simulation for prediction of vehicle efficiency, performance, range and lifetime: A review of current techniques and their applicability to current and future testing standards. In: *IET Hybrid and Electric Vehicles Conference 2014*. 2014-11. Dostupné z DOI: 10.1049/cp.2014.0959.
9. GENIKOMSAKIS, Konstantinos N.; MITRENTSIS, Georgios. A computationally efficient simulation model for estimating energy consumption of electric vehicles in the context of route planning applications. *Transportation Research Part D: Transport and Environment*. 2017, roč. 50, s. 98–118. ISSN 1361-9209. Dostupné z DOI: https://doi.org/10.1016/j.trd.2016.10.014.

10. (NREL), National Renewable Energy Laboratory. *FASTSim: Advanced Vehicle Simulator* [online]. 2024. [cit. 2024-05-20]. Dostupné z: https://www.nrel.gov/transportation/fastsim.html.

11. FIORI, Chiara; AHN, Kyoungho; RAKHA, Hesham A. Power-based electric vehicle energy consumption model: Model development and validation. *Applied Energy*. 2016, roč. 168, s. 257–268. ISSN 0306-2619. Dostupné z DOI: https://doi.org/10.1016/j.apenergy.2016.01.097.

12. THE PHP GROUP. *PHP: Hypertext Preprocessor*. 2025. Dostupné také z: https://www.php.net/. Accessed: 2025-03-09.

13. THE PALLETS PROJECTS. *Flask Documentation (3.1.x)*. 2025. Dostupné také z: https://flask.palletsprojects.com/en/stable/. Accessed: 2025-03-09.

14. THE PYTHON SOFTWARE FOUNDATION. *Python Programming Language – Official Website*. 2025. Dostupné také z: https://www.python.org/. Accessed: 2025-03-09.

15. THE REACT TEAM AT META. *React – A JavaScript library for building user interfaces*. 2025. Dostupné také z: https://react.dev/. Accessed: 2025-03-09.

16. EVAN YOU AND THE VUE.JS CORE TEAM. *Vue.js – The Progressive JavaScript Framework*. 2025. Dostupné také z: https://vuejs.org/. Accessed: 2025-04-17.

17. ANGULAR TEAM AT GOOGLE. *Angular – The web development framework for building modern apps*. 2025. Dostupné také z: https://angular.dev/. Accessed: 2025-04-19.

18. GOOGLE LLC. *Google Trends*. 2025. Dostupné také z: https://trends.google.com/trends/. Accessed: 2025-04-19.

19. NPMTRENDS.COM. *npm trends – Compare NPM package downloads*. 2025. Dostupné také z: https://npmtrends.com/. Accessed: 2025-04-19.

20. KROTOFF, Tanguy. *Front-end frameworks popularity (React, Vue, Angular and Svelte)*. 2025. Dostupné také z: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190. Accessed: 2025-04-19.

21. MICROSOFT. *TypeScript: JavaScript with Syntax for Types*. 2025. Dostupné také z: https://www.typescriptlang.org/. Accessed: 2025-03-09.

22. ORACLE CORPORATION. *Java – The #1 programming language and development platform*. 2025. Dostupné také z: https://www.java.com/en/. Accessed: 2025-04-19.

23. MICROSOFT CORPORATION. *.NET – Build. Test. Deploy.* 2025. Dostupné také z: https://dotnet.microsoft.com/en-us/. Accessed: 2025-04-19.

24. RUST PROJECT DEVELOPERS. *Rust Programming Language*. 2025. Dostupné také z: https://www.rust-lang.org/. Accessed: 2025-04-19.

25. RUBY CORE TEAM. *Ruby Programming Language*. 2025. Dostupné také z: https://www.ruby-lang.org/en/. Accessed: 2025-04-19.

26. GOOGLE, The Go Team at. *The Go Programming Language*. 2025. Dostupné také z: https://go.dev/. Accessed: 2025-03-09.

27. SPRING TEAM. *Spring Boot – Spring Framework*. 2025. Dostupné také z: https://spring.io/projects/spring-boot. Accessed: 2025-04-19.

28. MARTINEZ-ALMEIDA, Manu; COMMUNITY, Gin-Gonic. *Gin Web Framework*. 2025. Dostupné také z: https://gin-gonic.com/. Accessed: 2025-03-09.

29. RAMÍREZ, Sebastián. *FastAPI – Modern, Fast (High-performance) Web Framework for Building APIs with Python*. 2025. Dostupné také z: https://fastapi.tiangolo.com/. Accessed: 2025-04-19.

30. LUONG, Travis. *FastAPI vs Fastify vs Spring Boot vs Gin Benchmark*. 2025. Dostupné také z: https://www.travisluong.com/fastapi-vs-fastify-vs-spring-boot-vs-gin-benchmark/. Accessed: 2025-03-09.

31. REFERENCE, Deno - The Complete. *Go Gin vs Spring Boot WebFlux: Hello World Performance Comparison*. 2025. Dostupné také z: https://medium.com/deno-the-complete-reference/go-gin-vs-springboot-webflux-hello-world-performance-comparison-e25fd8d85216. Accessed: 2025-03-09.

32. GOOGLE LLC. *Google Trends for Programming Languages*. 2025. Dostupné také z: https://trends.google.com/trends/explore?date=today%205-y&geo=CZ&q=%2Fm%2F09gbxjr,%2Fm%2F07sbkfb,%2Fm%2F05z1_&hl=cs. Accessed: 2025-04-19.

33. GERRAND, Andrew. *Share Memory By Communicating*. 2010. Dostupné také z: https://go.dev/blog/codelab-share. Accessed: 2025-04-17.

34. MICROSOFT CORPORATION. *Microsoft SQL Server – Data Platform for Hybrid Scenarios*. 2025. Dostupné také z: https://www.microsoft.com/en-us/sql-server. Accessed: 2025-04-19.

35. ORACLE CORPORATION. *Oracle SQL – SQL Language*. 2025. Dostupné také z: https://www.oracle.com/database/technologies/appdev/sql.html. Accessed: 2025-04-19.

36. ORACLE CORPORATION. *MySQL – The World's Most Popular Open Source Database*. 2025. Dostupné také z: https://www.mysql.com/. Accessed: 2025-04-19.

37. GROUP, PostgreSQL Global Development. *PostgreSQL: The World's Most Advanced Open Source Relational Database*. 2025. Dostupné také z: https://www.postgresql.org/. Accessed: 2025-03-09.

38. REDGATE SOFTWARE. *DB-Engines – Knowledge Base of Relational and NoSQL Database Management Systems*. 2025. Dostupné také z: https://db-engines.com/. Accessed: 2025-04-19.

39. GITHUB, INC. *GitHub – Build and ship software on a single, collaborative platform*. 2025. Dostupné také z: https://github.com/. Accessed: 2025-04-19.

40. ATLASSIAN CORPORATION PLC. *Bitbucket – Git solution for teams using Jira*. 2025. Dostupné také z: https://bitbucket.org/product/. Accessed: 2025-04-19.

41. GITLAB INC. *GitLab – The most comprehensive AI-powered DevSecOps platform*. 2025. Dostupné také z: https://about.gitlab.com/. Accessed: 2025-04-19.

42. NĚMEC, Jan. *Vizualizace železniční sítě a výpočty energetické náročnosti jízdy vlaku* [online]. 2024. [cit. 2025-03-09]. Dostupné z: http://hdl.handle.net/10084/153890. Dipl. pr. VSB-TUO.

43. WIJK, Marco van. *Driving Resistances*. 2025. Dostupné také z: https://www.mvwautotechniek.nl/en/driving-resistances/. Accessed: 2025-04-19.

44. THE ENGINEERING TOOLBOX. *Rolling Resistance*. 2025. Dostupné také z: https://www.engineeringtoolbox.com/rolling-friction-resistance-d_1303.html. Accessed: 2025-04-19.

45. RIBBENS, William B. *Electric Vehicle Technology* [online]. 2013. [cit. 2025-03-09]. Dostupné z: https://www.iqytechnicalcollege.com/BAE%20685-Electric%20Vehicle%20Technology.pdf.

46. BERMEJO, David Jimenez; HERNANDEZ, Sara; FRAILE-ARDANUY, Jesus; ROMERO, Javier Serrano; POZO, Rubén; ALVAREZ, Federico. Modelling the Effect of Driving Events on Electrical Vehicle Energy Consumption Using Inertial Sensors in Smartphones. *Energies*. 2018-02, roč. 11, s. 412. Dostupné z DOI: 10.3390/en11020412.

47. AUTOCAR. *2016 Nissan Leaf 30kWh First Drive* [online]. 2024. [cit. 2024-05-20]. Dostupné z: https://www.autocar.co.uk/car-review/nissan/leaf-2011-2017/first-drives/2016-nissan-leaf-30kwh-first-drive.

48. GOOGLE LLC. *Google Maps Platform Documentation*. 2025. Dostupné také z: https://developers.google.com/maps/documentation. Accessed: 2025-04-19.

49. TOMTOM INTERNATIONAL BV. *TomTom Routing APIs*. 2025. Dostupné také z: https://www.tomtom.com/products/routing-apis/. Accessed: 2025-04-19.

50. GMBH, GraphHopper. *GraphHopper: Open Source Routing Engine*. 2025. Dostupné také z: https://www.graphhopper.com/. Accessed: 2025-03-09.

51. TEAM, Vue.js. *Vue Router Documentation*. 2025. Dostupné také z: https://router.vuejs.org/. Accessed: 2025-03-09.

52. TEAM, Vuetify. *Vuetify Documentation*. 2025. Dostupné také z: https://vuetifyjs.com/en/. Accessed: 2025-03-09.

53. TEAM, Vue.js. *Pinia Documentation*. 2025. Dostupné také z: https://pinia.vuejs.org/. Accessed: 2025-03-09.

54. TEAM, Vue.js. *Vuex Documentation*. 2025. Dostupné také z: https://vuex.vuejs.org/. Accessed: 2025-03-09.

55. JUSZCZAK, Jakub. *Vue Chart.js Documentation*. 2019. Dostupné také z: https://vue-chartjs.org/. Accessed: 2025-03-09.

56. DOWNIE, Nick; CONTRIBUTORS, Chart.js. *Chart.js Documentation*. 2025. Dostupné také z: https://www.chartjs.org/. Accessed: 2025-03-09.

57. CONTRIBUTORS, OpenStreetMap. *Nominatim: Search for OpenStreetMap Data*. 2025. Dostupné také z: https://nominatim.org/. Accessed: 2025-03-09.

58. SAENZ, Jeremy. *Martini: A Go Web Framework*. 2025. Dostupné také z: https://github.com/go-martini/martini. Accessed: 2025-03-09.

59. COMMUNITY, Golang-JWT. *Golang-JWT: A JWT Implementation for Go*. 2025. Dostupné také z: https://golang-jwt.github.io/jwt/. Accessed: 2025-03-09.

60. ZHANG, Jinzhu; COMMUNITY, GORM. *GORM: The Fantastic ORM Library for Golang*. 2025. Dostupné také z: https://gorm.io/index.html. Accessed: 2025-03-09.

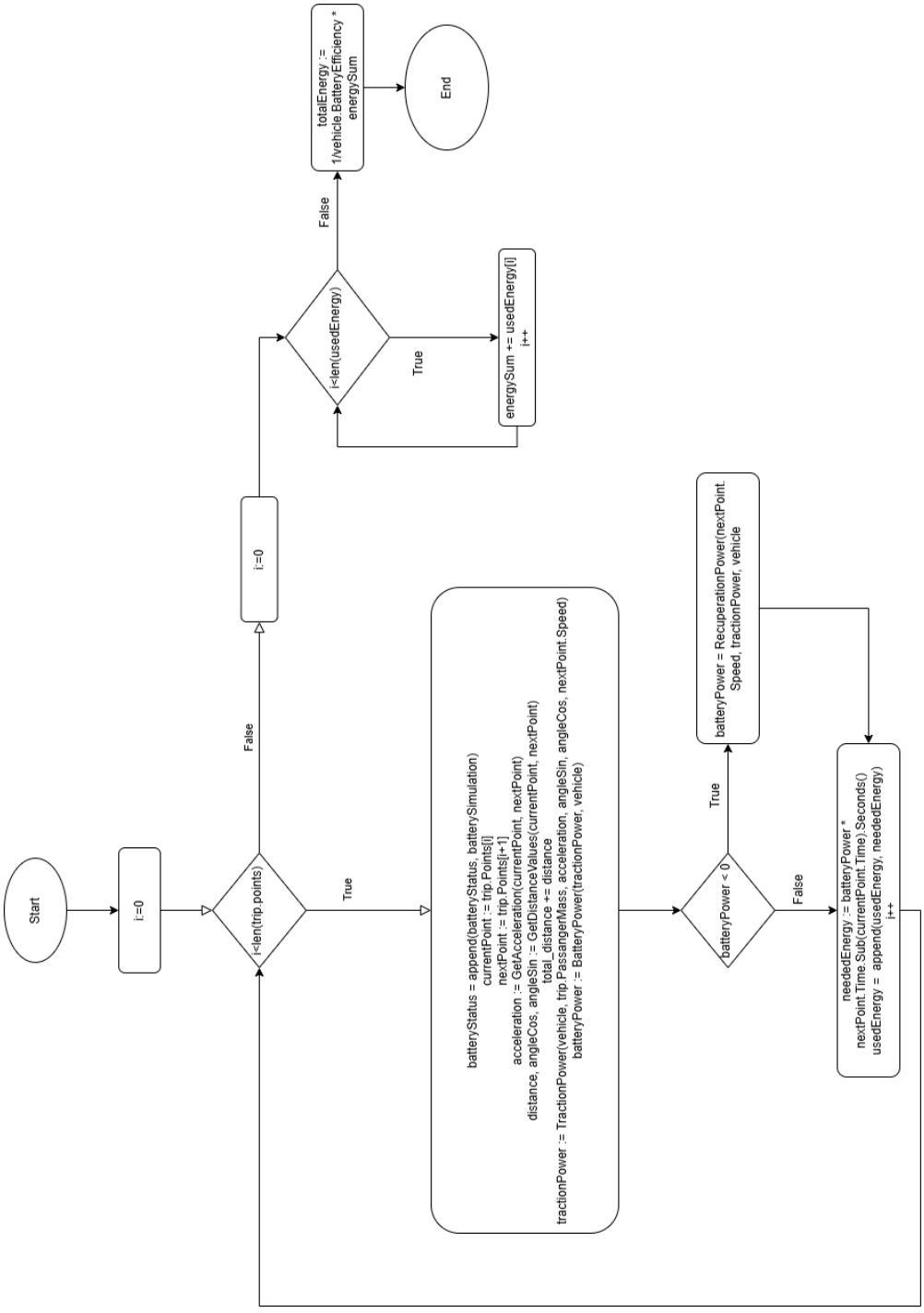
61. AUTOMOBILE-CATALOG.COM. *2021 Renault Zoe R135 Specifications and Performance Data*. 2021. Dostupné také z: https://www.automobile-catalog.com/car/2021/2984435/renault_zoe_r135.html. Accessed: 2025-03-09.

62. EV DATABASE. *Hyundai Kona Electric 65 kWh*. 2025. Dostupné také z: https://ev-database.org/car/1830/Hyundai-Kona-Electric-65-kWh. Accessed: 2025-03-09.

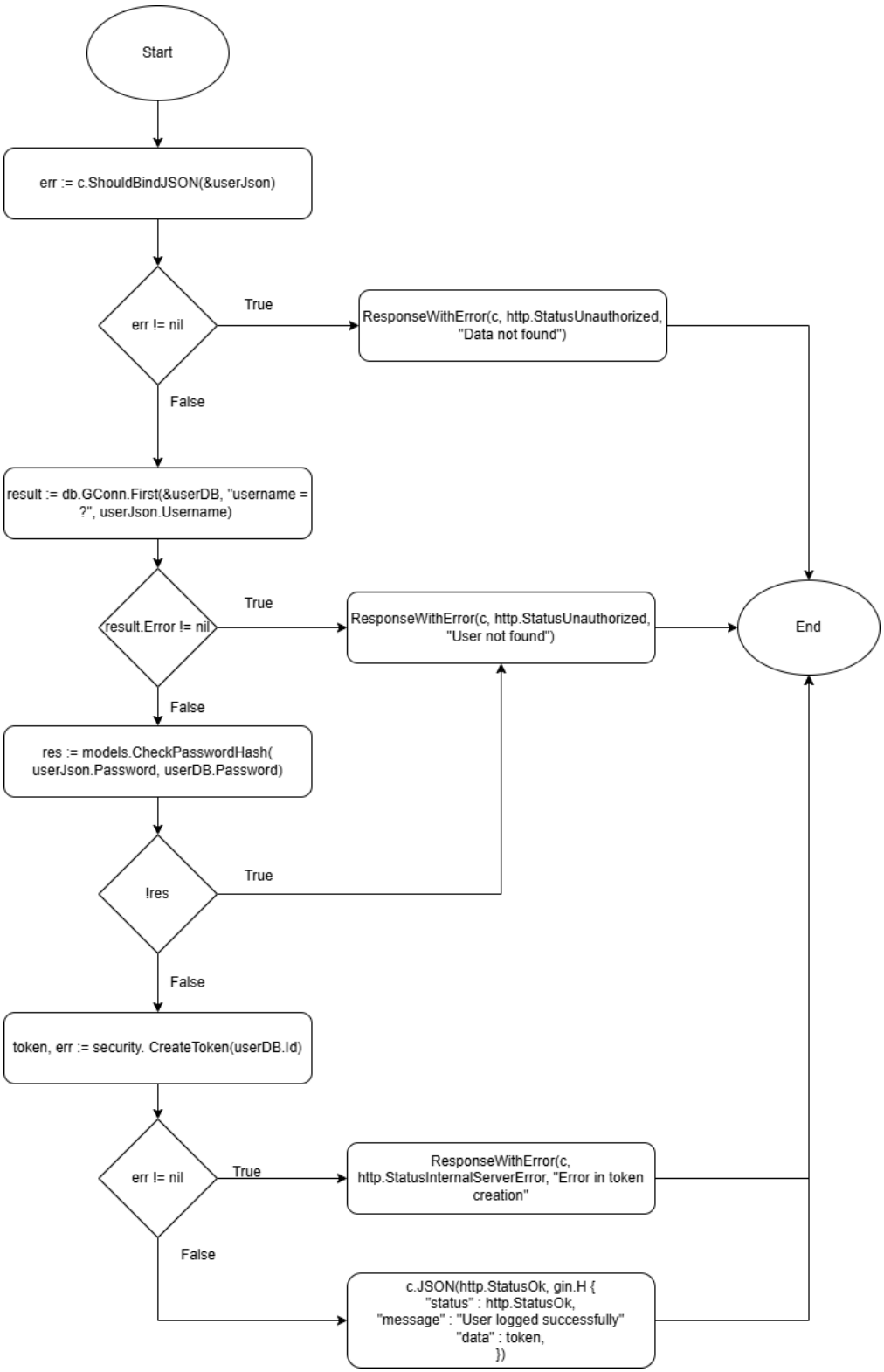
63. H2.LIVE. *Toyota MIRAI II – Brennstoffzellenlimousine mit Stückzahl*. 2025. Dostupné také z: https://h2.live/en/fuelcell-cars/toyota-mirai-ii/. Accessed: 2025-04-19.

Příloha A

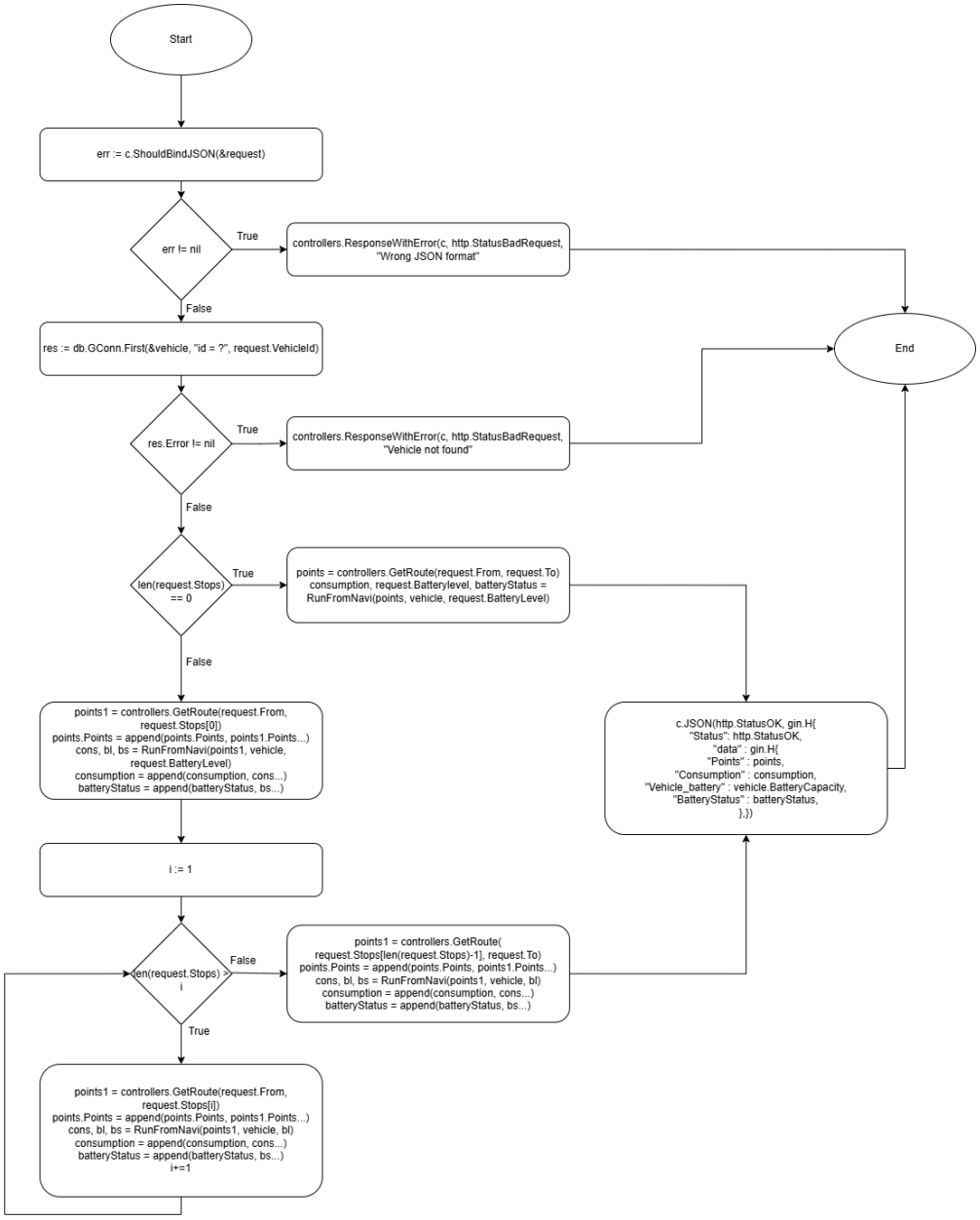
Obrázky



Obrázek A.1: Vývojový diagram popisující jednotlivé kroky modelu



Obrázek A.2: Vývojový diagram pokusu o přihlášení



Obrázek A.3: Vývojový diagram popisující fungování simulačního endpointu

Obrázek A.4: Výsledek spotřeby dle palubního počítače vozidla Hyundai Kona na trase VŠB-TUO budova Hard → Opava-Suché Lazce

Obrázek A.5: Výsledná spotřeba dle palubního počítače vozidla Renault ZOE R135 na Trase VŠB-TUO budova HARD → Kravaře-Dvořisko

Příloha B

Zdrojové kódy

```
<script setup>
import { ref, watch } from "vue";

const props = defineProps({
  show: Boolean,
});

const emit = defineEmits(["update:show", "result"]);

const localShow = ref(props.show);

watch(
    () => props.show,
    (newVal) => {
      localShow.value = newVal;
    }
);

const confirmDelete = () => {
  emit("result", true);
  emit("update:show", false);
};

const cancelDelete = () => {
  emit("result", false);
  emit("update:show", false);
};
</script>
```

Výpis B.1: Příklad struktury script setup

```
const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'HomePage',
      component: HomeView
    },
    {
      path: '/about',
      name: 'about',
      component: () => import('../views/AboutView.vue')
    },
    {
      path: '/vehicles',
      name: 'vehicles',
      component: () => import('../views/Vehicles.vue')
    },
    {
      path: '/map',
      name: 'map',
      component: () => import('../views/MapVue.vue')
    }
  ]
})
```

Výpis B.2: Definice routeru z balíčku Vue-Router

```
<template>
  <v-card class="graph-card">
    <v-card-title>Route Analysis</v-card-title>
    <v-card-text>
      <div class="chart-container">
        <LineChart :chart-data="filteredChartData" :chart-options="chartOptions"
            />
      </div>
    </v-card-text>
  </v-card>
</template>

<script setup>
import { ref, computed, defineProps } from "vue";
import { LineChart } from "vue-chart-3";
import { Chart, registerables } from "chart.js";

Chart.register(...registerables);

const props = defineProps({ labels: Array, datasets: Array });

const datasets = ref(props.datasets.map(dataset => ({ ...dataset, visible: true })
    ));

const filteredChartData = ref({
  labels: props.labels || [],
  datasets: datasets.value.map(({ visible, ...data }) => data),
});


const chartOptions = computed(() => ({
  responsive: true,
  maintainAspectRatio: false,
  animation: { duration: 500, easing: "easeOutQuart" }, // Smooth fade out
  plugins: {
    legend: { display: datasets.value.some(d => d.visible) },
  },
```

```
  scales: { x: { title: { display: true, text: "Waypoints" } } },
}));
</script>

<style scoped>
.graph-card {
  width: 100%;
  margin-top: 20px;
}

.chart-container {
  position: relative;
}
</style>
```

Výpis B.3: Zdrojový kód komponenty Graph displayer

```
{
   "info": {
      "copyrights": [
         "GraphHopper",
         "OpenStreetMap contributors"
      ],
      "took": 6,
      "road_data_timestamp": "2024-12-24T15:00:00Z"
   },
   "paths": [
      {
         "distance": 39214.119,
         "weight": 2597.191265,
         "time": 2008984,
         "transfers": 0,
         "points_encoded": false,
         "bbox": [
            18.156722,
            49.593039,
            18.359615,
            49.833166
         ],
         "points": {
            "type": "LineString",
            "coordinates": [
               [
                  18.159824,
                  49.832795,
                  266.37
               ],
               ...
            ]
         },
         "instructions": [
            {
               "distance": 111.887,
               "heading": 291.06,
```

```
            "sign": 0,
            "interval": [
               0,
               1
            ],
            "text": "Pokračujte",
            "time": 40280,
            "street_name": ""
         },
         ...
      ],
      "legs": [],
      "details": {
         "average_speed": [
            [
               0,
               5,
               10.0
            ],
            ...
         ]
      }
   }
]
}
```

Výpis B.4: Příklad JSONu vráceném navigačním API GraphHopper

Příloha C

Tabulky

Tabulka C.1: Výsledky spotřeby v kWh pro první verzi modelu

Experiment #	Spotřeba EPA	Spotřeba (21 kWh baterie)	Spotřeba (30 kWh baterie)	Vypočtená spotřeba
1	0,0736	0,0659	0,0940	0,0731
2	0,0736	0,0692	0,0990	0,0862
3	0,0736	0,0735	0,1050	0,0836
4	0,0736	0,0748	0,1075	0,0906