VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

CHYTRÁ DOMÁCNOST: UČÍCÍ SE ŘÍZENÍ VYTÁPĚNÍ

SMART HOME: LEARNING CONTROL OF HEATING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **TOMÁŠ MILOSTNÝ**
AUTHOR

VEDOUCÍ PRÁCE — **ZDENĚK MATERNA, Ing., Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Milostný Tomáš**
Program: Informační technologie
Název: **Chytrá domácnost: učící se řízení vytápění**
Smart Home: Learning Control of Heating
Kategorie: Vestavěné systémy

Zadání:

1. Proveďte rešerši existujících řešení pro řízení zdroje tepla a jednotlivých termostatických ventilů.
2. Navrhněte hardwarovou i softwarovou část vlastního řešení umožňující zlepšování řízení v čase.
3. Realizujte navržené řešení.
4. Ověřte funkčnost řešení dlouhodobým testem.
5. Vytvořte video prezentující vaši práci, její cíle a výsledky.

Literatura:

- Zhang, Zhiang, and Khee Poh Lam. "Practical implementation and evaluation of deep reinforcement learning control for a radiant heating system." *Proceedings of the 5th Conference on Systems for Built Environments*. 2018.
- Katić, Katarina, et al. "Neural network based predictive control of personalized heating systems." *Energy and Buildings* 174 (2018): 199-213.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1-2.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Materna Zdeněk, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Tato bakalářská práce se zabývá implementací prototypu ekosystému chytré domácnosti zaměřeného na domácí vytápění. Cílem je vytvořit software běžící na zařízení typu Raspberry Pi, který s pomocí předpovědí strojového učení bude schopen ovládat a monitorovat síť elektrických přímotopných konvektorů ovládané spínáním jednoduchého Wi-Fi relé Shelly 1PM. Výsledné řešení by mělo oproti konvenčním chytrým termostatům disponovat také nižší cenou.

Abstract

This bachelor thesis deals with the implementation of a prototype smart home ecosystem focused on home heating. The goal is to create software running on a Raspberry Pi device that, with the help of machine learning predictions, will be able to control and monitor a network of electric direct heating convectors controlled by switching a simple Wi-Fi relay Shelly 1PM. The resulting solution should also have a lower price than conventional smart thermostats.

Klíčová slova

chytrá domácnost, strojové učení, domácí vytápění, přímotopný konvektor

Keywords

smart home, machine learning, home heating, direct heating convector

Citace

MILOSTNÝ, Tomáš. *Chytrá domácnost: učící se řízení vytápění.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Zdeněk Materna, Ing., Ph.D.

Chytrá domácnost: učící se řízení vytápění

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Zdeňka Materny, Ing., Ph.D.. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....

Tomáš Milostný

18. května 2022

Obsah

Kapitola 1

Úvod

S nástupem popularity chytré domácnosti a rozšířené možnosti veškerá připojená zařízení jednoduše ovládat i na dálku s požitím mobilního telefonu byly rozšířeny i způsoby domácího vytápění v podobě chytrých termostatů. V současné době jsou tato řešení ovšem často velmi nákladná a bývají také často omezeny na infrastrukturu ústředního vytápění se zdrojem tepla a rozvody.

Moje domácnost například tuto infrastrukturu vůbec nemá. Místo toho je v každé místnosti umístěn přímotopný konvektor připojený do zásuvky. Výhodami těchto jednotek jsou snadná instalace, možnost samostatně a nezávisle fungovat, i úsporný provoz. Nelze na nich však přímo nastavit požadovanou teplotu v místnosti, tudíž může docházet k přetápění. Navíc je do ovládání chytrým centrálním termostatem není možné zapojit.

Cílem této práce je tedy vyvinout systém chytré domácnosti zaměřený na vytvoření sítě topných jednotek, které bude připojením k Wi-Fi možné vzdáleně monitorovat a ovládat. Pro tento účel bylo zvoleno relé Shelly 1PM, které umožňuje přímotop připojit do sítě Wi-Fi, sledovat jeho spotřebu, a navíc s pomocí teplotního čidla měřit teplotu v okolí. Toto relé tedy zajistí potřebnou funkcionalitu za nízkou cenu na jednotku. Pro centrální řízení této sítě bude použit minipočítač Raspberry Pi 4 Model B. V rámci softwarové části bude také pro pohodlí uživatele vytvořena mobilní aplikace, ve které bude možné nakonfigurovat nastavení jednotlivých topení i jejich další monitorování.

Toto řešení by tedy mělo v existující domácnosti používající „hloupé“ přímotopné konvektory zajistit cenově dostupný systém, který by měl zvýšit uživatelské pohodlí i budoucí energetická úspora.

Kapitola 2

Systémy vytápění

Tato kapitola představuje v jakém testovacím prostředí poběží vytvořený systém chytrého vytápění a shrnuje existující řešení, které byly inspirací během implementace řešení.

2.1 Elektrické vytápění

Dle [23] elektrické vytápění je proces, při kterém se elektrická energie přeměňuje na tepelnou energii. Běžnými aplikacemi jsou prostorové vytápění, ohřev vody a průmyslové procesy. Elektrický ohřívač je elektrické zařízení, které přeměňuje elektrický proud na teplo. Topné těleso uvnitř každého ohřívače je elektrický odpor a funguje na principu *Jouleova ohřevu*: elektrický proud procházející rezistorem přemění tuto elektrickou energii na tepelnou.

Základním rozdělením je vytápění samotného prostoru (infraohřívače, konvekční ohřívače, teplovzdušné ventilátory, akumulační kamna, elektrické podlahové vytápění, systém vytápění osvětlením a tepelná čerpadla) a vytápění kapalinou (ponorné ohřívače, cirkulační potrubní ohřívače a ohřívače elektrod).

Pro tuto práci byl použit nástěnný přímotopný konvektor **AEG WKL 1003 U**. Jedná se tedy o konvekční ohřívač, což je dle [14] typ ohřívače, který využívá k ohřevu a cirkulaci vzduchu konvekční proudy. Tyto proudy cirkulují v celém těle spotřebiče a přes jeho topné těleso. Tento proces na principu vedení tepla ohřívá vzduch, snižuje jeho hustotu oproti chladnějšímu vzduchu a způsobuje jeho stoupání. Jak molekuly zahřátého vzduchu stoupají, vytlačují molekuly chladnějšího vzduchu dolů směrem k topnému zařízení. Vytlačený chladný vzduch se v důsledku toho ohřeje, tím je snížena jeho hustota, stoupá vzhůru směrem ke stropu a cyklus se opakuje.

Konvektor AEG WKL 1003 U pracuje s pevným příkonem 1000 W, který spíná vestavěným termostatem s nastavením teploty výběrem z úrovní 1 až 7, protizámrazovou ochranu na 6 °C a volbou MAX, která by měla vytopit místnost až na 30 °C. Na pravé straně má také fyzický spínač zapnuto/vypnuto.

Obrázek 2.1: Fotografie přímotopného konvektoru AEG WKL 1003 U instalovaného v malém pokoji v jednopodlažním rodinném domě.

2.2 Chytrý termostat Nest Learning

Zajímavým zástupcem chytrých termostatů je Nest Learning Thermostat vyvíjený výrobcem Nest Labs. Dle [26] se jedná o programovatelný a samoučící se Wi-Fi termostat využívaný k optimalizaci vytápění a chlazení domácností i podniků za účelem úspory energie. Zařízení je založeno na algoritmu strojového učení, kdy první týdny musí uživatelé regulovat termostat, aby poskytli referenční soubor dat. Termostat se pak může naučit rozvrh lidí, na jakou teplotu jsou zvyklí a kdy. Interagovat s termostatem lze klikáním a posouvání jeho ovládacího kolečka, anebo mobilní aplikací Google Home. Pomocí vestavěných senzorů a umístění telefonů se může přepnout do režimu úspory energie, když si uvědomí, že nikdo není doma.

Termostaty Nest jsou kompatibilní s většinou standardních systémů HVAC (topení, větrání a klimatizace), které využívají ústřední vytápění a chlazení. Do prostředí s přímotopnými konvektory tedy nasadit nelze.

Obrázek 2.2: Přední obrazovka termostatu Nest

2.3 Chytré elektrické přímotopy

Na světovém trhu ovšem existují i řešení v podobě elektrických topení se zabudovanou Wi-Fi (například Nedis WIFIHTPL20FWT [5], který umožní vzdálené ovládání a programování pomocí aplikace Nedis SmartLife, a také podporu Amazon Alexa a Google Assistant). Toto řešení může být postačující, zásadně se však ale neodlišuje od tradičních přímotopných konvektorů a výměna jednotek v existující instalaci tedy může být zbytečně nákladná.

Obrázek 2.3: Wi-Fi přímotopný konvektor Nedis WIFIHTPL20FWT

2.4 Tuya Wi-Fi termostat

Posledním zajímavým produktem ve světě je Tuya Wi-Fi termostat [12], který by svou formou byl z předchozích zmíněný nejvhodnější volbou do prostředí s existujícími přímotopnými konvektory, které bychom chtěli připojit k Wi-Fi a vzdáleně je ovládat a monitorovat. Svým hardwarovým provedením je také podobný řešení představeném v podkapitole 3.1.

Obrázek 2.4: Tuya Wi-Fi termostat

Kapitola 3

Návrh řešení

Tato kapitola se zabývá návrhem softwarové i hardwarové části a nastíní technologie použité pro vytvoření navrženého řešení.

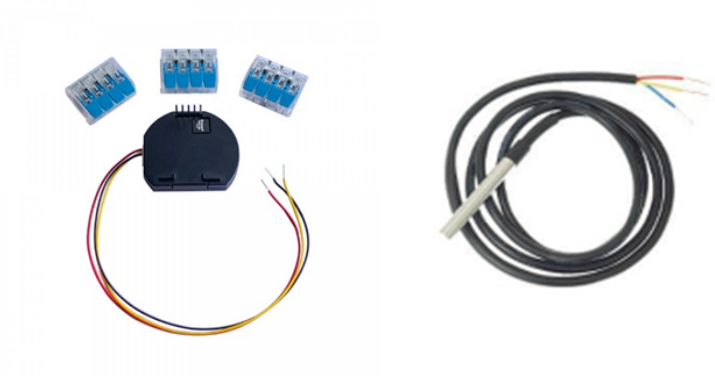
3.1 Shelly 1PM

Shelly 1PM (série 1 + *power measurement*) je Wi-Fi relé, které bylo zvoleno pro svou nízkou cenu a širokou funkcionalitu. Umožňuje spínání zásuvky, do které je zapojen přímotop, monitorování její spotřeby ve Wattech a s pomocí přídavného modulu i měřit teplotu.

Po zapojení hardware je ve vestavěném webovém rozhraní možné toto relé připojit k domácí Wi-Fi síti a mimo jiné mu například přiřazení statické IP adresy, nastavení zdrojového napětí 110 nebo 220 V, nastavení časové zóny nebo aktualizace firmware [6].

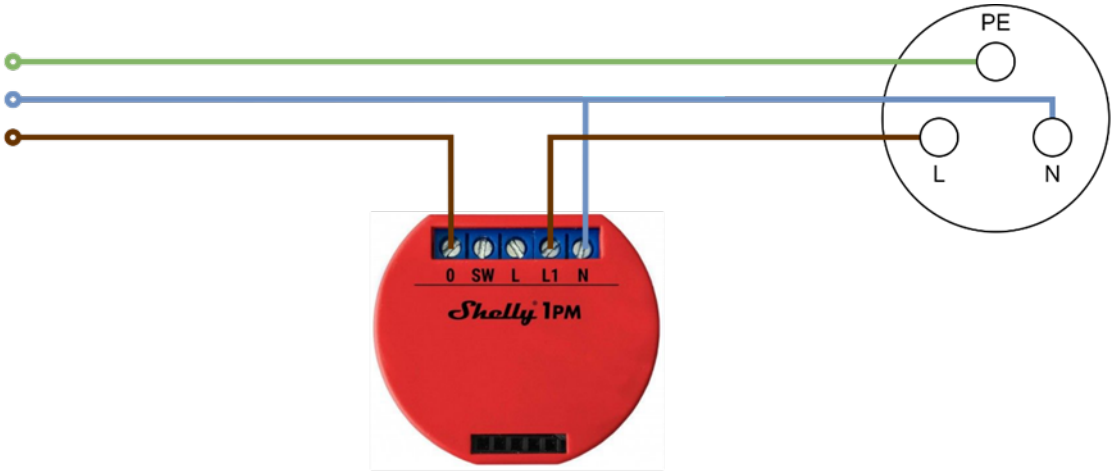
Pro komunikaci se Shelly 1PM nabízí výrobce kompatibilitu s populárními platformami Android, iOS, Amazon Alexa, Google Assistant a domácí automatizační servery s použitím protokolů MQTT, CoAP a REST API [6]. V této práci bude později využita komunikace přes rozhraní REST.

Přídavný modul teplotního senzoru [11] doplňuje základní funkcionalitu relé Shelly 1 nebo 1PM o získání teploty naměřené v místnosti (dle nastavení ve °C nebo °F). K tomuto modulu je tedy ještě potřeba připojit samotný senzor DS18B20 [7], které lze připojit dokonce až 3, nebo DHT22. Pro tento projekt byl zvolen DS18B20 přímo od výrobce Shelly a bude se pracovat s teplotou ve °C.

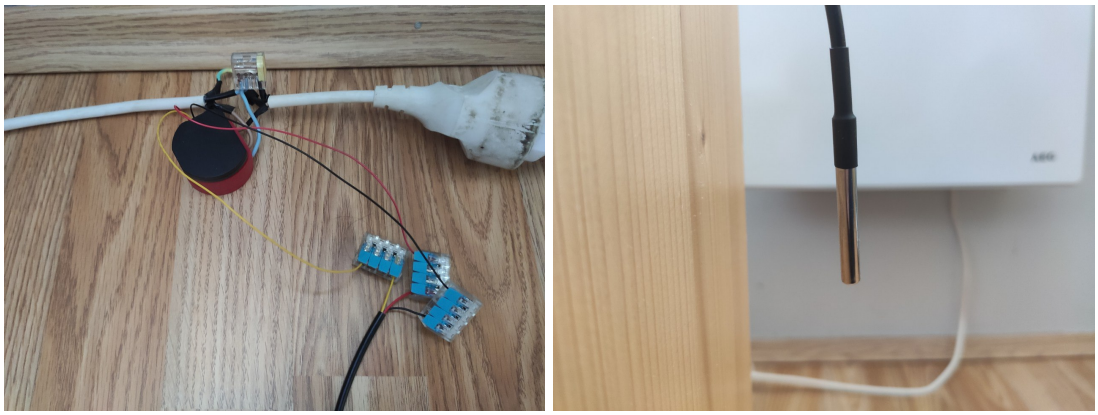


Obrázek 3.1: Přídavný modul pro Shelly 1/1PM a teplotní senzor DS18B20

V této práci bude použito zapojení zobrazené v následujícím diagramu. Relé Shelly 1PM bude vloženo do prodlužovacího kabelu, skrze který bude přímotopný konvektor připojen k zásuvce. Tento postup byl zvolen pro zjednodušení prvotní testovací instalace. Zařízení je jinak možné instalovat více způsoby, například přímo do zásuvky zabudované ve stěně budovy.



Obrázek 3.2: Diagram použitého zapojení relé Shelly 1PM

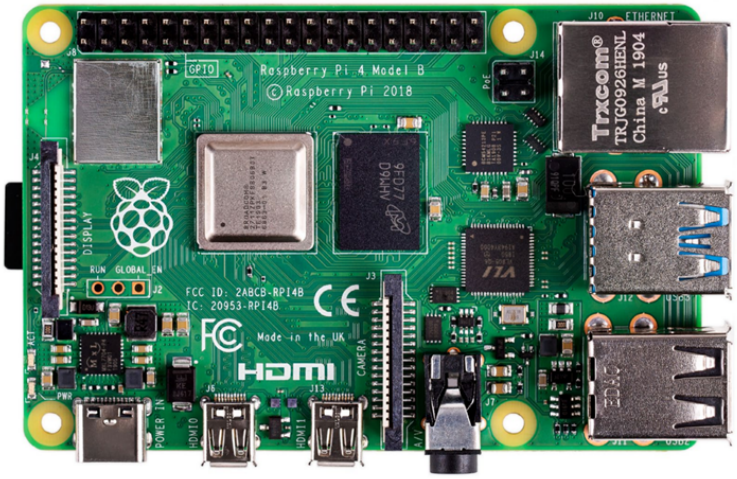


Obrázek 3.3: Fotografie instalovaného Shelly 1PM s modulem teplotního senzoru a DS18B20 umístěný v blízkosti ovládaného přímotopu

3.2 Raspberry Pi 4 Model B

Jako centrální prvek, který bude komunikovat s jednotlivými topeními připojených pomocí relé Shelly, bylo zvoleno Raspberry Pi 4 Model B ve variantě s RAM 2 GB LPDDR4 (další možné varianty jsou s 1, 4 nebo 8 GB). Dle [18] se jedná o nejnovější a nejvýkonnější model z oblíbené řady jednodeskových počítačů Raspberry Pi. Mezi jeho klíčové vlastnosti patří výkonný 64bitový čtyřjádrový procesor, podpora dvou displejů v rozlišení až 4K prostřednictvím dvojice micro-HDMI portů, hardwarové dekódování videa až 4K 60 FPS, dvoupásmová bezdrátová LAN 2,4/5,0 GHz, Bluetooth 5.0, Gigabit Ethernet, USB 2.0/3.0 a napájení 5V DC přes USB-C, GPIO header, nebo PoE (*Power over Ethernet*).

Zajímavý pro použití v této práci je právě svým procesorem Broadcom BCM2711. Jedná se o 64bitové čtyřjádrové SoC (*System on Chip*) Cortex-A72 běžící na taktovací frekvenci 1,5 GHz [18]. Ten by tedy pro správu více připojených topení a zpracování předpovědí strojovým učením měl zajistit optimální výkon. Architektura ARM v8 je také podporována virtuálním běhovým prostředím CLR (*Common Language Runtime*), pod kterým bude běžet software napsaný v jazyce C#.



Obrázek 3.4: Raspberry Pi 4 Model B

3.3 C# a platforma .NET

Dle [28] C# je moderní objektově orientovaný a staticky typovaný programovací jazyk vytvořený společností Microsoft. Umožňuje vytvářet mnoho typů bezpečných a robustních aplikací, které běží na platformě .NET. Má své kořeny v rodině jazyků typu C a jeho syntaxe je založena na jazycích C++, Java a JavaScript. Objektový model jazyka C# je komponentně orientovaný, což podporuje programování nezávislých a vyměnitelných modulů.

Tento jazyk také obsahuje řadu funkcí, které napomáhají vytváření robustních a odolných aplikací:

- *Garbage Collection* automaticky uvolňuje alokovanou paměť obsazenou nedostupnými objekty
- *Nullable types* (typy s možností nabytí hodnoty null) zavádějí pokročilejší ochranu před proměnnými, které se neodkazují na alokované objekty.
- *Zpracování výjimek* poskytuje strukturovaný a rozšiřitelný přístup k detekci chybových stavů a obnově chodu programu.
- *Lambda výrazy* podporují techniky funkcionálního programování.
- Syntaxe *LINQ (Language Integrated Query)* představuje společný vzor využívající lambda výrazy pro práci s daty z jakéhokoliv zdroje.

- Jazyková podpora pro asynchronní operace poskytuje syntaxi pro vytváření distribuovaných systémů.
- Má jednotný typový systém, kde všechny typy, včetně primitivních typů jako jsou int a double, dědí z jediného kořenového typu object. Všechny typy tedy sdílí sadu běžných operací (například metoda ToString). Hodnoty jakéhokoliv typu mohou být uloženy, přemisťovány a manipulovány konzistentním způsobem. Podporuje uživatelsky definované referenční (třídy a záznamy) i hodnotové typy (struktury). Třídy kolekcí navíc obsahují iterátory, které umožňují definovat vlastní chování v klientském kódu (například v cyklech foreach).

Programy v jazyce C# ve virtuálním systému zvaném *Common Language Runtime* (CLR). CLR je implementace *common language infrastructure* (CLI), mezinárodního standardu od společnosti Microsoft. Zdrojový kód napsaný v C# je kompilován do *intermediate language* (IL), který odpovídá specifikaci CLI. Kód IL a prostředky, jako jsou řetězce, bitmapy a další, jsou uloženy v souborech sestavení obvykle s příponou .dll, který také obsahuje manifest s dalšími informacemi, jako jsou typy, verze programu nebo kultura (pro vícejazyčné aplikace). Když je program v C# spuštěn, sestavení se načte do CLR, které provádí Just-In-Time kompilaci pro převod kódu z IL do nativních strojových instrukcí.

Jazyková interoperabilita je klíčovou vlastností .NET. IL kód vytvořený kompilátorem C# odpovídá *Common Type Specification* (CTS). Může tedy interagovat s kódem vygenerovaným z .NET verzí jazyků F#, Visual Basic, C++ a více než 20 dalších programovacích jazyků kompatibilních s CTS. Díky tomu může jedno sestavení obsahovat více modulů napsaných v různých jazycích pod platformou .NET. Typy se mohou navzájem odkazovat, jako by byly napsány ve stejném jazyce.

Kromě běhových služeb zahrnuje .NET také rozsáhlé knihovny. Tyto knihovny podporují mnoho různých pracovních zátěží, které jsou uspořádány do jmenných prostorů poskytující širokou škálu užitečných funkcí. Knihovny zahrnují vše od vstupu a výstupu souborů přes manipulaci s řetězci až po analýzu XML, rozhraní webových aplikací až po ovládací prvky knihoven jako je Windows Forms. Typická aplikace v jazyce C# široce využívá knihovnu tříd .NET ke zpracování běžných operací (například práce se soubory).

3.4 ASP.NET Core Web API

Serverovou částí vypracovaného řešení bude software běžící pod frameworkem *ASP.NET Core*. Tento framework prezentuje dle [20] řadu funkcí, jako je správa závislostí (*dependency injection*), konfigurace, *middleware* a další. ASP.NET Core podporuje vytváření webových aplikačních rozhraní pomocí *controllerů* [21] dle návrhového vzoru MVC (*Model-View-Controller*) nebo pomocí minimálních API. V této práci je použit přístup *minimálního API*, jelikož se primárně nejedná o robustní webovou aplikaci.

Aplikace vytvořené pomocí dostupných webových šablon obsahují spouštěcí kód aplikace v souboru *Program.cs*, kde jsou konfigurovány aplikací požadované služby a kanál (*pipeline*) zpracování požadavků aplikace je definován jako řada komponent middlewaru.

Vkládání závislostí (DI) zpřístupňuje nakonfigurované služby v celé aplikaci. Služby jsou přidány do kontejneru DI pomocí *WebApplicationBuilder.Services* jako *Singleton* (jediná instance dostupná v rámci celé aplikace), *Transient* (pro každou injekci je vytvořena nová instance třídy) a další. Služby se obvykle získávají z DI pomocí konstruktorového vkládání (*constuctor injection*), kde jsou služby specifikovány v parametrech konstruktoru.

Zpracování požadavků je složeno jako řada middlewarových komponent. Každá komponenta provádí své operace v rámci *HttpContext* a buď skončí úspěšně a vyvolá další middleware v kanálu, nebo ukončí požadavek. Typicky používanými middleware jsou například autorizace, vývojářská stránka s informacemi o chybách, anebo přesměrování na HTTPS. Pomocí plánovacího middleware je v této práci zajištěno periodické volání služeb frameworku strojového učení a databázového klienta pro ukládání záznamů o měřeních teploty v místnosti a venkovního počasí.

ASP.NET Core poskytuje implementaci serveru Kestrel, který může běžet jako veřejně přístupný server vystavený přímo internetu. Kestrel je však často provozován v konfiguraci reverzní proxy se servery Nginx nebo Apache.

3.5 ML.NET

ML.NET je dle [13] open-source multiplatformní framework strojového učení pro vytváření aplikací na platformě .NET v jazycích C# nebo F#. Obsahuje funkce jako automatizované strojové učení (*AutoML*) a nástroje jako *ML.NET CLI* a *ML.NET Model Builder*.

Aplikace strojového učení využívají k předpovědi vzorů v datech, místo toho aby je bylo nutné explicitně programovat. Ústředním bodem ML.NET je model strojového učení. Model specifikuje kroky potřebné k transformaci vstupních dat do predikce. ML.NET umožňuje trénovat vlastní model nebo importovat předem trénovaný model TensorFlow či ONNX.

ML.NET dle [19] podporuje řadu úloh strojového učení: *klasifikace/kategorizace* (například rozdělení zpětné vazby na pozitivní/negativní), *regrese* (předpověď spojité hodnoty, například ceny domu na základě velikosti a lokality), *detekce anomálií* (například detekce podvodných bankovních transakcí), *doporučení* (například produktu na základě předchozích nákupů zákazníka), *časové řady* (sekvenční data, například předpověď počasí) a *klasifikace obrázků* (například kategorizace druhů rostlin).

V této práci se bude pracovat s po sobě jdoucími měřeními vývoje teploty v místnosti, které se určitým způsobem vyvíjí v čase. Jedná se tedy o časovou řadu. Analýza časových řad podporovaná frameworkem ML.NET pomáhá poskytnout odpověď na otázku budoucího vývoje teploty tím, že se podívá na historická data, identifikuje vzorce a použije tyto informace k předpovědi. Algoritmus rozhodnutí o stavu topení pracující s touto předpovědí bude popsán v kapitole 4.

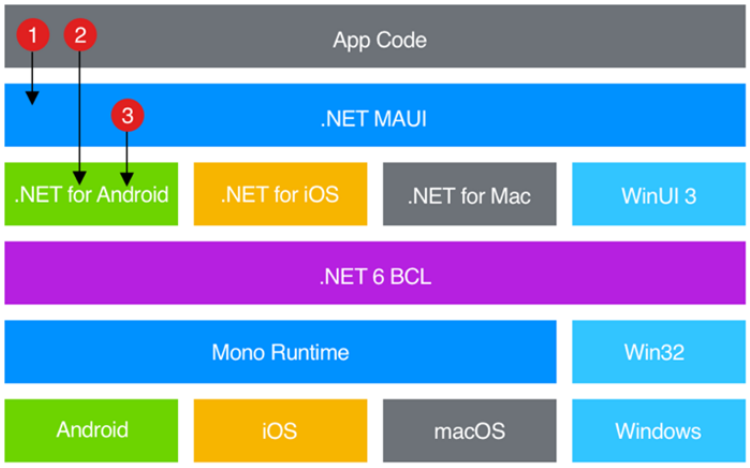
Dle [29] ML.NET pro analýzu dat používá techniku jednorozměrné analýzy časových řad. Jednorozměrná analýza časových řad se dívá na jediné numerické pozorování za určité časové období ve specifických intervalech. Framework k tomuto účelu používá algoritmus *analýzy singulárního spektra (SSA)*. SSA funguje tak, že rozkládá časovou řadu na sadu hlavních komponent. Tyto složky lze interpretovat jako části signálu, které odpovídají trendům, šumu, sezónnosti a mnoha dalším faktorům. Následně jsou tyto komponenty rekonstruovány a použity k předpovědi hodnot někdy v budoucnu.

3.6 .NET MAUI

Pro vytvoření klientské aplikace se správou a monitorováním připojených topení bude použita knihovna .NET MAUI (Multi-platform App User Interface). Dle [30] se jedná o multiplatformní knihovnu od společnosti Microsoft, která je nástupcem aktuálního frameworku Xamarin.Forms. Umožňuje vytváření nativních mobilních a desktopových aplikací s použitím jazyků C# a XAML podporovaný operačními systémy Android, iOS, macOS a Windows. Hlavní výhodou je tedy využití jednoho sdíleného zdrojového kódu v jednom projektu pro všechny platformy. Navíc v případě potřeby umožňuje přidání platformě specifického kódu ve specifických podadresářích, nebo pomocí direktiv překladače.

.NET 6 poskytuje řadu platformě specifických frameworků pro vytváření aplikací: .NET pro Android, .NET pro iOS, .NET pro macOS a knihovnu Windows UI 3 (WinUI 3). Všechny tyto frameworky mají přístup ke stejné knihovně základních tříd .NET 6 (BCL). Tato knihovna abstrahuje podrobnosti o základní platformě. Verze .NET runtime pro Android, iOS a macOS využívají implementaci *Mono*, zatímco verze pro Windows používá *Win32*.

Obrázek 3.5 zobrazuje architekturu aplikace používající .NET MAUI. Klientský aplikační kód typicky interaguje s .NET MAUI API (1), které poté přímo volá funkce API nativní platformy (3). Pokud je vyžadováno, funkce nativní platformy může klientský kód volat i přímo (2) [30].



Obrázek 3.5: Diagram architektury .NET MAUI

3.6.1 XAML

Dle [31] je XAML (*eXtensible Application Markup Language*) jazyk založený na XML. Je to alternativa k programování kódem, kdy jsou vytvářeny instance objektů organizované v hierarchiích. Umožňuje definovat uživatelská rozhraní aplikací pomocí značek. XAML není vyžadován v aplikaci .NET MAUI, ale je to doporučený přístup k vývoji uživatelského rozhraní, protože je často stručnější, vizuálně koherentnější a má podporu nástrojů v IDE. XAML se také dobře hodí pro použití se vzorem *Model-View-ViewModel (MVVM)*, kde XAML definuje pohled, který je propojen s kódem viewmodelu v C# prostřednictvím datových vazeb.

3.6.2 Model-View-ViewModel

Návrhový vzor Model-View-ViewModel (MVVM) dle [22] vynucuje oddělení mezi třemi softwarovými vrstvami – uživatelským rozhraním, nazývaným pohled (view), podkladovými daty, nazývanými model, a prostředníkem mezi pohledem a modelem, nazývaným viewmodel. Pohled a viewmodel jsou často propojeny prostřednictvím datových vazeb definovaných v XAML, kde je vlastnost *BindingContext* obvykle instancí daného viewmodelu.

3.7 InfluxDB

Pro ukládání dat o připojených topeních byla zvolen databázový software *InfluxDB* vytvořený společností InfluxData. Dle [16] je to open-source platforma pracující s časovými řadami. To zahrnuje rozhraní API pro ukládání a dotazování nad daty, jejich zpracování pro účely monitorování a upozornění, uživatelské panely a vizualizace dat v přehledném rozhraní pod `http://localhost:8086/` (případně po nasazení na Raspberry Pi dostupné pod jeho IP adresou místo rozhraní localhost).

Pro zápis a dotazování nad daty nebo jakékoliv používání rozhraní API, je nejprve nutné vytvořit uživatelské přihlašovací údaje, organizaci a segment. Vše v InfluxDB je organizováno pod konceptem organizace. API je navrženo jako multi-tenant (sdílené skupinou uživatelů). Segmenty představují místo, kde jsou ukládány data časové řady.

Dotazy na tuto databázi využívají jazyka *Flux*. Dle [2] je to open-source funkcionální datový skriptovací jazyk určený pro dotazování, analýzu a práci s daty. Flux podporuje více typů zdrojů dat, včetně databází časových řad (jako je InfluxDB), relační SQL databáze (jako MySQL a PostgreSQL) a CSV. Jazyk Flux sjednocuje kód pro dotazování, zpracování a zápis dat do jediné syntaxe.

```
from(bucket: "example-bucket")
    |> range(start: -1d)
    |> filter(fn: (r) => r._measurement == "example-measurement")
    |> mean()
    |> yield(name: "_results")
```

Obrázek 3.6: Příklad dotazu pro čtení z databáze InfluxDB v jazyce Flux [3]

3.8 Nginx

Dle [17] Nginx je open-source softwarový webový server s managementem zátěže a reverzní proxy. Pracuje s protokoly HTTP (i HTTPS), SMTP, POP3, IMAP a SSL. Zaměřením tohoto serveru je především vysoký výkon a nízké nároky na paměť.

V projektu je server Nginx použit dle [25] jako reverzní proxy server, který naslouchá na standardním HTTP portu 80 a přeposílá dotazy aplikaci ASP.NET Core běžící na rozhraní localhost na portu 5232.

Kapitola 4

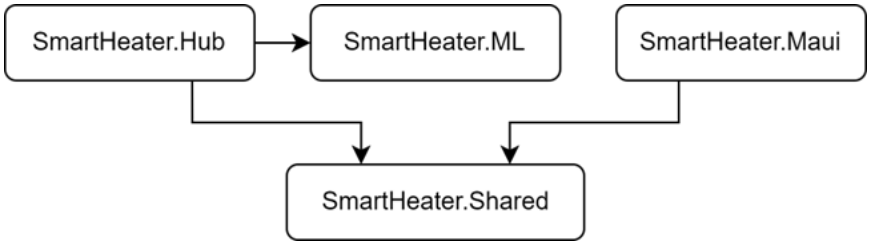
Implementace a testování

Tato kapitola se zabývá detaily implementace systému *SmartHeater* vytvořeného v rámci této práce, který je nasazen jako řídící software na dříve zmíněném minipočítači Raspberry Pi. Uživateli také umožní správu a monitorování sítě chytrých topení v jednoduché vlastní mobilní aplikaci. Závěr této kapitoly se zabývá testováním výsledků získaných během sběru dat implementovaným systémem.

4.1 Architektura řešení

Kód řešení (*SmartHeater*) je rozdělen na klientskou, serverovou a sdílenou část. Tento systém je rozdělen do čtyř projektů:

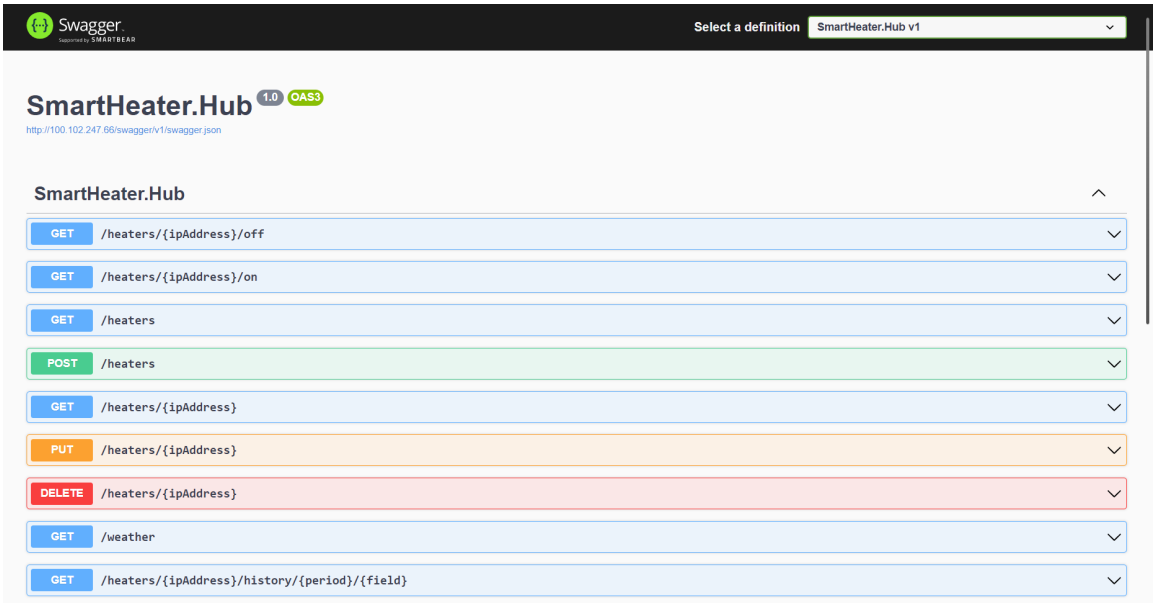
- **SmartHeater.Hub**: řídící hub zajišťující serverovou část pod ASP.NET Core Web API a periodické řízení a monitorování registrovaných ovladačů topení.
- **SmartHeater.ML**: knihovna tříd nad frameworkem ML.NET pro strojové učení.
- **SmartHeater.Maui**: klientská multiplatformní aplikace nad .NET MAUI.
- **SmartHeater.Shared**: knihovna tříd obsahující sdílené prostředky používané řídícím hubem i mobilní aplikací (například výčtové typy a třídy použité v HTTP komunikaci pro serializaci a deserializaci ve formátu JSON).



Obrázek 4.1: Graf závislostí mezi projekty řešení

4.2 Řídící hub

Centrálním bodem této práce je software běžící na Raspberry Pi, které zde má roli hlavního řídícího hubu. Cílem této části je správa, ovládání a monitorování připojených topných jednotek a poskytnutí rozhraní webového API pro komunikaci s klientskou aplikací. Jádrem je projekt *SmartHeater.Hub*. V souboru *Program.cs* má nakonfigurováno zajištění vytrénování modelu ML, registraci služeb dále poskytnuté DI kontejnerem, plánovač poskytnutý knihovnou *Coravel* [10], který periodicky spuští kód tříd ze složky *Invocables*, ladící prostředí *Swagger* a HTTP koncové body poskytovaného webového API.



Obrázek 4.2: Prostředí Swagger pro ladění koncových bodů webového API

Předpisy rozhraní služeb a jedna z jejich možných implementací se v projektu nacházejí ve jmenném prostoru *SmartHeater.Hub.Services*. Využití rozhraní bylo zvoleno z důvodu možnosti budoucí rozšiřitelnosti a nahrazení aktuální implementace bez nutnosti přepisování kódu, ve kterém jsou tato rozhraní dále využívána.

```
17    var builder = WebApplication.CreateBuilder(args);
18
19    builder.Services.AddEndpointsApiExplorer();
20    builder.Services.AddSwaggerGen();
21
22    builder.Services.AddTransient<StatsCollectorInvocable>();
23    builder.Services.AddTransient<MLInvocable>();
24    builder.Services.AddScheduler();
25
26    builder.Services.AddSingleton<IHeatersRepository, HeatersRepository>();
27    builder.Services.AddSingleton<IDatabaseService, InfluxDbService>();
28    builder.Services.AddSingleton<IWeatherService, OpenWeatherService>();
29    builder.Services.AddSingleton<ICoordinatesService, IpApiService>();
30
31    builder.Services.AddSingleton(sp => new HttpClient
32    {
33        Timeout = TimeSpan.FromSeconds(3)
34    });
```

Obrázek 4.3: Registrace implementací rozhraní služeb v souboru *Program.cs*, které bude DI kontejner poskytovat v rámci aplikace

4.2.1 Data o počasí

Pro sledování vlivu počasí na teplotu uvnitř v místnosti bylo do projektu přidáno rozhraní *IWeatherService*, které zprostředkovává metody pro čtení teploty ve °C i °F (v této práci se primárně pracuje se °C). Pro pohodlí uživatele bylo také přidáno rozhraní *ICoordinatesService*, které umožňuje automatické zjištění polohu zařízení, která bude použita k získání dat o počasí z daného místa.

Implementace ve třídě *IpApiService* k tomuto využívá webovou službu `https://ipapi.co/`, která vrací souřadnice dle veřejné IP adresy zařízení. Tyto souřadnice jsou uloženy v poli, aby se zamezilo zbytečnému síťovému provozu. Pracuje se zde s předpokladem, že zařízení, na kterém software poběží, typu Raspberry Pi bude instalováno na jednom místě a nebude měnit svou polohu.

Samotné rozhraní pro získávání dat o počasí implementuje třída *OpenWeatherService*. Jako zdroj používá volně dostupné webové API **OpenWeather**, které pro svůj provoz vyžaduje pouze registraci a vytvoření API klíče. Získaný klíč je následně uložen do souboru *appsettings.json*. Tento soubor obsahuje konfiguraci pro ASP.NET projekt, ze které lze snadno číst pomocí vestavěného rozhraní *IConfiguration*.

```
4 references
18  public async Task<double?> ReadCelsiusAsync()
19  {
20      (var lat, var lon) = await _coordsService.GetLatitudeLongitudeAsync();
21      if (lat is null || lon is null)
22      {
23          return null;
24      }
25      var requestUri = $"https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={_apiKey}&units=metric";
26      try
27      {
28          var weatherModel = await _httpClient.GetFromJsonAsync<OpenWeatherModel>(requestUri);
29          return Convert.ToDouble(weatherModel?.Data?["temp"].ToString(), CultureInfo.InvariantCulture);
30      }
31      catch
32      {
33          Console.Error.WriteLine($"{DateTime.Now}: Error while getting weather.");
34          return null;
35      }
36  }
37
    1 reference
38  private class OpenWeatherModel
39  {
40      [JsonPropertyName("main")]
        1 reference
41      public Dictionary<string, object>? Data { get; set; }
42  }
43  }
44
```

Obrázek 4.4: Metoda *ReadCelsiusAsync* pro získání souřadnic aktuální polohy zařízení volá službu *ICoordinatesService*, které použije v HTTP požadavku na službu *OpenWeather*. Získaná data ve formátu JSON jsou deserializována do privátní třídy (zde je důležitá položka *"main"*, jejíž položky jsou mapovány na slovník). Ze slovníku *Data* je z položky *"temp"* vyčtena teplota ve °C a vrací se jako hodnota typu double. V případě chyby je vrácena hodnota null.

4.2.2 Databázové rozhraní

Rozhraní *IDatabaseService* zprostředkovává připojení k databázi. Obsahuje metody umožňující zápis měření (jako parametry vyžaduje záznam stavu topení a počasí) a čtení historie (vrací kolekci databázových záznamů pro konkrétní model topení ve vybraném období a z vybraného sloupce tabulky v databázi).

Implementace ve třídě *InfluxDbService* využívá knihovnu *InfluxDB.Client* poskytující třídy pro práci s databázovým systémem InfluxDB. Podobně jako v předchozí podkapitole používá *InfluxDbService* konfiguraci uloženou v souboru *appsettings.json*, kde má uložený název organizace a segmentu, které je třeba nastavit v databázi InfluxDB po její instalaci, a databázovým systémem vygenerovaný přístupový token.

```
2 references
public string WriteMeasurement(HeaterStatusModel heater, double? weather)
{
    var point = PointData
        .Measurement(DbFields.MeasurementName)
        .Tag(DbFields.HeaterTag, heater.IPAddress)
        .Field(DbFields.Temperature, heater.Temperature ?? double.NaN)
        .Field(DbFields.Weather, weather ?? double.NaN)
        .Field(DbFields.Power, heater.Power ?? double.NaN)
        .Timestamp(heater.MeasurementTime, WritePrecision.Ns);

    using var client = CreateDbClient();
    using var writeApi = client.GetWriteApi();
    writeApi.WritePoint(point, _bucket, _organization);

    return point.ToLineProtocol();
}
```

Obrázek 4.5: Metoda *WriteMeasurement* zapisuje záznam o měření stavu topení a počasí vytvořením objektu s daty s položkami daného měření a časovou značkou. Klient InfluxDB obsahuje zapisovací API, které daný bod zapíše do databáze dle nakonfigurované organizace a segmentu.

```
var queryBuilder = new StringBuilder()
    .Append($"from(bucket: \"{_bucket}\")")
    .Append($" |> range(start: -{period})")
    .Append($" |> filter(fn: (r) => r[\"_measurement\"] == \"{DbFields.MeasurementName}\")")
    .Append($" |> filter(fn: (r) => r[\"_field\"] == \"{field}\")")
    .Append($" |> filter(fn: (r) => r[\"{DbFields.HeaterTag}\"] == \"{heater.IpAddress}\")")
    .Append($" |> aggregateWindow(every: {HistoryPeriods.AggregationWindow(period)}, fn: mean, createEmpty: false)")
    .Append( " |> yield(name: \"mean\")");

using var client = CreateDbClient();
var tables = await client.GetQueryApi().QueryAsync(queryBuilder.ToString(), _organization);
var measurements = new List<DbRecordModel>();

foreach (var record in tables.SelectMany(table => table.Records))
{
    measurements.Add(new DbRecordModel
    {
        Value = Convert.ToSingle(record.GetValue()),
        MeasurementTime = record.GetTimeInDateTime()
    });
}
return measurements;
```

Obrázek 4.6: Část metody *ReadHistoryAsync*, kde je vytvořen dotaz v jazyce Flux, kterým získá data ze zadaného časového období a vybraného sloupce databázové tabulky pro zvolené topení, které je filtrované dle své IP adresy. Získané záznamy jsou navráceny v kolekci s objekty typu *DbRecordModel*. Jedná se o vlastní model, který je dále využíván v rámci implementace mobilní aplikace. Obsahuje hodnotu měření typu float a časové razítko.

Jmenný prostor *SmartHeater.Shared.Static* obsahuje pomocné statické třídy *DbFields* a *HistoryPeriods* zajišťují kontrolu validity vyžádané položky databáze a časového úseku. K tomuto používají předdefinované konstanty typu string. Obsahují metodu *GetAll* sloužící jako iterátor přes všechny definované konstanty a metodu *IsValid*, která zjistí validitu zadaného parametru porovnáním jeho existence s položkami iterátoru *GetAll*.

```
1 reference
public static string? AggregationWindow(string period) => period switch
{
    Minute1 => "1s",
    Minutes5 or Minutes15 or Minutes30 or Minutes45 or Hour1 => "10s",
    Hours2 or Hours3 or Hours6 => "1m",
    Hours12 => "2m",
    Hours24 => "4m",
    Days2 => "10m",
    Days7 => "30m",
    Days14 => "45m",
    Days30 => "1h",
    _ => null
};
```

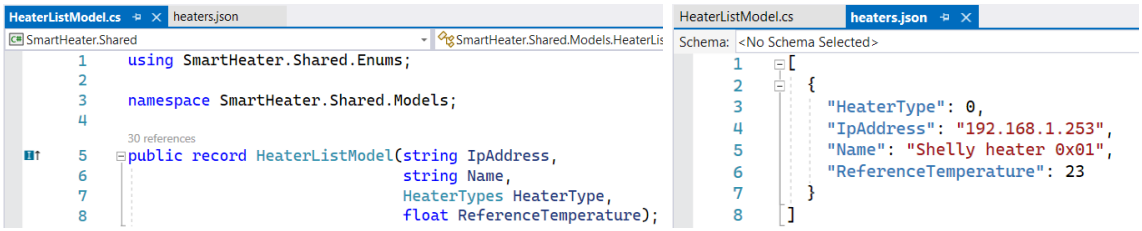
Obrázek 4.7: Třída *HistoryPeriods* navíc pro část Flux dotazu obsahuje metodu *AggregationWindow*, která pro dané časové období (dle odpovídající definované konstanty) vrací příslušnou velikost agregačního okna. Tyto velikosti byly odvozeny od velikostí agregačního okna používané webovým rozhraním InfluxDB pro dané časové úseky. Delší vyžádané časové období představuje větší množství naměřených hodnot a s tím spojené vyšší nároky na paměť a síťový provoz.

4.2.3 Správa a ovládání topení

Rozhraní *IHeatersRepository* poskytuje metody pro správu sítě topení. Umožňuje získání informací o registrované jednotce (*GetHeaterAsync*) i možnost získání detailu s měřením teploty v místnosti a spotřeby energie (*GetHeaterDetailAsync*), čtení i zápis kolekce registrovaných jednotek (*ReadHeatersAsync* a *WriteHeatersAsync*), vkládání (*InsertAsync*), aktualizace (*UpdateAsync*) a mazání jednotlivých topení (*DeleteAsync*), a získání služby implementující rozhraní *IHeaterControlService* pro ovládání daného topení (pro všechny registrované jednotky *GetHeaterServicesAsync*, nebo pro jedno konkrétní registrované topení *GetHeaterService* a *GetHeaterServiceAsync*).

Implementace ve třídě *HeatersRepository* pracuje s kolekcí topení uloženou v souboru *heaters.json* (a pokud neexistuje, vytvoří si jej). Běžně by bylo vhodné tyto záznamy ukládat jako databázové entity. Použitá databáze InfluxDB však pracuje pouze s časovými řadami, pro které je optimalizovaná. V případě tohoto projektu by ale instalace jiného relačního databázového serveru, který by běžel vedle InfluxDB, byla zbytečná.

Jmenný prostor *SmartHeater.Shared.Models* obsahuje třídy použitých datových modelů. Jedním z nich je *HeaterListModel* představující model registrovaného topení. Topení jsou rozlišována dle **IP adresy**. Dále model obsahuje informace o názvu topení, typu topné jednotky (výčtový typ *HeaterTypes* aktuálně obsahuje pouze *Shelly1PM*, ale do budoucna je možné řešení rozšířit i o další typy), a nastavené referenční teplotě, kterou se bude ovládající část snažit v místnosti udržet.



Obrázek 4.8: *HeaterListModel* a soubor *heaters.json* obsahující reprezentaci jejich kolekce ve formátu JSON

Dalším datovým modelem je *HeaterStatusModel*, který představuje záznam o stavu topení. Obsahuje informace, zda je topení ve stavu zapnuto nebo vypnuto, naměřenou teplotu v místnosti, záznam o okamžité spotřebě elektrické energie ve Wattech a časové razítko měření. Model použitý pro získání detailu o topení *HeaterDetailModel* obsahuje informace uložené v rámci *HeaterListModel* i záznam o měření v položce typu *HeaterStatusModel*.

Oddělené rozhraní ovladače topení *IHeaterControlService* obsahuje metody pro zapnutí (*TurnOnAsync*) i vypnutí (*TurnOffAsync*) topné jednotky a získání jejího stavu (*GetStatusAsync*) v modelu *HeaterStatusModel*.

```
5 references
public IHeaterControlService? GetHeaterService(HeaterListModel heater)
{
    return heater.HeaterType switch
    {
        HeaterTypes.Shelly1PM => new ShellyRelayService(_httpClient, heater.IpAddress),
        _ => null
    };
}
```

Obrázek 4.9: Metoda třídy *HeatersRepository* vracející příslušnou implementaci rozhraní *IHeaterControlService* dle konkrétního typu topení. Pro neznámý neimplementovaný typ vrací hodnotu null. V této práci je však použito výhradně zařízení Shelly 1PM, pro které je implementace rozhraní *IHeaterControlService* vytvořena ve třídě *ShellyRelayService*. Ta stojí na komunikaci s vestavěným lokálním REST API [9], a proto je vyžadována konfigurace IP adresy zařízení.

```
3 references
public async Task TurnOnAsync() => await SendTurnRequestAsync("on");

3 references
public async Task TurnOffAsync() => await SendTurnRequestAsync("off");

2 references
private async Task SendTurnRequestAsync(string state)
{
    var data = new[]
    {
        new KeyValuePair<string, string>("turn", state)
    };
    try
    {
        await _httpClient.PostAsync(Relay0Url, new FormUrlEncodedContent(data));
    }
    catch (Exception ex)
    {
        Console.Error.WriteLine($"{DateTime.Now}: {ex.Message}");
    }
}
```

Obrázek 4.10: Třída *ShellyRelayService* obsahuje metody *TurnOnAsync* a *TurnOffAsync* využívají k zasílání příslušného příkazu zapnout/vypnout HTTP metodu *POST* na koncový bod **/relay/0**, jíž je v těle zprávy předán příkaz *"turn" "on"* nebo *"off"* ve formátu *application/x-www-form-urlencoded* dle [1].

```
5 references
public async Task<HeaterStatusModel?> GetStatusAsync()
{
    try
    {
        var response = await _httpClient.GetFromJsonAsync<ShellyRelayStatus>(StatusUrl);
        var status = new HeaterStatusModel(IPAddress, DateTime.UtcNow)
        {
            IsTurnedOn = ReadRelayState(response),
            Temperature = ReadTemperature(response),
            Power = ReadPower(response)
        };
        return status;
    }
    catch
    {
        return null;
    }
}
```

Obrázek 4.11: Metoda *GetStatusAsync* získá data o stavu zařízení Shelly použitím HTTP metody *GET* na koncový bod **/status**. Obsah odpovědi ve formátu JSON je deserializován do privátní třídy *ShellyRelayStatus*, jejíž položky následně dekóduje do objektu typu *HeaterStatusModel* voláním pomocných metod.

Třída *ShellyRelayStatus* mapuje dle [8] následující položky JSON odpovědi:

- *"relays"*: Jedná se o seznam relé obsahující položky s jejich stavem (Shelly 1PM obsahuje pouze jedno relé, takže čteme z první položky). Položkou je zde slovník z jehož položky *"ison"* lze vyčíst, zda je relé ve stavu zapnuto nebo vypnuto.
- *"ext_temperature"*: Seznam přípojených externích teplotních senzorů (v této práci je pro každé topení brán v potaz pouze jeden senzor, takže opět čteme z první položky). Položkou je zde slovník z jehož položky *"tC"* vyčteme teplotu ve °C.
- *"meters"*: Obsahuje informace o naměřené okamžité spotřebě elektrické energie ve Wattech, jejíž hodnota je vyčtena z položky *"power"*.

4.2.4 Komunikace klient-server

Klientská aplikace může komunikovat s řídícím hubem pomocí HTTP metod dle registrovaných koncových bodů na dostupném webovém API ASP.NET Core.

```
79      app.MapGet("/heaters",
80          async (IHeatersRepository hp)
81              => await hp.ReadHeatersAsync()
82      );
83
84      app.MapGet("/heaters/{ipAddress}",
85          async (IHeatersRepository hp, string ipAddress)
86              => await hp.GetHeaterDetailAsync(ipAddress)
87      );
88
89      app.MapPost("/heaters",
90          async (IHeatersRepository hp, HeaterListModel heater)
91              => await hp.InsertAsync(heater)
92      );
93
94      app.MapPut("/heaters/{ipAddress}",
95          async (IHeatersRepository hp, string ipAddress, HeaterListModel heater)
96              => await hp.UpdateAsync(ipAddress, heater)
97      );
98
99      app.MapDelete("/heaters/{ipAddress}",
100         async (IHeatersRepository hp, string ipAddress)
101             => await hp.DeleteAsync(ipAddress)
102     );
```

Obrázek 4.12: Příklad způsobu registrace HTTP metod pomocí minimálního webového API v souboru *SmartHeater.Hub/Program.cs*. V implementaci jsou použity HTTP metody *GET*, *POST*, *PUT* a *DELETE*. Tyto koncové body využívají služby poskytované DI kontejnerem. Na poskytnuté instanci služby jsou volány její asynchronní metody.

Dle zvolené HTTP metody koncové body poskytují následující činnosti:

- *"/heaters"*: Požadavkem metodou *GET* vrací kolekci registrovaných topení typu *HeaterListModel*. Metoda *POST* naopak *HeaterListModel* očekává v těle zprávy s vyplněnými informacemi o novém topení, které přidá do kolekce a uloží do souboru *heaters.json*.
- *"/heaters/{ipAddress}"*: Metoda *GET* pro tento koncový bod vrací detailní informace o topení v modelu *HeaterDetailModel*. Požadavek metodou *PUT* upraví specifikovaný záznam o topení přijatým *HeaterListModel* v těle zprávy. Dále metodou *DELETE* smaže specifikované topení z kolekce uložené v souboru *heaters.json*.
- *"/weather"*: Požadavkem metodou *GET* vrací záznam o aktuální venkovní teplotě získaný ze služby *OpenWeather*.
- *"/heaters/{ipAddress}/history/{period}/{field}"*: Metoda *GET*, která vrací historii uloženou v databázi InfluxDB, která je filtrována v dotaze v jazyce Flux dle topení specifikovaného IP adresou, požadovaným časovým úsekem a typem položky databáze.
- *"/periods"*: Vrací kolekci podporovaných časových úseků pro filtrování Flux dotazem specifikované ve statické třídě *HistoryPeriods*.
- *"/fields"*: Vrací kolekci v implementaci použitých položek uložených v databázi InfluxDB (sloupce tabulky).
- *"/smartheater-availability-test"*: Používaný mobilní aplikací pro snadné zjištění dostupnosti serveru.

4.3 Mobilní aplikace

Multiplatformní aplikace implementovaná nad frameworkem .NET MAUI byla vytvořena jako klient pro komunikaci se serverem běžícím pod frameworkem ASP.NET a umožní uživateli snadnou správu a monitorování sítě topení instalovaných v domácnosti.

Ústředním bodem zdrojového kódu aplikace je její konfigurace v souboru *MauiProgram.cs*. Zde je podobně jako v konfiguraci řídícího hubu poskytován kontejner DI, který poskytuje stránky, viewmodely a další služby (například *HttpClient* pro komunikaci s webovým API). Další konfigurace zdrojů pro XAML se nachází v souborech *App.xaml* a *App.xaml.cs*.

Jako základní prvek propojující uživatelské rozhraní této aplikace je použit vestavěný .NET MAUI typ *Shell*, který dle [27] snižuje složitost vývoje aplikací tím, že poskytuje základní funkce vyžadované většinou aplikací, včetně jediného místa pro popis vizuální hierarchie aplikace, navigační schéma založené na URI, které umožňuje navigaci na jakoukoli stránku v aplikaci a integrovaný obslužný program vyhledávání.

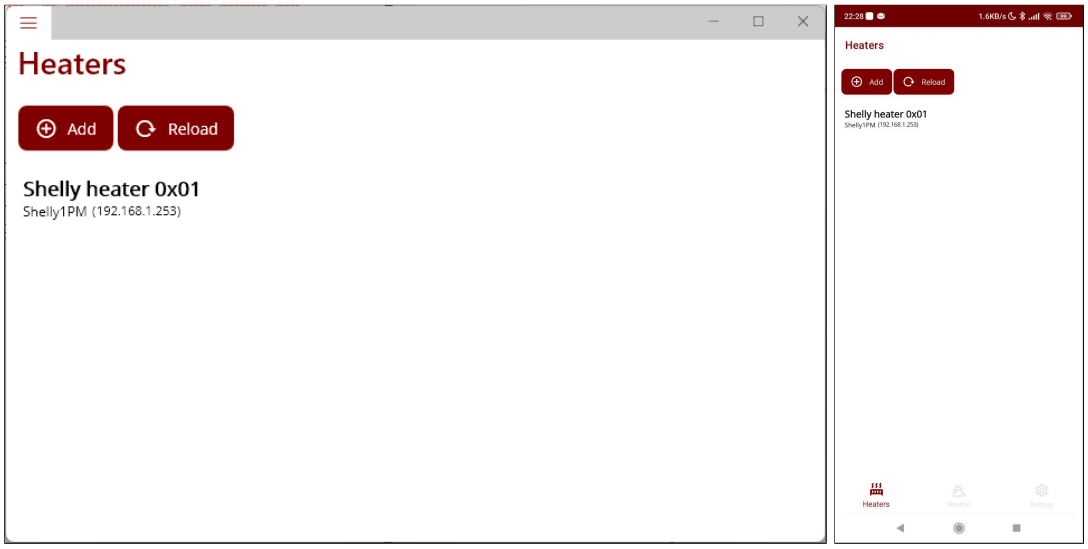
```
1 reference
5   public static Shell Create()
6   {
7       Routing.RegisterRoute(nameof(AddHeaterPage), typeof(AddHeaterPage));
8
9       #if ANDROID || IOS
10          Routing.RegisterRoute("HeaterDetailPage", typeof(HeaterDetailMobilePage));
11          return new MobileShell();
12      #else
13          Routing.RegisterRoute("HeaterDetailPage", typeof(HeaterDetailDesktopPage));
14          return new DesktopShell();
15      #endif
16  }
```

Obrázek 4.13: Statická třída *AppShellProvider* ze jmenného prostoru *SmartHeater.Maui.Providers* zprostředkovává hlavní rozhraní aplikace vestavěného typu *Shell*. Využívá možností direktiv pro překladač, kdy pomocí definovaných proměnných běhového prostředí lze zjistit, zda je překládáno pro mobilní či desktopovou platformu. Dle toho vrací *MobileShell* nebo *DesktopShell*, které jsou definovány ve složce *Pages*, a také upravuje směrování na příslušnou stránku detailu topení, jejíž rozhraní je také upraveno dle platformy.

Zdrojový kód aplikace obsahuje také platformě specifický kód ve složce *Platforms*. Během vývoje byla testována pouze verze pro **Windows**, kde nebylo oproti vytvořené šabloně potřeba nic měnit, a platformu **Android**, pro kterou byl upraven pouze soubor *AndroidManifest.xml* přidáním vyžadovaných systémových oprávnění pro přístup k síťovému rozhraní zařízení.

Verze pro platformy *iOS*, *Mac Catalyst* a *Tizen* nebyly v rámci této práce otestovány, kvůli nemožnosti přístupu k potřebnému hardware (smartphony *Apple iPhone* a počítače *Mac*). Díky vlastnostem použitého frameworku .NET MAUI a sdíleného zdrojového kódu lze ale předpokládat jejich plnou funkčnost.

Samotný sdílený kód stránek aplikace v jazyce XAML se nachází ve složce *Pages*. Všechny dodržují návrhový vzor MVVM, kdy hlavní značka stránky *ContentPage* má specifikovaný svůj datový typ viewmodelu specifikovaný v parametru *x:DataType*. Toto umožňuje použití prvků viewmodelu pro datové vazby v XAML značkách nižší úrovně. V příslušném xaml.cs souboru je v parametru konstruktoru třídy stránky specifikován vyžadovaný viewmodel a přiřazen vlastnosti *BindingContext*. Po spuštění je instance objektu viewmodelu poskytnuta DI kontejnerem.



Obrázek 4.14: Ukázka použití *DesktopShell* pro Windows, který pro navigaci aplikace používá *Flyout* hamburger menu, a *MobileShell* pro Android, který používá navigaci pomocí záložek *TabBar*. Úvodní stránka aplikace zobrazuje seznam registrovaných topných jednotek (v případě testovací instalace je zobrazeno jedno zařízení Shelly 1PM).

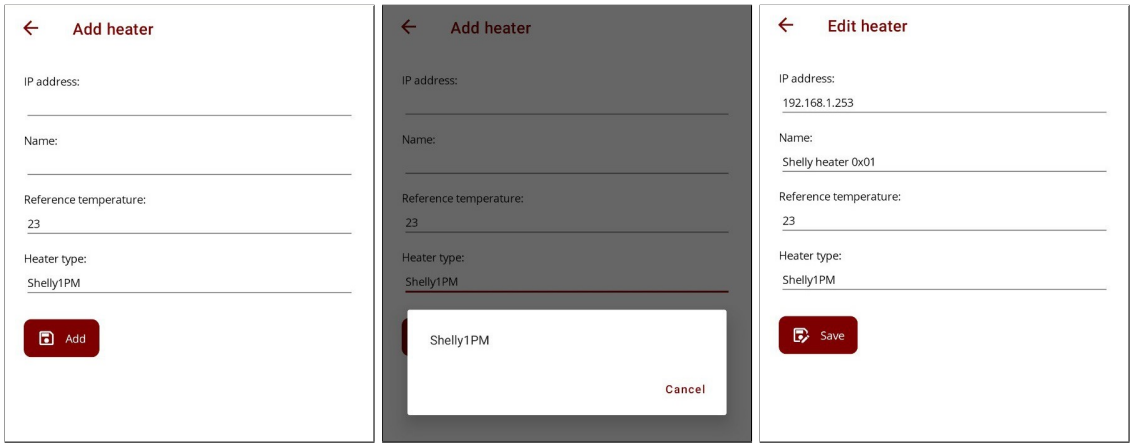
4.3.1 Stránky pro správu topení

Zobrazení seznamu registrovaných topení se nachází v souboru *HeatersPage.xaml*. Krom kolekce zobrazeném v XAML značce *CollectionView* obsahuje také tlačítka pro přidání topení a znovunačtení seznamu (využívá se například po prvotním spuštění, kdy ještě nebyla nakonfigurována IP adresa řídícího hubu na stránce nastavení).

Stránka pro přidání topení *AddHeaterPage* obsahuje textová pole pro prvky třídy *HeaterListModel*, která je součástí kódu sdíleného mezi mobilní aplikací a řídícím hubem. Vyplněná instance *HeaterListModel* je v metodě viewmodelu *AddHeaterViewModel* serializována do formátu JSON a odeslána na webové API řídícího hubu pomocí HTTP metody *POST*. Toto jedno uživatelské rozhraní lze znovupoužít i pro editaci jednotky pomocí *Shell* podporované URI navigace. V případě editace je pro uložení použita metoda *PUT*.

```
127     public void ApplyQueryAttributes(IDictionary<string, object> query)
128     {
129         if (query.Keys.Count > 0)
130         {
131             IpAddress = query["IpAddress"].ToString();
132             Name = query["Name"].ToString();
133             HeaterType = (HeaterTypes)query["HeaterType"];
134             ReferenceTemperature = (float)query["ReferenceTemperature"];
135 
136             Title = "Edit heater";
137             ButtonText = "Save";
138             SaveIcon = IconFontHelper.ContentSaveEditOutline;
139             _originalIpAddress = IpAddress;
140         }
141     }
```

Obrázek 4.15: Viewmodel pro použití URI navigace implementuje rozhraní *IQueryAttributable* a jeho metodu *ApplyQueryAttributes* z jejíhož slovníkového parametru lze získat například informace o editovaném topení. Na stránku je možné se navigovat i bez parametrů, v tomto případě se pak jedná o přidání.



Obrázek 4.16: Uživatelské rozhraní *AddHeaterPage* má v případě přidávání textová pole prázdná a v případě editace vyplněná dle vybraného topení. Výběr typu topení je zde pro případnou budoucí podporu více typů předpřipraven v XAML značce *Picker*. Po kliknutí na tlačítko "Uložit" je formát IP adresy zadané v textovém poli zkontrolován voláním .NET knihovní metody *System.Net.IPAddress.Parse*. Ta v případě chyby vyhodí výjimku *FormatException*, která je zachycena a HTTP požadavek se tudíž neodešle.

Další částí pro samotné monitorování stavu topení je stránka detailu, která se zobrazí po kliknutí na položku v seznamu registrovaných topení na úvodní stránce *HeatersPage*. K této stránce patří viewmodel *HeaterDetailViewModel*, který implementuje podporu URI navigace s parametrem IP adresy topení. Ta je použita jako parametr metody *GetHeaterDataAsync*, která pomocí HTTP metody *GET* stáhne data o daném topení a deserializuje je do objektu typu *HeaterDetailModel*.

Nahoře na stránce se nachází tlačítka s akcemi nad topením pro znovunačtení aktuálních informací o topení, navigaci na editační stránku topení a jeho smazání. Znovunačtení je provedeno voláním metody *ReLoad*, která opět volá *GetHeaterDataAsync* s parametrem IP adresy aktuálně zobrazeného topení. Kliknutí na editační tlačítko naviguje na stránku *AddHeaterPage*, které jsou předány parametry topení k editaci. Tlačítko smazání zobrazí dialog o potvrzení smazání, a pokud je zvolena možnost *"Ano"* posílá HTTP požadavek metodou *DELETE*. Zvolením *"Ne"* je dialog uzavřen a požadavek na smazání serveru není odeslán.

```
128     private async void GoToEditPage()
129     {
130         var parameters = new Dictionary<string, object>
131         {
132             { "IpAddress", HeaterDetail.IpAddress },
133             { "Name", HeaterDetail.Name },
134             { "HeaterType", HeaterDetail.HeaterType },
135             { "ReferenceTemperature", HeaterDetail.ReferenceTemperature }
136         };
137         await Shell.Current.GoToAsync(nameof(AddHeaterPage), parameters);
138     }
```

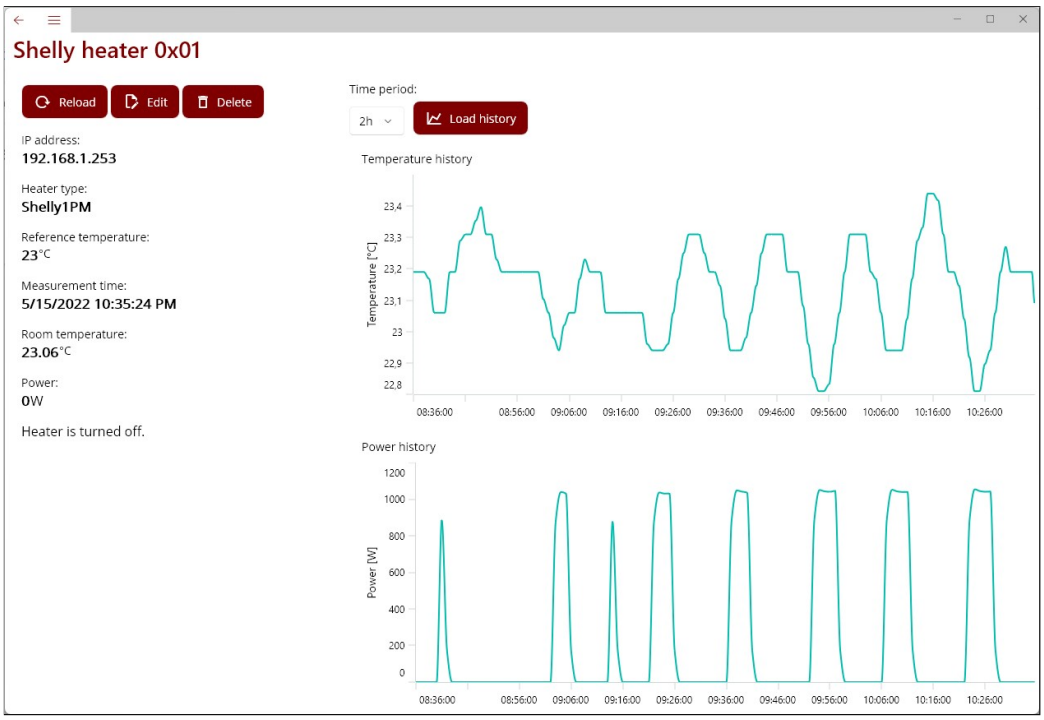
Obrázek 4.17: Metoda *HeaterDetailViewModel* pro URI navigaci na editační stránku topení.

Tato stránka má dvě varianty na základě platformy (mobilní pro Android a iOS, a desktopovou pro ostatní), jak již bylo zmíněno u obrázku 4.13. Kvůli tomuto rozdělení je kód pro zobrazení informací o topení extrahován do pohledu *HeaterDetailView* a *HeaterChartsView*. Tímto se stávají znovupoužitelnými a zamezuje se kopírování kódu.

HeaterDetailView zobrazuje informace z objektu typu *HeaterDetailModel*, který mu je předán pomocí XAML datové vazby.

HeaterChartsView obsahuje grafy z knihovny *Syncfusion.Maui.Charts* dle [24]. Ty umožní uživateli zobrazit si a mít přehled o historii spotřeby elektrické energie a teploty v místnosti, kde je topení umístěno. Tento pohled obsahuje viewmodel typu *HeaterChartsViewModel*, ve kterém jsou uloženy kolekce s daty pro zobrazení v grafech.

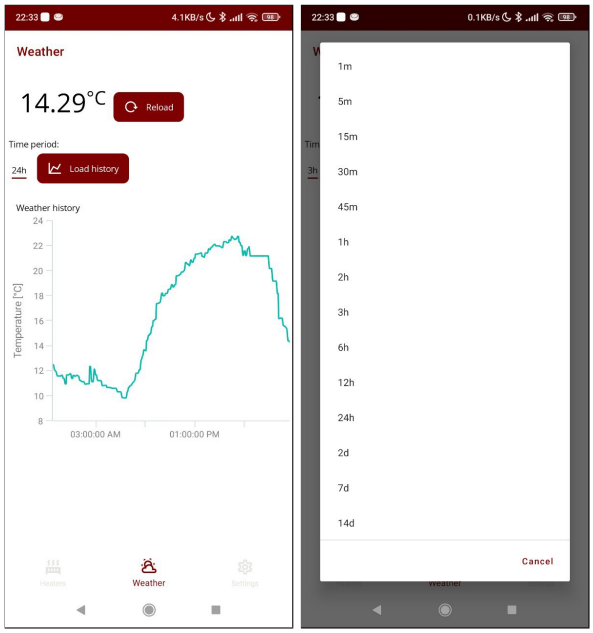
Pro výběr časového období a následného stažení dat byl navíc přidán pohled *PeriodSelectorView*, který pracuje s vlastním viewmodelem typu *PeriodSelectorViewModel*. Ten v parametru konstruktoru přijímá delegát akce, která se vykoná stiskem tlačítka. Předaný delegát metody v případě *HeaterChartsViewModel* odešle HTTP požadavek metodou *GET* pro stažení dat ze zvoleného časového období o položkách databáze teploty a spotřeby. Získaná data jsou následně uložena do kolekcí, které mají datové vazby na grafy.



Obrázek 4.18: Desktopové uživatelské rozhraní detailu topení zobrazuje grafy v *HeaterChartsView* na pravé straně od informací o topení v *HeaterDetailView* a tlačítky s dostupnými akcemi. Pro mobilní zařízení jsou z důvodu úzké obrazovky tyto pohledy umístěny pod sebou.

4.3.2 Zobrazení dat o počasí

O zobrazení dat o venkovním počasí se stará stránka *WeatherPage*. K této stránce patří viewmodel typu *WeatherViewModel*, který se po své inicializaci pomocí HTTP metody *GET* pokusí získat informace o aktuální venkovní teplotě. Tato stránka také pro zobrazení historie počasí obsahuje grafy z knihovny *Syncfusion.Maui.Charts* dle [24] a volbu časového období pomocí *PeriodSelectorView*.



Obrázek 4.19: Stránka zobrazení dat o počasí a výběr časového období.

4.3.3 Stránka nastavení

Lokální nastavení aplikace je uživateli umožněno na stránce *SettingsPage*. Jedná se tedy hlavně o nastavení IP adresy řídícího hubu, kam chodí HTTP požadavky z ostatních stránek. Toto nastavení je v rámci celé aplikace sdíleno třídou *SettingsProvider*, která jej ukládá lokálně do souboru *settings.json*.

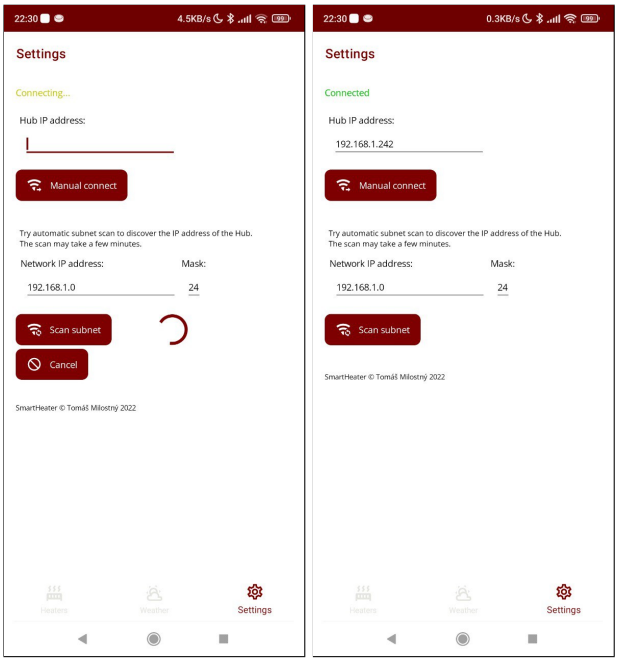
```
33      private static string SettingsJsonFilePath()
34      {
35          var localDir = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
36          var settingsJsonFile = "settings.json";
37          return Path.Combine(localDir, settingsJsonFile);
38      }
```

Obrázek 4.20: Metoda pro získání cesty k souboru *settings.json*, který lze zapsat do speciální složky vyhrazené pro lokální aplikační data.

```
16      public static SettingsProvider LoadFromJson()
17      {
18          var filePath = SettingsJsonFilePath();
19          if (File.Exists(filePath))
20          {
21              var jsonStr = File.ReadAllText(filePath);
22              return JsonConvert.DeserializeObject<SettingsProvider>(jsonStr);
23          }
24          return new();
25      }
```

Obrázek 4.21: K serializaci a deserializaci formátu JSON je použita samotná třída *SettingsProvider*. Této statické metody využívá též kontejner DI pro její inicializaci.

IP adresu řídícího hubu lze zadat do textového pole manuálně, nebo pomocí skeneru lokální sítě. Nahoře na stránce je následně vidět stav připojení.



Obrázek 4.22: Po výběru adresy sítě a délky masky je po kliknutí na tlačítko *"Scan subnet"* spuštěn proces vyhledávání v síti. Tento proces je naznačen indikátorem aktivity a v textu nahoře je zobrazen stav připojování. Pro delší masku sítě může sken trvat déle, a z tohoto důvodu bylo přidáno také tlačítko na zrušení akce.

Zjištění dostupnosti řídícího hubu je z *SettingsPage* odděleno do *HubScannerView*. Dostupnost na dané IP adrese je zjištěna HTTP požadavkem metodou *GET* na koncový bod webového API *"/smartheater-availability-test"*. Metoda *AutoDiscover* používá .NET knihovní metodu *Parallel.ForEachAsync*, která využívá možnosti paralelního spuštění asynchronních úloh pro posílání HTTP požadavků. Pokud od zařízení pod aktuálně zkoumanou IP adresou přijde očekávaná odpověď, řídící hub byl v síti nalezen, hledání je ukončeno a běžícím paralelním úlohám je odesláno upozornění o zrušení, které je propagováno pomocí *CancellationToken*. V případě manuálního zadání IP adresy je dostupnost řídícího hubu zjištěna obdobným způsobem, jen je tento HTTP požadavek odeslán pouze na tuto jednu zadanou IP adresu. Pro samotné procházení podsítě byla přidána pomocná třída *SubnetHelper*.

4.3.4 Pomocné třídy

Implementace mobilní aplikace, krom pomocných tříd modelů a výčtového typu sdílených s řídícím hubem, obsahuje také vlastní pomocné třídy používané jejími stránkami.

SubnetHelper je pomocná třída používaná během skenování lokální sítě na stránce nastavení. Udržuje si zadanou IP adresu, délku masky a pole typu byte s bitovou maskou. Pro inkrementaci IP adresy se pracuje s její podobou v poli typu byte, na které je následně aplikována maska sítě logickou operací *OR*, a pokud jsou všechny prvky po aplikaci masky rovny maximální hodnotě 255, jedná se o poslední adresu, jinak se pokračuje v inkrementaci.

IconFontHelper je pomocná třída vygenerovaná webovou službou *IconFont2Code* (dostupné na https://andreinitescu.github.io/IconFont2Code). Cílem této třídy je poskytnout sadu konstant typu string, které zpřehlední kód v místě použití ikony. Tyto konstanty odpovídají speciálním kódům ikon z fontu *materialdesignicons-webfont.ttf* (soubor je k nalezení mezi zdroji aplikace ve složce *Resources/Fonts*).

```
9           var builder = MauiApp.CreateBuilder();
10          builder
11              .UseMauiApp<App>()
12              .ConfigureSyncfusionCore()
13              .ConfigureFonts(fonts =>
14              {
15                  fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
16                  fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
17                  fonts.AddFont("materialdesignicons-webfont.ttf", "MaterialDesignIcons");
18              });
```

Obrázek 4.23: Konfigurace aplikace je obsažena souboru *MauiProgram.cs*. Font ikon je zde registrován pod klíčem *"MaterialDesignIcons"*.

```
10      <Tab Title="Heaters">
11          <Tab.Icon>
12              <FontImageSource Glyph="{x:Static helpers:IconFontHelper.Radiator}"
13                               FontFamily="MaterialDesignIcons" />
14          </Tab.Icon>
15          <ShellContent ContentTemplate="{DataTemplate pages:HeatersPage}" />
16      </Tab>
```

Obrázek 4.24: Příklad použití třídy *IconFontHelper* pro vložení ikony radiátoru do záložky stránky se správou topení.

Poslední z použitých pomocných tříd jsou třídy ze složky *Converters*, které implementují rozhraní *IValueConverter*. Tyto třídy jsou použity v kódu XAML pro konverzi hodnoty jednoho typu na jiný. Dále jsou v aplikaci dostupné pod klíčem registrovaným v *ResourceDictionary* v souboru *App.xaml*.

Tyto konvertory jsou použity například na stránce nastavení pro konverze hodnoty typu *bool?* na string a barvu dle stavu připojení k řídícímu hubu a konverze hodnoty typu bool na string dle stavu topení zapnuto/vypnuto.

```
30  <StackLayout IsVisible="{Binding LastMeasurement, Converter={StaticResource ShowNotNullConverter}}">
31      <Label Text="Measurement time:" />
32      <Label Text="{Binding LastMeasurement.MeasurementTime, Converter={StaticResource UtcToLocalConverter}}"
33             FontAttributes="Bold"
34             Margin="0, 0, 0, 15"
35             FontSize="17"/>
```

Obrázek 4.25: Dalším příkladem je použití konvertorů je na stránce detailu topení konverze hodnoty typu *DateTime* z formátu UTC do lokálního času a konvertor, který měření stavu topení zobrazí pouze, pokud je validní a není null.

4.4 Předpověď strojovým učením

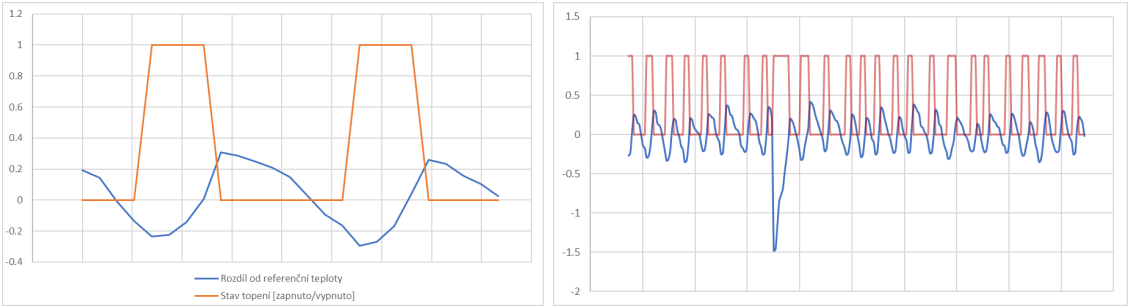
Součástí implementovaného řešení je knihovna tříd *SmartHeater.ML*, která obsahuje projekt postavený nad frameworkem ML.NET. Ústředním bodem, který je používán řídícím hubem je třída *SmartHeaterModel*, která představuje funkce modelu strojového učení. Má dvě hlavní veřejné metody - *EnsureTrainedAsync*, která zajistí případné generování testovacích dat a vytrénování modelu strojového učení; *Forecast* načte vytrénovaný model pro dané topení a dle vstupního objektu *ModelInput* vrátí výsledek předpovědi v objektu typu *ModelOutput*.

4.4.1 Trénovací data

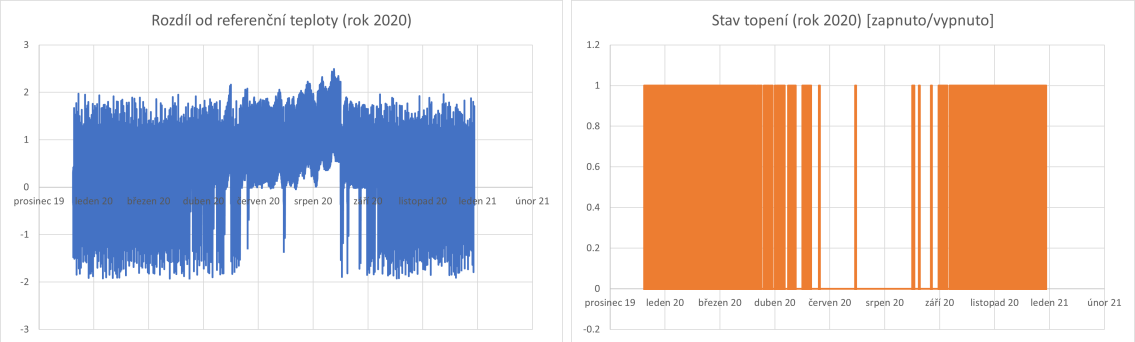
Pro používání modelu strojového učení, jehož předpovědi stojí na algoritmu SSA, potřebuje základní trénovací data představující přibližný průběh funkce běžného domácího vytápění přímotopným konvektorem. Model strojového učení se tuto funkci následně snaží napodobit a na jejím základě sestaví předpověď budoucího vývoje.

Na této myšlence byla postavena třída *MLDataGenerator*, jejíž metoda *RunAsync* zajistí vygenerování trénovacích dat do souboru *training.csv* a testovací data, která na ně navazují, do souboru *testing.csv*. Výsledkem je časová posloupnost, která představuje rozdíl od nastavené referenční teploty. Generované hodnoty se tedy pohybují s určitým rozptylem okolo hodnoty 0. Tento postup byl zvolen z důvodu generování co nejvíce univerzálních dat pro praktické použití (uživatel aplikace si může nastavit libovolnou referenční teplotu v místnosti, kterou se model bude snažit v místnosti udržovat).

Generátor rovněž pracuje se stavem simulovaného topení a používá jej pro vypočtení dalšího stavu teploty v místnosti (stoupá při stavu zapnuto a klesá při stavu vypnuto). Také je zde použita pseudonáhodnost pro generování nedokonalé křivky s určitým rozptylem. Tyto hodnoty jsou počítány s krokem jedné minuty (po dobu roků 2018 a 2019 pro generování trénovacích dat a 2020 pro testovací data). Řídící hub následně po nasazení v praxi používá stejný krok jedné minuty pro plánování ovládání reálných topení.



Obrázek 4.26: Příklad generovaných dat z csv souboru zobrazených v grafech programu *Microsoft Excel*. Krom sledovaného rozdílu od referenční teploty soubor csv obsahuje také datum a čas zobrazené na ose Y, a označení 0/1 představující stav topení. Typický průběh, který je vidět na druhém grafu, obsahuje i výkyvy teploty, kdy teplota vnějším vlivem spadne níže a topení je zapnuto na delší dobu. Opačně může nastat i situace přetopení, kdy je topení déle vypnuto.



Obrázek 4.27: Generátor dat také pracuje s veřejně dostupnými daty o venkovní teplotě poskytované Českým hydrometeorologickým úřadem dle [15] v souladu se zákonem 123/1998 Sb. o právu na informace o životním prostředí. Poskytovaná data představují průměrné denní teploty z období od roku 1961 po rok 2021. V této práci je pro generování použit soubor dat pod Krajem Vysočina z Moravských Budějovic (nejbližší lokace k testovací instalaci v Jaroměřicích nad Rokytnou). Na grafech je vlivem poskytnuté venkovní teploty vidět kolísání rozdílu teploty nad hodnotou 0 a vypnuté topení v letních měsících.

4.4.2 Model strojového učení

Trénování modelu *SmartHeaterModel* probíhá v metodě *TrainPipeline*, jíž jsou předána trénovací data načtená z generovaného csv souboru a objekt typu *MLContext*, což je dle [4] společný kontext pro všechny operace ML.NET, který poskytuje způsob, jak vytvořit komponenty pro přípravu dat, konstrukci funkcí, trénování, predikci a vyhodnocení modelu.

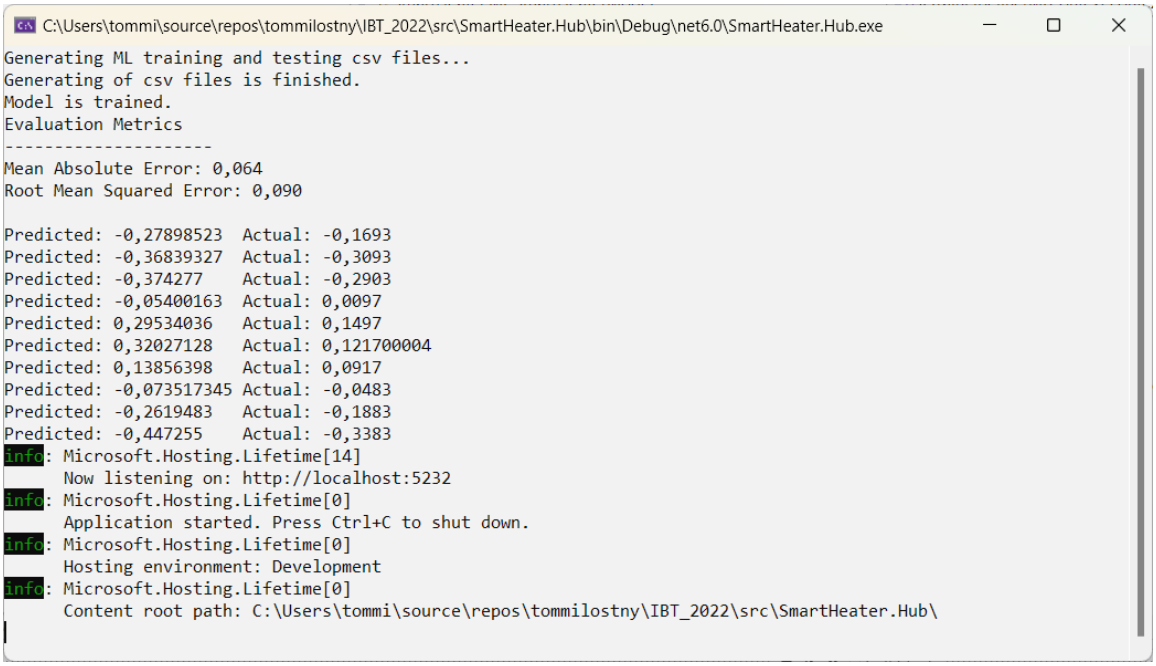
```
87      private static ITransformer TrainPipeline(MLContext context, IDataView trainData)
88      {
89          var pipeline = context.Forecasting
90              .ForecastBySsa(windowSize: 4,
91                             seriesLength: 12,
92                             trainSize: 1051200,
93                             horizon: 10,
94                             confidenceLevel: 0.95f,
95                             outputColumnName: @"temperatureDiff",
96                             inputColumnName: @"temperatureDiff",
97                             confidenceLowerBoundColumn: @"temperatureDiff_LB",
98                             confidenceUpperBoundColumn: @"temperatureDiff_UB");
99          var model = pipeline.Fit(trainData);
100         return model;
101     }
```

Obrázek 4.28: Metoda trénování modelu strojového učení si prvně vytvoří *pipeline* předpovědi nad časovými řadami používající SSA. Tato část zdrojového kódu byla částečně vygenerována nástrojem *Model Builder*, který poskytuje automatické trénování na základě poskytnutých dat. Na konfigurované *pipeline* je volána metoda *Fit*, která provádí samotné trénování modelu hledáním funkce blížící se bodům předaných trénovacích dat.

Celkově dva roky po minutových krocích představuje 1 051 200 bodů trénovacích dat. Časová řada je parametrem *seriesLength* vzorkována ve 12minutových intervalech, což dává celkem 87 600 vzorků, které jsou jednotlivě analyzovány ve 4minutových oknech. Model je pak nastaven, aby prováděl předpovědi v horizontu 10 minut.

Nakonec dle [29] víme, že předpověď je informovaný odhad a není vždy 100% přesná. Proto je dobré znát rozsah hodnot v nejlepším a nejhorším scénáři, jak je definováno horní

a dolní hranicí. V tomto případě je úroveň spolehlivosti pro dolní a horní mez nastavena na 95 %. Čím vyšší hodnota, tím širší je rozsah mezi horní a dolní hranicí pro dosažení požadované úrovně spolehlivosti.



Obrázek 4.29: Řídící hub na začátku svého hlavního souboru *Program.cs* obsahuje volání metody *EnsureTrainedAsync*, která zkontroluje existenci souboru *SmartHeaterModel.zip*, který obsahuje vytrénovaný model, a pokud neexistuje pokračuje ke generování trénovacích a testovacích dat, pokud opět neexistují. Vytvořená trénovací data načte do proměnné typu *IDataView* a předá je metodě *TrainPipeline*. Nakonec je z vytrénovaného modelu vytvořen objekt typu *TimeSeriesPredictionEngine*, který voláním metody *CheckPoint* uloží naučený model do souboru *SmartHeaterModel.zip*.

Na závěr je po uložení vytrénovaného modelu načten i soubor s testovacími daty a předán metodě *Evaluate*, která dle [29] provede výpočet relevantních metrik nad testovacími daty oproti testovací předpovědi:

- *Střední absolutní chyba* (MSE) měří, jak blízko jsou předpovědi skutečné hodnotě.
- *Střední kvadratická odchylka* (RMSE) shrnuje chybu v modelu.

Hodnota obou metrik se pohybuje mezi 0 a nekonečnem a u obou platí, že, čím blíže je hodnota k 0, tím lepší je kvalita modelu.

Následně je po postupu generování, trénování i vyhodnocení modelu strojového učení metoda *EnsureTrainedAsync* dokončena a je inicializováno webové API běžící na frameworku ASP.NET Core a model je připraven pro použití v rámci ovládání reálných topení.

4.4.3 Ovládání topení na základě předpovědi strojového učení

Vytrénovaný model strojového učení je použit v rámci plánovaného periodického spouštění pomocí knihovny *Coravel* [10]. Pro tento účel je v souboru *Program.cs* řídícího hubu tento plánovač nakonfigurován, aby v minutových intervalech pravidelně spouštěl metodu *Invoke* implementovanou ve třídě *MLInvocable*.

```
14      public async Task Invoke()
15      {
16          foreach (var heater in await _heatersRespository.ReadHeatersAsync())
17          {
18              await SmartHeaterActionAsync(heater);
19          }
20      }
```

Obrázek 4.30: Asynchronní metoda *Invoke* využívá službu rozhraní *IHeatersRepository* jejíž implementace je doplněna kontejnerem DI. Tímto způsobem získá kolekci všech registrovaných topení, na kterých provede rozhodnutí o jejich ovládání voláním metody *SmartHeaterActionAsync*.

Metoda *SmartHeaterActionAsync* získá službu pro ovládání topení, ze které následně vyčte jeho aktuální stav. Pro vytvoření objektu *ModelInput* je vypočten rozdíl od referenční teploty nastavené u konkrétní topné jednotky. Dále je volána metoda *Forecast*, které je krom vytvořeného vstupu modelu předána i IP adresa aktuálně ovládaného topení.

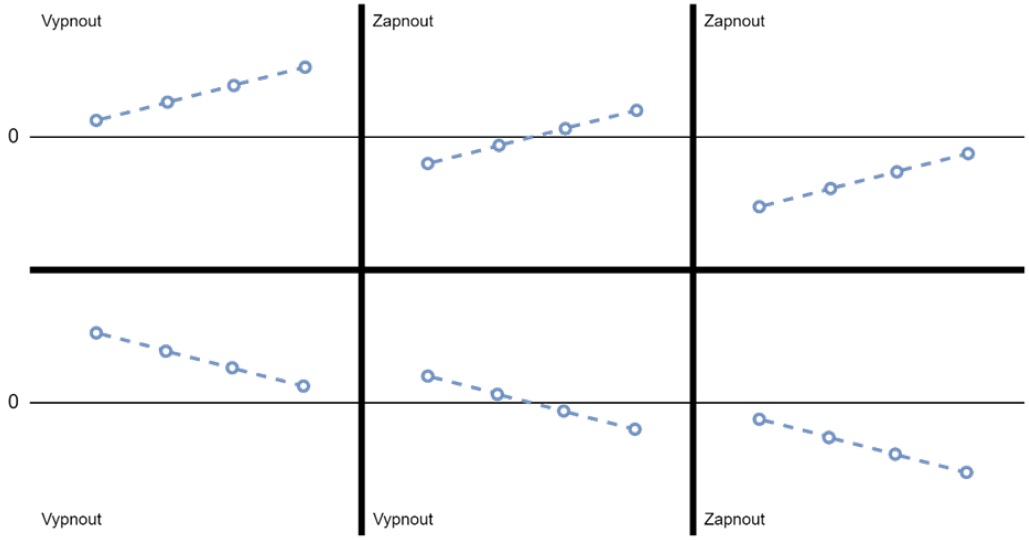
Každému ovládanému topení je přiřazen vlastní zip soubor modelu. Ten začíná jako kopie souboru *SmartHeaterModel.zip* a postupem času je po predikcích ukládán metodou *Checkpoint*. Tímto způsobem je dosaženo postupné zlepšení modelu díky získaným zkušenostem s konkrétní topnou jednotkou.

```
73      private static TimeSeriesPredictionEngine<ModelInput, ModelOutput> CreatePredictEngine(MLContext mlContext,
74                                                                                              string? ipAddress,
75                                                                                              out string modelPath)
76      {
77          modelPath = ModelPathFromIP(ipAddress);
78          if (!File.Exists(modelPath))
79          {
80              File.Copy(MLContants.DefaultModelFilePath, modelPath);
81          }
82          var mlModel = mlContext.Model.Load(modelPath, inputSchema: out var _);
83          return mlModel.CreateTimeSeriesEngine<ModelInput, ModelOutput>(mlContext);
84      }
```

Obrázek 4.31: Metoda *Forecast* využívá pro načtení *TimeSeriesPredictionEngine* metodu *CreatePredictEngine*. Ta nejdříve zjistí existenci souboru modelu pro konkrétní topení a v případě, že neexistuje jej vytvoří. Následně tento model načte a vrátí. Metoda *Forecast* na něm dále zavolá metodu *Predict*, jejíž výstup typu *ModelOutput* představuje výsledek předpovědi s nímž může volající metoda *SmartHeaterActionAsync* dál pracovat.

Výsledek předpovědi je nyní možné použít pro rozhodnutí o budoucím stavu topení. Proto je volána metoda *GetDecisionMetrics*, která získá tři rozhodující metriky:

- Dochází k *přetápění*, nebo je teplota v místnosti vyšší externím vlivem. Tento případ nastává, pokud jsou všechny hodnoty předpovědí větší než 0.
- Dochází k *nedostatečnému vytápění*, pokud jsou všechny hodnoty předpovědi menší než 0.
- Hodnoty předpovědi jsou kolem 0, tak je rozhodnuto dle směru předpovězené funkce.



Obrázek 4.32: Konečného rozhodnutí je tedy dosaženo následujícím způsobem. V případě přetápění je topení odeslán příkaz *vypnout* a v případě nedostatečného vytápění je odeslán příkaz *zapnout*. V obou těchto případech tedy nezáleží na směru předpovězené funkce. V posledním případě, kdy se hodnoty předpovězené funkce pohybují kolem 0, je topení zapnuto, pokud je směr funkce rostoucí, a vypnuto v případě klesajícího směru (vývoj reálné teploty v místnosti by měl tedy odpovídat směru předpovědi).

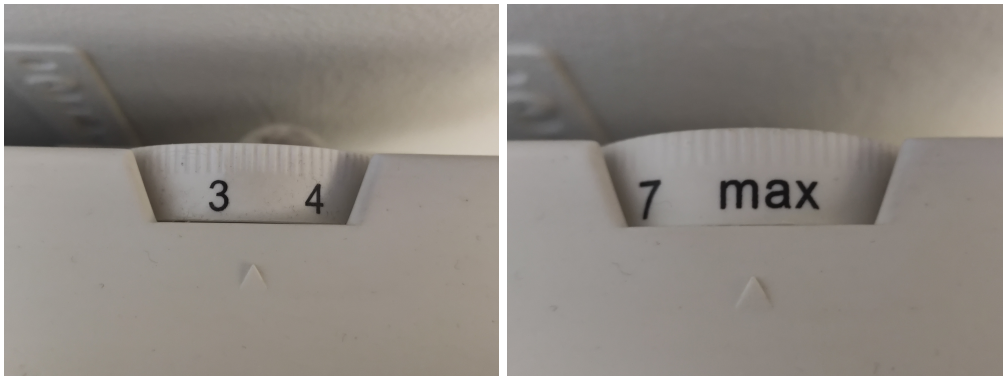
```
78      //Compute over and under heating booleans.
79      overheating  = new Lazy<bool>(() => forecast.TemperatureDiff.All(x => x > 0));
80      underheating = new Lazy<bool>(() => forecast.TemperatureDiff.All(x => x < 0));
81
82      //Compute the trend as the average value of differences between predicted values.
83      trend = new Lazy<float>(() =>
84      {
85          float sum = .0f;
86          int cnt = 0;
87          do
88          {
89              sum += forecast.TemperatureDiff[cnt + 1] - forecast.TemperatureDiff[cnt];
90          }
91          while (++cnt < forecast.TemperatureDiff.Length - 1);
92          return sum / cnt;
93      });
```

Obrázek 4.33: Metoda *GetDecisionMetrics* využívá k výpočtu těchto metrik koncept *odložené inicializace*, kterou umožňuje .NET třída *Lazy*. Tohoto principu je využito například v případě, kdy je již rozhodnuto o stavu topení na základě aktuálního přetápění, je již zbytečné plýtvat zdroji na výpočet směru předpovězené funkce.

4.5 Sledování chování systému

Sběr dat pro monitorování topení je prováděn třídou *StatsCollectorInvocable*, která je periodicky spouštěna plánovačem *Coravel* podobně jako samotné ovládání strojovým učením. Má na starosti sběr dat o všech registrovaných topných jednotkách (okamžitá spotřeba elektrické energie a teplota v místnosti) a o počasí, která získá pomocí služeb *IHeaterControlService* a *IWeatherService*. Následně je pomocí služby *IDatabaseService* zapíše do databáze. Takto získaná data jsou dále také zpřístupněna uživateli mobilní aplikace.

Kód metody *Invoke* této třídy je spouštěn v 10sekundových intervalech. Bylo potřeba zvolit periodu dostatečně krátkou pro zajištění dostatečně detailní data pro zachycení různých vlivů prostředí na teplotu v místnosti.

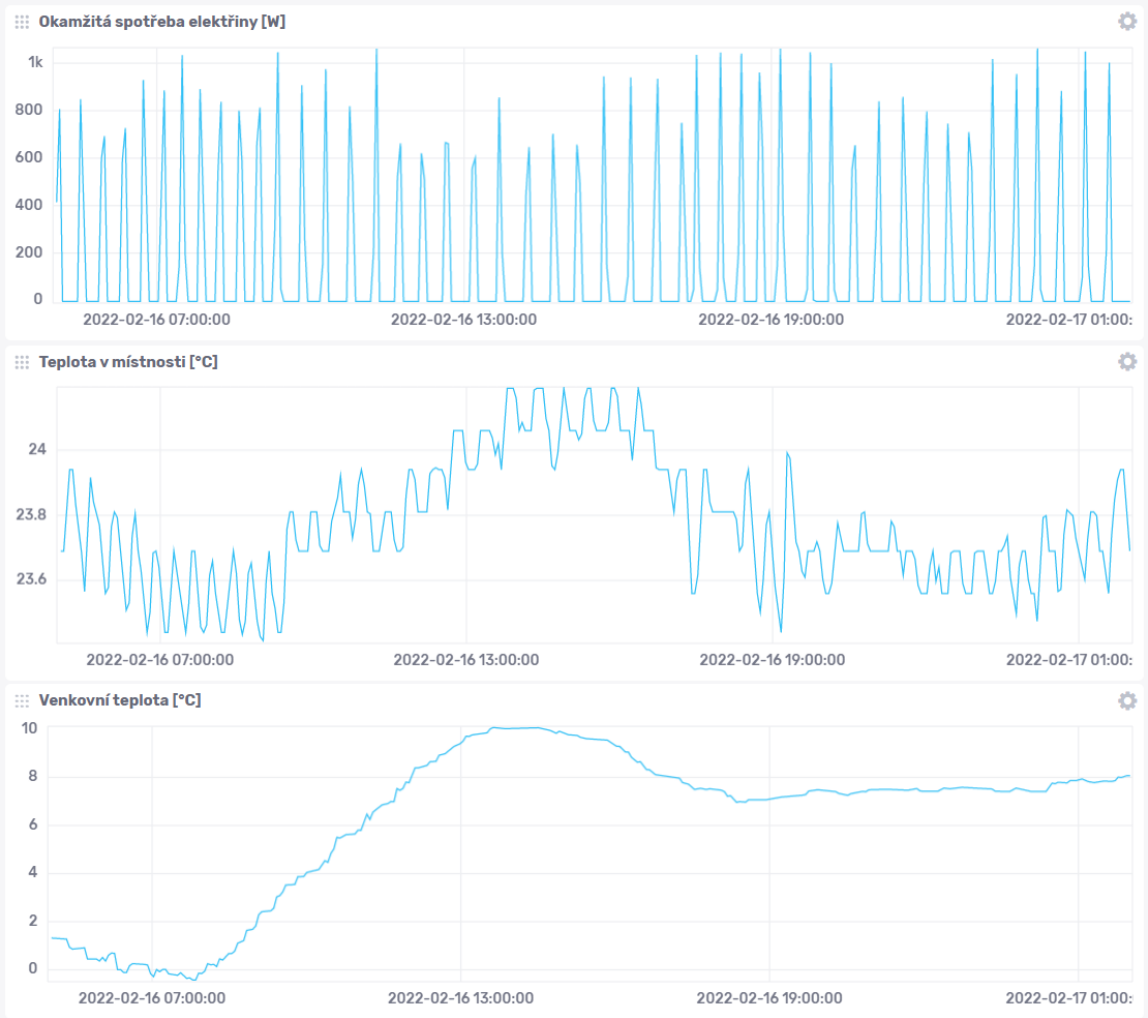


Obrázek 4.34: Fotografie nastavení termostatu použité při testování. Původní nastavení bez ovládání strojovým učením mělo zajistit teplotu v místnosti okolo 23 °C. Na druhé fotografii je tento termostat nastaven na hodnotu MAX a je tedy plně ovládán řídícím softwarem běžícím na Raspberry Pi.

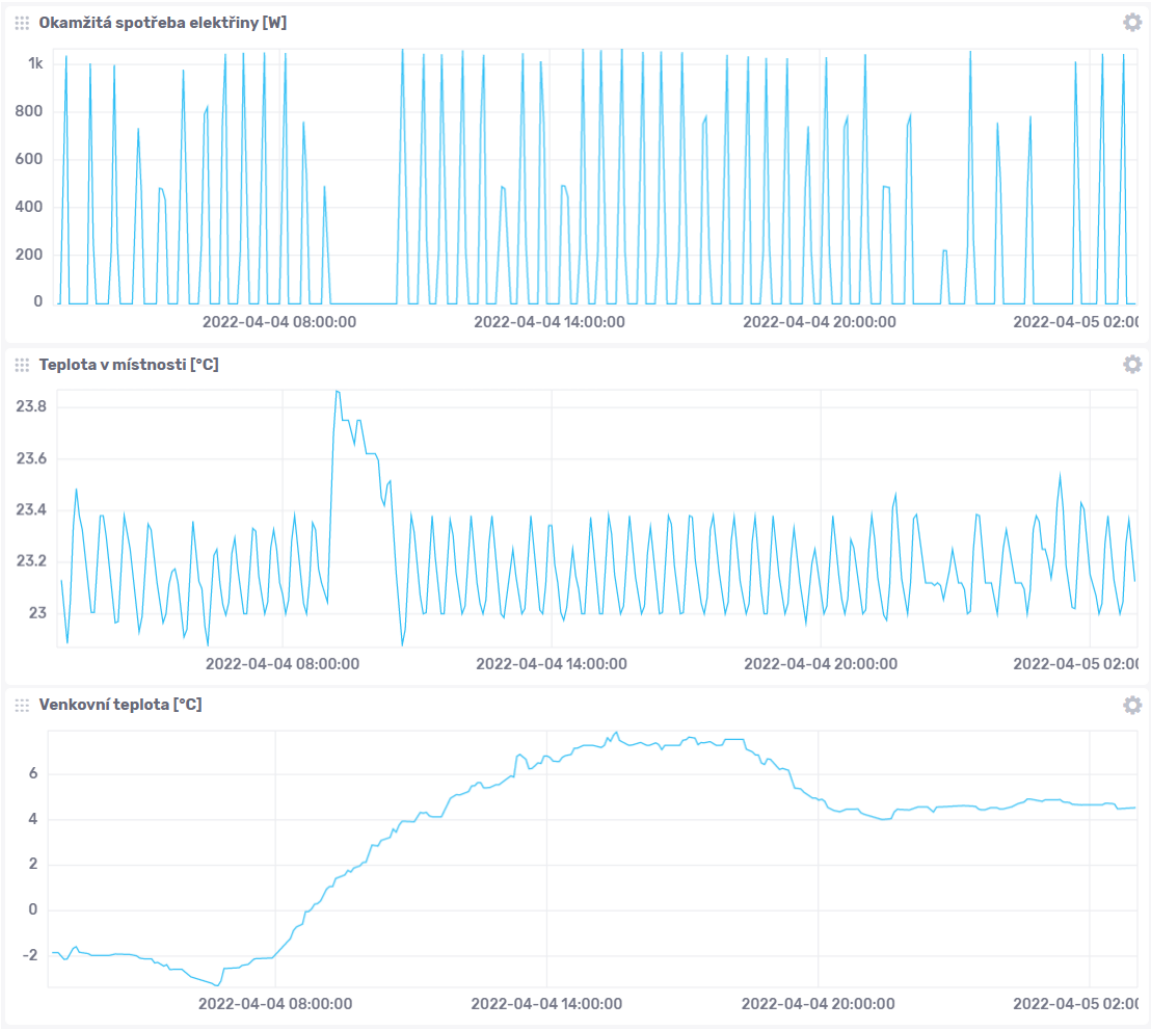
Chování původního termostatu přímotopného konvektoru bylo sledováno v období od 8. února 2022 do 27. března 2022. Zařízení Shelly 1PM, ze kterého byla čtena data bylo instalováno a nastaveno tak, aby bylo neustále zapnuté a zastávalo tedy pouze roli sondy umístěné mezi topení a zásuvku.

Na základě pozorovaného chování původního termostatu byl právě navržen generátor trénovacích dat pro strojové učení, jímž generovaný průběh by měl odrážet spínání topení a nárůsty a poklesy teploty v určité toleranci oproti té referenční.

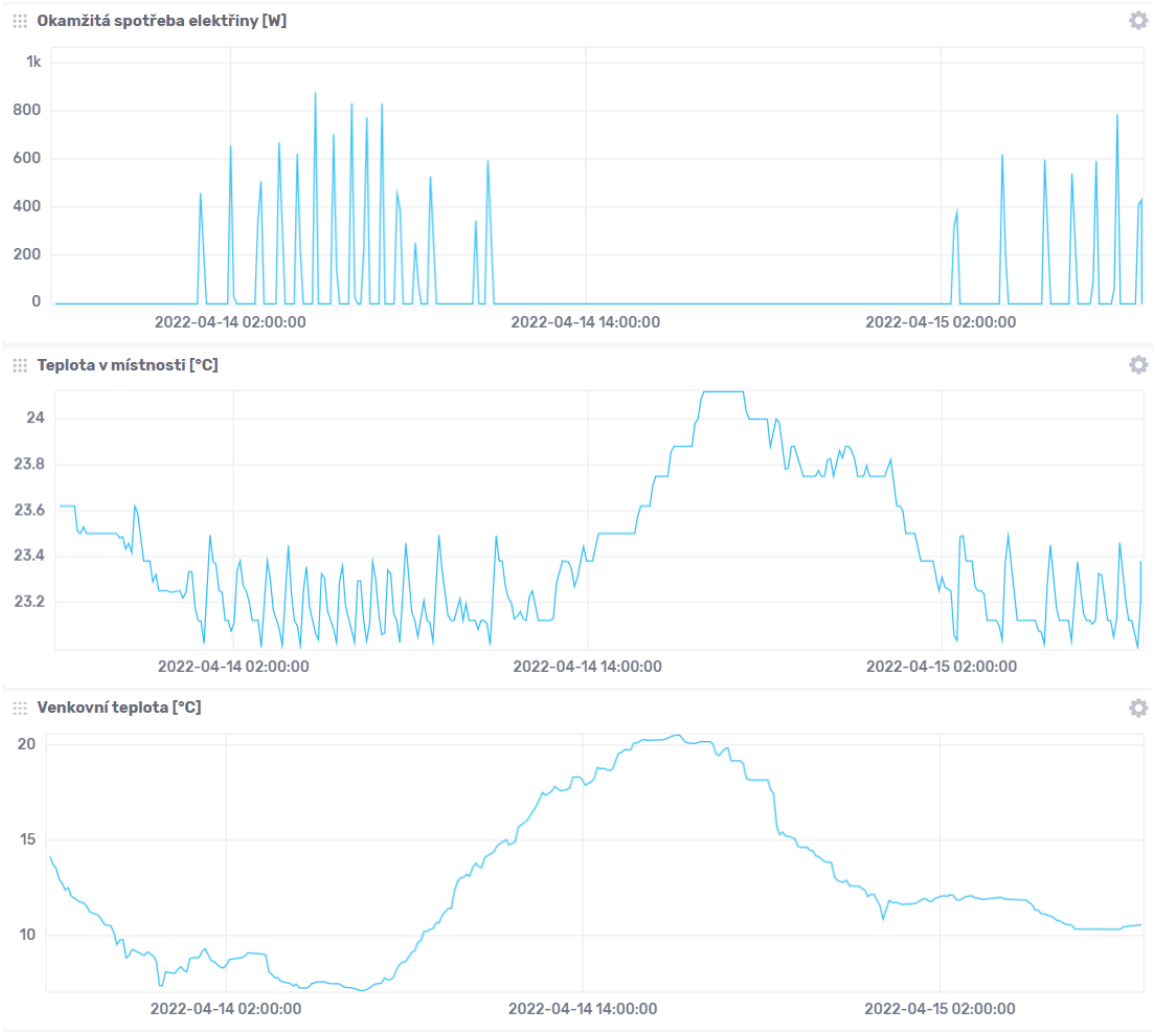
Získaná data jsou zobrazena v grafech získané z webového prostředí databáze InfluxDB. Hodnoty jsou vzhledem k širším pohledům agregovány a nejsou zobrazeny pouze hodnoty 0 nebo ~1000 W.



Obrázek 4.35: Typické chování původního termostatu ukazují například data z 16. února 2022. Na tomto grafu pozorování je vidět, že topení je v průběhu dne schopno udržovat příjemnou teplotu. Stále je však stejným způsobem spínáno i v případě, že externím vlivem nastává situace ”přetopení”.



Obrázek 4.36: Při použití softwarového ovládání na příkladu ze 4. dubna 2022 lze pozorovat podobné stabilní udržení teploty. Navíc je ale toto řešení schopné reagovat na výkyvy teploty, a topení se v případě situace možného přetápění nespíná. V chladné dny jsou tyto výkyvy způsobeny různými externími zdroji (například vytápění rodinného domu dřevem v krbových kamnech).



Obrázek 4.37: V závěrečné části testování lze například na datech z 14. a 15. dubna 2022 pozorovat vliv vyšší venkovní teploty, na který je řízení topení schopno reagovat vypnutím topení na delší dobu. Topení se následně například spíná až s večerním poklesem venkovní teploty.

Kapitola 5

Závěr

Cílem této práce bylo vytvořit systém chytré domácnosti se specifickým zaměřením na domácí elektrické vytápění. Na základě analýzy existujících řešení se podařilo navrhnout a implementovat kompletní systém běžící na platformě .NET, která poskytla všestranné nástroje. Vytvořený systém pracuje se sítí přímotopných konvektorů, které jsou připojeny pomocí Wi-Fi relé Shelly 1PM. Pro uživatelské pohodlí se rovněž podařila vytvořit multiplatformní mobilní aplikace umožňující snadnou správu a monitorování instalovaných topných jednotek.

Systém je následně schopen tato připojená topení automaticky ovládat s využitím logiky založené na předpovědi strojovým učením nad časovou řadou. Topení jsou ovládána jednotlivě a vytvořený systém je schopen se pro každé z nich přizpůsobit místnosti, ve které je dané topení instalováno.

Během vývoje tohoto systému bylo myšleno i na jeho budoucí rozšíření o možnou podporu dalších zařízení mimo Shelly 1PM (například podporu jiných systémů vytápění, nebo existující chytré přímotopné konvektory). Vytvořená klientská mobilní aplikace také otevírá možnost budoucí implementace dalších klientských služeb (například integrace do otevřené platformy chytré domácnosti typu *Home Assistant*). Nakonec na základě dlouhodobého testu vytvořeného systému ovládání by bylo například síť topných jednotek rozšířit o systém chlazení, kterým by byla zajištěna spolehlivější regulace výkyvů teploty a rozšířena možnost smysluplnějšího využití systému i v teplejších letních měsících.

Literatura

[1] *Common HTTP API* [online]. Shelly Cloud. Allterco Robotics LTD. Dostupné z: https://shelly-api-docs.shelly.cloud/gen1/#common-http-api.

[2] *Flux documentation* [online]. InfluxData. Dostupné z: https://docs.influxdata.com/flux/v0.x/.

[3] *Get started with Flux* [online]. InfluxData. Dostupné z: https://docs.influxdata.com/flux/v0.x/get-started/.

[4] *MLContext Class* [online]. Microsoft Docs. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.mlcontext?view=ml-dotnet.

[5] *Nedis WIFIHTPL20FWT* [online]. Seznam.cz, a.s. Dostupné z: https://www.zbozi.cz/vyrobek/nedis-wifihtpl20fwt/.

[6] *Shelly 1PM knowledge base* [online]. Shelly Cloud. Allterco Robotics LTD. Dostupné z: https://shelly.cloud/knowledge-base/devices/shelly-1pm/.

[7] *Shelly temperature sensor* [online]. Shelly Cloud. Allterco Robotics LTD. Dostupné z: https://www.shelly.si/en/accessories-for-shelly-products/23-shelly-temperature-sensor-3809511202456.html.

[8] *Shelly1/1PM: /status* [online]. Shelly Cloud. Allterco Robotics LTD. Dostupné z: https://shelly-api-docs.shelly.cloud/gen1/#shelly1-1pm-status.

[9] *Shelly1/PM: Overview* [online]. Shelly Cloud. Allterco Robotics LTD. Dostupné z: https://shelly-api-docs.shelly.cloud/gen1/#shelly1-pm-overview.

[10] *Task Scheduling* [online]. Coravel. Dostupné z: https://docs.coravel.net/Scheduler/.

[11] *Temperature Sensor AddOn for Shelly 1/1PM* [online]. Shelly Cloud. Allterco Robotics LTD. Dostupné z: https://shop.shelly.cloud/temperature-sensor-addon-for-shelly-1-1pm-wifi-smart-home-automation.

[12] *TUYA WIFI TERMOSTAT* [online]. Smart-switch.cz. Dostupné z: https://www.smart-switch.cz/tuya-smart-life/tuya-wifi-termostat.

[13] *What is ML.NET?* [online]. Microsoft .NET. Dostupné z: https://dotnet.microsoft.com/en-us/learn/ml-dotnet/what-is-mldotnet.

[14] *Convection heater* [online]. Wikipedia, The Free Encyclopedia, prosinec 2021. Dostupné z: https://en.wikipedia.org/w/index.php?title=Convection_heater&oldid=1062355519.

[15] *Denní data dle zákona 123/1998 Sb.* [online]. Český hydrometeorologický úřad, 2021. Dostupné z: https://www.chmi.cz/historicka-data/pocasi/denni-data/Denni-data-dle-z.-123-1998-Sb.

[16] *InfluxDB README.md* [online]. InfluxData, listopad 2021. Dostupné z: https://github.com/influxdata/influxdb/blob/master/README.md.

[17] *Nginx* [online]. Wikipedie: Otevřená encyklopedie, listopad 2021. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Nginx&oldid=20683059.

[18] *Raspberry Pi 4 Product Brief* [online]. Raspberry Pi Trading Ltd., leden 2021. Dostupné z: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf.

[19] *What is ML.NET and how does it work?* [online]. Microsoft Docs, září 2021. Dostupné z: https://docs.microsoft.com/en-us/dotnet/machine-learning/how-does-mldotnet-work.

[20] *ASP.NET Core fundamentals overview* [online]. Microsoft Docs, březen 2022. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-6.0&tabs=linux.

[21] *Create web APIs with ASP.NET Core* [online]. Microsoft Docs, duben 2022. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-6.0.

[22] *Data binding and MVVM* [online]. Microsoft Docs, duben 2022. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/xaml/fundamentals/mvvm.

[23] *Electic heating* [online]. Wikipedia, The Free Encyclopedia, únor 2022. Dostupné z: https://en.wikipedia.org/w/index.php?title=Electric_heating&oldid=1069710541.

[24] *Getting Started with .NET MAUI Chart* [online]. Syncfusion Inc. Documentation, duben 2022. Dostupné z: https://help.syncfusion.com/maui/cartesian-charts/getting-started.

[25] *Host ASP.NET Core on Linux with Nginx* [online]. Microsoft Docs, duben 2022. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/linux-nginx?view=aspnetcore-6.0.

[26] *Nest Thermostat* [online]. Wikipedia, The Free Encyclopedia, květen 2022. Dostupné z: https://en.wikipedia.org/w/index.php?title=Nest_Thermostat&oldid=1083084115.

[27] *.NET MAUI Shell overview* [online]. Microsoft Docs, duben 2022. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/fundamentals/shell/.

[28] *A tour of the C# language* [online]. Microsoft Docs, březen 2022. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

[29] *Tutorial: Forecast bike rental service demand with time series analysis and ML.NET* [online]. Microsoft Docs, duben 2022. Dostupné z: https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/time-series-demand-forecasting.

[30] *What is .NET MAUI?* [online]. Microsoft Docs, duben 2022. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/what-is-maui.

[31] *XAML* [online]. Microsoft Docs, leden 2022. Dostupné z: https://docs.microsoft.com/en-us/dotnet/maui/xaml/.