Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Návrh a implementace webového aplikačního rozhraní a mobilní aplikace pro ovládání klimatické jednotky

Bc. Nikola Jačková

Diplomová práce
2023

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Nikola Jačková**
Osobní číslo:	**I21275**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Návrh a implementace webového aplikačního rozhraní a mobilní aplikace pro ovládání klimatické jednotky**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

V teoretické části práce budou popsány současné trendy v oblasti chytrých domů, tzv. smart homes a to jak z pohledu technologií tak aplikací s důrazem na jejich ovládání.

V rámci praktické části diplomové práce bude prvně nutné navrhnout a implementovat webové aplikační rozraní, a to nad existujícím databázovým modelem. Webové aplikační rozhraní bude nasazené na webovém serveru v rámci kontejnerového systému a přístupné přes platformu Swagger. V dalším kroku bude nutné navrhnout vhodné UI aplikace a to tak, aby odpovídalo současným trendům. Následně se diplomat již zaměří na vlastní návrh a implementaci mobilní aplikace, která uživateli umožní vzdálené ovládání klimatizační a rekuperační jednotky, a to včetně základní konfigurace. Mobilní aplikace by měla být primárně navržena pro prostředí iOS popřípadě i pro platformu Android. Výsledná aplikace po bude testována v reálném prostředí a bude nasazena do App Store, popřípadě Google Play.

Rozsah pracovní zprávy: **60**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

REYNDERS,Fanie. *Modern API design with ASP.net core 2: building cross-platform back-end systems.* New York, NY: Springer Science+Business Media, 2018. ISBN 978-1484235188

HERMES, Dan. *Xamarin Mobile Application Development: Cross-Platform C# and Xamarin.Forms Fundamentals*. California: Apress, [2015]. ISBN 978-1484202159.

Vedoucí diplomové práce: **Ing. Jan Fikejz, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **8. listopadu 2022**
Termín odevzdání diplomové práce: **19. května 2023**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 30. listopadu 2022

Prohlašuji:

Tuto práci jsem vypracovala samostatně. Veškeré literární prameny a informace, které jsem v práci využila, jsou uvedeny v seznamu použité literatury.

Byla jsem seznámena s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 18. 8. 2023

Nikola Jačková

PODĚKOVÁNÍ

Ráda bych zde v první řadě poděkovala svému vedoucímu diplomové práce Ing. Janu Fikejzovi, Ph.D. za jeho odborné vedení, trpělivost a vstřícnost při konzultacích a vypracování práce. Další poděkování patří Ing. Janu Mesarčovi za jeho praktické rady při vývoji mobilní aplikace.

ANOTACE

Diplomová práce se zabývá návrhem a implementací webového aplikačního rozhraní a mobilní aplikace pro ovládání klimatizační jednotky. Klientská mobilní aplikace využívá vybudované API pro volání funkcí systému a pro její implementaci byl zvolen multiplatformní framework .NET MAUI. V teoretické části je podrobněji popsán koncept a využití internetu věcí a softwarové technologie, které byly použity při implementaci.

KLÍČOVÁ SLOVA

Chytré domy, IoT, REST API, .NET, mobilní aplikace.

TITLE

Design and implementation of a web application interface and a mobile application for controlling the climate unit

ANNOTATION

The diploma thesis deals with the design and implementation of a web application interface and a mobile application for controlling the air conditioning unit. The client mobile application uses the API for calling system functions, and the .NET MAUI multi-platform framework was chosen for its implementation. In the theoretical part, the concept and use of the Internet of Things and the software technology used in the implementation are described in more detail.

KEYWORDS

Smart homes, IoT, REST API, .NET, mobile application.

OBSAH

Seznam obrázků **9**

Seznam tabulek **10**

Seznam zkratek **11**

Úvod **12**

1 Internet věcí (IoT) **14**

1.1 Definice a charakteristika 14

1.2 Architektura a technologie IoT 15

1.3 Oblasti využití 16

1.3.1 Smart Homes 18

2 Přehled technologií **21**

2.1 .NET 21

2.1.1 ASP.NET Core 22

2.2 .NET MAUI 23

2.2.1 Xamarin.Forms a .NET MAUI 26

2.2.2 Kontroverze .NET MAUI 27

2.3 Docker 28

2.4 Azure DevOps 31

2.5 Visual Studio App Center 33

3 Návrh webového aplikačního rozhraní a mobilní aplikace **34**

3.1 Koncept aplikace 34

3.2 Popis databáze 35

3.3 Návrh webového rozhraní 38

3.3.1 Funkční požadavky 38

3.3.2 Nefunkční požadavky 38

3.3.3 Zásadní koncové body 39

3.4 Návrh mobilní aplikace 40

3.4.1 Funkční požadavky 40

3.4.2 Nefunkční požadavky 41

3.4.3 Návrh uživatelského prostředí 42

4 Implementace webového aplikačního rozhraní a mobilní aplikace **44**

4.1 Vývoj webového aplikačního rozhraní 44

4.1.1 Projekt NNDIP.Api 44

4.1.2 Kontejnerizace aplikace 50

4.2 Vývoj mobilní aplikace 52

4.2.1 Projekt NNDIP.ApiClient 52

4.2.2 Projekt NNDIP.Maui 53

4.2.3 Výsledné uživatelské prostředí aplikace 57

5 Testování a nasazení mobilní aplikace **58**

5.1 Testování aplikace 58

5.2 Nasazení aplikace do oficiálních mobilních obchodů 59

5.2.1 Nasazení aplikace do Google Play....59
5.2.2 Nasazení aplikace do App Store....60
Závěr....61
Použitá literatura....63
Přílohy....66

SEZNAM OBRÁZKŮ

Obrázek 1: Architektura IoT 15
Obrázek 2: Koncept chytrého domu 18
Obrázek 3: Multiplatformní framework .NET MAUI 24
Obrázek 4: Schéma architektury .NET MAUI 25
Obrázek 5: Nejpoužívanější rozvržení v .NET MAUI 26
Obrázek 6: Architektura využívající obslužné třídy v .NET MAUI 27
Obrázek 7: Porovnání vývoje softwaru s nástrojem Docker a bez 29
Obrázek 8: Porovnání nasazení softwaru s nástrojem Docker a bez 30
Obrázek 9: Docker image a kontejnery 31
Obrázek 10: Koncept DevOps 32
Obrázek 11: Koncept aplikace 34
Obrázek 12: Schéma fyzického řešení systému 35
Obrázek 13: Diagram tabulky *ADDRESS_STATE* 36
Obrázek 14: Diagram tabulek *DATA* a *SENSOR* 36
Obrázek 15: Diagram databázových tabulek souvisejících s tabulkou *PLAN* 37
Obrázek 16: Obrazovky mobilní aplikace 42
Obrázek 17: Příklad návrhu obrazovek aplikace 43
Obrázek 18: Struktura projektu *NNDIP.Api* 44
Obrázek 19: Zdrojový kód třídy *LimitPlanSettings* 46
Obrázek 20: Rozhraní *IGenericRepository<T>* 47
Obrázek 21: Část zdrojového kódu třídy *DataController* 48
Obrázek 22: Metoda *GetActualSensorData* ve třídě *DataController* 48
Obrázek 23: Návrhový vzor repository 49
Obrázek 24: Ukázka Swagger UI 50
Obrázek 25: Dockerfile aplikace *NNDIP.Api* 51
Obrázek 26: Uživatelské prostředí programu NswagStudio 52
Obrázek 27: Struktura projektu *NNDIP.ApiClient* 53
Obrázek 28: Struktura projektu *NNDIP.Maui* 54
Obrázek 29: Dependency injection ve třídě *LoginPage* 55
Obrázek 30: Použití direktivy preprocesoru pro platformě závislý kód 55
Obrázek 31: Konečný vzhled aplikace pro operační systém Android 57

SEZNAM TABULEK

Tabulka 1: Historický přehled verzí .NET.....21

SEZNAM ZKRATEK

API	Application Programming Interface
BCL	Base Class Library
CIL	Common Intermediate Language
CLR	Common Language Runtime
CSS	Cascading Style Sheets
CTS	Common Type System
DDoS	Distributed Denial of Service
DI	Dependency injection
DoS	Denial of Service
DTO	Data Transfer Object
HTTP	Hypertext Transfer Protocol
HVAC	Heating, Ventilating, Air-conditioning
IIoT	Industrial Internet of Things
IoT	Internet of Things
MVVM	Model-View-ViewModel
ORM	Object-Relational Mapping
RFID	Radio Frequency Identification
SQL	Structured Query Language
UI	User Interface
URI	Uniform Resource Identifier

ÚVOD

V posledních letech se čím dál více využívají moderní technologie pro ovládání a monitorování nejrůznějších zařízení. Je před námi čtvrtá průmyslová revoluce, která bývá označována pod názvem Průmysl 4.0. Tento pojem zahrnuje výrobní procesy, které využívají nejnovější technologie s cílem dosáhnout vyšší efektivity, spolehlivosti a flexibility. Soustředí se také na digitalizaci a propojení těchto procesů, což umožňuje výrobcům rychleji reagovat na měnící se tržní podmínky. Využívání technologií, jako je například robotika, internet věcí, umělá inteligence nebo automatizace, může podnikům zvýšit kvalitu výrobku a výrazně snížit náklady na jeho výrobu.

Tyto moderní technologie se ale netýkají pouze velkých podniků. Dnes je již běžné, že je téměř každá moderní domácnost využívá pro své ovládání. Koncept Smart Home, nebo také chytrý dům, zahrnuje domy a byty, které obsahují nejrůznější zařízení, která komunikují mezi sebou a s uživateli a poskytují tak bezpečnější a pohodlnější bydlení. Komunikace může probíhat drátově nebo bezdrátově s využitím internetu nebo pomocí různých komunikačních standardů. Díky tomu lze snadno ovládat například zabezpečení, audio systémy, kuchyňské spotřebiče, okna a rolety, osvětlení, topení nebo klimatizaci. Monitorování spotřeby elektrické energie pomocí chytrých zásuvek nebo teploty pomocí tepelných senzorů a následné ovládání topení na základě naměřené teploty v místnosti umožňuje získat kontrolu nad spotřebou energií v domácnosti a snížit tak životní náklady.

Ovládání chytrého domu a jeho zařízení může probíhat pomocí tlačítek, hlasových pokynů, gest nebo chytrého telefonu či tabletu. Pro zobrazování dat a správu zařízení se typicky využívá mobilní aplikace. Chytrý mobilní telefon již standardně vlastní většina lidí a také ho má neustále po ruce, jeho použití pro tyto účely je tedy pohodlné a praktické, umožňuje uživateli rychle reagovat na jakékoli události kdykoli a odkudkoli. U mobilních aplikací této kategorie se dbá především na přehledné a jednoduché zobrazení zásadních údajů a základní ovládání zařízení.

Cílem této diplomové práce je vytvořit mobilní aplikaci, která primárně umožní uživateli vzdálenou správu systému HVAC zajišťující topení, chlazení a provětrávání objektu rodinného domu. Dále bude možné sledovat naměřené hodnoty teploty, vlhkosti a obsahu oxidu uhličitého v ovzduší. Aplikace bude také nabízet základní konfiguraci plánů a režimů domácnosti. Nad existujícím systémem HVAC bude nutné vytvořit aplikační rozhraní, které umožní mobilní aplikaci volat vybrané funkce systému. Hlavní motivací pro implementaci

aplikace je možnost ovládání již existujícího modelu chytrého domu z mobilního zařízení a zjednodušit tak zobrazování dat a nastavování hodnot.

1 INTERNET VĚCÍ (IOT)

Internet věcí (IoT – Internet of Things) představuje moderní a rychle rozvíjející se oblast technologií, která propojuje fyzická zařízení a umožňuje jejich komunikaci prostřednictvím internetu nebo vlastní sítě. Tento pojem poprvé představil Kevin Ashton v roce 1999 a má počátky v dřívějších technologiích, jako jsou senzorické sítě a vestavěné systémy. Následující kapitoly se budou věnovat bližší definici tohoto pojmu, technologiím a využití. [2], [3]

1.1 Definice a charakteristika

McEwan v [1] definuje Internet of Things jako síť fyzických zařízení se specifickými vlastnostmi. Tato zařízení pro komunikaci s okolím nebo mezi sebou využívají internet. Pro snímání dat ze svého okolí využívají různé snímače (senzory) a typicky je odesílají pro uchovávání a zpracování internetem. Zařízení, která v síti zastávají roli aktivního člena (aktuátoru), reagují na datové informace a transformují je na mechanický stav.

Zařízení si ale zároveň stále zachovává svoji původní formu a účel. Chytrý deštník, který indikuje nadcházející déšť, je stále využíván pro ochranu před deštěm nebo pračka, která nabízí ovládání přes mobilní aplikaci a dávkování prostředků dle potřeby, ale jejíž hlavní funkcionalita stále spočívá ve vyprání prádla. [1]

Dimitrios v [2] představuje koncept IoT systémů, který poskytuje přesnější popis této technologie než původní termín IoT. Většina zařízení je totiž propojena do větších systémů pro konkrétní účely, než aby tato zařízení vystupovala jako samostatné jednotky pro obecný přístup v rámci sítě internet. Systém se skládá primárně ze senzorů, případně obsahuje i několik aktuátorů.

Aby bylo možné splnit požadavky na provoz IoT systémů, musí být jejich spotřeba energie velice nízká, jelikož hraje klíčovou roli v celkových provozních nákladech těchto systémů. Pro dosažení požadované spotřeby energie je nutné věnovat pozornost návrhu hardwaru, softwaru a algoritmům aplikace. Další důležitou a sledovanou vlastností je zabezpečení. Bezpečnost systémů IoT je menší ve srovnání se systémy Windows, Mac nebo Linux, tyto problémy pramení z používání výchozích hesel, nedostatečnými bezpečnostními funkcemi u hardwaru nebo ve špatném návrhu softwaru. Nezabezpečená zařízení představují riziko nejen pro samotný IoT systém, ale i pro zbytek internetu, jelikož tato zařízení využívají útočníci při útocích typu DoS nebo DDoS. Při návrhu a provozu IoT systému je tedy nutné dbát na dostatečné zabezpečení. [2]

1.2 Architektura a technologie IoT

Technologie umožňující využití IoT se liší v závislosti na použití. Pro realizaci inteligentní dopravy je nutno aplikovat flexibilní technologie, které zajišťují propojení velkého počtu uzlů. Naopak ve zdravotnictví je hlavním kritériem pro výběr technologií spolehlivost a bezpečnost. Hassan v [8] uvádí pětivrstvý model architektury internetu věcí, který je zobrazen na obrázku č. 1. Model rozděluje technologie na základě jejich funkce.

Obrázek 1: Architektura IoT, vlastní zpracování dle [8]

Percepční vrstva obsahuje fyzická zařízení, která zajišťují sběr dat, případně na tyto informace reagují. Důležitým kritériem této vrstvy je spotřeba energie a schopnost komunikace (jednosměrná nebo obousměrná). Zařízení percepční vrstvy se mohou dělit na:

- Pasivní – podporují pouze jednosměrnou komunikaci a nepotřebují žádný zdroj energie, např. QR kódy, RFID čipy.
- Semi-pasivní – většinou obsahují malou baterii, aby mohli přijímat data od uživatele, např. semi-pasivní RFID čipy, infračervené snímače.
- Aktivní – umožňují přijímat a odesílat data, obsahují zdroj energie, případně jsou k němu připojeni, např. chytré aktuátory, nositelné, vestavěné nebo samostatné snímače. [8]

Síťová vrstva se skládá z technologií zajišťujících přenos dat. Umožňuje, aby zařízení z první vrstvy byla identifikovatelná v síti a mohla spolu komunikovat. Mnoho moderních zařízení využívá více než jednu technologii. Příkladem mohou být chytré hodinky, které využívají typicky Bluetooth, Wi-Fi a NFC. Dalším příkladem je otevřený protokol Z-Wave pro bezdrátovou komunikaci, který se používá pro vytvoření sítě pro zařízení v chytré domácnosti. Zařízení se do této sítě připojí a komunikují spolu pomocí rádiových frekvencí. Alternativou k Z-Wave je komunikační protokol Zigbee. [7], [8]

Servisní vrstva nebo také middleware vrstva slouží pro zpracování přijatých dat a poskytování požadovaných služeb, ke kterým uživatel přistupuje ve vyšších vrstvách. Pro ukládání dat jsou většinou využívány cloudové služby. [8]

Čtvrtá vrstva, aplikační, je zodpovědná za poskytování požadovaných služeb prostřednictvím jednoduchého rozhraní. Uživatelé IoT přistupují ke službě (např. nastavení teploty) pomocí mobilních telefonů nebo notebooků přes webové nebo mobilní aplikace. [8]

Provozní vrstva je využívána správci zejména pro návrh, analýzu, implementaci a monitorování systémů IoT. V této vrstvě lze přistupovat k vývojovým diagramům, grafům a obchodním modelům. Pro analýzu velkého objemu dat se využívá např. open-source technologie Apache Spark nebo Apache Apex. [8]

1.3 Oblasti využití

Velkou oblastí, ve které se IoT využívá, je průmyslové odvětví, kde se označuje jako IIoT (Industrial Internet of Things). Koncept IIoT zahrnuje všechny aspekty průmyslového provozu a zaměřuje se nejen na efektivitu procesů, ale také na správu aktiv a údržbu. Ačkoli základní charakteristiky jsou stejné, IIoT oproti IoT má větší požadavky na provozní technologie, neustálý provoz a bezpečnost, jelikož výpadek zařízení nebo narušení bezpečnosti s sebou může nést nejen zvýšené náklady, ale i ohrožení zdraví. [3]

Společnost Dundee Precious Metals, která působí v hornickém průmyslu, hledala nové možnosti, jak může zvýšit produktivitu, bezpečnost a provozní životnost ve svém předmětu podnikání. Byl vytvořen IIoT projekt, který měl tyto cíle společnosti naplnit. Jednou z největších výzev v tomto projektu byla realizace komunikace mezi zařízeními a senzory v dolech, protože standardní řešení využívající Wi-Fi by zde nefungovalo. S využitím Cisco technologií byla pro tyto účely vytvořena vlastní bezdrátová IP síť. IIoT projekt obsahoval mj. využití RFID čipů, které byly vloženy do vozidel a helem těžařů, což zvyšuje jejich

bezpečnost. Před realizováním změn byl cíl nastaven na 30% zvýšení produktivity, po realizaci projektu společnost dosáhla zvýšení produktivity o 400 %. [4]

Další oblastí, ve které se technologie IoT využívá, je zdravotnictví. Používá se při detekcích zdravotních problémů, poskytování pohotovostních služeb pacientům, nouzovém hlášení nebo při rehabilitacích. Pro snímání dat jsou využívány senzory, které jsou připojeny k pacientovi. Naměřená data jsou posílána do řídícího zařízení, typicky mobilní telefon, které je následně předá do centrální jednotky pro monitorování zdravotního stavu. Pacientův stav tak může být vzdáleně kontrolován. Při využití IoT ve zdravotnictví je nutno dbát zejména na bezpečnost, jelikož se pracuje s citlivými údaji. [5]

Mezi další významnou oblast, která využívá IoT, jsou tzv. chytrá města (Smart Cities). Tato města využívají komunikační a informační technologie pro své fungování za účelem zlepšení kvality života svých obyvatelů. Pelton v [6] udává několik klíčových vlastností, které by chytré město mělo splňovat:

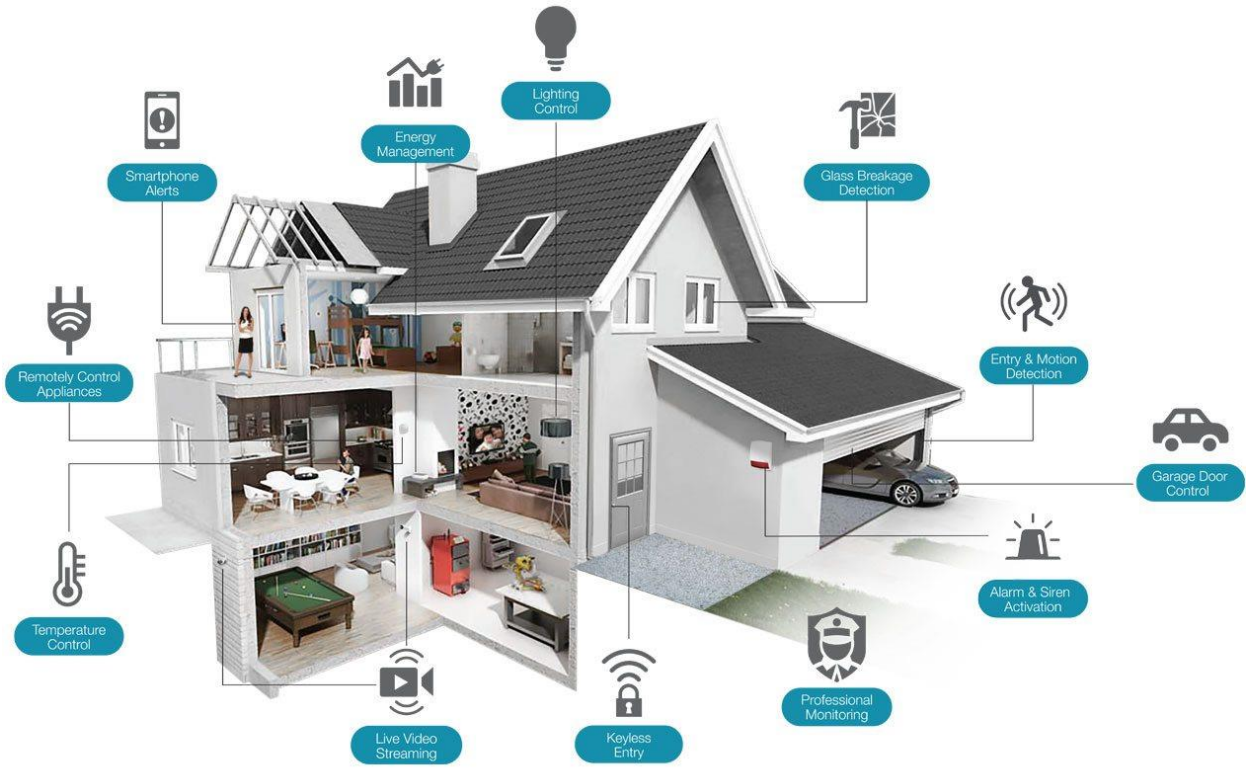
- naplnění potřeb obyvatelů a podniků v oblasti vzdělávání, zdravotnictví, bydlení a dopravy,
- vytvoření enviromentální udržitelnosti a oběhového hospodářství,
- zajištění dostatečného množství pracovních míst a konkurenceschopnosti,
- zapojení obyvatelstva do chytrého plánování,
- vylepšení infrastruktury a zdrojů,
- využití technologií a umělé inteligence pro podporu potřeb obyvatelů,
- zajištění dostatečné bezpečnosti.

Chytrá města nabízí ale svým obyvatelům nejen výhody, jako je například menší spotřeba energie, efektivnější dopravní infrastruktura nebo nízké znečištění ovzduší, ale přináší s sebou i několik nevýhod v podobě rizika kybernetických útoků, snížené míry soukromí nebo nízkou míru interakce s ostatními lidmi. [6]

V neposlední řadě se IoT hojně využívá v chytrých domácnostech (Smart Homes). Diplomová práce se zaměřuje na využití technologií právě v této oblasti, proto se následující kapitola podrobněji věnuje chytrým domácnostem a nabízí přehled nejrůznějších zařízení, která se v nich používají.

1.3.1 Smart Homes

Koncept chytré domácnosti, ve které lze mnoho rutinních věcí automatizovat a ve které lze její zařízení ovládat vzdáleně pomocí hlasu, gest nebo aplikace, je nyní dostupný většině běžných domácnosti i bez hlubokých technických znalostí. Mnoho výrobců nabízí široký výběr zařízení se snadnou konfigurací a integrací včetně návodů a tipů pro vytvoření chytré domácnosti. Většinu běžných zařízení lze ovládat s využitím již existujících aplikací, např. Google Home od společnosti Google nebo HomeKit od společnosti Apple. Pro komplexnější řešení chytrého domu lze využít služeb specializovaných firem. Obrázek č. 2 zobrazuje, v jakých oblastech lze využít chytré technologie např. při dálkovém ovládání spotřebičů, osvětlení nebo při správě vytápění. [7]



Obrázek 2: Koncept chytrého domu [10]

Pro plné využití potenciálu chytré domácnosti je potřeba mít k dispozici řadu prvků:

- chytrá domácí zařízení,
- digitální hlasové asistenty,
- aplikaci v telefonu či tabletu a online přístup. [7]

Typickým využitím chytrých domácích zařízení, zde konkrétně žárovek, jsou systémy osvětlení. Jednotlivá světla, případně skupiny světel lze ovládat přes aplikaci, dálkové

ovládání nebo hlasového asistenta. Aplikace umožňují řadu nastavení osvětlení od barvy až po intenzitu. [7]

Mezi další systémy, které chytrý dům obsahuje, je inteligentní systém vytápění, který umožňuje řízení topení a regulaci teploty s využitím chytrých zařízení. Typicky se ovládá přes mobilní aplikaci, ve které může uživatel spravovat nejen teplotu, ale i režimy nebo časové plány pro vytápění. Pokročilejší systémy navíc nabízí automatickou regulaci teploty v závislosti na tom, kolik osob se v dané místnosti nachází, což přináší maximální komfort a pohodlí pro obyvatele domu. Díky přesnému řízení teploty lze také snížit spotřebu energie a tím i náklady na vytápění. Chytré termostaty a regulátory teploty mohou také analyzovat spotřebu energie a poskytovat informace o tom, jakým způsobem lze ušetřit. [7]

Mnoho chytrých domů využívá chytré systémy zabezpečení, které zajišťují monitorování a kontrolu bezpečnosti domu pomocí chytrých zařízení. Kamery a senzory umožňují sledovat situaci v domě v reálném čase a notifikovat uživatele o neobvyklých událostech, případně na ně reagovat např. spuštěním sirény. Sofistikovanější kamery obsahují technologie pro rozpoznávání obličejů, tudíž lze nastavit ignorování např. členů rodiny. Mezi další zařízení, která se využívají v bezpečnostních systémech jsou chytré zámky využívající např. biometrické ověření nebo chytré zvonky. Inteligentní zabezpečovací systém přináší uživateli vyšší míru zabezpečení jeho majetku a pocit bezpečí. [7]

Chytré technologie pronikly již i do těch nejobyčejnějších spotřebičů v domě. Není výjimkou vlastnit robotický vysavač, který dokáže zmapovat celý dům a následně ho uklízet dle potřeby. Chytré lednice pak například nabízí monitorování potravin a jejich doby spotřeby. Mezi další chytré spotřebiče patří varné konvice, pračky nebo myčky na nádobí. Pokud nyní neexistuje chytrá verze nějakého domácího spotřebiče, pravděpodobně se brzy na trhu objeví. [7]

Většinu zařízení lze ovládat pomocí hlasového asistenta, který je součástí chytrého reproduktoru. Hlasovému asistentovi uživatel řekne příkaz a on na něj patřičně zareaguje, např. rozsvítí světla nebo zapne hudbu. Mezi hlavní chytré reproduktory s asistentem na trhu patří:

- Amazon Echo Dot s hlasovým asistentem Alexa,
- Google Home a Google Home Mini s hlasovým asistentem Google Assistant,
- Apple HomePod s hlasovým asistentem Siri. [7]

Ačkoli chytrá domácnost má mnoho výhod, přináší s sebou i úskalí v podobě ztráty osobního kontaktu nebo negativního dopadu na soukromí uživatele. V moderní době, kdy jsou lidé stále více spojeni přes sociální sítě a elektronické komunikace, se osobní kontakt stává stále vzácnějším zážitkem. Chytrá domácnost může tuto situaci ještě zhoršit, protože způsobuje, že lidé tráví více času v uzavřeném prostoru, kde svůj čas věnují svým telefonům, tabletům nebo počítačům, a méně času tráví interakcí s ostatními lidmi. Dlouhodobé používání zařízení může vést k závislosti na internetu a k rozvoji sociální úzkosti. Chytré domácnosti navíc sbírají mnoho dat o uživateli a je zde tedy riziko jejich sdílení se třetími stranami. Tyto údaje pak mohou být využity např. k cílené reklamě. Kromě toho chytré reproduktory s virtuálními asistenty jsou navrženy tak, aby neustále naslouchaly, což může vést k pocitu nedostatku soukromí. [7]

IoT technologie se v posledních letech čím dál více používají i v běžných domácnostech. Svým uživatelům nesporně přináší mnoho výhod v podobě ušetření energií nebo zvýšení kvality bydlení. Při používání těchto zařízení je ale důležité pamatovat na to, že byla navržena tak, aby sloužila uživatelům a ne naopak.

2 PŘEHLED TECHNOLOGIÍ

Pro zpracování praktické části diplomové práce byly použity různé technologie, které stručně popisuje tato kapitola. Celá práce využívá programovací jazyk C#. Serverová část je implementována pomocí frameworku ASP.NET Core a pro její kontejnerizaci byl využit software Docker. Pro vytvoření mobilní aplikace byla využita technologie .NET MAUI. Použité technologie byly vybrány s ohledem na jejich dlouhodobé využívání na trhu a multiplatformní podporu.

2.1 .NET

.NET je bezplatná multiplatformní sada technologií a nástrojů pro vytváření různých typů aplikací vyvíjená společností Microsoft. Její zdrojové kódy jsou volně přístupné na platformě GitHub v několika úložištích. Aplikace mohou být programované v jazyce C#, F# nebo Visual Basic. [11]

V roce 2002 byla uvedena platforma .NET Framework ve verzi 1.0 pro vývoj Windows aplikací. Její poslední verzí je verze 4.8, která zůstává plně podporovaná Microsoftem. Nástupcem .NET Framework je .NET Core, později pouze .NET. Jedná se multiplatformní soubor technologií pro vývoj aplikací nejen pro operační systémy Windows, ale i pro Linux nebo macOS. Microsoft plánuje vydávat nové verze .NET jednou ročně. Historický přehled verzí je uveden v tabulce č. 1. [11]

Tabulka 1: Historický přehled verzí .NET [12]

Verze	**Datum uvedení**
.NET Framework 1.0	únor 2002
.NET Framework 1.1	duben 2003
.NET Framework 2.0	listopad 2005
.NET Framework 3.0	listopad 2006
.NET Framework 3.5	listopad 2007
.NET Framework 4.0	duben 2010
.NET Framework 4.5	srpen 2012
.NET Framework 4.6	červenec 2015
.NET Framework 4.7	březen 2017
.NET Framework 4.8	květen 2018
.NET Core 3.0 a 3.1	září 2019
.NET 5.0	listopad 2020
.NET 6.0	listopad 2021
.NET 7.0	listopad 2022

Architektura technologií .NET se skládá z několika vrstev a komponent. Hlavní součástí je společný běhový systém Common Language Runtime (CLR), který zajišťuje běh programů

přeložených z různých programovacích jazyků do mezijazyka Common Intermediate Languge (CIL). CLR spravuje kód v době jeho vykonávání a poskytuje různé služby, jako je správa paměti, spouštění vláken, ověřování bezpečnosti kódu, kompilaci a další systémové služby. [13]

Dalším důležitým prvkem je společný systém typů, tzv. Common Type System (CTS), který umožňuje snadnou spolupráci mezi kódem napsaným v různých programovacích jazycích a určuje, jak jsou datové typy reprezentovány v paměti a jaké operace jsou s nimi dovoleny. CTS definuje pravidla pro dobu života objektů, dědění, viditelnost datových typů a jejich složek. [13]

Součástí .NET je dále knihovna tříd Base Class Library (BCL), která poskytuje základní datové typy a funkce pro práci s nimi. Obsahuje např. vstupní a výstupní operace nebo různé datové struktury. [13]

2.1.1 ASP.NET Core

Framework ASP.NET Core se využívá pro vytváření webových aplikací a služeb, webových rozhraní nebo aplikací pro zařízení IoT. Lze jej použít pro vývoj aplikací pro operační systémy Windows, macOS nebo Linux. ASP.NET Core nabízí mnoho moderních nástrojů a výhod pro vývoj aplikací, mezi které patří:

- snadná integrace moderních klientských frameworků,
- jednotné rozhraní pro vývoj webového UI a API,
- podpora návrhového vzoru dependency injection,
- systém připravený pro nasazení do cloudové prostředí,
- nástroje pro snadné testování. [14]

Jelikož se diplomová práce zabývá vytvořením webového rozhraní, budou se následující odstavce zaměřovat na oblast vývoje API. ASP.NET Core vytváří API jako webovou službu implementující standardy REST. REST je architektura, která umožňuje přístup k datům prostřednictvím standardních metod HTTP. Při vytváření webového rozhraní pomocí frameworku ASP.NET Core se typicky vytváří třídy, tzv. kontroléry, které dědí z třídy *ControllerBase*. Kontroléry přebírají potřebné závislostí s využitím návrhového vzoru dependency injection (DI) a obecně dodržují pravidla objektově-orientovaného programování. Ve třídě se definují koncové body a jejich přístupové metody. Chování tříd a metod lze ovlivnit pomocí různých anotací. Alternativním řešením pro vytváření minimalistických API je využití specifických metod a definování obslužného kódu pomocí lambda výrazů. [15]

Při vývoji webových rozhraní se často využívají některé nástroje Swagger. Jedná se o volně dostupnou sadu nástrojů pro návrh, tvorbu, dokumentaci a konzumaci REST API na základě OpenAPI specifikace. Mezi hlavní nástroje patří:

- Swagger Editor – webový editor pro psaní OpenAPI definicí,
- Swagger UI – dle OpenAPI specifikace vytváří interaktivní dokumentaci,
- Swagger Codegen – umožňuje generovat zdrojové kódy v různých jazycích na základě OpenAPI definice. [16]

OpenAPI specifikace (OAS) definuje formální popis pro REST API. Pomocí ní lze popsat podrobně vlastnosti webového rozhraní – dostupné koncové body a metody, parametry a návratové hodnoty operací nebo bezpečnostní mechanismy. Specifikace umožňuje popis ve formátu YAML nebo JSON. Mezi dvě hlavní implementace OpenAPI pro .NET patří Swashbuckle a NSwag. [16], [27]

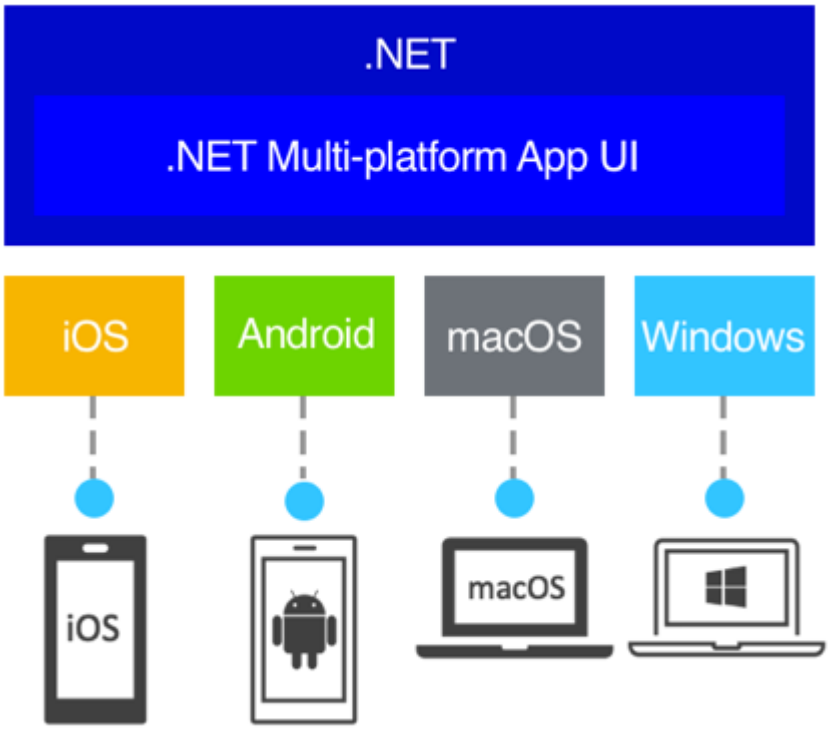
Framework ASP.NET Core umožňuje využít pro práci s databází technologii Entity Framework Core (EF Core). Framework je vyvíjen společností Microsoft a je dostupný pro platformy Windows, Linux a macOS. Vývojáři umožňuje s databází komunikovat pomocí objektového přístupu, jedná se o tzv. objektově-relační mapování (ORM). Databázové tabulky a jejich sloupce jsou mapované do klasických tříd a atributů. Při práci s těmito třídami si framework sám generuje a provádí SQL dotazy s využitím objektu třídy *DbContext*, který poskytuje abstrakci mezi EF Core a doménovým modelem. *DbContext* představuje hlavní část EF Core a reprezentuje jednu databázovou relaci, která je využívána pro realizaci SQL dotazů. Pro použití EF Core v aplikaci je nutná instalace NuGet balíčku podporujícího práci s konkrétní databází. [28], [30]

Existují dva přístupy pro využití této technologie. První přístup, Code First, znamená nejprve vytvoření tříd a poté vygenerování odpovídajících databázových tabulek pomocí EF Core. Druhý přístup, Database First, se využívá, pokud již existují databázové tabulky. V aplikaci je pak nutné vytvořit jejich odpovídající třídy. EF Core nabízí příkazem *Scaffold-DbContext* automatické vytvoření databázového kontextu a potřebných entit na základě daného databázového schématu. Tento přístup byl využit v praktické části této diplomové práce. [28], [29]

2.2 .NET MAUI

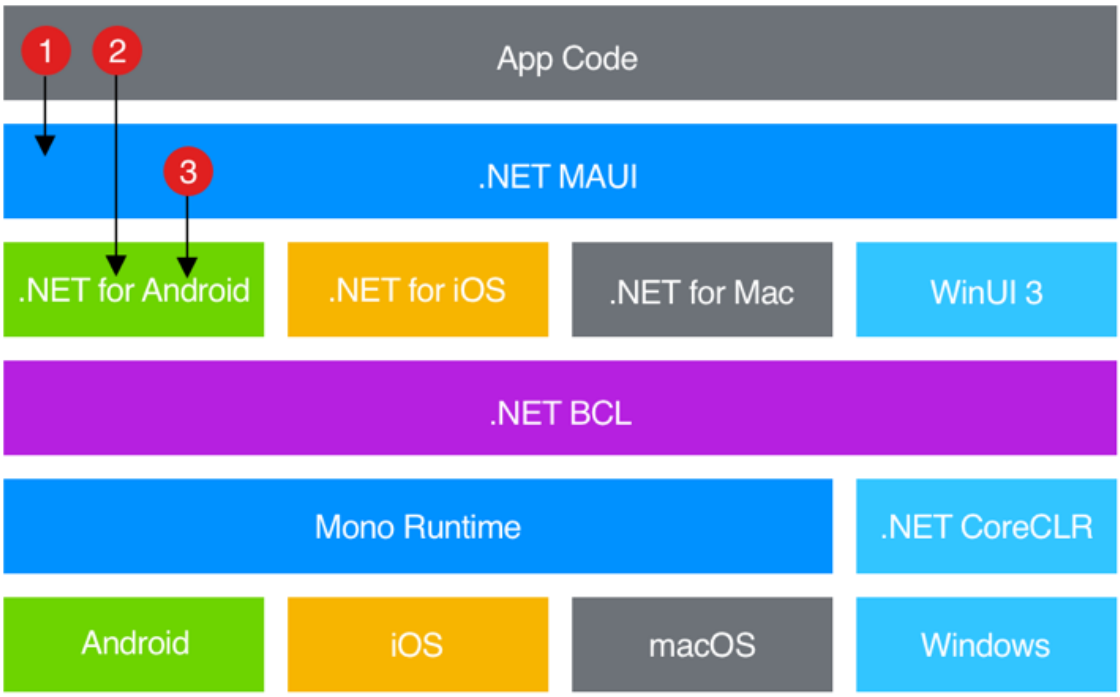
V květnu roku 2022 vydala společnost Microsoft novou technologii .NET MAUI (.NET Multi-platform App UI), která je součástí vývojářské platformy .NET od verze 6. Je to

multiplatformní framework pro vytváření mobilních a desktopových aplikací s využitím jazyka C# a XAML. Tato vývojová platforma je bezplatná s otevřeným a volně dostupným kódem. Zaměřuje se zejména na schopnost sjednotit strukturu aplikace pro zařízení s různým operačním systémem. Pomocí .NET MAUI lze vytvářet aplikace pro zařízení s operačním systémem Android, iOS, macOS nebo Windows, což schematicky zobrazuje obrázek č. 3. [17]



Obrázek 3: Multiplatformní framework .NET MAUI [17]

Hlavní výhodou .NET MAUI je sdílení zdrojového kódu mezi cílovými platformami. Při programování aplikace lze psát kód, který využívá rozhraní .NET MAUI (č. 1 na obrázku č. 4). Toto rozhraní slouží pro zastřešení tříd a datových struktur jednotlivých platforem (č. 3 na obrázku č. 4). V případě potřeby může vývojář psát kód, který přímo využívá rozhraní platformy (č. 2 na obrázku č. 4). Všechna nativní rozhraní platforem pak mají přístup k základní knihovně .NET Base Class Library (BCL), která poskytuje běhové prostředí aplikace. Pro Android, iOS a macOS je využíváno prostředí Mono a pro Windows pak .NET CoreCLR. [17]



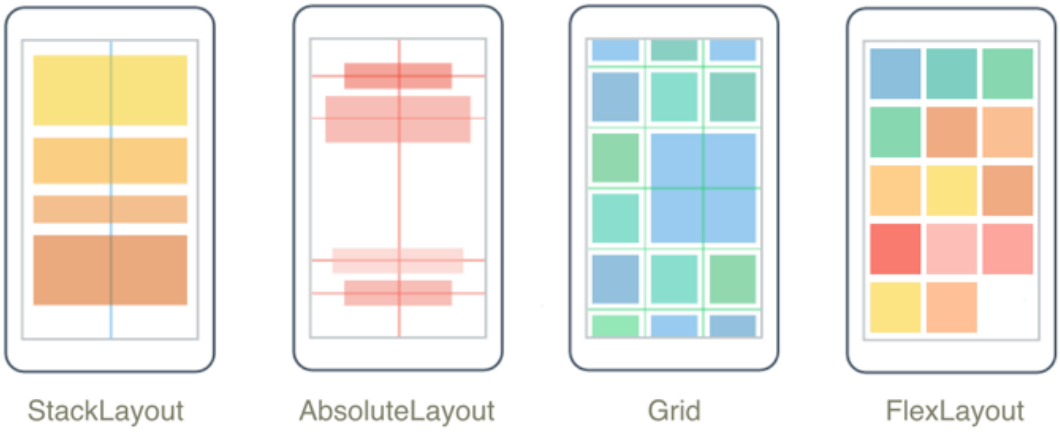
Obrázek 4: Schéma architektury .NET MAUI [17]

Ovládací prvky v .NET MAUI aplikaci lze rozdělit do třech základních skupin, a to:

- stránky (pages),
- rozvržení (layouts),
- grafické komponenty (views). [18]

Aplikace se obvykle skládá z jedné a více stránek, které obsahují jedno a více rozvržení. Nejpoužívanějšími typy rozvržení jsou:

- StackLayout – prvky jsou uspořádány za sebe v horizontálním nebo vertikálním směru,
- AbsoluteLayout – prvky jsou umístěny na specifické místo vzhledem k předkovi,
- GridLayout – prvky jsou uspořádány do mřížky,
- FlexLayou – prvky jsou řazeny do řádku nebo sloupce s možností zalomení. [19]



Obrázek 5: Nejpoužívanější rozvržení v .NET MAUI [19]

Rozvržení uspořádává grafické komponenty do specifické struktury, příklady jsou zobrazené na obrázku č. 5. Grafické komponenty zahrnují běžné prvky jako jsou např. tlačítka, vstupní, vyhledávací, datumové nebo zaškrtávací pole nebo tabulky. Vzhled komponent může být upraven přímo v XAML souboru nebo lze využít kaskádové styly (CSS). [19]

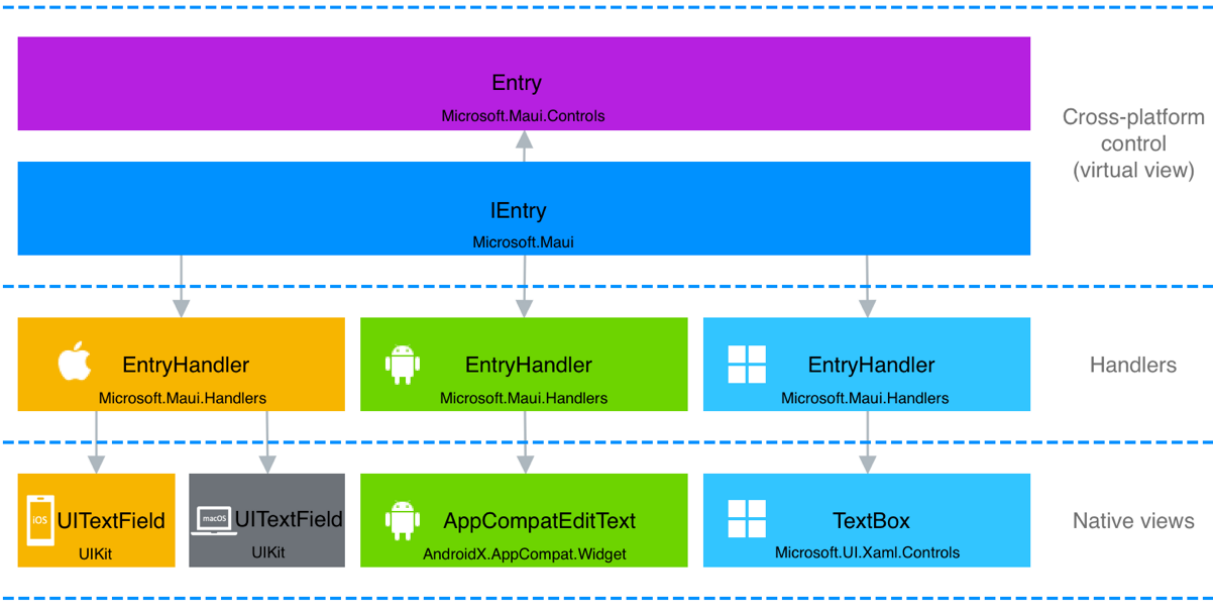
2.2.1 Xamarin.Forms a .NET MAUI

Framework .NET MAUI je nástupcem Xamarin.Forms, což je technologie od společnosti Microsoft pro vývoj multiplatformních mobilních aplikací, která umožňuje vývojáři sdílet až 90 % zdrojového kódu napříč platformami. Tyto dvě technologie mají mnoho společného, většinu funkcionalit a komponent, které nabízí Xamarin.Forms lze využít i v .NET MAUI projektu. Nicméně je zde několik zásadních rozdílů. [20]

Jedna z nejvýraznějších změn je struktura projektu. Xamarin.Forms vytváří pro každou platformu jeden projekt v rámci aplikace na rozdíl od .NET MAUI, kde je pro aplikaci vytvořen pouze jeden projekt. Tento projekt obsahuje složku *Platforms* s podsložkami reprezentující jednotlivé platformy, které slouží pro definování specifického zdrojového kódu. Tato zjednodušená struktura umožňuje programátorovi lepší orientaci v projektu a menší množství duplicitního kódu. [20]

Další změnou, která ovlivňuje rychlost aplikace, je vykreslování grafických komponent. Xamarin.Forms obsahuje vykreslovací architekturu, která zpomaluje běh aplikace a výrazně zvětšuje její velikost. Pokud je potřeba upravit vzhled grafické komponenty, je nutné si vytvořit i vlastní vykreslovač. V .NET MAUI je použita architektura s využitím obslužných tříd tzv. handlerů, které zvyšují výkon aplikace a zároveň snižují její velikost. Každý ovládací prvek implementuje příslušné rozhraní. Tyto prvky se nazývají virtuální pohledy, tzv. virtual

views. Obslužné třídy, handlery, pak mapují virtuální pohledy na nativní ovládací prvky jednotlivých platforem, tzv. nativní pohledy neboli native views. Příkladem je mapování multiplatformního ovládacího prvku *Entry* pomocí obslužné třídy *EntryHandler* na nativní ovládací prvek reprezentovaný třídou *UITextField* pro iOS nebo *AppCompatEditText* pro Android. Tato architektura je zobrazena na obrázku č. 6. [20], [23]



Obrázek 6: Architektura využívající obslužné třídy v .NET MAUI [21]

Dalším rozdílem je podpora okamžité aktualizace kódu a jeho následného spuštění bez nutnosti ručního restartování aplikace v .NET MAUI, tzv. hot reload. Xamarin.Forms tuto funkcionalitu nepodporoval, případně pouze v omezené míře v závislosti na verzi vývojového prostředí Microsoft Visual Studio. Možnost funkce hot reload usnadňuje a urychluje vývoj aplikací. [20]

V neposlední řadě .NET MAUI umožňuje vytvářet i webové aplikace s využitím frameworku Blazor, který slouží pro vytváření interaktivních uživatelských prostředí na straně klienta pomocí .NET. Pro sjednocení těchto dvou technologií se používá prvek BlazorWebView, který umožňuje zobrazit webové rozhraní v nativním prostředí .NET MAUI. Pro tvorbu grafického uživatelského prostředí se využívá HTML, CSS a C#. [22]

2.2.2 Kontroverze .NET MAUI

Mnoho vývojářů si stěžuje, že .NET MAUI obsahuje velké množství chyb a nedotažených funkcí a jeho vydání přišlo příliš brzy. Na sociálních sítích se objevovaly komentáře typu:

- „Zkouším MAUI a jsem zmaten obrovským množstvím chyb a problémů, se kterými se setkávám.“,
- „Tento framework potřeboval ještě nejméně rok vývoje před jeho oficiálním vydáním.“,
- „MAUI mi připomíná Xamarin Forms z roku 2014-2015, naprosto nepoužitelný.“
- „Býval jsem vývojář aplikací v Xamarinu a podle mého MAUI nebude stabilní ještě rok. Vytvářel jsem aplikaci, která používala MAUI a samotné Visual Studio je plné chyb v souvislosti s novým frameworkem. Vývoj byl příliš uspěchaný a nesoustředil se na kvalitu oproti kvantitě.“. [24]

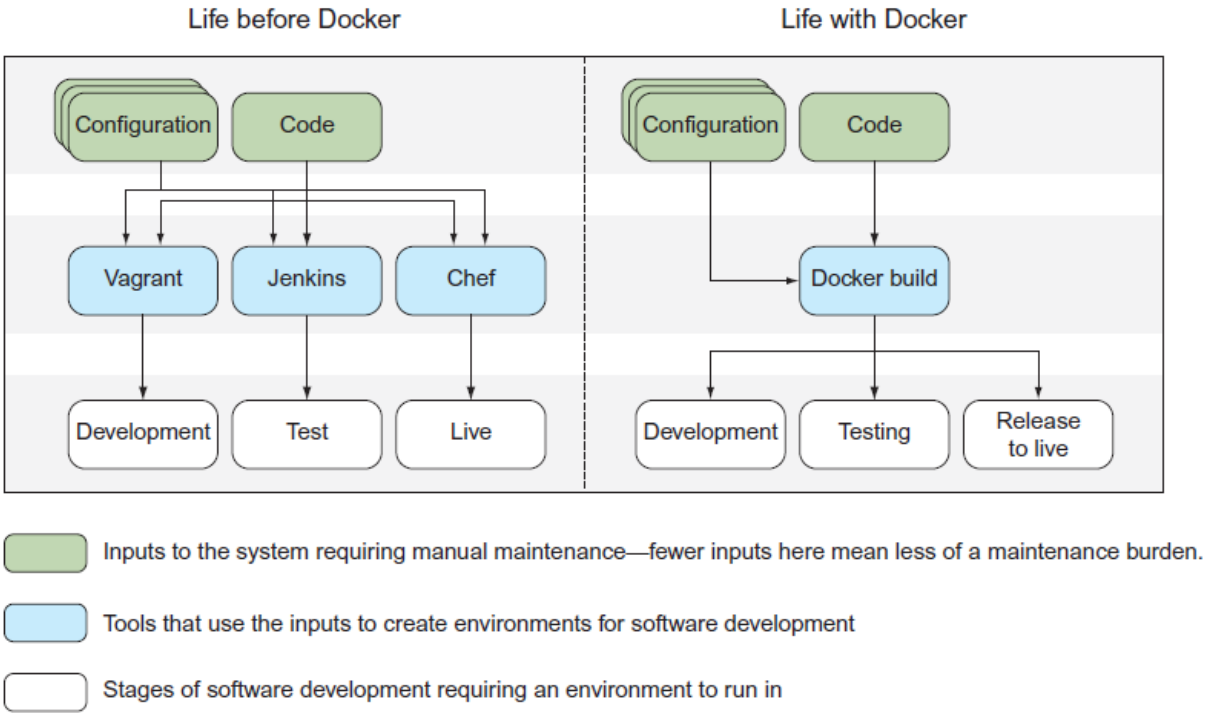
Na platformě GitHub, kde jsou zpřístupněny zdrojové kódy frameworku, bylo k datu 28. 3. 2023 evidováno 2168 otevřených případů a 5335 již uzavřených [25]. Vylepšení frameworku a rozsáhlejší oprava chyb byla realizována s příchodem frameworku .NET 7, který se soustředí zejména na zvýšení výkonnosti a rychlosti aplikací [26].

Na druhou stranu se ale objevují i pozitivní komentáře, které oceňují zejména úsilí, které společnost Microsoft vkládá do vývoje této technologie a věří v potenciál této technologie pro vývoj uživatelského prostředí. [24]

2.3 Docker

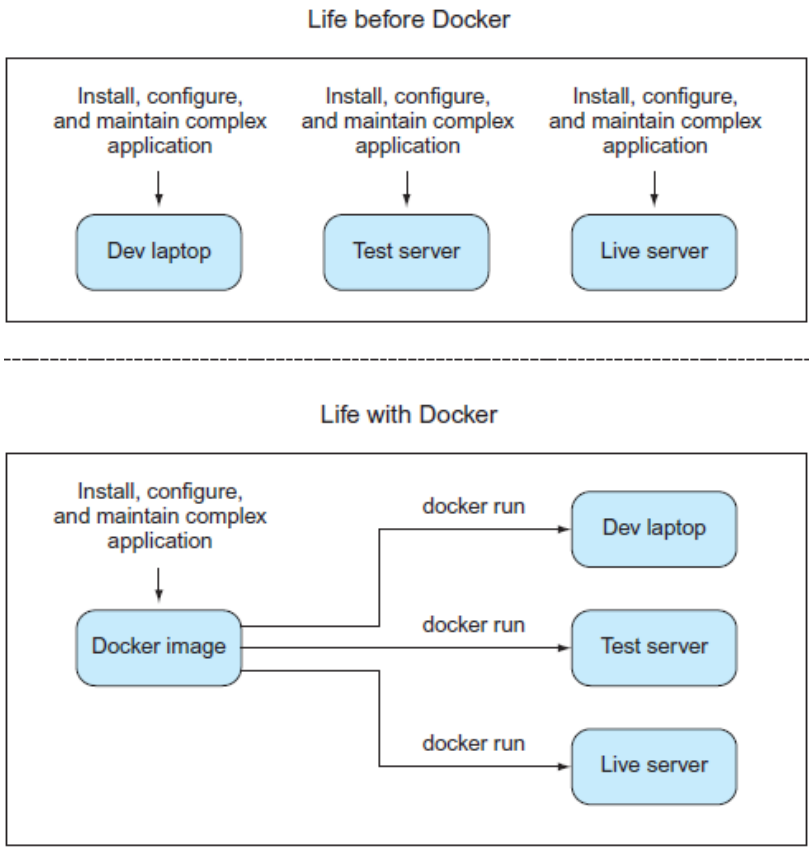
Docker je nástroj, který umožňuje snadné sestavení, distribuování a nasazení aplikací pomocí tzv. kontejnerů. V dnešní době je použití tohoto softwaru považováno za standardní řešení nákladné části vývoje softwaru, a to nasazení a distribuce softwaru. [9]

Před tím, než se Docker začal používat, bylo nutné při vývoji aplikací využívat různé nástroje pro nasazování a distribuování např. virtuální stroje, nástroje pro správu konfigurace a systémy pro správu balíčků. Udržování různých konfigurací a nástrojů může být časově i finančně náročné. Docker umožňuje pomocí jednotného jazyka vytvořit ze zdrojových souborů a konfigurací jeden výstup, který lze použít na jakémkoli zařízení. Toto porovnání je zobrazeno na obrázku č. 7, kde zelené položky představují vstupy v podobě zdrojového kódů a konfigurací, modré položky reprezentují nástroje, které pracují se vstupy, a nakonec bílé položky znázorňují fáze vývoje softwaru, které vyžadují prostředí pro běh nebo vývoj softwaru. [9]



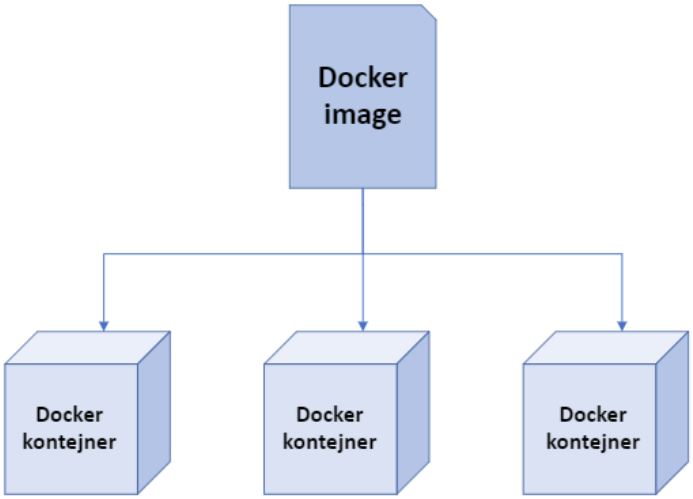
Obrázek 7: Porovnání vývoje softwaru s nástrojem Docker a bez [9]

Jedna z nejvýznamnějších výhod použití nástroje Docker je úspora času a nákladů při spouštění aplikace v různých prostředích. Obrázek č. 8 zobrazuje, že s využitím technologie Docker je instalování, konfigurování a udržování aplikace prováděno na jednom místě a výstupem je tzv. Docker image. Ten se distribuuje na různá prostředí a danou aplikaci pak lze snadno spustit. Naopak bez použití nástroje Docker je nutné v každém prostředí, ve kterém se aplikace spouští, provést instalaci a konfiguraci a pravidelně aplikaci udržovat. [9]



Obrázek 8: Porovnání nasazení softwaru s nástrojem Docker a bez [9]

Klíčovými pojmy při práci s technologií Docker jsou Docker image a kontejner. Docker image přestavuje zjednodušeně šablonu pro vytváření kontejnerů, tzn. kontejner je instancí Docker image. Skládá se ze souborů, které obsahují zdrojový kód, knihovny, frameworky a další potřebná data pro běh aplikace, a metadat. Docker image může vzniknout na základě souboru Dockerfile pomocí příkazu *docker build* nebo z již běžícího kontejneru pomocí příkazu *docker commit*. Z obrázku č. 9 je patrné, že z jednoho Docker image lze vytvořit několik na sobě nezávislých kontejnerů. Kontejnery tvoří prostředí pro běh dané aplikace se všemi potřebnými soubory. [9]



Obrázek 9: Docker image a kontejnery

Pro sdílení Docker image lze využít např. registr Docker Hub spravovaný společností Docker Inc. Uživatelé mohou stahovat Docker image, které jsou publikované jinými uživateli, a použít je ve svých aplikacích nebo mohou sdílet své vlastní. Platforma nabízí i soukromé repozitáře, což umožňuje možnost distribuovat Docker image pouze v rámci organizace nebo týmu. [9]

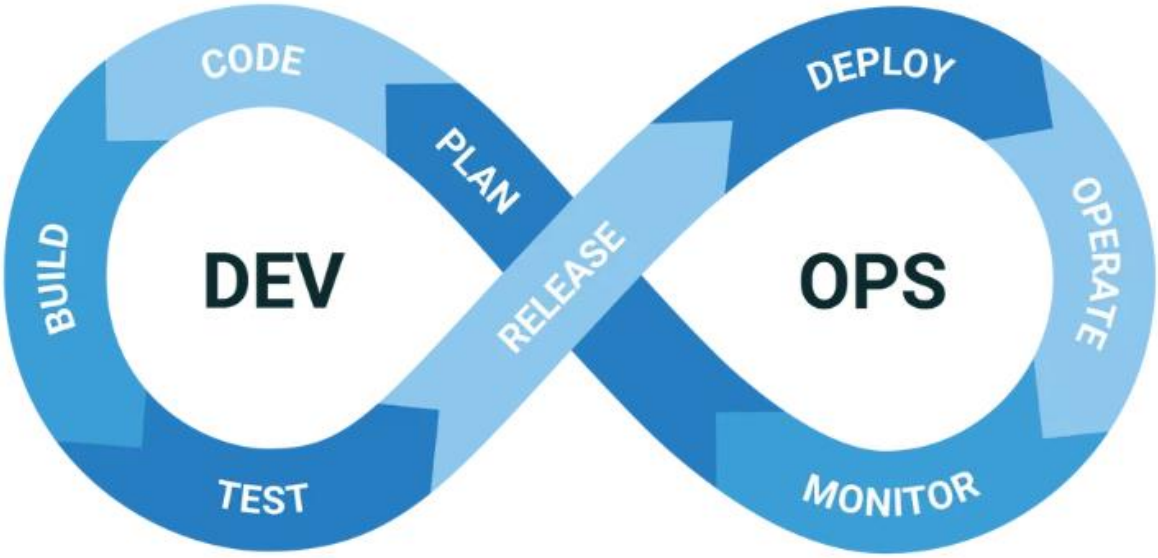
2.4 Azure DevOps

Dnes již hojně používaný pojem DevOps představuje složeninu anglických výrazů Development (vývoj) a Operations (provoz). Jedná se o koncepci vývoje softwaru, která se zaměřuje na sjednocení a automatizaci procesů vývoje, testování a nasazení aplikací. Sjednocuje vývojářský a provozní tým s cílem rychlejšího a efektivnějšího vývoje softwaru a zajištění jeho bezproblémového nasazení a provozu. [31]

DevOps ovlivňuje celý životní cyklus aplikace, od návrhu, přes vývoj a nasazení až po monitorování systému v provozu. V první části vývoje se DevOps zaměřuje na rychlou a efektivní tvorbu softwaru. Zdrojové kódy se nachází v centrálním úložišti (repozitáři) a jsou verzovány. Využívá se zde technika průběžné integrace (Continuous Integration, CI), která umožňuje automatizovat sestavení a testování aplikace. Změny v aplikaci jsou do repozitáře integrovány na denní bázi a každá tato integrace je ověřena automatickým sestavením aplikace a spuštěním testování, aby chyby byly objeveny co nejdříve a mohly být následně opraveny. [31]

Ve fázi nasazování se využívá průběžné nasazení (Continuous Delivery, CD). Cílem je umožnit nasazování nových verzí aplikace do produkčního prostředí s minimálními náklady a

riziky. Zahrnuje testování softwaru v různých prostředí pro minimalizaci chyb. Spolu s CI tvoří metodiku nazývanou souhrnně CI/CD. Koncept DevOps je zobrazen na obrázku č. 10. [31]



Obrázek 10: Koncept DevOps [32]

V roce 2018 Microsoft představil novou platformu Azure DevOps pro správu softwarového vývoje. Nabízí širokou sadu funkcí, která je dostupná přes webový prohlížeč nebo pomocí integrovaného vývojového prostředí, zahrnující:

- Git repozitáře pro správu zdrojového kódu,
- služby pro sestavení a distribuování aplikací (CI/CD),
- nástroje pro testování aplikací,
- agilní nástroje pro podporu plánování a sledování práce na projektu. [31]

Azure DevOps nabízí širokou škálu služeb a nástrojů pro správu projektů a procesů, které jsou vzájemně propojeny a umožňují tak snadnou a rychlou spolupráci týmu. Mezi nejdůležitější patří:

- Azure Boards – nástroj pro plánování a správu projektů, který umožňuje plánovat a řídit iterace, sledovat postup práce a výkonnost projektu.
- Azure Repos – nástroj pro správu zdrojového kódu, který umožňuje verzování, správu větví a řešení konfliktů v souborech.
- Azure Pipelines – nástroj pro automatizované sestavování, testování a nasazování aplikace (CI/CD).

- Azure Test Plans – nástroj pro testování, který umožňuje vytvářet a spravovat testovací plány a sledovat výsledky testů.
- Azure Artifacts – nástroj pro správu balíčků a artefaktů. [31]

Prostřednictvím projektového portálu, který je dostupný přes webový prohlížeč, je možné zlepšit spolupráci mezi všemi zúčastněnými stranami, jelikož Azure DevOps podporuje různé role zahrnující vývojáře, testery, architekty a další. Azure DevOps navíc nabízí také vytváření vlastních doplňků, případně integraci dalších nástrojů, např. Microsoft Excel pro plánování projektu. Platforma tak nabízí řešení pro zavedení kvalitního, automatizovaného a robustního DevOps procesu. [31]

2.5 Visual Studio App Center

Visual Studio App Center je cloudová platforma používaná při vývoji mobilních a desktopových aplikací, která poskytuje vývojářům kompletní sadu nástrojů pro sestavování, testování a nasazování aplikací a je dostupná přes webový prohlížeč. Platforma nabízí uživateli pět služeb. [33]

Služba pro sestavování (Build) umožňuje sestavování aplikací pro Android, iOS, macOS a UWP. Aplikaci lze sestavit z Git repozitáře umístěného na platformě Azure DevOps, Bitbucket, GitHub nebo GitLab. Služba pro testování (Test) slouží pro automatizované testování mobilních aplikací na různých zařízení a operačních systémech. Platforma App Center ukládá výsledky testů po dobu až šesti měsíců. Nástroj pro distribuování (Distribute) slouží pro snadné a rychlé nasazení aplikace do koncových uživatelských zařízení a umožňuje řízení a správu verzí. Služba podporuje Android, iOS, macOS, UWP, WPF a WinForms aplikace. Službu lze využít jak při vývoji, kdy každému členu v týmu přijde notifikace o vydání nové verze a uživatel si ji musí nainstalovat, nebo při publikování aplikace např. do obchodu Google Play nebo Apple App Store. Služba pro monitorování aplikace (Diagnostics a Analytics) umožňuje sledovat a analyzovat chování uživatelů v aplikaci, sbírat data o využívání aplikace a poskytovat zpětnou vazbu pro vývojáře. [33]

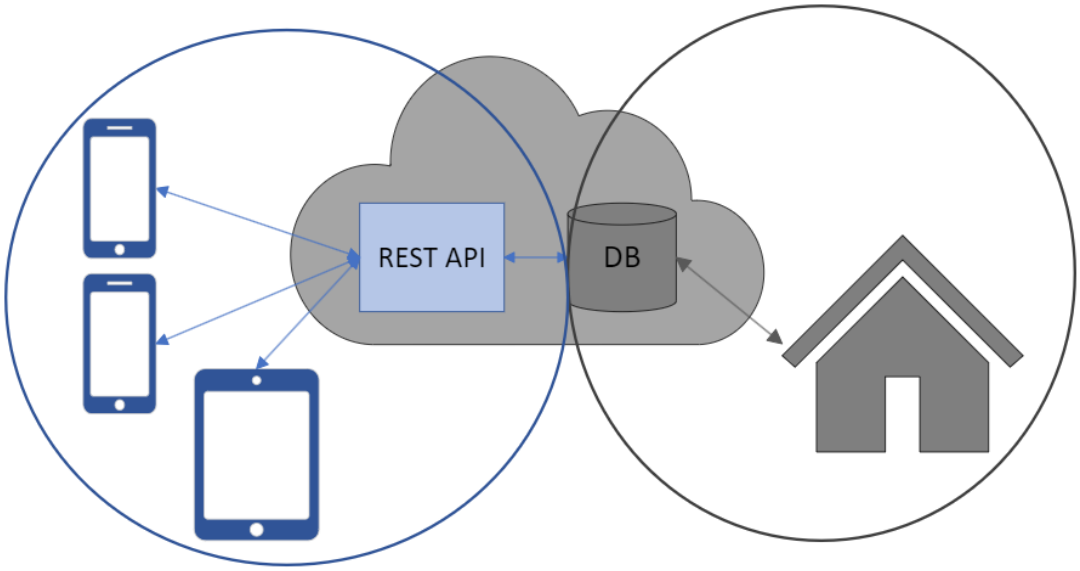
Platforma Visual Studio App Center je výkonným a uživatelsky přívětivým nástrojem pro vývoj zejména mobilních aplikací. Nabízí mnoho funkcí a služeb pro sestavování, testování, nasazování a monitorování aplikací, které mohou výrazně zlepšit proces vývoje a poskytovat tak koncovým uživatelům kvalitnější a spolehlivější aplikace.

3 NÁVRH WEBOVÉHO APLIKAČNÍHO ROZHRANÍ A MOBILNÍ APLIKACE

Tato kapitola diplomové práce se věnuje popisu stávajícího řešení a návrhu webového aplikačního rozhraní a mobilní aplikace pro řízení klimatické jednotky v chytré domácnosti. Při navrhování se definují např. funkční požadavky, architektura aplikace nebo rozvržení uživatelského rozhraní. Mezi hlavní výhody tvorby návrhu aplikace patří strukturovaný a organizovaný vývoj, udržovatelnost, rozšiřitelnost a opakované využití aplikace.

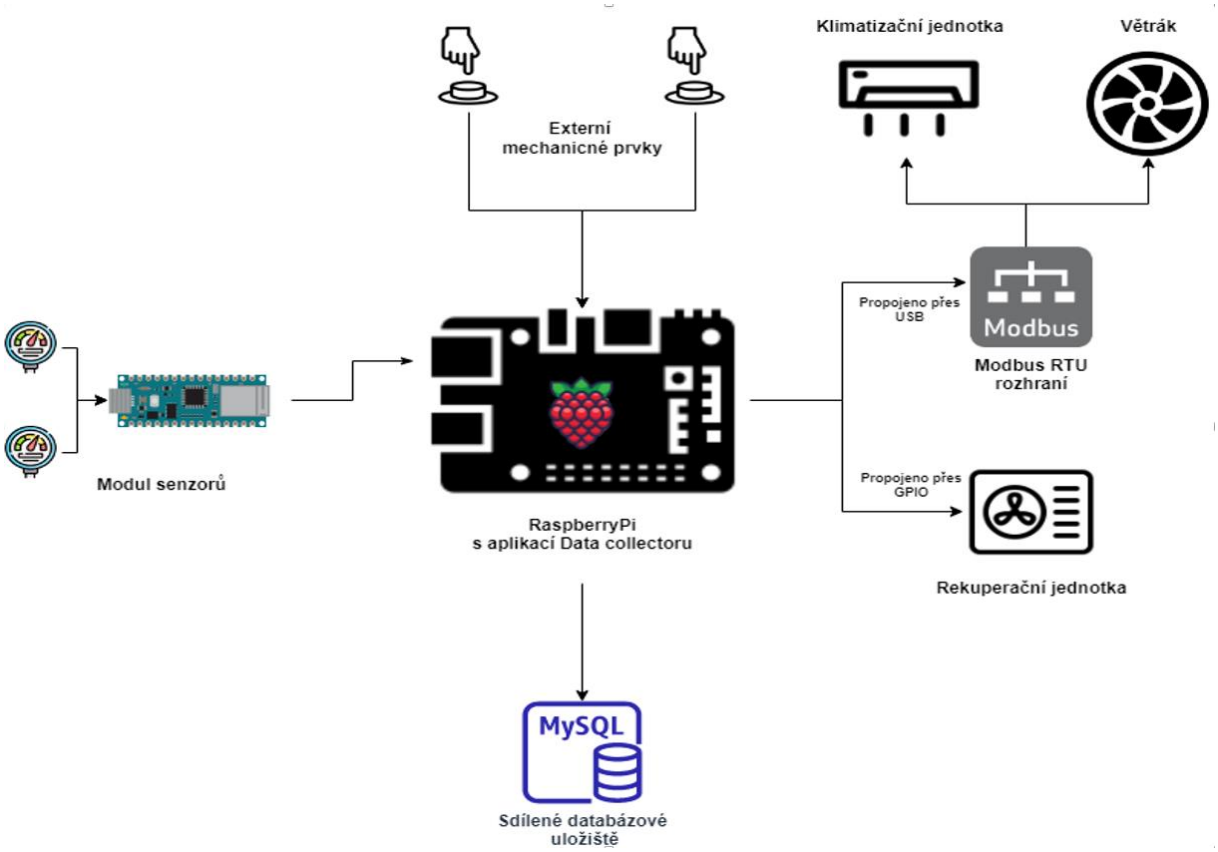
3.1 Koncept aplikace

V dnešní době se klade důraz na jednoduché a rychlé ovládání fyzických zařízení. K tomu se nejvíce hodí využití mobilních aplikací, protože chytrý mobilní telefon je již součástí života téměř každého člověka. Jelikož v současné době existuje k ovládání a správě systému HVAC pouze webová aplikace, která plní i funkci servisní aplikace, zaměřuje se diplomová práce na návrh a implementaci mobilní aplikace pro platformu Android a iOS, která přinese vybrané uživatelské funkce přímo do mobilního zařízení, což uživateli přinese vyšší komfort. Dále se věnuje také vybudování aplikačního rozhraní, které umožní mobilní aplikaci volat vybrané funkce systému. Na obrázku č. 11 je zobrazen koncept celého systému. Modře jsou vyznačeny komponenty, které byly nově vytvořeny a šedě jsou zvýrazněny již existující součásti systému. Uživatel skrze mobilní aplikaci interaguje s REST API, které zpracovává jeho požadavky a komunikuje s databází. Fyzická zařízení jsou pak řízena na základě hodnot v databázi.



Obrázek 11: Koncept aplikace

Obrázek č. 12 blíže popisuje fyzický systém, který je na předchozím obrázku představen šedou elipsou. Systém se skládá z klimatizační jednotky, která poskytuje chlazení a topení, a rekuperační jednotky a je řízen pomocí RaspberryPi. Vytvoření tohoto systému bylo předmětem diplomové práce v [39], která obsahuje jeho bližší popis.

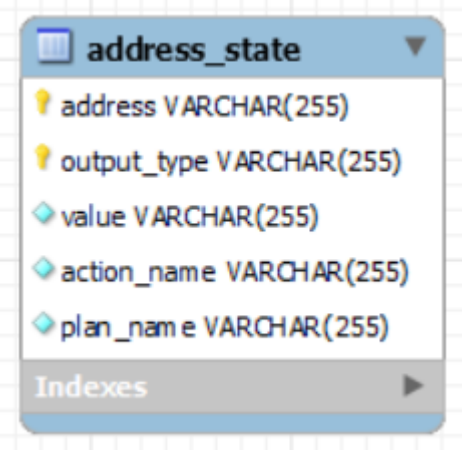


Obrázek 12: Schéma fyzického řešení systému [39]

3.2 Popis databáze

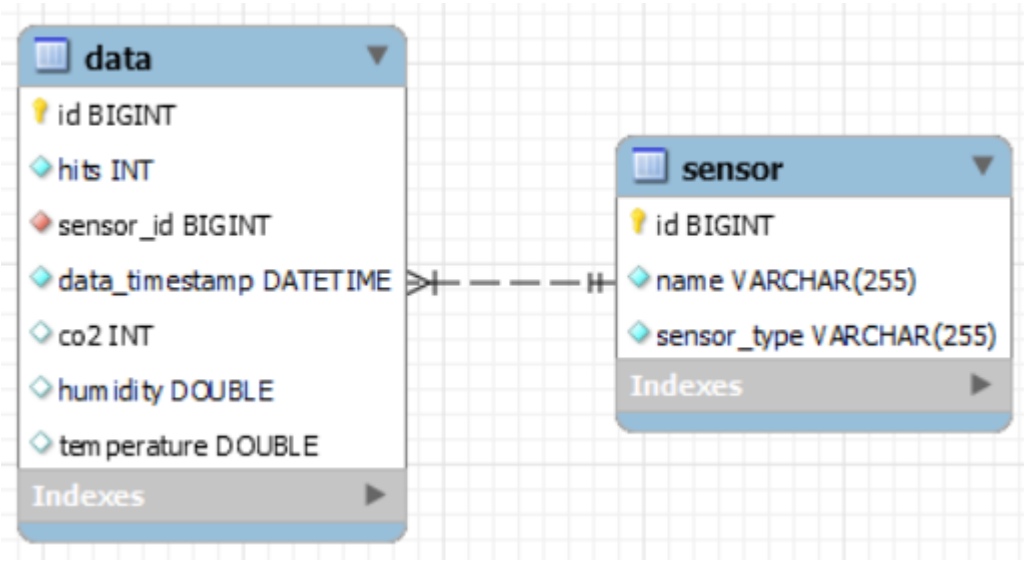
Pro ukládání dat byla využita již existující MySQL databáze ve verzi 8.0.31. Databáze obsahuje potřebné tabulky pro ovládání fyzických zařízení naplněné reálnými daty. Jelikož se práce nevěnuje návrhu a vytváření databáze, následující odstavce pouze stručně popisují vybrané tabulky.

Mezi nejzásadnější tabulky pro mobilní aplikaci patří tabulka *ADDRESS_STATE*, která je zobrazena na obrázku č. 13. Obsahuje informace o aktuálním stavu fyzických zařízení. Ve sloupci *ACTION_NAME* je název aktuálně probíhající akce, např. AC unit – ON nebo AC unit Mode – Heat, a ve sloupci *VALUE* je její hodnota. Seznam možných akcí a jejich hodnot je uložen v tabulce *ACTION*.



Obrázek 13: Diagram tabulky *ADDRESS_STATE*

Mezi další důležité tabulky patří tabulka *DATA*, která uchovává naměřené hodnoty ze sensorů. Je možné ukládat teplotu, vlhkost a koncentraci CO_2. Data jsou ukládána v intervalu jedné minuty a časové razítko záznamu je uloženo ve sloupci *DATA_TIMESTAMP*. Seznam dostupných sensorů je uložen v tabulce *SENSOR*. Jeden sensor může snímat jednu a více veličin. Popisované tabulky a jejich relace je zobrazena na obrázku č. 14.

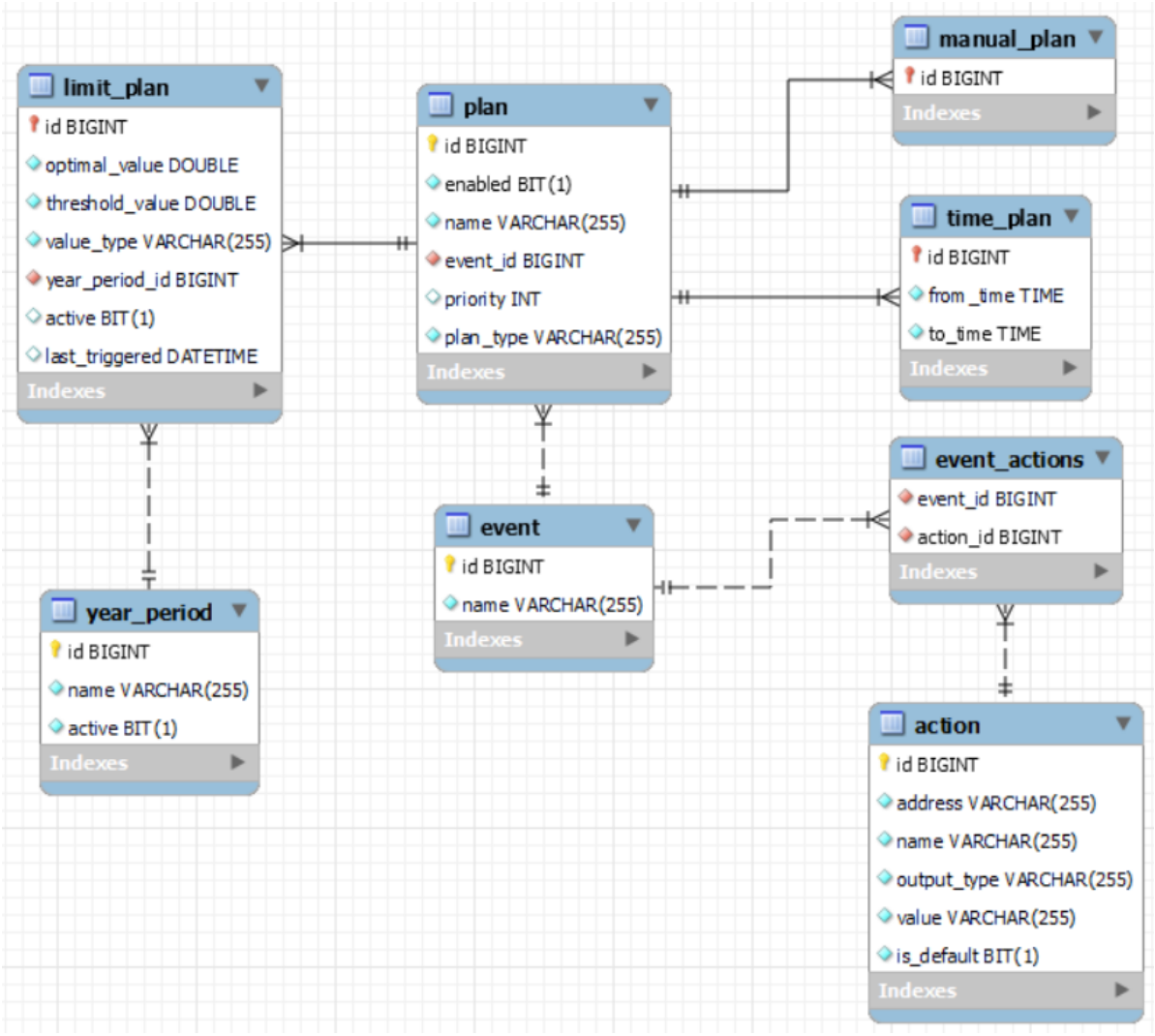


Obrázek 14: Diagram tabulek *DATA* a *SENSOR*

Další důležitou tabulkou je tabulka *PLAN* a její podtypy, a to:

- *LIMIT_PLAN* – označující limitní plán,
- *TIME_PLAN* – označující časový plán,
- *MANUAL_PLAN* – označující manuální plán.

U limitního plánu lze definovat letní nebo zimní režim. Seznam režimů je uložen v tabulce *YEAR_PERIOD*. Informace o tom, jaké období je v dané chvíli aktivní, se udržuje ve sloupci *ACTIVE*. Pro každý režim si uživatel může určit limitní hodnoty pro danou veličinu a událost, která se spustí při dosažení nastavené hodnoty. Seznam událostí je definován v tabulce *EVENT*. Každá událost se může skládat z více akcí, které jsou uložené v tabulce *ACTION*. Časový plán umožňuje uživateli spustit událost v daném časovém intervalu. Pro manuální plán lze definovat událost, která se spustí ihned v případě, že je plán aktivní. Zdali je plán aktivní, je uloženo ve sloupci *ENABLED* v tabulce *PLAN*. Obrázek č. 15 zobrazuje diagram tabulek, které souvisejí s tabulkou *PLAN* a byly popsány výše v tomto odstavci.



Obrázek 15: Diagram databázových tabulek souvisejících s tabulkou *PLAN*

Příkladem může být situace, kdy si uživatel definuje v limitním plánu v zimním režimu teplotu s dolní limitní hodnotou 19 °C a horní limitní hodnotou 30 °C. Ke spodní limitní hodnotě bude přiřazena událost topení a horní limitní hodnotě bude přiřazena událost

chlazení. V případě, že teplota v domě nabyde hodnoty menší, resp. větší hodnoty, než je definovaná v plánu, spustí se událost topení, resp. chlazení, na kterou již zareaguje vytápěcí systém, resp. klimatizace.

3.3 Návrh webového rozhraní

Webové rozhraní slouží jako prostředek pro komunikaci mezi databází a mobilní aplikací. Při jeho návrhu je potřeba zvážit zejména zásadní koncové body, které bude následně využívat mobilní aplikace. Důležitým aspektem je také definování funkčních a nefunkčních požadavků.

3.3.1 Funkční požadavky

Funkční požadavky zjednodušeně určují, co bude systém dělat. Cílem webového aplikačního rozhraní je poskytovat potřebná data z databáze skrze zabezpečený přístup. Pro webové rozhraní byly definovány následující funkční požadavky:

- FP 1 – Systém bude umožňovat zabezpečený přístup.
    - FP 1.1 – Systém bude umožňovat ověření uživatele pomocí uživatelského jména a hesla.
    - FP 1.2 – Systém bude na základě přihlašovacích údajů generovat Bearer token.
    - FP 1.3 – Token bude uchovávat identifikátor uživatele a uživatelské jméno.
- FP 2 – Systém bude komunikovat s databází.
    - FP 2.1 – Systém bude umožňovat načítání a editaci dat z databáze.
- FP 3 – Systém bude umožňovat načtení aktuálních a historických dat ze senzorů.
- FP 4 – Systém bude umožňovat načtení aktuálního stavu klimatické jednotky.
- FP 5 – Systém bude umožňovat načtení a editaci dat plánu:
    - FP 5.1 – limitního,
    - FP 5.2 – časového,
    - FP 5.3 – manuálního.
- FP 6 – Systém bude umožňovat vytvoření, editaci a smazání plánu:
    - FP 6.1 – časového,
    - FP 6.2 – manuálního.

3.3.2 Nefunkční požadavky

Nefunkční požadavky určují, jak budou v systému implementovány funkční požadavky a definují nároky na kvalitu, výkon, spolehlivost, bezpečnost a rozšiřitelnost systému. V rámci

diplomové práce jsou nefunkční požadavky určeny zvolenými technologiemi pro implementaci, testování a nasazení systému. Při jejich výběru je nutno posoudit požadavky na udržitelnost, bezpečnost a podporované platformy. Po zvážení těchto aspektů byly vybrány následující technologie, které již byly představeny v teoretické kapitole č. 2:

- NP 1 – Pro vývoj systému bude použit jazyk C#.
    - NP 1.1 – Pro vývoj systému bude použit framework ASP.NET Core.
    - NP 1.2 – Pro vývoj systému bude použit framework Entity Framework Core.
- NP 2 – Systém bude dokumentován pomocí nástroje Swagger UI.
- NP 3 – Systém bude testován pomocí nástroje Swagger UI.
- NP 4 – Pro distribuci a nasazení systému bude využit software Docker.

3.3.3 Zásadní koncové body

Koncové body představují specifické URL adresy, na kterých klienti mohou komunikovat s webovým rozhraním a získávat nebo odesílat data. Správný návrh koncových bodů je klíčový pro efektivní a intuitivní používání rozhraní a zajištění kvalitní interakce mezi klientem a serverem. Při návrhu webového rozhraní byly definovány následující zásadní koncové body:

- POST *baseURL*/api/login – generuje token na základě předaného uživatelského jména a hesla,
- GET *baseURL*/api/address-state/results – poskytuje data o aktuálním stavu klimatické jednotky,
- GET *baseURL*/api/data/actual – poskytuje aktuální data ze všech senzorů,
- GET *baseURL*/api/data/actual/{sensorId} – poskytuje aktuální data konkrétního sensoru, jehož identifikátor je předán v proměnné sensorId,
- GET *baseURL*/api/data/historical – poskytuje kompletní data ze všech senzorů,
- GET *baseURL*/api/data/historical/{sensorId} – poskytuje kompletní data konkrétního sensoru, jehož identifikátor je předán v parametru sensorId,
- GET *baseURL*/api/limit-plan/settings – poskytuje aktuální nastavení limitního plánu,
- PUT *baseURL*/api/limit-plan/settings – umožňuje editaci nastavení limitního plánu.
- GET *baseURL*/api/manual-plan – poskytuje seznam manuálních plánů,
- POST *baseURL*/api/manual-plan – vytváří manuální plán,
- PUT *baseURL*/api/manual-plan/{manualPlanId} – umožňuje editaci manuálního plánu, jehož identifikátor je předán v proměnné manualPlanId,

- DELETE *baseURL*/api/manual-plan/{manualPlanId} – umožňuje smazání manuálního plánu, jehož identifikátor je předán v proměnné manualPlanId,
- GET *baseURL*/api/time-plan – poskytuje seznam časových plánů,
- POST *baseURL*/api/time-plan – vytváří časový plán,
- PUT *baseURL*/api/time-plan/{timePlanId} – umožňuje editaci časového plánu, jehož identifikátor je předán v proměnné timePlanId,
- DELETE *baseURL*/api/time-plan/{timePlanId} – umožňuje smazání časového plánu, jehož identifikátor je předán v proměnné timePlanId,
- GET *baseURL*/api/year-period/active – poskytuje data o aktivním režimu.

3.4 Návrh mobilní aplikace

Mobilní aplikace je určena k ovládání již existující klimatické jednotky a její stěžejní funkcí je přehledné zobrazení aktuálních i historických dat, která byla zaznamenána různými sensory. Další funkcionalitou je také správa jednotlivých plánů a nastavování jejich parametrů. Při návrhu mobilní aplikace je nutné zvážit několik faktorů, a to vzhled uživatelského prostředí, výkonnost a ovladatelnost aplikace a podporu pro různé operační systémy. Cílem návrhu je definování funkčních a nefunkčních požadavků a vytvoření uživatelsky přívětivého a funkčního návrhu uživatelského prostředí.

3.4.1 Funkční požadavky

Hlavní funkcionalitou mobilní aplikace je přehledné zobrazení dat pomocí různých vizuálních prvků a správa jednotlivých typů plánů. Pro mobilní aplikaci byly definovány následující funkční požadavky:

- FP 1 – Mobilní aplikace bude komunikovat s databází prostřednictvím webového rozhraní.
- FP 2 – Mobilní aplikace bude ověřovat uživatele.
    - FP 2.1 – Uživatel se bude muset ověřit prostřednictvím uživatelského jména a hesla.
    - FP 2.2 – Mobilní aplikace bude bezpečně ukládat získaný token z API.
- FP 3 – Mobilní aplikace bude zobrazovat data získaná z webového rozhraní.
    - FP 3.1 – Mobilní aplikace bude zobrazovat aktuální informace o klimatické jednotce.
    - FP 3.2 – Mobilní aplikace bude zobrazovat aktuální data z jednotlivých sensorů.

    - FP 3.3 – Mobilní aplikace bude umožňovat zobrazení historických dat prostřednictvím grafu.
    - FP 3.4 – Historická data se budou zobrazovat pro vybraný jeden den.
- FP 4 – Mobilní aplikace bude umožňovat správu limitního plánu.
    - FP 4.1 – Mobilní aplikace bude umožňovat zobrazení aktuálního nastavení limitního plánu.
    - FP 4.2 – Mobilní aplikace bude umožňovat editaci parametrů limitního plánu.
- FP 5 – Mobilní aplikace bude umožňovat správu manuálního plánu.
    - FP 5.1 – Mobilní aplikace bude umožňovat zobrazení manuálních plánů.
    - FP 5.2 – Mobilní aplikace bude umožňovat vytvoření manuálního plánu.
    - FP 5.3 – Mobilní aplikace bude umožňovat editaci manuálního plánu.
    - FP 5.4 – Mobilní aplikace bude umožňovat smazání manuálního plánu.
    - FP 5.5 – Mobilní aplikace bude umožňovat editaci dat manuálního plánu.
- FP 6 – Mobilní aplikace bude umožňovat správu časového plánu.
    - FP 6.1 – Mobilní aplikace bude umožňovat zobrazení časových plánů.
    - FP 6.2 – Mobilní aplikace bude umožňovat vytvoření časového plánu.
    - FP 6.3 – Mobilní aplikace bude umožňovat editaci časového plánu.
    - FP 6.4 – Mobilní aplikace bude umožňovat smazání časového plánu.
    - FP 6.5 – Mobilní aplikace bude umožňovat editaci dat časového plánu.

3.4.2 Nefunkční požadavky

Při definování nefunkčních požadavků u mobilní aplikace bylo potřeba zvážit technické požadavky na aplikaci, pro který operační systémy bude určena nebo jaké jazyky bude podporovat. Následující výčet definuje jednotlivé nefunkční požadavky mobilní aplikace:

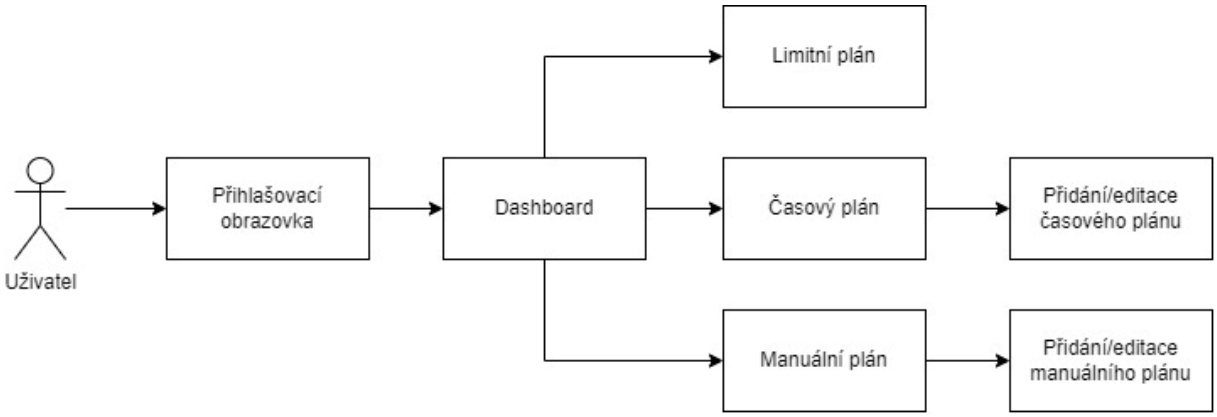
- NP 1 – Mobilní aplikace bude podporovat operační systém iOS a Android.
    - NP 1.1 – Pro vývoj mobilní aplikace bude využit jazyk C#.
        - NP 1.1.1 – Pro vývoj mobilní aplikace bude využit framework .NET MAUI.
- NP 2 – Aplikace bude podporovat zobrazení na výšku (režim portrait) a na šířku (režim landscape).
- NP 3 – Aplikace bude využívat jazykovou sadu pro jazyk:
    - NP 3.1 – angličtina,
    - NP 3.2 – čeština.

- NP 4 – Design mobilní aplikace bude tvořen v souladu s designem již existující webové aplikace.

3.4.3 Návrh uživatelského prostředí

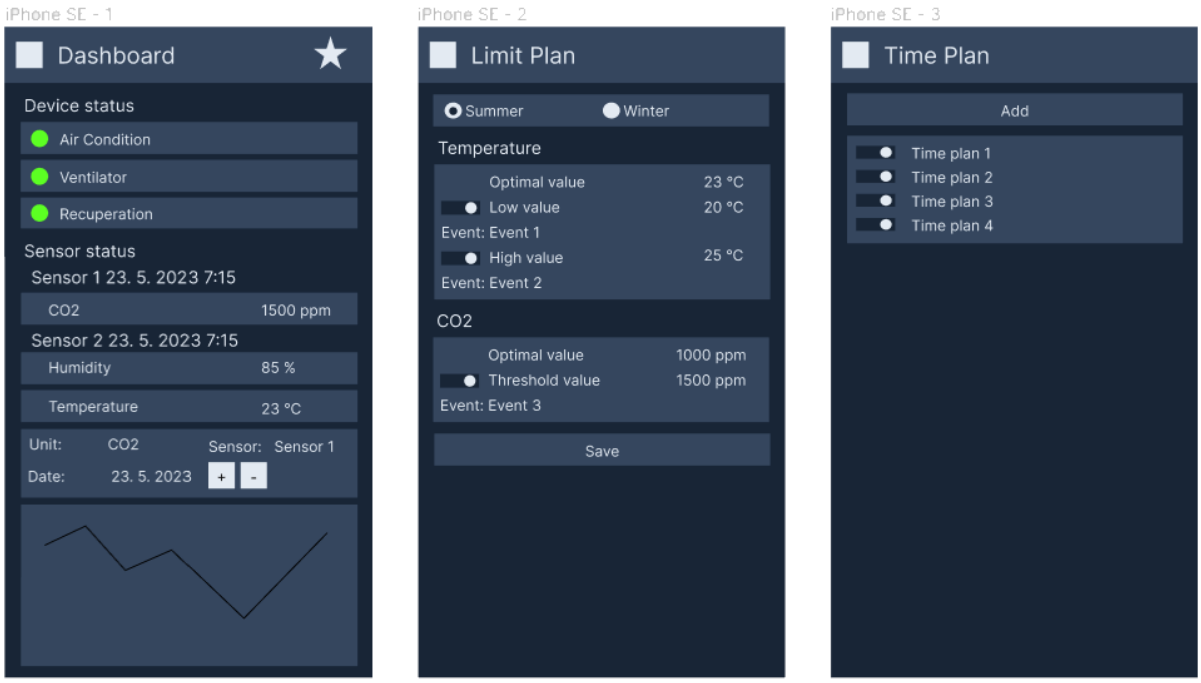
Správný návrh uživatelského prostředí mobilní aplikace hraje klíčovou roli při vytváření přívětivé uživatelské zkušenosti. Při návrhu je důležité brát v úvahu několik aspektů, které přispívají k uživatelsky příjemnému prostředí a snadnému ovládání aplikace. Je nezbytné, aby vizuální stránka aplikace byla atraktivní, srozumitelná a odpovídala jejímu zaměření. Použití vhodných barev, typografie, ikon a vizuálních prvků pomáhá vytvořit jednotný a esteticky příjemný vzhled aplikace.

Obrázek č. 16 zobrazuje přehled obrazovek mobilní aplikace a jejich návaznost. Při spuštění aplikace se uživateli, pokud není v zařízení uložen platný token, zobrazí přihlašovací obrazovka. Po přihlášení je uživatel přesměrován na hlavní stránku, kde jsou zobrazeny nejdůležitější údaje. Z postranního menu si uživatel může zobrazit jednotlivé obrazovky plánů, případně plány přidat nebo editovat.



Obrázek 16: Obrazovky mobilní aplikace

Samotný design uživatelského prostředí byl tvořen v souladu s designem již existující webové aplikace. Při návrhu designu byl kladen důraz na jednoduchost, funkčnost a přehledné zobrazení požadovaných informací. Obrázek č. 17 zobrazuje příklad návrhu hlavních obrazovek aplikace. Výsledná aplikace dodržuje vytvořený návrh, navíc přidává k některým prvkům tematické ikony pro vyšší přehlednost, což bude zobrazeno v následující kapitole.



Obrázek 17: Příklad návrhu obrazovek aplikace

4 IMPLEMENTACE WEBOVÉHO APLIKAČNÍHO ROZHRANÍ A MOBILNÍ APLIKACE

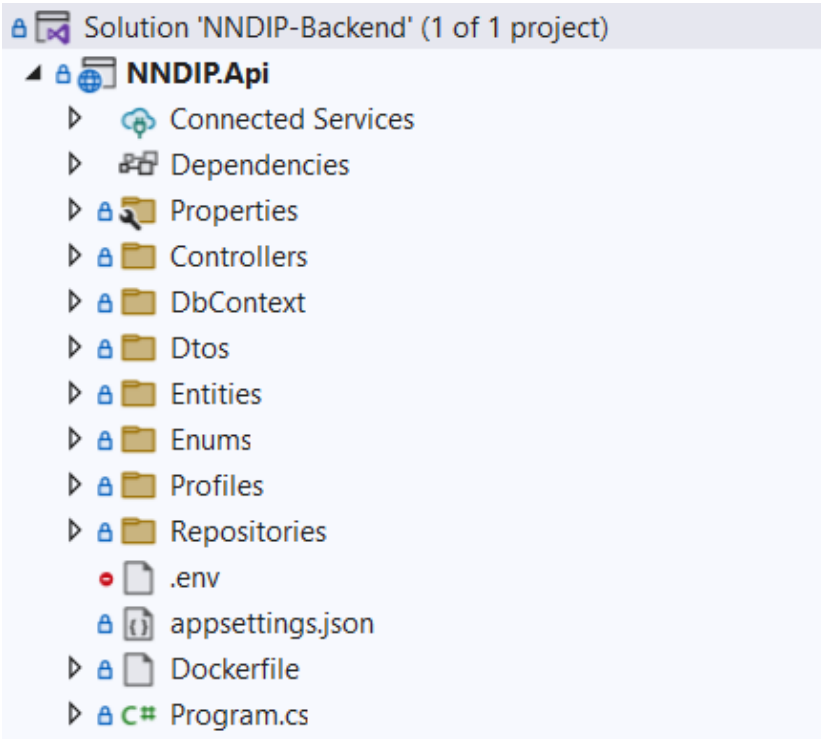
Následující kapitoly diplomové práce popisují implementaci praktické části, která se skládá ze dvou aplikací. První aplikací je webové aplikační rozhraní, které zprostředkovává komunikaci s databází. Druhou aplikací je pak samotná mobilní aplikace s názvem SmartHome. Obě aplikace byly vytvářeny ve vývojovém prostředí Microsoft Visual Studio 2022 v bezplatné edici Visual Studio Community na přenosném počítači s operačním systémem Windows 10.

4.1 Vývoj webového aplikačního rozhraní

Webové aplikační rozhraní bylo vytvořeno s využitím frameworku ASP.NET Core ve verzi 7 pomocí programovacího jazyka C#. Cílem bylo navrhnout a vytvořit REST API, které bude sloužit jako rozhraní pro komunikaci mezi klientem a již existující databází. Řešení se skládá z jednoho projektu s názvem *NNDIP.Api*.

4.1.1 Projekt NNDIP.Api

Zdrojové kódy aplikace jsou rozděleny dle logických částí do několika složek, což je zobrazeno na obrázku č. 18. Tato struktura projektu zvyšuje čitelnost, udržitelnost a rozšiřitelnost kódu.



Obrázek 18: Struktura projektu *NNDIP.Api*

Složka *DbContext* obsahuje jednu třídu *NndipDbContext*, která dědí ze třídy *DbContext* a která definuje entity a jejich vztahy, mapování na databázové tabulky a konfiguraci dalších nastavení spojení s databází. Tento princip je součástí ORM frameworku Entity Framework Core, který byl zmíněn v teoretické části. Jelikož bylo vytvářeno rozhraní nad již existující databází, byl uplatněn tzv. přístup Database First. EF Core nabízí nástroj pro automatické generování kontextu a jednotlivých modelů tabulek přímo z databáze. Pro vygenerování lze použít příkaz *Scaffold-DbContext*, kde je nutné zadat údaje k připojení do databáze ve formě řetězce tzv. connection string. Dále se specifikuje poskytovatel připojení do databáze, v tomto případě byl využit NuGet balíček *Pomelo.EntityFrameworkCore.MySql* ve verzi 7.0. Je možnost také určit výstupní složku pro třídy modelů nebo název kontextové třídy. Podrobnější popis a možnosti příkazu lze nalézt v [28].

Vygenerované modely na základě databázových tabulek obsahuje složka *Entities*. Platí, že po každou tabulku byla vygenerována právě jedna třída. Třídy obsahují pouze jednotlivé atributy, které představují sloupce tabulky, odkazy na jiné entity, které představují relace, a případně konstruktor pro inicializaci kolekcí.

Složka *Dtos* obsahuje třídy definující tzv. DTO (Data Transfer Object). Slouží k zapouzdření dat a jejich přenosu mezi klientem a API. Třídy obsahují pouze datové vlastnosti, které jsou potřebné pro přenos, což minimalizuje přenos nepotřebných dat přes síť. Zdrojové soubory v této složce jsou dále organizovány do podsložek dle jejich logické domény. DTO častokrát odpovídají jednotlivým entitám datové vrstvy, případně vynechávají nepotřebné atributy. V některých případech může být vhodné definovat DTO, který shlukuje dohromady atributy z více entit. Na obrázku č. 19 je zobrazena třída *LimitPlanSettings*, která představuje aktuální nastavení limitního plánu, jejíž zdrojovými entitami jsou *LimitPlan*, *Plan*, *Event* a *YearPeriod*. Některé atributy jsou zapouzdřené třídou *Threshold*, která představuje limitní hodnoty pro teplotu a koncentraci CO_2. Tento přístup umožňuje snadnější zpracování příchozích dat v klientské aplikaci.

```
namespace NNDIP.Api.Dtos.Plan.LimitPlan
{
    9 references
    public class LimitPlanSettings
    {
        5 references
        public SimpleYearPeriodDto YearPeriodDto { get; set; } = null!;
        3 references
        public double OptimalValueTemperature { get; set; }
        2 references
        public double OptimalValueCo2 { get; set; }
        3 references
        public Threshold TemperatureLow { get; set; } = null!;
        3 references
        public Threshold TemperatureHigh { get; set; } = null!;
        3 references
        public Threshold Co2 { get; set; } = null!;
        0 references
        public bool IsWinterActive
        {
            get
            {
                return YearPeriodDto.Name == EnumExtender.GetEnumDescription(YearPeriodType.WINTER);
            }
        }

        0 references
        public bool IsSummerActive
        {
            get
            {
                return YearPeriodDto.Name == EnumExtender.GetEnumDescription(YearPeriodType.SUMMER);
            }
        }
    }
}
```

Obrázek 19: Zdrojový kód třídy *LimitPlanSettings*

Složka *Enums* obsahuje potřebné výčty využívané v aplikaci, např. výčet *PlanType* definuje jednotlivé typy plánů nebo výčet *SensorDataType* definuje typ dat, které lze získat ze sensorů. Složka *Profiles* obsahuje jednu třídu *AutoMapperProfile*, která slouží pro mapování objektů mezi různými typy, v tomto případě mezi entitou a třídou DTO. Pro mapování je využit NuGet balíček *AutoMapper.Extensions.Microsoft.DependencyInjection* ve verzi 12.0.0. Použití knihovny zjednodušuje mapování objektů, zlepšuje čitelnost a údržbu kódu, snižuje duplicitu a zrychluje vývoj aplikace.

Složka *Repositories* je částí aplikace, která obsahuje třídy, tzv. repozitáře pro přístup k datům v databázi. Každá třída dědí z obecné třídy *GenericRepository<T>*, která obsahuje základní metody pro práci s daty a instanci třídy *NndipDbContext*, kde *T* označuje konkrétní datovou entitu. Rozhraní této třídy je zobrazeno na obrázku č. 20. Využití obecné třídy snižuje míru duplicitního kódu a zvyšuje přehlednost.

Obrázek 20: Rozhraní *IGenericRepository<T>*

Další důležitou třídou ve složce *Repositories* je *RepositoryWrapper*, která slouží jako fasáda pro přístup ke všem repozitářům. Tímto způsobem se centralizuje a zjednodušuje využívání repozitářů v kontrolérech.

Klíčovou součástí aplikace je složka *Controllers* obsahující třídy, tzv. kontroléry, které jsou odpovědné za zpracování příchozích HTTP požadavků z klientských aplikací, které komunikují s API. Každý kontrolér obsahuje metody, které zpracovávají parametry z požadavku a provádějí potřebnou logiku aplikace, jako je například volání jiných služeb nebo práce s databází. Nakonec klientovi vrací odpověď. Tyto metody jsou volány z konkrétní adresy pomocí HTTP metod. Aplikace využívá čtyři základní metody:

- GET – pro získání dat,
- POST – pro vytvoření nových dat,
- PUT – pro modifikaci dat,
- DELETE – pro smazání dat.

Každý kontrolér je potomkem třídy *ControllerBase* a obsahuje anotaci *ApiController* a *Route* s příslušnou adresou. Anotace *Authorize* povoluje přístup k metodám pouze autentizovaným uživatelům. Obrázek č. 21 zobrazuje užití výše uvedených anotací ve třídě *DataController*. S využitím návrhového vzoru dependency injection je třídě injektován objekt implementující

rozhraní *IRepositoryWrapper* zprostředkovávající komunikaci s databází a *IMapper* umožňující mapování datových entit a DTO.

```
[Authorize]
[Route("api/data")]
[ApiController]
1 reference
public class DataController : ControllerBase
{
    private readonly IRepositoryWrapper _repositoryWrapper;
    private readonly IMapper _mapper;

    0 references
    public DataController(IRepositoryWrapper repositoryWrapper, IMapper mapper)
    {
        _repositoryWrapper = repositoryWrapper;
        _mapper = mapper;
    }
```

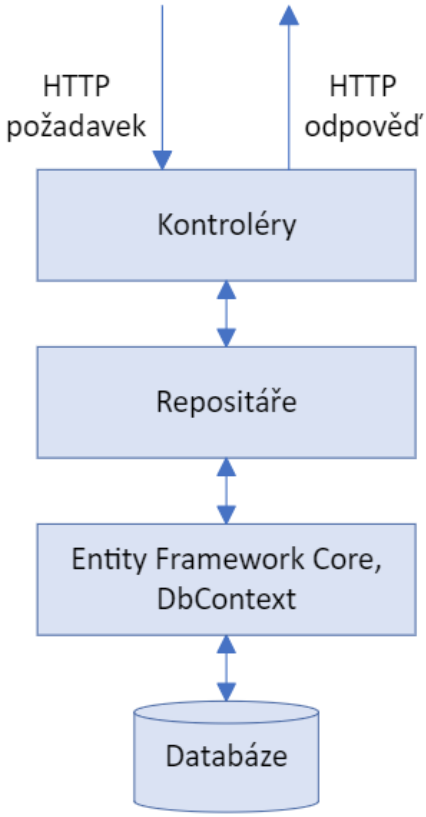
Obrázek 21: Část zdrojového kódu třídy *DataController*

Na obrázku č. 22 je zobrazena metoda *GetActualSensorData* ve třídě *DataController*. Anotace *HttpGet* určuje typ HTTP požadavku a cestu (URI) pro zavolání metody. Anotace *SwaggerOperation* definuje jedinečný identifikátor operace, který je použit pro vygenerování dokumentace Swagger. Použití tohoto identifikátoru je zásadní pro správné vygenerování API klienta v mobilní aplikaci. Metoda přijímá jeden číselný parametr představující identifikátor sensoru a její návratovou hodnotou je typ *ActionResult<IEnumerable<DataDto>>*. *ActionResult<T>* slouží jako obálka pro výsledná data a umožňuje vrátit různé typy HTTP stavových kódů např. NotFound (404) pro neexistující zdroj. První řádek metody získává aktuální data konkrétního sensoru a druhý řádek je mapuje na DTO a vrací je v odpovědi klientovi.

```
[HttpGet("actual/{sensorId}")]
[SwaggerOperation(OperationId = "GetActualSensorData")]
0 references
public async Task<ActionResult<IEnumerable<DataDto>>> GetActualSensorData(long sensorId)
{
    IEnumerable<Data> data = await _repositoryWrapper.DataRepository.GetActualSensorDataAsync(sensorId);
    return _mapper.Map<IEnumerable<DataDto>>(data).ToList();
}
```

Obrázek 22: Metoda *GetActualSensorData* ve třídě *DataController*

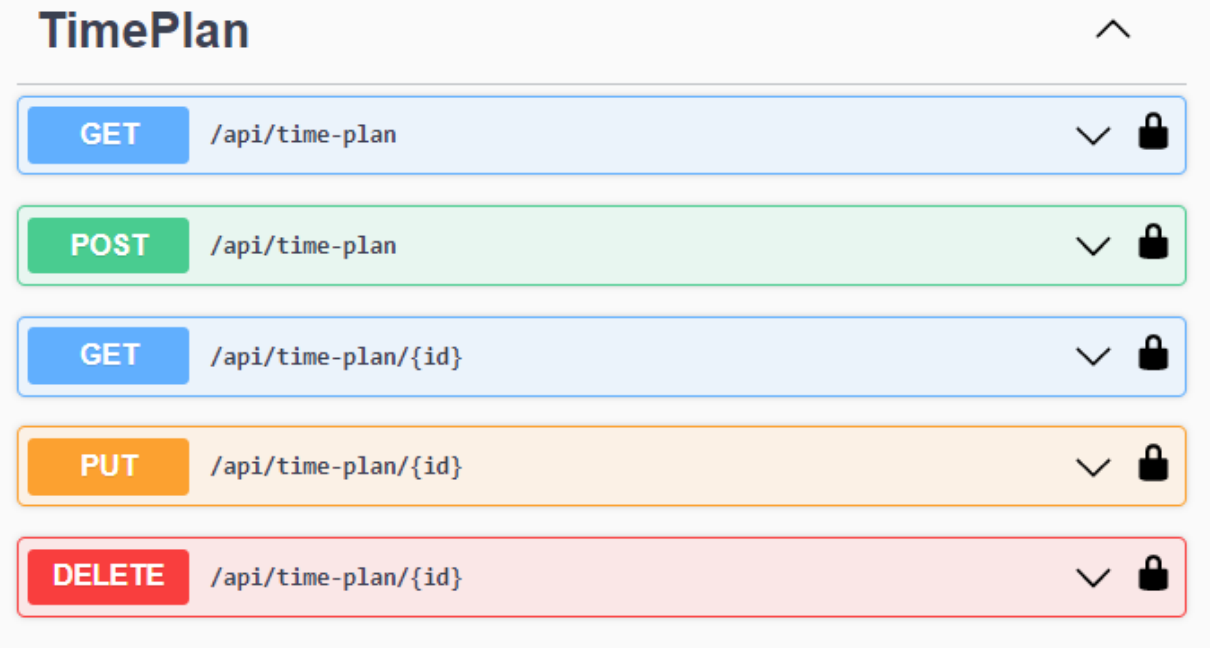
Obrázek č. 23 schematicky zobrazuje komunikaci mezi zásadními typy tříd a princip návrhového vzoru repository. Kontroléry komunikují s klientem a pro získání požadovaných dat využívají repozitáře, které s databází komunikují prostřednictvím třídy dědící z *DbContext*.



Obrázek 23: Návrhový vzor repository

Soubor *Program.cs* představuje vstupní bod aplikace a provádí konfiguraci potřebných služeb, např. nástroje Swagger UI. Projekt dále obsahuje soubor *.env*, který definuje potřebné proměnné prostředí, které není vhodné mít uvedené přímo ve zdrojovém kódu. Soubor *Dockerfile* slouží pro sestavení docker image a je podrobněji popsán v následující kapitole.

Aplikace využívá Swagger UI jako nástroj pro vizualizaci a interakci s API. Obrázek č. 24 zobrazuje příklad seznamu koncových bodů a jejich přístupovou adresu. Každý tento koncový bod lze interaktivně testovat s různými parametry. V případě, že volání koncového bodu vyžaduje autentizaci, což je označeno zámkem na konci řádku, je potřeba se ve webovém rozhraní autentizovat. Z webového rozhraní lze také exportovat soubor ve formátu JSON, který obsahuje popis API dle OpenAPI specifikace.



Obrázek 24: Ukázka Swagger UI

4.1.2 Kontejnerizace aplikace

V rámci vývoje webového aplikačního rozhraní byl využit nástroj Docker, který byl již představen v kapitole 2.3. Docker byl zvolen pro svou relativní jednoduchost použití a schopnost umožnit aplikaci rychle a snadno nasadit na jakýkoli systém.

Prvním krokem bylo vytvoření souboru Dockerfile. Jedná se o textový soubor, který obsahuje sérii instrukcí pro sestavení Docker image včetně definice závislostí, nastavení proměnných prostředí a dalších konfigurací. Obrázek č. 25 zobrazuje obsah souboru Dockerfile pro sestavení Docker image aplikace *NNDIP.Api*. První řádek definuje základní image, ze kterého bude nový Docker image vycházet, .NET SDK ve verzi 7.0 jako prostředí pro sestavení aplikace. Druhý řádek nastavuje pracovní adresář kontejneru. Další část příkazem COPY kopíruje všechny složky z aktuálního adresáře do pracovního adresáře. Následně jsou obnoveny potřebné NuGet balíčky a aplikace je sestavena. V dalších řádcích je definován jiný základní Docker image ASP.NET ve verzi 7.0 pro spuštění aplikace. Řádek č. 14 zajišťuje kopírování sestavené aplikace z předchozího image do aktuálního. Na posledním řádku je definován příkaz, který se provede při spuštění kontejneru – spuštění aplikace.

Dockerfile skládající se z více než jedné fáze sestavení využívá tzv. multi-stage sestavení. Každá tato fáze je definovaná v Dockerfile jako samostatná sekce. V tomto případě je první fází sestavení aplikace a druhou fází, která začíná na řádku č. 12, zkopírování potřebných souborů pro běh aplikace do finální image, čímž je dosaženo její nižší velikosti.

```
1   FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build-env
2   WORKDIR /App
3
4   # Copy everything
5   COPY . ./
6   # Restore as distinct layers
7   RUN dotnet restore
8   # Build and publish a release
9   RUN dotnet publish -c Release -o out
10
11  # Build runtime image
12  FROM mcr.microsoft.com/dotnet/aspnet:7.0
13  WORKDIR /App
14  COPY --from=build-env /App/out .
15  ENV ASPNETCORE_URLS=http://+:8081
16  EXPOSE 8081
17  ENTRYPOINT ["dotnet", "NNDIP.Api.dll"]
```

Obrázek 25: Dockerfile aplikace *NNDIP.Api*

Samotné sestavení Docker image je realizováno příkazem *docker build*. Přepínačem *-t* se definuje název a případně tag ve formátu název:tag. Přepínač *-f* určuje cestu, kde se nachází Dockerfile. Následující řádek zobrazuje vytvoření Docker image s názvem *nikolajackova/nndip* s tagem *api* pomocí příkazu *docker build*.

docker build -t nikolajackova/nndip:api -f NNDIP.Api/Dockerfile .

Následující příkaz *docker run* vytváří z existujícího image kontejner. Přepínač *--name* určuje název kontejneru. Přepínače *-p* propojuje port 8081 na hostitelském počítači s portem 8081 v kontejneru, díky tomu lze komunikovat s běžící aplikací. Parametr *--env-file* definuje soubor s proměnnými prostředí a parametr *--env* nastavuje proměnnou prostředí pro připojení do databáze, což umožňuje změnit databázi bez sestavování nového Docker image. V posledním kroku se definuje image, ze které má kontejner vzniknout.

docker run --name nndip -p 8081:8081 --env-file .env --env "CONNECTIONSTRING__NNDIPDBCONN=Server=server;User ID=user;Password=password;Database=database" nikolajackova/nndip:api

Pomocí příkazu *docker push* lze Docker image nahrát do repozitáře DockerHub, který umožňuje jeho snadné sdílení. Pro stáhnutí image se pak používá příkaz *docker pull*.

docker push nikolajackova/nndip:api
docker pull nikolajackova/nndip:api

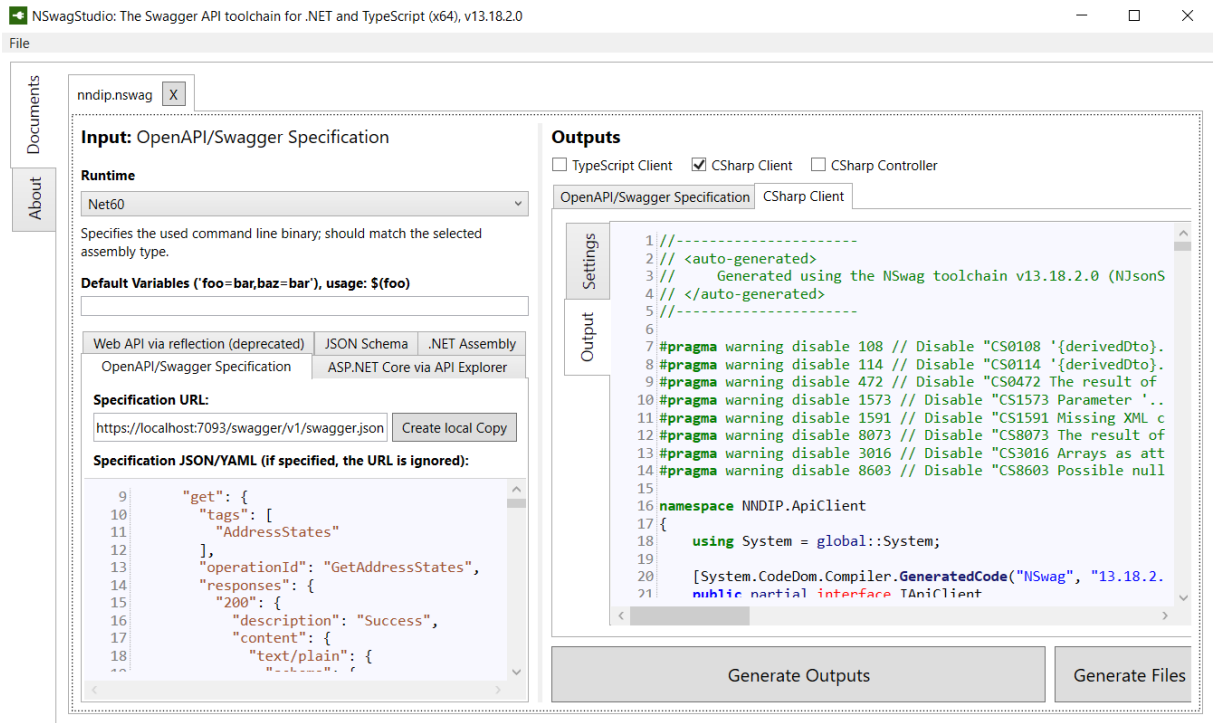
Aplikace byla nahrána na server a přístup k dokumentaci webového rozhraní je možný na adrese https://api.nixedev.eu/swagger/index.html.

4.2 Vývoj mobilní aplikace

Mobilní aplikace SmartHome se skládá celkem ze dvou projektů. První projekt, *NNDIP.ApiClient*, zajišťuje komunikaci s webovým aplikačním rozhraním. Druhý projekt, *NNDIP.Maui*, vytváří mobilní uživatelské prostředí a je závislý na prvním projektu. Aplikace byla takto rozdělena z důvodu přehlednosti a oddělení webové komunikace od samotného uživatelského prostředí mobilní aplikace. Cílem bylo vytvořit přehlednou mobilní aplikaci pro zobrazování důležitých dat a nastavování základních údajů klimatické jednotky.

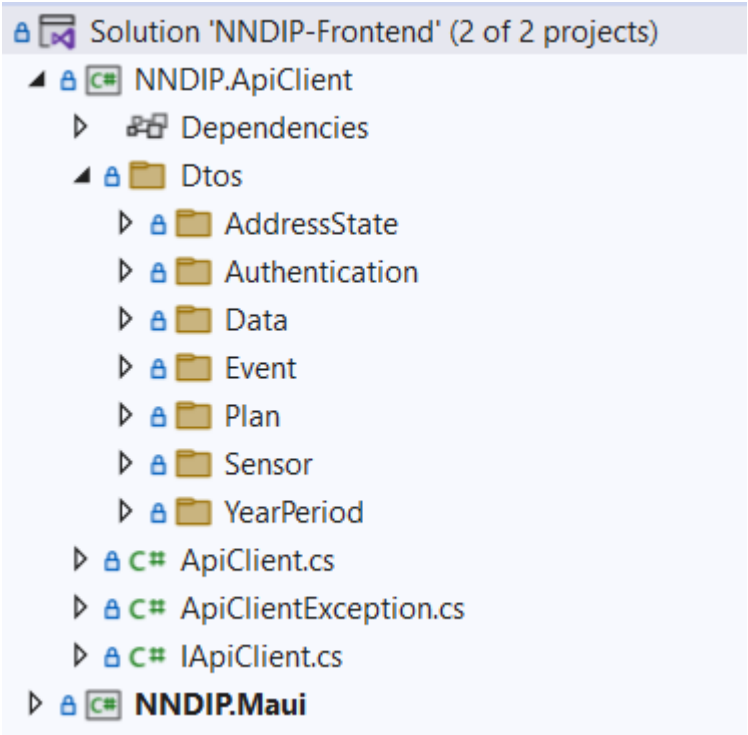
4.2.1 Projekt NNDIP.ApiClient

Prvním projektem v rámci řešení mobilní aplikace je *NNDIP.ApiClient*, který zajišťuje pouze komunikaci s webovým rozhraním. Všechny zdrojové soubory jsou vygenerované pomocí programu NSwagStudio, který využívá nástroj nswag a jehož uživatelské prostředí je zobrazeno na obrázku č. 26. Vstupem je URL, na které se nachází OpenAPI/Swagger specifikace webového rozhraní ve formátu JSON. Výstupem je soubor, který obsahuje všechny potřebné třídy a metody pro realizaci funkčního spojení s webovým aplikačním rozhraním. Pro správnou funkčnost generovaných tříd a metod je nutné v koncových bodech webového rozhraní definovat jedinečný atribut *OperationId* pomocí anotace *SwaggerOperation*, což je zobrazeno na obrázku č. 22.



Obrázek 26: Uživatelské prostředí programu NswagStudio

Použití nástroje NSwagStudio urychluje vývoj aplikace a minimalizuje chyby, které se mohou do zdrojových kódů zanést při ručním programování. Jelikož program generuje všechny třídy do jednoho souboru, byla struktura projektu pro přehlednost upravena tak, jak je zobrazeno na obrázku č. 27.

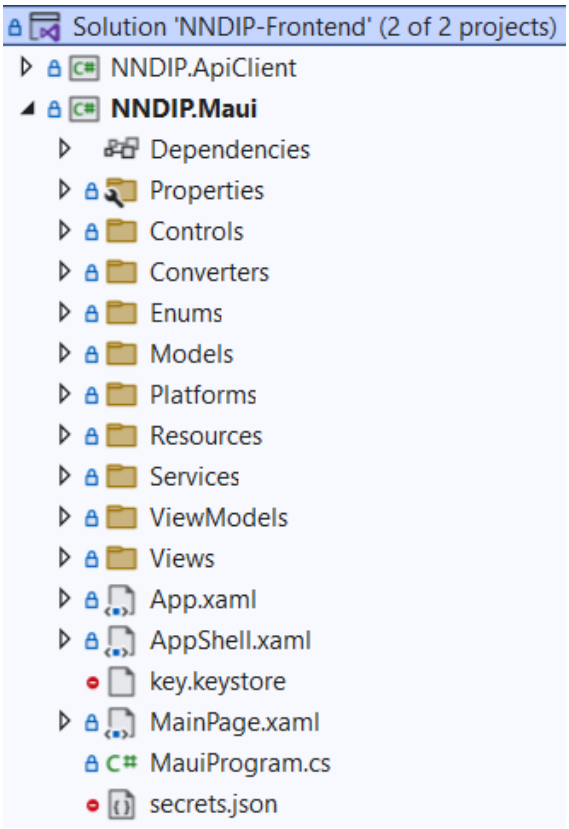


Obrázek 27: Struktura projektu *NNDIP.ApiClient*

4.2.2 Projekt NNDIP.Maui

Projekt *NNDIP.Maui* vytváří uživatelské prostředí pro mobilní aplikaci s využitím relativně nového frameworku .NET MAUI pro .NET ve verzi 7. Framework byl zvolen z důvodu jeho multiplatformní vlastnosti, jelikož aplikace bude publikována jak pro operační systém iOS, tak i pro Android. Jako programovací jazyk byl použit moderní vyšší programovací jazyk C#.

Při implementaci byl využit architektonický návrhový vzor MVVM (Model-View-ViewModel), který od sebe odděluje tři softwarové vrstvy – uživatelské rozhraní (View), modelová data (Model) a prostředníka mezi těmito dvěma vrstvami (ViewModel) a díky tomu zvyšuje přehlednost, znovupoužitelnost a flexibilitu zdrojového kódu a umožňuje použití vazby dat mezi vrstvami. Vzor MVVM byl realizován s využitím NuGet balíčku *CommunityToolkit.Mvvm* ve verzi 8.1.0, který usnadňuje jeho implementaci. Na obrázku č. 28 je zobrazena struktura projektu, kde nejzásadnějšími složkami jsou právě složky obsahující zdrojové soubory jednotlivých vrstev – *Models*, *ViewModels* a *Views*.



Obrázek 28: Struktura projektu *NNDIP.Maui*

Složka *Views* obsahuje soubory, které jsou rozděleny do podsložek dle jejich logické domény:

- *Dashboard* – pro hlavní stránku,
- *Plan* – pro stránky zobrazující plány,
- *Startup* – pro přihlašovací obrazovku a stránku načítání.

Tyto zdrojové soubory využívají značkovací jazyk XAML pro vytvoření jednotlivých stránek uživatelského prostředí. Každá třída představující stránku je závislá na třídě z vrstvy ViewModel, která jí je předána v konstruktoru pomocí návrhového vzoru dependency injection. Pro využití atributů a metod z injektované třídy je nutné použít datovou vazbu, která je realizována přiřazením objektu dané třídy do vlastnosti *BindingContext*, což je zobrazeno na obrázku č. 29. V XAML dokumentu je pak nutné jednotlivé atributy nebo metody přiřadit danému ovládacímu prvku.

```
9 references
public partial class LoginPage : ContentPage
{
    0 references
    public LoginPage(LoginPageViewModel loginPageViewModel)
    {
        InitializeComponent();
        BindingContext = loginPageViewModel;
    }
}
```

Obrázek 29: Dependency injection ve třídě *LoginPage*

Složka *ViewModels* obsahuje třídy z vrstvy ViewModel, které představují logiku uživatelského rozhraní. Zdrojové soubory v této složce se zaměřují na manipulaci s daty a poskytování správné funkcionality aplikace. Stejně jako v předchozí popisované složce jsou soubory rozděleny do složek dle jejich logické domény. Jedna třída z vrstvy ViewModel může být sdílena více třídami z vrstvy View, což je jedním z přínosů architektury MVVM, jelikož se snižuje míra duplicity kódu. V projektu je např. sdílena třída *ManualPlanPageViewModel* mezi třídy *ManualPlanPage* a *ManualPlanPageIOS*, kde se pro danou obrazovku využívá jiný grafický prvek v závislosti na operačním systému. Tento platformě specifický kód lze odlišit pomocí direktivy preprocesoru, jak je zobrazeno na obrázku č. 30.

```
                new ShellContent
                {
                    Icon = Icons.ManualPlan,
                    Title = AppResources.manualPlanPage_PageTitle,

#if IOS             ContentTemplate = new DataTemplate(typeof(ManualPlanPageIOS)),
#else
                    ContentTemplate = new DataTemplate(typeof(ManualPlanPage)),
#endif
                }
            };
```

Obrázek 30: Použití direktivy preprocesoru pro platformě závislý kód

Složka *Models* slouží obecně pro uchovávání tříd, které reprezentují datový model aplikace. Většina těchto tříd je definována v projektu *NNDIP.ApiClient*. Projekt *NNDIP.Maui* obsahuje pomocné modelové třídy, které byly vytvořeny za účelem vhodnější reprezentace zdrojových dat pro jednodušší zobrazení v aplikaci.

Ve složce *Services* jsou definované třídy poskytující aplikaci služby. Nejdůležitější třídou je statická třída *RestService*, která obsahuje atribut typu *IApiClient* z projektu *NNDIP.Api* pro realizaci komunikaci s webovým rozhraním. Tuto třídu využívají zejména třídy z vrstvy ViewModel při čtení nebo zapisování dat skrze webové rozhraní. Další důležitou službou je

služba, která je reprezentována třídou *AuthenticationService*. Jejím účelem je poskytování, zpracování a uložení autentizačního tokenu do zařízení. Poslední třídou této složky je *ExceptionHandlingService*, která se stará o zpracování výjimek.

Složka *Controls* slouží pro uchovávání vlastních uživatelských ovládacích prvků. Pokud je potřeba definovat si vlastní prvek např. tlačítko, seznam, záložku, který není dostupný v základní sadě prvků, je možné si definovat vlastní.

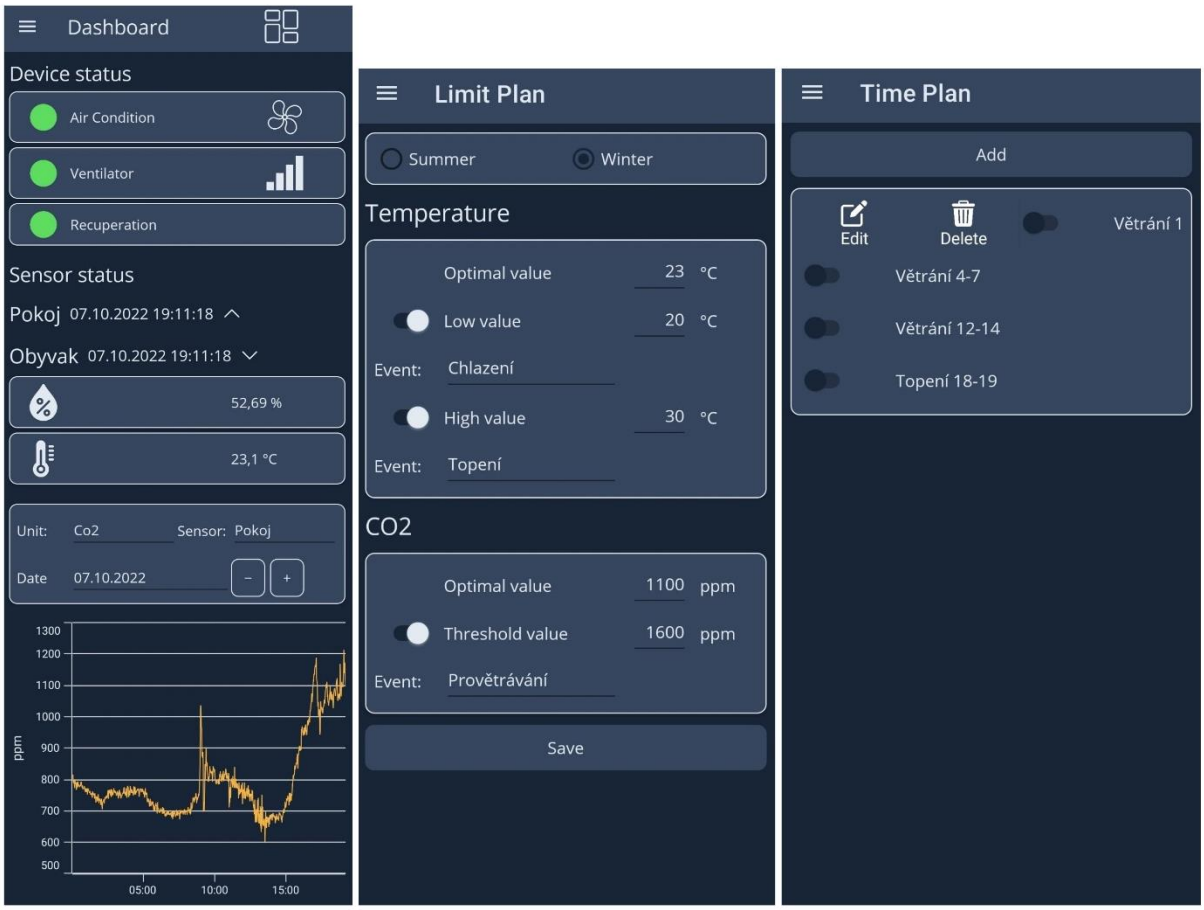
Složka *Converters* obsahuje třídy, které se používají k transformaci dat mezi různými formáty. V projektu jsou použity zejména třídy, které konvertují řetězcovou nebo booleovskou hodnotu získanou z webového rozhraní na odpovídající ikonu pro zobrazení v uživatelském prostředí.

Složka *Enums* obsahuje potřebné výčty pro daný projekt. Složka *Platforms* obsahuje podsložky pro jednotlivé platformy, které obsahují specifický kód pro danou platformu. Ve složce *Resources* jsou umístěny soubory uchovávající různé typy prostředků. V podsložce *Images* se nachází soubory s používanými ikonami v aplikaci. V podsložce *Languages* se nachází soubory s překlady textů do různých jazyků. Tento přístup umožňuje snadnou lokalizaci a poskytuje mezinárodní jazykové prostředí aplikace. Podsložka *Styles* obsahuje stylovací prostředky, jejichž použití zajišťuje konzistentní vzhled aplikace a stylizaci prvků na jednom místě.

Původně byl projekt vytvořen pod verzí .NET 6, ale z důvodu mnoha oprav ve frameworku .NET MAUI, které byly realizovány právě ve verzi 7, byl projekt přemigrován do novější verze. Problémem, který byl nejzásadnější a byl hlavním důvodem pro přemigrování, bylo chybné chování komponenty *RefreshView*, která zajišťuje možnost obnovení obsahu přesunem obrazovky dolů, v kombinaci s komponentou *ScrollView*, která umožňuje posouvání obsahu. Tento problém byl řešen v rámci požadavku č. 11375 v repozitáři dotnet/maui na platformě GitHub [34]. Při vývoji aplikace bylo nutné několikrát přistoupit k náhradním řešením některých chyb, které se vyskytovaly ve frameworku. Tyto chyby se týkaly zejména operačního systému iOS, kde byl problém zejména s komponentami *ScrollView*, *CollectionView* a *RefreshView*. Při použití některých knihoven třetích stran, např. knihovna Syncfusion, nastávaly při kompilaci projektu chyby, které znemožňovaly využití dané knihovny v aplikaci. Odstranění těchto chyb vyžadovalo opravu příslušných knihoven jejím autorem. Při vývoji se potvrdily některé komentáře uvedené v kapitole 2.2.2 o nedostatečné kvalitě frameworku .NET MAUI.

4.2.3 Výsledné uživatelské prostředí aplikace

Ve výsledném vzhledu aplikace jsou drobné změny oproti návrhu, který byl představen v předchozí kapitole. Nejvýraznější změna je na hlavní obrazovce v části *Sensor status*, kde údaje naměřené jednotlivými senzory lze pomocí dotyku sbalit nebo rozbalit. V případě většího množství senzorů přináší tato funkce vyšší přehlednost hlavní stránky. Tato možnost je dostupná pouze pro operační systém Android, jelikož chování komponenty *CollectionView*, jejíž použití je v tomto případě nutné, bylo pro operační systém iOS opraveno až s verzí .NET 8 [35]. Na stránce časového plánu, resp. manuálního plánu, byla funkcionalita pro editaci nebo smazání záznamu realizována pomocí posunu položky vpravo. Obrázek č. 31 zobrazuje konečný vzhled vybraných obrazovek aplikace pro operační systém Android.



Obrázek 31: Konečný vzhled aplikace pro operační systém Android

5 TESTOVÁNÍ A NASAZENÍ MOBILNÍ APLIKACE

Poslední kapitola se zaměřuje na konkrétní postupy a procesy spojené s testováním a nasazením mobilní aplikace na operační systémy iOS a Android. Klade si za cíl poskytnout přehled použitých technologií a postupů, které pomáhají zajistit efektivní testování a úspěšné nasazení mobilní aplikace s vysokou kvalitou a uživatelskou spokojeností.

5.1 Testování aplikace

Testování je zásadním krokem v životním cyklu vývoje aplikace, který slouží k ověření správné funkčnosti, kvality a výkonu aplikace. Cílem testování je odhalit případné chyby, problémy a nedostatky, které by mohly ovlivnit uživatelský zážitek nebo stabilitu aplikace.

Aplikace byla testována s využitím fyzických mobilních zařízení, a to konkrétně mobilního telefonu Samsung Galaxy A53 s operačním systémem Android 13 a iPhone 7 s operačním systémem 15.1. Obě zařízení bylo nutné připojit k počítači a povolit u nich režim ladění přes USB. Alternativou k testování na fyzických zařízení je využití emulátorů, které simulují prostředí fyzického mobilního zařízení, což ale může být náročnější na prostředky počítače.

Pro účely sdílení aplikace pro otestování na zařízeních dalších uživatelů byla využita platforma Visual Studio App Center, která byla představena v teoretické části. Před samotným sdílením aplikace je nutné ji sestavit a získat instalační soubor. K tomuto účelu byla využita již představená část Azure Pipelines platformy Azure DevOps, kde byly vytvořeny celkem čtyři samostatné automatizované postupy (tzv. pipeline). Jedna pipeline slouží pro sestavení mobilní aplikace pro operační systém Android, resp. iOS, a druhá realizuje nahrání instalačního souboru na platformu Visual Studio App Center vývojáři a urychluje a zkvalitňuje tak její vývoj i testování.

Operační systém Android pro vytvoření instalačního souboru vyžaduje podepsaný klíč, který má příponu .keystore a lze ho vygenerovat z vývojového prostředí pomocí nástroje keytool. Vygenerovaný soubor je nutné uvést při sestavování projektu. Výstupem sestavení je podepsaný instalační soubor s příponou .aab (Android App Bundle), který nahrazuje dříve používaný soubor s příponou .apk. Instalační soubor je pak nahrán do Visual Studio App Center, odkud si ho mohou další uživatelé stáhnout a nainstalovat na svá zařízení. [36]

Pro distribuci aplikací pro operační systém iOS je nutné mít aktivní účet Apple Developer Account, který poskytuje vývojářům kompletní sadu nástrojů a zdrojů potřebných k úspěšnému vývoji aplikací a zajišťuje, že aplikace splňují požadavky a standardy stanovené

společností Apple. Účet je přístupný přes webový prohlížeč na adrese https://developer.apple.com. Po přihlášení do vývojářského účtu je nutné vytvořit certifikát. Pro účely testování se vytváří certifikát typu iOS App Development a pro nasazení do obchodu App Store se vytváří certifikát typu iOS Distribution. Při vytváření certifikátu je nutné připojit soubor .csr (Certificate Signing Request), který lze získat pomocí programu OpenSSL. Tento soubor obsahuje žádost o podepsání certifikátem ze strany certifikační autority. Při jeho generování je nutné uvést název organizace, emailovou adresu a další údaje. Po získání certifikátu, což je soubor s příponou .cer a lze ho stáhnout ve vývojářském účtu Apple, je dalším krokem vytvoření souboru s příponou .pem, který slouží pro uložení certifikátu a soukromého klíče. Pro jeho vygenerování lze využít program OpenSSL. Tento soubor je vstupem následujícího příkazu *openssl pkcs12* opět v programu OpenSSL, který vytváří soubor s příponou .p12. Tento soubor obsahuje certifikát a soukromý klíč a je chráněn heslem a využívá se pro nainstalování daného certifikátu na fyzické zařízení. [37]

Ve vývojářském účtu je nyní dalším krokem vytvoření identifikátoru aplikace, tzv. App ID, který slouží k identifikaci, jaká aplikace se bude distribuovat nebo aktualizovat na Apple App Store. Identifikátor je nutné také nastavit v projektu mobilní aplikace. Po vytvoření App ID následuje vytvoření profilu, který lze exportovat z vývojářského účtu ve formě soboru s příponou .mobileprovision. Profil je svázán s konkrétní aplikací prostřednictvím App ID a s již vytvořeným certifikátem. [38]

Při sestavování projektu je nezbytné uvést certifikát se soukromým klíčem, který je uložen v souboru typu .p12 a jeho přístupové heslo. Dále je také nutné uvést profil uložený v souboru ve formátu .mobileprovision. Po sestavení projektu je vytvořený instalační soubor s příponou .ipa nahrán na platformu Visual Studio App Center.

5.2 Nasazení aplikace do oficiálních mobilních obchodů

Nasazení mobilní aplikace zahrnuje procesy a postupy, které jsou nezbytné pro úspěšné spuštění a provoz aplikace v reálném prostředí, takže je dostupná uživatelům ke stažení a používání.

5.2.1 Nasazení aplikace do Google Play

Pro publikování mobilních aplikací pro operační systémy Android se využívá obchod Google Play Store. Pro samotné nasazení aplikace je nutné mít založený účet vývojáře Google Play, se kterým lze využívat službu Google Play Console, která slouží pro samotné publikování a správu mobilních aplikací. V prvním kroku je nutné vytvořit základní záznam o aplikaci, což

zahrnuje poskytnutí klíčových informací, jako je název aplikace, popis a kategorie. Důležité je také nahrání ikony, snímků obrazovky a dalších grafických prvků, které budou vizuálně aplikaci reprezentovat. V dalších krocích uživatel pokračuje s detailní konfigurací, která zahrnuje definici cenové politiky aplikace nebo případná omezení týkající se regionu nebo zařízení. Po vyplnění všech povinných údajů je možné nahrát podepsaný soubor .aab, který již byl v tomto případě nahrán na platformu Visual Studio App Center.

Mobilní aplikace SmartHome byla nahrána k publikování do obchodu Google Play jako bezplatná aplikace a aktuálně je ve fázi schvalování.

5.2.2 Nasazení aplikace do App Store

Pro publikování mobilních aplikací pro operační systémy iOS se využívá obchod App Store. Pro nasazení aplikace je nutné mít aktivní účet Apple Developer, který byl zmiňován v kapitole 5.1. Samotný proces publikace předchází vytvoření certifikátu typu iOS Distribution a dalších potřebných souborů k podpisu aplikace a vytvoření výsledného instalačního souboru. Tento proces je totožný s tím, který byl popisován v kapitole 5.1 v odstavci věnujícímu se testování aplikace pro platformu iOS.

Dalším krokem je založení projektu aplikace ve vývojářském účtu, kde je nutné vyplnit povinné údaje, jako je např. název aplikace, popis, ikona nebo snímky obrazovky. Instalační soubor lze do projektu nahrát pomocí nástroje od společnosti Apple s názvem Transporter. Tento nástroj daný soubor zkontroluje, zdali vyhovuje požadavkům např. na kvalitu ikony aplikace.

Mobilní aplikace SmartHome byla nahrána k publikování do obchodu App Store a nyní je ve fázi schvalování.

ZÁVĚR

Hlavním cílem této diplomové práce bylo navrhnout a implementovat mobilní aplikaci pro již existující HVAC systém skládající se z klimatizační a rekuperační jednotky pro operační systémy Android a iOS. K mobilní aplikaci bylo nutné vytvořit webové aplikační rozhraní, které umožňuje komunikaci aplikace se systémem.

První kapitola teoretické části práce se věnuje konceptu internetu věcí. Představuje ucelený pohled na definici a charakteristiku IoT, jeho architekturu a základní přehled používaných technologií. Je zde zdůrazněno, že internet věcí nachází uplatnění v mnoha oblastech, od průmyslu až po zdravotnictví. Blíže bylo představeno téma stále populárnějších tzv. chytrých domácností, které mj. automatizují rutinní činnosti, a tím přinášejí i vyšší uživatelský komfort.

Druhá kapitola práce detailně popisuje významné technologie, které byly využity v praktické části. Kromě dalších důležitých technologických aspektů byl klíčovým bodem relativně nový framework .NET MAUI. Byl zdůrazněn jeho multiplatformní přístup, který umožňuje vyvíjet mobilní aplikace pro více operačních systému s využitím sdíleného kódu. Byly zmíněny přednosti a nové vlastnosti frameworku .NET MAUI vůči jeho předchůdci Xamarin.Forms.

Praktická část se ve své první kapitole zaměřuje na návrh webového aplikačního rozhraní a mobilní aplikace. V rámci kapitoly byl představen koncept aplikace, kde byl zdůrazněn komfort, který uživateli přinese. Kapitola stručně popisuje model databáze, se kterou komunikuje webové aplikační rozhraní. Pro mobilní aplikaci a webové aplikační rozhraní byly definovány funkční a nefunkční požadavky. V poslední řadě byl vytvořen návrh uživatelského prostředí mobilní aplikace.

Další část diplomové práce se věnuje tvorbě webového aplikačního rozhraní a mobilní aplikace, kterou popisuje čtvrtá kapitola. Při samotném vývoji bylo zjištěno, že některé části frameworku .NET MAUI v použité verzi jsou nestabilní a obsahují chyby ve standardním kódu. Tyto nedostatky měly za následek komplikace při implementaci, testování a ladění mobilní aplikace, což mohlo negativně ovlivnit kvalitu a spolehlivost výsledného produktu. Při vývoji však byly nalezeny způsoby, jak tyto chyby obejít a výsledkem je plně funkční mobilní aplikace pro řízení HVAC systému. Kapitola přináší poznatky o reálném procesu implementace s využitím frameworku .NET MAUI.

V poslední kapitole práce byl popsán průběh testování mobilní aplikace a její nasazení na fyzická zařízení. Tato fáze je klíčová pro ověření funkčnosti, spolehlivosti a uživatelského

komfortu aplikace před jejím uvedením do reálného prostředí. Součástí kapitoly je také popis procesu publikování mobilních aplikací pro operační systémy iOS a Android.

Celkově tato diplomová práce přináší nejen funkční mobilní aplikaci, ale také poznatky z praktického použití moderního frameworku .NET MAUI. Vytvořená mobilní aplikace poskytuje uživatelům pohodlný a efektivní způsob ovládání jejich klimatizačního zařízení.

POUŽITÁ LITERATURA

[1] MCEWEN, Adrian a Hakim CASSIMALLY. *Designing the Internet of Things*. United Kingdom: John Wiley, 2014. ISBN 978-1-118-43062-0.

[2] DIMITRIOS, Serpanos a Marilyn WOLF. *Internet-of-Things (IoT) Systems: Architectures, Algorithms, Methodologies* [online]. Switzerland: Springer International Publishing, 2018 [cit. 2023-03-10]. ISBN 978-3-319-69715-4. Dostupné z: https://doi.org/10.1007/978-3-319-69715-4

[3] TRIPATHY, B.K. a J. ANURADHA, ed. *Internet of Things (IoT): Technologies, Applications, Challenges and Solutions*. Boca Raton: Taylor & Francis, CRC Press, 2018. ISBN 978-1-138-03500-3.

[4] Industry 4.0 mining industry case: Dundee Precious Metals. *I-SCOOP* [online]. [cit. 2023-04-03]. Dostupné z: https://www.i-scoop.eu/internet-of-things-iot/industrial-internet-things-iiot-saving-costs-innovation/industrial-internet-mining-case/

[5] SELVARAJ, Sureshkumar a Suresh SUNDARAVARADHAN. Challenges and opportunities in IoT healthcare systems: a systematic review. *SN Applied Sciences* [online]. 2020, 2 [cit. 2023-03-24]. Dostupné z: https://doi.org/10.1007/s42452-019-1925-y

[6] PELTON, Joseph N. a Indu B. SINGH. *Smart Cities of Today and Tomorrow: Better Technology, Infrastructure and Security* [online]. Switzerland: Cham: Springer International Publishing, 2019 [cit. 2023-03-24]. ISBN 978-3-319-95822-4. Dostupné z: https://doi.org/10.1007/978-3-319-95822-4

[7] VANDOME, Nick. *Smart homes in easy steps: Master smart technology for your home*. United Kingdom: In Easy Steps Limited, 2018. ISBN 978-1-840-78825-9.

[8] HASSAN, Qusay F. *Internet of Things A to Z: Technologies and Applications*. New Jersey: Wiley-IEEE Press, 2018. ISBN 978-1-111-945674-2.

[9] MIELL, Ian a Aidan H. SAYERS. *Docker in Practice*. 2nd ed. Shelter Island: Manning Publications Co., 2019. ISBN 978-1-617-29480-8.

[10] Smart Home Security For Beginners. In: *Smart Home Works* [online]. Australia, 2018 [cit. 2023-04-03]. Dostupné z: https://smarthomeworks.com.au/2018/03/05/smart-home-security-for-beginners/

[11] What is .NET? Introduction and overview. *Microsoft Docs* [online]. 2023 [cit. 2023-04-03]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/core/introduction

[12] .NET. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-04-03]. Dostupné z: https://cs.wikipedia.org/wiki/.NET

[13] ŘÍHA, Pavel. Lekce 1 - Struktura prostředí .NET. *ITnetwork* [online]. Praha, 2022 [cit. 2023-04-03]. Dostupné z: https://www.itnetwork.cz/csharp/historie-net/net-struktura-prostredi

[14] ROTH, Daniel, Rick ANDERSON a Shaun LUTTIN. Overview of ASP.NET Core. *Microsoft Docs* [online]. 2022 [cit. 2023-04-04]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0

[15] Choose between controller-based APIs and minimal APIs. *Microsoft Docs* [online]. 2023 [cit. 2023-04-04]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/apis?view=aspnetcore-7.0

[16] What Is OpenAPI?. *Swagger Docs* [online]. 2023 [cit. 2023-04-04]. Dostupné z: https://swagger.io/docs/specification/about/

[17] What is .NET MAUI?. *Microsoft Docs* [online]. 2023 [cit. 2023-04-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-7.0

[18] Controls. *Microsoft Docs* [online]. 2023 [cit. 2023-04-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/?view=net-maui-7.0

[19] Layouts. *Microsoft Docs* [online]. 2023 [cit. 2023-04-04]. https://learn.microsoft.com/en-us/dotnet/maui/user-interface/layouts/?view=net-maui-7.0

[20] KATHIRESAN, Selva Ganapathy. Xamarin Versus .NET MAUI. *Syncfusion: Blog* [online]. 2022 [cit. 2023-04-04]. Dostupné z: https://www.syncfusion.com/blogs/post/xamarin-versus-net-maui.aspx

[21] Customize controls with handlers. *Microsoft Docs* [online]. 2022 [cit. 2023-04-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/handlers/customize?view=net-maui-7.0

[22] CHARBENEAU, Edward. Blazor Hybrid Web Apps with .NET MAUI. *Code Magazine* [online]. 2021 [cit. 2023-04-04]. Dostupné z: https://www.codemag.com/Article/2111092/Blazor-Hybrid-Web-Apps-with-.NET-MAUI

[23] Handlers. *Microsoft Docs* [online]. 2023 [cit. 2023-04-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/handlers/?view=net-maui-7.0

[24] RAMEL, David. Did .NET MAUI Ship Too Soon? Devs Sound Off on 'Massive Mistake'. *Visual Studio Magazine* [online]. 2022 [cit. 2023-04-04]. Dostupné z: https://visualstudiomagazine.com/articles/2022/09/29/net-maui-complaints.aspx

[25] GitHub: dotnet/maui [online]. 2023 [cit. 2023-08-03]. Dostupné z: https://github.com/dotnet/maui/issues

[26] What's new in .NET MAUI for .NET 7. *Microsoft Docs* [online]. 2023 [cit. 2023-04-04]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/whats-new/dotnet-7?view=net-maui-7.0

[27] NIENABER, Christoph a Rico SUTER. ASP.NET Core web API documentation with Swagger / OpenAPI. *Microsoft Docs* [online]. 2022 [cit. 2023-04-04]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-7.0

[28] Creating a Model for an Existing Database in Entity Framework Core. *Entity Framework Tutorial* [online]. 2020 [cit. 2023-04-13]. Dostupné z: https://www.entityframeworktutorial.net/efcore/create-model-for-existing-database-in-ef-core.aspx

[29] Entity Framework Core. *Entity Framework Tutorial* [online]. 2020 [cit. 2023-04-13]. Dostupné z: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx

[30] Entity Framework Core: DbContext. *Entity Framework Tutorial* [online]. 2020 [cit. 2023-04-13]. Dostupné z: https://www.entityframeworktutorial.net/efcore/entity-framework-core-dbcontext.aspx

[31] ROSSBERG, Joachim. *Agile Project Management with Azure DevOps: Concepts, Templates, and Metrics* [online]. United States: Apress Berkeley, CA, 2019 [cit. 2023-04-30]. ISBN 978-1-4842-4483-8. Dostupné z: https://link.springer.com/book/10.1007/978-1-4842-4483-8

[32] CORRIGAN, Andy. What is DevOps?. *Octopus Deploy* [online]. 2022 [cit. 2023-04-30]. Dostupné z: https://octopus.com/devops/

[33] Visual Studio App Center documentation. *Microsoft* [online]. 2023 [cit. 2023-05-01]. Dostupné z: https://learn.microsoft.com/en-us/appcenter/

[34] Remove unnecessary MeasureOverride from RefreshView: #11357. In: GitHub: dotnet/maui [online]. 2023 [cit. 2023-08-03]. Dostupné z: https://github.com/dotnet/maui/pull/11357

[35] Make CollectionView on iOS measure to content size: #14951. In: GitHub: dotnet/maui [online]. 2023 [cit. 2023-08-03]. Dostupné z: https://github.com/dotnet/maui/pull/11357

[36] Create a Signed and Publishable .NET MAUI Android App in VS2022 [video]. Gerald Versluis. 7. 2. 2022, 16:11 minut. [cit. 2023-07-15]. Dostupné z: https://www.youtube.com/watch?v=jfSVb_RR7X0&ab_channel=GeraldVersluis

[37] Updated: Create iOS Certificate Signing Request and .p12 file on Windows [video]. Courses by iBrent. 6. 6. 2017, 11:17 minut. [cit. 2023-07-15]. Dostupné z: https://www.youtube.com/watch?v=0iLtDb2ZKAE&ab_channel=CoursesbyiBrent

[38] Release Your .NET MAUI iOS App to the Apple App Store [video]. Gerald Versluis. 14. 2. 2022, 27:22 minut. [cit. 2023-07-15]. Dostupné z: https://www.youtube.com/watch?v=kpZi5xAvpZA&t=792s&ab_channel=GeraldVersluis

[39] DRYML, Jan. Návrh a implementace systému pro řízení vzduchotechniky [online]. Pardubice, 2022 [cit. 2023-08-03]. Dostupné z: https://dk.upce.cz/handle/10195/80438. Diplomová práce. Univerzita Pardubice. Vedoucí práce Ing. Jan Fikejz Ph.D.

PŘÍLOHY

Příloha A – Zdrojové kódy webového aplikačního rozhraní.....67
Příloha B – Zdrojové kódy mobilní aplikace.....68

PŘÍLOHA A – ZDROJOVÉ KÓDY WEBOVÉHO APLIKAČNÍHO ROZHRANÍ

K diplomové práci jsou přiloženy zdrojové kódy webového aplikačního rozhraní.

PŘÍLOHA B – ZDROJOVÉ KÓDY MOBILNÍ APLIKACE

K diplomové práci jsou přiloženy zdrojové kódy mobilní aplikace.