Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Návrh a implementace arduino modulu jako prvku systému smart home

Diplomová práce

Autor: Bc. Miloš Gábrle
Studijní obor: AI2

Vedoucí práce: prof. Ing. Ondřej Krejcar, Ph.D.

Hradec Králové srpen 2020

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 11.8.2020 Bc. Miloš Gábrle

Poděkování:

Děkuji vedoucímu diplomové práce prof. Ing. Ondřeji Krejcarovi, Ph.D. za metodické vedení práce. Také své rodině a přítelkyni za podporu během studia.

Anotace

Diplomová práce řeší téma chytré domácnosti, které se v poslední době těší stále větší popularitě. Analyzuje vědecké přístupy dané oblasti a existující komerční řešení. Z této analýzy vyvstaly požadavky na nový systém, který má za hlavní cíl snížení nákladů koncového uživatele. Čehož je dosaženo pomocí návrhu architektury, kde je server umístěn na zařízení mimo uživatelovu domácnost a pouze skrze API rozhraní a webovou aplikaci poskytuje služby, které umožňují uživatelům ovládat chytrou domácnost. Pomocí vybraných technologií je zde navrhnut a implementován celý systém od serveru a jeho databáze, až po koncový bod, který na základě instrukcí serveru ovládá zařízení v reálném světě.

Annotation

Title: Design and implementation of Arduino module as a part of smart home system

This master ‘s thesis is focused on smart home solution systems, which is currently hot topic in information technology. It analyzes scientific approaches and commercial solutions on this subject. New demands on the system arose from this analysis. Main goal of new system is to save costs for smart home users. Cost savings are mainly achieved by designing a new architecture, which has server already running on a computer, and is only providing services for controlling smart home with API and web application. There is designed the whole system, which consists of database, web application, API and the end point, with the help of selected technologies.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V dnešní době jsou moderní domy, již od počátku navrhovány a stavěny s myšlenkou úspory energie, a to jak při vytápění domu, tak i při jeho osvětlování v pozdějších denních hodinách. Tyto domy jsou tedy cíleně stavěny v prostředí, na které dlouho svítí slunce, je postaveno z moderních materiálů, které dobře izolují teplo uvnitř domu, okna jsou situována především na jižní stranu a tak dále. Všechna tyto vylepšení pak umožňují nízkonákladový provoz celého domu.

Dále moderní domy často obsahují řídící systémy, které umožňují automatizování některých kroků při udržování daného domu a tím tak nejen zvýšit pohodlí obyvatelů, ale dokonce potencionálně dále zvýšit úspory provozu domu. Je možné tak eliminovat neefektivní způsob vytápění domu, kdy například uživatel zapomene přiložit palivo do kotle, ten vyhasne a důsledkem celý dům vychladne a trvá pak delší dobu a stojí více paliva dům vyhřát zpět na obyvatelům pohodlnou teplotu.

Chytrý dům je tedy dům, obsahující hardwarové a softwarové prvky, umožňující výše zmíněnou automatizaci. Hardwarovými prvky rozumíme senzory, měřící momentální hodnoty dat (tepla, osvětlení atd.) v daném pokoji, ale také zařízení umožňující tyto hodnoty regulovat (relé pro zapínání elektrických součástek, elektrický motorek pro ovládání žaluzií, či termostatické hlavice na topení pro ovládání teploty pokoje). Softwarovými prvky rozumíme program, který rozhoduje na základě získaných hodnot a předem určených pravidel, jestli je třeba hodnoty regulovat, či nikoliv. Dále také program, který na základě těchto rozhodnutí řídí jednotlivé hardwarové prvky.

Chytrý dům možné řídit vzdáleně, skrze zabezpečený komunikační protokol, pomocí mobilní, nebo webové aplikace. Například si uživatel při delší absenci může ověřit, jestli někdo vstoupil do domu, jsou-li vypnuté všechny spotřebiče a po případě je vypnout. Dále tento systém může být použit k vytopení domu na požadovanou teplotu, ještě před samotným příjezdem.

2 Cíle a metodika práce

Cílem této práce je seznámit se s pojmem „chytrý dům“, analyzovat existující řešení dané problematiky. Na základě získané analýzy bude následně navrhnut nový systém s požadovanými vlastnostmi. Bude navržena nová vícevrstvá architektura umožňující komunikaci mezi jednotlivými prvky systému. Pro samotnou implementaci jednotlivých vrstev dané architektury budou vypsány dostupné technologie umožňující realizaci návrhu a z tohoto seznamu budou po porovnání vybrány konkrétní technologie, pomocí kterých bude implementován návrh dané vrstvy.

Implementace bude zaměřena bude především na realizaci softwarové části řešení, která má umožnit řídícím prvkům jejich správnou funkčnost a jeho využití alespoň jedním řídícím prvkem. Řídící prvek bude složen z mikro kontroléru od společnosti Arduino, společně s dalšími komponentami umožňujícími komunikaci s API serverem, měření potřebných dat a následnou regulaci měřené hodnoty na hodnotu požadovanou pomocí regulátorů a dalších elektronických komponent.

Bude také velice důležité aplikaci řádně zabezpečit a zamezit tak její zneužití neoprávněnou osobu, což by mohlo vést nejen k tomu, že by daná osoba mohla ovládat jednotlivé elektronické prvky daného domu, ale pokud by byl skrze aplikaci ovládán i zámek domu, tak by toto zneužití mohlo vést i k potenciální krádeži majetku.

Výstupem této práce bude tedy API server, webová aplikace pro nastavení uživatelských pravidel k ovládání jednotlivých kontrolérů a řídící modul využívající jednu z dostupných metod API serveru k ovládání určité veličiny chytré domácnosti.

3 Rešerše existujících řešení

Domácí automatizace je ve světě informačních technologií relativně žhavé téma, a to jak z hlediska zabezpečení domu, tak zvětšení pohodlí obyvatelů chytrého domu. Existuje tak mnoho vědeckých výzkumů a vzniká více a více reálných produktů, které se zabývají touto problematikou [1]. Oba přínosy pro chytré domy budou popsány níže v této kapitole.

3.1 Vědecké výzkumy

Článek [2] studuje existující řešení chytrých domů a všudypřítomné výpočetní technologie [3]. Zjišťuje, že systém bude vždy potřebovat nějakou interakci s uživatelem, například při úvodním nastavování systému, výběr z více možností nastavení, nebo potřeba potvrzení uživatelem k vykonání nějaké akce. Vývoj plně autonomního systému, který řídí dům pouze na základě analýzy chování uživatele, je vzhledem k počtu uživatelů a stále se vyvíjejícího trhu s technologiemi téměř nemožný. Jednou z navrhovaných možností, jak zefektivnit a zpříjemnit uživatelům jejich interakci se systémem, je použití adaptivního uživatelského rozhraní [4], které se přizpůsobuje uživatelským potřebám v závislosti na daném uživateli a čase. Dále pro adresování problému se stále se vyvíjejícími technologiemi navrhuje využití middleware jako mostu mezi novými hardwarovými prvky a navrhnutým systémem. V závěru navrhuje implementaci MyUI, URC [5] a OpenHAB [6] k řešení jmenovaných problémů.

Článek [7] představuje nový algoritmus, využívající strojového učení k určení návyků jednotlivých uživatelů a následnou reprodukci těchto návyků pomocí jednotlivých prvků chytrého domu. Tento algoritmus může zaznamenat i změny v návycích a následně upravit model, aby odpovídal i nově vzniklým návykům. Dalším možným využitím těchto dat je jejich analýza, kvůli možné indikaci například i zdravotních problémů uživatele (častější navštěvování toalety, často přerušovaný spánek atd.) a následné doporučení navštívení lékaře.

V článku [8] se autoři zabývají stále zvyšující se zátěží elektrické sítě v důsledku rostoucího množství elektrických spotřebičů v domácnostech. Požadavky domácností pomalu převyšují dostupnou kapacitu elektrické sítě, a tak

autoři navrhují systém, který přiřazuje jednotlivým spotřebičům čas, kdy budou aktivní a tím minimalizoval celkové nároky na elektrickou síť, a tak celou síť i stabilizoval.

V [9] autoři využívají senzorů chytrého mobilního telefonu k určování lidské aktivity jako je sedání, lehání, nebo běh. Jelikož v dnešní době má téměř každý mobilní telefon neustále u sebe, bylo by možné implementovat toto řešení i v chytrých domech. Například pokud by důchodce upadl a nezvedal se, šlo by spustit nějaké nouzové opatření, nebo kontaktovat záchranou službu.

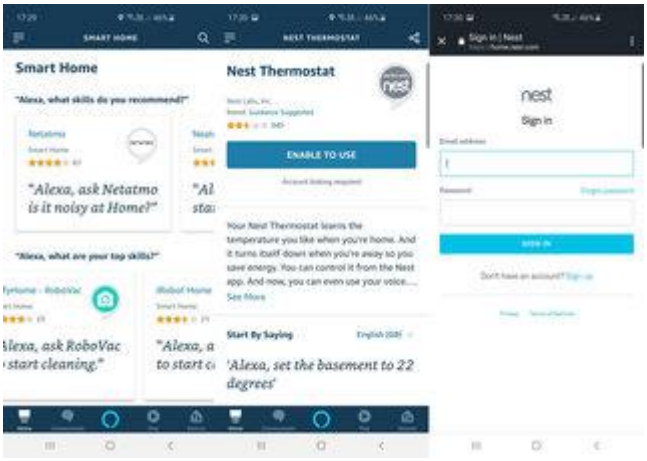
3.2 Existující řešení

Jelikož se jedná o rychle se vyvíjející trh, nabízející stále nové řešení od více a více firem, budou níže rozebrány pouze přední výrobci, kteří jsou dostupní v České republice.

3.2.1 Amazon Alexa

Je řešení využívající chytrého asistenta Alexa od společnosti Amazon. K jeho nastavení je třeba nejprve zprovoznit veškerá chytrá zařízení podle návodu dodaného výrobcem daných zařízení. Obvykle tato instalace zahrnuje zapojení do elektrického obvodu, nainstalování aplikace od výrobce, registrace na stránkách výrobce a přiřazení namontovaného zařízení do seznamu svých zařízení [10]. Jakmile jsou zařízení již nainstalovaná u výrobce, je možné tato zařízení přidat do systému společnosti Amazon, odkud je možné je ovládat.

O přidání nových zařízení a samotnou komunikaci mezi aplikací Alexa a chytrým zařízení se stará API s názvem *Smart home skills*, kde každý skill umožňuje komunikaci právě s jedním výrobcem chytrých zařízení. Aby byla možná komunikace je třeba nejprve vyhledat skill pro daného výrobce, přidat ho do své knihovny a následně se přihlásit pomocí přihlašovacích údajů, které jsme zadávali při instalaci chytrého zařízení u jejich výrobce. Obrázek 1 zobrazuje celý postup přidání a instalace skillu společně s vzhledem grafického rozhraní mobilní aplikace.



Obrázek 1: Přidání skillu do aplikace Amazon Alexa [10]

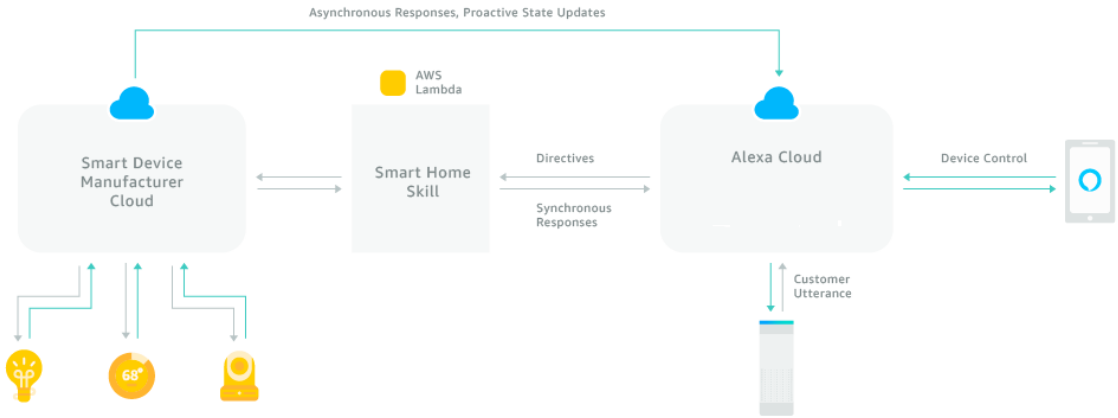
Jakmile budou všechna zařízení přidána, je možné editovat jejich název, rozdělit je do pokojů, ovládat jednotlivá zařízení pomocí aplikace, nebo hlasového asistenta, či definovat pravidla, určující jejich chování (například: „rozsviť světla v obývacím pokoji po západu slunce“).

Aby bylo možné podporovat výše zmíněné akce, tak *Smart Home Skill API* má následující vlastnosti [11]:

- **Schopnostní rozhraní** – Tyto rozhraní popisují funkcionalitu jednotlivých zařízení. Kde jednotlivá zařízení můžou implementovat pouze ty rozhraní, které nejlépe reprezentují jejich vlastnosti. Například světla, která jde zapnout, vypnout, i nastavit jejich jas budou implementovat rozhraní *PowerController* a *BrightnessController.*
- **Synchronní a asynchronní zasílání zpráv** – Komunikace mezi Alexou a koncovým zařízením může probíhat synchronně i asynchronně. Záleží na uživateli, který způsob mu více vyhovuje.
- **Notifikace o změně stavu zařízení** – Alexa je pomocí skillů informována i o změně stavu každého zařízení. Je tak možné uživatele o této změně informovat. Například uživatel může dostat upozornění po odemčení vchodových dveří.

- **Možnost dotazů** – Mimo sledování změny stavu je i možné se dotázat na aktuální stav zařízení. Uživatel si tak přímo v aplikaci může ověřit, zda zamkl, nebo zhasnul světla před odchodem z domu.

Obrázek 2 ukazuje, že pokud uživatel chce změnit stav nějakého zařízení, musí Alexu informovat o novém stavu pomocí aplikace, nebo hlasového asistenta. Například řekne asistentovi „nastav světla v kuchyni na 50 %“. Asistent následně sestaví direktivu pro každé světlo, které je přiřazené do místnosti s názvem kuchyň, obsahující autentizační token uživatele, identifikační číslo koncového zařízení a samotný příkaz k nastavení jasu na požadovanou hodnotu. Následně je direktiva odeslána synchronně či asynchronně výrobci, který změní stavy světel na hodnotu obdrženou pomocí příkazu v direktivě a odpoví Alexe o úspěšnosti změny stavu pomocí zprávy zvané *event.*



Obrázek 2: Komunikace mezi uživatelem a koncovým zařízením [11]

Tento způsob komunikace ovšem znamená, že API musí přímo podporovat komunikaci s výrobcem koncového zařízení, aby bylo možné dané zařízení ovládat. Systém Alexa se chlubí největším počtem kompatibilních zařízení oproti jeho přímým konkurentům. V případě Google Asistenta je počet pouze nepatrně vyšší, ale oproti Apple HomeKit má Alexa daleko větší počet zařízení, se kterými umí pracovat a celý seznam kompatibilních zařízení nalezneme na [12]. K dostání je také Amazon Echo, umožňující ovládání koncových zařízení pomocí hlasového asistenta.

3.2.2 Apple HomeKit

Apple oproti Amazonu přistupuje ke koncovým zařízením trochu jinak. Drží striktní kontrolu nad zařízeními, které autorizuje ke spolupráci se svým systémem. Tento přístup má za následek daleko menší počet kompatibilních zařízení, ale na druhou stranu umožňuje udržovat větší bezpečnost a lepší komunikaci mezi HomeKitem a koncovým zařízením [13].

Na rozdíl od předchozího řešení, není třeba u HomeKitu přidávat nové skilly a hledat v místní síti nová zařízení, stačí pouze naskenovat identifikační kód uvnitř balení chytrého zařízení pomocí fotoaparátu chytrého telefonu a aplikace přidá automaticky toto zařízení do chytré domácnosti uživatele, kde je dále možné zařízení rozřazovat do jednotlivých pokojů, ovládat je, nebo jim přiřazovat různá pravidla, kterými se budou zařízení řídit.

Stejně jako u řešení společnosti Amazon je zde možné zachytávat změny ve stavu jednotlivých zařízení a informovat o nich uživatele (například formou upozornění v mobilním telefonu).

Naprostou většinu úkonů nicméně není možné vykonávat bez tzv. *HomeKit Hubu*. Jako takový hub je možné využít některý z následujících produktů:

- **HomePod** – Je pravděpodobně nejčastější implementací, je neustále zapojený, a kromě hlasových příkazů pro domácnost, může fungovat i jako reproduktor pro přehrávání hudby.
- **Apple TV** – Pokud již uživatel vlastní Apple TV, není potřeba aby kupoval HomePod. Obdobně jako HomePod je stále zapojená. Jeho největší nevýhodou je fakt, že příkazy k ovládání domácnosti je možné zadávat pouze skrze ovladač.
- **iPad** – Jeho největší výhodou je mobilita zařízení, takže je možné ho volně přenášet a využívat ho k ovládání nejen pomocí hlasových příkazů, ale i skrze grafické rozhraní. Největší nevýhodou tohoto řešení je nicméně fakt, že samotný iPad je napájen z baterie a pokud uživatel zapomene iPad nabít, tak po jeho vypnutí uživatel ztratí veškeré výhody, které HomeKit Hub přináší.

Po nastavení hubu se uživatelovi odemkne spousta dalších možností zacházení s jeho chytrou domácností. Bude nyní možné ovládat chytrý dům i pokud uživatel není zrovna doma. Velkou výhodou je zde možnost vytvoření tzv. „scén“, což umožňuje uživateli definovat více akcí jediným příkazem. Například je možné vytvořit si scénu „pryč z domu“, která by uzamkla dveře, vypnula všechna světla připojená k chytré domácnosti a snížila kvůli šetření nákladů teplotu, kterou by termostat udržoval. Dále je možné využívat tzv. automatizací, které umožňují stejně jako v případě Alexy reagovat na vzniklé události. Například lze nastavit, že pokud dorazím domů (HomeKit Hub detekuje můj telefon doma), rozsvítí se světla na chodbě atd. [14].

I když je nabídka chytrých zařízení pro HomeKit oproti konkurenci značně omezená, tak většina typů produktů má alespoň jedno zastoupení, které je kompatibilní s HomeKitem. Uživatel tak může automatizovat téměř veškeré úkony podle svých představ, i když nebude mít tak širokou nabídku produktů. Seznam kompatibilních produktů lze nalézt v dokumentaci [15].

3.2.3 Fibaro

Fibaro [16] je systém od společnosti Fibar Group S.A. a je v České republice dostupná skrze několik distributorů, kteří nabízejí i kompletní instalaci do domu zákazníka. Uživatel si pro svou domácnost smí vybrat z následujících komponent:

- Chytrá zásuvka
- Detektor kouře, pohybu, vysoké hladiny oxidu uhličitého a zaplavení
- Intercom
- Termostatickou hlavici k topení
- Magnetické senzory na okna a dveře
- Moduly pro ovládání žaluzií a barevných LED pásků
- Stmívač světel
- Teplotní čidla
- Chytré zámky

Kompletní seznam všech komponent lze nalézt na stránkách výrobce. Jádrem celého systému je řídící jednotka Home Center, který komunikuje s jednotlivými komponentami pomocí standartu Z-Wave, což je bezdrátová komunikační technologie, která neoperuje na stejné frekvenci jako WiFi, či Bluetooth, tím pádem nedochází k narušení komunikace těmito technologiemi. Spojení je tak spolehlivější, ale nedokáže přenést tak velké množství dat jako WiFi. Proto lze Z-wave použít pouze v případech, kde se dá očekávat přenos menšího objemu dat, jako je získání dat ze senzorů a ovládání komponent, ale pro větší objemy dat, jako jsou záznamy z kamery s vysokým rozlišením, je třeba využít jiného standartu [17].

Centrální prvek Home center má momentálně dostupné 3 varianty:

- **Home Center Lite** – Je zaměřen na zákazníky s menším rozpočtem, umožňuje ovládat až 230 jednotlivých zařízení.
- **Home Center 2** – Je výkonnější než Home Center Lite. Umožňuje navíc i spojování více zařízení v jedno virtuální, například více termostatových hlavic u topení, lze nastavit pouze jako jedno zařízení a ovládat tak požadovanou teplotu všech najednou. Dále nabízí možnost vytváření scén obdobně jako v systému HomeKit od společnosti Apple.
- **Home Center 3** – Je obdobou systému Home Center 2, ale nabízí větší zabezpečení, je výkonnější a nabízí možnost komunikace s mobilním zařízením uvnitř domu pomocí Z-vawe i při výpadku internetu.

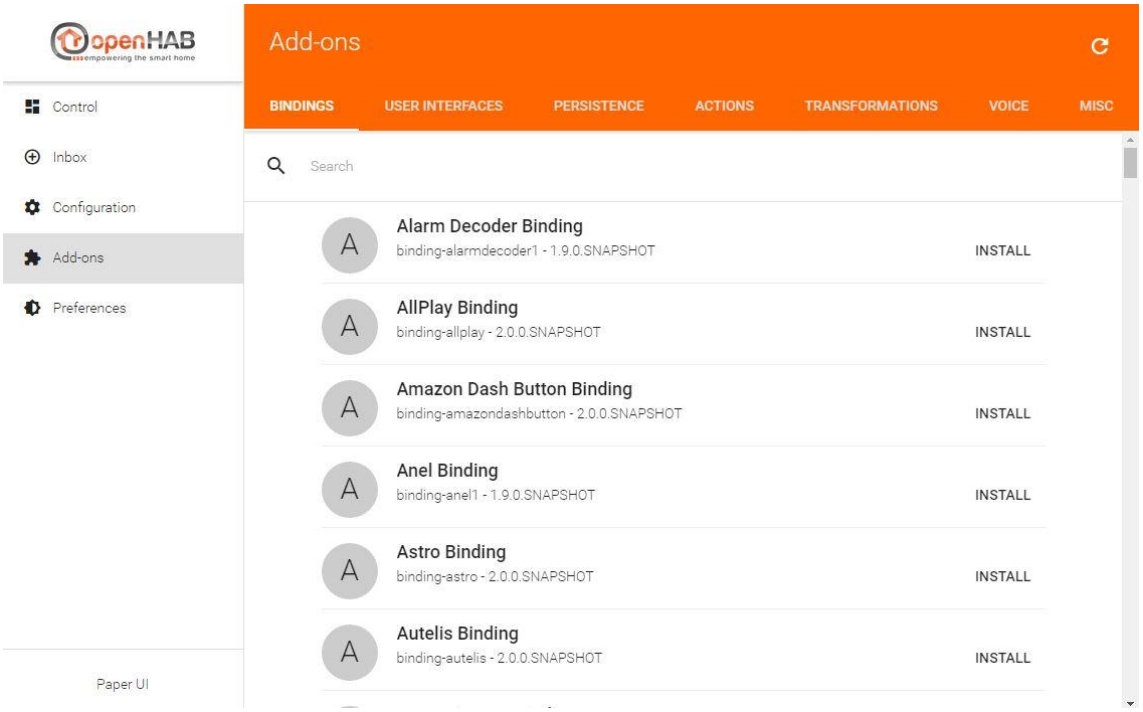
Co se týče kompatibility s komponentami ostatních výrobců, tak obdobně jako předchozí systémy, má tento systém pouze omezený seznam výrobků, které je možné integrovat do ekosystému Fibaro a jeho kompletní seznam lze nalézt na stránkách výrobce.

Pokud Uživatel nechce využívat žádnou verzi řídícího prvku Home Center, komponenty Fibaro jsou kompatibilní se systémem Apple HomeKit, a je tak možné k řízení komponent využívat systém společnosti Apple.

3.2.4 OpenHAB

Open Home Automation Bus (ve zkratce jen openHAB [18]), je volně dostupný software, který je určen k řízení chytrého domu, jehož možná implementace byla nastíněna i v článku [2]. Jeho cílem je poskytnout uniformní rozhraní pro celý systém, nehledě na počet výrobců použitých chytrých zařízení a počet subsystémů v domácnosti. Jedná se o program napsaný v programovacím jazyce Java [19], takže je možné ho nainstalovat i na systémy jako Linux, macOS, Raspberry Pi a další. Jednotlivé návody k instalaci je možné najít na stránkách výrobce.

Pokud již uživatel vlastní nějaké chytré zařízení, tak po nainstalování openHABu je zapotřebí pomocí tzv. *Paper UI* (Obrázek 3) přidat do programu tzv. *add-ony*, které umožňují komunikaci se zařízením a přidávání daného zařízení do systému. Dále si uživatel smí navrhnout své uživatelské rozhraní, které mu bude nejvíce vyhovovat či definovat pravidla, kterými se systém bude řídit, například „pokud je teplota v pokoji menší než 20 °C, zapni topení“.



Obrázek 3: openHAB – Paper UI [18]

Velkou výhodou openHABu je podpora komunikace s velkým množstvím chytrých zařízení. Add-ony ke komunikaci jsou nicméně spravovány komunitou, takže je možná i jejich nefunkčnost. Dále podporuje mnoho DIY („Do It Yourself“, v překladu „po domácku vyrobených“) zařízení, které ale vyžaduje více znalostí a větší míru porozumění než u komerčních řešení.

3.3 Analýza řešení

Ať už se jedná o využití adaptivního uživatelského rozhraní, nebo využití studie chování obyvatelů domácnosti k predikci jejich potřeb a jejich možnou realizaci ještě před tím, než uživatel zadá příkaz k jejich vykonání, tak tyto vědecké přístupy nerealizují kompletní řešení chytré domácnosti, pouze navrhují, jak je možné zlepšit určitou část celkového řešení. Nicméně v této oblasti přináší zajímavé poznatky, nad kterými se vyplatí zamyslet a uvažovat, jestli by nebylo možné tyto studie využít ke zlepšení systému, který se chystáme navrhnout.

Dříve byl velký problém komerčních řešení jejich úmyslná nízká kompatibilita s konkurencí. Uživatel si tak při rozšiřování své chytré domácnosti nemohl vybrat do svého systému takový řídící prvek, který by nejvíce vyhovoval jeho domácnosti, ale byl donucen zakoupit prvek od stejného výrobce, od jakého zakoupil zbytek systému. I když se dnes vývojáři systémů pro chytrou domácnost snaží rozšířit spolupráci s více firmami, tak aby se nabídka řídících prvků zvětšila, stále existuje mnoho prvků, které jsou se systémem nekompatibilní a nebudou správně, nebo vůbec, se systémem komunikovat. V případě Apple HomeKit je oproti konkurenci nabídka řídících prvků dokonce zlomková.

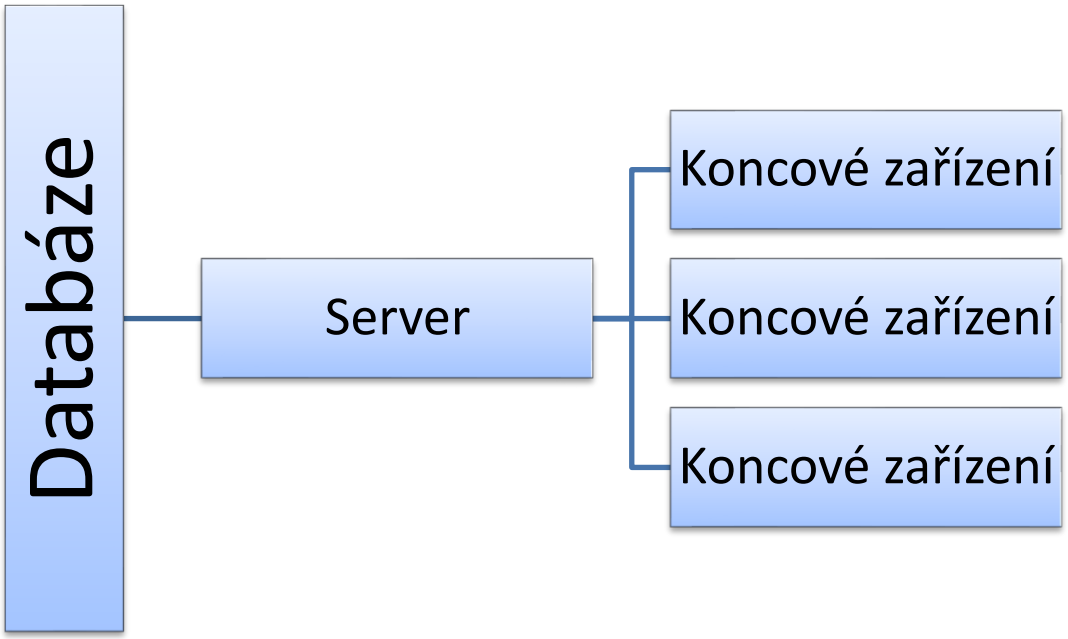
Pokud by ovšem uživatel chtěl ušetřit a navrhnout si vlastní řídící prvky, tak žádný ze systémů nepodporuje tyto DIY řešení, nebo je jeho implementace natolik složitá, že se i lidem, co si chtějí navrhnout vlastní řídící prvky, aby ušetřili, se více vyplatí je zakoupit hotové a jednoduše je nainstalovat dle návodu výrobce. Dále je třeba využít řídícího serveru, který bude ovládat všechny ostatní prvky. Lze využít existující řešení, které je ale nutné nainstalovat na počítač, který bude neustále zapnutý, což dále zvětší výdaje potřebné k realizaci řešení.

Je tedy třeba navrhnout systém pro chytrou domácnost umožňující implementaci DIY řídících prvků, a poskytnout jeho kompletní dokumentaci, aby byl

návrh vlastního řídícího prvku a jeho implementace v systému co možná nejjednodušší. Pro snížení nákladů bude centrální server běžet na externím zařízení a všem uživatelům poskytovat služby skrze API rozhraní. Uživatel tak nemusí pořizovat vlastní počítač, na kterém by server byl nainstalovaný.

4 Návrh nového systému

Z předchozí kapitoly vyvstaly základní požadavky na systém, pro jehož správnou funkcionalitu byla navržena architektura, kterou znázorňuje Obrázek 4.



Obrázek 4: Návrh architektury

Architektura je rozložena do následujících vrstev, kde každá plní konkrétní roli v celém systému:

- **Databáze** – Bude sloužit k ukládání uživatelských dat. Budou zde uchovávány informace týkající se chování koncových zařízení, na jejichž základě bude server rozhodovat o jejich zapnutí či vypnutí, uživatelské údaje, pomocí kterých bude možné určit, zda má uživatel právo ovládat konkrétní koncová zařízení, historii těchto změn a v neposlední řadě historii naměřených hodnot.
- **Server** – Je centrální prvek celého systému. Na základě dat uložených v databázi bude server skrze připravené API řídit chování všech koncových zařízení. Součástí serveru bude webová aplikace, pomocí které bude uživatel moci určovat základní pravidla, kterými se server bude řídit, spravovat uživatele, kteří budou moci ovládat jeho chytrý dům, či sledovat, jak příslušní uživatelé s jeho chytrou domácností

zacházejí. Server bude možné v budoucnu rozšířit i o API pro komunikaci s mobilními aplikacemi, a tak umožnit rozšíření systému o mobilní aplikaci. Tím i poskytnout vývojářům třetí strany možnost navrhnout vlastní mobilní aplikace, k ovládání chytré domácnosti.

- **Koncové zařízení** – Je nejnižší vrstva navrhnuté architektury. Jedná se o hardwarové zařízení, obsahující prostředky umožňující získávání dat z periferií, které jsou k němu přímo připojené. Jedná se o především o periferie sloužící k měření dat z prostředí v reálném čase (senzory), zpětné ovlivňování těchto hodnot (regulátory, světla, spínače apod.) a modul umožňující komunikaci s API serverové vrstvy.

Struktura jednotlivých vrstev společně s výběrem technologií k jejich implementaci budou rozebrány v kapitolách níže.

5 Výběr technologií

Pro realizaci systému je nejprve nutné vybrat technologie pro jednotlivé vrstvy navržené architektury, pomocí kterých budou implementovány.

5.1 Server

Jelikož je hlavním prvkem celé architektury server, bude vybrán nejdříve programovací jazyk a frameworky právě pro něho a následně budou vybrány technologie k vývoji ostatních vrstev navržené architektury s ohledem především na jejich celkovou kompatibilitu s vybranými technologiemi serveru.

5.1.1 Výběr programovacího jazyka

Jelikož bude kód volně dostupný, je zde vysoká pravděpodobnost, že uživatelé, kteří ho budou chtít využít, budou využívat různých platforem pro jeho nasazení. Je tedy zapotřebí vybrat takový programovací jazyk, který bude podporovat tzv. *cross-platform* nasazení systému. Jedná se o takový systém, který je po jeho sestavení schopný fungovat na různých operačních systémech jako je například Windows, macOS, nebo Linux [20]. Takových jazyků není dostupných mnoho a při výběru je podle [21] třeba brát v potaz hlavně následující možnosti.

Java byl podle [22] nejpoužívanějším programovacím jazykem v roce 2019 a je tomu tak prakticky nepřetržitě od jejího vytvoření v 90. letech. Využívá se v bankovnictví, robotice, mobilních telefonech a dalších. Je to mocný nástroj podporovaný rozsáhlou komunitou se širokou nabídkou různých frameworků, knihoven a dalších nástrojů usnadňující programování v Javě, kde je většina z nich volně dostupná ke stažení. Jedná se o objektově orientovaný programovací jazyk, který je přehledný, snadno udržovatelný a lehce rozšiřitelný. Je proto velice populární ve velkých řešeních. Ke spuštění programu napsaném v jazyce Java se využívá *Java Virtual Machine* (JVM). Kód je zkompilován do tzv. *Java byte kódu* a následně spuštěn pomocí JVM na požadovaném zařízení. Pokud tedy vybraná platforma podporuje JVM, je možné na ní spustit jakýkoliv kód zkompilovaný do Java byte kódu. Nicméně Java má poměrně velkou náročnost na zdroje (vyžaduje rychlejší hardware) a při vývoji například her, nebo aplikací pro zpracování video souborů, je třeba využít modulů navrženy v C++ a integrovat je do Java projektu.

C# je další populární objektově orientovaný programovací jazyk a přímý konkurent Javy. Samotný kód je kompilován do mezikódu s názvem *Common Intermediate Language* (CIL) a dále je interpretován do strojového kódu, který je spouštěn přímo na procesoru. Platforma .NET byla původně určena pouze pro Windows, časem byl založen projekt Mono [23] podporující vývoj v .NET na operačních systémech jako macOS, Linux, FreeBSD a další. Mono byl ale projekt založený komunitou, takže jeho podpora nebyla téměř srovnatelná s oficiální verzí .NET společnosti Microsoft. Roku 2016 přišlo řešení tohoto problému v podobě představení oficiální platformy .NET Core, která podporuje operační systémy Windows, OS X a několik Linuxových distribucí. .NET Core sdílí některé funkce společně s .NET Frameworkem, obsahuje navíc určitou funkcionalitu spojenou s možností nasadit aplikaci na více platforem, a naopak postrádá například některé funkce které existují výhradně pro operační systém Windows [24].

C++ je rozšíření programovacího jazyka C. Na rozdíl od jeho předchůdce je objektově orientovaný. Poskytuje oproti předešlým jazykům větší výkon pro aplikace na kterékoliv platformě, nabízí přímý přístup do RAM, CPU, GPU a jejich kontrolérů. Je to jazyk využívaný pro svou rychlost především k vývoji her, zpracování obrazu a videí, analýzu rozsáhlých dat a další. Nicméně samotné programování v tomto jazyce není tak jednoduché, jako s pomocí C# nebo Javy. Je zde třeba spravovat přiřazené zdroje, naučit se používání pointerů. Jelikož jazyk využívá globální proměnné a pointery, existují zde také určitá bezpečnostní rizika.

Python je další z objektově orientovaných programovacích jazyků podporující cross-platform nasazení, které je třeba zmínit. Jedná se o jazyk, který byl na žebříčku popularity za rok 2019 na 3. místě. Jedná se o relativně jednoduchý jazyk, se syntaxí zaměřenou na čitelnost kódu. Podporuje využití modulů, což znamená že jednou navržený modul může být použit ve více projektech a je velice populární v oblasti rychlého vývoje aplikací. Jedná se interpretovaný jazyk, takže před jeho spuštěním není potřeba jeho překlad [25]. Kód je ale čitelný, snadno zjistitelný a může být snadno přepsán, což může vést k zanesení chyb.

Z výše uvedených programovacích jazyků nám nejvíce vyhovuje Java a C#, kde oba nabízí obdobnou funkcionalitu. Vzhledem ke zkušenostem s jednotlivými jazyky

byl nakonec vybrán k implementaci serverové části systému C# a konkrétně platforma .NET Core. Jako vývojové prostředí bylo zvoleno Microsoft Visual Studio.

5.1.2 Výběr frameworků a podpůrných technologií

Pro zjednodušení a urychlení práce na vývojové části projektu je užitečné využít již existujících knihoven a frameworků, které jsou spravovány buď komunitou, nebo v případě Visual Studia i samotným Microsoftem. Samotné knihovny je možné přidat do našeho projektu pomocí správce balíčků s názvem *NuGet* [26]. Při spolupráci více vývojářů si NuGet hlídá, zda mají všichni nainstalované balíčky, které jsou v projektu přidány a chybějící balíčky automaticky přidává. Další velkou výhodou tohoto správce je fakt, že si vývojář zvolí k nainstalování pouze knihovnu, se kterou bude přímo pracovat a všechny další knihovny, které zvolená knihovna využívá, jsou nainstalovány a přidány do projektu automaticky.

5.1.2.1 Entity Framework

Pro správu databáze a komunikace mezi aplikací a databází bude do projektu přidán Entity Framework, konkrétně bude přidána odlehčená multiplatformní verze Entity Framework Core.

Dříve bylo potřeba napsat velké množství kódu například v *ADO.NET*, či *Enterprise Data Access Block,* pro získávání dat z databáze a převod získané kolekce dat do objektů, se kterými aplikace při jejich zobrazování uživateli umí pracovat. Obdobně pro ukládání dat do databáze je třeba data uložená v objektech převést na kolekci dat, kterým bude rozumět příslušná databáze a správně je uloží. To vše je možné vytvořit automaticky s pomocí Entity Frameworku.

Framework podporuje dva možné přístupy k vývoji.

- **Database First –** Pokud má vývojář již existující databázi, ke které pouze navrhuje aplikaci, je možné pomocí příkazu *Scaffold-DbContext* do *package manager console*. Framework následně vytvoří modely do složky, kterou si vývojář určí společně se třídou zvanou DbContext, která zprostředkovává převod dat mezi aplikací a databází, zároveň umožňuje získávání a ukládání dat pro aplikaci. Dále uchovává informace jako jsou

různá datová omezení (např. maximální délka řetězce, povinná data a další). Ukázku konkrétního příkazu lze najít na [27].

- **Code First –** Vývojář nejprve musí navrhnout tzv. model databáze, což jsou třídy, které obsahují atributy. Třídy odpovídají jednotlivým tabulkám v databázi a atributy odpovídají jednotlivým sloupcům v dané tabulce. Dále je třeba přidat třídu, která bude dědit od třídy DbContext a bude obsahovat seznam modelových tříd, které budou mapovány na relační databázi, společně s nastavením připojení do konkrétní databáze. Datová omezení lze definovat jak v třídě DbContext, tak s pomocí anotací, které se v jazyce C# píšou do hranatých závorek nad daný atribut. Ukázku DbContextu společně s modelovací třídou lze vidět v ukázkovém kódu níže [28].

```
using Microsoft.EntityFrameworkCore;
using System.ComponentModel.DataAnnotations;

namespace EFModeling.DataAnnotations.Required
{
    class MyContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        [Required]
        public string Url { get; set; }
    }
}
```

Jelikož tato aplikace nevyužívá již existující databáze tak byl vybrán přístup code first. Tento přístup umožňuje spravovat databázi přímo z vývojového prostředí, ve kterém bude psána celá aplikace, tak se eliminuje potřeba psát skripty pro vytvoření databáze, stačí upravit modelovou třídu a aplikovat změny, jak je uvedeno níže. Entity Framework podporuje různé databázové poskytovatele, stačí stáhnout balíček z NuGet, který podporuje námi zvolenou databázi, nastavit typ

databáze v automaticky vygenerované třídě *Startup.cs* a Entity Framework následně bude vytvářet skripty v jazyce, který odpovídá zvolené databázi.

Během vývoje aplikace se model databáze často mění, a tak je třeba vždy synchronizovat model s databází. Je možné databázi aktualizovat jejím smazáním a nahráním nové verze, to ale vede ke ztrátě všech dat a je tím pádem nepřípustné. Entity Framework zajišťuje přechod mezi jednotlivými verzemi databáze pomocí tzv. migrací. Po každé změně v modelu je třeba přidat novou migraci pomocí příkazu *Add-Migration* a po vyzvání zadat název nové migrace, což vygeneruje následující soubory:

- soubor s metadaty pro Entity Framework
- momentální verzi modelu (aby bylo možné v další migraci určit, co se oproti předchozí verzi změnilo)
- soubor se skripty jak pro úpravu tabulky ze staré na novou, tak z nové na starou

Pokud chceme například u nového sloupce nastavit u existujících dat nějakou výchozí hodnotu, je třeba upravit vygenerované skripty. Následně stačí zadat do konzole příkaz *update-database* a Entity Framework spustí připravené skripty na připojené databázi [28].

5.1.2.2 LINQ

Entity Framework dále podporuje tzv. *Language Integrated Query (LINQ)*. Což je nástroj, umožňující využívat jazyka C# pro vytváření dotazů na databázi. V LINQ se vždy pracuje s modely. To znamená, že vývojáři píšou svůj dotaz vždy stejně a nemusí se starat o intepretaci pro jednotlivé typy databází (XML dokumenty, SQL databáze, ADO.NET data sety atd.). To vše za ně udělá automaticky framework LINQ.

Samotný dotaz je v LINQ pouhý kontejner k uchovávání informací, jak požadovaná data vyfiltrovat, seřadit či v jakém tvaru je prezentovat. Dotaz tedy neprovádí žádnou akci, ani nevrací žádná data, až do chvíle, kdy je nad získanou kolekcí dat iterováno později v kódu. Je tedy možné skládat dotaz a přidávat do něho podmínky na více místech v kódu. Tento způsob volání dat se nazývá *odložené provedení.* Je možné donutit LINQ k okamžitému provedení kódu tím, že se na konci

dotazu zavolá některá z funkcí *Count, Max, Average, First,* nebo *ToList* [29]. Na kódu níže, lze vidět dotaz, který získá z databáze všechny blogy, které v URL obsahují řetězec „dotnet“.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Where(b => b.Url.Contains("dotnet"))
        .ToList();
}
```

V praxi lze tuto funkci využít například pokud uživatel chce filtrovat data s pomocí více filtrů, lze jednoduchými podmínkami zajistit, aby se neprázdné filtry přidaly k dotazu, dále je možné data seřadit, či pomocí metod *Skip* a *Take* rozdělit na stránky pro snížení náročnosti na výkon a zkrácení času, který je potřeba k získání požadovaných dat.

5.1.2.3 Identity Framework

Jedná se o framework, který poskytuje webové aplikaci základní přihlašovací funkcionality. Spravuje uživatele, hesla (včetně šifrování), uživatelské role, potvrzovací emaily a další. Veškeré informace se ukládají v databázi a jsou i vývojářům nedostupné, nicméně je možné tabulku v databázi upravit tím, že v databázovém modelu navrhneme novou třídu, která bude dědit od třídy s názvem *IdentityUser*. V nové třídě bude možné přidat další informace, které chceme o uživateli uchovávat. Dále je možné využít externích přihlašovacích providerů jako jsou Facebook, Google, Microsoft Account a Twitter.

Pro přidání Identity Frameworku do projektu je třeba při zakládání projektu vybrat položku *Individuální uživatelské účty* v záložce s autentizací a následně Visual Studio vygeneruje nový projekt se šablonami připravenými k autentizaci a autorizaci uživatelů [30].

5.1.2.4 Bootstrap

Je nejpoužívanější CSS framework, používaný k vývoji responsivních webových aplikací. Je volně dostupný ke stažení, optimalizovaný i pro mobilní zařízení a umožňuje vývojářům psát profesionálně vypadající aplikace.

Obsahuje vlastní rozložení, které je možné jednoduše využít. Je možné přímo v něm definovat obrázky, které se budou automaticky přizpůsobovat velikosti displeje. Má předpřipravené komponenty jako jsou tabulky, tlačítka, menu, ukazatele průběhu a další, které lze jednoduše použít. Má několik vlastních JQuery pluginů, které umožňují využít různých předpřipravených animací a přechodů k zatraktivnění vzhledu aplikace. Díky popularitě frameworku bylo také vyvinuto mnoho atraktivních šablon, které lze při vývoji aplikace využít [31].

5.1.3 Výběr komunikačního rozhraní

Jelikož rychlost komunikace mezi centrálním serverem a jednotlivými koncovými body bude společně s frekvencí dotazů na server do velké míry ovlivňovat, jak responsivní celý systém bude. Je třeba vybrat takový způsob komunikace, který bude dostatečně rychlý a spolehlivý.

Na serveru bude připravené rozhraní API, poskytující metody k řízení chytrého domu, na které budou koncové body zasílat své dotazy společně s naměřenými hodnotami a jejímiž odpověďmi se budou řídit. Jelikož jsme již zvolili jako programovací jazyk serveru C#, máme na výběr z následujících protokolů.

5.1.3.1 TCP

Neboli *Transmiting Control Protocol*, je protokol popisující sadu pravidel a procedur pro přenos dat mezi zařízeními po datové síti. Definuje, jak mají být data rozdělena do paketů, adresována, přenášena, směrována a obdržena na cílovém zařízení. Jedná se o zabezpečený protokol, který sleduje stav odeslaných paketů a zaručuje jejich doručení [32]. Může být tedy využit pro přenos dat mezi serverem a koncovými body v systému chytrého domu.

Pokud chceme navrhnout TCP server, v .NET Core jej lze definovat pomocí tzv. TCP Listeneru [33], kde je třeba definovat formát zpráv, které budou

vyměňovány při komunikaci s klientem a napsat dekodér, který bude holá data překládat do objektů, se kterými by mohl server dále pracovat.

Jelikož se zde definuje konkrétně co a jak se má přenášet, nedochází k přenosu nepotřebných dat a tento protokol má tak potenciál být při vhodném návrhu velice rychlý. Klienti nicméně musí formátovat svá data tak, aby odpovídali navrženému způsobu komunikace, takže při jakékoliv změně na straně serveru, se musí automaticky opravit i odesílaná data od klienta. Zároveň může být dekodér špatně optimalizován a tím tak rapidně zpomalit celkovou rychlost celé komunikace.

I když má protokol vysoký potenciál být velice rychlý, z důvodů nutnosti návrhu vlastního formátu a možnosti zanesení chyb, které by zbytečně zpomalovali celý systém, byl protokol TCP ve svém čistém stavu zavrhnut.

5.1.3.2 XML-RPC

Extensible Markup Language – Remote Procedure Call (XML-RPC) byl navržen s úmyslem umožnit komunikaci mezi zařízeními na různých operačních systémech a po prvé implementován v roce 1998. Slouží ke vzdálenému volání procedur. Jedná se o relativně jednoduchý protokol, kde klient zakóduje dotaz na server do XML souboru, ten pomocí HTTP protokolu odešle na server, ten dotaz zpracuje a svou odpověď opět zakóduje do XML souboru, který je odeslán klientovi [34]. Příklad takového dotazu na server je vidět na v ukázkovém kódu níže. Je zde také na první pohled patrné, že k zakódování dotazu je potřeba relativně velké množství informací určujících význam odesílaných dat a samotná data v celém souboru zabírají pouhý zlomek místa. Odpověď serveru klientovi je zakódována obdobně, tím pádem jsou soubory zbytečně veliké a celý proces komunikace je přenosem více dat zpomalený.

```
<?xml version="1.0">
<methodCall>
 <methodName>getUserAllDetails</methodName>
 <params>
  <param>
   <value>
    <string>alec</string>
   </value>
  </param>
 </params>
</methodCall>
```

K implementaci XML-RPC v .NET Core jsou skrze NuGet dostupné knihovny, jako například *Kveer.XmlRPC.*

V dnešní době rychlého internetu není větší objem dat až takový problém, ale samotný protokol je zastaralý, není tak podporován jako následující protokoly a byl proto také zamítnut.

5.1.3.3 SOAP

Celým názvem *Simple Object Acces Protocol*, je protokol specifikující výměnu strukturovaných dat mezi zařízeními, která by mohla mít heterogenní systémy. Obdobně jako v předchozím případě využívá XML souborů zasílaných skrze HTTP protokol. Do nedávna byl SOAP považován za standart webových služeb. Definuje formát zpráv zasílaných pomocí tohoto protokolu. Každá zpráva musí být zabalena v párovém tagu *<envelope>*, který je dále dělen na hlavičku a tělo pomocí tagů *<header>* a *<body>*. Tělo zprávy dále může obsahovat tag *<fault>*, do kterého se zapisují informace o případné chybě, která by mohla vzniknout špatným dotazem na server, nebo chybou v serverové části systému [35].

Obdobně jako v předchozích případech, i zde je možné využít dostupných knihoven k implementaci SOAP protokolu v našem projektu, které jsou volně dostupné pomocí NuGet.

Struktura dokumentů zasílaných pomocí SOAP je daleko komplexnější než v případě volání procedur pomocí XML-RPC a nabízí více možností zabezpečení zpráv, jako je například WS-Security. SOAP dále podporuje využití WSDL, který popisuje funkce celé API služby, jako jsou názvy funkcí, typy jejich parametrů a návratové hodnoty.

Nicméně protokol stále využívá zasílání XML dokumentů, což znamená větší objem dat k přenosu. Jelikož je zde objem dat kvůli přidaným funkcím ještě větší než v případě XML-RPC, byl protokol SOAP i přes své výhody také zamítnut.

5.1.3.4 REST

Representational State Transfer (REST), je API využívající HTTP požadavků pro práci s daty. Využívá existující metodologie HTTP protokolu, jako využívání jednotlivých typů volání ke konkrétním *CRUD* (vytvoření, upravení a mazání) operacím s daty a jsou rozdělena následovně:

- POST – vytvoření nových dat
- PUT – úprava existujících dat
- DELETE – odstranění existujících dat

Celý protokol je tzv. *"bez stavový"*, což znamená, že neexistuje žádný záznam předchozí interakce a každý požadavek musí být zpracován čistě na základě informací, které jsou společně s ním odeslány [36].

REST API bylo navrženo již roce 2000 jako část dizertační práce Dr. Roye Fieldinga [37], kde ve své páté kapitole popisuje několik základních omezení, která musí webová služba splňovat, aby mohla být považována za REST API. Zmíněná omezení jsou následující:

- **Klient-server** – Jednotlivé strany by měly být vyvíjeny nezávisle na sobě. Oddělením funkcionality zobrazení dat a práce s daty získáme

možnost systém dále jednoduše rozšířit bez ovlivňování druhé strany, a zvětšujeme portabilitu uživatelských rozhraní na více platforem.

- **Bez stavová** – Jak bylo zmíněno již výše, aby bylo dosažené tohoto požadavku, musí každý požadavek obsahovat veškeré informace, které jsou nezbytné k jeho správnému zpracování.
- **Cache** – S cílem zajistit vyšší účinnost počítačové sítě je třeba přesně určit, jestli je možné data ukládat do cache a pokud ano, tak je klientovy uděleno právo data z cache znovu načíst.
- **Uniformní rozhraní** – Mezi jednotlivými komponentami by mělo existovat uniformní rozhraní, umožňující komunikaci se všemi klienty. Podporující využití na více platformách.
- **Vrstvený systém** – Celý systém by měl být rozdělen do několika hierarchických vrstev a jednotlivé by neměli z bezpečnostních důvodů vidět dále než na komponenty, se kterými přímo pracují. Tím se podporuje možnost rozšíření a znovupoužití již navržených komponent.
- **Kód na vyžádání** – Na rozdíl od předchozích protokolů, kde se data zpět zasílají pouze v XML, či JSONu (JavaScript Object Notation), umožňuje REST zasílat i spustitelný kód klientovi (například ve formě appletů, či skriptů), což umožňuje větší rozšiřitelnost a možnost využití například větší funkcionality pro firemní zařízení, díky stáhnutým Java třídám.

Jelikož se jedná o velice populární implementaci webových služeb, je ve vývojovém prostředí přímo nabízeno několik základních šablon pro vytvoření REST API a .NET Core přímo jej přímo podporuje, bez nutnosti stahování externích balíčků.

Samotné API je velice dobře podporováno i na mobilních zařízeních, a tak bude v budoucnu jednoduché rozšířit celý systém i o mobilní, či desktopovou aplikaci.

Díky všeobecné podpoře tohoto rozhraní napříč platformami, jeho celkové jednoduchosti, rozšiřitelnosti a faktu, že není třeba mnoho režijních dat k přenosu dat potřebných, bylo jako základní rozhraní pro server zvoleno REST API.

5.2 Databáze

Celý systém bude ukládat velké množství informací, obdržených skrze REST API od jednotlivých koncových bodů, dále informace týkající se jednotlivých uživatelů a jejich chytrých domů. Je tedy třeba vybrat vhodnou databázi, která bude správně pracovat společně s navrženým serverem, k ukládání a následné práci se získanými daty.

Jelikož jsme v předchozí kapitole k práci s daty zvolily na práci s daty Entity Framework, což je nástroj pro mapování objektových dat do relačních databází, je třeba zvolit SQL relační databázi, která s ním bude kompatibilní. Podle dokumentace [38], máme na výběr z poskytovatelů jako jsou Oracle, Microsoft SQL, MySQL a dalších. Jelikož Entity Framework Core ze základu podporuje Microsoft SQL databázi a není tak třeba dále nic měnit, či stahovat dodatečné knihovny, byla pro server zvolena právě Microsoft SQL databáze.

5.3 Koncové zařízení

Pro jednoduchost implementace byla vybrána k návrhu koncových zařízení platforma Arduino [39]. Jedná se o volně dostupnou elektronickou platformu, obsahující desku s mikrokontrolerem a jednoduché vývojové prostředí, umožňující programování zvolených hardwarových prvků.

Jak uvádí samotný výrobce, důvody, proč zvolit právě Arduino můžou být následující:

- **Nízkonákladovost** – Oproti ostatním mikrokontrolerům je Arduino relativně levné.
- **Cross-platformní** – Jeho vývojové prostředí může být spuštěno na Windows, Macintosh OSX i na Linuxových operačních systémech. Takže uživatel není omezen platformou, na které chce mikrokontroler naprogramovat.
- **Jednoduché vývojové prostředí** – Je jednoduchý, aby přilákal začátečníky a zároveň flexibilní, čehož můžou využít pokročilí programátoři.

- **Volně dostupná dokumentace k hardware i software** – Veškerý hardware a software má svou dokumentaci, která je publikována na jejich stránkách. Takže si pokročilí návrháři desek mohou navrhnout svou vlastní verzi modulu, který potřebují. Software je volně dostupný a pokročilí programátoři si ho mohou upravit a rozšířit dle vlastních potřeb skrze C++ knihovny.

5.3.1 Výběr desky

Arduino nabízí několik variant svých desek. Má zde na výběr z několika procesorů, které se liší svou frekvencí, velikostí, či vstupním napětím. Dále nabízí několik možností počtu vstupů a výstupů, velikostí své Flash paměti a v neposlední řadě se jednotlivé desky liší i svou celkovou velikostí. Tabulka 1 popisuje několik základních desek, společně s jejich vlastnostmi.

Name	**Processor**	**CPU Speed**	**Analog In/Out**	**Digital IO/PWM**	**Flash [kB]**
Mega 2560	ATmega2560	16 MHz	16/0	54/15	256
Micro	ATmega32U4	16 MHz	12/0	20/7	32
Uno	ATmega328P	16 MHz	6/0	14/6	32
Zero	ATSAMD21G18	48 MHz	6/1	14/10	256
Due	ATSAM3X8E	84 MHz	12/2	54/12	512
Ethernet	ATmega328P	16 MHz	6/0	14/4	32
Leonardo	ATmega32U4	16 MHz	12/0	20/7	32
Mega ADK	ATmega2560	16 MHz	16/0	54/15	256
Mini	ATmega328P	16 MHz	8/0	14/6	32
Nano	ATmega168 ATmega328P	16 MHz	8/0	14/6	16 32
Yùn	ATmega32U4 AR9331 Linux	16 MHz 400MHz	12/0	20/7	32 64MB

Tabulka 1: Seznam nejpoužívanějších Arduino desek [39]

Existují i desky na specializované účely, jako například Arduino LilyPad, zaměřené na implementaci v textilním průmyslu, nebo Arduino Yùn, který krom klasického čipu ATmega32U4 obsahuje také procesor Atheros AR9331, který společně s porty umožňující komunikaci s dalšími periferiemi, umožňuje využití operačního systému Linux v našem projektu. Kompletní tabulku všech dostupných desek, společně s jejich parametry, lze nalézt na stránkách výrobce [39].

Jelikož cílem práce je vytvořit platformu pro po domácku vyrobená koncová zařízení. Každý uživatel si musí desku vybrat sám s ohledem na své vlastní požadavky. Deska by měla mít dostatečný počet portů, být odpovídající velikosti a mít dostatečný výkon k vykonání úkonů, ke kterým byla navržena. Je dobré také počítat s možností rozšíření koncového bodu o další periferie do budoucna.

5.3.2 Komunikace se serverem

Aby vše fungovalo, jak má, je třeba aby se koncové zařízení při úvodním spuštění registrovalo unikátním klíčem v systému a předal o sobě informace jako je jeho typ. Každé koncové zařízení tedy ponese unikátní identifikátor, pomocí kterého se registruje v systému a díky kterému ho bude moci správce přidat do své chytré domácnosti, kde jej bude moci ovládat.

Každé koncové zařízení bude muset odesílat data serveru, který je zpracuje a na základě těchto dat a předepsaných pravidel pro dané koncové zařízení rozhodne o jeho následném chování. Je tedy třeba jej navrhnout tak, aby byla možná komunikace se serverem skrze síť internet. Je nezbytné nejprve zvolit periferie umožňující komunikaci Arduino modulu s okolním světem. Pro tuto funkcionalitu máme na výběr z následujících variant:

- **Ethernet –** První možností je ethernetový kabel. Takovéto připojení je velice rychlé, a není potřeba náročné konfigurace k samotnému připojení se směrovačem. Každé koncové zařízení by muselo být připojeno k síťovému přepínači pomocí kabelu složeného ze čtyř párů kroucené dvoulinky. Rychlost, které je možné dosáhnout na tomto médiu je podle standartu *IEEE 802.3ae* až 10 Gbps [40]. I když maximální

rychlost, které je možné dosáhnout je velice lákavá, každé koncové zařízení by muselo být spojeno s rozbočovačem kabelem. Pokud vezmeme v potaz, že každý pokoj bude mít své vlastní koncové zařízení ovládající osvětlení, v domácnosti o 6 pokojích by bylo potřeba 6 kabelů a rozbočovač s odpovídajícím počtem portů pouze na osvětlení. Pokud bychom chtěli systém dále rozšířit o koncová zařízení ovládající topení, zámek domu, nebo jiné, počet kabelů v domácnosti by rapidně stoupl. Rozsáhlá kabeláž by zabrala spoustu místa a byla by u takto rozsáhlých systémů nákladná. Zároveň se jedná o rychlost pouze teoretickou, jelikož tato rychlost je limitována rychlostí přenosu dat mezi uživatelem a jeho poskytovatelem internetu. Pokud by uživatel měl ovládací server a jednotlivá koncová zařízení na jedné domácí síti, bylo by možné při použití vhodných technologií dosáhnout dříve zmíněné maximální rychlosti.

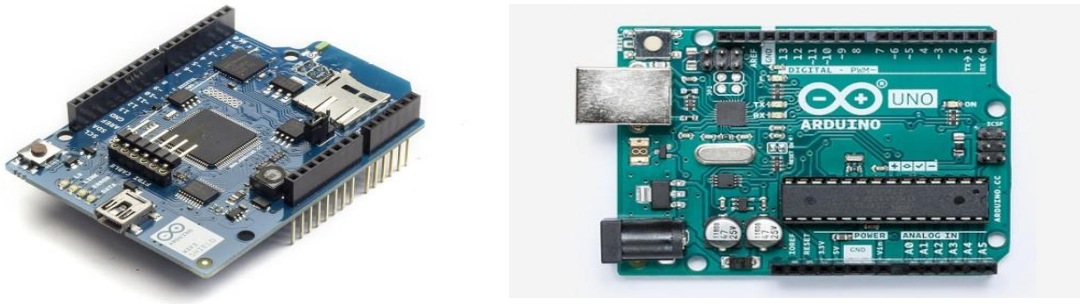
- **Wi-Fi –** Je bezdrátová technologie umožňující komunikaci mezi zařízeními. Řídí se standardem *IEEE 802.11*, který definuje specifikace pro bezdrátový přenos dat po lokální síti. Přenos dat využívající tohoto standartu probíhá nejčastěji na frekvencích 2.4 GHz a 5 GHz, i když například standart *IEEE 802.11ad* definuje přenos dat pomocí frekvence až 60 GHz, s teoretickou maximální přenosovou rychlostí okolo 6Gbps [41]. Hlavní výhodou této technologie je eliminace potřeby propojení všech zařízení pomocí kabelů. Tím značně klesnou náklady na komunikační médium, ale je třeba zakoupit router, s Wi-Fi vysílačem, který je v dnešní době přítomen v téměř každé domácnosti a jednotlivá koncová zařízení vybavit moduly, které jsou schopné komunikovat po bezdrátové síti. Hlavní výhodou je tedy ušetření nákladů na kabeláži, jednodušší instalace a ušetření prostoru, který by byl vytvořenou kabeláží zabrán. Nevýhodou je pak možný odposlech signálu, je tedy třeba data řádně šifrovat a zabezpečit.
- **XBee –** Je technologie, která umožňuje zařízením s čipem XBee komunikovat bezdrátově pomocí radiových frekvencí. Celá komunikace je definována standardem *IEEE 802.15.4* a je zaměřený na nízké náklady

a nízkou spotřebu. Nicméně tyto zaměření mají za příčinu nízkou přenosovou rychlost a relativně nízkou vzdálenost, na kterou je možné data přenášet. Zařízení je možné mezi sebou spojit pomocí hvězdicové, či mesh topologie. Obdobně jako Wi-Fi má i XBee několik verzí, které se liší jak přenosovou rychlostí, tak například jednoduchostí jejich implementace v systému, pokud spolu čipy mají správně komunikovat je nezbytné je mít nastavené na stejnou radiovou frekvenci. Teoretická maximální rychlost této technologie se nicméně pohybuje pouze okolo 250 kbps, ale jelikož je zaměřená na nízkou spotřebu a náklady, rychlost přenosu dat těchto čipů se obvykle pohybuje kolem 20, či 40 kbps [42].

XBee je technologie, která sice nabízí mnoho výhod, ale jejich dosah je velice malý (oproti Wi-Fi, která má na frekvenci 2,4 GHz dosah ve vnitřních prostorách okolo 50 m a ve venkovních prostorách až 90 m). Dále umožňuje pouze komunikaci mezi zařízeními, která vlastní XBee moduly a jelikož je server vzdálený od celého systému, je tato technologie pro navrženou architekturu nevhodná. K zprovoznění by bylo třeba přidat novou vrstvu do navržené architektury systému, která by zprostředkovávala komunikaci nižší částí systému (jednotlivými koncovými zařízeními) a vzdáleným serverem. Ethernet naopak eliminuje nebezpečí odposlechu signálu, rozsáhlejší systém by potřeboval velice rozsáhlou kabelovou síť, což by zvýšilo náklady na systém a stejně tak jeho praktičnost, jelikož by tyto kabely zabraly velkou část obývacích prostor.

Ze všech výše zmíněných důvodů byla tedy pro komunikaci zařízení se serverem zvolena technologie Wi-Fi. K připojení Arduino modulu k domácí Wi-Fi síti je zapotřebí připojit patřičný modul k námi vybrané desce Arduino.

První z možných řešení přidání Wi-Fi kompatibility deskám Arduino je například *Arduino WiFi Shield* [43]. Arduino Shield je označení pro modulární desky, které je možné přímo připojit k vybraným Arduino deskám a tím tak rozšířit jejich základní funkcionalitu. Arduino WiFi Shield lze jednoduše připojit k Arduino Uno přiložením pinů shieldu k odpovídajícím portům Arduino Una a zamáčknout, dále je třeba ho pouze naprogramovat. Na Obrázek 5, je viditelné, že po instalaci shieldu k původní desce, neztrácíme přístup k žádným portům, neboť jsou přímo vyvedeny na odpovídající porty shieldu a získáme navíc funkcionalitu daného shieldu. Dále je velkou výhodou možnost skládání jednotlivých shieldů na sebe. Pokud je například potřeba přidat funkcionalitu přehrávání zvuků, stačí zvukový shield připojit na vrch WiFi shieldu. Možné rozšíření koncového zařízení o další funkcionalitu je tedy velice jednoduché, ale je potřeba si dávat pozor na to, aby žádné další periferie, nevyužívaly stejné piny s periferiemi již připojenými. Koncové zařízení s mnoha shieldy by teoreticky mohlo zabírat daleko více místa než to s moduly stejné funkcionality.



Obrázek 5: Arduino WiFi Shield [43], Arduino Uno [44]

Další možností je připojení již zmíněného modulu, s možností připojení k WiFi síti, k Arduino desce. Nejpopulárnější WiFi modul je bez pochyby modul s označením *ESP8266*. Tato varianta připojení k síti internet je levnější než využití WiFi shieldu, zabírá znatelně méně místa, ale je náročnější na instalaci.

Z uvedených důvodů byl pro komunikaci navrženého koncového zařízení s domácí sítí vybrán modul *ESP8266.*

5.3.3 Periferie

Přímo ke koncovému zařízení budou připojeny periferie, které budou umožňovat ovládaní některé části chytré domácnosti. Z periferií je nejprve zapotřebí fyzických tlačítek, které budou určovat, jestli bude například světlo stále vypnuté, stále zapnuté, či bude ovládané automaticky. Toto tlačítko by mělo mít nejvyšší prioritu, takže pokud například uživatel fyzicky nastaví pomocí přepínače, že mají být světla stále vypnutá, nebude pak možné, aby další uživatel světla zapínal pomocí webové, či mobilní aplikace. Dále mohou být ke koncovému zařízení připojeny již zmíněné senzory, které budou měřit veličiny v chytré domácnosti v reálném čase a odesílat tyto hodnoty na server, ten následně data zpracuje a na základě uživatelem předepsaných pravidel rozhodne, zda je třeba zásah systému pro regulaci měřené veličiny pomocí ovládacích prvků jako jsou například světla, termostatické hlavice a další.

Kdyby každá periferie měla svou vlastní Arduino desku s již připojeným WiFi Shieldem, tak by se celkový počet koncových zařízení rapidně zvýšil a s tím tak vzrostly náklady na celý systém. Na druhou stranu, pokud by měla vše v pokoji zařizovat jediná Arduino deska, vzrostly by nároky na její výkon a počet pinů. Při velkém množství periferií by mohlo být využití jediné desky dokonce nemožné. Aby bylo možné co nejvíce eliminovat počet koncových zařízení, tak by měly být navrženy tak, aby každé koncové zařízení bylo zaměřeno na jednu veličinu příslušného pokoje a měl k sobě připojené všechny periferie potřebné jak ke komunikaci se serverem, tak pro regulaci dané veličiny ve zvoleném pokoji. Například jeden pokoj bude obsahovat jedno koncové zařízení zaměřené na ovládaní osvětlení, který k sobě bude mít připojený například WiFi Shield pro komunikaci se serverem, jeden přepínač umožňující volit stav světel manuálně, senzor pro zaznamenávání úrovně světla v pokoji a relé pro samotné zapnutí světel v pokoji, v případě moderních osvětlovacích LED pásek, není potřeba ovládat průchod proudu z externího zdroje do žárovky, ale jelikož je jejich napájení přibližně 5V, je možné jej napájet přímo z pinů Arduino desky.

Periferie, které je možné připojit k desce a využít je tak k ovládání chytré domácnosti mohou být následující:

- Světlo
- Tlačítko
- Spínač
- Relé
- LED pásek
- RGB LED pásek
- Zámek
- Teploměr
- Vlhkoměr
- Senzor osvětlení
- Senzor pohybu
- Termostatická hlavice

Každé periferie mají vlastní parametry, které je třeba jim předat, aby fungovaly správně. Například u RGB LED pásku je třeba nastavit hodnoty pro svítivost červené, zelené a modré barvy, které jsou reprezentovány čísly 0-255 (8 bitů). Výsledná barva je pak tvořena kombinací výše zmíněných barev.

6 Návrh systémových částí

Před samotnou implementací, je třeba nejprve navrhnout logickou strukturu jednotlivých vrstev navržené architektury, aby bylo dosaženo požadované funkcionality. Návrh jednotlivých prvků celého systému tak bude rozebrán níže.

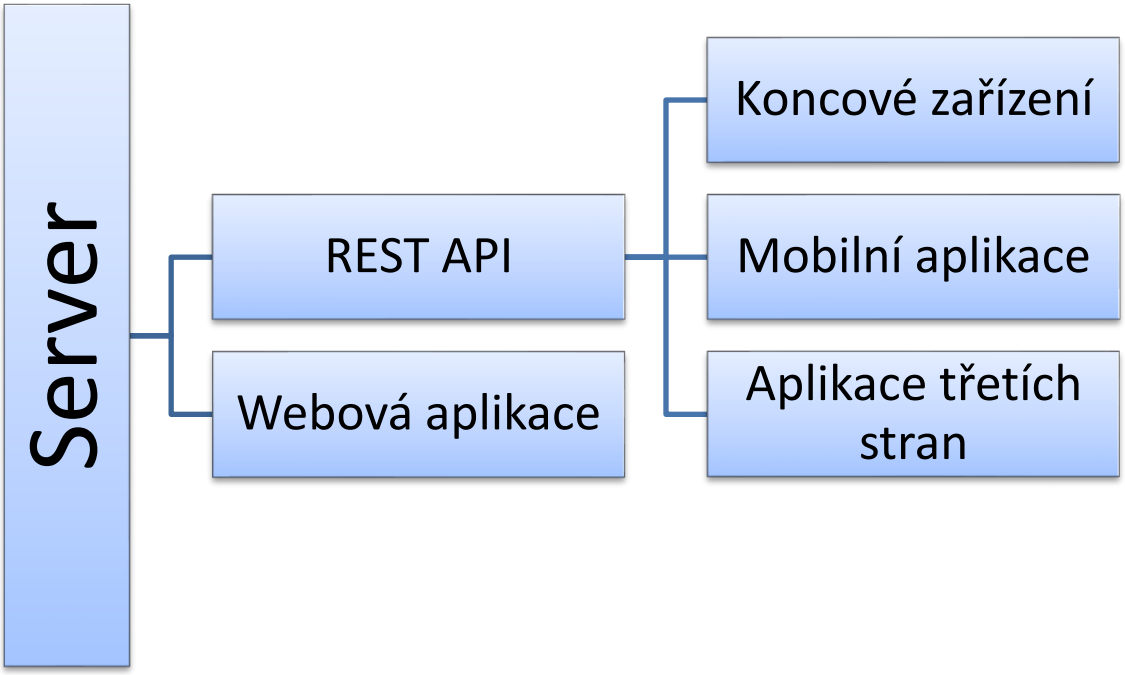
6.1 Server

Server je centrální prvek celého systému, který obstarává jak správu chování jednotlivých koncových zařízení, tak komunikaci s uživatelem.

Jak bylo zvoleno v předešlé kapitole, server bude poskytovat REST API, skrze které bude možné ovládat jednotlivá koncová zařízení, dále ho bude možné rozšířit o funkce, které budou poskytovat služby pro mobilní aplikace. Do budoucna tedy bude možné navrhnout vlastní mobilní aplikaci, která bude řídit koncová zařízení s pomocí nabízeného REST API. Návrh mobilní aplikace nicméně není předmětem této diplomové práce a jedná se pouze o možné rozšíření systému do budoucna.

Webová aplikace bude využívat návrhového vzoru MVC a její logika tak bude oddělena od zbytku serveru. Výhodou tohoto přístupu je fakt, že lze při vývoji využít MVC šablon, připravených jak pro autentizaci uživatelů a jejich správu v samotné aplikaci, tak pro CRUD operace nad daty, což značně usnadní vývoj celé aplikace. Nevýhodou pak je, rozdělení logiky serveru a možný vznik toho, že se jedna akce, která je potřeba vykonat jak na webové, tak na mobilní aplikaci, bude provádět na více místech. Pokud pak později bude třeba zavést nějaké změny, je třeba změny provést na obou místech, což může vést k zavádění dalších chyb, či odlišné chování mobilní a webové aplikace. Řešení této situace je nicméně relativně jednoduché. Stačí navrhnout novou servisní vrstvu, která bude nabízet požadovanou funkcionalitu a obě aplikace budou moci volat funkce v této servisní vrstvě s příslušnými parametry, následně servisní vrstva vrátí požadovaná data, odpovídající vstupním datům obdržených skrze parametry dané funkce. Jakmile je servisní vrstva takto připravená, je možné celý systém relativně jednoduše rozšířit a možné úpravy logiky systému bude možné dělat pouze na jednom místě.

Ukázka komunikace mezi serverem a zbytkem systému je znázorněna na Obrázek 6.



Obrázek 6: Komunikace systému se serverem

Komunikace s uživatelem bude zpočátku probíhat výhradně skrze webové rozhraní, a jak již bylo řečeno, bude zde možnost systém rozšířit s pomocí připravené REST API. Webová aplikace bude uživateli umožňovat následující operace.

Mezi hlavní operace bude možnost přidávání a správy svých chytrých domácností. Uživatel uvidí seznam svých chytrých domů a domů, ke kterým má oprávněný přístup. Zde bude možné přidávat další chytré domácnosti a editovat informace o jednotlivých domácnostech.

Při zobrazení detailu domácnosti, bude uživatel moci přidávat jednotlivá koncová zařízení k vybrané chytré domácnosti. Budou zde také vypsány základní informace k vybrané domácnosti. Níže se bude vyskytovat seznam koncových zařízení společně s jejich základními ovládacími prvky. O jaký typ ovládacích prvků se bude jednat, bude záležet na typu koncového zařízení. Koncová zařízení, kde bude požadovaný výstup pouze vypnuto a zapnuto, jako například jednoduchá světla (vyjímaje RGB, či stmívajících světel), budou mít jako ovládací prvek jednoduché tlačítko pro změnu jejich stavu. Naopak bude-li třeba nastavit například teplotu v bytě, či intenzitu světla, zobrazí se jako ovládací prvek např. posuvné tlačítko.

Například u RGB světel bude zapotřebí jak tlačítka pro vypnutí a zapnutí světel, tak jedno posuvné tlačítko pro intenzitu každé ze tří barev, ve výsledku tedy 4 ovládací prvky, pro jediné koncové zařízení. Jako poslední funkcionalita v tomto pohledu, bude možnost udělovat oprávnění uživatelům k ovládání zvolené chytré domácnosti, k tomu bude mít ale přístup pouze uživatel, který danou domácnost do systému přidal.

Administrátor domu (uživatel, který chytrou domácnost přidal do systému) bude moci kliknout na detail koncového zařízení, kde bude moci editovat základní informace, odebrat ho z příslušné domácnosti a také upravovat pravidla, kterými se zvolené koncové zařízení bude řídit. Všechny tyto funkce budou přístupné pouze administrátorovi chytré domácnosti.

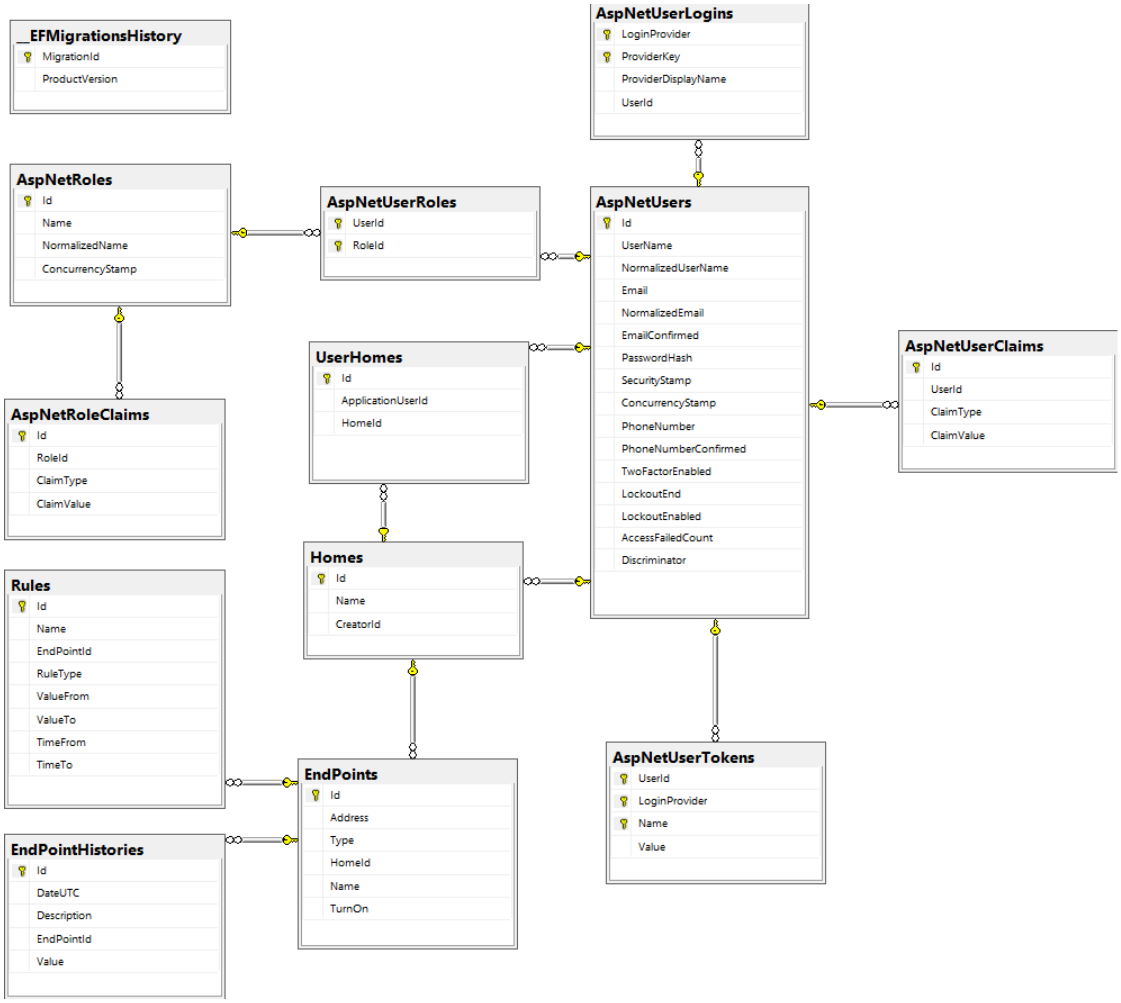
6.2 Databáze

V kapitole 5.2 byla vybrána jako technologie k vývoji databáze relační databáze, konkrétně MSSQL databáze, od společnosti Microsoft.

Dále díky Entity Frameworku, není zapotřebí psát skript pro vytvoření DB, ale pouze se ve vývojovém prostředí napíšou modelové třídy s patřičnými anotacemi a skripty pro vytváření databáze a její editaci při úpravě modelů se generují a provádí automaticky na příkaz vývojáře.

Veškerá data budou ukládána v databázi, a to včetně údajů o jednotlivých uživatelích, chytrých domácnostech, koncových bodech, jejich typech, historií změn stavů a naměřených hodnot. Budou zde uloženy i pravidla, kterými se budou jednotlivá koncová zařízení řídit.

Navržená struktura databáze vycházející z požadavků na systém, je znázorněna na Obrázek 7.



Obrázek 7: Navržená struktura databáze

6.3 Koncové zařízení

Jelikož hlavním bodem této práce je navrhnout server pro chytrou domácnost, který bude ovládat veškerá uživatelská koncová zařízení na základě uživatelem definovaných pravidel. Bude zde navržen pouze jedno koncové zařízení pro vyzkoušení funkčnosti komunikace se serverem a otestování funkčnosti algoritmu pro zhodnocování definovaných pravidel. K otestování serveru bude navrženo koncové zařízení pro ovládání osvětlení.

Pro snížení nákladů a minimalizování celkové velikosti koncového zařízení, byl pro komunikaci se serverem vybrán Wi-Fi modul ESP8266 a společně s deskou Arduino, by měl tvořit základ každého koncového zařízení, pokud koncový uživatel nepreferuje jinou formu komunikační technologie. Po nastavení parametrů a

připojení k místnímu Wi-Fi routeru, bude pomocí tohoto modulu a Arduino desky možná komunikace se serverem, který bude zajišťovat automatické řízení každého koncového zařízení. K tomuto jádru se následně budou moci připojovat další periferie, a tím bude možné rozšířit jejich funkcionalitu.

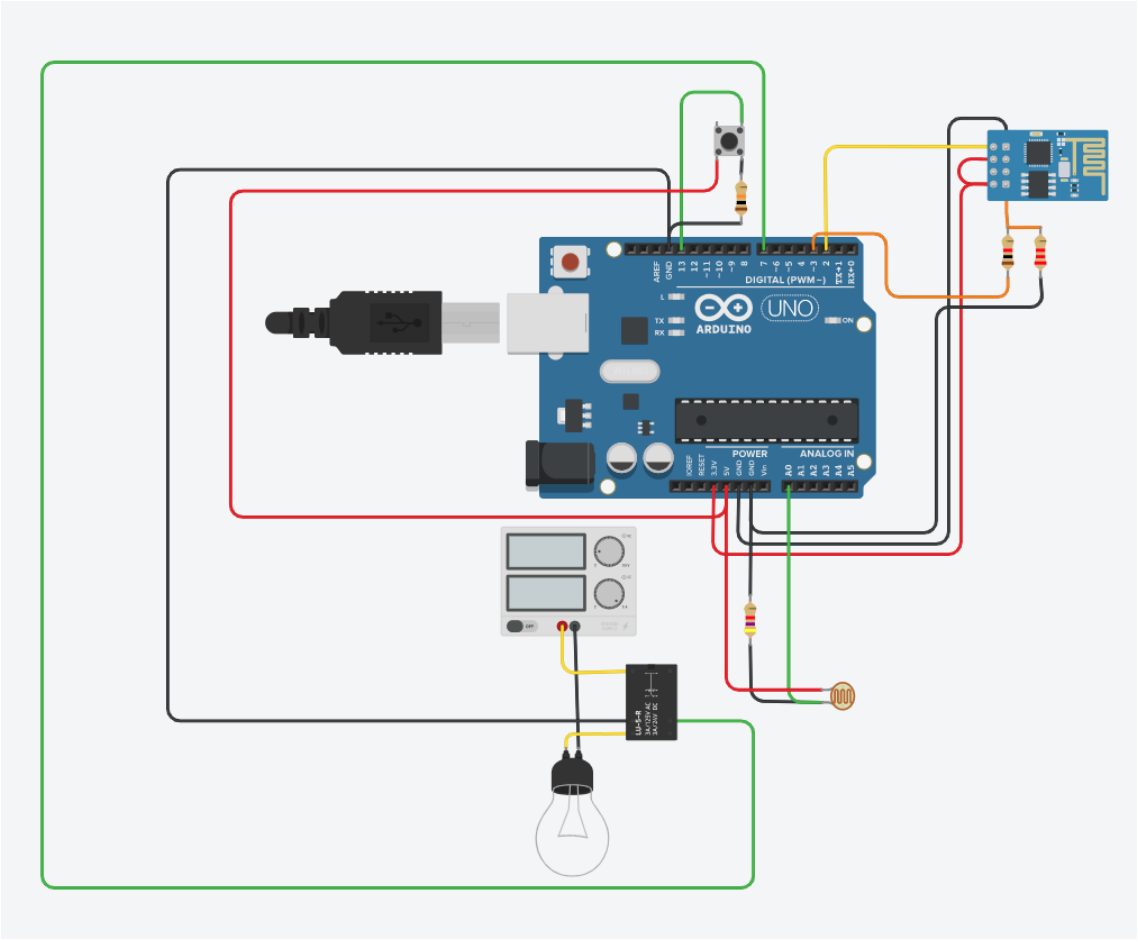
Jako první periferii, kterou je třeba připojit je tlačítko, které bude určovat, jestli bude žárovka stále vypnutá, stále zapnutá, nebo bude její zapínání řídit server. Bude se jednat o jednoduché tlačítko, které po zmáčknutí bude přepínat mezi třemi výše zmíněnými stavy.

K měření osvětlení je možné využít fotorezistor. Jedná se o elektronickou součástku, která má proměnný odpor v závislosti na míře osvětlení. S rostoucí mírou osvětlení, klesá celkový odpor fotorezistoru. Takto naměřená hodnota je pak společně s unikátním identifikátorem daného koncového zařízení odeslána pomocí dotazu na API serveru. Ten na základě pravidel definovaných pro dané koncové zařízení data zhodnotí a odpoví koncovému zařízení, zda má osvětlení zapnout, či nikoliv.

Na základě této odpovědi bude světlo rozsvíceno, nebo zhasnuto. Jelikož běžné žárovky potřebují externí zdroj energie, není možné jej napájet přímo z desky a je za potřebí regulovat průtok proudu v obvodu žárovky. Toho lze dosáhnout pomocí relé. Které je na základě hodnot na jeho vstupu buď propustné, či nepropustné. Samotné relé má dva módy (běžně otevřené a běžně zavřené), které jsou si navzájem opačné. V běžné otevřeném módu je relé propustné, jeli na řídícím pinu (pin IN vpravo na Obrázek 8) logická nula (0 V). Naopak běžně zavřený mód je propustný, jeli na řídícím pinu logická jednička (5 V). Pro zapojení do běžně otevřeného módu je třeba obvod žárovky přerušit a připojit jej na piny COM a NO. Podrobný návod lze nalézt v článku [46].

Obrázek 8: Relé modul pro arduino [45]

Samotný návrh zapojení koncového zařízení je zobrazen na Obrázek 9.



Obrázek 9: Návrh zapojení koncového zařízení

7 Implementace

Jakmile máme vybrané technologie a určené požadavky na systém, je možné přejít k samotné implementaci.

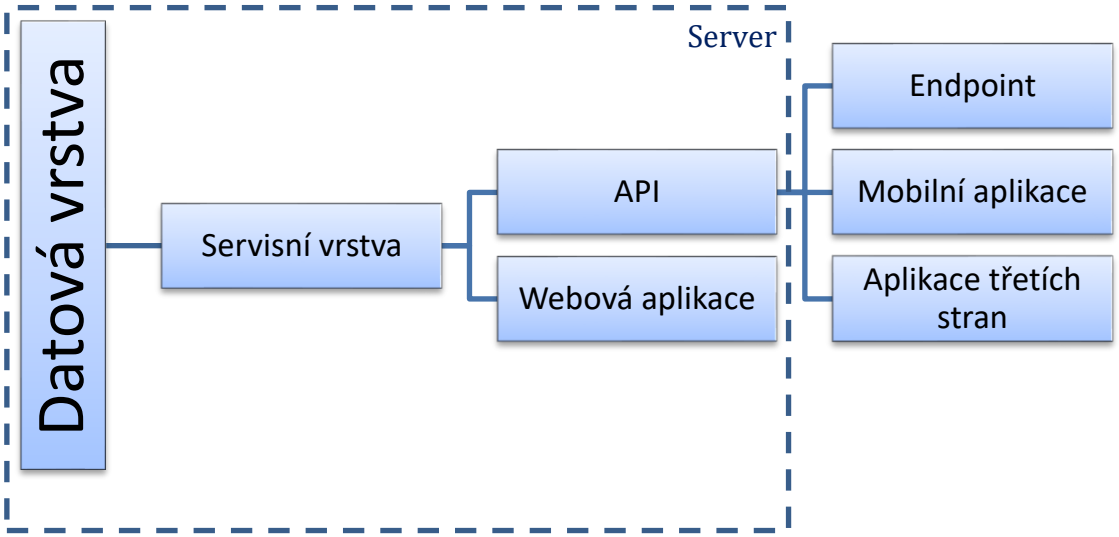
7.1 Server

Celý server byl napsán pomocí technologie .NET Core od společnosti Microsoft. Hlavním důvodem výběru této technologie je fakt, že je multiplatformní a není tak jeho nasazení limitováno pouze na servery s operačním systémem Windows.

Jeho funkcionalita je rozdělena do 3 hlavní částí, které jsou:

- **Datová vrstva** – umožňující přístup k datům uložených v databázi, která může být provozována na stejném zařízení jako zbytek serveru, nebo na jiném zařízení a připojení bude definováno v nastavení datové vrstvy
- **Webová aplikace** – zprostředkovává komunikaci mezi uživatelem a datovou vrstvou skrze webové rozhraní
- **API** – umožňuje komunikaci mezi datovou vrstvou a koncovými zařízeními, či dalšími aplikacemi.

Jak již i bylo zmíněno, jelikož s databází pracuje více nadřazených vrstev, jako je webová aplikace a API pro komunikaci s koncovými zařízeními a mobilními aplikacemi. Aby práce s daty byla konzistentní napříč celým řešením, nedocházelo k opakování stejného kódu a případného zanášení chyb při jeho možné úpravě v budoucnu, byla přidána servisní vrstva (Obrázek 10), která nabízí veškeré potřebné metody pro práci s daty, včetně autorizace a autentizace uživatelů.



Obrázek 10: Hierarchie vrstev řešení chytré domácnosti

Veškeré metody pro práci s daty jsou tak přístupné na jednom místě a vyšší vrstvy mohou pouze implementovat třídy servisní vrstvy a provádět tak operace s daty skrze tyto třídy.

Další velkou výhodou přidání této vrstvy je možnost jednoduchého rozšíření aplikace, nebo možné úpravy existujícího kódu v budoucnu. Jelikož je kód pro práci s daty na jednom místě a každá vrstva nepřistupuje k datům zvlášť, neopakuje se zbytečně kód na více místech daného řešení a pokud je to nezbytné, je velice jednoduché nalézt místo, kde je třeba kód upravit a není tak třeba ho hledat ve všech vrstvách, které pracují s příslušnými daty.

7.1.1 Datová vrstva

V kapitole 5, bylo řečeno že k vytvoření datového modelu bude využit Entity Framework. Ten zpřístupňuje databázi pro aplikaci, za pomocí tzv. datového kontextu a umožňuje jednoduchou práci s databázovým modelem přímo ve vývojovém prostředí MS Visual Studio.

Nejprve je třeba návrh datových modelů pomocí modelových tříd, kde každá třída odpovídá jedné tabulce v databázi. Entity Framework nabízí možnost vytvoření datového modelu z existujících tříd, nebo vytvoření datového modelu z již existující databáze. Zde byl vybrán přístup generování databáze z navrhnutých modelových tříd. Každá třída musí v atributech obsahovat unikátní identifikační

klíč, pomocí kterého bude v databázi adresován. Tzv. závislá entita obsahuje vždy cizí klíč k entitě hlavní [47]. Aby entity framework řádně rozeznal, který z atributů dané třídy je unikátní klíč, cizí klíč, nebo zavedl různá omezení na jednotlivé atributy, jako například omezení počtu znaků atributu, aby atribut měl tvar emailové adresy a další, je třeba u těchto atributů daná kritéria specifikovat. Lze toho dosáhnout pomocí anotací, nebo pomocí tzv. Fluent API.

- **Anotace** lze definovat přímo modelu a v jazyce C# se uvádějí do hranatých závorek nad každý atribut. Výhodou tohoto přístupu je fakt, že jsou veškeré definice a omezení přímo u parametrů a není třeba je nikde hledat. Nevýhodou je naopak fakt, že u rozsáhlých modelů dochází k tomu, že se kód pro velké množství informací na jednom místě stává nepřehledným.
- **Fluent API** odděluje strukturu modelů od jejich chování a tím dokáže značně zpřehlednit rozsáhlý kód. Hodí se tak pro rozsáhlejší modely a v některých případech není ani možné požadovaná omezení zapsat v Entity Framework Core pomocí anotací, tak jsme nuceni využít Fluent API.

Aby bylo možné jednoduše odkázat na návazné entity, umožňuje entity framework odkazovat na tyto entity pomocí virtuálních atributů. V následné ukázce kódu lze vidět použití anotací, cizích klíčů i virtuálních atributů. Anotace *Key* označuje unikátní klíč. Pomocí anotace *ForeignKey* nad atributem *CreatorId* je Entity framework schopný načíst odpovídající entitu, která bude přístupná skrze virtuální atribut Creator.

```
public class Home
    {
        [Key]
        public int Id { get; set; }

        public string Name { get; set; }

        [ForeignKey("Creator")]
        public string CreatorId { get; set; }
        public virtual IdentityUser Creator { get; set; }
        public virtual ICollection<ApplicationUserHome> ApplicationUsersHomes
{ get; set; }
        public virtual ICollection<EndPoint> EndPoints { get; set; }
    }
```

Syntaxe jazyka C# nám umožňuje zapisovat gettery a settery pro přístup k hodnotám, které jsou v atributech uloženy a vkládání nových hodnot do atributů, pomocí zapsání get, nebo set, za atribut do složených závorek.

Dále je na kódu výše vidět, že je třída Home ve vztahu s jednou entitou třídy IdenityUser, jelikož obsahuje cizí klíč *CreatorId* a virtuální atribut *Creator*, skrze který jsou přístupné informace o uživateli, který vytvořil danou domácnost. Jednotlivé entity mezi sebou mohou být ve vztazích:

- **One-to-one** – Je příklad, kdy jedna entita jedné třídy je ve vztahu pouze s jednou entitou třídy druhé. Tento vztah by se definoval tak, že závislá entita nebude mít svůj vlastní primární klíč, ale bude mít pouze cizí klíč, odkazující na primární klíč entity hlavní.
- **One-to-many** – Zde má jedna entita jedné třídy vztah s několika entitami třídy druhé. Hlavní entita v tomto vztahu bude entita, která je ojedinělá a závislé budou všechny entity ve vztahu k ní. V případě ukázkového kódu výše, je vztah one-to-many vztah mezi entitou Home a EndPoint (česky: „koncové zařízení“), kde dům může obsahovat několik koncových zařízení, ale každé koncové zařízení přísluší pouze do jednoho domu. Cizí klíč tedy budou mít v sobě entity třídy EndPoint.
- **Many-to-many** – Je vztah, kde může mít několik entit jedné třídy vztah s několika entitami třídy druhé. Zde je třeba vytvořit novou

třídu, obsahující cizí klíče do obou entit tohoto vztahu a obě entity budou mít v atributech kolekci této nově vytvořené třídy. V předešlém příkladu se jedná o vztah mezi uživateli a domy, kde jeden uživatel může ovládat více domů a jeden dům smí být ovládán více uživateli.

Jakmile budou vytvořeny všechny modelové třídy, je třeba je přidat do databázového kontextu. Jedná se o třídu, která je vytvořená po přidání Entity Frameworku do projektu, ze které samotný framework pozná, podle kterých modelových tříd má vytvořit, či upravit databázi. Dosáhne se toho přidáním atributu *DbSet* s patřičnými parametry pro každou modelovou třídu. Dále je zde možné přepsat metodu *OnModelCreating*, kde je možné pomocí již zmíněné Fluent API [48] upravit pravidla atributů jednotlivých tříd. Níže, je v této metodě vidět využití Fluent API k určení unikátnosti adresy koncového zařízení. Dále lze vidět přidání veškerých modelových tříd do databázového kontextu.

```
public class ApplicationDbContext : IdentityDbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
            : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder builder)
        {
            builder.Entity<EndPoint>().HasIndex(i => i.Address).IsUnique();
            base.OnModelCreating(builder);
        }

        public DbSet<ApplicationUser> ApplicationUsers { get; set; }
        public DbSet<ApplicationUserHome> UserHomes { get; set; }
        public DbSet<EndPoint> EndPoints { get; set; }
        public DbSet<EndPointHistory> EndPointHistories { get; set; }
        public DbSet<Home> Homes { get; set; }
        public DbSet<Rule> Rules { get; set; }
    }
```

Jakmile je vše připraveno, je možné přidat veškeré změny do migrace a aktualizovat databázi, aby odpovídala předepsanému modelu. Je třeba v nástroji zvaném *Package Manager Console* zadat následující příkazy:

- `add-migration` – po tomto příkaze je třeba zadat název nové migrace a entity framework automaticky vygeneruje skript pro úpravu databáze z migrace předchozí (v případě první migrace, založí skript novou databázi)
- `update-database` – tento příkaz aplikuje vygenerovaný skript na připojenou databázi (pokud je třeba vrátit databázi do určitého stavu, je možné použít příkaz `update-database -migration „název požadované migrace“`)

7.1.2 Servisní vrstva

Servisní vrstva byla přidána, aby poskytovala metody vyšším vrstvám architektury pro jednotnou práci s daty. Zároveň se tímto přístupem eliminuje možnost zanášení chyb, při možném upravování kódu se stejnou funkcí na více místech v projektu.

Veškeré servisní třídy byly programovány oproti rozhraní a k jejich konkrétní implementaci lze využít *Dependency Injection* [49], což umožní vložit do konstruktoru třídy, kde je servisní rozhraní využíváno, konkrétní instanci implementující dané rozhraní. O samotnou implementaci rozhraní a jeho životní cyklus se pak stará kontejner pro správu služeb. V konfigurační třídě *Startup.cs* je metoda *ConfigureServices*, které umožňuje konfigurovat tento kontejner. Nejprve je potřeba do služeb určit životní cyklus dané služby a třídu, kterou bude dané rozhraní implementovat. Životní cyklus může být jeden z následujících:

- **Transient** – Jsou služby, které jsou vytvořeny při každém požadavku a ukončeny, jakmile je požadavek splněn.
- **Scoped** – Jsou vytvořeny jednou, pro každého klienta a smazány, jakmile je spojení s klientem ukončeno.
- **Singleton** – Jsou služby, které jsou vytvořeny při prvotním požadavku a všechny následující požadavky využívají tu samou instanci a jsou smazány až při vypnutí serveru.

```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<ApplicationDbContext>(options =>
                options.
                //UseLazyLoadingProxies().
                UseSqlServer(
                    Configuration.GetConnectionString("DefaultConnection")));
            services.AddDefaultIdentity<IdentityUser>(options =>
options.SignIn.RequireConfirmedAccount = true)
                .AddEntityFrameworkStores<ApplicationDbContext>();
            services.AddControllersWithViews();
            services.AddRazorPages();
            services.AddControllersWithViews().AddNewtonsoftJson(options =>
options.SerializerSettings.ReferenceLoopHandling =
Newtonsoft.Json.ReferenceLoopHandling.Ignore);

            services.AddScoped<IHomeService, HomeService>();
            services.AddScoped<IRuleService, RuleService>();
            services.AddScoped<IEndPointService, EndPointService>();
        }
```

Výše je ukázka konfigurace služeb. V dolní části je určeno, pokud bude použito například rozhraní *IHomeService* bude implementována nová instance třídy *HomeService*. Je zde také definováno, jaký typ databáze je použit, jaký connection string použít k jejímu připojení, či případné alternativní metody načítání dat z příbuzných entit do virtuálních atributů modelových tříd. Možnosti načítání těchto dat jsou následující [50]:

- **Eager loading** – Data příbuzných entit jsou do virtuálního atributu načítána specificky na vyžádání při práci s daty prostřednictvím jazyka LINQ. Výhodou této metody je, že nenačítá zbytečná data, a jsou z databáze získána pouze ty data, která jsou vyžádána. Nevýhodou pak, že pokud nejsou data specificky vyžádána, nebudou dostupná pro vyšší vrstvy.
- **Explicit loading** – Data jsou načítána explicitně pomocí API *DbContext.Entry*. Takže lze definovat, že při načítání jedné entity budou automaticky do virtuálního atributu načteny data příbuzných entit.
- **Lazy loading** – Zde jsou automaticky vyžadována veškerá data z příbuzných entit a zapisována do virtuálních atributů. Velkou výhodou je fakt, že se o příbuzná data nemusíme starat a jsou

požadována automaticky, nevýhodou pak, že dochází k přenosu zbytečně rozsáhlých dat, což může ovlivnit výkon celé aplikace. Nebo v krajním případě může dojít dokonce i k zacyklení. Kde příbuzná data k dané entitě mohou při hlubším průchodu odkazovat sama na sebe. Například v databázi tohoto projektu (Obrázek 7), pokud je entita třídy *House*, která má svého vlastníka, ten má jako uživatel několik k sobě přiřazených domů, ty mají opět vlastníka atd. Podmínkou použití tohoto načítání dat by tedy měl být fakt, že neexistuje v databázi takováto vazba, která by tento proces zacyklila. Pokud chceme využít lazy loadingu, je třeba jej definovat v konfigurační metodě *ConfigureServices* (ukázkový kód výše).

Byl tedy zvolen eager loading, který lze využít pomocí metody *Include* v jazyce LINQ, při práci s databázovým kontextem. Pokud bychom chtěli načíst i další příbuzná data, lze využít metody *ThenInclude*. Metoda pro získání informací domu podle jeho identifikačního čísla, společně s informacemi o koncových zařízeních, která jsou k domu přiřazená, vypadá následovně.

```
public Home GetHomeById(int id, string userId)
        {
            if (CanUserAccessHome(id, userId))
            {
                var home = _context.Homes.Include(i => i.Creator)
                           .Include(i => i.EndPoints)
                           .FirstOrDefault(m => m.Id == id);
                return home;
            }
            return null;
        }
```

Všechny metody servisní vrstvy pracující s daty, která jsou přístupná pouze oprávněným uživatelům. Přijímají jako poslední parametr unikátní textový řetězec, který je generován každému uživateli a vracen při jeho přihlášení. A následně jej předává privátní metodě (v tomto případě *CanUserAccessHome)*, která na základě identifikátoru uživatele a primárního klíče entity ke které se snaží získat přístup vyhodnotí, zda má uživatel oprávnění zacházet s požadovanými daty, či nikoliv.

Pokud uživatel oprávnění má, je pomocí jazyka LINQ sestaven dotaz k získání požadovaných dat.

- *_context* – Je databázový kontext, odkazující na určitou databázi.
- *Homes* – Označuje, že se data budou získávat z tabulky Homes.
- *Include* – Je metoda, která oznamuje entity frameworku, že budou požadována i data majitele a koncových zařízení, které byly k domu přiřazeny.
- *FirstOrDefault* – najde první entitu, která má ID shodné s ID, které bylo předánu v parametru metody *GetHomeById*. Pokud nenalezne žádnou takovou entitu, vrací null.

Jednou z hlavních výhod jazyka LINQ je to, že je možné v něm stále přidávat podmínky pro vyhledávání a výsledný dotaz bude zavolán v programu až při iterování skrze výsledný seznam. Pokud tedy existují nějaké podmínky jako třeba filtrování výsledků, tak má-li parametr s filtrem nějakou hodnotu, lze jednoduše k existujícímu dotazu přidat další podmínku, a celý dotaz bude zavolán například až po zavolání metody *ToList* nad celkovým dotazem.

7.1.3 API

Tato vrstva serveru poskytuje metody pro práci s daty pro mobilní aplikace, možné aplikace třetích stran a samotná koncová zařízení (Obrázek 10). K implementaci byla v kapitole 5 vybrána technologie REST. Jelikož se jedná o bez stavový protokol, je třeba získat veškeré potřebné informace v rámci jednoho požadavku na server.

Aby aplikace rozeznala, že se jedná o kontrolér pro API je třeba vše určit pomocí anotací přímo v kontroléru. V ukázkovém kódu níže, je znázorněn jeden z API kontroléru celého serveru. Jedná se o kontrolér pro komunikaci s koncovými zařízeními. V horní části je pomocí anotace *Route* určena cesta, na které bude kontrolér dostupný, níže je anotace, která oznamuje že se jedná o API kontrolér. Konstruktor v parametru přijímá rozhraní ze servisní třídy pro entitu, se kterou kontrolér bude pracovat a s pomocí dependency injection, který je definovaný v metodě pro konfiguraci služeb a nachází se v třídě *Startup.cs*, je automaticky

implementována třída, která pro toto rozhraní byla definována. V tomto případě bude do privátní proměnné *_endPointService* implementována třída *EndPointService*. Pokud kontrolér potřebuje přístup k dalším servisním třídám, stačí přidat novou privátní proměnnou (přístupná pouze uvnitř dané třídy) a v konstruktoru třídy ji stejným způsobem naplnit. Výsledná adresa metody se bude skládat z adresy serveru, cesty kontroléru uvedené v anotaci *Route* a cesty konkrétní metody v anotaci *HttpGet*, která zároveň uvádí, jestli bude metoda volána pomocí http požadavku GET. Pokud bychom chtěli volat například pomocí požadavku POST byla by zde anotace *HttpPost* atd. V případě metody *RegisterDevice* bude výsledná cesta *https://localhost:44341/api/EndPointCommunication/RegisterDevice/{address}/{type}*

- *localhost:44341* - bude nahrazen IP adresou serveru, na který bude aplikace nahrána
- *{address}* – nahradí MAC adresou daného koncového zařízení
- *{type}* - bude označovat o jaký typ koncového zařízení se jedná (1 – světla, 2 – topení, 3 – zámek)

Přímo v samotných metodách kontroléru nedochází k žádné práci s daty. Kontrolér pouze vezme hodnoty, které obdrží v http požadavku a předá je příslušné metodě servisní vrstvy. Ta, pokud je třeba, zjistí, zda je uživatel autorizován k práci s příslušnými daty, následně data předaná parametrem zpracuje a vrátí požadované hodnoty, nebo potvrzení o úspěšném smazání, vložení, či úpravě entity. API kontrolér pak pouze předá získaná data uživateli. Takto jsou uživatelská data předávána servisní vrstvě i z ostatních vyšších vrstev aplikace, což zajistí konzistentnost práce s daty napříč celou aplikací.

```
[Route("api/[controller]")]
    [ApiController]
    public class EndPointCommunicationController : ControllerBase
    {
        private readonly IEndPointService _endPointService;

        public EndPointCommunicationController(IEndPointService
endPointService)
        {
            _endPointService = endPointService;
        }

        [HttpGet("NewValue/{address}/{value}")]
        public bool? NewValue(string address, int value)
        {
            return _endPointService.AddNewValueToEndpoint(address, value);
        }

        [HttpGet("RegisterDevice/{address}/{type}")]
        public bool? RegisterDevice(string address, int type)
        {
            return _endPointService.RegisterNewDevice(address, type);
        }
    }
```

Koncové zařízení bude se serverem komunikovat pomocí metody *NewValue*, která jako parametr přijímá adresu zařízení a naměřenou hodnotu. Obě tyto hodnoty následně předá servisní vrstvě. Ta si vytáhne pomocí unikátní adresy příslušné koncové zařízení z databáze, nově získanou hodnotu přidá do historie (tabulka *EndPointHistories* v databázi), společně s časem a datem, kdy byla hodnota získána. Tyto hodnoty by mohli později být zhodnocovány a mohlo by z nich být zjištěno například, kde se dá ušetřit, návyky uživatelů s možnou predikcí chování a další. Dále získá veškerá pravidla, která jsou uložena v databázi pro dané koncové zařízení a postupně u všech zjistí, jestli nejsou splněna. Pravidla mohou být dvojího typu:

- **Založená na čase** – Jedná se o pravidla, která určují, že se dané koncové zařízení má spustit, pokud je momentální čas mezi hodnotami X a Y, uloženými v databázi.
- **Založená na hodnotě** – Tyto pravidla rozhodují, zda se má koncové zařízení zapnout na základě hodnoty, která byla naměřena senzorem

a odeslána na API serveru. Pokud je naměřená hodnota v určitém rozmezí, je koncové zařízení zapnuto.

Všechna pravidla jsou vyhodnocována postupně a jakmile je alespoň jedno pravidlo splněno, je vyhodnocování kvůli snížení výpočetní náročnosti ukončeno a servisní vrstva vrací informaci o tom, že má být akční prvek, připojený ke koncovému zařízení, zapnut (světla, regulátor topení atd.).

Dále API nabízí metody pro práci s veškerými daty a autentizaci uživatelů. Veškeré metody jsou rozděleny podle entity, se kterou pracují a každá entita v databázi má svůj vlastní API kontrolér s CRUD operacemi. Tyto metody přijímají jako poslední parametr unikátní klíč uživatele, aby mohlo být v servisní vrstvě určeno, jestli má uživatel k požadovaným datům přístup.

7.1.4 Webová aplikace

Část systému, se kterou bude uživatel nejvíce v interagovat je právě webová aplikace, kde po nutné registraci bude uživatel schopný vytvářet nové chytré domácnosti, do kterých bude možné přiřazovat nová koncová zařízení a spravovat uživatele, kteří budou mít přístup k domácnosti. Dále zde bude možné definovat pravidla jednotlivých koncových zařízení, pomocí kterých se budou koncová zařízení řídit.

Nejprve je třeba realizovat správu uživatelů a možnost autentizace jednotlivých uživatelů a jejich autorizaci k přístupu ke konkrétním datům. S tímto prvním požadavkem pomůže Identity Framework, který je součástí vývojového prostředí Visual Studio, a je možné přidat funkcionalitu pro správu uživatelů, rolí, potvrzení registrace přes email a další. Jelikož je Identity Framework součástí vývojového prostředí, vše je možné přidat již při vytváření nového projektu, kde stačí vyžádat o založení nového .NET Core MVC projektu s přihlašovacími schopnostmi. Vývojové prostředí následně vygeneruje:

- Základní kostru aplikace, obsahující možnost registrace uživatele, jeho následného přihlášení a autorizace v rámci celé aplikace. K samotné registraci je také možné, po řádném nastavení, využít externích přihlašovacích providerů jako je Facebook, Google, Twitter

a Microsoft Account. Registrace také může po nastavení SMTP protokolu v aplikaci odesílat emaily pro potvrzení identity uživatele na registrovaný email.

- Databázové modely pro uložení dat o uživatelích a jejich rolích, které ale budou nepřístupné. Pokud budeme chtít tento model rozšířit, je možné vytvořit novou modelovou třídu, která bude dědit od třídy kterou chceme rozšířit a přidat zde následně veškeré požadované doplňující informace. Novou modelovou třídu je pak třeba přidat do databázového kontextu, vytvořit nové migrace a aplikovat změny v databázi.

Pokud chceme nějaké metody, či celý kontrolér zpřístupnit například jen registrovaným uživatelům, stačí do anotací nad příslušné metody či nad samotný kontrolér použít anotaci *Authorize*. Obdobně by bylo možné omezit přístup pouze pro uživatele s rolí administrátor atd.

Webová aplikace je napsána pomocí návrhového vzoru MVC [51] (Model View Controller).

- **Model** – Určuje, v jakém stavu budou data prezentována uživateli. Jedná se o třídy, předepisující tvar získaných dat.
- **View** – Neboli pohled, prezentuje data uživateli skrze předepsané uživatelské rozhraní. V .Net Core je pohled na data psán pomocí *Razor View Engine*, který může využívat i kódu psaném v C# pro generování dynamického obsahu prezentující data, která byla do pohledu předána modelem.
- **Controller** – Kontroléry mají na starost celkové chování aplikace, zpracovávají uživatelské vstupy a starají se o interakci s uživatelem, vybírá model a pohled, který bude uživateli prezentován.

V praxi uživatel odešle požadavek na metodu kontroléru serveru s patřičnými parametry, ten požadavek zpracuje, zhodnotí a pokud má uživatel k dané metodě přístup, naplní získaná data do modelu, který je následně předán pohledu a ten je prezentován uživateli.

Pohledy mohou být v .NET Core psané pomocí syntaxe zvané Razor [52], která umožňuje mezi HTML kód zapsat psaný v C# pomocí symbolu „@“, čímž označíme, že se jedná o kód v jazyce C#. Razor zhodnotí zapsaný kód v C# a vygeneruje HTML. Pohledy mohou být silně typované, což znamená, že mají předepsaný model dat, který přijímají, a jsou schopny generovat obsah pouze pokud data která obdrží jsou ve tvaru, který odpovídá předem určenému modelu. Výhodou tohoto přístupu s pomocí syntaxe Razor je jednoduchý a přehledný kód pro generování dynamického obsahu. Dále tento přístup při odesílání formuláře na server, automaticky převádí data z formuláře na novou instanci třídy, která byla určená jako model pro daný pohled. Pokud daná modelová třída obsahuje nějaké atributy, které je třeba validovat (je třeba definovat pomocí anotací pro každý atribut), jsou data validována a ve volané metodě kontroléru je možné na případně neplatnou validaci reagovat pomocí metody *ModelState.IsValid*, která vrací hodnotu *true*, pokud je vše validní a *false* v opačném případě.

Níže je vypsána modelová třída, určující tvar dat, která budou předávána z kontroléru do pohledu. Modelová třída *HomeViewTableModel* má v sobě kolekci jednotlivých položek *HomeViewModel*, které nesou základní informace o každé domácnosti.

```
public class HomeViewTableModel{
        public List<HomeViewModel> Items { get; set; }
    }
    public class HomeViewModel {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Creator { get; set; }
        public bool IsUserCreator { get; set; }

        public List<EndPointsViewModel> EndPoints { get; set; }
    }
```

V kontroléru (kód níže) je metoda pro správu domácností, konkrétně metoda pro zobrazení domácností, ke kterým má přihlášený uživatel povolený přístup. Obdobně jako v API i zde metody kontroléru pouze předávají potřebné údaje servisní vrstvě, která je zpracuje a požadovaná data odešle ve správném formátu zpět. Třídy servisní vrstvy jsou zde implementovány, stejně jako v případě API kontroléru, pomocí dependency injection. Tato metoda následně vyhledá pohled,

který má stejný název, jako metoda kontroléru a předá mu data získaná ze servisní vrstvy ve formátu předepsaného modelu. Takto připravený pohled je následně prezentován uživateli.

```
[Authorize]
    public class HomesController : Controller
    {
        private readonly ApplicationDbContext _context;
        private readonly IHomeService _homeService;
        private readonly IEndPointService _endPointService;

        public HomesController(ApplicationDbContext context, IHomeService
homeService, IEndPointService endPointService)
        {
            _context = context;
            _endPointService = endPointService;
            _homeService = homeService;
        }

.
.
.
        public async Task<IActionResult> Index()
        {
            return View(_homeService.GetUserHomes(GetLoggedInUserId()));
        }
    }
```

Pohled pro metodu *Index* (ukázkový kód níže) je silně typovaný, a modelová třída je označená na prvním řádku za příkazem *@model*. Nyní je možné se na data obdržená z konstruktoru odkazovat níže v pohledu pouze pomocí příkazu *Model*. Hlavička generované tabulky s domácnostmi uživatele je generována standartním HTML kódem a dynamický obsah je vytvářen pomocí syntaxe Razor, kde v cyklu *foreach* pro každou položku v modelu generuje, pomocí párových HTML tagů *<tr>* pro řádek a *<td>* pro sloupec, jeden řádek do tabulky, obsahující informace o dané položce. Je také jednoduché vytvořit podmínky pro generování určitých elementů. Například v tomto pohledu se, s pomocí jednoduché podmínky *if,* tlačítka pro editaci a smazání domácnosti vygenerují pouze, pokud je přihlášený uživatel zároveň uživatelem, který danou domácnost založil. Pro vzhled bylo využito knihovny Bootstrap, která styluje jednotlivé HTML elementy podle hodnot uvedených

v atributu *class* a výčet jednotlivých zkratek, které lze využít můžeme nalézt v dokumentaci [53].

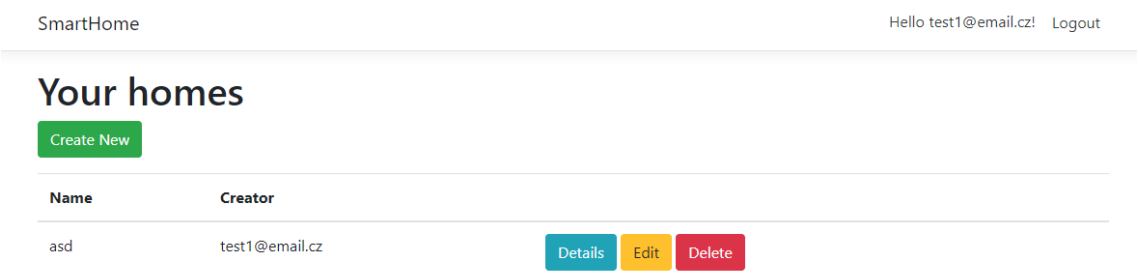
```
@model HomeViewTableModel

@{
    ViewData["Title"] = "Your homes";
}

<h1>@ViewBag.Title</h1>

<p>
    <a asp-action="Create" class="btn btn-success">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                Name
            </th>
            <th>Creator</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model.Items){
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                @item.Creator
            </td>
            <td>
                <a asp-action="Details" asp-route-id="@item.Id" class="btn
btn-info">Details</a>
                @if (item.IsUserCreator){
                <a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-
warning">Edit</a>
                <a asp-action="Delete" asp-route-id="@item.Id" class="btn btn-
danger">Delete</a>
                }
            </td>
        </tr>
}
    </tbody>
</table>
```

Tento obsah je následně vložen do rozložení (anglicky grid), které obsahuje hlavičku aplikace s menu, včetně údajů o přihlášeném uživateli a uživatelské menu. To se zobrazí po kliknutí na email přihlášeného uživatele a umožňuje správu osobních údajů. Konečný pohled pro seznam domácností, ke kterým má uživatel přístup, by po obdržení veškerých potřebných dat by mohl vypadat následovně (Obrázek 11).



Obrázek 11: Ukázka vzhledu webové aplikace

Jeli uživatel přihlášený, smí vytvářet nové domácnosti, spravovat domácnosti které vytvořil, a to včetně správy uživatelů, kteří mají k jeho domácnosti přístup a přidávání a odebírání koncových zařízení, které byli již registrovány skrze API. Veškeré tyto operace jsou z důvodu bezpečnosti vyhrazeny pro uživatele, který domácnost založil (tzv. správce domácnosti). Uživatelé, kterým byl správcem domácnosti přidělen přístup k domácnosti mohou společně se správcem pouze ovládat jednotlivá koncová zařízení v domácnosti a pravidla, kterými se řídí.

Veškeré tyto operace jsou chráněny jak na straně klienta (nejsou mu vykresleny prvky, umožňující tyto operace), tak na straně serveru (u každé operace je kontrolováno, zda má uživatel, který zadal požadavek k vykonání operace oprávněn k práci s příslušnými daty).

7.2 Koncová zařízení

Je koncový prvek celého systému, jehož základem je Arduino deska společně se zařízením umožňujícím komunikaci se sítí internet (lze vybírat z možností vypsaných v kapitole 5.3.2). V našem případě byl vybrán, pro jeho kompaktnost, modul *ESP8266.* Po navázání komunikace se serverem, bude koncové zařízení na

základě odpovědí serveru na jeho požadavky ovládat periferie k němu připojené. Konečná implementace závisí na uživateli.

Návrh celého obvodu byl proveden pomocí aplikace Tinkercad, která nabízí jak prostředí pro návrh 3D modelů, tak elektronických obvodů [54]. Samotný návrh zapojení koncového zařízení je zobrazen na Obrázek 9.

Aby komunikace mohla probíhat v pořádku, je třeba nejprve v metodě *setup*, která je spuštěna pouze jednou při spuštění, u každé Arduino desky provést registraci pomocí MAC adresy vybrané periferie pro připojení k síti internet. Adresa je v databázi vedená jako unikátní atribut, tudíž duplicitní adresy nebudou registrovány opětovně. Tuto adresu je třeba odeslat na API serveru společně s označením, o jaký typ koncového zařízení se jedná (světla, topení, zámek atd.) jednotlivé typy se rozlišují pomocí celočíselného identifikátoru, který je ve webové aplikaci pro přehlednější práci s daty reprezentován pomocí pojmenovaných celočíselných konstant zvaných Enum. Nakonec stačí ve webové aplikaci pomocí této adresy přiřadit registrované zařízení k požadované domácnosti.

Každé koncové zařízení bude potřebovat komunikovat se sítí internet, což zprostředkuje modul *ESP8266*. A podrobný návod k jeho připojení k Arduino desce, včetně veškerého potřebného kódu pro připojení k existující Wi-Fi síti, lze nalézt na [55]. K samotnému připojení je možné využít již existující knihovny, které přidáme pomocí příkazu *#include „název knihovny“* na úplném začátku kódu. Pro připojení modulu k existující Wi-Fi síti a volání metod v připraveném API budou v kódu zahrnuty následující knihovny.

```
#include <ESP8266WiFi.h>
#include <HTTPClient.h>
```

První knihovna usnadňuje práci s modulem a druhá umožňuje jednoduše vytvářet http požadavky a odesílat je na určitou adresu. Následně je možné v metodě *setup*, která je volána pouze při spuštění arduina, připojit k existující wifi síti a provést registrování modulu. Skrze připravenou API metodu *RegisterDevice*, která zaznamená nové koncové zařízení do databáze a umožní uživateli pomocí MAC

adresy přidat zařízení do jeho domácnosti. Před zavoláním je ale nejdříve nezbytné definovat následující proměnné a konstanty, které budou v kódu využívány.

- *ssid* – je název Wi-Fi sítě, ke které se chceme připojit
- *password* – je heslo, pomocí kterého je existující Wi-Fi síť zabezpečená
- *apiName* – označuje odkaz na API
- *macAddress* – je MAC adresa získaná z připojeného Wi-Fi modulu, kterou lze jednoduše získat pomocí metody *Wifi.macAddress*, tato metoda ale vrací pole bajtů a je třeba jej převést na textový řetězec, čehož je dosaženo nově vytvořenou metodou *mac2string*
- *enpointType* – je číselné označení typu koncového zařízení, které je definováno na straně serveru, prozatím server nabízí podporu 3 typů:
    - světelný – označeno číslem 1
    - tepelný – označeno číslem 2
    - zámek – označeno číslem 3

Smyčka *while* v metodě *setup* zajišťuje opakování pokusu o připojení každých 500 ms, dokud není připojení úspěšné. Po úspěšném připojení je zařízení registrováno pomocí odeslání výše zmíněných hodnot na API připraveného serveru. Dotaz je odeslán až po zavolání metody *http.GET*.

```
WiFi.begin(ssid,password);
 while (WiFi.status() != WL_CONNECTED)
{
        delay(500);
}
macAddress = mac2String(WiFi.macAddress());
String url = apiName+"/RegisterDevice/"+macAddress+"/"+endpointType;
http.begin(url);
int httpCode = http.GET();
http.end();
```

Další metodou, která na rozdíl od metody *setup* je volána v nepřetržité smyčce je nazvána *loop*. V případě koncového zařízení pro ovládání osvětlení jsou zde nepřetržitě získávány hodnoty a odesílány na server, stejně jako v případě registrace, nicméně je vše odesíláno na API metodu *NewValue*. Aby nedocházelo k zahlcení serveru nadbytečným množstvím dat, je nutné, aby se data neodesílala neustále, ale v našem případě stačí i jednotky sekund. I když budou data ze serveru získávána v řádech sekund, systém se stále bude jevit jako responzivní, bez ukládání nadbytečných dat a zbytečného zatěžování serveru. Čím větší bude prodleva, tím se bude systém jevit pomalejší, ale na druhou stranu se nebude na straně serveru zpracovávat nadbytečné množství informací, které by zbytečně zatěžovalo server.

K desce je připojeno tlačítko, které je také obsluhováno ve smyčce *loop* a při každém stisknutí přepíná proměnnou *endpointState* mezi hodnotami

- 1 – reprezentuje stav, kde koncové zařízení udržuje světlo neustále zapnuté
- 2 – světlo je neustále vypnuté
- 3 – hodnota z připojeného fotorezistoru je odesílána na server, který podle předepsaných pravidel pro dané koncové zařízení vrátí hodnotu, která určuje, zda má být světlo zapnuto, či nikoliv.

Výhodou tohoto tlačítka je fakt, že bude možné s jeho pomocí světla ovládat i pokud bude server vypnutý.

Fotorezistor, ze kterého jsou odečítána data, která jsou následně odesílána na server, musí být umístěn venku a měřit tak míru venkovního osvětlení a nesmí být umístěn ve stejné místnosti jako je světlo, které deska ovládá. Pokud by byl ve stejné místnosti a míra osvětlení dosáhla hodnot ve kterých server rozhodne o rozsvícení světla, následné rozsvícení světel by ovlivnilo další naměřenou hodnotu a server by si myslel, že je míra osvětlení dostačující a světla opět vypnul. Tento cyklus by se opakoval až do chvíle, kdy opět osvětlení při vypnutém světle dosáhne hodnot, kdy mohou světla zůstat vypnutá.

```
buttonState = digitalRead(btn1);
 if(buttonState == HIGH){
        endpointState = ((endpointState + 1)%3) + 1;
 }
 if(endpointState == 1)
 {
        lightOn();
 }
 else if(endpointState == 2)
 {
        lightOff();
 }
 else if(endpointState == 3)
 {
        delay(30000);
        int senzorValue = analogRead(A0);
        String url = apiName+"/NewValue/"+macAddress+"/"+senzorValue;
        http.begin(url);
        int httpCode = http.GET();
        if (httpCode > 0){
                String response = http.getString();
        if(response == "1")
        {
                lightOn();
        } else {
                lightOff();
        }
        http.end();
 }
```

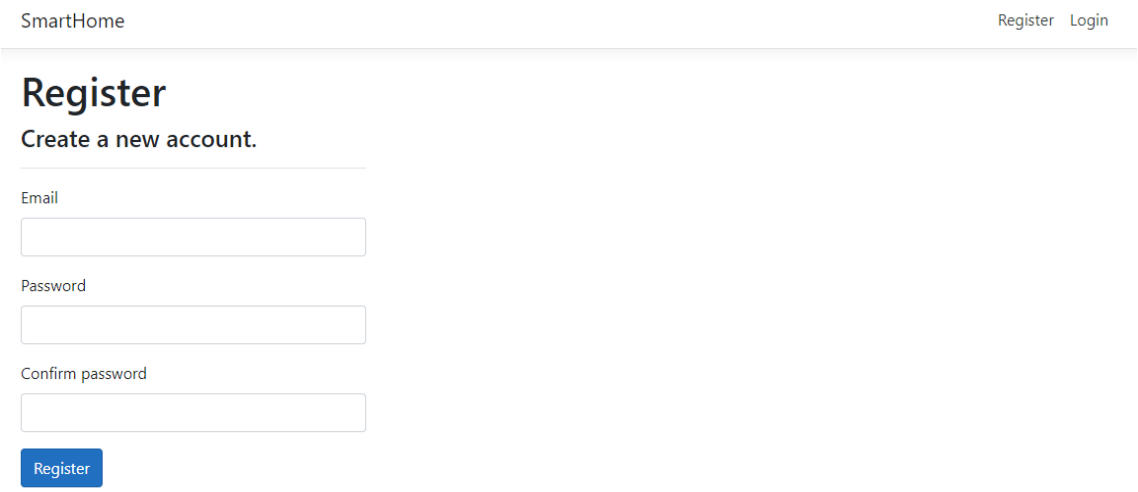
8 Testování aplikace

Jelikož je celá práce zaměřená na služby poskytované koncovým uživatelům, bude v této kapitole testována a předvedena funkcionalita webové aplikace společně s API rozhraním pro komunikaci s koncovými zařízeními.

8.1 Webová aplikace

Aby byla práce s celým systémem pro koncového uživatele příjemná, je třeba aby bylo grafické rozhraní aplikace dostatečně přehledné a intuitivní. Proto zde budou rozebrány jednotlivé funkce webové aplikace, společně s postupem k jejich dosažení.

Ke všem krokům je nezbytná registrace uživatele a jeho přihlášení v aplikaci (Obrázek 12). Zde jsou veškeré vstupy formuláře pro registraci řádně validovány a pokud uživatel zadá správně formátovaná data a emailová adresa nebyla doposud registrována, je automaticky přihlášen do aplikace na stránku se seznamem domácností, ke kterým má uživatel povolen přístup (Obrázek 11).



Obrázek 12: Registrace uživatele

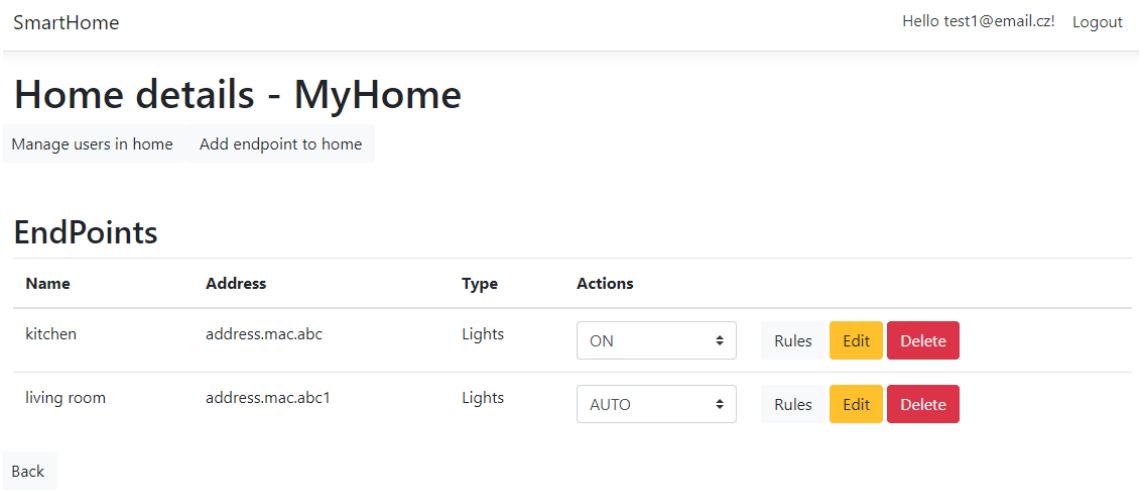
Po přihlášení uživatel smí také kliknout na svůj email v pravém horním rohu aplikace, kde může editovat své osobní údaje, povolit více faktorovou autentizaci, nebo smazat svůj účet společně s veškerými osobními daty.

V přehledu domácností je také možné vytvořit novou domácnost. K editování a mazání domácnosti má přístup pouze uživatel, který vybranou domácnost založil.

Při rozkliknutí detailu domácnosti je uživatel navigován na přehled koncových zařízení (Obrázek 13), která jsou k domácnosti přiřazená. Jsou zde také zobrazeny další ovládací prvky domácnosti. Uživatel zde může přímo v tabulce *EndPoints* pomocí nabídky ve sloupci *Actions* přepínat mezi stavy:

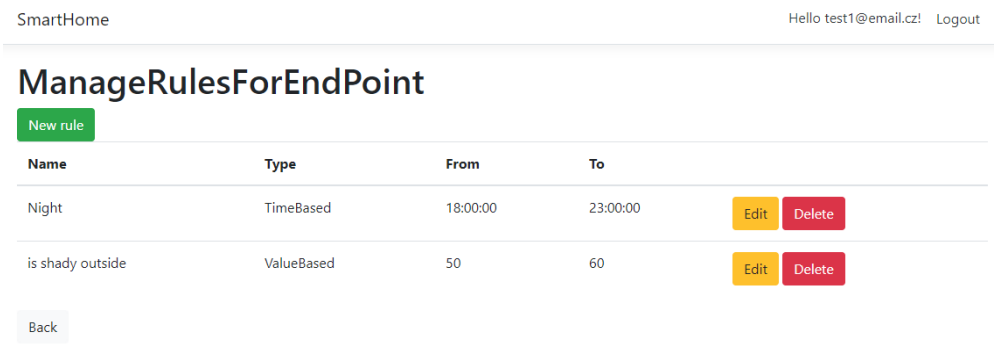
- ON – koncové zařízení je stále zapnuté
- OFF – koncové zařízení je stále vypnuté
- AUTO – koncové zařízení se řídí pravidly, která jsou mu přiřazena

Právo pro správu uživatelů domácnosti, společně s mazáním, přidáváním a editací koncových zařízení opět přísluší pouze uživateli, který domácnost vytvořil.



Obrázek 13: Detail domácnosti

V tabulce koncových zařízení domácnosti se lze pomocí tlačítka *Rules* navigovat na přehled pravidel daného koncového zařízení (Obrázek 14). K těmto pravidlům mají přístup pouze autorizovaní uživatelé, kteří byli oprávněni k přístupu správcem domácnosti. Pravidly se koncová zařízení řídí, pokud jsou přepnutá do stavu AUTO. Veškerá pravidla jsou vyhodnocována postupně a jakmile je nějaké splněno, ukončí se proces ověřování pravidel a koncové zařízení je zapnuto. Na obrázku níže, lze vidět, že koncové zařízení bude zapnuto každý den mezi 18-23 hodinou, nebo pokud hodnota osvětlení klesne pod stanovenou mez (tato hodnota se liší pro každou implementaci koncového zařízení a sám uživatel si musí zjistit jaké hodnoty jím navržené koncové zařízení bude odesílat na server).



Obrázek 14: Pravidla koncového zařízení

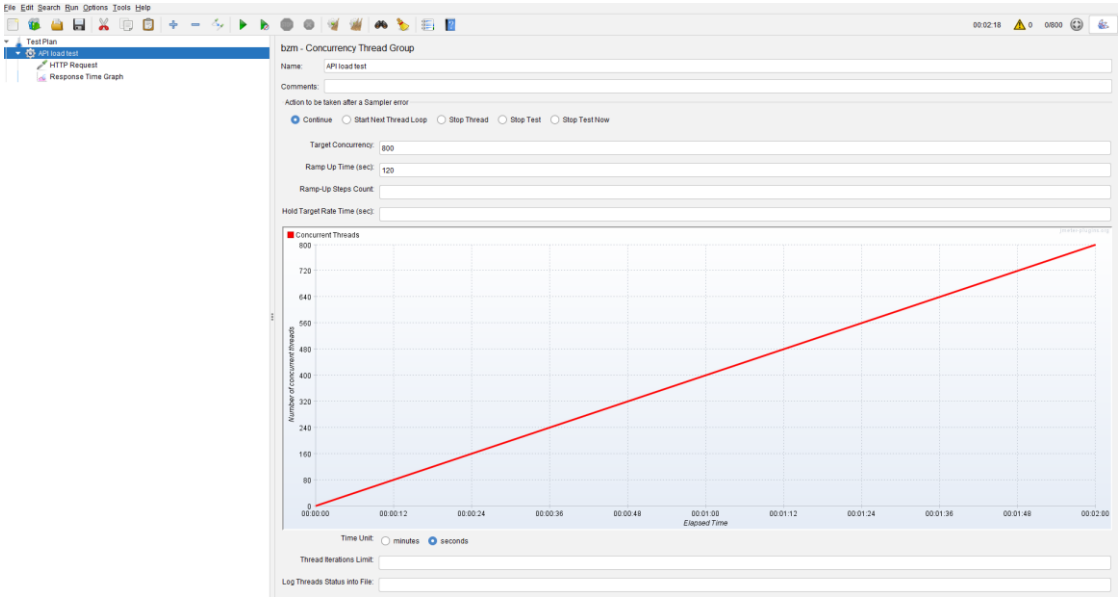
8.2 API

Jak už bylo mnohokrát řečeno, cílem celé aplikace je koncovému uživateli usnadnit implementaci vlastní chytré domácnosti tím, že nabídne API rozhraní, které poskytuje základní metody pro ovládání koncových bodů a webovou aplikaci popsanou v předchozí kapitole.

Je tedy třeba zjistit, kolik koncových bodů je možné reálně připojit, bez velkého zvýšení odezvy ze strany serveru. Jestli se celý systém bude jevit responzivní, závisí na několika faktorech. Mezi hlavní faktory v tomto případě patří především hardware serveru a optimalizace nabízených metod. Optimalizace samotná se nejprve nemusí jevit jako velice důležitá, ale při rostoucím počtu dotazů bude rozdíl mezi jejich potřebným časem na zpracování stále více patrný až do stavu, kde při stejném počtu dotazů může trvat serveru s neoptimalizovanými metodami jejich zpracování až o několik sekund déle.

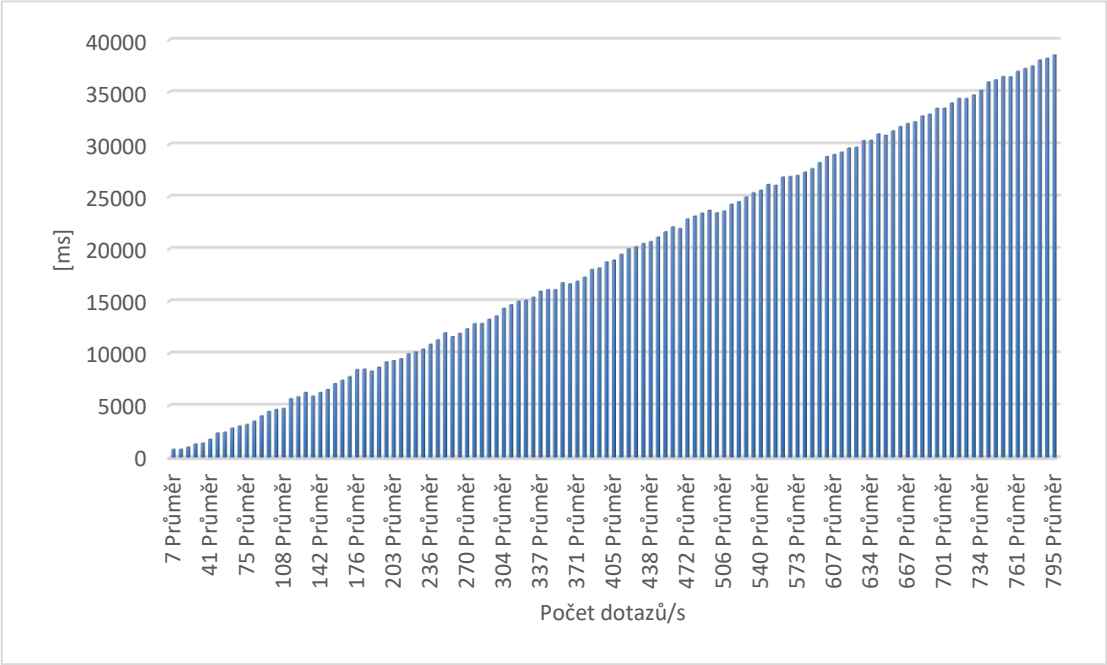
Server byl testován na počítači s procesorem *Intel i7 – 9700k* s pracovní frekvencí 3,6 GHz a *16 GB RAM* s frekvencí 2667 MHz. A k jeho otestování byl využit volně dostupný nástroj *Apache JMeter* [56], který společně s pluginem *Concurrency Thread Group* [57] měří čas odezvy serveru v závislosti na počtu dotazů za vteřinu. Na Obrázek 15 lze vidět vzhled aplikace *JMeter* společně s úvodním nastavením testovacích kritérií. Takto nastavená aplikace bude každou vteřinu postupně zvětšovat počet dotazů po dobu 2 minut, a to od úvodního jednoho dotazu za vteřinu až po celkových 800 dotazů za vteřinu. Bude testována metoda *NewValue*, která koncovým zařízením na základě definovaných pravidel určuje, zda mají být vypnutá,

či zapnutá a je tedy stěžejní, co se týče zjištění maximálního možného počtu připojených koncových zařízení.



Obrázek 15: Nastavení testovacích kritérií

Kvůli eliminaci chyb měření byl test desetkrát opakován a celkové časy, které server potřeboval pro zpracování všech dotazů, byly zprůměrovány. Takto zpracovaná data je možné vidět v podobě grafu na Obrázek 16. Na ose x je vybraných 25 zprůměrovaných počtů dotazů za vteřinu a na ose y je čas odezvy serveru v milisekundách.



Obrázek 16: Odezva serveru vzhledem k počtu dotazů

Z grafu je patrné že při 795 dotazech za vteřinu se aplikace stává prakticky nepoužitelná, jelikož odezva serveru dosahuje v průměru téměř 40 vteřin. Aby byla odezva v jednotkách vteřin a systém se tak jevil responzivní, je třeba aby počet dotazů byl pod 200 za vteřinu. Budeme-li předpokládat, že bude každý koncový bod na server odesílat dotaz každé 3 vteřiny, bude možné komunikovat celkově s 600 koncovými zařízeními. Pokud by pak měla jedna domácnost v průměru 4 místnosti a v každé místnosti alespoň dva koncové body (například topení a osvětlení), server s výše uvedenou konfigurací by mohl reálně obsloužit okolo 75 domácností.

Pokud by byl počet zařízení na jeden server vyšší, bylo by možné pořídit výkonnější hardware, nebo pro další domácnosti pořídit nový server. Vše je ale plně závislé na tom, že uživatelé budou při programování svých koncových zařízení dodržovat předem určenou frekvenci dotazů na server. Jelikož to se nedá nijak kontrolovat, je možné tento problém řešit pomocí tzv. *rate limitingu*, kterým lze omezit počet dotazů za časovou jednotku ze stejného zařízení [58].

9 Závěry a doporučení

Cílem práce byl návrh systému pro chytrou domácnost, který by bylo možné využít pro po domácku vyrobená koncová zařízení. Uživatel si tak může ušetřit práci s psaním vlastního serveru, a navíc se zbaví potřeby dedikovat jeden počítač pro běh již zmíněného serveru, čímž ušetří čas, který je potřebný k vývoji daného systému, i náklady, které jsou spojené s pořízením serveru a jeho provozem.

V kapitole určené pro průzkum existujících řešení bylo zjištěno že většina systémů chytrých domácností nabízí možnost řídit pouze chytrá zařízení, která splňují daná kritéria a jsou předem schválená výrobcem systému, jako je například Amazon Alexa, nebo Apple HomeKit. Existuje i velké množství systémů, které cíleně zamezují spolupráci s chytrými zařízeními od jiných společností a nabízí pouze řešení, které využívá výhradně koncová zařízení stejné společnosti. Nejblíže se cílům práce přiblížil systém OpenHAB, který nabízí volně dostupný software, umožňující řízení koncových zařízení pomocí skládání pravidel. I když samotný software je multiplatformní a podporuje běh systémech Linux, Windows, macOS i mikropočítačích jako je Raspberry Pi, stále potřebuje vlastní hardwarové zařízení, na kterém by běžel. Celé toto řešení tedy eliminuje čas, který je potřeba pro vývoj aplikace, ale náklady na pořízení a provoz serveru zůstávají.

Postupně se práce zaobírá návrhem vícevrstvé architektury systému, výběrem technologií pro vývoj jednotlivých vrstev. Dále se soustředí na návrh logické struktury jednotlivých vrstev a její implementaci s celkovým cílem snížení nákladů koncového uživatele.

Byla vyvinuta aplikace, která poskytuje nástroje pro správu uživatelů, chytrých domácností a koncových zařízení. Je v ní možné definovat pravidla a přiřazovat je k jednotlivým koncovým zařízením, podle kterých se budou daná zařízení zapínat či vypínat. Dále poskytuje API rozhraní s metodami pro komunikaci s koncovými zařízeními a případnými aplikacemi třetí strany.

Při testování aplikace bylo prokázáno, že je reálně možné obsloužit jedním serverem s konfigurací, která je uvedena v kapitole 8, okolo 75 domácností. Za zmínku také stojí při celém testování nedocházelo k žádným pádům serveru, ani chybovým hláškám, které by vznikly nadměrným počtem dotazů. I když čas odezvy

serveru s rostoucím počtem dotazů také stoupal, všechny dotazy byly správně zpracovány.

Nicméně bylo také zjištěno, že celý koncept jednoho centrálního serveru obsluhující všechny požadavky s sebou nese i řadu nevýhod. Jednou z nich je i fakt, že jelikož je daný server centrálním prvkem celého systému, který obsluhuje několik domácností, je tento prvek také největší slabinou celého systému. Pokud by došlo k výpadku serveru, všechny domácnosti by přestala fungovat, jak mají. Při výpadku internetu v domácnosti by byl scénář pro danou domácnost totožný. Je tedy nezbytné navrhnout koncová zařízení tak, aby bylo možné je ovládat i v případě výpadku komunikace se serverem a to manuálně, přímo v místnosti, kde je koncové zařízení umístěno. Stejně, jako je tomu v případě navrženého koncového zařízení pro ovládání osvětlení, které je možné ovládat pomocí tlačítka, připojeného přímo k Arduino desce. V neposlední řadě je třeba, aby uživatel, který koncové zařízení navrhl věděl, jaké hodnoty bude jeho senzor vracet, aby mohl adekvátně nastavit požadovaná pravidla pro ovládání koncového zařízení.

Ve výsledku tedy nový systém pomáhá snížit náklady a čas na zavedení chytré domácnosti a svůj cíl tedy splnil. Celý systém nabízí celou řadu oblastí pro další zkoumání. Jako například analýzu získaných dat pro zjišťování možných úspor, nebo možnou predikci chování uživatelů, systém by tak mohl ovládat domácnost, ještě před obdržením příkazu od uživatele. Prostorem pro další zlepšení systému by mohla být eliminace velké závislosti funkčnosti systému na jediném řídícím prvku, tedy serveru. Jednou z možností, jak eliminovat závislost funkčnosti celého systému pouze na serveru by mohlo být například přidání nové vrstvy do navržené architektury. Ta by se nacházela mezi koncovým zařízením a hlavním serverem. Kvůli snížení nákladů by mohla běžet i na mikropočítači a příslušel by jeden mikropočítač pro jednu domácnost. Tato vrstva by komunikovala s centrálním serverem, získávala a uchovávala pouze informace a pravidla o koncových zařízeních v její domácnosti. Následně by se mohla chovat ona jako řídící server pro jí přiřazená koncová zařízení a podle nastavených pravidel by byla schopná udržet domácnost v provozu i bez dočasného přístupu k internetu. U takto navržené architektury by kvůli pořizovací ceně mikropočítače sice stoupla cena pro koncového uživatele, ale klesl by celkový počet požadavků za sekundu z každé

domácnosti na hlavní server. Ten by následně mohl při stejné konfiguraci obsloužit více domácností. Dále by bylo možné se zaměřit na návrh a výrobu nových kompatibilních koncových zařízení, což by značně rozšířilo počet potencionálních uživatelů systému. Nebo návrh mobilní aplikace s využitím připraveného API rozhraní.

10 Seznam použité literatury

[1] The smart home: a glossary guide for the perplexed [online]. T3.com, 2015 [cit. 2020-03-25]. Dostupné z: https://www.t3.com/features/the-smart-home-guide

[2] SMIREK, Lukas, Gottfried ZIMMERMANN a Daniel ZIEGLER. Towards Universally Usable Smart Homes – How Can MyUI, URC and openHAB Contribute to an Adaptive User Interface Platform? CENTRIC. 2014, 2014, 29-38. ISSN 2308-3492.

[3] What is ubiquitous computing. IoTAgenda [online]. 2019 [cit. 2020-03-30]. Dostupné z: https://internetofthingsagenda.techtarget.com/definition/pervasive-computing-ubiquitous-computing

[4] BROWNE, Dermot. Adaptive User Interfaces. Elsevier Science, 2016. ISBN 1483294250.

[5] OpenURC [online]. 2009 [cit. 2020-03-30]. Dostupné z: https://www.openurc.org/

[6] OpenHab - empowering the smart home [online]. 2020 [cit. 2020-03-30]. Dostupné z: https://www.openhab.org/

[7] RASHIDI, P. a D.J. COOK. Keeping the Resident in the Loop: Adapting the Smart Home to the User. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans [online]. 2009, 39(5), 949-959 [cit. 2020-03-30]. DOI: 10.1109/TSMCA.2009.2025137. ISSN 1083-4427. Dostupné z: http://ieeexplore.ieee.org/document/5196706/

[8] JINDAL, Anish, Bharat Singh BHAMBHU, Mukesh SINGH, Neeraj KUMAR a Kshirasagar NAIK. A Heuristic-Based Appliance Scheduling Scheme for Smart Homes. IEEE Transactions on Industrial Informatics [online]. 2020, 16(5), 3242-3255 [cit. 2020-03-30]. DOI: 10.1109/TII.2019.2912816. ISSN 1551-3203. Dostupné z: https://ieeexplore.ieee.org/document/8695798/

[9] AHMED, Nadeem, Jahir Ibna RAFIQ a Md Rashedul ISLAM. Enhanced Human Activity Recognition Based on Smartphone Sensor Data Using Hybrid Feature Selection Model. Sensors [online]. 2020, 20(1) [cit. 2020-03-30]. DOI: 10.3390/s20010317. ISSN 1424-8220. Dostupné z: https://www.mdpi.com/1424-8220/20/1/317

[10] How to set-up an Alexa smart home [online]. Pocket-lint, 2019 [cit. 2020-04-09]. Dostupné z: https://www.pocket-lint.com/smart-home/news/amazon/143539-how-to-set-up-an-alexa-smart-home

[11] Understand the Smart Home Skill API [online]. Amazon.com, 2020 [cit. 2020-04-09]. Dostupné z: https://developer.amazon.com/en-US/docs/alexa/smarthome/understand-the-smart-home-skill-api.html

[12] Smart Home Devices & Systems [online]. Amazon.com, 2020 [cit. 2020-04-09]. Dostupné z: https://www.amazon.com/b?node=6563140011

[13] We compared Google Assistant, Amazon Alexa, and HomeKit to see which smart home platform is the best — and Alexa wins when it comes to device support [online]. Business Insider, 2020 [cit. 2020-04-09]. Dostupné z: https://www.businessinsider.com/homekit-vs-google-assistant-vs-amazon-alexa

[14] Apple HomeKit: What Is It, and How Do You Use It? [online]. Tom's Guide, 2020 [cit. 2020-04-09]. Dostupné z: https://www.tomsguide.com/us/apple-homekit-faq,review-4195.html

[15] HomeKit - All Accessories - Apple [online]. Apple, 2020 [cit. 2020-04-09]. Dostupné z: https://www.apple.com/shop/accessories/all-accessories/homekit

[16] Fibaro [online]. 2020 [cit. 2020-04-17]. Dostupné z: https://www.fibaro.com/cz/

[17] What is Z-Wave? [online]. Safety, 2019 [cit. 2020-04-17]. Dostupné z: https://www.safety.com/z-wave/

[18] OpenHAB [online]. 2019 [cit. 2020-04-17]. Dostupné z: https://www.openhab.org/

[19] Java | Oracle [online]. [cit. 2020-04-17]. Dostupné z: https://www.java.com/en/

[20] What is Cross-Platform Software? [online]. Medium.com, 2019 [cit. 2020-04-28]. Dostupné z: https://medium.com/@hakanasal51/what-is-cross-platform-software-38ee57b7304a

[21] GENERAL OVERVIEW OF CROSS-PLATFORM LANGUAGES [online]. Skelia, 2015 [cit. 2020-04-28]. Dostupné z: https://skelia.com/articles/general-overview-of-cross-platform-languages/

[22] A Look At 5 of the Most Popular Programming Languages of 2019 [online]. Stackify, 2019 [cit. 2020-04-28]. Dostupné z: https://stackify.com/popular-programming-languages-2018/

[23] Mono [online]. 2020 [cit. 2020-04-28]. Dostupné z: https://www.mono-project.com/

[24] .NET Goes Cross-Platform with .NET Core [online]. Microsoft Docs, 2016 [cit. 2020-04-28]. Dostupné z: https://docs.microsoft.com/en-us/archive/msdn-magazine/2016/april/net-core-net-goes-cross-platform-with-net-core#net-core

[25] What is Python? Executive Summary [online]. Python.org, 2020 [cit. 2020-04-28]. Dostupné z: https://www.python.org/doc/essays/blurb/

[26] An introduction to NuGet [online]. Microsoft Docs, 2019 [cit. 2020-05-02]. Dostupné z: https://docs.microsoft.com/en-us/nuget/what-is-nuget

[27] Entity Framework Core [online]. Entity Framework Tutorial, 2020 [cit. 2020-05-03]. Dostupné z: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx

[28] Entity Framework Core [online]. Microsoft Docs, 2016 [cit. 2020-05-03]. Dostupné z: https://docs.microsoft.com/en-us/ef/core/

[29] Introduction to LINQ Queries (C#) [online]. Microsoft Docs, 2015 [cit. 2020-05-03]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries

[30] Introduction to Identity on ASP.NET Core [online]. Microsoft Docs, 2020 [cit. 2020-05-03]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-3.1&tabs=visual-studio

[31] What is Bootstrap: A Beginners Guide [online]. Career Foundry, 2020 [cit. 2020-05-03]. Dostupné z: https://careerfoundry.com/en/blog/web-development/what-is-bootstrap-a-beginners-guide/

[32] TCP/IP (Transmission Control Protocol/Internet Protocol) [online]. TechTarget, 2020 [cit. 2020-05-06]. Dostupné z: https://searchnetworking.techtarget.com/definition/TCP-IP

[33] TcpListener Class [online]. Microsoft Docs, 2020 [cit. 2020-05-06]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.tcplistener?view=netcore-3.1

[34] How to write XML-RPC clients [online]. PaperCut Blog, 2017 [cit. 2020-05-06]. Dostupné z: https://blog.papercut.com/write-xml-rpc-clients/

[35] How to Call WCF Services and Create SOAP Services with ASP.NET Core [online]. Stackify, 2017 [cit. 2020-05-08]. Dostupné z: https://stackify.com/soap-net-core/

[36] RESTful API (REST API) [online]. TechTarget, 2020 [cit. 2020-05-08]. Dostupné z: https://searchapparchitecture.techtarget.com/definition/RESTful-API

[37] FIELDING, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Irvine, CA 92697, United States, 2000. Disertace. UNIVERSITY OF CALIFORNIA, IRVINE.

[38] Database Providers [online]. Microsoft Docs, 2019 [cit. 2020-05-08]. Dostupné z: https://docs.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli

[39] Introduction [online]. Arduino, 2020 [cit. 2020-05-12]. Dostupné z: https://www.arduino.cc/en/guide/introduction

[40] The Three Most Common Ethernet Speeds [online]. Small business, 2019 [cit. 2020-05-20]. Dostupné z: https://smallbusiness.chron.com/three-common-ethernet-speeds-69375.html

[41] Wi-Fi Channels, Frequencies, Bands & Bandwidths [online]. Electronics Notes, 2019 [cit. 2020-05-26]. Dostupné z: https://www.electronics-notes.com/articles/connectivity/wifi-ieee-802-11/channels-frequencies-bands-bandwidth.php

[42] What are Xbee Modules? [online]. Core electronics, 2018 [cit. 2020-05-27]. Dostupné z: https://core-electronics.com.au/tutorials/what-are-xbee-modules.html

[43] ARDUINO WIFI SHIELD [online]. Arduino, 2018 [cit. 2020-05-27]. Dostupné z: https://store.arduino.cc/arduino-wifi-shield

[44] ARDUINO UNO REV3 [online]. Arduino, 2020 [cit. 2020-05-27]. Dostupné z: https://store.arduino.cc/arduino-uno-rev3

[45] RELAY MODULE FOR ARDUINO [online]. Smartqat electronics [cit. 2020-06-30]. Dostupné z: https://smartqat.com/en/home/50-1-relay-module-for-arduino.html

[46] Arduino - Relay | Arduino Tutorial [online]. Arduino get started [cit. 2020-06-30]. Dostupné z: https://arduinogetstarted.com/tutorials/arduino-relay

[47] Relationships [online]. Microsoft Docs, 2019 [cit. 2020-07-02]. Dostupné z: https://docs.microsoft.com/en-us/ef/core/modeling/relationships?tabs=fluent-api%2Cfluent-api-simple-key%2Csimple-key

[48] Fluent API in Entity Framework Core [online]. Entity Framework Tutorial, 2019 [cit. 2020-07-02]. Dostupné z: https://www.entityframeworktutorial.net/efcore/fluent-api-in-entity-framework-core.aspx

[49] Dependency injection in ASP.NET Core [online]. Microsoft Docs, 2020 [cit. 2020-07-02]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-3.1

[50] Loading Related Data [online]. Microsoft Docs, 2016 [cit. 2020-07-02]. Dostupné z: https://docs.microsoft.com/en-us/ef/core/querying/related-data

[51] Overview of ASP.NET Core MVC [online]. Microsoft Docs, 2020 [cit. 2020-07-03]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1

[52] Razor syntax reference for ASP.NET Core [online]. Microsoft Docs, 2020 [cit. 2020-07-06]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-3.1

[53] Introduction - Bootstrap v4.5 [online]. Bootstrap, 2018 [cit. 2020-07-15]. Dostupné z: https://getbootstrap.com/docs/4.5/getting-started/introduction/

[54] Tinkercad Circuits [online]. Autodesk tinkercad, 2020 [cit. 2020-07-13]. Dostupné z: www.tinkercad.com

[55] How to Connect ESP8266 to WiFi | A Beginner's Guide [online]. Electronics hub, 2018 [cit. 2020-07-13]. Dostupné z: https://www.electronicshub.org/connect-esp8266-to-wifi/

[56] Apache JMeter [online]. Apache, 2020 [cit. 2020-07-31]. Dostupné z: https://jmeter.apache.org/

[57] Concurrency Thread Group [online]. Custom Plugins for Apache JMeter, 2020 [cit. 2020-07-31]. Dostupné z: https://jmeter-plugins.org/wiki/ConcurrencyThreadGroup/

[58] Everything You Need To Know About API Rate Limiting [online]. Nordic APIs, 2019 [cit. 2020-07-31]. Dostupné z: https://nordicapis.com/everything-you-need-to-know-about-api-rate-limiting/

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. Miloš Gábrle**
Studium:	I1700486
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Návrh a implementace arduino modulu jako prvku systému smart home**
Název diplomové práce AJ:	Design and Implementation of Arduiono Module as a Component of Smart Home Sytem

Cíl, metody, literatura, předpoklady:

Cílem práce je na základě analýzy existujících řešení navrhnout, realizovat a otestovat systém pro chytrou domácnost.

- Úvod
- Cíle práce
- Analýza
  - Vědecké přístupy
  - Existující řešení
  - Požadavky na nový systém
- Návrh architektury nového systému
- Výběr technologií pro implementaci
  - Server
  - Databáze
  - Elektronický modul
- Logický návrh systémových částí
  - Server
  - Databáze
  - Elektronický modul
- Implementace
- Testování a představení aplikace
- Závěr

- JINDAL, Anish, Bharat Singh BHAMBHU, Mukesh SINGH, Neeraj KUMAR a Kshirasagar NAIK. A Heuristic-Based Appliance Scheduling Scheme for Smart Homes. IEEE Transactions on Industrial Informatics. 2020, 16(5), 32423255 [cit. 2020-03-30]. DOI: 10.1109/TII.2019.2912816. ISSN 1551-3203. https://ieeexplore.ieee.org/document/8695798/
- RASHIDI, P. a D.J. COOK. Keeping the Resident in the Loop: Adapting the Smart Home to the User. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans. 2009, 39(5), 949-959. DOI: 10.1109/TSMCA.2009.2025137. ISSN 1083-4427. http://ieeexplore.ieee.org/document/5196706/
- AHMED, Nadeem, Jahir Ibna RAFIQ a Md Rashedul ISLAM. Enhanced Human Activity Recognition Based on Smartphone Sensor Data Using Hybrid Feature Selection Model. Sensors. 2020, 20(1). DOI: 10.3390/s20010317. ISSN 1424-8220. https://www.mdpi.com/14248220/20/1/317

Garantující pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	prof. Ing. Ondřej Krejcar, Ph.D.

Datum zadání závěrečné práce: 14.1.2015