ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

BAKALÁRSKA PRÁCA

POČÍTAČOVÉ INŽINIERSTVO

DANIEL JEŽÍK

Webová aplikácia pre vizualizáciu nameraných dát v IoT prostredí

Vedúci práce: Ing. Martin Húdik, PhD.

Registračné číslo: 28360820171299

Žilina, 2017

Čestné vyhlásenie

Čestne prehlasujem, že som prácu vypracoval samostatne s využitím dostupnej literatúry a vlastných vedomostí. Všetky zdroje použité v bakalárskej práci som uviedol v súlade s predpismi.

Súhlasím so zverejnením práce a jej výsledkov.

.....

V Žiline, dňa ..... Daniel Ježík

Poďakovanie

Chcem sa poďakovať vedúcemu práce Ing. Martinovi Húdikovi, PhD. za cenné rady, pripomienky a odborné vedenie pri vypracovaní bakalárskej práce.

ABSTRAKT V ŠTÁTNOM JAZYKU

JEŽÍK Daniel: Webová aplikácia pre vizualizáciu nameraných dát v IoT prostredí. [Bakalárska práca] – Žilinská Univerzita v Žiline. Fakulta riadenia a informatiky; Katedra technickej kybernetiky. – Vedúci práce: Ing. Martin Húdik, PhD. – Stupeň odbornej kvalifikácie: Bakalár – Žilina: FRI UNIZA, 2017. Počet strán: 37.

Bakalárska práca sa zaoberá riešením spracovania a vizualizácie nameraných dát vo webovej aplikácii prostredníctvom grafov alebo tabuliek, pričom na komunikáciu so zariadeniami využíva RESTové rozhranie. V úvode opisujeme riešenia dostupné na slovenskom a zahraničnom trhu. Priestor je venovaný vysvetleniu základných pojmov, s ktorými sa v tejto stretli. Teoretická časť sa podrobnejšie venuje téme IoT a RESTovému rozhraniu. Z viacerých hľadísk skúma ich praktické využitie, výhody a nevýhody. Práca opisuje frameworky, nástroje a balíčky, ktoré boli použité pri návrhu riešenia práce. To pozostáva z využitia frameworku Slim pre vytvorenie webovej aplikácie pre vizualizáciu nameraných dát a API rozhraním, ktoré dáta prijíma ale aj poskytuje tretím stranám. Záver je venovaný výsledku práce. Popísaný je pracovný postup nášho vlastného riešenia, princíp činností vybraných komponentov ako aj samotnej webovej aplikácie. Súčasťou práce sú náhľady kódov a užívateľská príručka. Popisuje možnosti, ktoré užívateľ môže vo webovej aplikácii vykonávať a ako s ňou môže pracovať.

Kľúčové slová: IoT, framework, Bootstrap, Slim, REST, API

ABSTRAKT V CUDZOM JAZYKU

JEŽÍK Daniel: Web application for visualization of measurement data in IoT applications. [Bachelor Thesis] – University of Žilina. Faculty of Management Science and Informatics; Department of Technical Cybernetics. – Supervisor: Ing. Martin Húdik, PhD. – Degree of professional qualifications: Bachelor – Žilina: FRI UNIZA, 2017. Number of pages: 37.

The Bachelor Thesis deals with the solution of processing and visualisation of measured data in web application within charts or tables while using the RESTful web services to communicate with devices that are using this web application. The Introduction contains some of the solutions available in Slovak and foreign markets. The space is devoted also to explanation of some basic terms. The theoretical part is devoted to IoT and REST interface their practical use, advantages and disadvantages from multiple levels. The Thesis is devoted to the description of frameworks, tools and packages, which were used during the proposition of the solution for the work. It consists of the usage of the Slim framework, used for creation of the web application for visualisation of measured data and API interface that can receive the data and provide this data to third parties. The part in our Thesis are code previews and user's guide. In the Conclusion of our Bachelor Thesis is described the workflow of our own solution, principles of selected components as well as web application itself.

Key words: IoT, framework, Bootstrap, Slim, REST, API

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam grafov

Zoznam skratiek

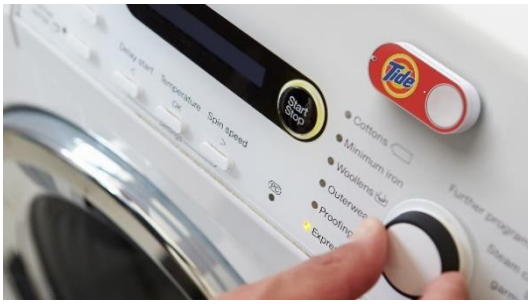
AJAX	Asynchronous JavaScript and XML
API	Application programming interface
CoAP	Constrained Application Protocol
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
M2M	Machine to machine
MQTT	Message Queue Telemetry Transport
MVC	Model–view–controller
REST	Representational state transfer
RPC	Remote procedure call
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WSDL	Web Services Description Language
XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

1 Úvod

Pojem IoT sa do nášho života dostáva čoraz viac aj keď o tom vôbec nevieme alebo si to len neuvedomujeme . Ťažko by sme čakali na odpoveď na význam pojmu Internet of things, či vysvetlenie pojmu Internet vecí od človeka, ktorý sa nevenuje informačným technológiám, aj keď je veľký predpoklad hlavne u mladšej generácie, že sa s touto technológiou stretáva a nevedomky ju používa aj v súčasnosti. Možno ste si práve položili otázku „kde a kedy“ ?

Napríklad medzi jednoduchý prvok IoT zariadenia môžeme zaradiť aj bezpečnostnú kameru, ktorá je cez počítač pripojená na internet, jej vysielaný obraz tak môžeme sledovať prostredníctvom webovej stránky kdekoľvek. Táto technológia pred pár rokmi nezažívala až taký boom, dovolil by som si tvrdiť, že príchod smartfónov vo veľkom ovplyvnil jej vývoj. Dnešný trend vyvíjaných smartfónov s veľkými displejmi, rýchlym pripojením na internet, či veľkým výpočtovým výkonom si našli široký okruh využitia. Užívateľ tak nemusí využívať notebook či domáce PC na prácu s IoT.

Moje prvé zoznámenie sa s touto technológiu sa viaže s rokom 2014, kedy americká spoločnosť Amazon uviedla na trh svoj produkt s názvom Amazon Dash. Ide o malé zariadenie s jedným tlačidlom (viď. obr. 1.1), ktoré je pomocou Wi-Fi pripojené na internet. Toto zariadenie si umiestnime, v tomto prípade do blízkosti práčky. Ak sa prací prášok minie, jediným stlačením tlačidla vykonáme objednávku. Tieto zariadenia vyžadujú minimálnu údržbu ako napr. výmenu batérie. Jeho pripojenie na Wi-Fi nakonfigurujeme pomocou smartfónu. Aktuálna cena za toto zariadenie je 4,99 €, je vyrábané v rôznych variantoch, ktoré nám umožňujú objednávať tovar alebo služby. [1]



Obrázok 1.1 Amazon dash BUTTON v praxi

Možnosť využiť nejaké takéto zariadenie napr. doma nás nadchla. Predstavili sme si obrovský potenciál, využitia tohto zariadenia. Vtedy v nás skrsol nápad, viac sa zamerať na oblasť senzorov. Konkrétne, ich využitie v domácom prostredí . Vedeli sme si ich predstaviť napr. ako merače teploty, vlhkosti prípadne senzor pohybu. Vzhľadom na to, že v roku 2014 sme mali nedostatočné skúsenosti v oblasti programovania, nedokázali sme sa bližšie venovať štúdiu „smart“ zariadení. Avšak myšlienka vytvorenia inteligentného domu nás nadchla. Na konci prvého ročníku na vysokej škole sme sa stretli s použitím vývojovej dosky Arduino . Tu sa začali naše prvé pokusy, od jednoduchého rozsvecovania LED diód až po zložitejšie simulácie. Išlo o simuláciu parkoviska, ktorá obsahovala senzory vzdialenosti, pohybu, či servomotory na ovládanie rampy.

Náš posledný projekt so zariadením Arduino má veľa spoločných prvkov s našou bakalárskou prácou. Pomocou senzorov vlhkosti a teploty sme zaznamenávali hodnoty týchto veličín. Tie sa nám zobrazovali na webovej stránke s malým oneskorením. To nám vnuklo nápad zaoberať sa touto problematikou komplexnejšie a tak sa zrodila téma našej bakalárskej práce. Vytvorenie webovej aplikácie, pomocou ktorej by sa dali zobrazovať namerané hodnoty zo zariadení, ktoré disponujú týmto senzormi na smartfóne alebo tablete. Naše prvé predstavy o téme bakalárskej práce sa uberali iným smerom, prvotne išlo o riešenie problematiky inteligentného domu. Nakoľko išlo o široký záber, po konzultácii s vedúcim práce sme sa rozhodli zamerať sa vo svojej práci konkrétne na vizualizáciu nameraných dát vo webovej aplikácii.

V súčasnosti sa stretávame stále častejšie so zariadeniami, ktoré sa dokážu pripojiť na internet a pomocou notifikácií nás dokážu informovať o udalostiach, ktoré sa dejú v našej blízkosti. Takýto systém je štandardný prvok v modernom inteligentnom dome. Pomocou tohto systému, prostredníctvom smartfónu či tabletu si môžeme rozsvecovať svetlo, odostrieť žalúzie, regulovať teplotu alebo spúšťať klimatizáciu.

Trend vývoja tejto technológie rokmi postúpil. Stal sa finančne dostupným pre široký okruh užívateľov. Priekopníkom v nej firma Philips, ktorá uviedla systém domáceho inteligentného osvetlenia HUE pre domácnosti a užívateľ ho tak môže pomocou smartfónu intuitívne ovládať . [2] V tomto trende nezaostali ani internetový giganti ako spoločnosť Google či Apple, ktorá predstavila svoju aplikáciu HomeKit. [3] Tá umožňuje využívať jej funkcie pomocou inteligentného osobného asistenta Siri, ktorá je súčasťou ich mobilných telefónov.

Na Slovensku na tento trend upozorňujú rôzne propagačné akcie . Ide napr. o akciu s názvom IoT Expo Bratislava a stále častejšie prednášky konajúce sa na vysokých školách, o ktoré je u študentov veľký záujem. Zo slovenských spoločností je to napr. spoločnosť Slovanet a.s. , ktorá spustila svoj IoT pod názvom Smart Connect. [4]

2 Súčasný stav riešenej problematiky doma a v zahraničí

V súčasnosti máme možnosť si vyberať si z veľkého množstva platforiem, ktoré internet ponúka zadarmo alebo riešenia od firiem, ktoré sa naplno venujú tejto problematike. Tieto služby môžeme rozdeliť do dvoch kategórií:

- služby, ktoré sú otvorené
- služby, ktoré sú vytvorené pre určitý segment produktov, a teda zväčša je to riešenie konkrétnej značky

Vzhľadom na to, že takýchto platforiem je mnoho, priblížime si len niekoľko riešení, ktoré nás bližšie uvedú do problematiky.

Jedno z riešení nám ponúka spomenutá spoločnosť Amazon. Platforma sa nazýva AWS. Táto platforma je cloudová a umožňuje pripojeným zariadeniam jednoduchú a bezpečnú interakciu s cloudovými aplikáciami a ostatnými zariadeniami. Samozrejme disponuje dátovým úložiskom, štatistikami atď. O týchto funkciách si povieme neskôr. [5]

Obrázok 2.1 Amazon Web Services

Ďalšie riešenie ponúka na Slovensku spoločnosť Slovak Telekom s ich službou SMARTHOME. Toto riešenie je práve z kategórie, ktorú sme spomenuli vyššie. Táto služba využíva určitý sortiment zariadení a teda pokrýva len službu inteligentného domu. Užívateľ pomocou mobilnej aplikácie spolu so zakúpením zariadení podľa jeho výberu môže vybaviť svoj dom inteligentnými prvkami, a tak ho premeniť na inteligentný dom [6].

Obrázok 2.2 Logo TELEKOM SMART Home

Vzhľadom na to, že slovenský trh nedisponuje riešením pre IoT, ktoré by nás dostatočne zaujalo zamerali sme sa na zahraničnú službu, ktorá ponúka spoločnosť Ubidots. Ich služba je známa pod názvom Ubidots. Túto platformu sme si vybrali kvôli tomu, že je inovatívna, prehľadná a efektívna. Naša práca sa teda v štádiu vývoja zameriava práve na vývoj podobnej aplikácie ako ponúka toto riešenie akurát v odľahčenej forme a teda bude sa vyznačovať značne menšou komplexnosťou.

Služba ponúka viac ako 50 knižníc pre zariadenia pre rýchle pripojenie týchto zariadení do ich služby. Tieto knižnice sú stále testované a vylepšované ich komunitou. Vzhľadom na to, že takéto zariadenia môžu podľa konfigurácie generovať veľké množstvo záznamov disponuje táto služba inteligentným správcom dát, ktorý ich vie vhodným spôsobom zhromažďovať a kategorizovať a pri prípade nehody ich obnoviť. Takéto záznamy z rôznych zariadení môžu prichádzať v rôznom formátovaní, kvôli tomu je takáto služba vybavená API rozhraním, ktoré je viac protokolové a teda dokáže vyhovieť rôznym formátom. Live Dashboards je užívateľský panel nástrojov, je to akoby princíp pracovnej plochy, kde vidíme prehľad zo zvolených senzorov v reálnom čase. Tieto dáta môžeme zdieľať cez verejné odkazy či zobrazovať ich na mobilných zariadeniach. [7]

2.1 IoT - Internet vecí

2.1.1 Čo je internet vecí?

Internet vecí (IoT - skratka z anglického Internet of Things) je názov pre technológie umožňujúce prepojenie zariadení / senzorov s nízkou spotrebou a nárokmi na objem prenesených dát. Má široké využitie v rôznych oblastiach so širokým spektrom možností, ako napr. vizualizácia, spracovanie, vyhodnocovanie dát. Budú náhradou za mnohé doteraz používané zariadenia a systémy na báze mobilných sietí. [8, 9]

2.1.2 Ako veľký je rozsah IoT ?

Vzhľadom na to, že éra konektivity notebookov a smartfónov je už dávno za nami kráčame do doby konektivity áut, nositeľného príslušenstva, inteligentných domov, inteligentných miest alebo zdravotníctva. Zjednodušene, smerujeme do doby, kedy bude prepojené prakticky všetko so všetkým. Správa spoločnosti Gartner, lídra v oblasti prieskumov informačných technológií, hovorí, že do roku 2020 bude prepojených naprieč všetkými technológiami 20,6 miliardy zariadení. [10, 11]

Tabuľka 2.1 Nárast počtu pripojených zariadení na základe prieskumu spoločnosti Garnet [10]

Rok	Počet pripojených zariadení
1990	0.3 miliónu
1999	90 miliónov
2010	Prihlasov5 miliárd
2013	9 miliárd
2025	1 bilión

2.1.3 Prínosy / výhody

Internet vecí sa stáva vítaným prvkom pre jeho úsporu peňazí. Úspory sa jednak môžu odzrkadliť na priaznivom vývoji firmy a tiež sa môže premietnuť do ceny produktov alebo služieb. IoT sa ukazuje ako veľmi užitočné v každodennej rutinnej práci tým, že zariadenia vzájomne komunikujú účinným spôsobom, čím šetria energiu a náklady s nimi spojené.

Tieto zariadenia umožňujú zdieľať dáta medzi sebou a ich transformáciu požadovaným spôsobom, to robí tieto systémy efektívne. Výhodou je predikcia, ktorá v konečnom dôsledku môže ušetriť množstvo času a finančných prostriedkov. Realtime monitoring týchto prostriedkov dokáže efektívne pomáhať v rôznych infraštruktúrach. Jej pomoc spočíva v pomoci s riadením, tým ju zefektívňuje a zrýchľuje, čím šetrí peniaze a čas.

- **Bezpečnosť, komfort, účinnosť**

Predstavte si, že môžete merať a kontrolovať nebezpečné prostredia bez toho, aby ste vystavili ľudí nejakému riziku pričom takéto meranie môže byť nepretržité a strojovo presné. Obslužný pracovník môže namerané údaje komfortne sledovať z pracoviska. Napríklad inteligentné montážne linky nahlásia chyby a varovania v reálnom čase, tak dokážu produkovať vyššie výnosy a menej prestojov. [12]

- **Informácia**

Je zrejmé, že viac informácií pomáha robiť lepšie rozhodnutia. Či už ide o banálne rozhodnutie vedieť, čo kúpiť v obchode alebo či má vaša firma dostatok materiálov a zásob, vedomosť je veľmi nutná. Pri nepoznaní takýchto faktov môže dôjsť k pozdržaniu či zastaveniu výroby a následným stratám. [12, 13]

- **Spotrebiteľské preferencie**

IoT môže analyzovať spotrebiteľské preferencie takým spôsobom, aby mu na základe návykov obsahu vyhľadávania, pohybu či aktivity boli ponúkané reklamné produkty. V prípade inteligentných chladničiek by mohlo ísť o informácie ako napr. kedy bude končiť trvanlivosť mlieka, koľko mlieka v chladničke je, kde bolo kúpené a aká je jeho aktuálna cena. Takýmto spôsobom vie informovať vopred o potrebe doplnenia potravín. Užívateľ tak má prehľad o potravinách. Tým sa zmenšuje predpoklad ich hromadenia a zároveň zabraňujeme vytváraniu odpadu, čo šetrí peniaze a čas strávený nákupmi . [12]

2.1.4 Zápory / nevýhody

- **Bezpečnosť**

Určite vám neunikla správy, ktoré boli prezentované v televíznych novinách o firmách, ktoré boli napadnuté hackermi. Takéto útoky sú stále častejšie. Útočník sa nevystavuje priamemu riziku a v konečnom dôsledku profituje zo svojho úkonu. Hackerský útok je za účelom krádeže identít, hesiel, kreditných kariet dokonca odcudzenie kľúčov áut, ktoré komunikujú prostredníctvom mobilných aplikácií. Skutočnosť, že sú tieto zariadenia pripojené do sietí či na internet ich vystavuje bezpečnostnému riziku. Vzhľadom na tento fakt sa stále posilňuje bezpečnosť takýchto zariadení aby sa predišlo ich zneužitiu. Jedným z najväčších nedostatkov je pomalé vydávanie bezpečnostných aktualizácií či zdokonaľovanie samotného hardwaru v novších zariadeniach. [12, 13]

- **Súkromie**

Otázkou je, kto sleduje tých, čo „sledujú“ ? Internet vecí totiž sprostredkováva obrovskú databázu citlivých dát. Pri inteligentnom dome je jednoduché zistiť vaše návyky, kedy idete spať, kde sa pohybujete, kedy ste v práci... Na jednej strane sa systém javí ako dobrý sluha, ale na druhej strane ako zlý pán. Najmä v dnešnej dobe kedy zákony rôznych krajín umožňujú sledovanie, v niektorých krajinách, ľudí len na základe podozrení. [12, 13]

- **Spracovanie dát**

Internet vecí vytvára nespočetné množstvo dát, ale obchodná hodnota týchto dát sa nemeria v bajtoch ,ale v ich analýze. Samotné namerané dáta nemajú žiadnu trhovú hodnotu. Ak si tieto dáta chceme predstaviť väčšinou ide len o číslo 1 alebo 0. V praxi väčšia komplexnosť znamená väčšiu šancu pre vznik chyby. Pokiaľ takéto dáta nedokážeme efektívne spracovať a vyhodnotiť stane sa to, že nás monitorovacie zariadenia len zasypú údajmi, čo bude viac neefektívne ako efektívne využitie ich potenciálu.

Takéto obrovské množstvo dát, ktoré sa neskôr analyzuje sa nazýva BIG DATA. Vzhľadom na to, že takýto objem dát nie je možné len tak analyzovať, spracovanie týchto dát podlieha niekoľkým fázam spracovania, zhromažďovanie, filtrovanie, fúzia, spracovanie a ich archivovanie. Spravovanie takéhoto množstva dát často podlieha automatizácií či používaní modelov pre spracovanie. [14, 15]

- **Kompatibilita**

V internete vecí neexistujú žiadne medzinárodné štandardy pre kompatibilitu. Čo môže mať za dôsledok problém pri komunikácií so zariadeniami, keďže často pochádzajú od rôznych výrobcov a z rôznych krajín. [12, 13]

- **Nahradenie človeka strojom**

Automatizácia môže byť dobrá vec, pokiaľ vás kvôli nej nenahradí v práci stroj. Na jednej strane spoločnosť investuje do modernizácie, skvalitnenia a zrýchlenia práce no na druhej strane na úkor ľudskej pracovnej sily. Ide o vývoj, kedy v našom živote stále viac zaberajú miesto zariadenia a technológie na úkor človeka. Človek tak stráca svoje základné interakcie s inými ľuďmi. [12, 13]

2.1.5 Aplikácie

Na základe internetového vyhľadávania sme vybrali 5 najčastejších využití:

- **Inteligentný dom**

Inteligentný dom alebo Smart Home je položka IoT, ktorá vyčnieva najviac zo všetkých aplikácií a zažíva najväčší boom v danej oblasti. Viac ako 60.000 ľudí v súčasnosti vyhľadáva termín "Smart Home" každý mesiac prostredníctvom webových vyhľadávačov. Predpokladá sa, že inteligentné domy budú v budúcnosti tak bežné ako mobilný telefón. Prvky takéhoto inteligentného domu dokážu efektívne pracovať so svetlami, vykurovaním a pod. a teda sľubujú úsporu energií, času a peňazí. [10]

- **Nositeľné príslušenstvo**

Záujem o nositeľné príslušenstvo v posledných rokoch prudko vzrástol na celom svete. Pod pojmom nositeľné príslušenstvo alebo z anglického slova „Wearables“ si môžeme predstaviť inteligentné náramky, hodinky, pásy na hruď a mnoho iných zariadení. Sú to zariadenia, ktoré disponujú senzormi. Tie zbierajú dáta a tiež nesú informáciu o ich užívateľovi ako sú napríklad výška, váha, vek, pohlavie. S týmito dátami neskôr pracujú aplikácie či ľudia, ktorí tieto dáta vyhodnocujú. Tieto zariadenia sú najčastejšie využívané pri športovej činnosti a v zdravotníctve. [10]

- **Automobilový priemysel**

Jeden z najhorúcejších príkladov je vývoj autonómnych "bez vodičových" motorových vozidiel. Svetová zdravotnícka organizácia uviedla najnovšie výsledky výskumu, ktoré hovoria, že ročný počet úmrtí spojených s autonehodou na celom svete je viac ako jeden milión. Väčšina týchto úmrtí je zapríčinená v dôsledku ľudskej chyby. IoT technológie, najmä vzostup bezpečnostne zameraných senzorov v automobiloch, má potenciál výrazne znížiť počet úrazov a úmrtí zapríčinených motorovým vozidlom. [10, 15]

- **Inteligentné mestá**

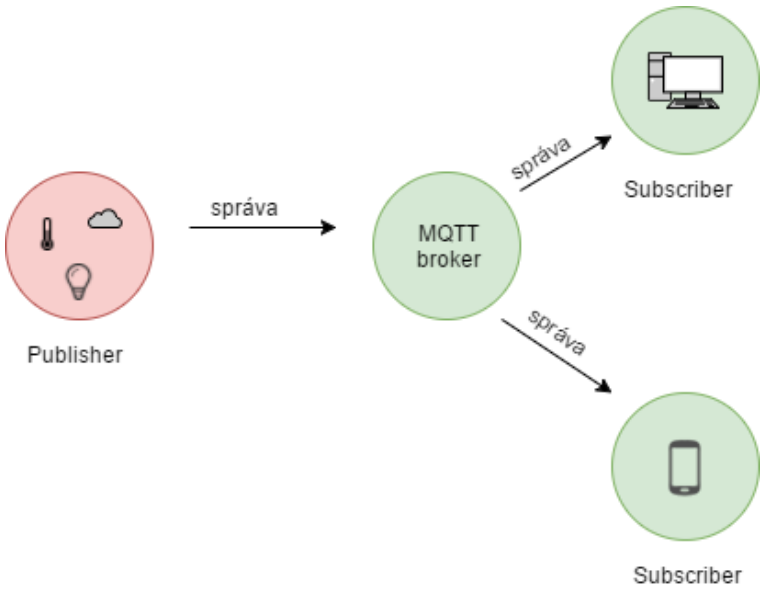
Inteligentné mestá ponúkajú rôzne využitie IoT napr. riadenie dopravy, distribúcia energií, monitorovanie životného prostredia či bezpečnosť. Faktom je, že dnes sú takéto mestá realitou. Systémy sú efektívne a nenáročné a riešia problémy miest a obyvateľov. [10]

- **Poľnohospodárstvo**

  Vzhľadom na stále stúpajúci dopyt po potravinách bolo nutné využiť pokročilé techniky a výskum na zvýšenie produkcie potravín. Poľnohospodári využívajú takto nahromadené dáta pre lepšie využitie znalostí za účelom zvýšenia efektívnosti a produktivity výroby pri nižších nákladoch. IoT v takomto odvetví dokáže farmárovi dopomôcť pri zisťovaní faktorov ako je hodnota vlahy či živín v pôde, spotreba vody pre rastliny či ich automatické zavlažovanie alebo automatické kŕmenie dobytku. [10]

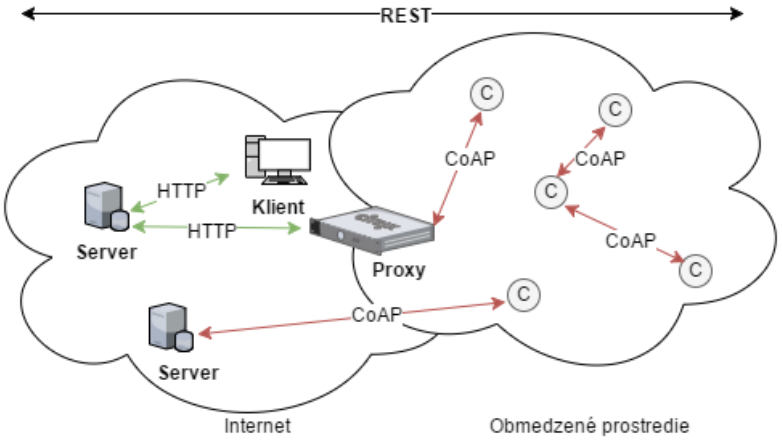
2.1.6 Protokoly pre IoT

MQTT (Message Queuing Telemetry Transport) je jednoduchý a nenáročný protokol pre predávanie správ medzi klientmi prostredníctvom centrálneho bodu tzv. brokeru. Pri protokole MQTT prebieha prenos pomocou TCP a používa návrhový vzor publisher – subscriber. Týmto spôsobom je možné v sieti na báze TCP/IP zverejňovať (publish) informácie, ktoré potom broker prepošle klientom alebo zariadeniam, ktoré túto správu odoberajú (subscribe). [17, 18, 19]



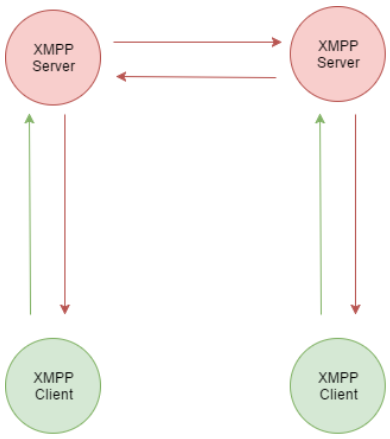
Obrázok 2.3 Popis MQTT architektúry

CoAP (Constrained Application Protocol) je webový komunikačný protokol založený na RESTovom modely. Je hlavne určený pre odľahčenú M2M komunikáciu vzhľadom na použitú binárnu hlavičku a nie textovú ako používa protokol http, z ktorého tento protokol vznikol. [17, 19, 20]



Obrázok 2.4 Popis CoAP architektúry

XMPP (Extensible Messaging and Presence Protocol) je otvorený protokol používaný na sieťovú komunikáciu, posielanie správ či zisťovanie stavu založený na XML.[20]



Obrázok 2.5 Popis XMPP architektúry

2.2 REST

REpresentational **S**tate **T**ransfer v skratke REST alebo RESTful. REST je webové rozhranie, ktoré je založené na architektúre, ktorá využíva HTTP protokol pre dátovú komunikáciu. Pojem REST bol definovaný v roku 2000 Royom Fieldingom v jeho dizertačnej práci. REST je orientovaný dátovo nie procedurálne, webové služby definujú vzdialené procedúry a protokol pre ich volanie, REST určuje ako sa bude pristupovať k dátam. Týmto sa líši od známejších technológií ako sú napr. RPC, WSDL či SOAP.

V RESTovej architektúre sú zdroje tzv. resources ľahko prístupné. Každý takýto zdroj má vlastný identifikátor URI. REST využíva viaceré reprezentácie zdrojov a to ako Text, JSON alebo XML, kde dnes najpoužívanejší formát pre webové služby je JSON a pre prístup k týmto dátam definuje 4 základné metódy a to GET, POST, PUT a DELETE. Webové služby založené na RESTovej architektúre sú známe pod pojmom RESTful Web Services. Takmer každý zásadný vývojový jazyk dnes obsahuje framework na tvorbu RESTful služieb a teda API. [22, 23]

- GET (Retrieve) – táto operácia sa využíva na získanie zdroja
- POST (Create) – táto operácia sa využíva na vytvorenie zdroja
- PUT (Update) – táto operácia sa využíva na zmenu hodnoty zdroja
- DELETE – táto operácia sa využíva na zmazanie zdroja

Obrázok 2.6 Popis RESTful architektúry

2.2.1 HTTP stavové kódy (status code)

Stavové kódy sú odpovede (response) zo strany servera, ktorý nám pomocou kódu hovorí aká akcia bola vykonaná na základe požiadavky (requestu). Napríklad si môžeme uviesť známy príklad stavového kódu, s ktorým sa užívateľ často stretáva a to 404 Not Found. Tento stavový kód nám hovorí, že sa pokúšame pristúpiť k zdroju, ktorý neexistuje. Vzhľadom na to, že HTTP je rozšíriteľný protokol je možné generovať vlastné stavové kódy. Z hľadiska vývoja by malo byť prioritou využívať všetky stavové kódy, no táto požiadavka sa nie vždy plní a preto sa zväčša používajú iba tieto stavové kódy : [21]

Stavové kódy sa rozdeľujú do 5 kategórií :

- 1xx Informational – informačné
- 2xx Success – úspešné
- 3xx Redirection – presmerovanie
- 4xx Client Error – chyba klienta
- 5xx Server Error – chyba serveru

Tabuľka 2.2 Stavových kódov

Kód	Popis
200	**OK** – požiadavka prebehla v poriadku
201	**Created** – pri POST, pokiaľ bol vytvorený nový obsah
304	**Not Modified** – pokiaľ od poslednej požiadavky nebol zmenený obsah
400	**Bad Request** – zlá požiadavka na server
401	**Unauthorized** – klient je neoverený
403	**Forbidden** – klient nemá prístup k danému obsahu
404	**Not Found** – zdroj neexistuje
422	**Unprocessable Entitiy** – chyba validácie dát
500	**Internal Server Error** – na serveri došlo k neočakávanej chybe

2.2.2 RESTFul Web Services / RESTové webové služby

V takomto prípade môžu aplikácie naprogramované v rôznych programovacích jazykoch bežiace na rôznych platformách využívať webové služby pre výmenu dát cez Internetovú sieť obdobným spôsobom akoby prebiehala komunikácia medzi procesmi na jednom počítači. Vďaka týmto otvoreným štandardom potom môžu medzi sebou jednoducho komunikovať systémy ako Windows a Linux či Java a Python atď.

Webové služby umožňujú jednoduchú komunikáciu medzi aplikáciami, najčastejšie pomocou jazyku XML a protokolu HTTP.[24]

2.2.3 REST v IoT

REST a celkovo aj API hrajú dnes v oblasti IoT zásadnú úlohu. Bez webového API by IoT platformy celkom stratili svoj význam. REST sa svojou jednoduchosťou a ľahkosťou stáva dokonalým riešením pre komunikáciu medzi zariadením a IoT platformou. REST tak pracuje ako aj vstupná brána pre prichádzajúce dáta zo zariadení. Tak aj spracúva požiadavky od iných klientov, ktorým svoje dáta pomocou webového API sprístupňuje.

2.2.4 API

„**Application programming interface** alebo skratkou **API** (rozhranie pre programovanie aplikácií). Je zbierkou procedúr, funkcií či protokolov a nástrojov pre budovanie softvérových aplikácií.

2.2.5 Vlastnosti RESTu

Každý systém využíva zdroje (resources). Tieto zdroje môžu byť obrázky, videá, webové stránky alebo hocijaká iná informácia, ktorá môže byť reprezentovaná počítačom. Účelom týchto služieb je poskytnúť užívateľovi prístup k týmto zdrojom. Architektúra RESTu ponúka vývojárom jednoduchú implementáciu, údržbu či rozšíriteľnosť. Vo všeobecnosti by RESTové služby mali disponovať týmito vlastnosťami: [22]

- Representations (reprezentácia)
- Messages (správy)
- URIs
- Uniform interface (jednotné rozhranie)
- Stateless (bezstavovosť)
- Links between resources (odkazy medzi zdrojmi)
- Caching

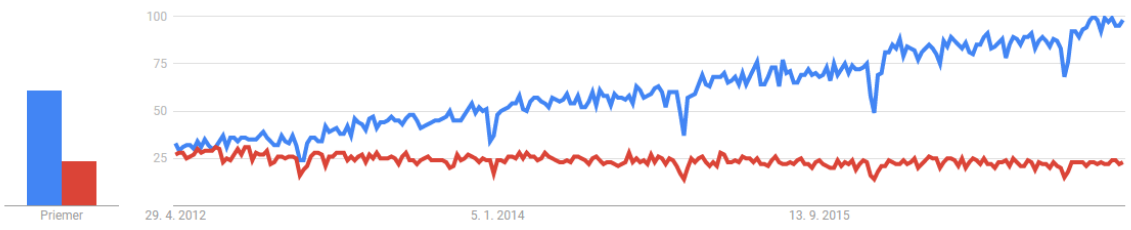
2.2.6 Reprezentácia dát

Hlavnou úlohou RESTových služieb je správa zdrojov a prideľovanie prístupu k týmto zdrojom. Takýto zdroj si môžeme predstaviť ako objekt pri OOP, tento zdroj/objekt môže obsahovať ďalšie zdroje. V takomto prípade je nutné najskôr identifikovať formát zdroju, s ktorým sa bude pracovať. Vzhľadom, na to, že REST nijako nelimituje formát pre prezentáciu zdrojov, je na vývojárovi aký si vyberie. Táto výhoda využitia viacerých formátov je vhodná vtedy, že takéto rozhranie vie komunikovať s rôznymi zariadeniami. [26, 19]

Napríklad, ak požiadavka na server obsahuje parametre pre odpoveď v určitom formáte a ak server disponuje takýmto formátovaním, dokáže jeho požiadavku spracovať. Odpoveď servera (response) by mala reprezentovať celý požadovaný obsah v čo najmenšom rozsahu. Pokiaľ to tak nie je možné, je nutné tieto zdroje rozdeliť na menšie a tým skrátiť dobu prenosu, a tak zrýchliť služby. [22]

Príklad zobrazenia JSON:

```
1. {
2. "ID": "1",
3. "Name": "Jozef",
4. "Email": "jozef@gmail.com",
5. "Country": "Slovakia"
6. }
```



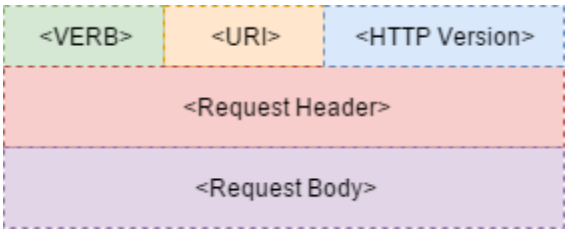
Graf 2.1 Vyhľadávané výrazy „xml api“ a „json api“

V grafe 2.1 bol použitý nástroj Google Trends pre porovnávanie vyhľadávaných výrazov pomocou Google vyhľadávania. Daný graf nám porovnáva hľadané výrazy pre „xml api“ a „json api“. Z tohto grafu je jednoznačne vidieť, že termín „xml api“ ako hľadaný výraz ale aj formát postupom času upadá, pričom formát JSON naberá na popularite.

2.2.7 Správy

HTTP Request

Klient a webová služba spolu komunikujú pomocou správ. Klient zašle požiadavku (request) na server a ten mu odpovie (resposne). Nehľadiac na aktuálne dáta, tieto správy obsahujú aj metadáta o danej správe. Tieto dáta požiadaviek a dáta odpovedí sú dôležité informácie pre navrhovanie RESTových služieb. [22]



Obrázok 2.7 Blokové zobrazenie HTTP požiadavky

<VERB> typ HTTP metódy napr. GET, PUT, POST, DELETE, OPTIONS atď.

<URI> Je URI adresa zdroja, na ktorej sa bude operácia vykonávať

<HTTP Version> je verzia HTTP, najčastejšie "HTTP v1.1"

<Request Header> Obsahuje metadáta tzv. kľúčové hodnoty (key-value). Tieto nastavenia obsahujú informáciu o správe a jej odosielateľovi, formáty, ktoré klient podporuje, formát tela správy, cache nastavenia odpovede a iné.

<Request Body> Je skutočný obsah správy.

Príklad POST požiadavky:

```
1.  POST http://MyService/Person/ HTTP/1.1
2.  Host: MyService
3.  Content-Type: text/xml; charset=utf-8
4.  Content-Length: 123
5.  <?xml version="1.0" encoding="utf-8"?>
6.  <Person>
7.    <ID>1</ID>
8.    <Name>Jozef</Name>
9.    <Email>jozef@gmail.com</Email>
10.   <Country>Slovakia</Country>
11. </Person>
```

Môžete vidieť príkaz POST, ktorý je nasledovaný adresou URI a verziou HTTP. Táto požiadavka obsahuje hlavičkové požiadavky. **HOST** je adresa serveru. **Content-Type** hovorí o type obsahu v tele správy, **Content-Length** je dĺžka dát v tele správy, tiež môže byť využitá na overenie či bola doručená celá správa. [22]

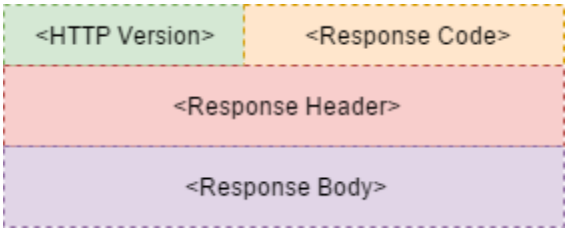
GET

V požiadavke GET sa nenachádza telo správy. **Accept** parameter v hlavičke odkazuje serveru možnosti formátovania, ktoré klient podporuje. Vzhľadom na tento parameter, ak server podporuje prezentáciu zdrojov v takomto formáte, tak požiadavke vyhovie. Inak môže vrátiť požiadavku spracovanú vo formáte servera alebo vrátiť chybovú hlášku. **User-Agent** obsahuje informácie o klientovi, ktorý vykonal požiadavku na server. **Accept-Encoding/Language** sú informácie o kódovaní a jazykovej podpore klienta. [22]

Príklad GET požiadavky:

```
1. GET http://www.w3.org/Protocols/rfc2616/rfc2616.html HTTP/1.1
2. Host: www.w3.org
3. Accept: text/html,application/xhtml+xml,application/xml; …
4. User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 …
5. Accept-Encoding: gzip,deflate,sdch
6. Accept-Language: en-US,en;q=0.8,hi;q=0.6
```

HTTP Response



Obrázok 2.8 Blokové zobrazenie HTTP odpovede

<Response Code> vracia server, ktorý obsahuje stav požiadavky t. j. stavový kód

<Response Header> obsahuje metadáta a nastavenia správy odpovede

<Response Body> obsahuje reprezentovaný zdroj ak bola požiadavka úspešná

Daná odpoveď požiadavky GET:

```
1. HTTP/1.1 200 OK
2. Date: Sat, 23 Aug 2014 18:31:04 GMT
3. Server: Apache/2
4. Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT
5. Accept-Ranges: bytes
6. Content-Length: 32859
7. Cache-Control: max-age=21600, must-revalidate
8. Expires: Sun, 24 Aug 2014 00:31:04 GMT
9. Content-Type: text/html; charset=iso-8859-1
10. <!DOCTYPE html PUBLIC "-
    //W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
    strict.dtd">
11. <html xmlns='http://www.w3.org/1999/xhtml'>
12. <head><title>Hypertext Transfer Protocol -- HTTP/1.1</title></head>
13. <body>
14. ...
```

Stavový kód **200 OK** znamená, že všetko prebehlo v poriadku a požadovaný obsah tela správy obsahuje platnú reprezentáciu zdroja, ktorá bola požadovaná. V tomto prípade bol požadovaný HTML dokument. [22]

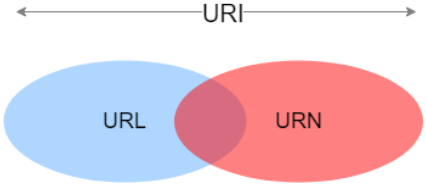
2.2.8 Adresovanie zdrojov

REST vyžaduje, aby bol každý zdroj adresovaný aspoň jednou URI. Pri adresovaní, sa snaží využívať čo najjednoduchšie odkazy, a to také , aby aj klient na základe tohto odkazu vedel pochopiť hierarchiu odkazov RESTovej služby. Adresa URI by mala byť jednoznačná cesta ku zdroju, nemala by prezrádzať operáciu, ktorá sa nad ňou bude vykonávať vzhľadom na to, že pri rozličnej HTTP požiadavke sa môže využívať rovnaká URI adresa. [22, 27]

2.2.9 URI

Jednotný identifikátor prostriedku (Uniform Resource Identifier) je najobecnejší z niekoľko príbuzných typov identifikátorov. URI môže popisovať zdroj z hľadiska identity (a neurčovať, kde je možné ho získať) a taktiež ako je zdroj možné nájsť (ale nepopisovať jeho identitu) a zároveň aj obe možnosti naraz.

URL na rozdiel od URI primárne popisuje ako sa ku zdroju dostať, naopak URN špecifikuje zdroj ako taký. URI tvorí akoby nad množinu týchto identifikátorov. [27, 28]



Obrázok 2.9 Zloženie adresy URI

Príklad ako by URI adresa nemala vyzerať:

```
1. http://MyService/FetcthPerson/1
```

alebo

```
1. http://MyService/DeletePerson?id=1
```

- URI by nemala obsahovať meno operácie, ktorá sa nad ňou bude vykonávať
- adresa by mala byť čo najvšeobecnejšia pre prehľadnosť či pre prípad, že by sa zdroj presunul
- nemala by obsahovať medzery namiesto toho by mali byť použité znaky ako
- _ (podtržník) alebo – (pomlčka)

Zhrnutie

REST naberá na význame spolu s formátom JSON. Pomaly sa stáva štandardom API webových služieb. Vzhľadom na rozšírenie techniky AJAX, ktorej REST vychádza v ústrety, pomáha jeho rozšíreniu tým, že sa zásadne nelíši od štandardného volania a získavania dát pomocou HTTP. REST sa svojou jednoduchosťou a rýchlosťou čoraz častejšie stáva podporovanými modernými frameworkami pre jeho jednoduchšiu implementáciu. Stáva sa štandardom webových služieb a môžeme vidieť, že aj spoločnosti ako Google, Twitter či Facebook využívajú služby RESTu.

3 Ciele práce

3.1 Návrh riešenia

Pri návrhu riešenia sme uvažovali nad vhodným výberom stavebných prvkov pre našu webovú aplikáciu a teda frameworkov, ktoré by uľahčili našu prácu pri vývoji. V našej webovej aplikácii by mal byť dodržaný princíp jednoduchosti a efektívnosti, mali by byť využité všeobecne známe štandardy pre jej použiteľnosť v praxi. Mala by byť orientovaná multiplatformovo, aby bola dostupná z čo najväčšieho počtu zariadení ale zároveň by mala byť bezpečná.

Použitie frameworkov pri vývoji tejto aplikácie bol hlavný bod tohto návrhu, frameworky obsahujú už integrované metódy pre prácu s databázou, formulármi či API rozhraním a tak by nám ponúkli overené funkčné riešenia.

3.2 Cieľ práce

Cieľom práce je vytvoriť webovú aplikáciu, ktorá bude vizualizovať dáta vo forme rôznych grafov alebo tabuliek. Aplikácia bude obsahovať hlavnú obrazovku na ktorej si bude užívateľ môcť navoliť aké namerané dáta sa budú zobrazovať a formu zobrazenia(typ grafu, tabuľka, mapa, atď.). Aplikácia bude mať RESTové rozhranie, cez ktoré bude prijímať nové dáta a taktiež aj poskytovať tieto dáta tretím stranám. Dáta budú ukladané do databázy. Autentifikácia jednej inštancie aplikácie bude založená na princípe API tokenov.

4 Metodika práce a metódy skúmania

4.1 Framework

Framework v preklade rámec alebo štruktúra, je softwarová štruktúra, ktorá pomáha programátorovi pri vývoji. Obsahuje už naprogramované funkcie, knižnice, API rozhrania a iné. Cieľom je ponúknuť vytvorenú štruktúru funkcií a metód, ktoré programátor bude ďalej len implementovať a tak nebude nútený tieto funkcie vytvárať od základu. Takáto možnosť značne urýchľuje prácu. Frameworky sú často aktualizované a preverované komunitou a teda jeho užívateľ si môže byť vedomý, že pracuje na bezpečnej a chybovo overenej platforme.

Na druhej strane zlým výberom sa práca na projekte môže značne skomplikovať. Aby s ním človek mohol pracovať, musí si najskôr naštudovať jeho dokumentáciu aby pochopil ako funguje, čo môže zabrať množstvo hodín ale pri budúcej práci na inom projekte znalosť frameworku prácu značne zefektívni a zrýchli.

4.1.1 Výber frameworku pre front-end

Front-end je opak back-endu. Stará sa ako o zobrazovanie dát z back-endu užívateľovi tak, aj o celkový design webstránky. Je to vlastne užívateľské rozhranie pomocou, ktorého sa ovláda logika back-endu.

Od daného frameworku bolo požadované aby bol schopný využívať najnovšiu verziu HTML 5 a pre zobrazovanie grafov podporu JavaScriptu. Vzhľadom na to, že chceme aby sa webová služba vedela prispôsobiť rôznym veľkostiam zobrazovacích zariadení, dbali sme na podporu responzívneho dizajnu.

Responzívny dizajn je spôsob štýlovania HTML kódu tak, aby sa zobrazovaná stránka dokázala prispôsobiť zariadeniu, na ktorej je stránka zobrazovaná. Stránka sa tak vie prispôsobiť displeju počítača či tabletu pri zobrazovaní rovnakého obsahu.

4.1.2 Bootstrap

Dá sa povedať, že tento framework bol vybraný už vopred. S Bootstrapom sme sa stretli už dávnejšie pri skúšaní tvorby HTML web stránok. Keď došlo na výber frameworku pre front-endovú časť po inom sme ani nepátrali. Bootstrap bol vytvorený vývojármi Twitteru a vydaný v roku 2011. Momentálne predstavuje jednotku vo front-endových frameworkoch, má skvelú podporu čo sa týka užívateľského fóra, návodov, video návodov či spracovanej dokumentácie. Pomocou neho môžeme behom pár minút vytvoriť užívateľskú navigáciu, formuláre, panely a mnoho ďalších komponentov web stránky.

Výhody:

- podpora responzívneho web dizajnu (táto možnosť sa môže vypnúť)
- prepracovaná dokumentácia
- aktuálnosť a pravidelné aktualizovanie a zlepšovanie

Nevýhody:

- pomerne veľká veľkosť balíčku vzhľadom na množstvo zahrnutých komponentov
- veľké množstvo HTML tried, ktoré môžu prinášať do návrhu neprehľadnosť

4.1.3 Podpora Bootstrapu

Podporované mobilné zariadenia: [29]

Tabuľka 4.1 Zobrazujúca podporu frameworku Bootstrap pre mobilné zariadenia

	Chrome	Firefox	Safari	Android prehliadač	Microsoft Edge
Android	podporované	podporované	N/A	Android v.5+	N/A
iOS	podporované	podporované	podporované	N/A	N/A
Windows 10 Mobile	N/A	N/A	N/A	N/A	podporované

Podporované desktopové prehliadače:

Tabuľka 4.2 Zobrazujúca podporu frameworku Bootstrap pre desktopové zariadenia

	Chrome	**Firefox**	**IE**	**Opera**	**Safari**
Mac	podporované	podporované	N/A	N/A	podporované
Windwos	podporované	podporované	IE10+	podporované	nepodporované

4.1.4 Inštalácia Bootstrapu

Inštalácia Bootstrapu je veľmi jednoduchá. Samotný návod na stránke frameworku ponúka viacero možností na inštaláciu. My si vysvetlíme dve.

Prvou možnosťou, ktorú sme využili aj my, je stiahnutie kompletného balíčku Bootstrapu, ktorý sa rozbalí a umiestni sa do zložky s projektom. Potom stačí vo vašej webstránke odkazovať na tento balíček a využívať tak funkcie Bootstrapu. Rozbalený balíček má potom nasledovnú štruktúru:

bootstrap/

├── css/

├── js/

└── fonts/

Druhá možnosť je inštalácia Bootstrapu pomocou Composeru. O Composeri si povieme neskôr, keď budeme rozoberať inštaláciu jednotlivých modulov PHP frameworku práve pomocou Composeru.

4.1.5 Použitie Bootstrapu

Použitie Bootstrapových knižníc sa nijako nelíši od napr. používania externých kaskádových štýlov CSS, JavaScriptu a pod.

Množstvo zahrnutých súborov pre kaskádové štýly, JavaScripty či iné komponenty závisí od užívateľských preferencií a teda základný balík Bootstrapu sa môže líšiť v závislosti od jeho požiadaviek.

- Načíta kaskádové štýly CSS Bootstrapu:

```
<link href="css/bootstrap.min.css" rel="stylesheet">
```

- jQuery nutné pre Bootstrapové moduly:

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js">
</script>
```

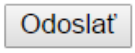
- Načíta všetky Bootstrapové moduly:

```
<script src="js/bootstrap.min.js"></script>
```

4.1.6 Praktický príklad

Práca s Bootstrapom je pomerne jednoduchá aj pre úplného začiatočníka, je intuitívna. Programátor vlastne nemusí tento framework vopred nejako hĺbkovo študovať, aby ho mohol používať. Ako bolo už spomenuté, Bootstrap disponuje vynikajúcou dokumentáciu, ktorá pri každom komponente obsahuje jeho stručný opis a následne praktickú ukážku použitia aj s kódom. Ak napr. pri vývoji programátor zistí, že potrebuje použiť tlačidlo pre odoslanie formulára, tak si v dokumentácii nájde sekciu Buttons (tlačidla), kde je daná problematika bližšie rozpísaná. Nižšie si uvedieme príklad na vytvorenie tlačidla klasickým spôsobom a pomocou Bootstrapu. V príklade budeme uvažovať len nad samotným kódom pre vytvorenie tlačidla. Dané zobrazenie tlačidla sa môže líšiť vzhľadom na použitý prehliadač.

- Základné zobrazenie tlačidla bez použitia kaskádových štýlov:



Obrázok 4.1 Základné tlačidlo vytvorené pomocou HTML

```
<button type="button">Odoslať</button>
```

- Základné zobrazenie tlačidla pomocou Bootstrapu:

Obrázok 4.2 Tlačidlo vytvorené pomocou frameworku Bootstrap

```
<button type="button" class="btn btn-success">Odoslať</button>
```

Takéto tlačidlo nielenže vyzerá krajšie, ale už svojou farbou napovedá, že vedie k operácii, ktorá by mala byť priaznivá/úspešná. Programový zápis samotného tlačidla sa od klasického zápisu líši len minimálne. Zápis **class** určuje vlastnosti tohto tlačidla využívajúceho triedu **btn** z knižnice Bootstrapu a druhý parameter **btn-success** hovorí, že pôjde o tlačidlo úspechu. Ako sme už spomínali používanie je naozaj intuitívne. Jednoduchou náhradou slova **success** za výraz default, waring či danger dokážu výrazne ovplyvniť vzhľad tlačidla. Pri takomto programovaní niekedy môže prevažovať znalosť anglického jazyka nad znalosťou frameworku. Programátor pri vývoji môže intuitívne skúsiť prepísať niektoré parametre komponentov. Ak napríklad používame navigáciu s parametrom **up** s vysokou pravdepodobnosťou, môže použiť aj parameter **down.** Nie je však pravidlom, že to tak funguje. V nasledujúcom príklade obsahuje každé tlačidlo názov parametra pomocou, ktorého bolo vytvorené napr. tlačidlo Danger obsahuje parameter **btn btn-danger.** [30]

Obrázok 4.3 Príklady vytvorených tlačidiel pomocou frameworku Bootstrap

4.1.7 Využitie šablóny

Tak isto ako nám framework ponúka možnosť tvorby jednotlivých komponentov do stránky, ponúka nám aj hotové riešenia vo forme templatov / šablón. Takáto šablóna zobrazuje vytvorenú stránku so všetkými elementmi, ktoré podporuje. Ak napríklad podporuje grafy, tak odkazuje na ukážkovú stránku, ktorá zobrazuje všetky možné zobrazenia grafov v danej šablóne. Programátor si tak vie predstaviť ako v danej šablóne graf bude vyzerať, podobne to vyzerá aj pri ostatných komponentoch.

Šablóny slúžia na zrýchlenie tvorby web stránky tým, že odbremeňuje programátora od programovania často používaných komponentov. Tieto šablóny sa ďalej implementujú metódami z back-endu.

V našom projekte sme sa rozhodli využiť voľne dostupnú šablónu zo stránky Bootstrapu s názvom **SB Admin.**

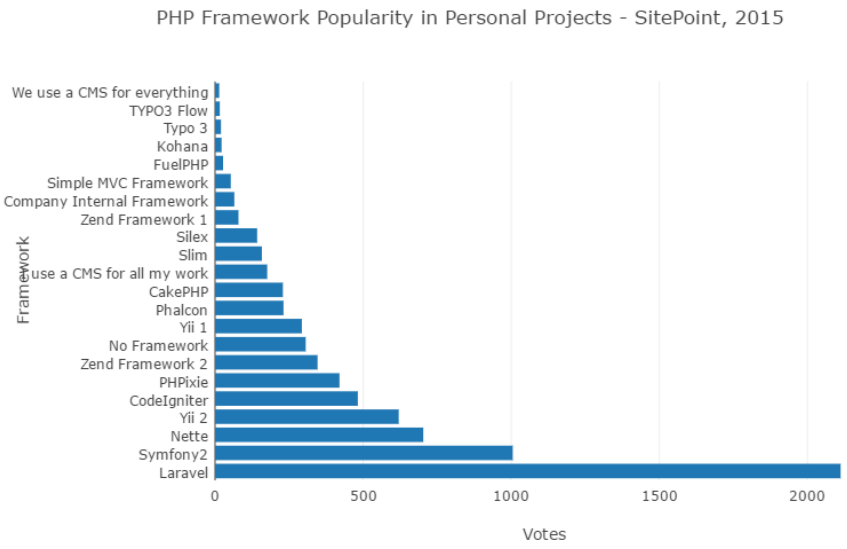
Spomenutá šablóna sa podobá spomínanému konceptu stránky Ubidots, je prehľadná a je orientovaná na administrátorskú tému. To znamená, že obsahuje prvky, pre správu web stránky, grafy či administrátorské rozloženie ikon, a teda sa dokonale hodí ako štartovací bod pre našu webovú aplikáciu. [30, 7]

4.1.8 Framework pre back-end

Back-end je časť aplikácie, ktorá sa stará o jej chod, výpočty a všetky dôležité úkony. Spracované údaje posiela front-endu, ktorý ich zobrazuje.

Keďže na začiatku práce na bakalárskej práci sme nemali žiadne skúsenosti z oblasti frameworkov, bolo nutné túto oblasť najprv naštudovať . Následne vybrať taký, ktorý bude najvhodnejší a bude mať dobré užívateľské referencie a podporu.

Už na začiatku sme sa rozhodli, že back-end teda časť webovej aplikácie, ktorá sa stará o správu databázy, programovú logiku a celkovú správu stránky bude vyvíjaná v programovom jazyku PHP. Tento jazyk sme si vybrali z toho dôvodu, že sme sa s ním stretli aj v škole na predmete Vývoj aplikácií pre internet a intranet. Vzhľadom na použitie tohto jazyku bolo nutné vybrať aj framework, ktorý daný jazyk podporuje. Internetový prieskum nás presvedčil, že medzi najosvedčenejšie frameworky patria Laravel, Symfony či Nette.



Graf 4.1 Popularita PHP frameworkov [31]

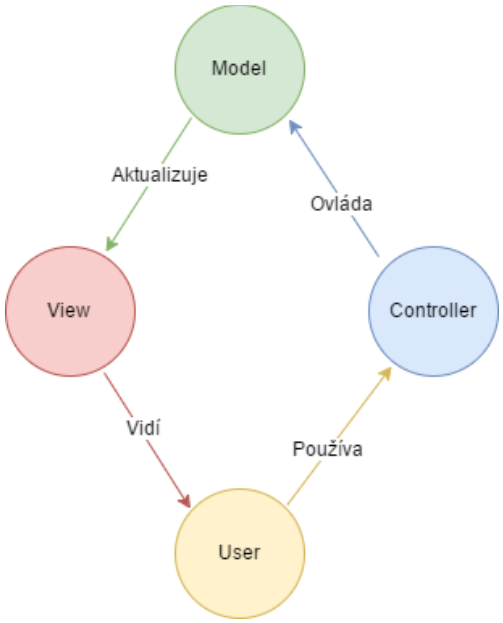
4.1.9 Nette

Framwerok Nette sme si na začiatku vybrali nielen kvôli informáciám, ktoré sme našli na internete, ale na základe referencií od skúsenejších programátorov či ľudí, ktorí sa tejto problematike venujú. Mali sme česť navštíviť dve firmy v našom okolí, ktoré sa zaoberajú tvorbou web stránok. Obe s ním pracovali takže, myšlienka jeho využitia v našej práci sa nám zdala byť na mieste.

Nette je stavaný tak, aby bol čo najpoužiteľnejší a ústretový a to naozaj je. O jeho významnosti hovoria jeho referencie. Konkrétne Nette si vybudovalo veľkú komunitu priaznivcov. Nasvedčuje tomu ich webové fórum, kde sa rieši mnoho problémov súvisiacich s týmto frameworkom. Využívajú ho popredné stránky a to napr. CSFD, Bandzone či ESET.

Nette ako aj iné frameworky pracuje na základe návrhového vzoru MVC(Model-View-Presenter). MVC je softwarová architektúra, ktorá rozdeľuje dátový model, užívateľské rozhranie a riadiacu logiku do troch nezávislých tried. [31, 32]

- **Model** – je vrstva pracujúca s dátami, je oddelená od aplikácie a komunikuje iba s Presenterom
- **View (Pohľad)** – je front-endová vrstva reprezentovaná modelom, ktorá zobrazuje dáta užívateľovi
- **Controller (Presneter)** – je prepojovacia vrstva Modelu a View, spracováva požiadavky, požaduje dáta od Modelu a vracia ich do View.



Obrázok 4.4 Popis MVC modelu

4.1.10 Používanie Nette

Práca s Nette od začiatku vyzerá veľmi jednoducho. Dostupný návod na vytvorenie prvej aplikácie sa nachádza na internetovej stránke tohto frameworku. Je jednoduchý a prehľadný. Po niekoľkých prípravných krokoch, prechádza návod do budovania prvej mini aplikácie. Pri tomto vytváraní prvej aplikácie nám veľmi pomáhala knižnica Tracy. Je to ladiaci (debugovací) nástroj pre Nette, nakoľko v jazyku PHP sa ťažko odhaľujú chyby. Naša prvá aplikácia sa podarila úspešne vytvoriť, ale s ďalším vývojom na tejto platforme začali prichádzať aj prvé problémy.

Ako už bolo spomenuté s jazykom PHP sme nemali väčšie skúsenosti. Našu znalosť by sme mohli ohodnotiť statusom začiatočník, dokonca laik. Nette v tomto štádiu začínalo byť pre nás viac prekážkou ako služobníkom, ktorý mal našu prácu uľahčovať. Pri širšom vyhľadávaní sme postupne narazili na návody ako vytvárať užívateľskú registráciu, ako pracovať s databázou a ako implementovať RESTové rozhranie pomocou dostupných balíčkov.

Na jednej strane sme mali teda výbornú platformu, na ktorej sme mohli rozvíjať našu webovú službu. Na druhej strane sme nemali dostatok znalostí na to, aby sme dokázali tento potenciál využiť v náš prospech.

Dospeli sme k záveru, že je treba hľadať framework, ktorý by pre nás predstavoval lepšie útočisko, niečo „ľahšie“. Pri výbere sme sa zamerali na jednoduchší, s menej funkciami, ale s dobrou užívateľskou podporou a dokumentáciou .

4.2 Slim

4.2.1 Úvod do Slimu

Počas hľadania nového frameworku boli najčastejšie sa objavujúce kľúčové slová vo vyhľadávači easy, framework, rest, restful, api. Výsledkom tejto kombinácie bol framework Slim.

Slim už na úvodnej stránke v svojom logu prezrádza svoj koncept „Slim a micro framework for PHP“ a teda Slim mikro framework pre PHP. Slim je opísaný ako ideálny nástroj pre vytváranie API, ktoré konzumuje, opakovane využíva a publikuje dáta. Podporuje rýchle vytváranie prototypov. Odkazuje sa na svoju jednoduchosť voči konkurenčným frameworkom ako Symfony alebo Laravel. Užívateľa láka na svoj jednoduchý kód, ktorý ako opisujú vývojári je možné zvládnuť za jedno odpoludnie.

Namiesto balíka plných funkcií prichádza Slim s implementáciou základných HTTP metód s tým, že necháva užívateľa aby si do tohto frameworku implementoval jemu potrebné komponenty. [33, 32]

4.2.2 XAMPP

Je nástroj, ktorý nám umožňuje vytvoriť Apache server na počítači bez nutnosti zakúpenia nejakej webovej služby. Je nevyhnutným prvkom pre spustenie PHP kódu či databáz MySQL.

4.2.3 Composer

Composer sme síce už spomenuli, ale jeho význam a funkcie v tomto projekte sme si ešte nepopísali. Composer je tzv. dependenci manager čo v preklade znamená manažér závislostí, ale tento termín sa v praxi nepoužíva. Pomocou tohto nástroja je možné sťahovať a inštalovať balíčky do projektov s tým, že sa tieto balíky inštalujú lokálne a nie globálne , takže v rôznych projektoch môžete používať rovnaké balíky rôznych verzií. Composer sa tak stará o jednoduchú inštaláciu a manažment balíkov.

Pre inštaláciu Slimu pomocou Composera stačí zadať nasledujúci kód (v závislosti od verzie):

```
1. composer require slim/slim "^2.*"
```

4.3 Výsledky práce

4.3.1 Nastavenie prostredia

Na začiatku bolo nutné zhromaždiť a nainštalovať potrebné balíčky / moduly, ktoré neskôr v práci budeme využívať. Tieto balíčky sme do projektu nainštalovali pomocou spomenutého Composeru. Vzhľadom na použitú verziu Slimu č. 2 bolo nutné zvoliť staršie verzie balíčkov pre zaručenie ich kompatibility. [33]

Obsah súboru composer.json :

```
{
  "autoload":{
    "psr-4":{
      "Options\\": "app/Options"
    }
  },
  "require": {
    "slim/slim": "2.*",
    "ircmaxell/random-lib": "^1.2",
    "twig/twig": "~1.0",
    "alexgarrett/violin": "^2.2",
    "illuminate/database": "^5.4",
    "hassankhan/config": "^0.10.0",
    "phpmailer/phpmailer": "^5.2",
    "slim/views": "^0.1.3"
  }
}
```

Pri tvorbe našej webovej služby sme začali najskôr pracovať so Slimom verzia č. 3. Táto verzia sa podstatne líši od verzie č. 2 štruktúrou, obsiahlosťou, prehľadnosťou. V našej práci sme sa rozhodli používať staršiu verziu vzhľadom na rozsiahlejšiu podporu balíkov a užívateľských návodov pri riešení problémov.

4.3.2 Návrh databázy

Pri návrhu a správe databázy sme využívali nástroj phpMyAdmin pomocou jazyka SQL, ktorý bol dostupný z prostredia XAMPP. Pred začatím programovania samotnej webovej služby bolo nutné najskôr navrhnúť databázu, do ktorej by sa ukladali informácie o užívateľoch, zariadeniach užívateľov a informácie o hodnotách, ktoré tieto zariadenia obsahujú.

4.3.3 Databáza

Naša databáza obsahuje 4 tabuľky:

- tabuľka **user** predstavuje užívateľa

Tabuľka 4.3 Popisuje jednotlivé polia databázovej tabuľky **user**

user	Popis
id	Automatické pridelenie ID každému užívateľovi
name	Prihlasovacie meno užívateľa
email	Prihlasovací email užívateľa
password	Heslo užívateľa
created_at	Dátum a čas vytvorenia účtu
updated_at	Dátum a čas úpravy účtu
api_key	Pridelený API kľúč

Užívateľovi je pri vytvorení účtu / registrácii pridelené ID, ktoré sa spája s jeho zariadeniami. Do účtu sa môže prihlásiť voľbou prihlasovacieho mena alebo emailu a hesla. Automaticky sa mu pridelí čas vytvorenia, čas úpravy a API kľúč.

- tabuľka **device** predstavuje zariadenie

Tabuľka 4.4 Popisuje jednotlivé polia databázovej tabuľky device

device	Popis
id_device	Automatické pridelenie ID každému zariadeniu
user_id	ID vlastníka zariadenia
id_type	ID typu zariadenia
device_name	Meno zariadenia
created_at	Dátum a čas vytvorenia zariadenia
updated_at	Dátum a čas úpravy zariadenia

Zariadenie vzniká jeho vytvorením cez užívateľské rozhranie vo webovej službe. Užívateľ môže vytvoriť viacero zariadení ale jedno zariadenie nemôže mať viacero vlastníkov. Každému zariadeniu je pridelený jeho typ, ID, ID vlastníka a meno zariadenia. Čas vytvorenia a úpravy sa mu automaticky pridelí.

- tabuľka **type** predstavuje typ zariadenia

Tabuľka 4.5 Popisuje jednotlivé polia databázovej tabuľky type

type	**Popis**
id_type	Automatické pridelenie ID každému typu
device_name	Pomenovanie typu
unit	Jednotka zobrazujúca sa pri hodnotách

Každé zariadenie musí byť nejakého typu. A teda ak užívateľ napr. pridáva zariadenie teplomer, z tabuľky typov zvolí typ „thermometer“. Typ obsahuje aj pole **unit**, ktoré obsahuje jednotku veličiny.

- tabuľka **device_value** predstavuje hodnoty zariadenia

Tabuľka 4.6 Popisuje jednotlivé polia databázovej tabuľky device_value

device_value	**Popis**
id	Automatické pridelenie ID každému typu
id_device	Pomenovanie typu
created_at	Dátum a čas vytvorenia hodnoty
updated_at	Dátum a čas úpravy hodnoty
device_val	Zaznamenaná hodnota

Každé zariadenie obsahuje hodnoty, ktoré zaznamená a posiela ich webovej aplikácii. Opäť je jednotlivej hodnote priradený dátum, čas vytvorenia a jej úpravy, ako aj ID. Jednotlivé hodnoty obsahujú samotnú hodnotu a aj ID zariadenia, ktorému patria.

Pri práci s databázou sme využívali balíček **illuminate/database**, ktorý natívne využíva ako databázovú vrstvu framework Laravel. Pri získavaní údajov z databázy nám prácu uľahčoval query builder, pomocou ktorého sa práca s databázou stala veľmi jednoduchá a intuitívna, no používali sme aj klasický zápis SQL. Nevýhodou tohto balíčku bola nutnosť dodržiavania návrhu jeho modelu čo v našom prípade znamenalo, že tabuľky, ktoré aktívne využívali tento balíček museli implementovať tento model a tak zahŕňať aj polia ako updated_at a created_at či dodržiavať názvy polí napr. id v tabuľke user, ktoré bolo pôvodne navrhnuté ako user_id. [34]

Príklad použitia query builderu pre získanie maximálnej hodnoty z vybraného zariadenia **:**

```
1. Capsule::table('device_value')->where('id_device','=',$id)->max('device_val');
```

4.3.4 Pohľady

Pohľad (view) je dotaz uložený v databáze a vytvára akúsi virtuálnu dynamickú tabuľku. Môže obsahovať dáta z jednej alebo viac tabuliek.

Výhody prečo sme ho použili:

- Môže slúžiť ako iné pomenovania pre tabuľky
- môže z tabuľky vybrať iba určité riadky alebo stĺpce
- zaberá veľmi málo miesta, databáza obsahuje iba definíciu a nie kópiu dát
- zabezpečuje vyššiu bezpečnosť
- môže spájať viacero tabuliek alebo viacero ďalších pohľadov
- môže obsahovať rôzne výpočty naprieč viacerých tabuliek

View **w_dasboard** nám umožnil načítať všetky potrebné údaje z databázy v jedom selecte. Vďaka tomu stúpla rýchlosť odozvy aplikácie pre používateľa, prehľadnosť kódu aplikácie a možnosť jednoduchých zmien. Načítavanie údajov do MySql View je vykonávané priamo v databáze.

V tabuľke **dashboard** (viď. Príloha B) sa ukladajú informácie aký užívateľ má aký typ dashboardu (widget) pre aké zariadenie. Pomocou view **w_dasboard** sme k nej pripojili doplňujúce informácie (viď. Príloha B) o nastaveniach daného widgetu a jednotku meraných hodnôt. Použitie daného pohľadu nám uľahčilo načítavania štyroch samostatných selectov v jedenom.

Dashboard (viď. Príloha D.3) v aplikácii slúži ako pracovná plocha užívateľa. Môže si doň pridávať miniaplikácie, reprezentujú ich ikony obsahujúce hodnoty z vybraných zariadení. Typy miniaplikácií sú vopred definované webovou aplikáciou, sú bližšie opísané v užívateľskej príručke.

4.3.5 Autentifikácia a validácia

Vedieť rozlíšiť užívateľa a jeho bezpečnosť je v našej aplikácii nevyhnutné. Aby nedošlo k neoprávnenému prístupu k dátam iných užívateľov, bolo nutné vytvoriť užívateľskú autentifikáciu a validáciu pre korektné vytváranie užívateľov, zariadení a ostatných údajov. Autentifikácia užívateľa prebieha viacerými spôsobmi, ktoré zaručujú, že webová aplikácia pracuje práve s údajmi užívateľa a že iba užívateľ má právo k týmto dátam pristupovať. [33]

API kľúč / token

Každému užívateľovi, ktorý si vytvorí účet pri registrácii je pridelený API kľúč. Tento API kľúč resp. API token je ďalej využívaný pri práci s naším RESTovým API rozhraním. Slúži ako parameter hlavičkového súboru pre autorizáciu užívateľa pred vykonaním určitej operácie cez API rozhranie. Je vyžadovaný v závislosti od požadovanej operácie. Generovanie kľúču prebieha pomocou kryptovacieho algoritmu MD5 a vyzerá nasledovne: [22]

Hashovanie:

```
1.  'api_key' => md5(uniqid(rand()))
```

Vygenerovaný kľúč:

```
2.  051c97bd4289a9eef0b4f39f2a1231af
```

Prihlasovanie

Užívateľ svoju identitu preukáže webovej aplikácii pomocou prihlásenia zadaním mena / emailu a hesla, ktoré sa overia s databázou. Pri úspešnom prihlásení sa priradí užívateľská session, na základe ktorej sa ďalej aplikácia riadi pri zobrazovaní užívateľských dát. Session je uložená na serveri, užívateľ k nej tak nemá prístup a nemôže ju modifikovať.

Validácia údajov:

```
1.  $v->validate([
2.          'identifier' => [$identifier, 'required'],
3.          'password' => [$password, 'required']
4.      ]);
```

Priradenie session:

```
1.  $_SESSION[$app->config->get('auth.session')] = $user->id;
2.
```

Prístup

Keď naša aplikácia vedela rozoznať autentifikovaného a neautentifikovaného užívateľa, na základe tejto skutočnosti vedela povoľovať prístup k určitým funkciám stránky. Nezaregistrovaný užívateľ by tak nemal mať prístup k stránke zobrazujúcej zariadenia, užívateľský profil či možnosti zmeny hesla. Naopak zobrazovať stránku registrácie či prihlásenia pre už autentifikovaného užívateľa by bolo zbytočné. Tento problém sme vyriešili funkciou **`authenticationCheck`**, ktorá tento problém rieši.

```
1.  $app->get('/devices/:id', $authenticated(), function ($id) use($app)
```

Pri požiadavke GET pre získanie stránky zobrazujúcej zariadenia parameter **`$authenticated()`** zavolá funkciu **`authenticationCheck`**, ktorá kontroluje autentifikáciu používateľa a na základe toho povoľuje prístup.

Validácia údajov

Pre validáciu v našej aplikácii sme použili balíček **violin** aj zabudované funkcie pre validáciu pomocou HTML 5. Implementácia bola jednoduchá a ponúkala širokú škálu preddefinovaných pravidiel pre validáciu. Jednotlivé pravidlá sa od seba oddeľujú symbolom „ | “.

```
$v->validate([
        'email' => [$email, 'required|email|uniqueEmail'],
        'name' => [$name, 'required|alnumDash|max(20)|uniqueUsername'],
        'password' => [$password, 'required|min(6)'],
        'password_confirm' => [$passwordConfirm, 'required|matches(password)'],
    ]);
```

Tabuľka 4.7 Popisuje vybrané validačné pravidlá

Názov	Popis
required	Vyžaduje zadanie hodnoty
min(6)	Zadaná hodnota musí mať minimálne 6 znakov
matches(password)	Kontroluje či sa zadaný reťazec zhoduje s parametrom

Použitie balíčka **violin** značne urýchlilo kontrolu vstupných údajov svojim jednoduchým použitím a možnosťou vytvárania vlastných pravidiel. Vybrané pravidlá pre kontrolu, ktoré boli použité v našej aplikácii je opísaná v tabuľke vyššie.

4.3.6 Zobrazovanie stránok

Zobrazovanie stránok v našej aplikácii prebieha dynamicky. Pomocou šablóny a fameworku Bootstrap bol vytvorený základný vzhľad stránky, ktorej jednotlivé časti boli implementované funkciami našej aplikácie. Na to, aby bolo zobrazovanie týchto stránok v prostredí frameworku Slim možné, bolo nutné použiť balíčky **Twig-View** a **Slim-Views**, ktoré spolu navzájom spolupracujú. Tieto balíky umožňujú prácu s PHP v našich šablónach s minimálnym kódom. Pôvodný PHP kód zjednodušujú vlastným zápisom. Samotný kód sa vkladá do tzv. blokov.

```
{% block content %}

...  obsah ...

{% endblock %}
```

Základ zobrazovania stránok v našej aplikácii tvorí šablóna `default.php`. Táto šablóna zobrazuje bočný panel pre odkazy ako aj navigačný panel s užívateľskou ponukou. Načíta všetky potrebné komponenty Bootstrapu a definuje **block content** a teda priestor, kde sa bude zobrazovať obsah ostatných stránok. Tento obsah taktiež podlieha autorizácii, určité odkazy sú tak zobrazované len neautentifikovaným alebo len autentifikovaným užívateľom.

```
1. {% if auth %}
2.
3. ... obsah ...
4.
5. {% endif %}
```

View a Presenter

Pre zobrazenie stránky potrebujeme vytvoriť jej šablónu (view) a back-end, ktorý ju bude definovať a bude sa starať o údaje, ktoré sa do nej pošlú (presenter). Štruktúra našej stránky obsahuje zložku **views**, ktorá obsahuje naformátované šablóny a zložku **routes**. Tá obsahuje presentery, ktoré tieto šablóny spravujú.

View pre jednotlivé stránky vyzerá nasledovne. Dedí prostredie základnej šablóny. Tak získava navigačné menu postranné linky, formátovanie, kaskádové štýly a všetko čo táto šablóna definovala a dopĺňa ju o svoj obsah. [32]

```
1. {% extends 'templates/default.php' %}
2.
3. {% block tittle %} ... nadpis ... {% endblock %}
4.
5. {% block content %}
6.
7. ... obsah ...
8.
9. {% endblock %}
```

Presentery šablóny definujú s akým obsahom bude šablóna pracovať a ako bude reagovať na požiadavky GET a POST.

```
1.  $app-
    >get('/devices_edit/:dev_name', $authenticated(), function ($dev_name) use($app)
2.  {
3.      $session = $_SESSION[$app->auth->user];
4.
5.          $device = Capsule::select('SELECT * FROM device WHERE device_name = "'.$d
    ev_name.'"');
6.          $types = Capsule::table('type')->get();
7.          var_dump($device);
8.          $app->render('user/devices_edit.php', [
9.
10.             'session' => $session,
11.             'dev' => $device,
12.             'types' => $types,
13.         ]);
14.
15. })->name('edit');
```

V danej ukážke skráteného kódu vidíme, čo sa udeje pri požiadavke o získanie tejto stránky. V prvom riadku definujeme adresu, na ktorej bude šablóna pre editovanie zariadení dostupná. Bude vytvorená za základe parametru `dev_name`. Po vykonaní ostatných operácií sa premenné, s ktorými budeme v šablóne pracovať vkladajú ako pole do funkcie render. V šablóne bude potom premenná `$device` dostupná pod názvom `dev`. Danej ceste môžeme priradiť meno ako v riadku č.15 **edit**.

Z ostatných stránok aplikácie na ňu potom môžeme odkazovať nasledujúcim spôsobom:

```
1.  <a href="{{ urlFor('edit') }}">Edit</a>
```

5 Záver

Cieľ práce sa nám podarilo splniť. Výsledkom našej práce je webová aplikácia, ktorá dokáže vizualizovať namerané dáta, ktoré prijíma z RESTového rozhrania. Dáta sú zobrazované pomocou tabuliek, grafov a miniaplikácií. Webová aplikácia ponúka jednoduché a prehľadné užívateľské rozhranie. Umožňuje užívateľovi správu zariadení, vytváranie užívateľského dashboardu a prehľad jednotlivých zariadení. Užívateľ po registrácií obdrží API kľúč, ktorým sa overuje pri určitých požiadavkách s API. Môže ho využiť na pridanie nového zariadenia alebo na získanie informácií o zariadeniach a jeho hodnotách. Pre zjednodušenie práce s API rozhraním a webovou aplikáciou sú k nim na webovej stránke priložené spracované manuály.

Prínosom našej práce je webová aplikácia, ktorú je možné používať na nenáročnú vizualizáciu dát. Tieto dáta môžu byť hodnoty nameraných teplôt, ktoré sú reprezentované formou grafov a tabuliek. Je vhodná napr. na prepojenie s mini počítačom Raspbery Pi obsahujúcim senzory, ktorých namerané hodnoty môžu byť vizualizované. Všetky použité prvky webovej aplikácie sú voľne dostupné, je možné ich voľne upravovať a rozširovať funkcie aplikácie. Tá bola vytvorená s čo najväčšiu jednoduchosťou a prehľadnosťou. Ostáva tu otvorený priestor pre úpravu aplikácie na ľahšie pridávanie nových modulov, pridanie podpory pre zobrazovanie live videa, máp alebo zasielanie notifikácií.

Jednou z podmienok splnenia cieľa bolo zvládnuť programovacie jazyky PHP, MySQL a HTML 5. Potom prišla na rad práca s frameworkami a ich dokumentáciami. Ďalším krokom bolo samotné programovanie riešenia zadania, ktoré bolo sprevádzané postupným zapisovaním jednotlivých krokov, čím bola zavŕšená naša výsledná práca.

Zoznam použitej literatúry

[1] Amazon, „Amazon Dash Button,“ 3 Marec 2017. [Online]. Available: https://www.amazon.com/Dash-Buttons/b?ie=UTF8&node=10667898011.

[2] alza, „Inteligentné osvetlenie Philips Hue,“ 3 Marec 2017. [Online]. Available: https://www.alza.sk/inteligentne-osvetlenie-philips-hue/18860240.htm.

[3] Apple, „HomeKit,“ 3 Marec 2017. [Online]. Available: https://developer.apple.com/homekit/.

[4] ATP Journal, „ATP Journal,“ 3 Marec 2017. [Online]. Available: http://www.atpjournal.sk/podujatia/iot-expo-bratislava-2016-internet-of-things.html?page_id=23477.

[5] Amazon, „How the AWS IoT Platform Works,“ 5 Marec 2017. [Online]. Available: https://aws.amazon.com/iot-platform/how-it-works/.

[6] Telekom, „MAGENTA SMARTHOME,“ 5 Marec 2017. [Online]. Available: https://www.telekom.sk/smarthome.

[7] ubidots, „Features,“ 5 Marec 2017. [Online]. Available: https://ubidots.com/features.

[8] slovanet, „Čo je "internet vecí",“ [Citácia], 6 3 2017. [Online]. Available: https://www.slovanet.net/sk/internet/iot/.

[9] slovanet, „Čo "internet vecí" prinesie?,“ [Citácia], 6 Marec 2017. [Online]. Available: https://www.slovanet.net/sk/internet/iot/.

[10] S. KASHYAP, „10 Real World Applications of Internet of Things (IoT) – Explained in Videos,“ 6 Marec 2017. [Online]. Available: https://www.analyticsvidhya.com/blog/2016/08/10-youtube-videos-explaining-the-real-world-applications-of-internet-of-things-iot/. [Cit. 26 August 2016].

[11] L. Columbus, „Roundup Of Internet Of Things Forecasts And Market Estimates, 2016,“ 7 Marec 2017. [Online]. Available: https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-

things-forecasts-and-market-estimates-2016/#29d3cc15292d. [Cit. 27 November 2016].

[12] BRANDON GAILLE, „16 Pros and Cons of the Internet of Things," 7 Marec 2017. [Online]. Available: http://brandongaille.com/16-pros-and-cons-of-the-internet-of-things/. [Cit. 23 August 2016].

[13] D. Karandikar, „Pros and Cons of Internet of Things (IoT) - What You Need to Know," 10 Marec 2017. [Online]. Available: http://www.buzzle.com/articles/pros-and-cons-of-internet-of-things-iot.html. [Cit. 14 August 2016].

[14] B. Schmarzo, „Difference between Big Data and Internet of Things," 15 Marec 2017. [Online]. Available: https://infocus.emc.com/william_schmarzo/difference-big-data-iot/. [Cit. 27 Február 2017].

[15] M. Abu-Elkheir, „Data Management for the Internet of Things: Design Primitives and Solution," 16 Marec 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3871070/. [Cit. 13 November 2013].

[16] AIG, „The Internet of Things: Benefits and Risks," 15 Marec 2017. [Online]. Available: https://www.aig.com/knowledge-and-insights/the-rise-ramifications-and-risks-of-the-internet-of-things. [Cit. 8 1 2016].

[17] T. Jaffey, „MQTT and CoAP, IoT Protocols," 10 Apríl 2017. [Online]. Available: https://eclipse.org/community/eclipse_newsletter/2014/february/article2.php.

[18] J. Berger, „Internet věcí a protokol MQTT," 10 Apríl 2017. [Online]. Available: http://vyvoj.hw.cz/internet-veci-a-protokol-mqtt.html. [Cit. 8 Február 2017].

[19] ROOT.CZ, „Protokol MQTT: komunikační standard pro IoT," 10 Apríl 2017. [Online]. Available: https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/. [Cit. 29 Jún 2016].

[20] A. Vojáček, „Základní úvod do oblasti internetu věcí (IoT)," 11 Apríl 2017. [Online]. Available: http://automatizace.hw.cz/zakladni-uvod-do-oblasti-internetu-veci-iot.html. [Cit. 16 September 2016].

[21] Wikipedia, „XMPP," 11 Apríl 2017. [Online]. Available:

https://en.wikipedia.org/wiki/XMPP.

[22] D. Hanák, „Stopařův průvodce REST API,“ 18 Marec 2017. [Online]. Available: https://www.itnetwork.cz/nezarazene/stoparuv-pruvodce-rest-api.

[23] R. Tamada, „How to create REST API for Android app using PHP, Slim and MySQL,“ 5 Apríl 2017. [Online]. Available: http://www.androidhive.info/2014/01/how-to-create-rest-api-for-android-app-using-php-slim-and-mysql-day-12-2/.

[24] Wikipédia, „Webová služba,“ 12 Apríl 2017. [Online]. Available: https://sk.wikipedia.org/wiki/Webov%C3%A1_slu%C5%BEba. [Cit. [Citácia]].

[25] M. Vaqqas, „RESTful Web Services: A Tutorial,“ 19 Marec 2017. [Online]. Available: http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069?pgno=1. [Cit. 23 September 2014].

[26] M. Malý, „REST: architektura pro webové API,“ 22 Marec 2017. [Online]. Available: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/. [Cit. 3 August 2009].

[27] Wikipédia, „Uniform Resource Identifier,“ 8 Apríl 2017. [Online]. Available: https://en.wikipedia.org/wiki/Uniform_Resource_Identifier.

[28] Myartve, „Jaký je rozdíl mezi URL a URI?,“ 10 Apríl 2017. [Online]. Available: http://www.myartve.net/jaky-je-rozdil-mezi-url-a-uri/.

[29] Bootstrap, „Supported browsers,“ 11 Apríl 2017. [Online]. Available: http://getbootstrap.com/getting-started/#support-browsers.

[30] Bootstrap, „Components,“ 11 Apríl 2017. [Online]. Available: http://getbootstrap.com/components/.

[31] Nette, „Seznámení s Nette Frameworkem,“ 12 Apríl 2017. [Online]. Available: https://doc.nette.org/cs/2.4/getting-started.

[32] Wikipédia, „Model–view–controller,“ 12 Apríl 2017. [Online]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

[33] Slim, „Slim Framework v2,“ 15 Apríl 2017. [Online]. Available: http://docs.slimframework.com/.

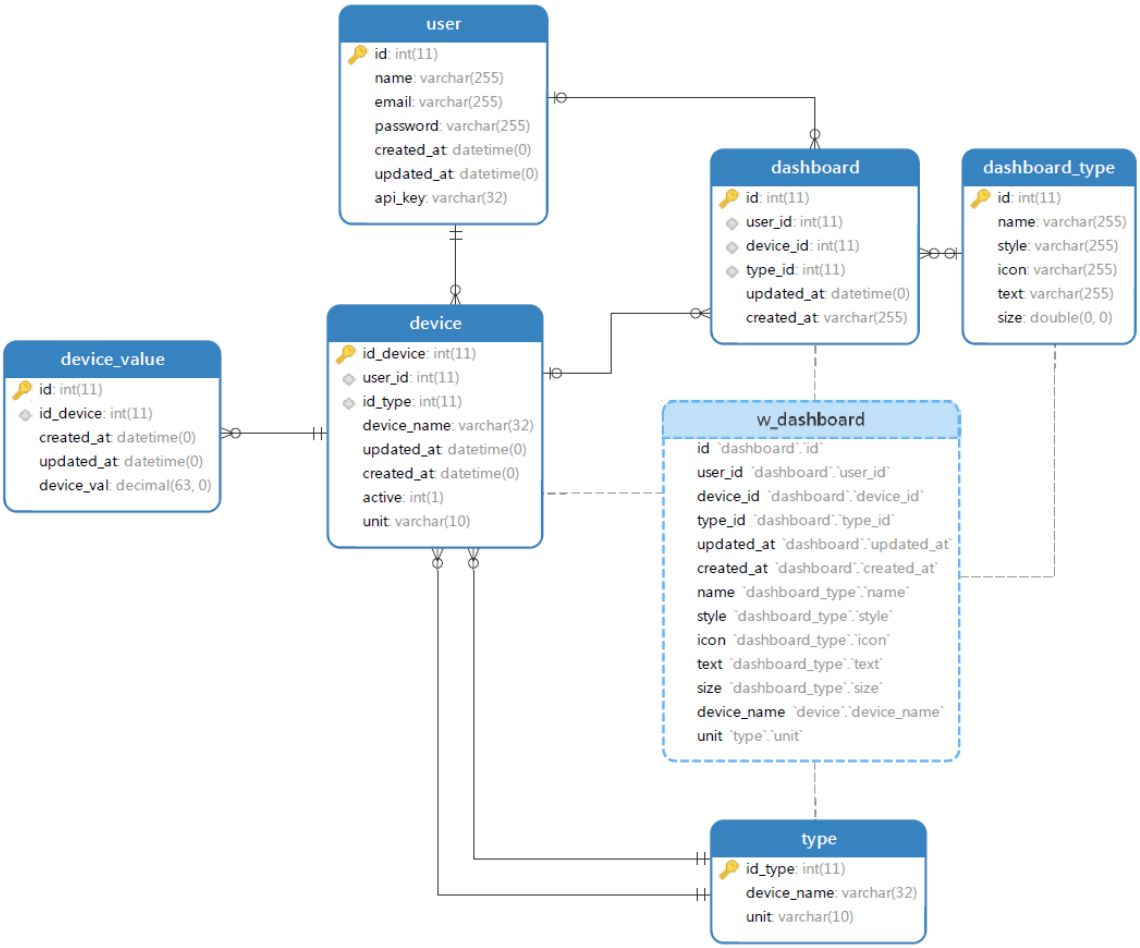
[34] „illuminate/database,“ 15 Apríl 2017. [Online]. Available: https://github.com/illuminate/database.

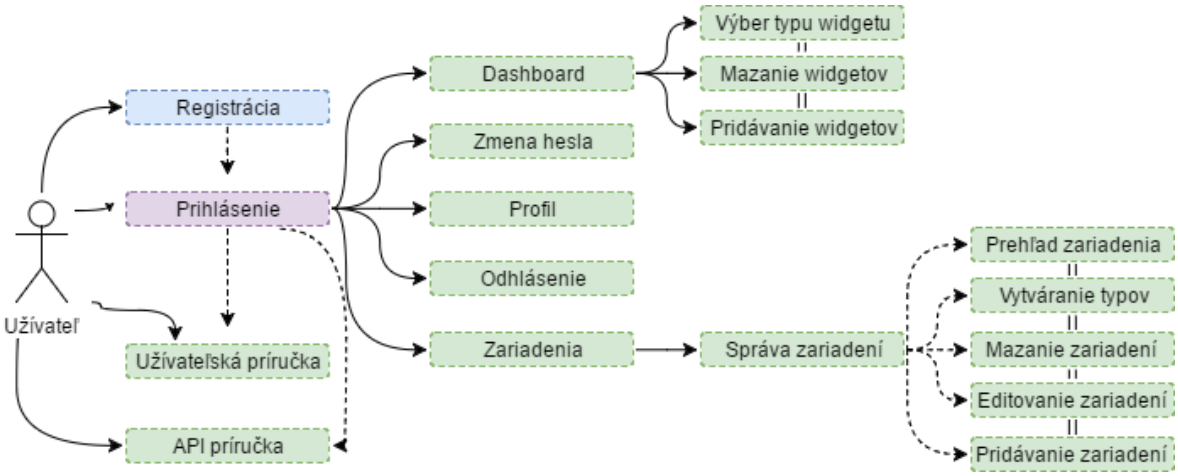
Referencie na obrázky

Prílohy

Príloha A: Architektúra webovej aplikácie

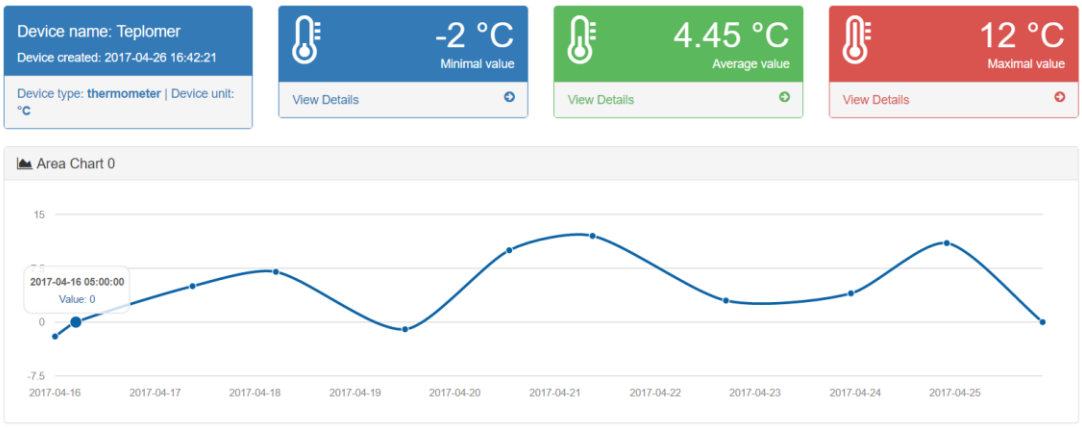
Príloha B: Relačný model databázy



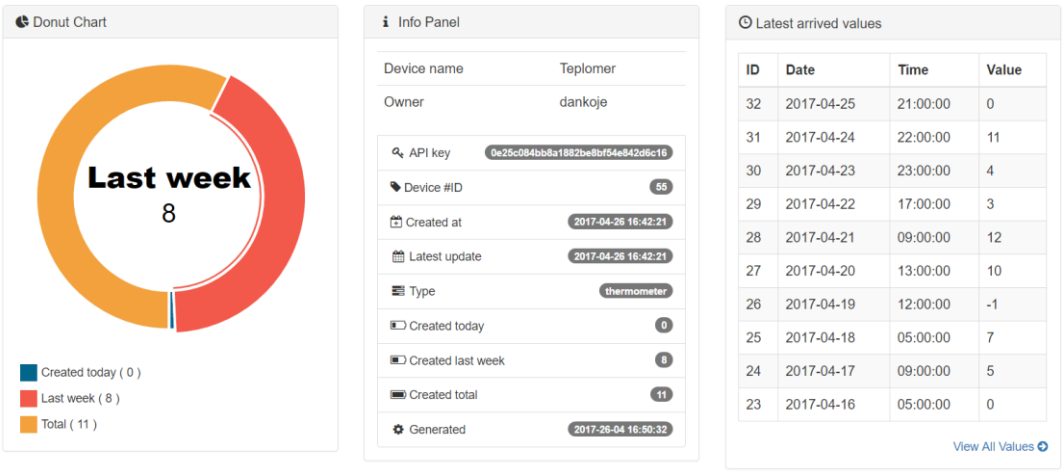


Príloha C: Diagram prípadov použitia

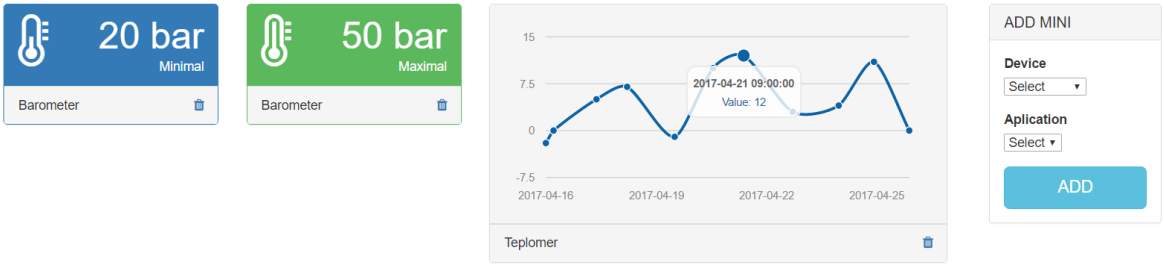
Príloha D: Náhľady stránky



Príloha D.1 Stránka zariadenia - zobrazenie hodnôt v grafe



Príloha D.2 Stránka zariadenia - zobrazenie hodnôt v koláčovom grafe a tabuľkách



Príloha D.3 Dashboard s pridanými miniaplikáciami z vybraných senzorov

Príloha E: Obsah CD

Priložené CD obsahuje:

- Práca v elektronickej podobe (formát PDF)
- Práca v elektronickej podobe (formát WORD)
- Praktickú časť
- Inštalačná príručka (formát PDF)