SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-16605-115084

WEB APLIKÁCIA NA SPRÁVU OPC UA SERVEROV

BAKALÁRSKA PRÁCA

2024 **Peter Likavec**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-16605-115084

WEB APLIKÁCIA NA SPRÁVU OPC UA SERVEROV

BAKALÁRSKA PRÁCA

Študijný program :	Aplikovaná informatika
Číslo študijného odboru:	2511
Názov študijného odboru:	9.2.9 Aplikovaná informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	Ing. Rudolf Pribiš, PhD.
Konzultant ak bol určený:	

2024

Peter Likavec

Slovenská technická univerzita v Bratislave
Ústav automobilovej mechatroniky

Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024
Evidenčné číslo: FEI-16605-115084

STU FEI

ZADANIE BAKALÁRSKEJ PRÁCE

Študent:	**Peter Likavec**
ID študenta:	115084
Študijný program:	aplikovaná informatika
Študijný odbor:	informatika
Vedúci práce:	Ing. Rudolf Pribiš, PhD.
Vedúci pracoviska:	Ing. Ján Cigánek, PhD.
Miesto vypracovania:	ÚAMT

Názov práce: **Web aplikácia na správu OPC UA serverov**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Cieľom práce je návrh a implementácia web aplikácie na správu OPC UA serverov s použitím technológií: ASP .NET/.NET core, MVC, AngularJS, ReactJS, prípadne obdobné technológie.
Úlohy:
1. Analyzujte aktuálny stav pre oblasti Ad-hoc konektivity, digitálny opis aktíva pre I40, infraštruktúry AAS.
2. Navrhnite riešenie zabezpečujúce ad-hoc konektivitu pomocou OPC UA technológie s využitím štandardizovaného digitálneho opisu cez Asset Administration Shell a správou cez webovú aplikáciu.
3. Implementujte navrhnuté riešenie.
4. Overte riešenie nasadením pomocou kontajnerizácie alebo cez IIS.

Rozsah práce: 30 až 40 strán (54 000 až 72 000 znakov)

Zoznam odbornej literatúry:

1. PRIBIŠ, Rudolf; BEŇO, Lukáš; DRAHOŠ, Peter. Asset administration shell design methodology using embedded OPC unified architecture server. *Electronics*, 10. s. 20.
2. PRIBIŠ, Rudolf; DRAHOŠ, Peter. *Digitálne technológie pre sémantickú interoperabilitu v Industry 4.0: dátum obhajoby 22.8.2022*. Dizertačná práca. Bratislava : 2022. 199 s.
3. PRIBIŠ, Rudolf; HAFFNER, Oto; BEŇO, Lukáš; JANECKÝ, Dominik; KUČERA, Erik; PAJPACH, Martin; PAĽA, Adam; HAMRÁK, Michal; ZAJAC, Šimon. Emerging Trends in Education For Industry 4.0 and 5.0 Engineers in Slovakia Using E-learning Web Applications. In: *MoSICom 2023*. Piscataway: IEEE, 2023, ISBN 979-8-3503-9341-5.

Termín odovzdania bakalárskej práce:	31. 05. 2024
Dátum schválenia zadania bakalárskej práce:	20. 02. 2024
Zadanie bakalárskej práce schválil:	prof. Dr. rer. nat. Martin Drozda – garant študijného programu

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program :	Aplikovaná informatika
Vyberte typ práce	Web aplikácia na správu OPC UA serverov
Autor:	Peter Likavec
Vedúci záverečnej práce:	Ing. Rudolf Pribiš, PhD.
Konzultant ak bol určený:	
Miesto a rok predloženia práce:	Bratislava 2024

Bakalárska práca sa zaoberá získaním analýzou informácií o OPC UA serveroch, administratívnej schránke aktív a ich prepojeniu s Industry 4.0, ktoré sú základným pilierom vedomostí webovej aplikácie. Úvodná časť je zameraná na získanie vedomostí a porozumeniu oblasti ohľadom Internet vecí, administratívnej schránke aktív, OPC UA serverov a Industry 4.0. Nasledujúcou časťou je spraviť analýzu pre možnosti riešenia implementácie a zvoliť vhodné technológie pre návrh a vývoj aplikácie. Cieľom bakalárskej práce je vďaka získaným poznatkom navrhnúť, implementovať a následne nasadiť aplikáciu v IIS s využitím vhodných technológií, ktorá ma slúžiť na prácu s OPC UA servermi, aby mal užívateľ k dispozícii jednoduchý nástroj na prácu s OPC UA servermi. Realizácia práce obsahuje štruktúru webovej aplikácie, ktorá pozostáva z *frontend-u* a *backend-u*. Riešenie

zahŕňa detailný opis aplikácie ako aj zoznam a vysvetlenie všetkých použitých technológií, prehľad funkcionality a dôležitých častí, ktoré sú obsiahnuté v užívateľskej príručke. V závere práce sú zdokumentované a zhodnotené výsledky projektu spolu s odporúčaním pre ďalší rozvoj aplikácie.

Kľúčové slová: OPC UA, Webová aplikácia, Industry 4.0, AAS, RAMI 4.0, React.js, ASP.NET Core

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Bachelor Thesis:	Web application for managing OPC UA servers
Autor:	Peter Likavec
Supervisor:	Ing. Rudolf Pribiš PhD.
Consultant:	
Place and year of submission:	Bratislava 2024

The bachelor's thesis deals with obtaining and analyzing information about OPC UA servers, Asset Administration Shell, and their connection with Industry 4.0, which are the basic pillar of web application knowledge. The introductory part is aimed at gaining the knowledge and understanding of the area about the Internet of Things, Asset Administration Shell, OPC UA servers and Industry 4.0. The next part is to make an analysis for the possibilities of the implementation solution an choose suitable technologies for the design and development of the application. The goal of the bachelor's thesis is to design, implement and the deploy and application in IIS using right technologies, which will server to work with OPC UA servers, so user has a simple tool for working and managing OPC UA servers. The implementation of the work includes structure of the web application which consist of backend and frontend. The solution includes a detailed description of the application as well as a list of all technologies used, and overview of functionality and important parts. At the end of the work,

the results of the project are documented and evaluated together with recommendation for further development of application.

Key words: OPC UA, Industry 4.0, AAS, RAMI 4.0, Web application, React.js, ASP.NET Core

Vyhlásenie autora

Podpísaný Peter Likavec čestne vyhlasujem, že som Bakalársku prácu Web aplikácie na správu OPC UA serverov vypracoval na základe poznatkov získaných počas štúdia a informácií z dostupnej literatúry uvedenej v práci.
Uvedenú prácu som vypracoval pod vedením Ing. Rudolf Pribiš, PhD.

V Bratislave dňa 10.05.2024

.....
podpis autora

Poďakovanie

Týmto spôsobom by som chcel poďakovať vedúcemu práce Ing. Rudolf Pribiš, PhD. za usmernenie pri písaní práce a odbornú pomoc pri riešení práce.

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek a značiek

AAS	Asset Administration Shell
AASX	Súborový formát balíka pre AAS
API	Application Programming Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
VDOM	Virtual Document Object Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IIS	Internet Information Services
IoT	Internet of Things
IIoT	Industrial Internet of Things
JSON	JavaScript Object Notation
NPM	Node Package Manager
OPC UA	Open Platform Communications Unified Architecture
RAMI 4.0	Reference Architectural Model Industry 4.0
REST API	Representational State Transfer API
SQL	Structured Query Language
XML	Extensible Markup Language
MVC	Model-View-Controller
URN	Uniform Resource Name
GUI	Graphical User Interface
UI	User Interface
RDBMS	Relational Database Management System
CRUD	Create, Read, Update, Delete
DCOM	Distributed Component Object Model
COM	Component Object Model
SPA	Single Page Application
URI	Uniform Resource Identifier

Úvod

Industry 4.0, známa aj ako štvrtá priemyselná revolúcia, predstavuje revolučný posun v oblasti priemyselnej výroby, kde sa inteligentné technológie a kybernetické systémy prelínajú s tradičnými výrobnými procesmi. V tomto novom priemyselnom paradigme zohráva kľúčovú úlohu komunikácia a interoperabilita, čo prináša do popredia technológiu OPC UA. Technológia OPC UA sa stala rozhodujúcim prvkom v kontexte Industry 4.0, poskytuje spoľahlivý a bezpečný spôsob prenosu dát medzi rôznymi zariadeniami a systémami. Ich využitie umožňuje efektívne riadenie a monitorovanie výrobných procesov, a to nielen v rámci jedného podniku. Cieľom bakalárskej práce je dostatočné porozumenie materiálov týkajúcich sa témy a následne spracovať nadobudnuté poznatky a vytvoriť si vedomostnú bázu. Následne získané informácie analyzovať a určiť potrebné technológie pre vývoj aplikácie. Pri analýze a návrhu je potrebné nájsť riešenie podporujúce ad-hoc konektivitu s využitím štandardizovaného digitálneho opisu cez *Asset Administration Shell*. Výstupom bakalárskej práce je webovú aplikácia, ktorá bude umožňovať správu OPC UA serverov.

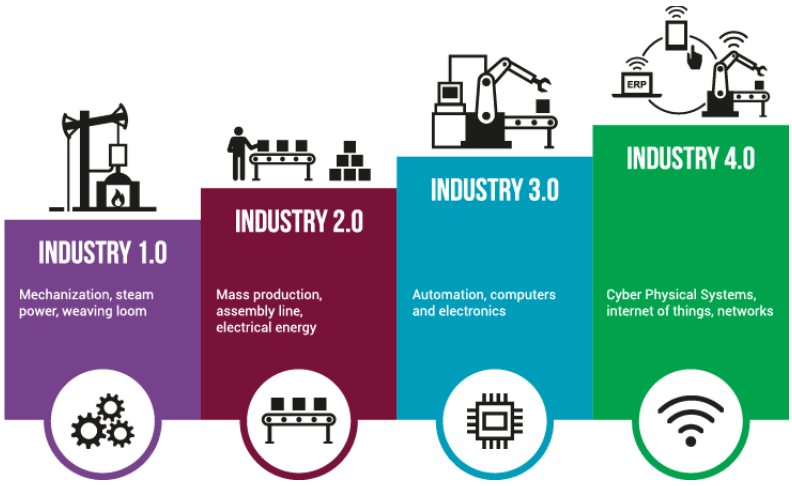
1 Teoretická časť

V úvodnej teoretickej časti práce sa budeme zameriavať na prehľad základných teoretických pojmov na ktorých je postavená aplikácia. Tieto pojmy sú kľúčové pre porozumenie a preto je dôležité aby pred pokračovaním do ďalších častí boli podrobne definované a vysvetlené vzťahy medzi jednotlivými pojmami.

1.1 Industry 4.0

Industry 4.0 označuje novú fázu priemyselnej revolúcie, ktorá môže byť definovaná integráciou inteligentných digitálnych technológií do výroby a priemyselných procesov. Toto umožňuje firmám a spoločnostiam inteligentnú výrobu a vytvorenie takzvaných inteligentných prevádzok. Cieľom je vylepšenie produktivity, efektívnosti a flexibility. Industry 4.0, ktorá veľmi úzko súvisí *s IIoT (Industrial Internet of Things),* spája fyzickú výrobu s operáciami s inteligentnými digitálnymi technológiami, strojovým učením a prácu s dátami aby vytvorili lepší ekosystém pre spoločnosti, ktoré sa zameriavajú na výrobu. Industry 4.0 je založená na 9 technologických pilieroch, pričom skutočná sila je dosiahnutá práve použitím technológií spoločne.

Komentár od [PL1]: https://www.techtarget.com/searcherp/definition/Industry-40



Obr. 1: Postupný vývoj Industry

1.2 OPC UA

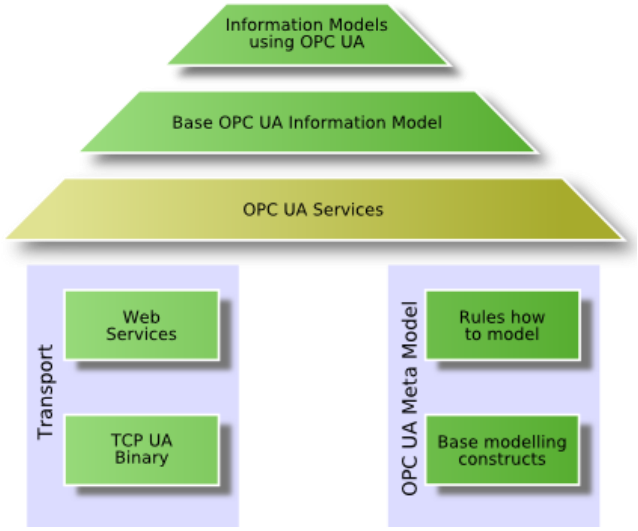
OPC UA je štandard pre interoperabilnú komunikáciu v priemysle, ktorý definovala organizácia *OPC Foundation*. Základy pôvodnej špecifikácie *vytvorili Fisher-Rosemount, Intellution, Opto 22* a *Rockwell Software* pre štandardný prístup k dátam založenom na *COM* a *DCOM*. OPCUA využíva mapovanie *(mapping)*, ktoré slúži na kódovanie dát nezávisle od technológie komunikačného protokolu.

Vďaka interoperabilite je OPC UA platformovo nezávislý komunikačný protokol s flexibilným informačným modelom, ktorý je dostatočne škálovateľný, aby mohol byť implementovaný v rôznych zariadeniach.

OPC UA sa používa v priemyselných doménach, ako sú priemyselné senzory a akčné členy, riadiace systémy, systémy na vykonávanie výroby a systémy plánovania podnikových zdrojov, vrátane priemyselného internetu vecí *(IIoT), Machine To Machine (M2M)*. Tieto systémy sú určené na výmenu informácií a na používanie príkazov a riadenia priemyselných procesov. OPC UA definuje spoločný model infraštruktúry na uľahčenie tejto výmeny informácií.

Komentár od [PL2]: OPC UA – história a špecifikácia – dočasný učebný text | Elearning portál - Mechatronika FEI STU - Mechatronika.cool

Komentár od [PL3]: Obrazek https://documentation.unified-automation.com/uasdkhp/1.4.1/html/_l2_opc_ua_overview.html



Obr. 2: Vrstvy OPC UA architektúry

1.2.1 OPC UA a vzťah s Industry 4.0

OPC UA hra kľúčovú rolu v rámci priemyslu 4.0, poskytuje štandardizovaný (IEC 62541) a platformovo nezávislý komunikačný protokol, na bezproblémovú komunikáciu medzi rôznymi priemyselnými zariadeniami. Podporuje dva komunikačné modely:

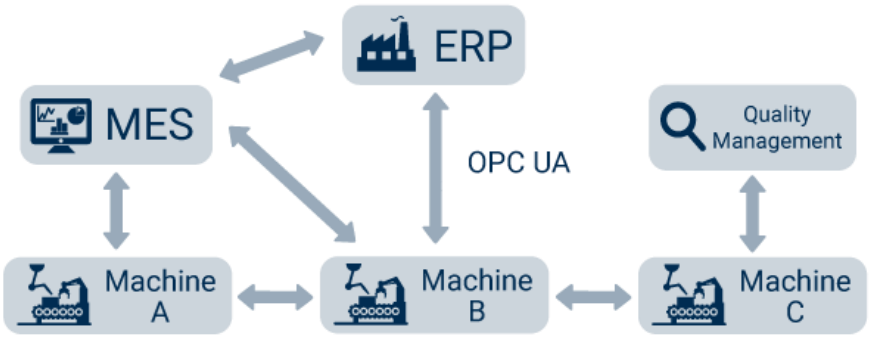
- Klient-Server
- Pub/Sub (publisher-subscriber)

OPC UA ma vstavanú bezpečnosť, ktorá je definovaná bezpečnostnými funkciami ako sú napr. šifrovanie, autentifikácia, autorizácia,...

Komentár od [PL4]: https://www.etmm-online.com/iot-basics-what-is-opc-ua-a-9be2f057ce7843b17843b2870298b591/

Komentár od [RP5]: Nie že podporuje, má ju vstavanú. Je to súčasť základných špecifikácií.

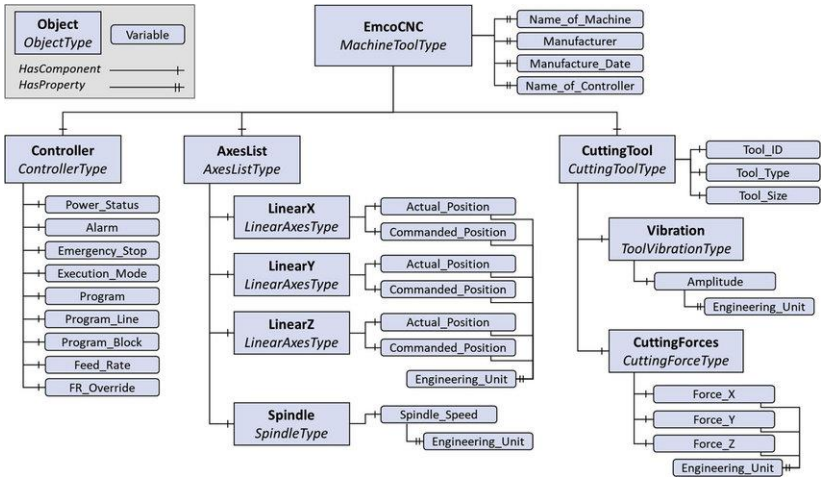
Komentár od [PL6]: https://www.opc-router.com/what-is-opc-ua/ obrazok



Obr. 3: Vzťah OPC UA s Industry 4.0

1.2.2 Informačný model

Informačný model v OPC UA je štandardizovaná reprezentácia štruktúry, organizácie alebo dát v systéme avšak rozumieť pod ním môžeme aj užívateľom vytvorený a skompilovaný Adresný priestor. Model nám umožňuje definovať ako sú informácie modelované, vymieňané medzi rôznymi zariadeniami a systémami, ktoré komunikujú prostredníctvom OPC UA.



Obr. 4: Príklad informačného modelu

Komentár od [PL7]: Obrazek https://www.researchgate.net/figure/OPC-UA-information-model-for-the-EMCO-Concept-105-Milling-Machine_fig2_332561075

Hlavnými komponentami informačného modelu sú:

- **Objekty a uzly (*nodes*) –** je uzol obsahujúci jednotlivé časti informácie ako sú premenné, metódy alebo *eventy*, objekty pozostáva z viacerých uzlov a formuje skupinu dát
- **Atribúty** – uzly v informačnom modeli s atribútmi, ktoré definujú dáta (hodnotu premennej, dátový typ,...)
- **Referencie** – definujú vzťahy medzi jednotlivými uzlami
- Dátové typy – OPC UA definuje set štandardizovaných dátových typov, ktoré popisujú dáta a ich hodnoty
- **Metódy** – informačný model môže zahŕňať metódy, ktoré reprezentujú funkcie alebo operácie ktoré môžu byt spustené

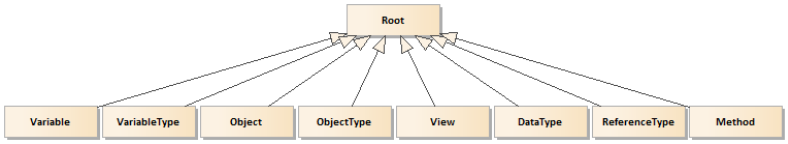
Informačný model predstavuje framework na popis štruktúr a sémantických dát, zabezpečenie konzistentnosti a interoperability medzi rôznymi zariadeniami. V skratke, informačný model OPC UA je štandardizovaná definícia jedinečných uzlov v adresnom priestore servera OPC UA. Jedinečnosť uzlov zabezpečuje ID uzla, pričom každý objekt, systém zariadení alebo aj celú továreň je možné reprezentovať pomocou prepojenia uzlov a ich vzťahov v adresnom priestore OPC UA servera.

1.2.3 Adresný priestor OPC UA

Hlavnou úlohou adresného priestoru (*AddressSpace*) v OPC UA je poskytnúť štandardný spôsob pre servery na reprezentáciu objektov pre klientov. Adresný priestor je modelovaný pomocou uzlov, ktoré sú sprístupnené pre klientov pomocou OPC UA služieb *(services)*. Uzly v adresnom priestore sa používajú na reprezentáciu reálneho objektu, ich definícii a referencii medzi sebou.

Model adresného priestoru je definovaný nasledovným setom typu *Node* (uzol):

- **Náhľad *(View)* –** definuje podmnožinu uzlov v adresnom priestore
- **Typ objektu *(ObjectType)* –** poskytuje definíciu objektov
- **Objekt *(Objekt)* –** používa sa na reprezentáciu komponentov, systémov, objektov reálneho sveta a softvérových objektov
- **Referenčný typ *(ReferenceType)* –** *používa sa na* definovanie vzťahov medzi uzlami
- **Dátový typ *(DataType)* –** slúži na definovanie jednoduchých a komplexných hodnôt premennej
- **Typ premennej *(VariableType)* –** používa sa na definovanie typu premennej
- **Premenná *(Variable)* –** *používa* sa ako úložisko premennej v danom okamihu, ktorá obsahuje hodnotu
- **Metóda *(Method)* –** je funkcia ktorej rozsah je ohraničený objektom, kde je definovaná



Obr. 5 Model adresného priestoru

1.2.4 OPC UA Server

OPC UA server je architektonický model serverového endpointu pre klientske/serverové interakcie.

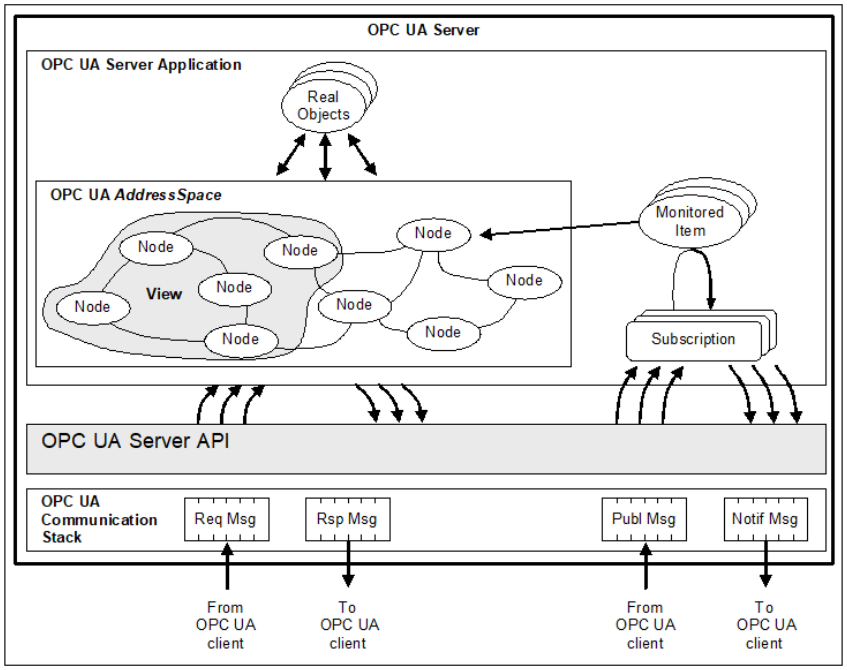
- **Reálne objekty *(Real Objects)* –** *sú* fyzické alebo softwarové objekty sprístupnené pre serverovú aplikáciu.

Komentár od [PL8]: https://reference.opcfoundation.org/Core/Part3/v104/docs/4

Komentár od [PL9R8]: https://commsvr.gitbook.io/ooi/semantic-data-processing/addressspaceaddressspacemodel

Komentár od [PL10]: https://reference.opcfoundation.org/Core/Part1/v104/docs/6.3

- **Serverová aplikácia *(Server application)*** – je kód, ktorý implementuje funkcionalitu serveru, využíva API na posielanie a prijímanie správ od klienta. API je interný *interface*, ktorý izoluje aplikáciu od OPC UA komunikačného *stacku*.
- **Adresný priestor *(AddressSpace)*** – je modelovaný ako kolekcia uzlov sprístupnených pre klienta použitím OPC UA služieb *(services)*.
- **Náhľad adresného priestoru *(AddressSpace View)*** – slúži na reštrikciu uzlov, ktoré server umožní pre klienta
- **Monitorované položky *(MonitoredItems)*** – sú entity v servery vytvorené klientom, ktoré monitorujú uzly v adresnom priestore.



Obr. 6: Architektúra OPC UA serveru

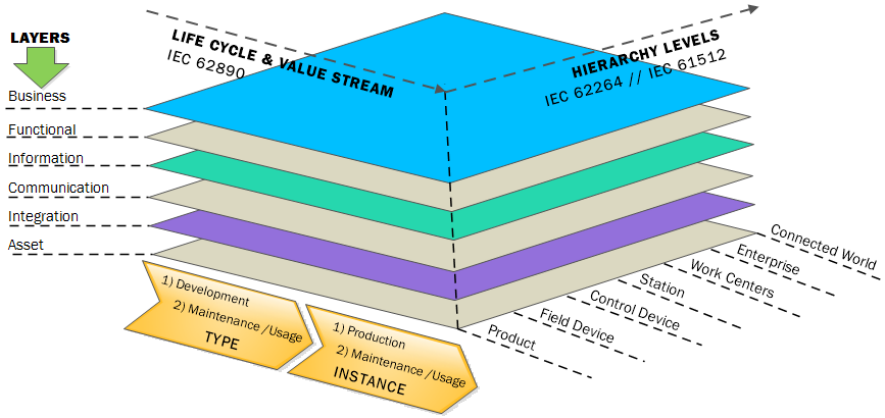
1.3 RAMI 4.0

RAMI 4.0 alebo *Reference Architectural Model Industry 4.0* je trojrozmerná mapa, ktorá ukazuje ako sa vysporiadať s problematikou Industry 4.0 systematickým a štruktúrovaným spôsobom. RAMI 4.0 je jednotný model pre všetky komponenty, ktorý zabezpečuje účastníkom zapojeným do ekosystému Industry 4.0 efektívne zdieľanie dát a informácii.

RAMI 4.0 mapuje všetkých účastníkov v priemyselnom odvetví do troch osí definície:

- Os vrstiev architektúry *(Layers axis)*
- Os životného cyklu a hodnotového toku *(Life Cycle Value Stream)*
- Os úrovní hierarchie (Hierarchy Levels axis)

Komentár od [PL11]: https://industry40.co.in/rami-reference-architecture-model-industry-4-0/



Obr. 7: 3-dimenzionálna mapa ako riešiť problém v Industry 4.0

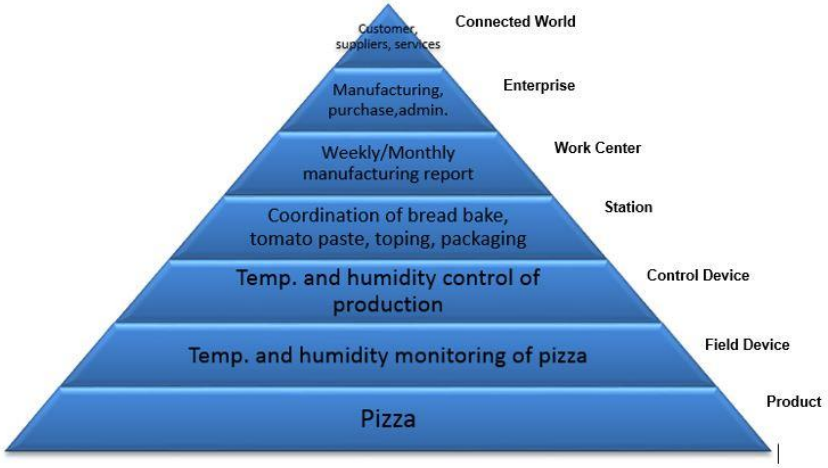
1.3.1 Layers axis (Os vrstiev architektúry)

Industry 4.0 architektúra na hierarchickej úrovni ukazuje funkčné priradenie komponentov. Táto os v rámci podniku alebo závodu sa riadi normami *IEC 62264 a IEC 61512*. Úroveň nad a pod oblasťou noriem *IEC* predstavuje kroky a popisuje ako skupiny továrni spoluprácu v rámci externých firiem.

Úrovne hierarchie sú:

- Pripojený svet (Connected World)
- Podnik (Enterprise)
- Pracovné centrum *(Work Center)*
- Stanica (Station)
- Riadiace zariadenie *(Control Device)*
- Poľné zariadenie *(Field Device)*
- Produkt *(Product)*

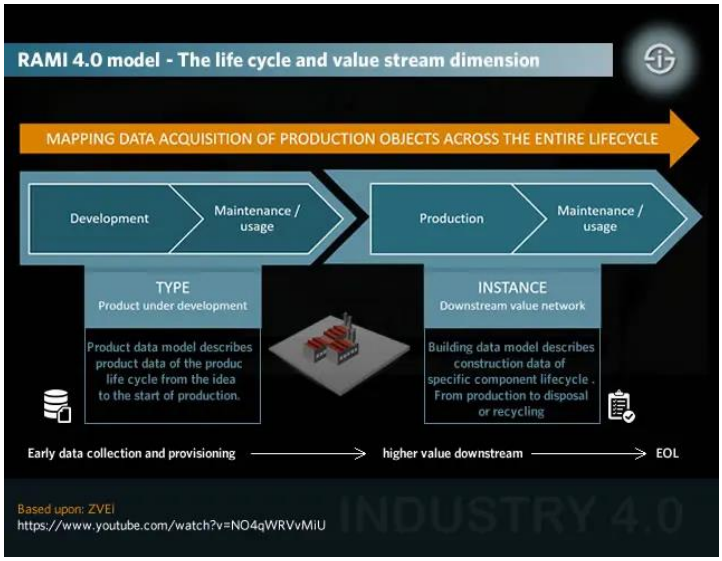
Komentár od [PL12]: https://community.sap.com/t5/technology-blogs-by-members/rami-4-0-for-pizza-lovers-part-3/ba-p/13321143 obrazek



Obr. 8 Opis hierarchického levelu na príkladu výroby pizze

1.3.2 Life Cycle Value Stream axis (Os životného cyklu a hodnotového toku)

Os životného cyklu a hodnotového toku je rozdelená na dve časti a sú nimi typ a inštancia. Typ je rozdelený na vývoj a údržbu/použitie, kým inštancia je rozdelená na produkciu a údržbu/použitie. Typ reprezentuje počiatočnú myšlienku vývoja produktu, kým každý vyrobený produkt reprezentuje inštanciu tohto typu. Inak povedané kým je produkt v štádiu vývoja označujeme ho ako "typ", akonáhle sa presunie do výroby označujeme ho "inštanciou". Hocikedy kedy je produkt redizajnovaný alebo je pridaná nová vlastnosť, opäť sa stáva typom.



Obr. 9 Mapovanie získavania údajov o výrobe počas celého životného cyklu

Komentár od [PL13]: https://congtykhaian.com.vn/industry-4-0-and-the-fourth-industrial-revolution-explained/ obrzek

1.3.3 Hierarchy Levels axis (Os vrstiev architektúry)

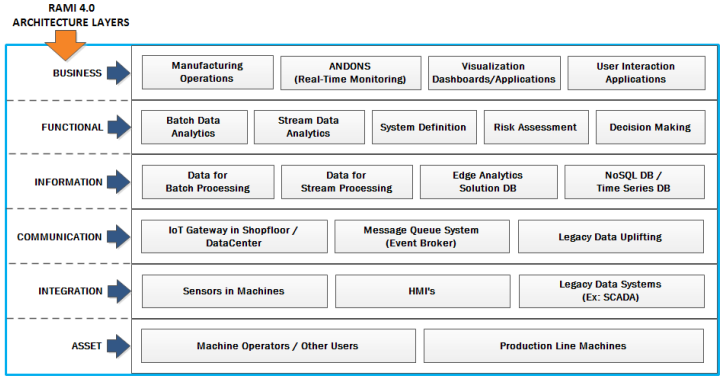
Sú to vrstvy architektúry, ktoré reprezentujú pohľad na tú istú vec z rôznych perspektív.

Obr. 10 Vrstvy architektúry RAMI 4.0

Komentár od [PL14]: Obrazek https://industry40.co.in/rami-reference-architecture-model-industry-4-0/

RAMI 4.0 rozdeľuje komplexné procesy do častí, čo umožňuje jednoduchšie pochopenie a už od začiatku zahŕňa ochranu údajov a bezpečnosť IT. Odpovedá na všetky otázky k problematike ohľadom sémantiky, identifikácie, funkcií, komunikačných štandardoch pre

inteligentnú továreň. S RAMI 4.0 architektúrou je továreň nie je vrchnou vrstvou ale sieťou interakcii medzi inteligentnými produktami a svetom.



Obr. 11 Zobrazenie architektúry riešenia analýzy RAMI 4.0

Komentár od [PL15]: https://industry40.co.in/rami-reference-architecture-model-industry-4-0/ obrazek

1.4 IoT – Internet of Things (Internet vecí)

Internet vecí opisuje sieť fyzických objektov – zariadení, ktoré obsahujú senzory, softvér a iné technológie na účel komunikácie a výmeny údajov s inými zariadeniami cez internet. Tieto zariadenia siahajú od bežných domácich zariadení až po sofistikované premyslené stroje. Medzi zariadenia patriace do *IoT* patrí každé zariadenie, ktoré môže byť monitorované alebo ovládané zo vzdialenej lokácie.

Komentár od [PL16]: https://www.oracle.com/internet-of-things/what-is-iot/

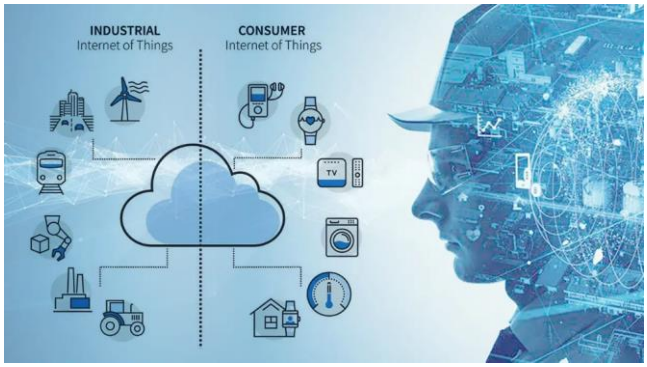
1.5 IIoT – Industrial Internet of Things (Priemyselný internet vecí)

Industrial Internet of Things je kolekcia senzorov, nástrojov a autonómnych zariadení pripojených cez internet k priemyselným aplikáciám. Táto sieť umožňuje zhromažďovať dáta, vykonávať analýzy a optimalizovať výrobu, zvyšovať efektivitu a znižovať náklady na výrobný proces a poskytovanie služieb. Priemyselné aplikácie sú systémy, ktoré prepájajú zariadenia a následne ich pracovníci s nimi riadia. Súčasné IIoT aplikácie sa predovšetkým sústreďujú na výrobu, dopravu a energetiku.

Komentár od [PL17]: https://www.iberdrola.com/innovation/what-is-iiot

Rozdielom medzi *IoT* a *IIoT* je, že kým *Internet of Things* sa zameriava na služby pre zákazníkov (spotrebiteľov), *Industrial Internet of Things* sa zameriava na zvýšenie bezpečnosti a efektivity vo výrobných prevádzkach. Inteligentné zariadenia pre domácnosť, virtuálne asistenti a teplotné senzory sú typickými príkladmi spotrebiteľskej technológie,

zatiaľ čo zariadenia *IIoT* predstavujú sieťové systémy, ktoré produkujú dáta určené na analytické účely.



Obr. 12: Industrial Internet of Things vs Internet of Things

Komentár od [PL18]: https://www.mokosmart.com/iiot-vs-iot-technologies obrazek

1.6 Asset Administration Shell

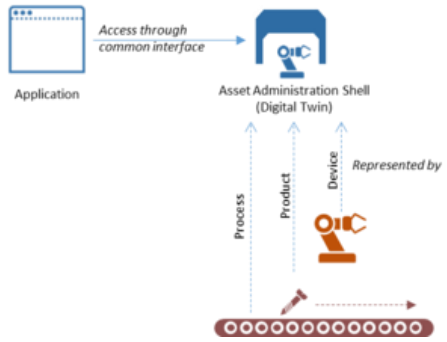
Asset Administration Shell (AAS) je štandardizovaná digitálna reprezentácia *aktíva (IEC 63278)*. Je nástrojom na tvorbu *„Industry 4.0 komponentov"*. Industry 4.0 komponenty sú kombináciou aktíva (senzor, stroj,..) a jej digitálnej reprezentácie – AAS. AAS môže byť logickou reprezentáciou jednoduchého komponentu alebo stroja. AAS pozostáva zo submodelu, v ktorých sú opísané všetky informácie ako funkcionalita daného aktíva, ktorá zahŕňa vlastnosti, charakteristiku, nastavenia, statusy, parametre a namerané dáta.

Komentár od [PL19]: https://reference.opcfoundation.org/I4AAS/v100/docs/4.1

Štruktúra AAS je definovaná pomocou UML a popisuje mapovanie do formátov XML, JSON alebo OPCUA, ktorými je možné zdieľať údaje v Industry 4.0 Jeho obsah je definovaný prostredníctvom submodelu špecifického pre doménu.

Komentár od [PL20]: https://wiki.eclipse.org/BaSyx_/_Documentation_/_AssetAdministrationShell odrazky + obrazek

- ***Type 1 Asset Administration Shell*** **–** Súbory typu JSON alebo XML, obsahujú statické informácie a môžu byť distribuované ako súbor
- ***Type 2 Asset Administration Shell*** **–** Existujú ako bežiace inštancie, sú hosťované na servery, môžu obsahovať statické informácie ale môžu interagovať s ostatnými komponentami. Type 2 AAS poskytujú frontend napríklad pre zariadenia, dáta zo senzorov...
- ***Type 3 Asset Administration Shell*** **–** Sú nadstavbou Type 2 AAS, avšak navyše dokážu samé od seba začať komunikáciu medzi sebou

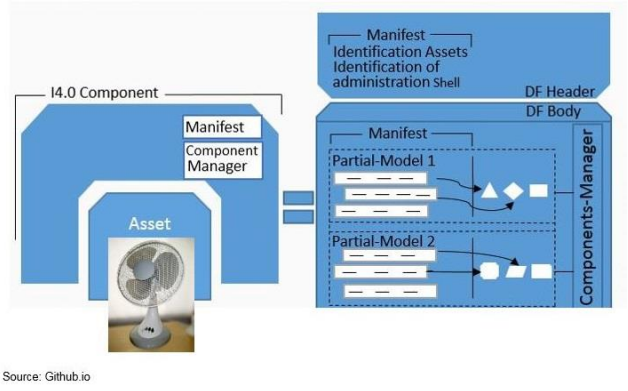


Obr. 13: AAS reprezentujúci produkt pre aplikáciu

1.6.1 Štruktúra AAS

AAS pozostáva z hlavičky a tela. Hlavička obsahuje informáciu pre identifikáciu, administráciu a použitie aktíva, sub-komponenty a AAS ako celok. Tieto informácie sú uložené v časti *manifest* v hlavičke. Telo AAS obsahuje submodely ktoré obsahujú hierarchicky organizované nastavenia aktíva. Tieto nastavenia obsahujú vlastnosti ktoré referujú dátam a metódam ktoré využíva aktívum. Telo AAS obsahuje takisto *manifest* ktorý pozostáva zo zoznamu všetkých submodelov.

Komentár od [PL21]: https://community.sap.com/t5/technology-blogs-by-members/asset-administration-shell-aas-for-beginners/ba-p/13346471 aj obrzek



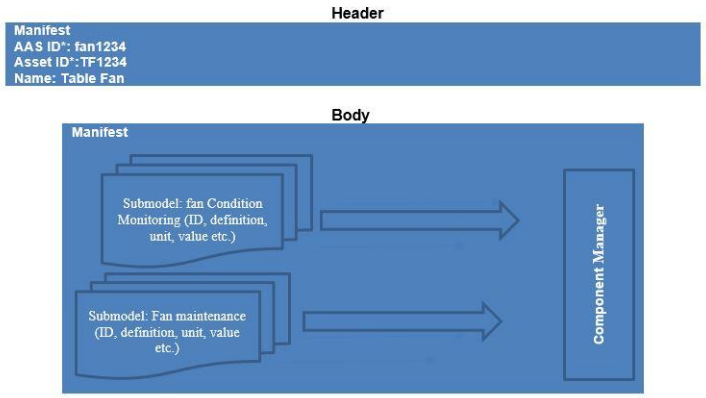
Obr. 14: Štruktúra AAS všeobecne

1.6.2 AAS Submodel

AAS obvykle obsahuje viacero submodelov. Submodely definujú aspekty a technické funkcionality (metódy, funkcie). Submodely môžu obsahovať vlastnosti, funkcie, *eventy*,

referencie, vzťahy. Toto umožňuje poskytovať veľké množstvo údajov pre submodely. AAS používa striktný formát ktorý organizuje dáta ako strom vlastností, pričom rovnaký formát je použitý aj pre štruktúru vlastností submodelov.

Komentár od [PL22]: https://community.sap.com/t5/technology-blogs-by-members/asset-administration-shell-aas-for-beginners/ba-p/13346471 aj obrzek



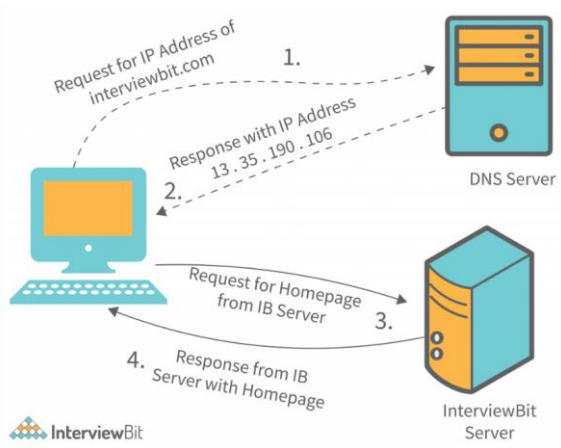
Obr. 15: AAS Submodel príklad [15]

1.7 Klient-server architektúra

Architektúra klient-server je spôsob komunikácie medzi klientom a serverom, pričom majú oddelené úlohy. Princíp komunikácie zodpovedá vzoru požiadavka-odpoveď *(request-response*) a mal by dodržiavať štandardný komunikačný protokol. Medzi tieto architektúry patri:

- Dvojvrstvová architektúra
- Dvojvrstvová architektúra
- N-vrstvová architektúra

Komentár od [PL23]: https://www.interviewbit.com/blog/client-server-architecture/ aj text aj obrazek



Obr. 16

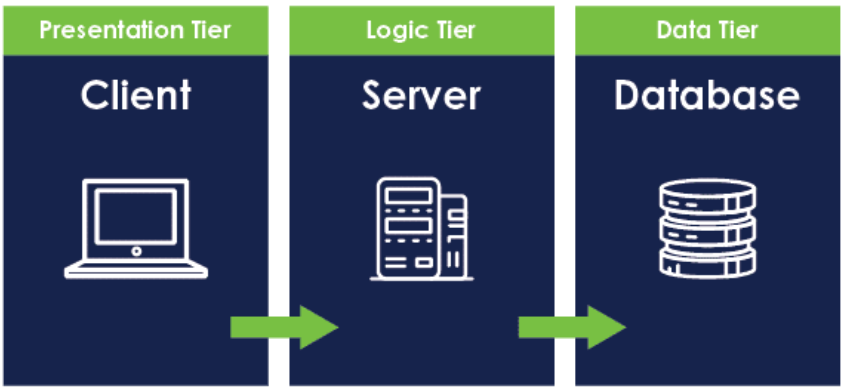
1.8 Trojvrstvová architektúra

Trojvrstvová architektúra *(Three Tier Architecture)* alebo aj *full-stack* architektúra je softvérová architektúra klient-server , ktorá rozdeľuje aplikáciu do 3 nezávislých vrstiev. Hlavným cieľom tejto architektúry je zlepšiť modularitu, škálovateľnosť a flexibilitu softvéru. Medzi tieto vrstvy patria:

- Klientska (Prezentačná) vrstva
- Serverová (Logická) vrstva
- Dátová vrstva

Tento vzor pomáha dosiahnuť separáciu medzi jednotlivými vrstvami a tým každá vrstva dokáže fungovať nezávisle, avšak prostredníctvom dobre definovaných rozhraní, je umožnená komunikácia medzi jednotlivými vrstvami.

Komentár od [PL24]: https://www.zirous.com/2022/11/15/three-tier-architecture-approach-for-custom-applications-2/ aj obrazek



Obr. 17:

1.8.1 Klientska vrstva

Táto vrstva reprezentuje rozhranie aplikácie a je zodpovedné za prezentáciu dát a interakciu s používateľom. Pozostáva z grafických prvkov používateľného rozhrania ako sú samotné webstránky, formuláre,... Táto časť aplikácie sa obvykle nazýva *frontend* a využíva kombináciu *HTML, CSS* a *JavaScript-u* na vytvorenie interaktívneho rozhrania. Často sa využívajú frameworky ako sú *React.js., Angular* alebo *Vue.js*, ktoré zvyšujú efektivitu vopred pripravených komponentov a zaisťujú štruktúrovaný a konzistentný prístup ku kódovaniu.

Komentár od [PL25]: Rovanke ako predtym hore

1.8.2 Serverová vrstva

Serverová alebo aplikačná vrstva slúži na spracovávanie zadaných príkazov alebo vstupov na klientskej (prezenčnej vrstve). Táto časť aplikácie sa obvykle nazýva *backend*. Vrstva funguje ako mozog aplikácie a vytvára riadený a bezpečný spôsob komunikácie medzi prezenčnou a dátovou vrstvou.

Keď používateľ vykoná akciu na prezenčnej vrstve aplikačná vrstva požiadavku (*request*) spracuje, vykoná interakciu s dátovou vrstvou a vráti odpoveď (*response*) aplikačnej vrstve. Tieto operácie zahŕňajú čítanie, vytváranie, aktualizáciu alebo odstraňovanie *(CRUD)* dát

Komentár od [PL26]: Rozvanke ako prednym

z dátovej vrstvy. Aplikačná vrstva *(backend)* býva často realizovaná ako REST API, čo umožňuje jednoduchú a štandardizovanú komunikáciu medzi klientom a serverom.

1.8.3 Dátová vrstva

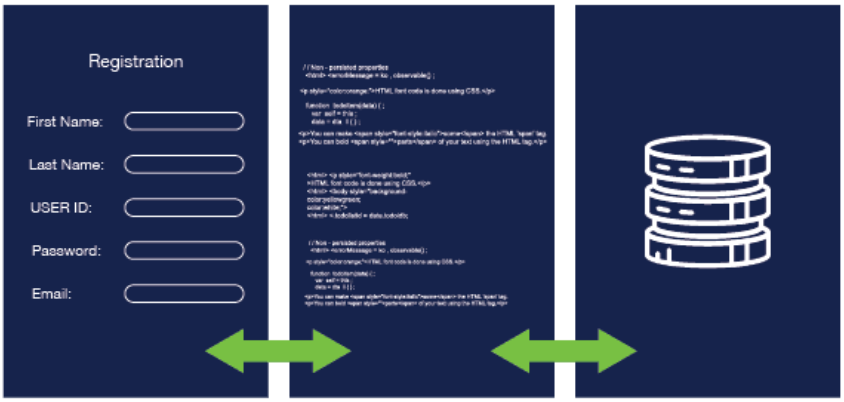
Dátová vrstva je súčasťou *backend-u* a je zodpovedná na ukladanie, získavanie a správu používaných údajov webovou aplikáciou. Slúži ako štruktúrované (SQL) alebo neštruktúrované (NoSQL) úložisko pre údaje a umožňuje manipuláciu s údajmi. Dátová vrstva príjme požiadavky z aplikačnej vrstvy, spracováva ich a následne vykonáva príslušne operácie nad databázou. Po vykonaní vracia odpoveď, ktorá obsahuje výsledky operácie.

1.8.4 Interakcia vrstiev

Jednoduchým príkladom interakcie medzi jednotlivými vrstvami je zobrazený na procese registrácie (Obr. 18). Najprv sa používateľ dostane na prezenčnú vrstvy a zadá informácie do formuláru. Po kliknutí na tlačidlo sa odošle požiadavka *(request)* do aplikačnej vrstvy kde sa dáta spracujú a uložia do dátovej vrstvy pomocou určitej logiky. Následne na to sa používateľovi vráti odpoveď *(response)* prostredníctvom prezenčnej vrstvy ci bola operácia úspešná alebo neúspešná.

Komentár od [PL27]: Teiz ten insty zdroj ako predtym

Komentár od [PL28R27]: Iba obrazek lebo text som pisal sam



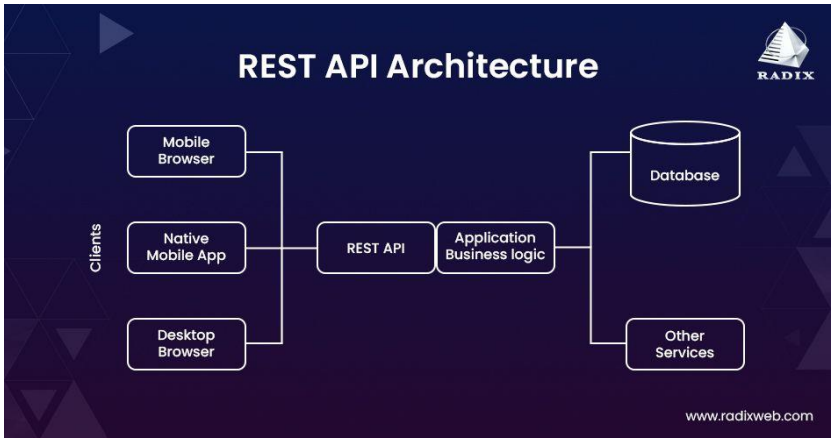
Obr. 18: Interakcia medzi jednotlivými vrstvami v trojvrstvovej architektúre

1.9 REST API

REST API alebo *RESTful API* je aplikačné programové rozhranie (API), ktoré umožňuje dvom aplikáciám medzi sebou komunikovať a vymieňať dáta. V rámci aplikácie REST API tvorí súčasť aplikačnej vrstvy *(backend-u)*. Musí dodržiavať určité pravidla ako môže aplikácie komunikovať prostredníctvom HTTP metód. Medzi metódy ktoré sa najväčšie používajú v rozhraní REST API patria:

- GET
- POST
- PUT/PATCH
- DELETE

Pomocou požiadaviek *(request-ov)* na API endpointy sa zabezpečuje prístup k údajom a operáciám v rámci *RESTful API*. Tieto cesty sú definované na strane servera a prestavujú URI, na ktorých klienti môžu komunikovať so serverom.



Obr. 19: Štruktúra REST API pri trojvrstvovej architektúre

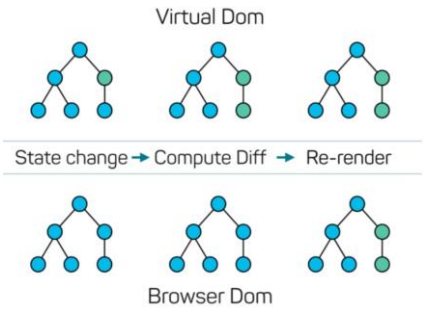
1.10 Virtual Document Object Model

Virtual DOM je kľúčový koncept pri využívaní *frameworkov*, ako je React.js, ktorý výrazne zlepšuje výkon aktualizácie skutočného DOM. Keď nastane zmena stavu v komponente, React.js namiesto priamej aktualizácie skutočného DOM vytvorí "ľahkú" repliku DOM ako *Virtual DOM*. React.js potom porovná aktualizovaný *Virtual DOM* s predchádzajúcim, aby detegoval zmeny, ktoré nastali, ktoré je potrebné použiť na skutočný DOM. Akonáhle sú

Komentár od [PL29]: https://radixweb.com/blog/graphql-vs-rest#REST aj obrzek

Komentár od [PL30]: https://dev.to/adityasharan01/react-virtual-dom-explained-in-simple-english-10j6

zmeny určené, React.js aktualizuje skutočný DOM optimalizovaným a efektívnym spôsobom bez zbytočného opätovného vykresľovania.



Obr. 20: Priebeh vykresľovania od zmeny po vykreslenie pri porovnaní VDOM a DOM

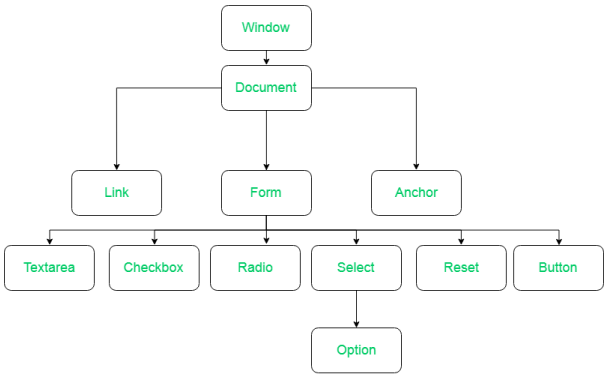
1.11 Document Object Model

Document Object Model alebo skrátene DOM je programovacie rozhranie pre webové dokumenty. DOM predstavuje štruktúru HTML webovej stránky, ku ktorej je možné pristupovať a manipulovať s ňou pomocou JavaScript-u. Keď sa webová stránka načíta, prehliadač vytvorí DOM, ktorý pozostáva zo všetkých prvok HTML komponentov na stránke.

Komentár od [PL31]: https://www.geeksforgeeks.org/dom-document-object-model/

Komentár od [PL32R31]: aj obrzke

Pomocou JavaScript-u je možné DOM upravovať a aktualizovať webovú stránku. Napríklad pokiaľ chce vývojár zmeniť textový obsah tlačidla, pomocou JavaScript-u vie prvok vybrať a následne pomocou DOM jeho textový obsah aktualizovať [19].



Obr. 21: Reprezentácia DOM [20]

2 Použité technológie

Pri návrhu a implementácii aplikácie je kľúčové zvoliť technológie, ktoré umožnia efektívne a spoľahlivé riešenie požiadaviek projektu, V tejto časti si prejdeme technológie ktoré sú použité v aplikácii.

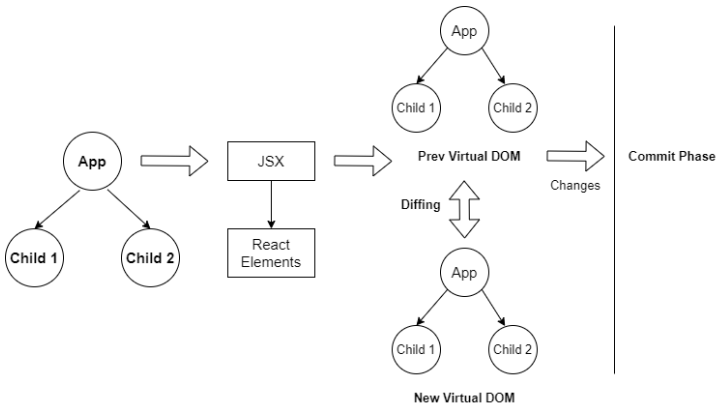
2.1 React.js

React.js je open-source JavaScript knižnica, vytvorená spoločnosťou Facebook, ktorej cieľom je zjednodušiť proces vytvárania interaktívnych používateľských rozhraní. Používateľské rozhranie je vytvorené pomocou React komponentov, pričom každý zodpovedá za výstup časti HTML kódu, ktorý môže byť opätovane použitý.

Pri vyvíjaní *frontend-u* aplikácie, React používa znovu použiteľné komponenty, ktoré môžu byť považované za nezávislé lego bloky. Tieto komponenty sú jednotlivé časti finálneho UI, pričom pri použití viacerých tvoria celkovú front-end aplikáciu.

Hlavnou úlohou React-u v aplikáciách je spracovať zobrazenie aplikácie, tým že poskytuje najlepšie a najefektívnejšie prevedenia vykresľovania na web stránke. Namiesto toho aby sa celé používateľské rozhranie bralo ako jeden celok, React.js umožňuje vývojárom UI rozdeliť na viacero jednotlivých používateľských komponentov, ktoré formujú celé používateľské rozhranie. React kombinuje rýchlosť a efektivitu JavaScript-u s efektívnou metódou manipulácie s VDOM objektami na rýchlejšie vykresľovanie webových stránok a vytváranie vysoko dynamických a responzívnych webových aplikácii.

Komentár od [PL33]: Obr https://blog.bitsrc.io/how-react-renders-a-component-on-screen-da97c56caf71?gi=4609f48204af



Obr. 22: Priebeh vykresľovania pri React.js

2.2 ASP.NET Core

ASP.NET Core je *cross-platformový*, vysoko výkonný *framework* pre vývoj moderných aplikácii. Je to open-source verzia ASP.NET, ktorá podporuje beh na Windowse, macOS a Linuxe, bola spustená v roku 2016 ako novšia verzia starého ASP.NET, ktorú podporoval iba Windows. Architektúra je postavená na N-vrstvovej architektúre, čím umožňuje používateľovi flexibilitu pri tvorbe aplikácii.

Komentár od [PL34]: https://www.simplilearn.com/tutorials/asp-dot-net-tutorial/asp-dot-net-core-tutorial

2.3 Eclipse BaSyx

Eclipse BaSyx je *open-source* platforma pre automatizáciu novej generácie. Eclipse BaSyx poskytuje bežné a opakovane použiteľné komponenty Industry 4.0, umožňuje jednoduché vytváranie nových funkcií okolo oficiálneho *HTTP REST* rozhrania AAS. Medzi hlavné komponenty *Eclipse BaSyx* patrí *AAS Server* a *AAS Registry* ako hlavné komponenty pre vývoj Industry 4.0 aplikácií.

Komentár od [PL35]: https://projects.eclipse.org/projects/dt.basyx

2.3.1 Eclipse BaSyx AAS Server

AAS Server komponent poskytuje prázdny AAS Server ktorý môže byť použitý ako host pre viacero AAS alebo Submodelov. Pri spustení AAS servera bez konfigurácie bude vrátený prázdny JSON. Na konfiguráciu AAS Server pri použití Docker kontajnerov slúži *aas.properties* súbor, kde sú nakonfigurované nastavenia ako napríklad cesta k súboru, ktorý ma byť použitý ako zdroj pre AAS Server.

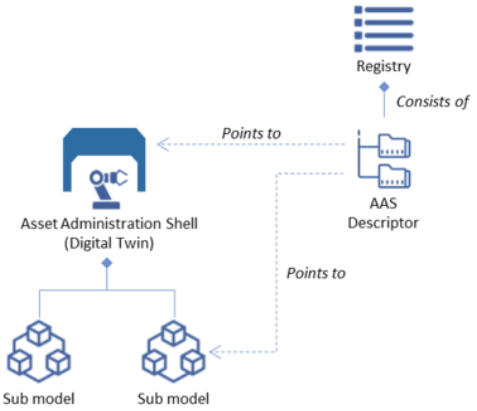
Komentár od [PL36]: https://wiki.eclipse.org/BaSyx_/_Documentation_/_Components_/_AAS_Server

2.3.2 Eclipse BaSyx AAS Registry

AAS Registry je komponent AAS infraštruktúry, ktorý slúži na zobrazenie všetkých dostupných AAS a ich submodelov pomocou jedinečných ID a ukladá dodatočné *meta-data* pre AAS. AAS Registry registruje AAS deskriptory ktoré popisuje *Asset Administration Shell* a aj submodely AAS. Registry komponent je dostupný ako Docker kontajner ako súčasť *open-source BaSyx middleware* podobne ako AAS Server.

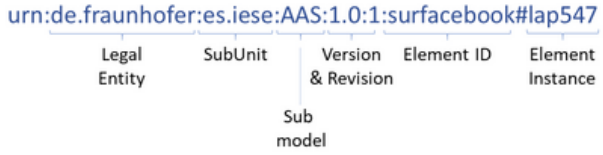
Komentár od [PL37]: https://wiki.eclipse.org/BaSyx_/_Documentation_/_Registry

Komentár od [PL38R37]: aj obrazek



Obr. 23: Registry komponent v Eclipse BaSyx

AAS musia byť registrované v Registry komponente aby sa zabezpečilo, že ich je možné nájsť podľa ich ID. Za jeho registráciu je zodpovedný komponent, ktorý pridáva nový AAS. Registry komponent v *Eclipse BaSyx* nemá stanovený pevný formát pre ID pre AAS alebo submodel, avšak je dôležité aby každé ID bolo jedinečné.



Obr. 24: Príklad ID pre AAS v Registry komponente

Eclipse BaSyx navrhuje formát ako by malo ID vyzerať:

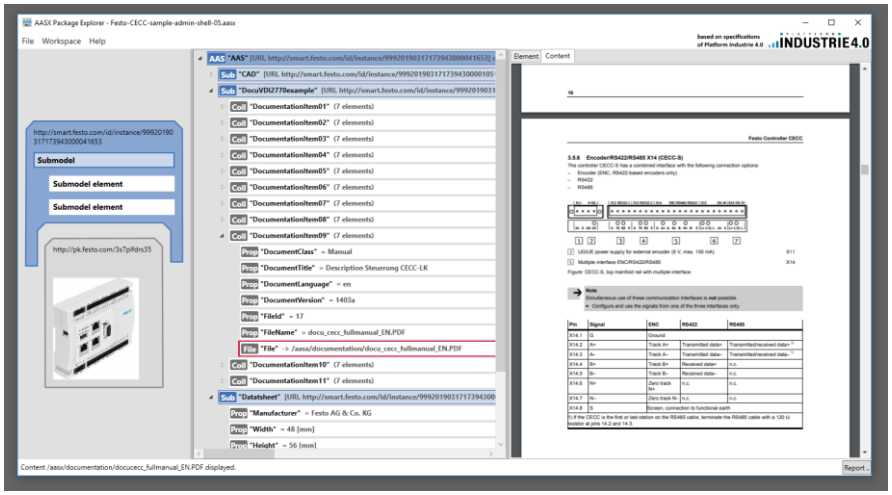
- ***Legal Entity* –** jedinečný identifikátor entity, ktorý využíva AAS
- ***SubUnit* –** odpovedá časti entity, napríklad divízii ktorá zodpovedá za aktívum
- ***SubModel*** – definuje submodel, ktorý je referovaný URN, ukazuje na AAS v tomto prípade alebo je to typ submodelu
- **Verzia *(Version)*** – definuje verziu AAS
- ***Revision* –** mala by byť inkrementačná každou zmenou AAS alebo submodelu
- ***Element ID* –** definuje typ aktíva, ktorý je referovaný AAS alebo submodel
- ***Element Instance* –** identifikuje konkrétne aktívum

2.3.3 Eclipse AASX Package Explorer

AASX Package Explorer je *open-source* aplikácia vyvinutá v jazyku C#, ktorá slúži na zobrazenie a editáciu *Industry 4.0 AAS*. *Eclipse AASX Package Explorer* je nástroj s grafickým rozhraním pomocou ktorého používateľ dokáže pracovať s AAS. Aplikácia taktiež obsahuje interný REST server. Pomocou aplikácie je možne vytvoriť AAS s ktorou dokážeme pracovať a využiť ako digitálne aktívum, ktoré je možné exportovať napr. do súboru s príponou .aasx.

Komentár od [PL39]: https://projects.eclipse.org/projects/dt.aaspe aj obrazek

Komentár od [PL40R39]: Obrazek https://github.com/eclipse-aaspe/aaspe



Obr. 25: Grafické rozhranie Eclipse AASX Package Explorer

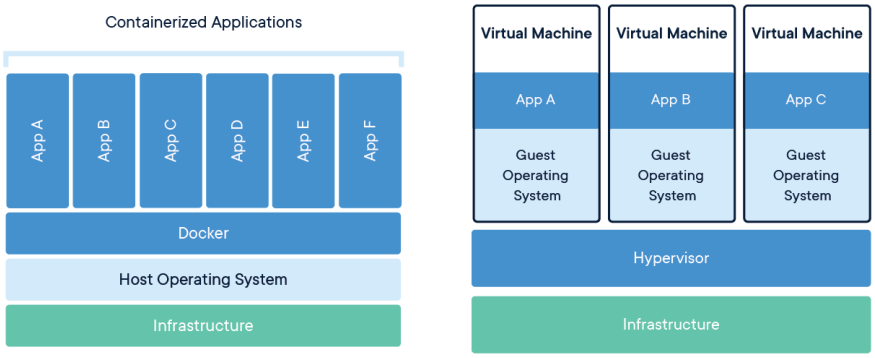
2.4 Docker

Docker je *open-source* platforma pre vývoj a spúšťanie aplikácii. Docker umožňuje oddeliť aplikácie od štruktúry, aby mohol byť softvér rýchlejšie dodaný. Docker poskytuje možnosť zabaliť aplikáciu a spustiť ju v izolovanom prostredí nazývanom Docker kontajner *(container)*. Izolácia a bezpečnosť vám umožnia súčasne spúšťať viacero aplikácii cez kontajnery na danom hostiteľovi. Kontajnery nie sú veľkostne náročné a obsahujú všetko čo je potrebné pre spustenie aplikácie, takže sa používateľ nemusí spoliehať na to čo je nainštalované na hosťujúcom zariadení. Docker klient komunikuje cez *Docker daemon*, ktorý zabezpečuje zostav a následne beh Docker kontajneru. Medzi najzákladnejšie prvky Docker-u patrí Docker image (obraz) a Docker kontajner. Docker image je *read-only template* (predloha/recept) s inštrukciami pre vytvorenie Docker kontajneru. Docker kontajner je bežiaca inštancia Docker image-u. Jednotlivé kontajnery sú izolované od iných a takisto aj od hosťujúceho zariadenia avšak dokážu aj medzi sebou komunikovať. Pre vytvorenie Docker image-u je potrebné definovať Dockerfile.

Komentár od [PL41]: https://docs.docker.com/get-started/overview/

Komentár od [PL42R41]: https://www.techtarget.com/searchitoperations/definition/Docker

Komentár od [PL43R41]: https://www.docker.com/resources/what-container/ image docker

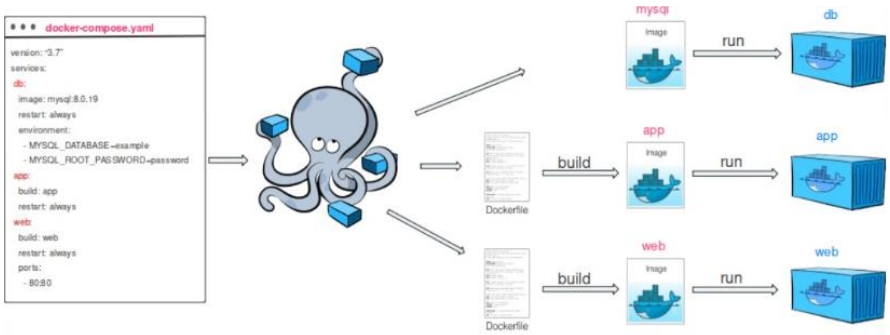


Obr. 26: Rozdiel medzi virtualizácia a kontajnerizáciou

2.4.1 Docker Compose

Docker Compose je nástroj, ktorý nám umožňuje definovať a zároveň spustiť viacero Docker kontajnerov naraz z jedného súboru. Docker Compose využíva YAML súbor na konfiguráciu jednotlivých aplikácii bežiacich v Docker kontajneroch.

Komentár od [PL44]: https://ostechnix.com/introduction-to-docker-compose/



Obr. 27: Príklad definovaného docker-compose súbor

Komentár od [PL45]: Docker compose image https://www.biaudelle.fr/docker-compose/

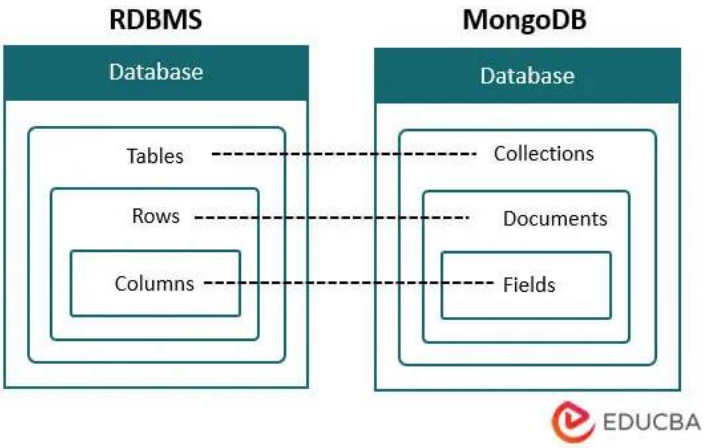
2.5 MySQL

MySQL je *open-source* relačná databáza, ktorá umožňuje používateľovi ukladať, manažovať a získavať štruktúrované dáta. MySQL je široko používaná pre rôzne aplikácie od malých projektov až po veľké webové aplikácie. Klient a server využívajú jazyk SQL pre komunikáciu v prostredí *RDBMS*.

2.6 MongoDB

MongoDB je *open-source, multiplatformová* nerelačná databáza navrhnutá pre jednoduchý vývoj aplikácií. Databáza je postavená na ukladanie veľkého množstva údajov v spolupráci s vysokým výkonom. MongoDB nepatrí medzi relačné databázy, je presným opakom databáz založených na SQL. Pri nerelačných databázach nemá tabuľka pevnú štruktúru, namiesto toho sa dáta ukladajú do kolekcií ako dokumenty založené na JSON a nevynucuje schémy. Nemá tabuľky, riadky ani stĺpce ako SQL databázy.

Komentár od [PL46]: Tento obr tu nedam



Obr. 28: Porovnanie štruktúry SQL databázy a MongoDB

3 Analytická časť

Pred samotným návrhom architektúry a rozhrania aplikácie, je potrebné spraviť analytickú časť, pri ktorej porovnáme rôzne technologické riešenia a architektúry a následne vyberieme najoptimálnejšie riešenia pre jednotlivé časti. Počas analytickej fázy vývoja sa veľká pozornosť venuje výberu správnych technologických nástrojov pre *backend* aj *frontend* aplikácie. Zahŕňa to hodnotenie rôznych programovacích jazykov, frameworkov databázových systémov na základe ich výkonu, efektivity, škálovateľnosti a kompatibility s požiadavkami projektu. Okrem samotného výkonu jednotlivých vrstiev aplikácie je potrebné aj plánovanie používateľského rozhrania s dorazom na jednoduché ovládanie s atraktívnym dizajnom.

3.1 Architektúry web aplikácii

Pri stanovení vhodnej architektúry pre webovú aplikáciu sme stáli pred kľúčovým rozhodnutím, ktoré ovplyvňuje nielen proces vývoja, ale aj prevádzku celého projektu. Existuje množstvo možností, z ktorých sme mohli vyberať, avšak každá z nich prináša so sebou svoje vlastné výhody a obmedzenia.

Medzi kľúčový bod pri výbere architektúry bolo rozdelenie aplikácie do viacerých vrstiev ako je v dnešnej dobe populárne čo umožňuje lepšiu údržbu, rozširovatelnosť a možnosť aktualizácie bez vplyvu na ostatné vrstvy. Ďalším kľúčovým bodom bol počet web stránok potrebných pre aplikáciu. Keďže pri interakcii s aplikáciou sa nemusíme presúvať na iné stránky ale iba meniť obsah webu zameriame sa skôr na *SPA (Single-page application).*

Na základe týchto kľúčových bodov sme sa rozhodli že bude najlepšie využiť framework pre front-end, ktorý nám vďaka VDOM značne uľahčí prácu pri zmene obsahu web stránky ako napríklad využívanie šablón pri *MVC (Model-View-Controller),* ktoré by generoval server a pri každej zmene obsahu by muselo dôjsť k obnoveniu stránky.

Kvôli týmto bodom našim požiadavkám najlepšie vyhovuje trojvrstvová architektúra alebo *full-stack* architektúra, ktorá pozostáva z *backend-u* a *frontend-u.*

3.2 Frontendové technológie

V tejto sekcií sa pozrieme na porovnania medzi jednotlivými *framework-ami*, nakoľko v dnešnej dobe s rastúcim počtom *framework-ov* je často ťažké rozhodnúť, ktorý by bol

najvhodnejší pre daný projekt. Pred samotným porovnaním si treba priblížiť prečo je lepšie využiť *framework* a VDOM na tvorbu klientskej časti ako sa spoľahnúť iba na HTML, CSS a *vanilla* JavaScript s použitím DOM.

3.2.1 Rozdiely medzi DOM a Virtual DOM

Najzásadnejší rozdiel je, že DOM reprezentuje aktuálnu HTML štruktúru webovej stránky, zatiaľ čo Virtual DOM je "ľahká" replika DOM.

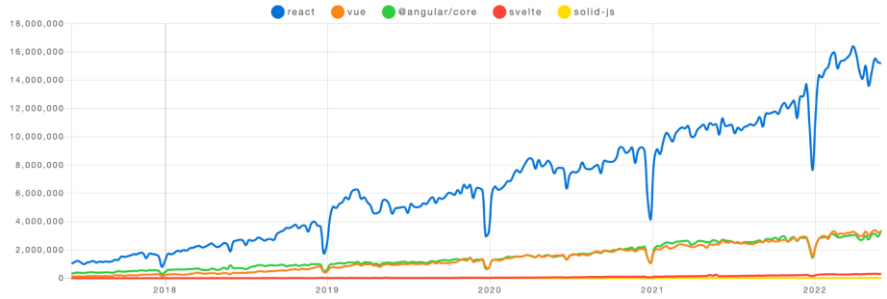
DOM	Virtual DOM
Reprezentuje webovú štruktúru HTML kódu.	Slúži ako zjednodušená reprezentácia DOM.
Je možná manipulácia so zobrazenými elementami.	Nie je možná manipulácia elementami zobrazeného na obrazovke.
Modifikácia v DOM spôsobí aktualizáciu celého DOM stromu.	Modifikácia aktualizuje iba relevantný uzol v strome.
Aktualizácia stránky je pomalá a neefektívna.	Proces aktualizácie je rýchly a efektívny.

Tab. 1: Porovnanie rozdielov medzi DOM a VDOM

3.2.2 Porovnanie frameworkov

Pri tvorbe moderných webových aplikácii zohráva výber frontendového frameworku kľúčovú úlohu vzhľadom na efektivitu a výkonnosť. V tejto časti sa pozrieme na najpopulárnejšie frontendové frameworky ktorými sú React.js, Angular a Vue.js.

Komentár od [PL47]: https://existek.com/blog/top-front-end-frameworks-2021/ obrazek



Obr. 29: Grafické porovnanie počtu stiahnutí jednotlivých frameworkov

React.js	Angular	Vue.js
Výkonovo efektívny vďaka použitiu VDOM, minimalizuje náročnosť pri vykresľovaní	Využíva štandardnú prácu s DOM čo môže mať vplyv na výkon	Výkonovo efektívny vďaka využitiu VDOM
Poskytuje väčšiu flexibilitu a voľnosť	Robustnejší so štandardizovanou štruktúrou	Flexibilný a jednoduchý na začatie projektu
Krivka učenia je jemná	Krivka učenia je strmá kvôli nutnosti použitia TypeScriptu	Krivka učenia je jemná

Tab. 2: Porovnanie frontendových frameworkov

Každý z daných frameworkov má svoje výhody aj nevýhody, avšak pri rozhodovaní sme brali fakt obľúbenosti Reactu medzi vývojármi aj jeho podpore pri tvorbe *SPA aplikácii*. Pri tvorbe UI sme následne ešte spracovali analýzu pre výber knižnice pre CSS, ktorá nám ma uľahčiť tvorbu komponentov a responzívneho dizajnu, pričom najlepšie obstáli *Bootstrap a Material UI*. Pri výbere z dvoch kandidátov sme sa rozhodli pre Bootstrap nakoľko je veľmi *user-friendly* a má skvelú podporu responzívneho dizajnu a modifikáciu komponentov v prípade potreby.

Komentár od [PL48]: https://www.spaceotechnologies.com/blog/front-end-frameworks-comparison/ tabulka

3.2.3 Bootstrap

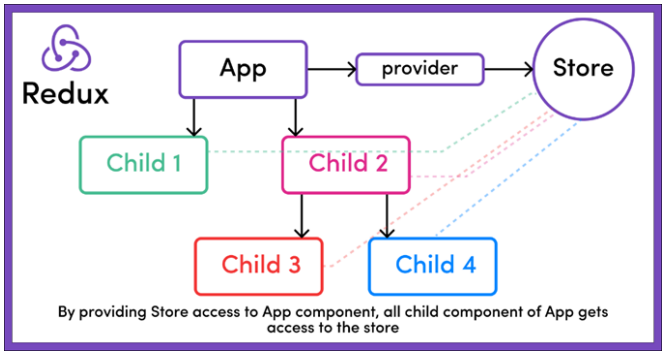
Bootstrap je *open-source* webový *framework/*knižnica, na vytváranie responzívnych web stránok a aplikácii. Bootstrap je veľkou kolekciou znovu použiteľného kódu napísaného v HTML, CSS a JavaScript-e. Umožňuje vývojárovi využiť viac času pri dizajne web stránky ako písania CSS kódu. Bootstrap patrí medzi poprednú CSS knižnicu pre responzívne HTML prvky a má veľkú podporu dizajnérov a vývojárov.

Komentár od [PL49]: https://www.hostinger.com/tutorials/what-is-bootstrap/

3.2.4 React Redux

Redux je open-source JavaScript knižnica na správu stavu aplikácii. Najčastejšie je využívaná s knižnicami ako je React.js alebo Angular pri vytváraní používateľských rozhraní. Redux sa využíva pri programovaní aplikácii kde sa dáta zdieľajú medzi jednotlivými komponentami, namiesto ukladania stavu v komponente sa využíva centrálne úložisko Store odkiaľ sa dáta zdieľajú do komponentov. Medzi komponenty patria:

- **Redux Store** – slúži ako centrálne úložisko stavov komponentov
- **Redux Reducer** – je funkcia, ktorá popisuje ako sa zmení stav komponentu a vráti novú hodnotu
- **Redux Actions** – je JavaScript objekt pozostávajúci z dvoch častí
  1. **Property** – popisuje typ akcie napr. používateľ ma zadať svoje meno
  2. **Payload** – obsahuje už konkrétne dáta



Obr. 30: Princíp zdieľania dát medzi komponentami pomocou React Redux-u

Komentár od [PL50]: https://www.technigo.io/explained/what-is-redux

Komentár od [PL51R50]: Obrazek https://www.freecodecamp.org/news/how-to-build-a-redux-powered-react-app/ 2

Komentár od [RP52]: Podľa mňa tu nevysvetľujete ako *„spôsob, akým sme umožnili zdieľanie "* toto je len oboznámenie s Redux. Podľa mňa to patrí do inej časti BP.

3.3 Backendové technológie

Dôležitou súčasťou aplikácie je samotná voľba technológie, ktorá bude zodpovedná za logiku. Pri výbere sme sa kládli dôraz na výber technológie, ktorá ma kompatibilitu s *frontend-ovými* technológiami ako je napr. React.js, ale takisto spĺňa požiadavky trojvrstvovej aplikácie. Na základe témy ktorou sa naša práca zaoberá OPC UA, sme sa rozhodli pre použitie .NET technológie, nakoľko .NET podporuje OPC UA a disponuje knižnicami pre prácu s touto technológiou. Pri porovnaní s inými sme dospeli k záveru, že ASP.NET Core je pre nás najlepšou voľbou.

ASP.NET Core	Java Spring
Webový *framework, open-source*	Java *framework, open-source*
Krivka učenia je mierna	Krivka učenia je strmá
Vývoj aplikácii pre podniky	Robustné, škálovateľné podnikové aplikácie
Veľmi efektívny a rýchly *framework*	Veľmi efektívny a rýchly *framework*
Podpora trojvrstvovej architektúry, možnosť využiť s *frontend-ovými frameworkami*	Podpora trojvrstvovej architektúry, možnosť využiť s *frontend-ovými frameworkami*

Tab. 3: Porovnanie bekhendových frameworkov ASP .NET a Java Spring

3.4 Databázové technológie

Technológie, ktoré pracujú s dátami sú dôležitou súčasťou moderných softvérových aplikácii, keďže bez nich by sme nevedeli uchovávať dáta a následne sa k nim vrátiť a pracovať s nimi.

3.4.1 Porovnanie databáz

Pri výbere typu databázy je dôležité sa zamyslieť, či budeme potrebovať štruktúrovanú databázu (SQL) alebo neštruktúrovanú (NoSQL).

SQL (MySQL)	NoSQL (MongoDB)
Relačný databázový systém	Distribuovaný databázový system
Tabuľky s fixnými riadkami a stĺpcami	Dokumenty JSON, grafy, tabuľky s riadkami a dynamickými stĺpcami, *key-pair* hodnoty
Pevná Štruktúra	Flexibilná štruktúra
Škálovanie je vertikálne, (zväčšenie s väčším serverom)	Škálovanie je horizontálne, rozšírenie naprieč komoditnými servermi

Tab. 4: Porovnanie vlastností SQL a NoSQL databázy

Na základe požiadaviek našej aplikácie, na uchovávanie dát pre OPC UA servery nám viac vyhovuje využiť SQL databázu, nakoľko naše dáta budú štruktúrované.

Komentár od [PL53]: https://www.mongodb.com/nosql-explained/nosql-vs-sql tabulecka

Eclipse BaSyx využíva na perzistenciu dát NoSQL databázu nakoľko dáta sú veľmi flexibilné. Vzhľadom na tieto fakty budeme pre perzistenciu dát pre komponenty využívať MongoDB, ktorú majú komponenty už nakonfigurovanú a umožňuje nám ju využívať v Dockeri.

Pri voľbe SQL databázy sme mali na výber z množstva možností, ale rozhodli sme sa pre MySQL vzhľadom na jej ľahkú konfiguráciu a takisto aj podporu v Docker-i, ktorý budeme využívať a spomenieme ho v ďalšej sekcii. K samotnej štruktúre dát pre obe databázy sa dostaneme v ďalších kapitolách pri návrhu databázového modelu.

3.5 Technológie pre infraštruktúru

Infraštruktúra aplikácie predstavuje základnú kostru a prostredie, ktoré sú neoddeliteľnou súčasťou chodu aplikácie. Pri tvorení infraštruktúry aplikácie sme dbali na dôraz aby bola aplikácia multiplatformová a škálovateľná. Tieto požiadavky nám umožňuje v oboch smeroch práve Docker.

Docker nám umožňuje zabalenie samotnej aplikácie, databáz a AAS komponentov do kontajnerov, ktoré sú nezávislé na operačnom systéme. Tento prístup umožňuje jednoduché nasadenie a spustenie na rôznych platformách (AWS, Azure) bez zbytočného ladenia a konfigurácie.

4 Návrh aplikácie

V danej časti je zobrazený návrh a štruktúra aplikácie, ktorá sa odvíja z teoretickej časti, použitých technológií a analýzy. Pri návrhu aplikácie je potrebná špecifikácia požiadaviek, aby boli splnená funkcionalita ad-hoc konektivity čo je hlavná požiadavka práce, pri klientskej časti je dôležité vytvoriť priateľské používateľské rozhranie, ktoré nebude náročné na obsluhu, a pri vývoji serverovej časti treba vytvoriť API komunikáciu a následne využiť Docker na zabalenie do kontajnerov. Pri riešení návrhu ad-hoc konektivity je potrebne využiť štandardizovaný digitálny opis cez AAS, na čo nám poslúžia komponenty od Eclipse BaSyx.

4.1 Špecifikácia požiadaviek

Špecifikácia požiadaviek je určená na presné definovanie a opísanie funkcionálnych a nefunkcionálnych požiadaviek, ktoré budú následne implementované v aplikácií. Cieľom špecifikácie požiadaviek je jasne popísať očakávané správanie aplikácie pri interakcii používateľa s jednotlivými komponentami.

4.1.1 Funkcionálne požiadavky

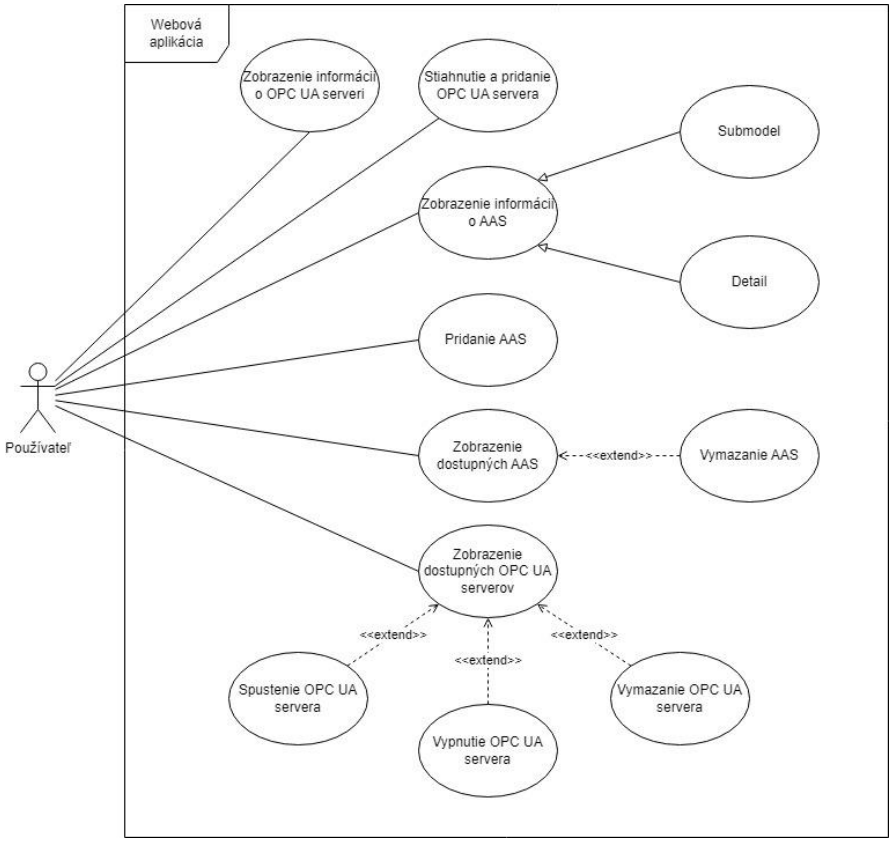
- Aplikácia umožňuje používateľovi zobraziť list s jednotlivými AAS, ktoré sú nahraté na servery
- Aplikácia umožňuje používateľom vymazať AAS z AAS Servera alebo pridať nove AAS do AAS Servera pomocou súboru s formátom .aasx
- Po kliknutí na inštanciu AAS v liste, aplikácia zobrazí používateľovi informácie (submodel a detail) o zvolenom AAS
- Aplikácia umožňuje používateľovi zobraziť list s jednotlivými OPC UA servermi, ktoré sú dostupne na serveri
- Aplikácia dovoľuje používateľovi manažovať OPC UA serveri a to vymazaním, spustením alebo vypnutím
- Používateľ ma možnosť stiahnuť a pridať OPC UA server pokiaľ ho AAS Submodel obsahuje
- Aplikácia umožňuje používateľovi zobraziť informácie o konkrétnom OPC UA serveri

Komentár od [PL54]: Tu si ujasnite čo je a na čo sa používa:
OPC UA - komunikacny protokol
OPC UA Server - softverovy komponent ktory vyuziva OPC UA protokol
AAS - Konceptualny model popisujuci strukturuu a vlastnosti aktiv
AASX - format suboru na ukladanie metadat AAS
AAS Server - Softvervoy komponentn ktory implementuje AAS koncept a poskytuje pristup k metadatam AAS

4.1.2 Nefunkcionálne požiadavky

- Systém musí zabezpečiť perzistenciu dát po vypnutí aplikácie pre OPC UA servery aj pre AAS
- Pri vypnutí aplikácie, systém musí zabezpečiť vypnutie všetkých bežiacich OPC UA serverov
- Pri vymazaní OPC UA servera, systém musí zabezpečiť že daný server je vypnutý, v prípade že je aktívny tak bude vypnutý

4.1.3 Diagram prípadov použitia

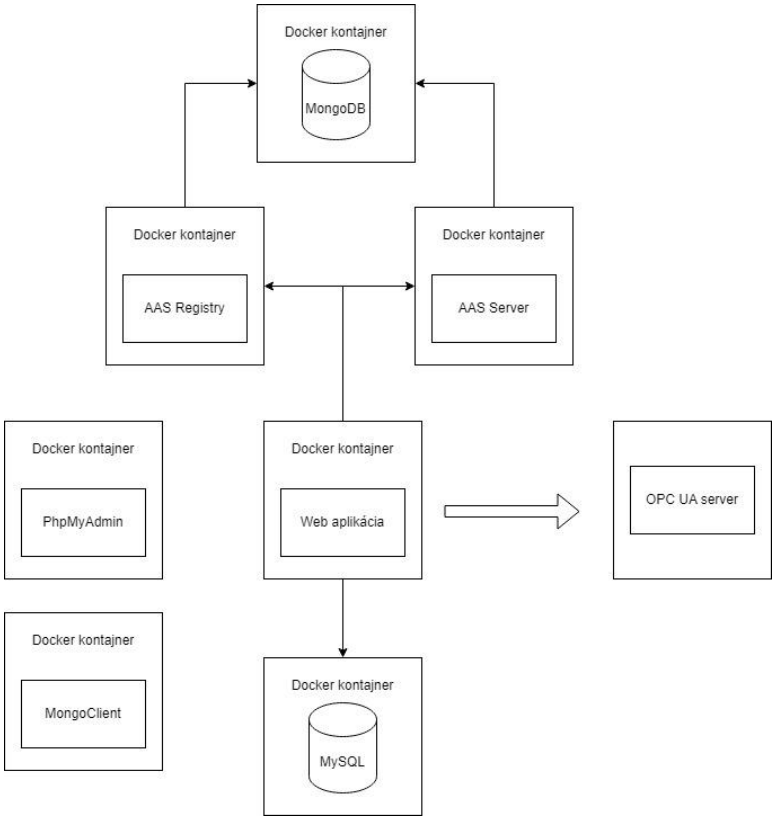


Obr. 31: Diagram prípadov použitia pre webovú aplikáciu

4.2 Architektúra aplikácie

Pri návrhu architektúry pre aplikáciu vychádzame z analýzy z predošlej kapitoly. Ako z analýz vyplýva naša aplikácia bude SPA, ktorá bude využívať trojvrstvovú architektúru. Vďaka tomu rozdelíme aplikáciu na dve časti a to frontend (prezentačnej vrstvy) a backend (aplikačná a dátová vrstva).

Aplikačnú vrstvu bude tvoriť REST API a aplikačná logika, ktorá bude mať za úlohu manažovanie OPC UA serverov (spustenie, vypnutie). Frontend bude teda komunikovať s backendovou REST API a využijeme ďalšie dve REST API, ktoré nám ponúkajú komponenty Eclipse BaSyx pre údaje o AAS. Architektúra bude obsahovať dve už spomenuté databázy. Jednotlivé komponenty zabalíme do Docker kontajnerov, cez ktoré budú vedieť komunikovať.



Obr. 32: Návrh architektúry pre webovú aplikáciu

Docker kontajner	Využitie Docker kontajnera
Web aplikácia	Frontend + Backend aplikácie
MySQL	Docker kontajner v ktorom beží MySQL databáza
MongoDB	Docker kontajner v ktorom beží MongoDB databáza
AAS Registry	Kontajner ktorý obsahuje list pre všetky AAS
AAS Server	Kontajner obsahujúci AAS Server kde sa nahrávajú AAS
MongoClient	Pomocný kontajner s UI pre prácu s MongoDB
PhpMyAdmin	Pomocný kontajner s UI pre prácu s MySQL

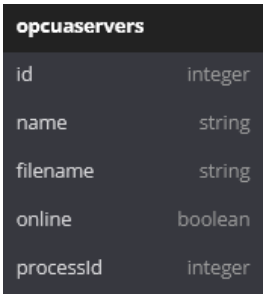
Tab. 5: Využitie jednotlivých Docker kontajnerov

Komentár od [PL55]: Tabulku zmenim

4.3 Štruktúra dátového modelu

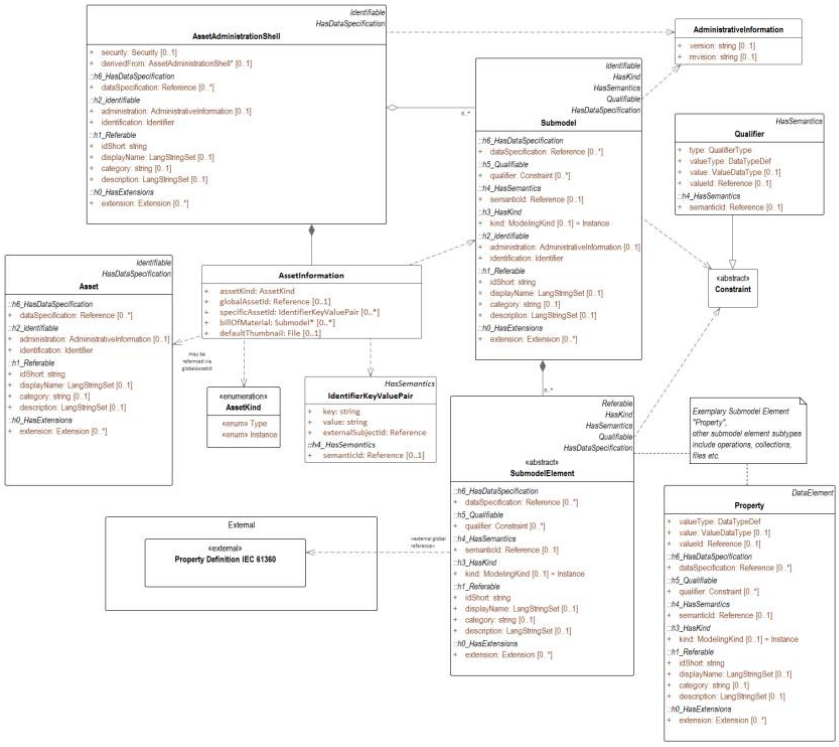
Štruktúra dátového modelu je hlavným prvkom pre organizáciu a správu dát v systéme. Poskytuje jednoznačné definovanie jednotlivých dátových modelov pre správne fungovanie systému. Pri definovaní dátového modelu vo webovej aplikácií sme zakladali na jednoduchosti pre nami definovaný dátový model.

Pre správu OPC UA serverov sme použili databázu MySQL kde sme definovali jeden dátový model pre OPC UA server.



Obr. 33: Návrh dátového modelu pre OPC UA server

Komponenty Eclispe BaSyx využívajú nerelačnú databázu MongoDB, ktorá nemá preddefinovanú schému, čo umožňuje flexibilitu pri práci s údajmi.



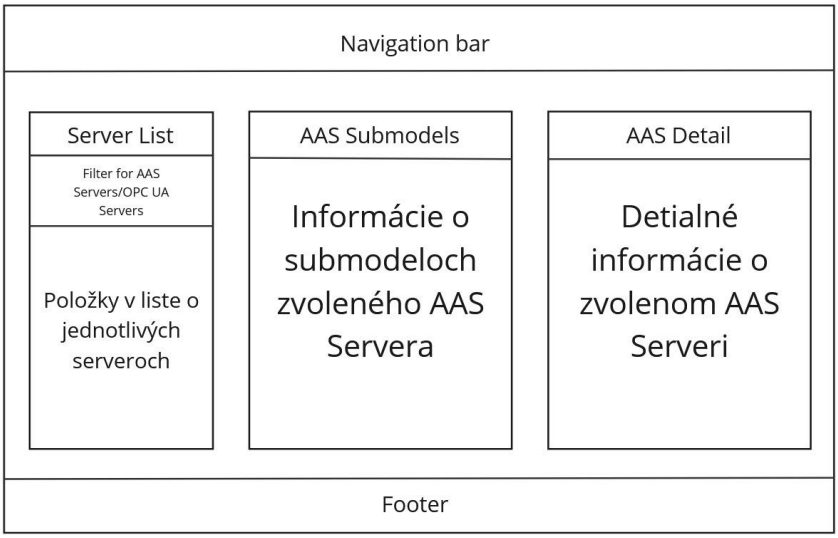
Obr. 34

4.4 Grafické rozhranie

Grafické rozhranie je kľúčovým prvkom pri tvorbe webovej aplikácie, s ktorým používateľ priamo interaguje, a pomocou ktorého, vie ľahko a efektívne ovládať aplikáciu. Pri návrhu GUI sme vychádzali zo 4 hlavných komponentov, s ktorými bude užívateľ interagovať.

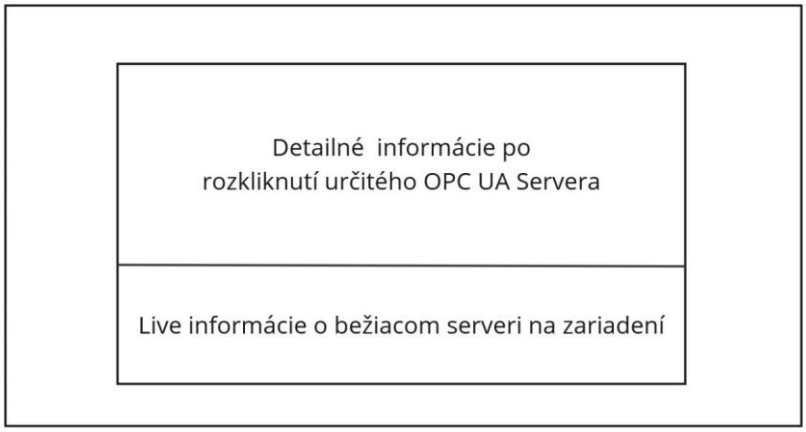
Ide o *List (zoznam)*, kde budú dostupné všetky *servery/digitálne aktíva. List* bude dynamický, čiže na základe zvoleného filtra budú zobrazené položky pre AAS alebo OPC UA servery. Ďalšie dva komponenty budú niesť informáciu o zvolenom AAS. Po užívateľovej interakcií s položkou v aktívnom zozname *AAS List* sa načítajú a zobrazia informácie o danom AAS v komponentoch *AAS Submodels* a *AAS Detail*.

Komentár od [PL56]: Toto asi prec



Obr. 35: Návrh GUI pre hlavnú stránku webovej aplikácie

Aby mal možnosť užívateľ zistiť detailnejšie informácie o jednotlivých OPC UA serveroch, po kliknutí na položku v zozname *OPC UA Servers* sa zobrazí okno, s detailnejšími informáciami o serveri.



Obr. 36: Návrh modálneho okna pre detailné zobrazenie informácií o OPC UA serveri

4.5 REST API

Táto sekcia popisuje rozhranie REST API, ktoré umožňuje komunikáciu medzi vrstvami aplikácie. REST API je navrhnuté tak, aby poskytovalo štandardný a jednoduchý spôsob manipulácie s dátami pomocou HTTP protokolu. Aplikácia využíva *backend* ako REST API, ktorá bude jednou časťou aplikačnej vrstvy. Okrem *backend-u* využijeme REST API, ktorú ponúkajú komponenty *Eclipse BaSyx*. Vďaka tomu budeme vedieť s jednotlivými komponentami komunikovať.

4.6 Ad-hoc konektivita

Medzi hlavnú požiadavku aplikácie je navrhnúť riešenie pre ad-hoc konektivitu. Ovládanie serverov či už spustenie alebo zastavenie vieme vyriešiť pomocou spustenia nového procesu s bežiacou službou *(service)* OPC UA servera alebo zabitia daného procesu. Pri tomto riešení by sa vytvorili *endpoint-y*, ktoré by slúžili na ovládanie serverov. Následne pri požiadavke na daný *endpoint* by služba *(service)* s potrebnou logikou spustila skript ktorý by obsahoval už potrebný kód.

5 Implementácia aplikácie

Táto časť práce sa zaoberá implementáciou jednotlivých krokov, ktoré boli uvedené pri návrhu aplikácie. Predstavíme si štruktúru aplikácie, vytvorenie dátového modelu, následne si rozoberieme backend aplikácie, ktorý pozostáva z dvoch častí. Po implementácií *backend-u* aplikácie sa zameriame na vývoj *frontend-u* či už dizajnu aplikácie alebo vysvetlenie funkcionalít na *frontend-e*. Po dokončení spomenutých častí si ukážeme kontajnerizáciu aplikácie.

5.1 Backend

Samotný backend aplikácie, ktorý predstavuje kľúčovú časť softvérového systému bude tvoriť REST API. Okrem REST API je dôležité na úvod spomenutie pomocných metód ktoré nám zabezpečia napríklad spustenie service ako nového procesu, alebo stiahnutie OPC UA serveru. V nasledujúcich častiach si podrobne prejdeme jednotlivé funkcionality serverovej časti webovej aplikácie.

5.1.1 REST API endpointy

Backend aplikácie, ktorý budeme implementovať pomocou ASP.NET Core bude slúžiť ako REST API. Webová aplikácia využíva jednu entitu, pre ktorú sú definované viaceré *endpointy*. Aplikácia využíva endpoint */api/opcuaserver*.

URL	Metóda	Status kódy
api/opcuaserver	GET	200, 500
api/opcuaserver	POST	200, 400, 500
api/opcuaserver/{id}/run-server	PUT	200, 404, 500
api/opcuaserver/{id}/stop-server	PUT	200, 404, 500
api/opcuaserver/{id}	DELETE	200, 400, 500
api/opcuaserver/{id}	GET	200, 404

Tab. 6: Definované endpointy pre prácu s OPC UA servermi

Pomocou *endpointov* dokáže vykonávať CRUD operácie nad jednotlivými servermi, pričom pri odpovedi na požiadavku od klienta vraciame dáta vo formáte JSON.

5.1.2 Logika aplikácie

Okrem bežiacej REST API na *backend-e* bolo potrebné vytvorenie logiky, ktorá bude riadiť na pozadí samotné manažovanie OPC UA serverov. Tá je implementovaná v *OPCUAService* a pomocné funkcie sa nachádzajú v priečinku */Helpers*.

5.1.3 Stiahnutie OPC UA Serveru

Aby sme dokázali mať stiahnutý server prítomný na serveri a nie u klienta, bolo potrebné sťahovanie riešiť cez backend. Na tento účel sú implementované pomocné triedy *DownloadHelper* a *UnzipHelper*. Po kliknutí na tlačidlo stiahnutia na *frontend-e* sa pošle požiadavka na *backend* aplikácie. Najprv zistíme či už daný súbor nie je stiahnutý a informácie o ňom nie sú zapísane v databáze, ak sa súbor nie je prítomný zavoláme metódu z *DownloadHelper* pomocou ktorej stiahneme .zip súbor na náš server do priečinku /Servers. Po úspešnom stiahnutí zavoláme druhú metódu z triedy *UnzipHelper*, ktorá nám umožňuje *rozipovanie* súboru. Pokiaľ počas niektorej z týchto operácii nastala chyba, vráti sa chybové upozornenie, ak nie vráti sa upozornenie o úspechu. Obe upozornenia sú zobrazené na *frontend-e.*

5.1.4 Spustenie/zastavenie OPC UA servera

Manažovanie stavu serverov riadi používateľ pomocou interakcie tlačidlami. Po kliknutí na tlačidlo, API spracuje požiadavku na základe *endpointov*, ktoré sme uviedli pri návrhu. Logika spustenia/zastavenia servera je založená na vytvorení/zabití procesu. Pre vytvorenie nového procesu sme implementovali triedu *ServerHelper*, ktorá spustí nový proces s OPC UA serverom. Následne do databázy uložíme číslo procesu a zmeníme stav. Pokiaľ chce používateľ server vypnúť, z databázy získame číslo procesu a proces zabijeme. V prípade chyby pri niektorej z operácii nastane výnimka, pričom výsledok je reprezentovaný daným upozornením na frontend-e.

5.1.5 Vymazanie servera

V prípade mazania servera sa okrem odstránenia daného záznamu o serveri z databázy, vymažeme aj priečinok so samotným serverom z priečinka */Servers*. Výsledok operácie je takisto zobrazený pomocou upozornenia.

5.1.6 ScriptExecutor

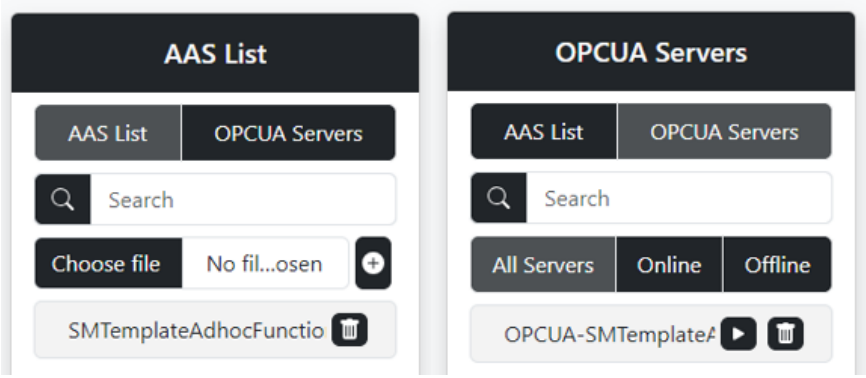
Aby používateľ vedel zistiť informácie o bežiacom OPC UA serveri, aplikácia má implementovanú pomocnú triedu *ScriptExecutor*. Pomocou tejto triedy aplikácia spustí na *backend-e* skript *(shell/batch)*, ktorý nám vráti informácie v JSON formáte. Vstupom do funkcie je ID procesu.

5.2 Frontend

Pri implementácii *frontend-u* prejdeme jednotlivými komponentami, a vysvetlíme funkcionalitu. Namiesto posielanie dát medzi jednotlivými komponentami pomocou *props*, implementujeme knižnicu *React Redux* ktorá umožňuje zdieľanie dát medzi jednotlivými komponentami.

5.2.1 List

Pre zobrazenie jednotlivých položiek týkajúcich sa OPC UA serverov alebo AAS bolo potrebné vytvoriť komponent zoznamu položiek. Ako bolo spomenuté pri návrhu GUI, tento komponent využijeme pre oba zoznamy. Používateľ bude mať možnosť vybrať si ktorý zoznam chce vidieť a bude mať k dispozícii filter na vyhľadávanie podľa názvu. Podľa výberu listu bude možné pre AAS (Obr. 37 vľavo) pridanie nového AAS do AAS serveru pomocou súboru alebo pre OPC UA servery (Obr. 37 vpravo) filter podľa statusu servera.



Obr. 37: UI List komponentov pre AAS a OPC UA servery

Po presunutí kurzorom na jednotlivé položky sa zobrazia používateľovi ikony na interagovanie (opäť podľa zvoleného typu zoznamu). Aplikácia umožňuje spustenie

zastavenie OPC UA servera alebo vymazanie položky pre oba typy. Po kliknutí na položku sa podľa typu zoznamu pre OPC UA server zobrazí modálne okno s detailom o serveri, alebo údaje tykajúce sa AAS, oba komponenty budú ukázane v tejto kapitole.

5.2.2 OPC UA Server detail

Aby mal používateľ k dispozícii informácie o konkrétnom OPC UA serveri, aplikácia umožňuje zobrazenie základných informácii o serveri. Po kliknutí na položku v zozname OPC UA serverov sa zobrazí modálne okno, kde sú vypísané údaje o serveri.

Obr. 38: Komponent OPC UA Server Detail pri bežiacom serveri a stlačení Execute Script

Z modálneho okna je takisto umožnené manažovanie serveru (štart, stop, vymazanie) ako aj pri interakcii v zozname. Okrem toho môže používateľ zistiť na akom porte beží OPC UA server . Po stlačí tlačidla *Execute Script* sa zobrazia informácie, ktoré sú získané pomocou skriptu *getServerDetails*.

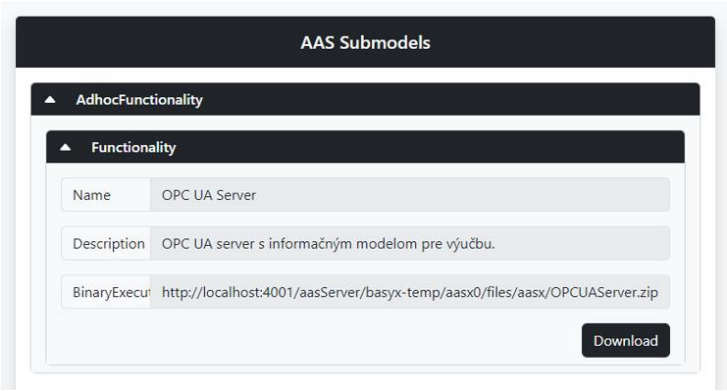
5.2.3 AAS Submodels

AAS Submodels je komponent, ktorý obsahuje detailné informácie o jednotlivých submodeloch. Informácie o jednotlivých submodeloch získavame vďaka API, ktorú nám poskytuje bežiaci *service aas-server* v Dockeri.

Endpoint	Využitie dát
/aasServer/shells/{aas_idShort} /aas/submodels/	Získanie údajov o jednotlivých submodelov daného AAS
/{submodels_endpoint}/{submodel_id}/ submodel/values	Detailné údaje o konkrétnom submodeli v danom AAS

Tab. 7: Využitie endpointov pre získanie dát o submodeloch

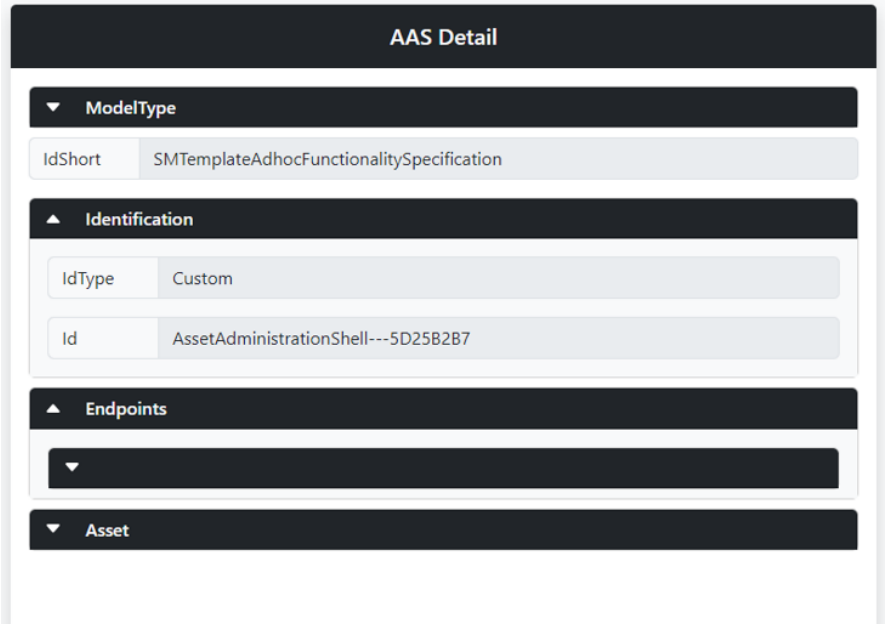
Ako bolo vysvetlené v teoretickej časti, sú tu definované technické funkcionality AAS .Pravé vďaka tejto vlastnosti využívame AAS ako digitálne aktívum, ktoré nám umožňuje zadefinovať do submodelu súbor, obsahujúci OPC UA server, ktorý následne vieme stiahnuť pomocou *requestu* na *backend* aplikácie. Po stiahnutí súboru je zobrazené upozornenie či bol súbor úspešne stiahnutý, alebo nastala chyba.



Obr. 39: AAS Submodels komponent s možnosťou stiahnutia OPC UA servera

5.2.4 AAS Detail

Posledným komponentom na *frontend-e* je AAS Detail, ktorý zobrazuje základné informácie o danom AAS, ktoré získavame *requestom* na *service aas-server* bežiaci v Dockeri. Medzi informáciami sú zobrazené informácie ako ID daného aktíva, *idShort*, *endpoint* aktíva vzhľadom na AAS Server a podobne. Ako aj pri predošlom komponente jednotlivé prvky sú zobrazené v stromovej štruktúre, ktoré dokáže používateľ rozbaliť/zabaliť.



Obr. 40: Komponent AAS Detail

5.3 Kontajnerizácia aplikácie

Po implementácii aplikácie bolo potrebné aplikácie kontajnerizovať ako bolo spomenuté pri architektúre a takisto pri analýze infraštruktúry. Pre kontajnerizáciu bolo potrebné vytvoriť *Dockerfiles* osobitne pre frontend aj backend aplikácie. Následne sme definovali jednotlivé *services* v *docker compose* súbore, ktorý nám umožňuje spustenie aplikácie bez potreby ďalšej konfigurácie. Jednotlivé kontajnery komunikujú medzi sebou pomocou siete ktorá je vytvorená Docker-om medzi jednotlivými kontajnermi. Aj keď sú kontajnery izolované, práve vďaka tejto vlastnosti dokážu medzi sebou komunikovať.

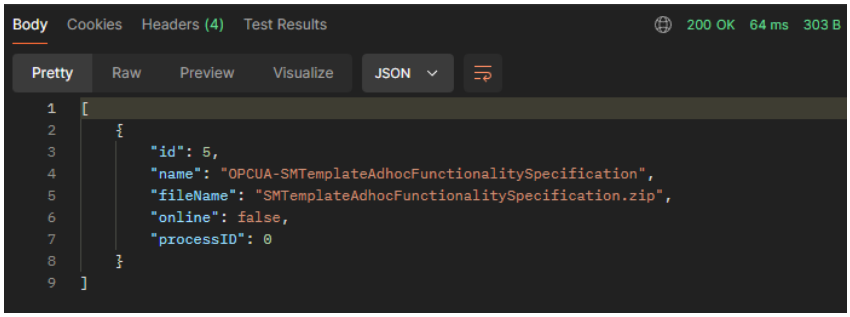
6 Testovanie

Testovanie aplikácie je proces overovania softvéru s cieľom zistiť, či aplikácie spĺňa požiadavky, ktoré boli vopred definované. Účelom testovania aplikácie je odhaliť možné chyby, problémy alebo nedostatky aplikácie pre uvedením do prevádzky. Pre testovanie aplikácie sme sa rozhodli pre vývojové testovanie a používateľské testovanie. Pri oboch spôsoboch testovania sme zvolili manuálne testovanie oproti automatizovanému testovaniu. Hlavným dôvodom pre voľbu manuálneho testovania bolo, že sme nemuseli implementovať množstvo testov (kódu) pre *frontend* a *backend*, ale mohli sme sa priamo zamerať na interakciu so softvérom.

6.1 Vývojové testovanie

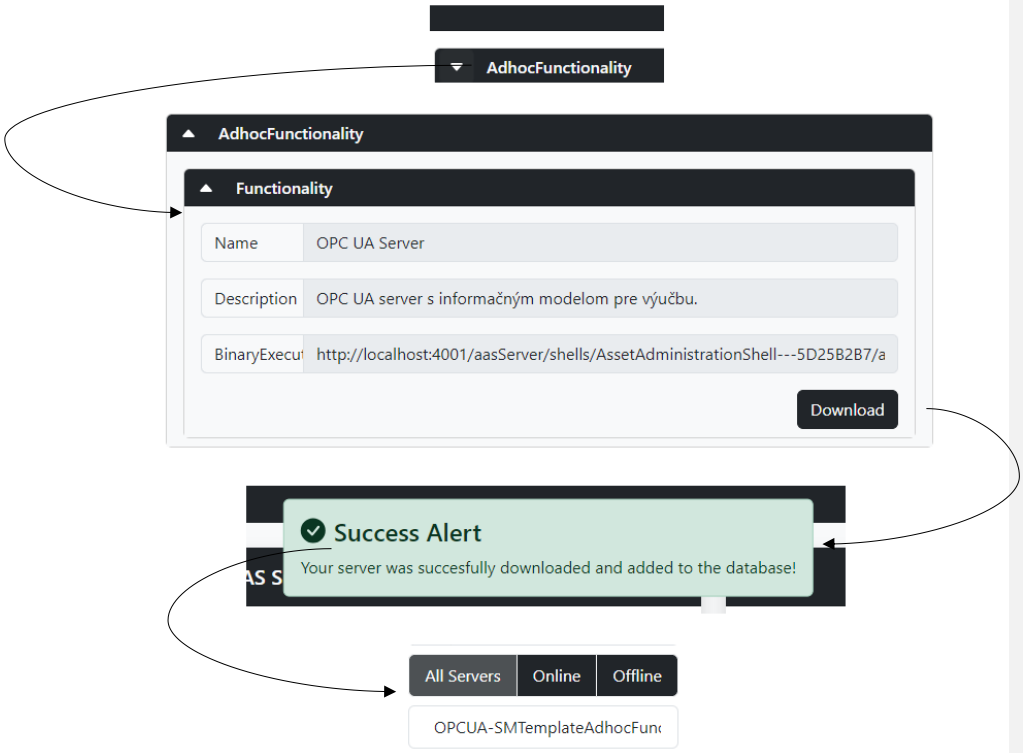
Pri vývojovom testovaní sme sa zamerali hlavne na testovanie komponentov, ktoré pozostávali z viacerých samostatných jednotiek. Testovanie bolo potrebné implementovať aj na backendovej ale aj frontendovej časti aplikácie.

Backend aplikácie nám tvorí REST API ako bolo uvedené v predošlých kapitolách, preto sme na testovanie jednotlivých endpointov využili *Postman*, pomocou ktorého sme mohli posielať requesty na jednotlivé endpointy a následne overiť ich funkčnosť.



Obr. 41: Príklad výstupu pri testovaní endpointu pomocou softvéru Postman

Pri testovaní *frontend-u* aplikácia sme sa zamerali na testovanie interakcie jednotlivých komponentov medzi sebou. Keďže frontend robí *request-y* na jednotlivé *services* po interakcii s tlačidlami, bolo potrebné overiť funkčnosť. Na tento účel sme využili manuálne testovanie s jednotlivými komponentami pri spustenej aplikácii. Tento prístup nám umožnil simulovať jednotlivé scenáre a zistiť ich správanie.



Obr. 42: Testovanie komponentu na frontend-e po jednotlivých interakciách

Príklad testovania si môžeme ukázať na interakcii AAS Submodel komponentu pri sťahovaní OPC UA serveru (Obr. 42). Pri zvolenom AAS postupne interagujeme s jednotlivými tlačidlami, rozbalením submodelu, a následne stiahnutím servera, pričom na konci očakávame notifikáciu o stave a následne zobrazenie v liste serverov.

6.2 Používateľské testovanie

Po dokončení implementácie aplikácie sme využili používateľské testovanie s cieľom použiteľnosť aplikácie a užívateľskú skúsenosť aplikácie. Tento prístup nám umožnil získať spätnú väzbu a identifikovať oblasti, ktoré je potrebné vylepšiť alebo upraviť. Pri testovaní používateľ interagoval s kompletnou aplikáciou pomocou akceptačných testov.

ID	1	**Názov**	Zobrazenie OPC UA serverov a AAS v zozname		
Prípad použitia			**Úroveň splnenia testu**	Musí – ~~Mal by~~ – ~~Mohol by~~	
Rozhranie		Systém			
Účel		Overenie funkčnosti systému pre zobrazenie dát			
Vstupné podmienky		Používateľ filtruje zoznam			
Výstupné podmienky		Zobrazenie položiek v zozname pre daný filter			
Krok	**Akcia**		**Očakávaná reakcia**		**Skutočná reakcia**
1.	Filter pre AAS		V zozname sú jednotlivé AAS		V zozname sú jednotlivé AAS
2.	Filter pre OPC UA		V zozname sú jednotlivé OPC UA servery		V zozname sú jednotlivé OPC UA servery
3.	Vyhľadávanie pomocou názvu		Systém zobrazí dáta, ktorých názov obsahuje zadaný reťazec		Systém zobrazí dáta, ktorých názov obsahuje zadaný reťazec
4.	Filter pre OPC UA vzhľadom na stav		V zozname sú OPC UA servery s daným stavom		V zozname sú OPC UA servery s daným stavom

ID	2	**Názov**	Stiahnutie OPC UA servera		
Prípad použitia			**Úroveň splnenia testu**	Musí – ~~Mal by~~ – ~~Mohol by~~	
Rozhranie		Systém, používateľ			
Účel		Overenie funkčnosti systému pre stiahnutie OPC UA servera			
Vstupné podmienky		Používateľ stlačí tlačidlo pre stiahnutie OPC UA servera			
Výstupné podmienky		Systém stiahne server pokiaľ ešte nie je o ňom záznam v databáze			
Krok	**Akcia**		**Očakávaná reakcia**		**Skutočná reakcia**
1.	Používateľ klikne na AAS v zozname		Načítanie dát v komponente AAS Submodels		Načítanie dát v komponente AAS Submodels
2.	Rozbalí submodel, ktorý obsahuje OPC UA server		Rozbalenie submodelu		Rozbalenie submodelu
3.	Používateľ klikne na tlačidlo “Download“		Odoslanie requestu na backend aplikácie		Odoslanie requestu na backend aplikácie
4.	V databáze ešte nie je záznam o serveri		Server je stiahnutý, rozipovaný, v databáze je vytvorený záznam, zobrazí sa notifikácia		Server je stiahnutý, rozipovaný, v databáze je vytvorený záznam, zobrazí sa notifikácia
5.	V databáze je záznam o serveri		Server nie je stiahnutý, zobrazí sa notifikácia		Server nie je stiahnutý, zobrazí sa notifikácia

Tab. 8: Akceptačný test pre stiahnutie OPC UA servera

ID	3	**Názov**	Manažovanie OPC UA serverov	
Prípad použitia			**Úroveň splnenia testu**	Musí – ~~Mal by~~ – ~~Mohol by~~
Rozhranie		Systém, používateľ		
Účel		Overenie funkčnosti manažovania OPC UA serverov		
Vstupné podmienky		Na serveri je stiahnutý OPC UA server		
Výstupné podmienky		Server zmenil svoj stav		
Krok	**Akcia**		**Očakávaná reakcia**	**Skutočná reakcia**
1.	Používateľ klikne na položku v liste serverov alebo prejde kurzorom na položku		Zobrazí sa OPC UA server detail alebo sa zobrazia tlačidlá na interakciu v liste	Zobrazí sa OPC UA server detail alebo sa zobrazia tlačidlá na interakciu v liste
2.	Používateľ klikne na tlačidlo na manažovanie serverov		Server zmení svoj stav, alebo je vymazaný, zobrazí sa notifikácia	Server zmení svoj stav, alebo je vymazaný, zobrazí sa notifikácia

Tab. 9: Akceptačný test pre manažovanie OPC UA serverov

ID	4	**Názov**	Spustenie skriptu pre informácie o bežiacom OPC UA serveri	
Prípad použitia			**Úroveň splnenia testu**	Musí – ~~Mal by~~ – ~~Mohol by~~
Rozhranie		Systém, používateľ		
Účel		Overenie zobrazenia informácia o bežiacom OPC UA serveri		
Vstupné podmienky		Server je online a používateľ má otvorený OPC UA server detail		
Výstupné podmienky		Server zobrazí informácie o bežiacom serveri		
Krok	**Akcia**		**Očakávaná reakcia**	**Skutočná reakcia**
1.	Používateľ klikne na tlačidlo Execute Script		Spustí sa skript a zobrazia sa informácie o bežiacom serveri	Spustí sa skript a zobrazia sa informácie o bežiacom serveri

Tab. 10: Akceptačný test pre zobrazenie informácii o bežiacom OPC UA serveri

ID	5	**Názov**	Vymazanie AAS	
Prípad použitia			**Úroveň splnenia testu**	Musí – ~~Mal by~~ – ~~Mohol by~~
Rozhranie		Systém, používateľ		
Účel		Overenie funkčnosti mazania AAS		
Vstupné podmienky		Na AAS serveri na nahratý AAS		
Výstupné podmienky		Vymazanie AAS		
Krok	**Akcia**		**Očakávaná reakcia**	**Skutočná reakcia**
1.	Používateľ prejde kurzorom na položku		Zobrazí sa tlačidlo na interakciu	Zobrazí sa tlačidlo na interakciu
2.	Používateľ klikne na tlačidlo vymazania		Zobrazí sa stav načítavania, AAS je odstránený	Zobrazí sa stav načítavania, AAS je odstránený

Tab. 11: Akceptačný test pre vymazanie AA

Záver

V závere je potrebné v stručnosti zhrnúť dosiahnuté výsledky vo vzťahu k stanoveným cieľom.

Zoznam použitej literatúry

[1] D. G., „What is MySQL: MySQL Explained for Beginners,“ 19 January 2024. [Online]. Available: https://www.hostinger.com/tutorials/what-is-mysql. [Cit. 2024 March 10].

[2] TutorialsTeacher, „What is MongoDB?,“ 2024. [Online]. Available: https://www.tutorialsteacher.com/mongodb/what-is-mongodb. [Cit. 2024 March 10].

[3] P. Pedamkar, „What is MongoDB?,“ 20 March 2023. [Online]. Available: https://www.educba.com/what-is-mongodb/. [Cit. 2024 March 10].