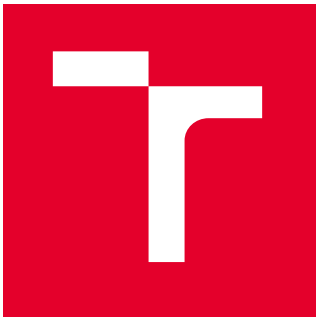


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

LOW-CODE / NO-CODE RULE BASED ENGINE NAD POZIČNÍMI DATY

LOW-CODE / NO-CODE RULE BASED ENGINE FOR POSITIONING DATA

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Natálie Vychodilová**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Radovan Juráň**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Studentka: Natálie Vychodilová ***ID:*** 230911

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Low-code / No-code rule based engine nad pozičními daty

POKYNY PRO VYPRACOVÁNÍ:

Moderní a přesné poziční systémy jsou nasazovány ve výrobě a logistice, kde se stávají součástí kritické infrastruktury daného podniku, podobně jako Wi-Fi. Při využívaní pozičních dat se nevyužívají přímo pozice, ale obvykle zóny, nad kterými se naprogramuje procesní logika daného výrobního nebo logistického procesu. Pomocí vizibility do daných procesů je možné zvyšovat efektivnost nebo bezpečnost v daném provozu.

Cílem práce je navrhnout uživatelsky přívětivý rule based engine, který eliminuje potřebu programovaní pro vybrané procesní problémy. Jedná se o pravidla, která propojují čas, polohu, počet a sekvence nad zónami nebo lokátory.

1. Nastudujte UWB poziční systém a jeho API. Obeznamte se s vybranými procesními problémy.
2. Proveďte analýzu dostupných frameworků vhodných pro low-code/no-code implementaci.
3. Navrhněte řešení pro realizaci rule based enginu včetně uživatelského rozhraní.
4. Navržené řešení implementujte.
5. Zhodnoťte dosažené výsledky a možnosti dalšího vývoje.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 26.5.2023

Vedoucí práce: Ing. Radovan Juráň
Konzultant: Jakub Lešinský

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Tato práce se zabývá návrhem a implementací softwarového systému určeného k eliminaci programování pro vybrané procesní problémy. Navrhovaný systém umožňuje uživateli vytvářet a spravovat pravidla nad pozičními daty. Práce se věnuje návrhu databáze, návrhu serverové a klientské části aplikace a uživatelského rozhraní vytvořeného pomocí no-code / low-code implementace. Návrh systému určeného pro vykonávání pravidel nad pozičními daty vychází ze znalostí technologie UWB a RTLS systémů.

KLÍČOVÁ SLOVA

Low-code / No-code implementace, Rule-based engine, Real-time locating system, poziční systémy, bezpečnost a efektivnost výrobních procesů

ABSTRACT

This thesis is focused on the design and implementation of a software system intended to eliminate programming needs for selected process problems. This system allows the user to create rules over positional data. It addresses design of the server and client parts of the application, design of the database and user interface. The design of the web application is based on knowledge of UWB technology and RTLS systems.

KEYWORDS

Low-code / No-code implementation, Rule-based engine, Real-time locating system, positioning systems, safety and efficiency of production processes

VYCHODILOVÁ, Natálie. *Low-code / No-code rule based engine nad pozičními daty*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 63 s. Bakalářská práce. Vedoucí práce: Ing. Radovan Juráň

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Natálie Vychodilová
VUT ID autora:	230911
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Low-code / No-code rule based engine nad pozičními daty

Prohlašuji, že svou závěrečnou práci jsem vypracovala samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autorka uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušila autorská práva třetích osob, zejména jsem nezasáhla nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědoma následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autorky*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Ráda bych poděkovala panu Jakubovi Lešinskému, za odborné vedení, čas věnovaný konzultacím a za podnětné návrhy k práci. Dále bych chtěla poděkovat vedoucímu bakalářské práce panu Ing. Radovanovi Juráňovi, za odborné vedení, konzultace a trpělivost.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

S postupným zvyšováním nároků na produktivitu a bezpečnost ve výrobních a logistických procesech se taktéž zvyšuje nárok na jejich automatizaci a využití informací. V současnosti existují přesné poziční systémy, které v reálném čase umožňují sledovat aktuální polohu jednotlivých objektů nebo osob v uzavřeném prostoru. Fyzické objekty jsou v těchto systémech reprezentovány jejich digitální kopií (tzv. digitálním dvojčetem), která umožňuje analyzovat chování objektů, a tato data mohou být následně i vizualizována. Systémy rovněž využívají virtuální zóny definované ve skutečném prostoru. Zóny jsou vymezeny systémem RTLS (Real-time Location System), který nad nimi programuje procesní logiku daného výrobního nebo logistického procesu.

Vzhledem k množství nasbíraných dat se tyto systémy stávají velmi složitými a komplexními. To vede k tomu, že jejich používání vyžaduje odborné znalosti a pro koncového uživatele se používání těchto systémů stává náročným. Řešením tohoto problému je vytvoření uživatelsky přívětivého rozhraní, ve kterém nad nasbíranými a zpracovanými pozičními daty bude uživatel vytvářet vlastní pravidla procesní logiky a spravovat tak konkrétní výrobní proces bez nutnosti znalosti logiky samotného pozičního systému nebo programování.

Cílem práce je tedy navrhnout komplexní systém s přívětivým uživatelským rozhraním, který koncovému uživateli umožní pracovat s nasbíranými daty a dovolí mu vytvářet pravidla nad procesy. Uživatelem vydefinovaná pravidla budou automaticky vykonávána systémem. Webové uživatelské rozhraní by mělo být přehledné a snadno ovladatelné, aby i osoba, která nemá programátorské znalosti, zvládla vytvořit jednoduchá pravidla.

První část této práce se zaobírá shrnutím procesních problémů a popisem důsledků, které vedou k využití systémů sledování pozic v reálném čase. Zaměřuje se na stručný popis bezdrátové technologie UWB (Ultra Wide-band) a na shrnutí vlastností RTLS systémů. Ve druhé části jsou popsány technologie nutné pro vytvoření všech částí navrhovaného systému. Třetí část práce se zaobírá popisem a porovnáním jednotlivých low-code/no-code frameworků a platforem, které budou při vytváření systému použity. Ve třetí části této práce je popsán návrh systému a požadavky na jeho funkci. Je zde definován datový model a jeho jednotlivé prvky. Taktéž je popsáno webové uživatelské rozhraní.

1 Lokalizace a lokalizační systémy

Řízení, plánování a sledování kritických činností v rámci výrobních a logistických procesů dnes využívá inovativních technologií, které podporují optimalizaci, zrychlování a zvyšování efektivity vykonávaných procesů. Mezi takové technologie se řadí i lokalizační systémy, jež umožňují společnostem shromažďovat potřebné informace o poloze sledovaných objektů (materiál, výrobky, výrobní a přepravní prostředky, nástroje, osoby,...) pro analýzu procesů a následnou optimalizaci jejich organizování, včetně jejich vizualizace. Získaná poziční data umožňují nejen monitorovat pohyb objektů v uzavřených prostorách, ale také nad nimi efektivně vykonávat logické procesy a vyhodnocovat význam polohy objektů v rámci jejich kontextu v procesech výroby či řízeného skladování [1, 2].

Systémy na základě získané aktuální poloze objektu vytváří digitální kopii monitorovaných procesů (tzv. digitální dvojče), která reprezentuje fyzické objekty virtuální formou. Digitální dvojče přebírá vlastnosti fyzického objektu a zároveň získává nové parametry, pomocí nichž lokalizační systém dokáže pomoci analyzovat chování jednotlivých objektů. Tyto objekty může vizualizovat, predikovat jejich chování a zefektivňovat procesy [3].

Lokalizační systémy nachází své uplatnění v různých oblastech, od sportu (sledování hráčů na hřišti), přes nemocnice (poloha lůžek, pacientů) až po továrny a sklady. Tato práce se zaměřuje převážně na využití lokalizačních systémů v oblasti průmyslu, výroby, skladování a inovativních řešení souvisejících s automatizací výrobních a logistických procesů.

1.1 Procesní problémy

S každým výrobním procesem přichází určitá rizika. Tato rizika nelze nikdy plně eliminovat, ale vydefinováním vhodných opatření (podmínek) mohou být tato rizika efektivněji předcházena a predikována. V současné době dochází k výrazné snaze o zautomatizování výrobních a logistických procesů. Tyto kroky pomáhají zkvalitnit, zrychlit a zvýšit bezpečnost výroby, a tím předcházet krizovým situacím, které v průběhu procesů mohou vzniknout, souhrnně nazývaným procesní problémy.

Procesním problémem rozumíme obecně neočekávanou událost, která ohrožuje bezpečnost osob, či bezpečnost chodu výrobního nebo logistického procesu. Tato událost je nepredikovatelná a je předcházena sledem dílčích neočekávaných událostí, zapříčiněných stroji nebo lidským faktorem.

1.1.1 Vybrané příklady procesních problémů

Procesní problémy ve výrobě

Ve výrobním procesu dochází k množství procesních problémů, které jsou nežádoucí a do značné míry ovlivňují chod továrny. Může se jednat o problémy způsobené lidským faktorem nebo stroji. Ať už je příčina procesního problému jakákoliv, nějakým způsobem dojde ke zpomalení, úplnému zastavení výrobního procesu nebo vzniku škody. Příčiny způsobené lidským faktorem mohou být události typu pád, srážka, použití poškozeného nářadí, nedodržení pracovního postupu nebo nedodržení použití nařízených bezpečnostních prvků. Tyto události mohou vést k úrazu pracovníka (obsluhy), k poškození nářadí nebo poničení stroje.

Přestože jsou tyto události nežádoucí a ohrožují bezpečnost zaměstnanců, mohou vzniknout i události, které ohrožují celý výrobní proces a jsou neméně závažné. Můžeme se setkat s procesními problémy, které způsobil stroj. Dojde-li ke kolizi více strojů nebo ke srážce stroje a zaměstnance, ohrožena je jak bezpečnost obsluhy, tak chod výrobního procesu. Příčinami vzniku procesních problémů nemusí být stroje nebo obsluha, mohou nimi být i jiné události.

Konkrétní příklady procesních problémů, které ve výrobním procesu mohou nastat, jsou uvedeny v následujících bodech:

- obsluha se ocitne v blízkosti vysokozdvižného vozíku. Přiblíží se příliš blízko a dojde ke kolizi,
- výrobek je v určitém kroku výroby delší dobu, než je předem definované,
- nářadí určené pro kompletaci výrobku není na svém místě,
- v prostoru se nezávisle na sobě pohybují dva vozíky vezoucí součástky. Jeden z nich se dostane do cesty druhému a srazí se.

Procesní problémy ve skladištích

Stejně jako ve výrobě dochází k problémům zpomalení procesu, dochází k nim i ve skladištích. Může docházet i k problémům, které nějakým způsobem ovlivňují logistiku (vyskladňování, expedice zboží).

Příkladem jsou skladiště, která jsou v dnešní době vystavěna tak, aby se obešla bez lidského faktoru na pracovišti. Většinu úkonů provádí stroje, které jsou naprogramované ke své činnosti. Mají jasně definované kroky, které vedou k urychlení a zefektivnění logistického procesu. Obsluha jen kontroluje proces a zasahuje do něj až ve chvíli, kdy dojde k neočekávané události.

Příklady procesních problémů, které mohou vzniknout ve skladištích, jsou uvedeny v následujících bodech:

- vzájemná kolize dvou samoobslužných strojů,

– počet kusů zboží ve skladišti je nižší než je požadováno pro expedici,
– výrobek umístěný ve skladišti není na svém místě,
– vyskladnění materiálu ze skladiště trvá déle, než je předem definováno.

Procesní problémy v administrativě

Ne vždy se za procesním problémem skrývá kolize nebo vykonávaná operace. Na problém se můžeme podívat i z jiného úhlu pohledu. Představme si administrativní budovu a zaměstnance, kteří v ní pracují. Ne všechny osoby mají umožněný přístup kamkoliv (pro každého zaměstnance jsou předem jasně vydefinované prostory, kam má vstup zakázán). Pokud se zaměstnanec pohybuje v místech, kde nemá umožněn přístup, stejně tak vzniká procesní problém. Tento problém jde přenést i do výše uvedených objektů. Jak ve skladišti, tak ve výrobní hale, mohou být vytvořeny zóny, do kterých určité objekty nemají přístup nebo naopak by se v dané zóně měly nějakou dobu vyskytovat.

1.1.2 Důsledky procesních problémů

Procesní problémy uvedené v předchozí podkapitole můžeme do určité míry předcházet a predikovat, ale nikdy riziko zcela neeliminujeme. Je potřeba si uvědomit, že jakýkoliv vzniklý problém stojí čas a peníze. V případě, že dojde procesnímu problému, provoz se zpomalí nebo dokonce zastaví. Časová prodleva, trvající než se provoz vrátí do stavu před touto událostí, je nežádoucí.

Stejně tak je nežádoucí samotný procesní problém, ať už se například jedná o srážku dvou strojů nebo o nedodržení času, který měl výrobek v konkrétním kroku výroby být. Pokud dojde ke kolizi, pravděpodobně dojde i k poničení stroje. Jeho oprava bude nějaký čas trvat a bude stát peníze. Při kolizi se taktéž může zničit součástka nebo hotový výrobek, který stroj převáží. Pokud výrobek zůstane v zóně déle, než je stanoveno, vzniká časový prostoj, který vede ke zpomalení (zpoždění) výrobního procesu a snížení množství vyrobených součástek. A taktéž může dojít ke znehodnocení výrobku.

Přirozeně přichází otázky, jak těmto událostem předcházet, jak zajistit nepřetržitý chod procesu, jak kontrolovat jednotlivé objekty nebo výrobní a logistické postupy. V současné době existují moderní poziční systémy, které jsou nasazovány ve výrobě a logistice a stávají se nedílnou součástí kritické infrastruktury daného podniku.

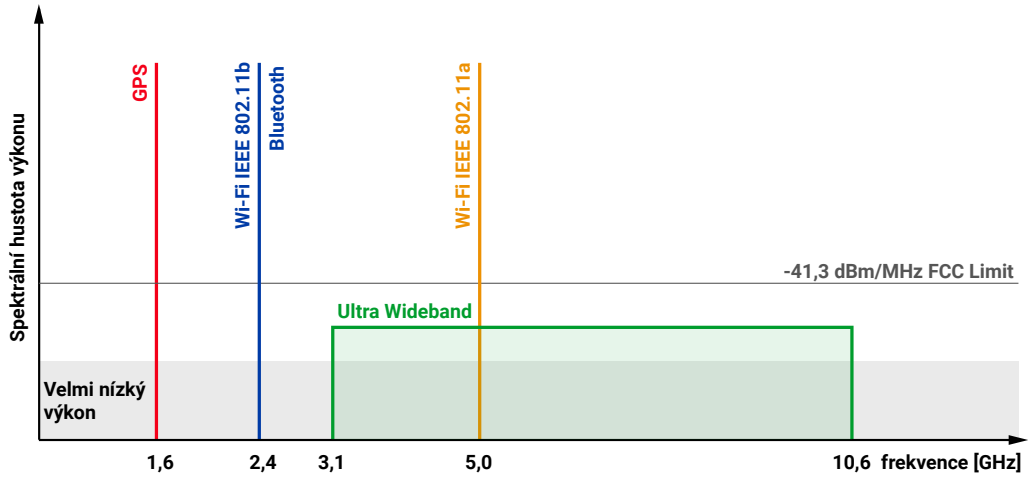
1.2 Technologie UWB

Počátky technologie UWB (Ultra-wideband) sahají do 20. století, kdy byla využívána původně pro vojenské aplikace. Na počátku 21. století bylo UWB standardizováno v rámci standardu IEEE 802.15.3 jako metoda vysokorychlostního přenosu dat. V roce 2007 bylo UWB specifikováno s standardu IEEE 802.15.4a jako rádiová technologie pro měření vzdálenosti (ranging). V roce 2020 bylo UWB standardizováno v IEE 802.15.4z, obohaceno zejména o prvky zabezpečení [4]. Přehledová tabulka 1.1 uvádí základní vlastnosti UWB.

Tab. 1.1: Základní vlastnosti UWB technologie

Frekvenční rozsah	3,1-10,6 GHz
Rychlost šíření	0,3 m/ns (rychlost světla)
Max. rychlost přenosu	480 Mbit/s
Lokalizační rozsah	10-200 m (v závislosti na umístění)
Přesnost lokalizace	0,1-0,5 m
Šířka pásma	až 500 MHz

Pro UWB je charakteristickým znakem velká šířka frekvenčního pásma (typicky kanály po 500 MHz). Komunikace je založena na vysílání časově krátkých pulsů přes široké spektrum a přijímač pulzní sekvenci převádí na data. Šířka pásma a krátké pulsy zajišťují nízkou latenci, robustnost a integritu komunikace, díky čemuž lze v reálném čase spolehlivě odhadnout polohu sledovaného objektu s přesností na centimetry [5]. Na obrázku 1.1 je srovnání spektrální hustoty výkonu UWB v porovnání s běžnými bezdrátovými technologiemi.



Obr. 1.1: Frekvenční pásma bezdrátových technologií [6]

1.2.1 Výhody technologie UWB

Jednou z výhod UWB technologie je nízká energetická spotřeba, která se stává klíčovým faktorem pro integraci do bezdrátových zařízení, jako jsou chytré telefony nebo lokalizační tagy [5]. Vyznačuje se nízkou složitostí, která vyplývá z povahy přenosu signálu. UWB nemoduluje a nedemoduluje složitou nosnou vlnu, takže nevyžaduje komponenty jako směšovače, filtry, zesilovače a lokální oscilátory. Technologie UWB má široké frekvenční pásmo, což jí dovoluje dosáhnout vysoké kapacity až do stovek Mbit/s se vzdálenostmi od 1 až do 10 metrů. Z pohledu lokalizace je přesnější než jiné bezdrátové technologie a má i vyšší odolnost vůči vnějšímu rušení.

1.2.2 Využití technologie UWB

V posledních letech došlo k rychlému vývoji technologií využívajících UWB signály. Technologie UWB může být použita pro lokalizační systémy v reálném čase, a to hlavně díky její přesnosti a odolnosti vůči rušení. Technologie umožňuje optimální sdílení dané šířky pásma mezi různými systémy a aplikacemi. Frekvenční pásmo je dostatečně široké, aby sdílení mezi systémy mohlo fungovat.

Automobilový průmysl

Většina vozidel je posledních několik let od výrobce vybavena asistenty, pomocníky nebo různými automatizacemi. Se zvyšující se úrovní těchto systémů se řidič ve voze stává čím dál méně důležitý a kontrolu nad automobilem přebírá autonomní řízení. Aby tyto systémy mohly svou funkci bezpečně vykonávat, potřebují spolehlivé a velmi rychlé senzory, které zajistí, že nedojde ke kolizím a událostem ohrožující život člověka.

Přesná a spolehlivá lokalizační technologie je u samo řiditelných vozů naprostou nutností. Dochází k vývoji technologií, jejichž cílem je například přimět automobil, aby sám zaparkoval na konkrétním místě vícepatrové garáže. Něco takového je možné pomocí technologie UWB, která je dostatečně přesná a na rozdíl od technologie GPS umožňuje fungovat v uzavřených prostorách [7].

V automobilovém průmyslu se s technologií UWB můžeme setkat i v oblasti lokalizace autoklíčů. Automobily jsou vybaveny anténou a díky ní dokáží přesně rozpoznat další zařízení vybavená UWB – jako je například digitální klíč nainstalovaný na chytrém telefonu. Automobil je schopný rozpoznat, kdy se klíč přiblíží dostatečně blízko, a automaticky odemknout vůz nebo rozsvítit světla [8].

Výroba a logistika

Vzhledem k tomu, že technologie UWB umožňuje rychle a spolehlivě přenášet velké množství dat, lze ji využít ke sledování objektů v továrnách, logistických skladech a dalších velkých průmyslových areálech. Sledovaným objektem může být součástka, vysokozdvižný vozík nebo hotový výrobek, který je uložen ve skladu a který pomocí čipu můžeme lokalizovat s přesností několika centimetrů.

Ve velkých továrnách jsou tisíce objektů různých typů (vozidla, nářadí, hotové výrobky), které se často přemisťují v prostoru. Personál stráví spoustu času hledáním konkrétních objektů a občas může dojít ke ztrátě nějakého z nich. Pomocí lokalizačního čipu lze rychle najít polohu konkrétního objektu, čímž se zkrátí doba, kterou by personál potřeboval k nalezení objektu.

Sledování zboží

Ve větších skladech se nachází tisíce položek a hledání každé z nich na základě obyčejných štítků může být časově velmi náročné. V případě použití lokalizačních čipů umístěných na konkrétním zboží nebo paletách se usnadní hledání jednotlivé položky a zrychlí se expediční procesy a organizace ve skladištích [9]. Operátoři mají díky sledování zboží lepší kontrolu nad svými aktivy. Mohou monitorovat zboží ale i stroje pohybující se ve skladištích (vysokozdvižné vozíky, ještěrky).

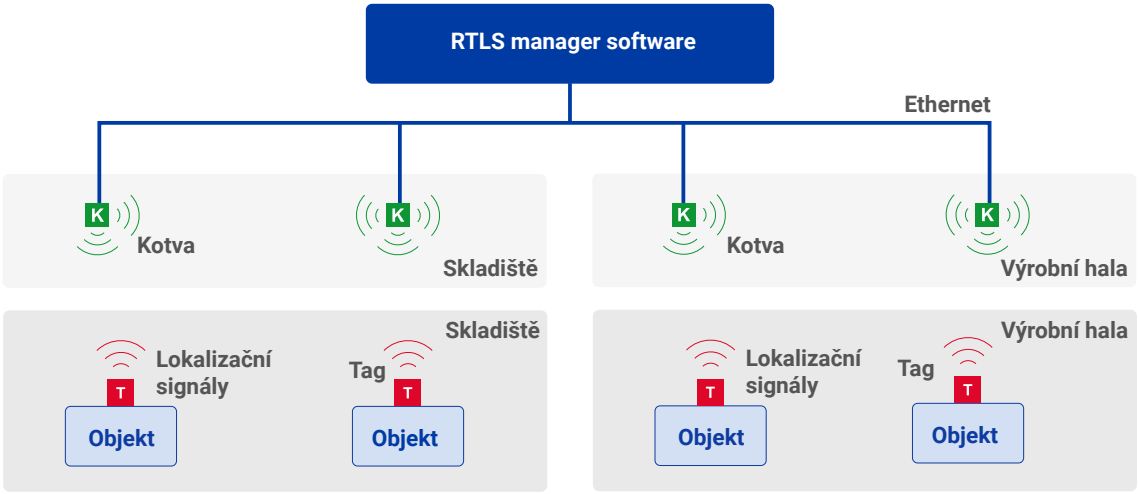
1.3 Real Time Location System

Real Time Location System (RTLS) je založen na konceptu automatického určování polohy objektů v rámci dané oblasti, kde je instalována jeho infrastruktura v reálném čase. Takovýto systém poskytuje uživatelům lepší kontrolu nad výrobním prostředím prostřednictvím nepřetržitého sledování různých objektů v reálném čase. Pomocí RTLS lze monitorovat různé položky a aktiva, jako jsou výrobky, nástroje, pracovníci a dopravní prostředky (automaticky naváděná vozidla, vysokozdvižné vozíky, dopravníky apod.) [10].

Získávání relevantních dat prostřednictvím RTLS je mnohem přesnější a časově méně náročné než užití jiných (manuálních) prostředků. Manuální nástroje, jako jsou stopky nebo video analýza, jsou časově náročné a nejsou příliš přesné. RTLS systémy poskytují aktuálnější, podrobnější a pravidelná data. Data získávaná v reálném čase zajišťují, že je systém dynamický, což pomáhá rychle a nepřetržitě vizualizovat výrobní a logistické procesy. RTLS výrazně zlepšují logistiku ve výrobě díky sledování trasy a pohybu vozidel a výrobků [11, 12].

Jak je znázorněno na obrázku 1.2, systém se standartě skládá ze tří prvků: **tagů**, **kotev** a **serveru**. Server a kotvy spolu většinou komunikují přes Ethernet nebo

Wi-Fi, zatímco kotva přijímá vysokofrekvenční (RF) signály z tagů. Každý tag má jedinečný identifikátor a může obsahovat i další informace o objektu, na kterém je umístěn, jako je sériové číslo, záznam o údržbě, stav baterie, teplota apod. Každý sledovaný objekt má na sobě umístěný tag, který je určený k jeho lokalizaci.



Obr. 1.2: Schéma RTLS systému

1.3.1 Případy použití RTLS

Server sbírá data o pozici jednotlivých sledovaných objektů. Na základě časových značek lze zjišťovat i rychlost objektů a jejich trajektorii. Data lze ukládat pro pozdější analýzu, např. pro optimalizaci trajektorií objektů [10]. Sledované objekty lze rozdělit do čtyř kategorií: **osoby**, **logistické prvky**, **materiál (výrobky)** a **nářadí (stroje)**. Pro každý typ objektu je nutné vydefinovat rozdílné specifické proměnné, které se mohou výrazně lišit. Například proměnná *pracovní stav* může být důležitá pro objekt typu osoba, ale nadbytečná pro objekt typu materiál. V tabulce 1.2 jsou uvedené příklady proměnných pro jednotlivé typy objektů.

Lidé/Osoby

V prostředí továren a skladišť se pohybují různé skupiny osob, jako jsou pracovníci, obsluha nebo manažeři, ale také návštěvníci nebo společnosti třetích stran (provádějící určité služby). Příkladem proměnných u osob kromě aktuální polohy, může být například pracovní stav (pracuje, přestávka, obědová pauza).

Tab. 1.2: Příklady proměnných pro jednotlivé typy objektů

Lidé/osoby	typ osoby (pracovník, manažer, návštěva), pracovní stav (pracuje, přestávka, oběhová pauza), zakázané zóny, stav baterie tagu
Logistické prvky	typ prvku (vozík, automaticky řízené vozidlo), provozní stav, stav baterie, oprávnění (pracovníci, kteří mohou stroj obsluhovat)
Materiál a výrobky	harmonogram, pracovní postup, celková doba pracovního postupu, prostor továrny, teplota výrobku (materiálu), čas ve výrobním procesu
Nářadí a stroje	provozní stav (používané/dostupné), oprávnění (pracovníci, kteří mohou výrobek používat)

Logistické prvky

Logistické prvky zahrnují veškeré vybavení, které je odpovědné za přepravní účely v továrnách a skladištích (vysokozdvižné vozíky, nákladní vozíky, automaticky řízená vozidla). Stavovou proměnnou může být provozní stav (plný/prázdný, v provozu/v poruše) nebo další informace jako je stav baterie.

Materiál a výrobky

Tyto objekty jsou v továrnách většinou spojeny s příkazy, které mají jasně definované pracovní postupy a harmonogramy. Je velmi důležité znát aktuální stav těchto objektů. Pracovní postupy jednotlivých výrobků jsou vydefinovány v RTLS. Systém může zároveň poskytovat další prostorové informace o přesné poloze, o příslušné zóně v prostorách továren nebo informace o teplotě nebo jiných vlastnostech produktu.

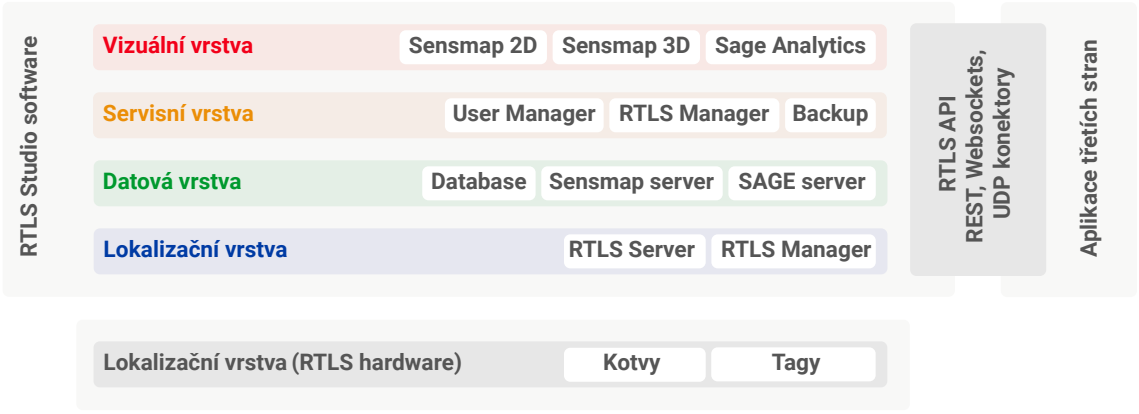
Nářadí a stroje

Nářadí a stroje jsou aktiva nezbytná pro provoz nebo podporu operací. Zatímco větší stroje jsou relativně stacionární, nářadí může být používáno v celé továrně, a proto je důležité znát aktuální polohu. Stavové proměnné spojené těchto objektů mohou souviset např. s provozním stavem (používáno/dostupné).

1.4 Sewio Indoor RTLS

Poziční systém společnosti Sewio používá technologii UWB a díky ní umožňuje sledování pohybu ve vnitřních prostorách v reálném čase. Systém zahrnuje hardwarové

komponenty a software [13]. Dohromady tyto části tvoří platformu, která nabízí plně škálovatelné řešení a umožňuje společnostem, které si řešení koupí, zvýšit produkci, snížit náklady a optimalizovat pracovní postupy ve výrobním nebo logistickém procesu.



Obr. 1.3: Schéma Sewio RTLS platformy

Architektura platformy je znázorněna na obrázku 1.3. Skládá se z pěti vrstev. Na hardwarové vrstvě se generují data. Tagy posílají signály a kotvy signály přijímají. Informace, které se odesílají z tagů pohybujících se v uzavřeném prostoru jsou přenášena pomocí technologie UWB. Tato nasbíraná data jsou posílána do lokalizační vrstvy, která má za úkol vypočítat polohu jednotlivých tagů a kotev. Datová vrstva je zodpovědná za ukládání dat o poloze a jejich reprezentaci pro další vizualizaci a analýzu. Servisní vrstva slouží pro řízení a optimalizaci výkonu systému. Analytická data a údaje o poloze poskytuje vizualizační vrstva.

Sewio RTLS Studio poskytuje vlastní API (Application Programming Interface), pomocí kterého lze Studio software propojit s aplikacemi třetích stran, kterými mohou být například interní systémy společností nebo vlastní aplikace zajišťující vizualizaci nasbíraných dat.

1.4.1 RTLS Hardware

Platforma Sewio UWB RTLS se skládá ze dvou typů hardwaru: přijímačů signálu (**kotev**) a vysílačů signálu (**tagů**). Tagy jsou malá elektronická zařízení umístěna na sledovaném objektu, kterým může být například paleta nebo zaměstnanec. Tagy v krátkých časových úsecích vysílají data, která jsou přijímána kotvami a následně jsou předána lokalizační vrstvě, která vypočítá jejich polohu v prostoru [14].

1.4.2 RTLS Software

RTLS Studio zajišťuje plnou kontrolu nad sledováním objektů ve vnitřních prostorách. Obsahuje několik funkcí, které umožňují konfigurovat nastavení systému, vizualizovat nasbíraná data nebo analyzovat trajektorie sledovaných objektů. Je dostupný pro 3 platformy: Virtual Box, Ubuntu Server a Docker.

RTLS Manager

Tento prvek RTLS Studia poskytuje jednoduchou, přístupnou a uživatelsky přívětivou správu lokalizačního systému v reálném čase. Zajišťuje inicializaci systému a umožňuje nastavení kotev a tagů. Při inicializaci těchto hardwarových komponent se vytvoří komunikační tabulka UWB, která poskytuje přehled o kvalitě spojení mezi všemi kotvami. RTLS Manager zobrazuje přehled o stavu všech kotev, tagů a serverů. Umožňuje uživatelům inicializovat, skenovat a znovu synchronizovat systém. Poskytuje plnou kontrolu nad nastavením a parametry systému. Jakmile je vydána nová verze firmwaru vzdáleně dojde k aktualizaci celého systému.

RTLS Planner

RTLS Planner zefektivňuje a optimalizuje instalaci infrastruktury RTLS. Systém automaticky navrhuje nejvhodnější umístění kotev uvnitř výrobních hal nebo skladišť, vybírá vhodný typ kotvy (směrová, všesměrová), kontroluje, zdali je pokrytý celý prostor nebo jestli není v prostoru umístěna překážka, přes kterou signál vysílaný z tagu neprojde. Podporuje návrh ve 2D a 3D pohledu.

RTLS Monitor

Zobrazuje úplný přehled o výkonu systému a aktuálním stavu sítě. Poskytuje přehled aplikací a uživatelův reálném čase, kontroluje stav systému, jeho procesy a využití.

Sensmap

Sensmap vizualizuje pohyb objektů a umístění kotev uvnitř uzavřených prostor v reálném čase. Umožňuje přidávat a upravovat budovy, přidávat budovám patra a automaticky rozmisťovat kotvy. V plánech budov lze přidávat různé zóny, cesty, stěny nebo východy. V reálném čase lze sledovat pohyb jednotlivých objektů (tagů umístěných na osobách nebo výrobcích) uvnitř budovy.

SAGE Analytics

Nástroj SAGE Analytics poskytuje možnost zobrazovat údaje o poloze pomocí grafických metrik. Může analyzovat výsledky sledování prostřednictvím map trajektorií objektů, teplotních map, diagramů nebo zónových dat.

Tento nástroj umožňuje přiřadit tagy ke kategoriím a vizualizovat výsledek jen pro konkretní z nich. Tato analýza umožňuje zvýšit produkci, bezpečnost a optimalizovat pracovní postupy ve výrobních nebo logistických procesech.

Metrikami nástroje SAGE Analytics jsou:

- Výskyt v zóně – v tabulce jsou zobrazeny tagy, které se v zóně vyskytly,
- Měření vzdálenosti – jakou vzdálenost ujel vozík za určitou dobu,
- Zónová mapa – ukazuje, jak dlouhou dobu strávil tag v konkrétní zóně,
- Grafy návštěvnosti – pro vybranou zónu ukazuje, zda je vybraný tag přítomen nebo není.

1.4.3 Sewio API

RTLS Studio poskytuje komplexní API (Application Programming Interface), které se používá k poskytování dat o poloze dat senzorů a dalších metadat souvisejících s vnitřní polohou, jako jsou informace o budově, půdorysy a zóny [15]. RTLS Studio podporuje tři způsoby přístupu k datům prostřednictvím API:

- **REST** – Poskytuje všechna dostupná data a údaje o poloze. Lze jej použít pro přístup ke statickým a historickým datům, jako jsou informace o kotvách, budovách, půdorysech atd.
- **Websockets** – Tento přístup se používá v aplikacích, kde je vyžadováno získávání polohy v reálném čase. Websockets poskytují údaje pouze o poloze.
- **UDP** – Používá se tam, kde jsou umístěny tagy s velmi rychlými intervaly obnovy. UDP stream poskytuje pouze údaje o poloze.

2 Vývoj webových aplikací

Při vytváření rozsáhlých a komplexních webových aplikací je důležité rozlišit a vhodně vydefinovat části systému. Každá část systému má svou nezaměnitelnou funkčnost a vlastnosti, a zároveň každá část si vzájemně vyměňuje data a nepřetržitě komunikuje s částí sousední. Rozdělením komplexních aplikací do menších celků rozkládáme problematiku a eliminujeme chybnou implementaci a zbytečnou nepřehlednost finální aplikace.

Obecně se setkáváme s rozdělením webových aplikací do dvou základních částí: **Frontend** a **Backend**. Zatímco frontend je klientská část aplikace a jeho funkcí je zpracovávat a vizuálně zobrazovat data koncovému uživateli, backend je serverová část systému, která implementuje logiku a zpracovává přijatá data [16].

2.1 Frontend – klientská část aplikace

Frontend je klientská část aplikace, která zajišťuje komunikaci s uživatelem. Je zodpovědná za zobrazování obsahu aplikace v uživatelském rozhraní, zajišťuje události vytvořené uživatelem, jako je například stisk tlačítka, žádost odeslání formuláře nebo navigování mezi jednotlivými stránkami aplikace. Klientská část aplikace komunikuje se serverem, na nějž se průběžně dotazuje a přebírá z něj zpracovaná data, která následně zobrazuje v uživatelském rozhraní.

Hlavním cílem je zajistit odezvu a výkon. Aplikace musí být navržena tak, aby byla responzivní (aby se správně zobrazovala na všech typech zařízení). Dalšími nároky kladené na klientskou část aplikace je ošetření všech vstupů uživatele a zajištění správné funkčnosti každého elementu a ošetření všech nestandardních chování.

HTML

HTML (Hypertext Markup Language) je značkovací jazyk používaný pro tvorbu webových stránek, který vznikl na začátku 90. let dvacátého století a jeho vývoj stále pokračuje. V současnosti webové aplikace využívají HTML5 [17].

Webové stránky vytvořené pomocí HTML jsou vzájemně propojeny hypertextovými odkazy. Jazyk má jasně definovanou strukturu dokumentu a je charakterizován množinou značek (párových i nepárových). Každá značka má vydefinované parametry a vlastnosti. Vydefinované vlastnosti jednotlivých značek se přenáší na text, který tyto značky obklopují (v případě párových). Například řetězec `<h1>Nadpis</h1>` symbolizuje nadpis úrovně jedna. Pomocí těchto značek, lze vytvářet celé webové stránky obsahující odlišné elementy.

CSS

CSS (Cascading Style Sheets) je jazyk, který popisuje způsob zobrazení elementů na webových stránkách. Jazyk odděluje vzhled dokumentu od jeho struktury a obsahu, což usnadňuje úpravu stylů a aktualizaci vzhledu webových stránek. Pomocí CSS lze měnit rozložení objektů, styl písma, barvy a další prvky v HTML dokumentu.

Každý typ HTML elementu může mít stejný styl. Je možné i pro jeden konkretní element vydefinovat specifický styl prostřednictvím identifikátorů. Více elementů obsahujících stejnou třídu přebírá styly této třídy. Jazyk CSS podporuje i vytváření stylů na základě vyvolané události (například přejetí myší po tlačítku nebo kliknutí na odkaz).

JavaScript

JavaScript je programovací jazyk, jehož syntaxe vychází z jazyků Java a C++. Jazyk je založený na událostech, a proto je vhodný pro vytváření dynamických a interaktivních webových stránek. Ke statické webové stránce přidává dynamickou funkčnost jako jsou například animace, validace formulářů a interakce s uživatelem aplikace.

Jazyk může být spouštěn na klientské části aplikace, což znamená, že je kód vykonáván v prohlížeči uživatele. Tyto skripty mohou být přidány do HTML dokumentu nebo volány ze souboru. JavaScript může být implementován i na serverové části aplikace (využíváno například pro aplikace pracující v reálném čase).

2.1.1 Frameworky

Frameworky jsou soubory předem vytvořeného kódu, které mohou vývojáři používat k efektivnějšímu vytváření webových aplikací. Poskytují sadu nástrojů, které vývojářům pomáhají vytvářet konzistentní a přehledný kód.

Mezi výhody používání frameworků patří:

- **Rychlost vývoje** – frameworky poskytují předem vytvořené komponenty a šablony, které lze poměrně rychle a snadno přizpůsobit a implementovat do vlastní vytvářené aplikace,
- **Jednotnost** – frameworky mají jednotnou strukturu, což vývojářům usnadňuje práci na stejném projektu,
- **Škálovatelnost** – frameworky jsou navrženy tak, aby pomocí nich mohly být vytvářené komplexní a rozsáhlé aplikace. Zároveň počítají se změnou aplikace v průběhu její existence (zajišťují přechod na novější verze a usnadňují údržbu,

- **Podpora a dokumentace** – komunita vývojářů zajišťuje aktualizace, poskytuje zdroje a kvalitně zpracované dokumentace.

Angular CLI

Angular je framework určený pro vytváření klientských částí webových aplikací. Je vyvíjen a spravován společností Google a aktuálně pracuje ve verzi 15.

Jedná se o framework poskytující širokou škálu nástrojů a funkcí, které dovolují vytvářet komplexní webové aplikace. Je založen na architektuře pracující s komponenty. Aplikace je rozdělena na modulární komponenty, které lze snadno a opakovaně používat v celé aplikaci. Komponenty mohou vzájemně přebírat své vlastnosti a dědit z jiných komponent v aplikaci. Angular poskytuje obousměrnou vazbu dat – změny dat se automaticky promítají do uživatelského rozhraní a naopak.

Framework má robustní systém směrování, který dovoluje relativně snadno vytvářet komplexní navigační struktury (směrování stránek webových aplikací a směrování mezi komponenty). Je vhodný pro vytváření komplexních podnikových aplikacích, které pracují na různých typech koncových zařízení [18].

2.2 Backend – serverová část aplikace

Backend je serverová část webové aplikace, která má za úkol zpracovávat a vyhodnocovat logiku aplikace. Jejím účelem je spravovat data přijímaná od uživatele nebo z databáze, zpracovávat data přijatá od uživatele, poskytovat dynamický obsah klientské části aplikace. Serverová část aplikace poskytuje logiku a funkce, jež umožňují správné a efektivní fungování frontendu.

Backend může být rozdělen do částí, z nichž má každá svou funkčnost: **server** a **databáze**. Zatímco server je zodpovědný za zpracování dat (žádostí z klientské části) a odesílání odpovědí zpět ke klientovi, databáze zajišťuje ukládání dat a vygenerování odpovědí na základě předaného dotazu.

Django framework

Stejně jako pro klientské části aplikace existují frameworky, které vývojářům usnadňují a zefektivňují práci, existují tyto nástroje i pro serverovou část aplikací. Příkladem takového frameworku je Django.

Jedná se o bezplatný webový framework s otevřeným zdrojovým kódem napsaný v jazyce Python. Umožňuje vytvářet komplexní aplikace a zajišťuje rychlé propojení s databází. Obsahuje objektově-relační mapování, které dovoluje komunikovat s databází pomocí kódu Pythonu namísto dotazů. Díky tomuto je komunikace s databází bezpečnější a snadnější (není nutné znát SQL (Structured Query Language)

dotazy). Django obsahuje administrátorské rozhraní, ve kterém lze spravovat záznamy a databázové tabulky. Poskytuje sytém zajišťující URL (Uniform Resource Locator) směrování aplikace a obsahuje engine, který odděluje prezentační logiku od obchodní logiky. To usnadňuje údržbu a aktualizaci uživatelského rozhraní aplikace.

2.3 Application Programming Interface

Každá aplikace má jasně vydefinované části, které spolu prostřednictvím rozhraní komunikují. Rozhraní API je sada protokolů a nástrojů, která umožňuje různým aplikacím vzájemně komunikovat. Při vývoji webových aplikací se rozhraní API používá k zajištění komunikace mezi klientskou a serverovou částí aplikace. Klientská část aplikace si prostřednictvím API vyžádá data nebo služby serverové části aplikace a ta odpoví požadovanými informacemi [19].

API funguje jako prostředník mezi klientskou a serverovou částí aplikace, poskytuje standardizovaný způsob vzájemné komunikace obou komponent. Oddělením částí aplikace je zajištěna větší škálovatelnost a díky implementaci API jsou jasně vydefinovaná rozhraní mezi jednotlivými částmi aplikace.

2.3.1 REST

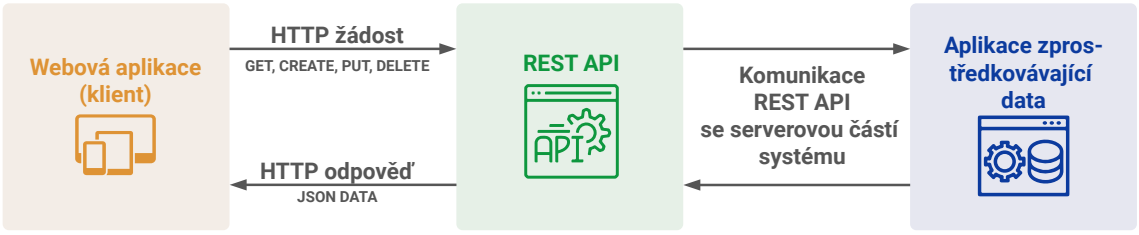
REST (Representational State Transfer) je typ rozhraní API používaného převážně v oblasti webových aplikací, který dovoluje různým systémům mezi sebou vzájemně komunikovat. V oblasti webových aplikací je používán z důvodu jednoduchosti, škálovatelnosti a kompatibilitě s HTTP (Hypertext Transfer Protocol).

Koncept principu REST API je založen na komunikaci mezi klientem a serverem. Server přijímá požadavky ze strany klienta, avšak neukládá žádné informace o klientovi ani příchozí data. Požadavek klienta musí obsahovat všechny potřebné informace k tomu, aby server rozuměl jeho žádosti a mohl ji zpracovat. Tento princip umožňuje serverové straně zpracovávat více klientských požadavků současně. Každý prostředek (dotazovaný objekt, data nebo služba) je jednoznačně identifikován pomocí své adresy URL a klienti mohou s těmito prostředky komunikovat pomocí standardních metod HTTP.

Princip komunikace

Princip komunikace je zobrazen na obrázku 2.1. Klient (webová aplikace) zahájí komunikaci odesláním požadavku HTTP na server. Odeslaný požadavek obsahuje: **informace o požadovaném prostředku** (URL), **hlavičku** a **parametry**. Server

tento klientem odeslaný požadavek přijme, interpretuje poskytnuté informace a provede příslušnou operaci (například se dotáže databáze). Následně vytvoří odpověď HTTP, která obsahuje požadovaná data nebo stavovou zprávu – standartě ve formátu *.json* (JavaScript Object Notation), a odešle ji. Klient obdrží odpověď HTTP a zpracuje získané informace z těla přijaté JSON zprávy.



Obr. 2.1: REST API model

Existují čtyři metody HTTP:

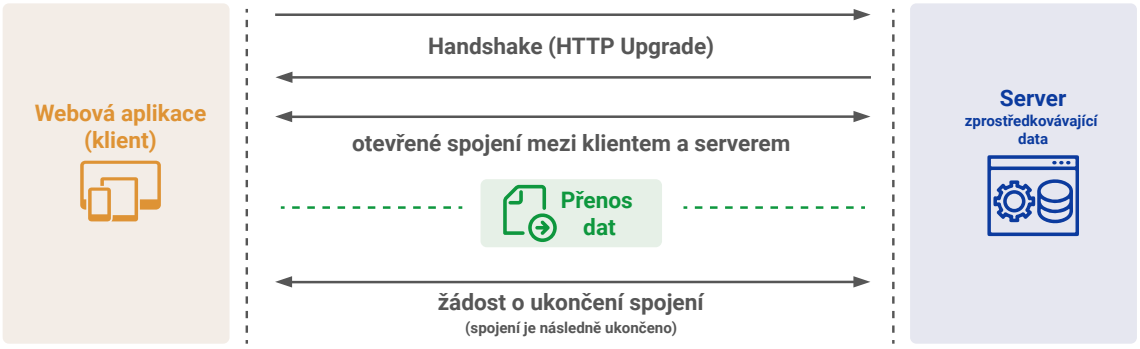
- **GET** – jedná se o metodu, která je používána pro načtení (získávání) dat. Server odpoví s požadovanou reprezentací zdroje v těle odpovědi,
- **POST** – používá se k odeslání dat na server za účelem vytvoření nového prostředku. Často je metoda posílána při vytváření nového objektu v databázi. Server přijatá data zpracuje a odpoví příslušnými stavovými informacemi,
- **PUT** – slouží k aktualizaci prostředku na serveru. Nahrazuje stávající reprezentaci prostředku aktualizovanými daty poskytnutými klientem. Pokud prostředek neexistuje, vytvoří nový. Server odesílá odpověď se stavovým kódem,
- **DELETE** – odstraňuje příslušný prostředek ze serveru. Server identifikuje a odstraní prostředek uvedený v adrese URL zadané klientem a následně odešle odpověď se stavovým kódem zpět klientovi.

Tyto metody, označované jako operace CRUD (Create, Read, Update, Delete), poskytují standardní způsob provádění operací se zdroji prostřednictvím rozhraní REST API.

2.3.2 Websockets

WebSocket API je komunikační technologie poskytující plně duplexní komunikační kanály prostřednictvím dlouhodobého nepřetržitého spojení mezi klientem (prohlížečem uživatele) a serverem. Pomocí tohoto rozhraní může API odesílat zprávy na server a přijímat odpovědi řízené událostmi. Spojení vytvořené pomocí WebSocket trvá tak dlouho, dokud jej kterákoli ze zúčastněných stran nepropustí. Jakmile jedna strana přeruší spojení, druhá strana nebude moci komunikovat, protože spojení se

automaticky přeruší na přední straně. Rozhraní je určeno pro výměnu dat v reálném čase a pouze pro čtení. Ověření uživatele se provádí pomocí klíče API [19].



Obr. 2.2: Diagram Websockets API připojení

Protokol WebSocket je postaven na protokolu TCP (Transmission Control Protocol) a pracuje na aplikační vrstvě TCP/IP. Během počátečního připojení používá proces handshake k navázání trvalého spojení mezi klientem a serverem. Jakmile je spojení navázáno, klient i server si mohou kdykoli vzájemně posílat data bez nutnosti opakovaných požadavků HTTP. Popis spojení pomocí Websockets je uveden na obrázku 2.2.

3 Low-code/No-code platformy a knihovny

Low-code/no-code je metoda vývoje, která umožňuje uživatelům, kteří nemají žádné nebo pouze minimální programátorské znalosti, vytvářet vlastní aplikace bez nutnosti kódování. Tento přístup urychluje vývoj aplikaci a snižuje prostředky nutné k vytvoření finální aplikace. Platformy a frameworky kladou důraz na vizuální rozhraní, která umožňují lidem bez programátorských znalostí relativně snadno vytvářet a nasazovat aplikace [20, 21].

Metoda dovoluje uživatelům vytvářet aplikace (části aplikací, prvky) pomocí vizuálních rozhraní a předem vytvořených komponent. Pomocí low-code/no-code frameworků mohou koncoví uživatelé přetahovat jednotlivé komponenty ve webové aplikaci, konfigurovat jejich parametry a vytvářet logiku pomocí vydefinovaných postupů. Tento přístup umožňuje rychle a efektivně vytvářet vlastní aplikace. Platformy a frameworky vytváří abstraktní vrstvu mezi vyvíjenou aplikací a technickým základem, která umožňuje uživatelům se soustředit na funkčnost aplikace a její prostředí. Technické detaily jsou spolu se základním kódem spravovány platformou nebo frameworkem a koncový uživatel do nich nijak nezasahuje.

Low-code

Low-code vývoj vyžaduje, aby měl uživatel alespoň určité znalosti v kódování, které při tvorbě aplikace bude aplikovat. Předpokládá se, že rozumí základním logickým operacím a dokáže vytvořit základní model své aplikace. Vývojáři a programátoři používají low-code, aby rychle vytvořili makety svých aplikací, které následně mohou testovat a odladovat. Low-code se používá k vývoji jednoduchých webových aplikací nebo k rozšíření funkcí v již existující aplikaci.

No-code

No-code vývoj se zaměřuje na uživatele, kteří nemají žádné technické znalosti. Uživatelé rozumí obchodním potřebám a pravidlům, ale mají malé nebo žádné zkušenosti s programováním. Uživatelé mohou pomocí no-code snadno a rychle vytvářet a nasazovat své obchodní aplikace. No-code se typicky používá při tvorbě jednoduchých aplikací, ve kterých se nespouští žádné procesy. No-code platformy jsou specializovaným typem cloudové platformy, ve které vizuální komponenty řeší funkce specifické pro odvětví nebo pro konkrétní obor podnikání.

Výhody používání low-code/no-code

Hlavními výhodami používání low-code/no-code frameworků a platforem jsou:

- **Rychlost** – Vzhledem k tomu, že hlavní část kódu je již vyvinuta, uživatelům stačí pouze vizuálně konfigurovat komponenty aplikace namísto ručního kódování. Dochází ke zkrácení doby vývoje aplikací [20].
- **Snížení složitosti** – Aplikace nejsou vytvářeny od začátku, díky tomu je vývoj aplikací zjednodušen. Software je přizpůsoben tak, aby vyhovoval požadavkům uživatelů [20].
- **Snížení nákladů** – Díky zkrácení času vývoje aplikace se snižují náklady. Aplikace jsou častěji vyvíjeny společnostmi a ne externími vývojáři.
- **Snadná údržba** – Je důležité, aby bylo možné rychle měnit části aplikace, které byly vytvořeny. Údržba softwaru je zásadní pro zajištění sladění mezi službou nabízenou aplikací a obchodními požadavky.

3.1 Platformy

Hlavním cílem low-code/no-code platforem je umožnit vývojářům ale i koncovým uživatelům rychle vytvářet aplikace, usnadnit jejich konfiguraci a dosáhnout efektivnosti. Použití těchto platforem zrychluje vytváření aplikací a minimalizuje konflikty mezi požadavky. Hlavní část kódu je předem vyvinuta – koncový uživatel pracuje pouze v grafickém uživatelském rozhraní, jehož prostřednictvím může vytvářet a přidávat komponenty. Toto rozhraní může komunikovat s již vytvořenými API třetích stran. Tvůrci aplikací mohou měnit uspořádání a opakovaně testovat moduly, dokud aplikace nebude fungovat podle očekávání.

Při běžném vývoji webových aplikací musí programátoři napsat stovky řádků kódu, aby vytvořili požadované funkce a vlastnosti aplikace. Tento proces vývoje vyžaduje značné znalosti, ať už se jedná o znalosti programovacích jazyků, vývojových prostředí nebo nasazování aplikací na servery. Při použití low-code/no-code platforem není potřeba mít tyto dovednosti. Platformy umožňují vizuálně pracovat s jednotlivými komponenty, které představují konkrétní kroky nebo vlastnosti aplikace a spojují je dohromady, aby vytvořili požadovaný pracovní postup.

Low-code/no-code platformy můžeme rozlišit podle jejich účelu. Některé platformy umožňují koncovým uživatelům vytvářet aplikace a webové stránky, jiné umožňují vyvíjet podnikové aplikace. Každá z těchto kategorií platforem nachází své uplatnění v jiné oblasti. Platformy se díky svému účelu do jisté míry liší, a proto je důležité si předem uvědomit, kterou z nich při tvorbě aplikace použít.

3.1.1 Platformy pro podnikové aplikace

Většina podnikových vývojových low-code platforem umožňuje uživatelům vytvářet interní podnikové aplikace s minimálním kódováním. To uživatelům, kteří nemají programátorské znalosti, dovoluje v reálném čase spolupracovat s programátory a vytvářet vlastní aplikace s výkonnou obchodní logikou podle potřeb společnosti. Typy vytvořených podnikových aplikací se mohou lišit od jednoduchých řešení pro sledování docházky zaměstnanců, plánování úkolů až po plnohodnotný CRM (Customer Relationship Management) software. Tento systém umožňuje společnostem shromažďovat veškeré potřebné údaje o zákaznících, vyhodnocovat jejich chování a tendence trhu, předvídat budoucí potřeby, zjednodušit reporting, analýzy a hlavně udržet přehled o tom, co se děje. Tento software propojuje jednotlivé části společnosti a organizuje jejich aktivity a poznámky do jednoho soudržného systému [23].

Retool

Retool je low-code platforma, která umožňuje společnostem vytvářet vlastní aplikace nebo části aplikací, jako jsou řídící panely, panely pro správu nebo nástroje určené ke spouštění důležitých operací. Platforma k vytváření těchto aplikací využívá stavební bloky (komponenty pro tabulky, formuláře, grafy). Aplikaci je tedy možné sestavit pomocí těchto bloků bez nutnosti kódování. Platforma dovoluje propojení s dalšími platformami a databázemi pomocí API.

Platforma není omezená pouze na stavební bloky, které obsahuje, ale dovoluje části aplikace psát kódem. Dovoluje vytvářet vlastní bloky a funkce, importovat externí JavaScript knihovny a testovat vyvíjené aplikace. Ovládá skupiny uživatelů a umožňuje jim přidávat práva a přístupy k jednotlivým modulům. Platforma kromě tvorby uživatelského rozhraní taktéž podporuje vytváření automatizaci pracovních postupů (zatím jen beta verze).

Platformu používá řada známých společností v různých odvětvích, jako je například Coinbase, NBC, Volvo nebo Amazon [24].

3.1.2 Platformy pro uživatelské aplikace a webové stránky

Některé vývojové low-code/no-code platformy umožňují koncovým uživatelům vytvářet aplikace a webové stránky pro použití, jako jsou klientské portály nebo software pro plánování schůzek. Tato řešení umožňují podnikům využívat výhod vývoje s nízkým obsahem kódu a vytvářet software určený např. pro potenciální zákazníky. Vytváření aplikací a webových stránek přináší mnoho výhod, avšak i rizika, která je potřeba brát na vědomí. Uživatelé, kteří nemají dostatečné znalosti v oblasti vývoje a nasazování aplikací, nemusí být obeznámeni s osvědčenými postupy zabezpečení.

Microsoft Power Apps

Power Apps je sada aplikací, služeb a konektorů, která poskytuje vývojové prostředí určené pro vytváření vlastních aplikací určených pro obchodní potřeby. Platforma vyvíjeným aplikacím dovoluje připojení k datům uložených v základní datové platformě (Microsoft Dataverse), nebo v různých online zdrojích jako je Microsoft 365, SQL server atd. Výhodou aplikací vyvíjených v platformě Power Apps je jejich responzivní design. Tyto aplikace lze bez problémů spustit ve webovém prohlížeči i na mobilních zařízeních (mobilní telefony a tablety).

Platforma Power Apps je vhodná jak pro uživatele, kteří prostřednictvím low-code/no-code dokáží vytvářet vlastní aplikace, tak i pro administrátory a vývojáře. Těm platforma dovoluje navíc spravovat prostředí, analyzovat získaná data a propojit ji s dalšími Microsoft platformami jako je např. platforma Power Automate. Vývojáři mohou používat logiku na straně serveru pomocí funkcí Azure, mohou využívat zásuvných modulů a rozšířit pracovní postupy, mohou integrovat s externími daty pomocí virtuálních entit nebo vytvářet vlastní konektory a ty následně vkládat do webových stránek [26].

Velkou výhodou platformy Power Apps je podpora konektivity s dalšími platformami a funkcemi společnosti Microsoft. Platforma Power Apps může například pracovat s Microsoft Azure a zároveň spravovat serverovou a klientskou část aplikace včetně databáze [27].

3.1.3 Platformy pro tvorbu a řízení procesů

Mnoho low-code/no-code platforem nabízí společnostem cestu k digitální transformaci prostřednictvím vytváření a správy obchodních procesů. Tyto nástroje umožňují uživatelům vytvářet zcela přizpůsobené procesy díky vizuálnímu uživatelskému rozhraní s minimálním kódováním. Platformy provádějí funkce podobné softwaru pro řízení obchodních procesů – dokáží analyzovat, automatizovat, optimalizovat a zlepšovat obchodní procesy. Používáním platforem, které umožňují přímé vytváření těchto procesů na úrovni kódování, dochází ke zlepšení a optimalizaci výsledku.

Webcon BPS

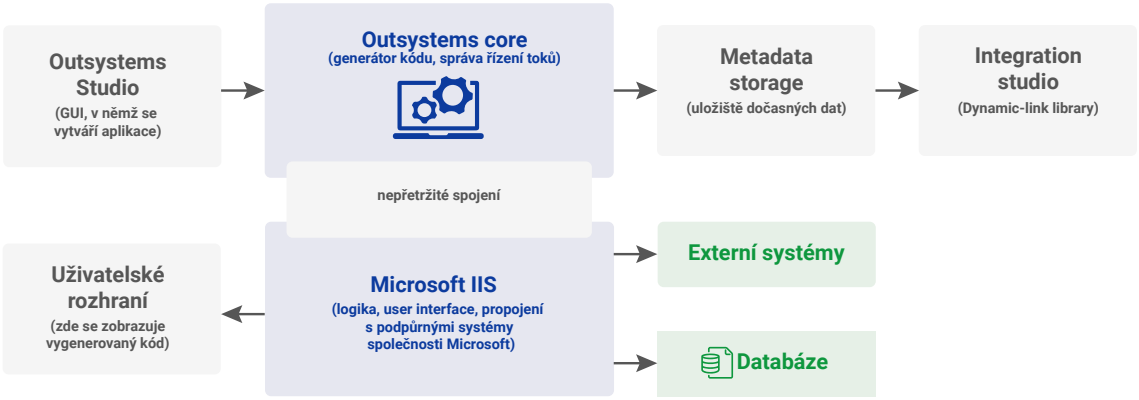
Low-code platforma Webcon BPS slouží k automatizaci digitálních procesů. Systém umožňuje vytvářet procesně orientované aplikace pro každou obchodní potřebu. Platforma nachází své využití v oblastech bankovnictví, službách ale i ve výrobě, skladování a automotive. Platforma umožňuje automatizovat opakující se úlohy, zefektivnit řízení projektů, řídit výrobní cyklus a efektivně implementovat nové pracovní postupy [28].

Procesy, které společnosti pomocí platformy mohou automatizovat a spravovat:

- **Řízení provozu skladu** – Společnosti zefektivňují proces expedice zboží ze skladu, zvyšují bezpečnost tím, že zabraňují krádežím nákladu, a poskytují úplný záznam dat týkajících se nákladních vozidel, řidičů a nákladu.
- **Náprava a preventivní opatření** – Aplikace automatizujte pracovní postupy, usnadňuje identifikaci problémů, analyzuje hlavní příčiny a vede záznamy o všech incidentech, ke kterým došlo.
- **Digitální pracovní návod** – Aplikací zprostředkovává interaktivní digitální prostředí a zajišťuje standardizaci dokumentů, které lze snadno aktualizovat.
- **Vývoj nového produktu** – Aplikace zefektivňuje proces vývoje produktu, zlepšuje spolupráci mezi odděleními a poskytuje bezproblémový přístup ke všem datům souvisejícím s produktem.

Outsystems

Outsystems je low-code platforma, která umožňuje vytvářet, testovat a nasazovat mobilní a webové aplikace. Využívá vizuálního programovacího jazyka k vytváření aplikací, které jsou flexibilní, agilní a škálovatelné. Platforma poskytuje mnoho vestavěných funkcí, jako je návrh uživatelského rozhraní, přístup k datům a nástroje pro vytváření sestav. Je možné její funkce rozšířit použitím knihoven třetích stran nebo integrováním své vlastní služby prostřednictvím adaptérů platformy. Knihovny lze taktéž jako konektory s externími systémy, šablonami uživatelských rozhraní nebo s komponenty třetích stran (Google Maps).



Obr. 3.1: Architektura platformy Outsystems [29].

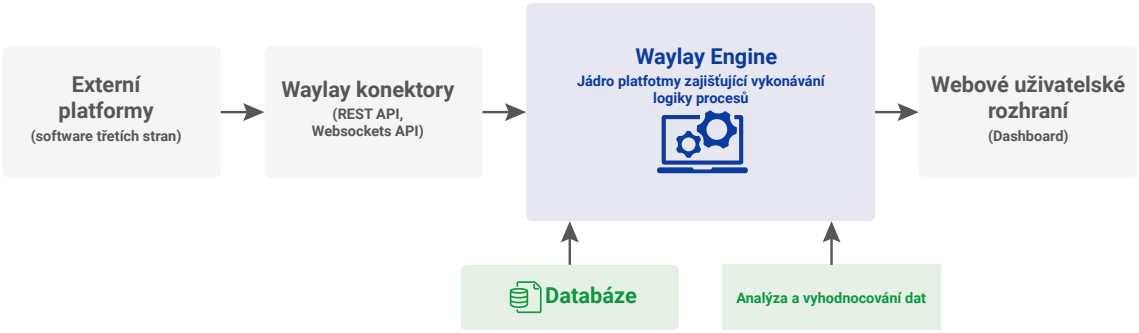
Na obrázku 3.1 je zobrazena architektura low-code platformy Outsystems. Platforma se skládá z několika důležitých logických bloků. **Server platformy** Outsystems je jádrem aplikace. Jedná se o serverovou komponentu, která doplňuje standardní zásobník webových aplikačních serverů o sadu služeb. Server platformy za-

jišťuje všechny kroky potřebné ke generování, optimalizaci, kompilaci a nasazení aplikací do standardního webového aplikačního serveru [30].

Generátor kódu přebírá aplikační model, který byl vyvinut ve vizuálním editoru, a generuje všechny aplikační komponenty připravené k nasazení na aplikační server. Když generátor kódu načte aplikační model, spustí hloubkovou grafovou analýzu aplikačního modelu, zkontroluje optimalizace, které lze provést, vygeneruje integrační proxy a nativní kód pro všechny vrstvy v aplikaci. **Služby nasazení** nahrají vygenerované aplikační komponenty na aplikační server a zajišťují konzistentní instalaci aplikace na každý server organizace [30].

Waylay IO

Low-code platforma Waylay IO automatizuje data a snižuje složitost procesu vývoje aplikací. Automatizační technologie platformy používá malé části kódu, které jsou opakovaně použité v různých procesech a je řízena rule-based enginem. Automatizační platforma nabízí potřebné nástroje na jednom místě a pomáhá vývojářům experimentovat s daty a rychle vytvářet aplikace a obchodní modely.



Obr. 3.2: Architektura platformy Waylay IO

Architektura platformy Waylay IO zobrazená na obrázku 3.2 je složena z několika bloků. Pomocí konektorů je platforma propojena s webovými aplikacemi třetích stran. K propojení využívá například REST API nebo Websockets API třetích stran. V hlavním bloku platformy logika vykonává všechny operace s daty přijímanými z aplikací třetích stran. Tato data převzatá z aplikací jsou nazývána digitální dvojčata a reprezentují fyzické objekty, nad kterými rule-based engine vykonává pravidla. Tento rule-based engine komunikuje s databází, do které jsou ukládány vytvořené toky a pravidla, a zároveň je propojen s vlastním webovým rozhraním, v němž vývojářům dovoluje vytvářet pravidla a analyzovat nasbíraná data [31].

3.1.4 Porovnání low-code platforem

Každá z výše uvedených platforem má své výhody a nevýhody, které je potřeba vzít v úvahu při výběru vhodné platformy pro vývoj vlastní aplikace. Vzhledem k tomu, že platforma **Retool** se zaměřuje na vývoj aplikací v oblasti podnikových aplikací (bankovnictví, marketing, propagace zboží) a platforma **Microsoft Power Apps** umožňuje vytvářet uživatelské aplikace a automatizovat logické toky, budou porovnávány pouze platformy **Webcon BPS**, **Outsystems** a **Waylay IO**, které nabízí nástroje pro vytváření toků (pravidel) nad výrobními a logistickými procesy.

Tyto platformy jsou porovnávány v následujících kategoriích: **funkcionalita platformy**, **uživatelské rozhraní** a **demo verze**.

Tab. 3.1: Low-code platformy pro tvorbu a řízení procesů.

	Funkcionalita	**Uživatelské rozhraní**	**demo verze**
Webcon BPS	vytváření toků nad procesy, propojení s API třetích stran, nabízí řešení na míru	ANO - web	neomezená
OutSystems	vytváření toků nad procesy, propojení s API třetích stran, real-time sledování	ANO - software	neomezená
Waylay IO	vytváření toků nad procesy, připojení API třetích stran (více konektory), rule based engine, developer nástroje	ANO - web	30 dní

V tabulce 3.1 je uvedeno srovnání low-code/no-code platforem určených pro tvorbu a řízení procesů. Všechny uvedené platformy mají uživatelské vývojové rozhraní a nástroje pro vytváření a automatizování procesů. Uživatelské rozhraní platformy Outsystems působí přívětivěji než prostředí platforem Webcon BPS a Waylay IO, ale je nutné ho nainstalovat do počítače. Všechny vývojová prostředí dovolují vytvářet toky nad procesy a do určité míry je analyzovat.

Platforma Outsystems nabízí přes 400 před-vytvořených konektorů které umožňují vytvářenou aplikaci propojit s databázemi nebo systémy třetích stran (pomocí API) [30]. Platforma Webcon se zaměřuje převážně na vývoj aplikací vyvíjených pro konkrétní problematiku [28]. Výhodou platformy Waylay IO je rule based engine, který zajišťuje rychlý a spolehlivý tok dat v aplikaci. Tato platforma rovněž umožňuje propojení s externími systémy pomocí různých konektorů jako je Websockets API, Google IoT connector a AWS IoT Core connector [31].

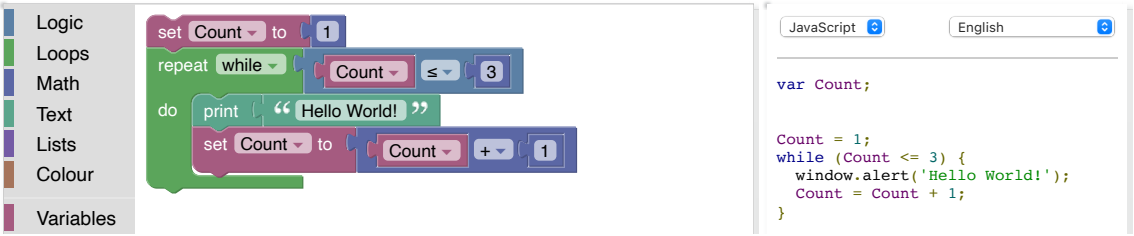
3.2 Knihovny

Low-code/no-code implementace může být do jisté míry dosažena i užitím různých knihoven. Tyto knihovny sice vývojářům nedovolují použití samotné low-code/no-code funkcionality, ale funkcionalitu ve vyvíjených aplikacích vytváří. Jedná se o knihovny pracující s drag-and-drop nástroji. Vydefinováním takovýchto objektů pomocí existujících knihoven mohou vývojáři vytvořit webové aplikace, které jejich koncovým uživatelům umožní použít low-code/no-code. Příkladem je níže uvedená webová aplikace Microsoft Power Automate, která obsahuje low-code/no-code implementaci pomocí níž si koncový uživatel vydefinuje logické toky.

Existuje celá řada low-code knihoven, které lze použít a pomocí nich vydefinovat vlastní implementaci low-code/no-code. Příkladem jsou následující uvedené:

Blockly

Blockly je knihovna pracující na straně klienta ve většině webových prohlížečů, která umožňuje vývojářům vytvářet drag-and-drop vydefinované bloky. Koncový uživatel s těmito před vytvořenými bloky manipuluje a tím vytváří logickou strukturu, která reprezentuje kód. Výstupem tohoto jednotného bloku vytvořeného uživatelem je syntakticky správný kód ve vybraném programovacím jazyce [32]. Ukázka vytvoření jednoduchého kódu pomocí této knihovny je zobrazena na obrázku 3.3.



Obr. 3.3: Ukázka vytváření kódu pomocí knihovny Blockly

Angular Drag and Drop

Je knihovna poskytovaná frameworkem Angular. Umožňuje snadno a rychle vytvářet uživatelské rozhraní drag-and-drop. Podporuje přetahování objektů do bloků, řazení v seznamech, přetahování objektů ze seznamů a dovoluje i vlastní definici přetahování [33]. Je založena na principu vydefinování elementů, z nichž některé mají funkci přetahování a jiné jsou pouze statické (nelze je přetahovat). Do těchto statických elementů je možné vkládat bloky s funkcí přetahování.

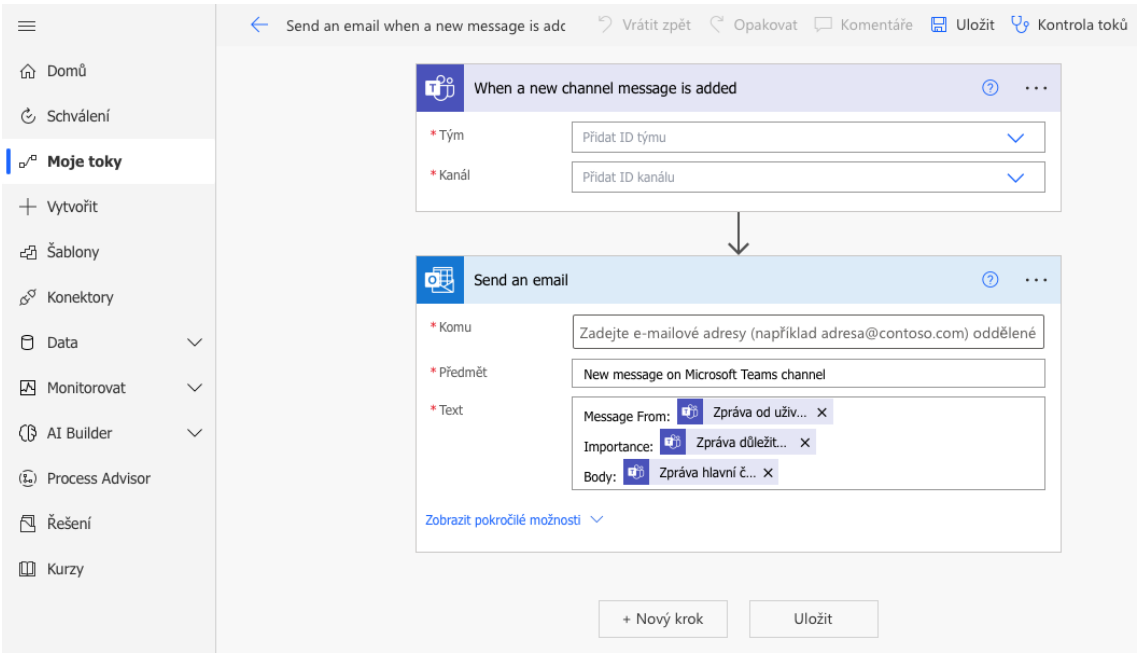
3.3 Aplikace využívající low-code/no-code

Platformy pomocí low-code/no-code implementace obecně zrychlují a usnadňují vývoj aplikací. Můžeme se ale setkat i s možností, kdy aplikace nebyla vyvinuta prostřednictvím low-code platforem, ale přesto používá vlastní implementaci low-code/no-code. Použití low-code/no-code v aplikaci může koncovým uživatelům usnadnit, zrychlit nebo do určité míry postup zautomatizovat.

Případem může být aplikace sloužící k vytváření jednoduchých pravidel nad správou chytré domácnosti. Místo toho, aby koncový uživatel vytvářel logická pravidla (např. *když je v místnosti méně jak 20 °C, zapni topení*) psaním kódu, využije stavební bloky, které jednotlivé části kódu reprezentují. Postup je obdobný jako u low-code platforem, ale namísto použití bloků k tvorbě webové aplikace jsou použity k vytvoření pravidel pro řízení chytré domácnosti. Vizuální blokové pravidlo se ve výsledku přeloží do kódu.

Microsoft Power Automate

Aplikace Power Automate je softwarový nástroj, který uživatelům umožňuje vytvářet automatizované pracovní postupy mezi různými aplikacemi a službami společnosti Microsoft pomocí low-code/no-code implementace.



Obr. 3.4: Vytváření toků v platformě Microsoft Power Automate

Pomocí Power Automate uživatel může vytvořit tok o několika krocích nebo si vybere ze stovek předem vytvořených šablon. Aby tyto toky mohly fungovat, je

vyžadováno propojení aplikace se zdroji dat (OneDrive, Outlook, SharePoint). Po nastavení pracovního postupu v Power Automate funguje tento postup na pozadí a upozorňuje uživatele na konkrétní oznámení, nahrání souboru nebo dokončuje požadované automatické úlohy. Pracovní postupy závisí na tom, kdy dojde ke spouštěcí události. Událost může být spuštěna periodicky, plánovaně nebo automaticky, kdykoliv dojde k vyvolání toku [34].

Příklady pracovních postupů:

- Při obdržení přílohy e-mailu z Outlooku ulož přílohu e-mailu na OneDrive.
- Když je naplánována nová událost, odešli upozornění na Outlook a Kalendář Google na základě události.
- Když je vytvořen tweet pomocí hashtagu, okamžitě tyto tweety ulož do konkrétního umístění SharePointu.

4 Implementace rule-based enginu

4.1 Požadavky na rule-based engine

Navrhovaný rule-based engine s uživatelským prostředím má zpracovávat a vyhodnocovat data přenášená v reálném čase pomocí bezdrátových technologií (UWB). Nad nasbíranými daty, která reprezentují polohu a informace o fyzických objektech pohybujících se v uzavřených prostorách (továren, skladišť), má vytvářet logiku a vykonávat pravidla. V podkapitole 1.3.1 jsou tyto objekty rozděleny do čtyř kategorií – pro každý typ objektu se mohou vytvářet odlišná pravidla. Například pro objekt typu *Materiál a výrobky* může být požadováno vytvoření pravidla, které kontroluje, že výrobek neopakoval krok výrobního procesu. Toto pravidlo naopak nemusí být užitečné pro objekt typu *Lidé (Osoby)*, protože osoba neprochází žádným výrobním procesem, pouze ho spravuje.

Logika

Systém bude muset překládat prostou (slovně interpretovanou) informaci zadanou uživatelem do počítačové logiky. V případě, že se bude jednat o složitější interpretaci, bude náročnější vymyslet logiku. Může se stát, že uživatel chybně vyjádří pravidlo nebo podmínka bude příliš dlouhá, a její kontrola bude vyžadovat delší dobu. S těmito problémy je potřeba počítat a zároveň jim předcházet. Řešením složitosti těchto pravidel je vytvořit šablony, které uživatel bude moct použít a nebude nucen vytvářet vlastní pravidla.

Čas

Čas přidává rule-based enginu (resp. části systému, která bude vyhodnocovat vytvářená pravidla) složitost. Systém bude kombinovat asynchronní datové toky (real-time přijímaná data) se synchronními informacemi (koncové API pro dotazování). Může se stát, že v době vykonávání pravidla podmínka nebude aktuální.

Představme si výrobní halu, v níž je umístěn stroj a kolem něj je vytvořena zóna. Nad touto zónou bude vytvořené pravidlo, že když do ní vstoupí jakákoliv osoba, dojde ke zpomalení stroje a spustí se výstražný zvuk. Předpokládejme, že osoba tohle varovné hlášení neuslyší a přiblíží se ke stroji nebezpečně blízko. V takovém případě bude nutné stroj zastavit úplně. Z toho vyplývá, že námi navrhovaný systém bude muset do jisté míry predikovat budoucnost a zároveň pracovat s minulostí (*Je podmínka stále aktuální? Jak dlouho podmínka bude splněna?*).

Nejistota

Z výše uvedené situace vychází i problém nejistoty. Osoba v předchozím případě neuslyšela varovné hlášení a dál se přibližovala ke stroji. Existuje ale jistá pravděpodobnost, že výstražné hlášení osoba zareaguje a od stroje odstoupí. S touto pravděpodobností by měl navrhovaný systém pracovat. Nejistota je ovlivňována i samotným přenosem signálu, protože ten je přenášen bezdrátovou technologií z tagů. Samotná životnost těchto hardwarových zařízení závisí na jejich baterii.

4.2 Pravidlo

Jedná se o základní stavební blok celého rule-based enginu vytvořený koncovým uživatelem aplikace – osobou, která rozumí výrobním a logistickým procesům a dokáže na základě jejich stavu vydefinovat vhodné spouštěcí podmínky a výstupní akce. Vydefinováním těchto spouštěčů a akcí vzniká pravidlo, které popisuje konkrétní procesní problém (situaci) a obsahuje všechny potřebné informace k jeho eliminaci nebo následnému logickému řešení.

Ve výpisu 4.1 je představen scénář, který bude chtít koncový uživatel v systému vydefinovat. Uživatel bude chtít vytvořit pravidlo, které bude kontrolovat vstup do zóny 41. V případě, že jakýkoliv tag do této zóny vstoupí, bude vypsáno uživatelem vytvořené upozornění. Aby pravidlo mohlo být vykonáno, rule-based engine musí získat informace o všech zónách v budově a informace o stavu jednotlivých tagů. Na základě těchto zjištěných hodnot, vyhodnocuje, jestli jakýkoliv tag vstoupil do zóny zvolené uživatelem, který pravidlo vytvořil. Jestliže tag do zóny 41 vstoupí, podmínka bude splněna a vykonají se všechny akce, které uživatel požadoval (v tomto případě se vypíše upozornění signalizující vstup do zóny).

Výpis 4.1: Scénář ve výrobním procesu

```
Scénář: Uživatel chce vypsat hlášení pokaždé, kdy jakýkoliv tag
vstoupí do zóny 41.

given:   stav tagů a informace o všech zónách
 when:   jakýkoliv tag vstoupil do zóny 41
 then:   vypsat uživatelm vydefinované hlášení
```

Z výše popsaného scénáře vyplývá, že se pravidlo bude skládat ze dvou částí. V prvním kroku bude vydefinovaná podmínka (**when**). Námi navrhovaný systém vyhodnocuje, zdali jsou podmínky splněny či nikoliv. Druhá část pravidla (**then**) se vykoná pouze v případě, že budou splněny podmínky vydefinované uživatelem.

Aby koncový uživatel nemusel vytvářet vlastní pravidla, bude moci využít již

předem vydefinovaná pravidla uložená do šablon. Použitím těchto šablon předcházíme složité logice a chybám v podmínce, kterých se uživatel mohl dopustit.

4.2.1 Části pravidel

Scénář uvedený ve výpisu 4.1 popisuje velmi jednoduché pravidlo, které obsahuje jednu spouštěcí událost, která aktivuje právě jednu akci. Pro následný popis budeme tyto dva rozdílné parametry pravidla rozlišovat na **trigger** (spouštěcí událost) a **akci** (událost vykonávanou v závislosti na výstupní hodnotě triggeru) [35].

Trigger

Vydefinovanou podmínku pravidla nazýváme **trigger**. Trigger je událost, která aktivuje pravidlo. Událost může být periodická (pravidlo se aktivuje pravidelně v určitý čas), manuální (pravidlo manuálně aktivuje uživatel spravující systém) nebo automatická (pravidlo aktivuje trigger). Jedno pravidlo může obsahovat více triggerů. Tyto triggery jsou mezi sebou budou odděleny logickými operacemi AND, OR nebo NOT. V bodech jsou uvedeny triggery, které mohou aktivovat pravidlo:

- úroveň baterie tagu je nižší než 20 procent,
- výrobek přeskočil výrobní krok,
- osoba se přiblížila do blízkosti hledaného nástroje,
- produkt byl ve výrobním procesu příliš dlouho.

Námi navržený systém se dotazuje RTLS Studia prostřednictvím REST a Websockets API. Z již existujícího RTLS systému přebírá pomocí těchto konektorů informace o objektech, zónách, budovách a dalších typech entit vydefinovaných v RTLS. Některé z výše uvedených triggerů zatím nejsou vydefinovány v Sewio RTLS softwaru. Sewio RTLS zatím nerozlišuje typy objektů (na všechny objekty pohlíží stejně) a nemá nadefinované pracovní postupy pro jednotlivé objekty. Z toho vyplývá, že některá pravidla nebude možné vytvořit, protože zatím neexistuje systém, který by zajistil jejich vydefinování.

Akce

Výstupem pravidla je **akce**. Vykoná se pro všechny objekty, u nichž je splněna uživatelem vydefinovaná podmínka (triggery). Akcí rozumíme událost, která zasahuje a ovlivňuje chod výrobního nebo logistické procesu, upozorňuje obsluhu a pracovníky, spravuje administrativu nebo plánuje události. Stejně jako trigger může spouštět více akcí, může i jedna akce být spuštěna více triggery.

Příklady akcí mohou být: naplánovat do kalendáře výměnu baterie, zpomalit pohyblivý stroj, odeslat upozornění všem osobám v zóně, exportovat trasu pohyblivého stroje do souboru.

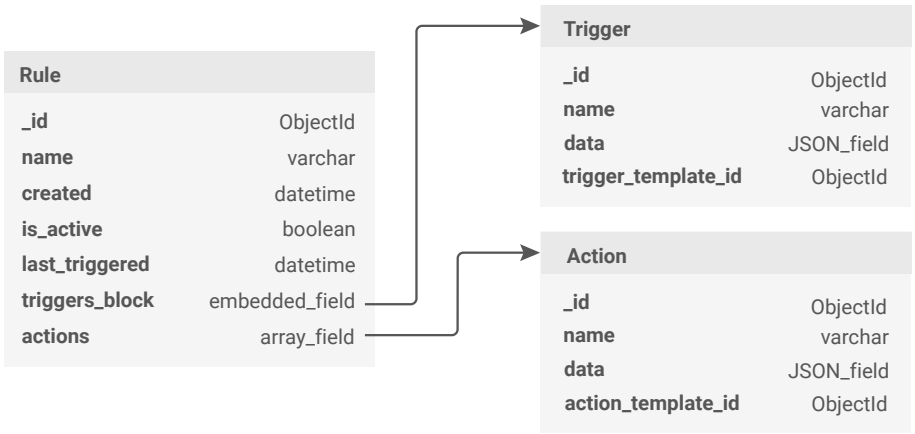
4.2.2 Struktura pravidel

Aby mohlo být pravidlo vytvořeno a správně vyhodnoceno vytvořeným rule-based enginem, musí obsahovat parametry, které jeho vyhodnocení umožní. Pravidlo je složeno z několika jednoduchých atributů a dvou entit nimiž jsou: **blok spouštěčů** a **seznam akcí**.

Blok spouštěčů obsahuje všechny triggery a logické operace mezi nimi vydefinované koncovým uživatelem. Tento blok je v hodnotě `true` pouze v případě, že engine vyhodnotí všechny podmínky jako splněné. Seznam akcí obsahuje všechny akce, které jsou vykonány v případě, že hodnota bloku spouštěčů je `true`.

Pravidlo kromě dvou výše popsaných entit obsahuje atributy:

- **name** – jméno pravidla, které není zadáváno uživatelem, ale je automaticky generováno na základě triggerů a akcí, z nichž je pravidlo složeno,
- **created** – informace, kdy bylo pravidlo vytvořeno,
- **is_active** – stavová hodnota, která indikuje, zdali je pravidlo aktivní. V případě, že je tato hodnota `true`, pravidlo je aktivní a engine jej vyhodnocuje,
- **last_triggered** – jedná se o časovou značku, která ukládá aktuální časovou hodnotu posledního spuštění pravidla.



Obr. 4.1: Zjednodušená struktura pravidla

Na obrázku 4.1 je zobrazen zjednodušený model pravidla. Pravidlo obsahuje všechny výše uvedené atributy. Atribut `actions` je pole akcí, z nichž se každá odkazuje na model **Action**. Ta obsahuje vlastní atributy `jméno`, `data` a vnořené

pole `šablony akce`. Atribut `triggers_block` je **Block** obsahující jasně definovanou strukturu jednotlivých triggerů a logických operací mezi nimi tvořící konečnou vstupní podmínku, která je v rule-based enginu vyhodnocována. Může se stát, že v atributu `triggers_block` bude uložen pouze jeden **Trigger**, což je také možné, protože pravidlo musí mít alespoň jeden spouštěč.

Výpis 4.2: Ukázka principu vytváření bloků

```
class Block:
    value: bool

class Trigger(Block):
    def __init__(self, name: str):
        self.name = name

class Operand(Block):
    operand_type: str

if __name__ == "__main__":
    a = Trigger("Trigger_A")
    b = Trigger("Trigger_B")
    c = Trigger("Trigger_C")

    a.value = True
    b.value = False
    c.value = False

    # Rovnice: [(Trigger_A AND Trigger_B) OR (NOT Trigger_C)]
    block = Operand("OR", Operand("AND", a, b), Operand("NOT", c))
```

Výpis 4.2 znázorňuje princip vytváření `bloku spouštěčů`. Třída **Block** má dvoustavovou hodnotu a tu dědí jak **Trigger**, tak **Operand**. Tato hodnota indikuje, zdali je výrok triggeru v okamžiku vykonávání pravdivý či nikoliv. Hodnota je předávána operandu, který na základě získaných hodnot vyhodnocuje hodnotu podmínky. Jedná se o princip zanořování, kdy tímto principem mohou vznikat velmi složité počáteční podmínky (bez omezení lze zanořovat do nekonečna).

V ukázce je znázorněn princip funkčnosti na jednoduché logické operaci, která je složena ze tří vstupních proměnných – tří triggerů aktivujících jedno pravidlo, z nichž má každý nastavenou hodnotu `true` nebo `false` v okamžiku vykonávání pravidla. Výstupem této podmínky je proměnná **block**, která reprezentuje logickou rovnici *výsledek = ((a AND b) OR (NOT c))*. V závislosti na počátečních hodnotách jednotlivých triggerů je výsledkem této rovnice hodnota `true`. V takovém to případě je pravidlo aktivováno a jsou provedeny uživatelem vydefinované akce.

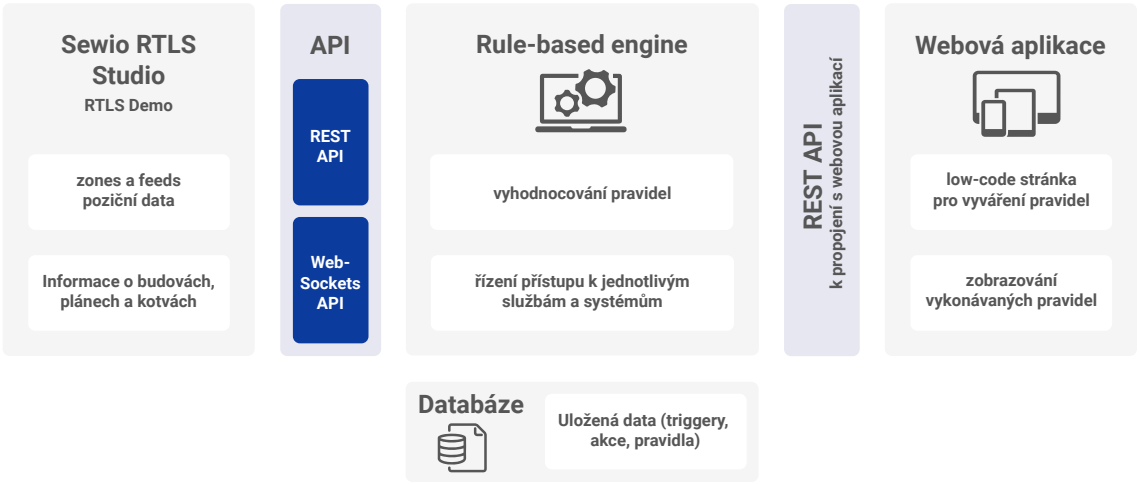
Z ukázky lze odvodit, že vydefinování `bloku spouštěčů` je poměrně náročná operace, která vede k mnoha problémům, které je třeba vzít v úvahu:

- koncový uživatel bude moci pravidla zanořovat do nekonečna pokud nedojde k určitému omezení (řešením je například omezit zanořování do úrovně 3),
- uživatel získá možnost vytvářet poměrně složité počáteční podmínky, jejichž vyhodnocování bude rule-based engine provádět déle, než vstupní podmínku pouze s jedním spouštěčem,
- pro samotného uživatele se kontrola vytvořené podmínky stane složitou a může se stát, že při jejím vytváření provede zásadní chybu, kdy vytvoří výrok, který například nikdy nebude moct nabýt negativní hodnoty, což může vést k závažným problémům spojeným s vykonáváním akcí.

Přestože tato struktura vstupního bloku funguje a je již implementována v modelu systému (je ji možné použít při vytváření pravidel), pro vývoj testovacího rule-based enginu je možné vytvořit vstupní blok obsahující pouze jeden trigger.

4.3 Model rule based enginu

Model navrhovaného rule-based enginu je zobrazen na obrázku 4.2. Navrhovaný systém se skládá z několika oddělených bloků, které jsou nezbytné pro správnou funkčnost enginu jako celku. Každá část systému obsluhuje jiné procesy a vykonává jiné úlohy. Tyto části spolu však vzájemně komunikují a předávají si data nezbytná pro funkčnost systému.



Obr. 4.2: Architektura navrženého rule-based enginu

Jádrem systému je rule-based engine, který komunikuje s ostatními bloky systému. Pomocí konektorů Websockets a REST API je propojený s již existujícím

RTLS systémem, ze kterého přebírá data v reálném čase a na základě získaných informací vyhodnocuje pravidla vytvořená ve webové aplikaci, která slouží jako vizuální rozhraní pro komunikaci s koncovým uživatelem. Komunikace mezi rule-based enginem a uživatelským rozhraním funguje prostřednictvím vytvořeného API, na které se webová aplikace dotazuje a serverová část enginu zpracovává příchozí požadavky. Jádro systému tedy vykonává hned několik funkcí nezávisle na sobě:

- slouží jako serverová část aplikace, která se dotazuje databáze a získaná data předává webové aplikaci,
- přijímá data v reálném čase prostřednictvím Websockets ze Sewio RTLS,
- vyhodnocuje pravidla vytvořená uživatelem.

4.3.1 Komunikace s webovou aplikací

Systém je vytvořen prostřednictvím frameworku *Django*, který používá programovací jazyk *Python*. Pro komunikaci s webovou aplikací je používán *Django REST framework*. Jedná se o subsystém, který vytváří vydefinované REST API, na který se dotazuje webová aplikace. Framework zajišťuje serializovaní dat, které následně předává v těle odpovědi zpět do uživatelského rozhraní.

Pro každý objekt je vytvořen **Model**, pomocí kterého jsou data ukládána do databáze, a **Serializer**, který definuje požadovaný formát dat a ověřuje validní strukturu dat (ať už při ukládání do databáze nebo pro zobrazení dat). **View** je komponenta založená na funkci nebo třídě, která zpracovává příchozí požadavky a generuje příslušné odpovědi HTTP. Komponenta je zodpovědná za validní generování dat a vracení serializovaných dat.

Pomocí těchto tří prvků serverová část aplikace zpracovává požadavky přijaté z klientské aplikace a navrací serializovaná data v jednotlivých `views` ve formátu `.json`. Navrácená data si webová aplikace zpracovává nezávisle na serveru.

Výpis 4.3: Vzory URL pro entitu Rule

```
urlpatterns = [

    #RULES Views
    path('templates/rules', RuleTemplateCreateAPIView.as_view()),
    path('rules', RuleAPIView.as_view()),
    path('rules_all', RuleCreateAPIView.as_view()),
    path('rules/<ObjectId:_id>', RuleUpdateView.as_view()),

    #ENGINE:
    path('engine', RuleEngineAPIView.as_view()),
]
```

Ve výpisu 4.3 jsou uvedeny URL adresy pro entitu **Rule**.

Cesta `'templates/rules'` zobrazuje všechny před-vytvořené šablony, pomocí kterých uživatel může definovat vlastní pravidla. Cesta `'rules'` zobrazuje název a stav pravidla, `'rules_all'` slouží k vytvoření nového pravidla a zobrazení všech atributů každého z nich. Pomocí cesty `'rules/<ObjectId:_id>'` lze aktualizovat parametry pro konkrétní pravidlo. Speciální cestou je `'engine'`, která vytváří dataset všech aktivních pravidel pro samotný rule-based engine, který se na API dotazuje.

4.3.2 Propojení s RTLS systémem

V okamžiku, kdy je aktivní celý systém, je navázáno Websocket spojení se Sewio RTLS Studiem. Aby mohlo být spojení navázáno, je potřeba znát cílovou URL adresu serverová části `DEST_URI` a klíč `X_API_KEY`, který je vložen do hlavičky při vytváření spojení.

Výpis 4.4: Navázání Websockets spojení s RTLS systémem

```
def handle(self, *args, **options):
        ws = websocket.WebSocketApp(DEST_URI,
                on_open=on_open,
                on_message=on_message,
                on_error=on_error,
                on_close=on_close)
        ws.run_forever()
```

Navázání spojení je zobrazeno ve výpisku 4.4. Při spuštění aplikace je otevřen Websockets kanál, který zůstává aktivní po celou dobu chodu systému. Spojení obsahuje čtyři typy zpráv, které jsou zpracovávány. Funkce `on_open` je spuštěna při navázání spojení a má za úkol dotázat se na správnou URL adresu a předat klíč. V případě, kdy nedojde k úspěšnému připojení, je zavolána funkce `on_error`. Funkce `on_message` naslouchá a v okamžik, kdy klientská část přijme zprávu, je předána do rule-based enginu, který na základě dat v ní vyhodnocuje pravidla.

4.3.3 Princip vyhodnocování pravidel

Rule-based engine na jedné straně přijímá data v reálném čase z RTLS a na druhé se dotazuje na všechna aktivní pravidla vytvořená uživatelem. Má tedy k dispozici aktuální data `data_query` a všechna aktivní pravidla `active_rules`.

Předem má jasně definované proměnné dat, které přijímá z RTLS. Tím rule-based engine zajišťuje, že bude schopný zpracovat všechna data, aniž by došlo k nestandardní události, která by vedla k chybnému vyhodnocení pravidla. Zároveň má engine uložené všechny akce, které jsou koncovému uživateli dovoleny vydefinovat.

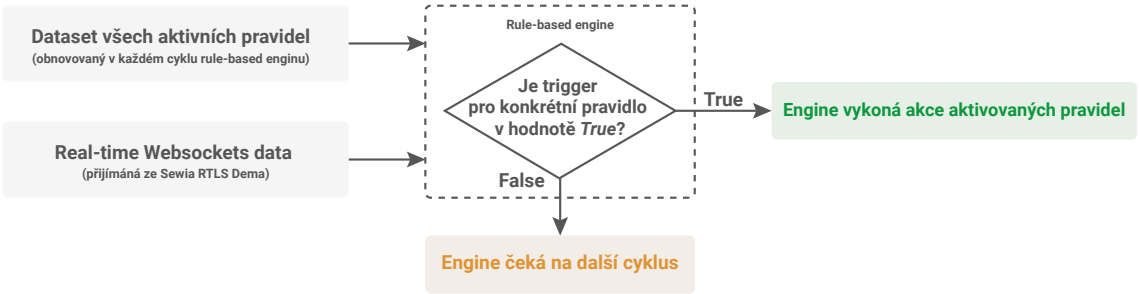
Ve výpisku 4.5 je zobrazen kód spravující a zajišťující správnou funkčnost jádra enginu. V proměnné `data_query` jsou uložena data přijatá v reálném čase prostřednictvím spojení Websockets API. Proměnná `active_rules` slouží pro dočasné uložení všech aktivních pravidel. Je důležité se do databáze dotazovat v každém časovém cyklu rule-based enginu, protože se stav jednotlivých pravidel může měnit v závislosti na koncovém uživateli, který tato pravidla upravuje za chodu systému.

Výpis 4.5: Jádro rule-based enginu

```
def engine_core(data):
    data_query = Data(**create_data_dict(data))
    active_rules = []

    rules_objects = Rule.objects.filter(is_active__in=[True])
    for rule in rules_objects:
        ruleSer = RuleEngineSerializer(rule, many=False)
        active_rules.append(ruleSer.data['data'])

    export_rule_data(RuleVariables, RuleActions)
    run_all(rule_list=active_rules,
            defined_variables=RuleVariables(data_query),
            defined_actions=RuleActions(data_query),
            stop_on_first_trigger=True
            )
```

Po načtení všech důležitých dat je důležité je předat enginu. To provádí funkce `run_all()`, která přijímá všechny aktivní pravidla, všechny definované proměnné v příchozích datech a všechny akce vydefinované v systému.

Princip vyhodnocování pravidel je zobrazen na obrázku 4.3. Postup vyhodnocování je analogický ke kódu uvedeném ve výpisku 4.5. Jakmile jsou rule-based enginu předána všechna potřebná data, dochází ke kontrole vstupních podmínek. V případě, že je podmínka u konkrétního triggeru splněna (podmínka v hodnotě `true`), pravidlo je aktivováno a tím dojde k vykonání příslušné akce.



Obr. 4.3: Princip vykonávání pravidel v jádru rule-based enginu.

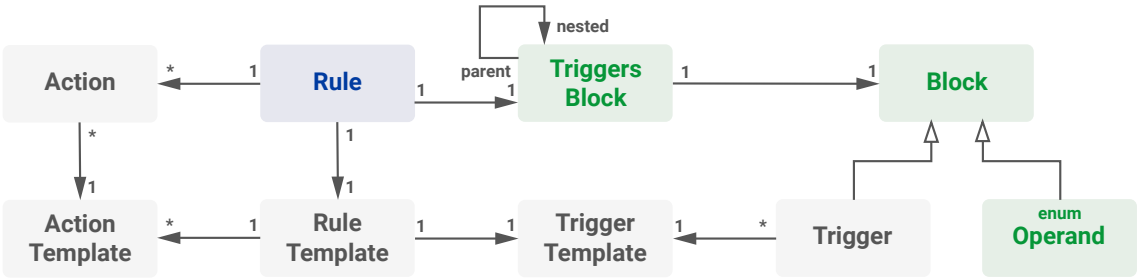
4.4 Model databáze

V databázi jsou uložena všechna data o triggerech, akcích a pravidlech. Zvolena byla databáze *MongoDB* a rule-based engine je s databází propojen prostřednictvím knihovny *djongo*, která zajišťuje komunikaci mezi databází a REST API systému. *MongoDB* je databáze založená na dokumentovém modelu. Data jsou uložena v dokumentech typu `json`, což umožňuje každému dokumentu mít vlastní strukturu.

Tento typ databáze byl zvolen na základě struktury ukládaných dat. Každé pravidlo má vydefinované atributy, z nichž jsou některé strukturované – pro každou ukládanou entitu v dokumentu jednotné. Těmito strukturovanými atributy v případě entity pravidla jsou:

- automaticky vytvořené jméno pravidla,
- časová značka vytvoření pravidla,
- dvoustavová hodnota indikující, zdali je pravidlo aktivní či nikoliv,
- atribut ukládající časovou značku, kdy bylo pravidlo naposledy aktivováno,
- pole uživatelem vydefinovaných akcí.

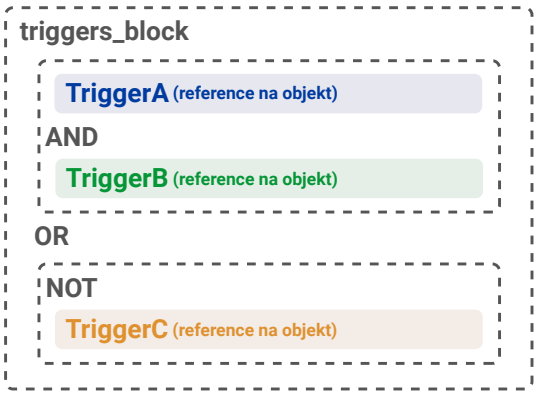
Nestrukturovaným atributem v entitě pravidla je `triggers_block`. Do bloku je ukládána celá spouštěcí podmínka, kterou může být logická funkce podobná funkci zobrazené ve výpisku 4.2 nebo jen jeden samostatný **Trigger**. Tím, že v tomto atributu je ukládán spouštěcí blok, který není strukturovaný, je nutné použít typ dokumentové databáze, která uložení takového bloku umožní.



Obr. 4.4: Diagram datového modelu databáze

Na obrázku 4.4 je zobrazen zjednodušený datový model databáze popisující základní princip vztahů mezi jednotlivými entitami systému. Hlavní entitou je pravidlo **Rule**, které může mít právě jednu šablonu **RuleTemplate**. Tato šablona obsahuje jeden spouštěč **Trigger** uložený v atributu `triggers_block` a pole akcí `actions`[] obsahujících odkazy na jednotlivé entity **Action**. Z modelu lze odvodit, že jedna akce **Action** má vždy právě jednu šablonu **ActionTemplate**. To stejné platí i pro entitu **Trigger**, který se odkazuje na šablonu **TriggerTemplate**, z níž byl vytvořen.

V případě, že v entitě `triggers_block` není uložen odkaz pouze na jeden spouštěč **Trigger** aktivující pravidlo, ale spouštěcím blokem je logická operace, tak je tento blok vytvořen pomocí objektu **Block**, z něhož dědí entita **Trigger** a **Operand**. Postupným zanořováním jednotlivých bloků se vytváří struktura atributu `triggers_block`.



Obr. 4.5: Ukázka struktury atributu `triggers_block`

Struktura bloku logické funkce *((a AND b) OR (NOT c))* je ilustrována na obrázku 4.5. Jednotlivé bloky jsou zanořovány do sebe.

V případě, že entita `triggers_block` se dále nezanořuje, je v ní uložen **Operand**, který odděluje dva nebo jeden (v případě logické operace NOT) spouštěče. Tyto bloky jsou v databázi ukládány separátně, v atributu pravidla je uložen pouze odkaz na objekt uložený v dokumentu databáze.

4.5 Uživatelské rozhraní

Webová aplikace komunikuje s jádrem systému a slouží jako uživatelské rozhraní, ve kterém je koncovému uživateli umožněno vytvářet pravidla a ta následně spravovat. Uživatelské rozhraní bylo vytvořeno jak v low-code platformě Outsystems, která reprezentuje systém umožňující rychlou a snadnou implementaci, tak pomocí frameworku Angular, který naopak dovoluje do vytvářené aplikace vývojáři implementovat low-code/no-code prvky, pomocí kterých koncový uživatel snadněji vydefinuje vlastní pravidla. Oba způsoby vývoje aplikací ukazují různé možnosti použití low-code/no-code.

4.5.1 Uživatelské rozhraní v low-code platformě Outsystems

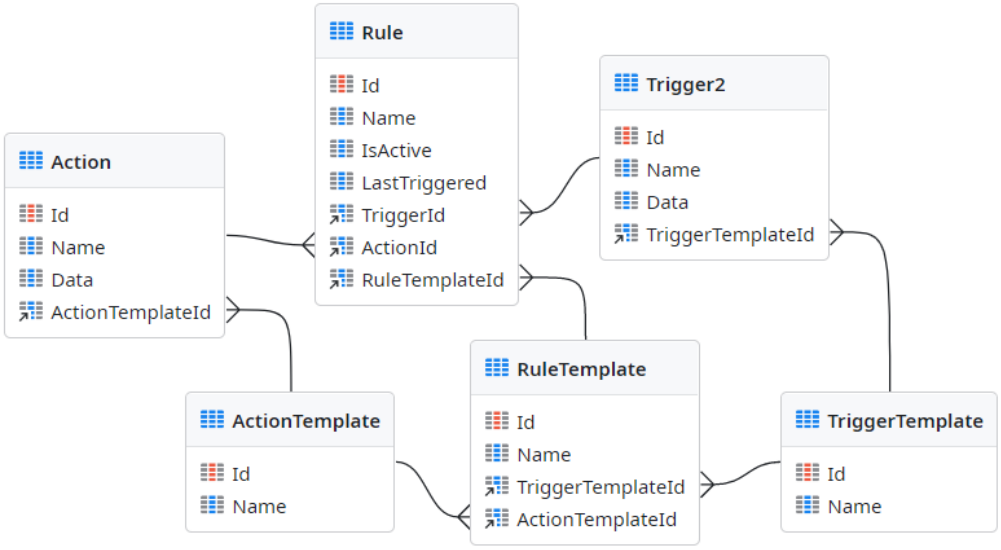
Low-code platforma Outsystems vyžaduje software, který je nutný nainstalovat do počítače, a následně je přístup do aplikace umožněn pouze v případě, že je uživa-

tel zaregistrovaný. Podrobnější informace a princip funkčnosti jednotlivých funkčních bloků této platformy je popsán v sekci 3.1.3.

V platformě je vytvořeno uživatelské rozhraní, v němž si uživatel vydefinuje vlastní pravidla, která jsou následně vyhodnocována rule-based enginem. Samotné jádro systému (samostatně funkční rule-based engine) je v této platformě obtížné vytvořit. Přestože platforma umožňuje instalaci komponentů potřebných pro komunikaci s *MongoDB* databází a Websockets API pro navázání spojení s již existujícím RTLS, samotná implementace balíčků nefungovala zcela správně. Tyto balíčky jsou vyvářeny komunitou, která přispívá k rozšíření možností platformy.

Datový model

V platformě je vytvořen jednoduchý datový model zobrazený na obrázku 4.6. Model uvažuje pouze s jedním triggerem aktivujícím pravidlo v případě, že jeho hodnota nabude stavu `true`, a jednou akcí, jež se následně vykoná. Data jsou ukládána v platformě Outsystems, která má zabudovanou lokální databázi. V platformě lze tuto databázi v grafickém uživatelském prostředí naplnit daty, čehož bylo využito při importu vydefinovaných šablon pro triggery a akce.

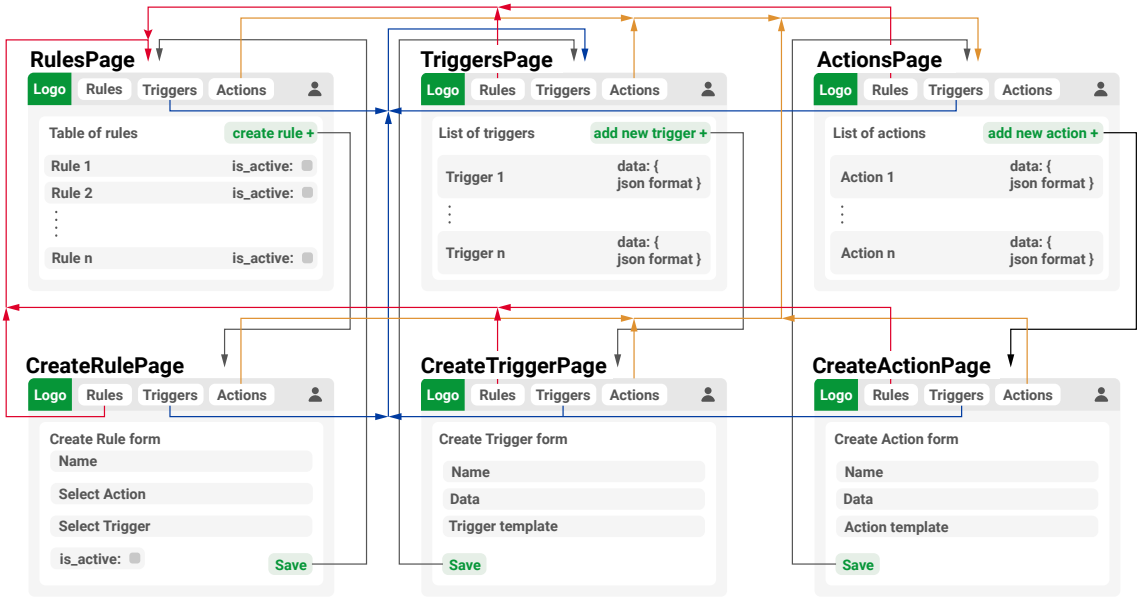


Obr. 4.6: Zjednodušený datový model v platformě Outsystems

Navigace mezi stránkami aplikace

Webová aplikace se skládá celkem z šesti stránek, které jsou vzájemně propojeny odkazy. Stránky aplikace mají stejné (sdílené) navigační menu umístěné na horní liště.

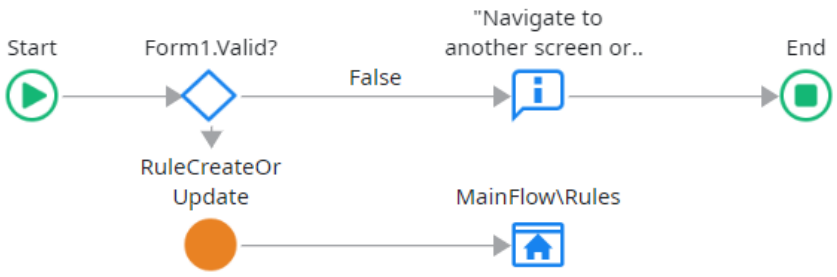
Navigační menu bylo automaticky vygenerováno platformou na základě vytvořeného datového modelu. Navigace mezi stránkami je zobrazena ve schématu 4.7.



Obr. 4.7: Navigace mezi stránkami v platformě Outsystems

Každá z entit (**Rule**, **Trigger** a **Action**) má vlastní stránku, na níž jsou všechny uživatelem vytvořené entity zobrazeny. Pro každou z entit je taktéž vygenerovaná stránka, v níž může uživatel vytvářet nový objekt. Pro entitu **Trigger** a **Action** lze na těchto stránkách na základě předem vytvořených šablon uložených v databázi definovat nové objekty. Analogicky lze na stránce `RuleDetail` vytvářet nové pravidlo složené z jednoho triggeru a jedné akce, které je nutné mít již předem vytvořené.

Odkaz vyznačený v obrázku 4.7 černou barvou zajišťuje přesměrování ze stránky `RuleDetail` na stránku `Rules` a zároveň tlačítko slouží k uložení uživatelem vydefinovaného pravidla.



Obr. 4.8: Tok zajišťující uložení pravidla

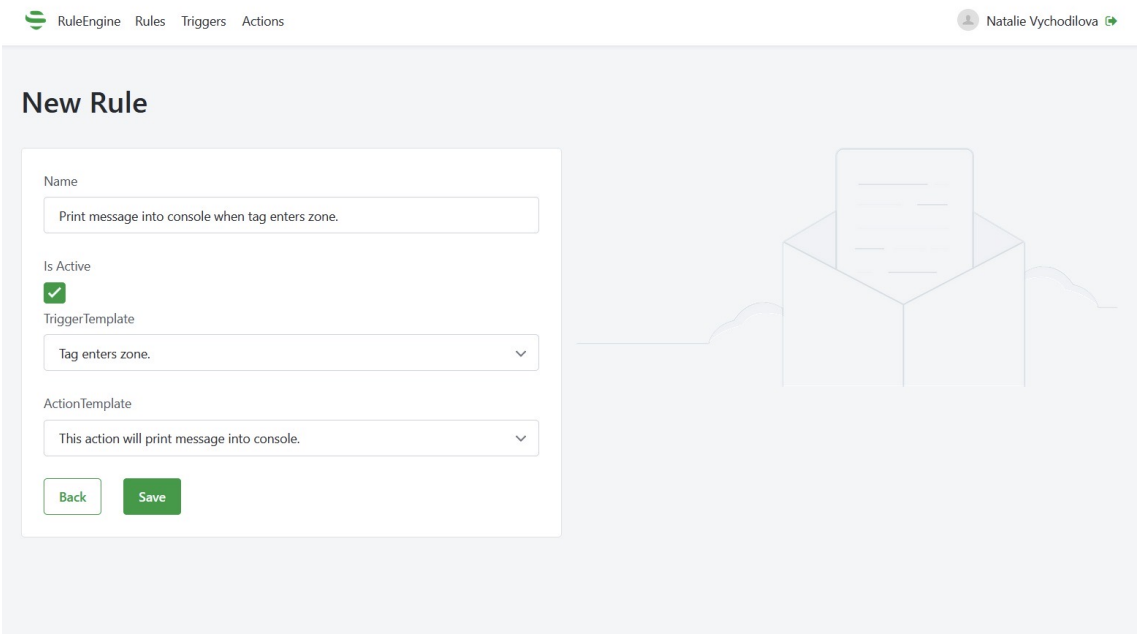
Datový tok zajišťující tuto funkci je zobrazen ve schématu 4.8. Logický tok prvně ověří, zdali jsou data zadaná uživatelem validní. V případě, že jsou, je zavolaná další

vytvořená funkce, která uloží pravidlo do databáze. Následně je uživatel přesměrován na stránku `Rules`. Pokud data nejsou validní, je uživatel taktéž automaticky přesměrován na stránku `Rules` a je upozorněn, že pravidlo nebylo možné vytvořit.

Vytváření pravidel v uživatelském rozhraní

Pro vytvoření nového pravidla slouží stránka `RuleDetail`, která zároveň dovoluje uživateli editovat a upravovat již existující pravidla. Vzhled této stránky je zobrazen na obrázku 4.9. Vzhled (stylování) stránky byl vygenerován pomocí low-code platformy. Obsah stránky byl vytvořen pomocí drag-and-drop elementů. Formulář pro vytvoření pravidel obsahuje:

- okénko `input` určené pro zadání jména pravidla,
- zaškrtávací položku `checkbox` indikující zdali má být pravidlo po vytvoření v aktivním či neaktivním stavu,
- dvě výběrová menu `select` pro akci a trigger.



Obr. 4.9: Stránka určená pro vytváření pravidla v platformě OutSystems

Formulář pro vytváření pravidel je relativně prostý, ale svou funkci plní dostatečně. Platforma Outsystems (obecně všechny low-code/no-code platformy) neumožňují vytvářet složitější a pro specifickou implementaci vhodnější formuláře. Důvodem je, že do platforem nelze instalovat balíčky (knihovny), které vývojáři dovolí využít vlastní low-code/no-code implementace. Možnost vytváření vhodnějších formulářů je značně omezena.

4.5.2 Uživatelské rozhraní ve frameworku Angular

Webová aplikace vytvořená pomocí frameworku Angular slouží jako rozhraní s koncovým uživatelem. Aplikace využívá low-code/no-code implementaci, čímž koncovému uživateli, který nemá žádné programátorské znalosti, dovoluje vydefinovat požadovaná pravidla nad určitým procesem.

Servisní komponenty aplikace

Webová aplikace komunikuje s jádrem rule-based enginu prostřednictvím REST API, na které se dotazuje v případě získávání nebo odesílání dat. Zároveň je pomocí stejného konektoru propojena se Sewio Demo RTLS, ze kterého přebírá informace o existujících budovách, zónách a objektech, jež se v nich pohybují. K propojení klientské části aplikace se dvěma oddělenými systémy slouží servisní komponenty.

Výpis 4.6: Funkce servisní komponenty

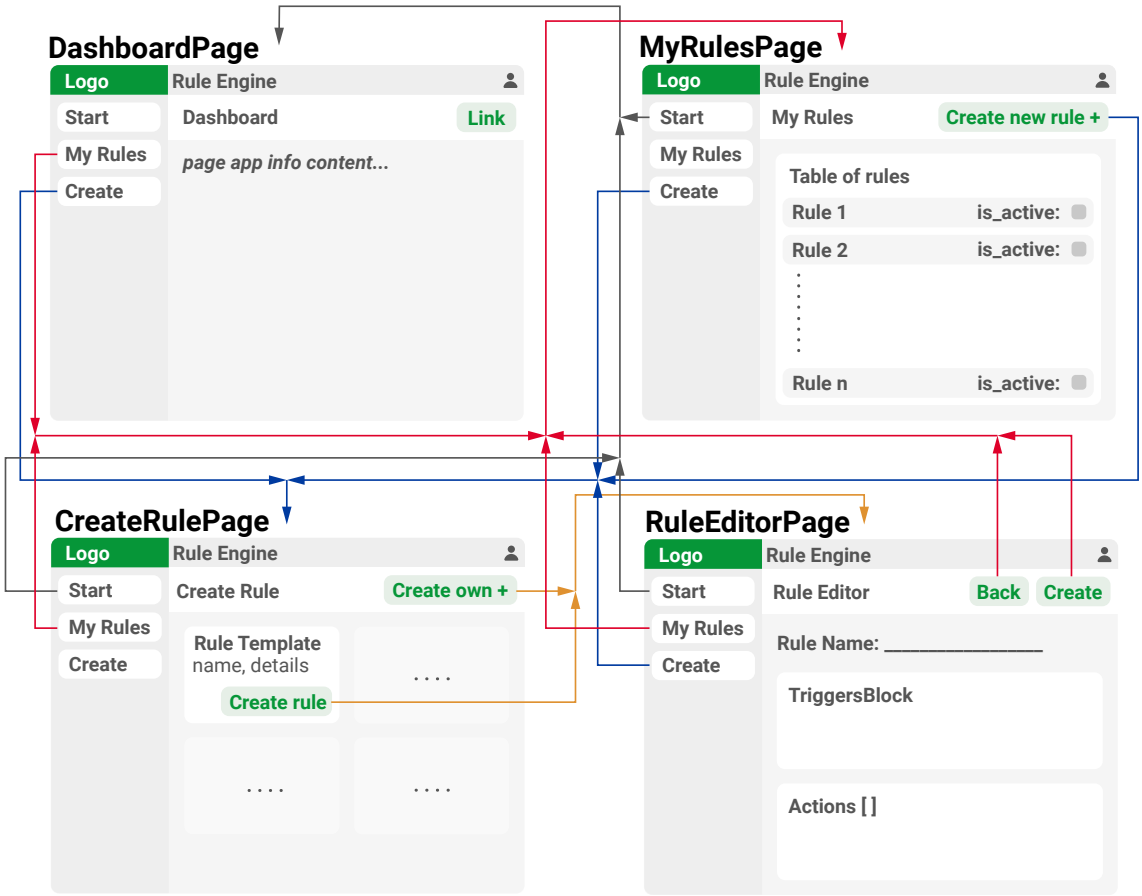
```
export class SewioApiService {
  private apiUrl = environment.sewioApiUrl;
  private apiKey = environment.sewioApiKey;
  constructor(private http: HttpClient) {}

  public getZones(id: number, plan: string): Observable<any> {
    const url=`${this.apiUrl}/buildings/${id}/plans/${plan}/zones`;
    const headers = new HttpHeaders().set('X-ApiKey', this.apiKey);
    return this.http.get(url, { headers });
  }
}
```

Ukázka funkce `getZones()` dotazující se na všechny zóny v konkrétní budově je zobrazena ve výpisku 4.6. Servisní komponenta má vytvořené jednotlivé funkce, v nichž jsou vydefinovány URL adresy. Vzhledem k tomu, že se aplikace dotazuje na aplikaci třetí strany, v hlavičce dotazu musí být uveden API klíč. Obdobně jako dotazování na Demo RTLS se aplikace dotazuje i na REST API rule-based enginu, se kterým komunikuje prostřednictvím HTTP metod.

Servisní komponenty slouží k propojení s jinými částmi systémů, ale mohou být užitečné i pro komunikaci ve webové aplikaci mezi jednotlivými stránkami (komponentami). V aplikaci je tohoto principu využito mezi stránkami `create` a `edit`. V případě, že chce uživatel vytvořit pravidlo pomocí zvolené šablony, jsou tato data šablony přenesena do komponenty `edit`. Navigace mezi těmito stránkami je zobrazena na obrázku 4.10.

Navigace mezi stránkami aplikace

Webová aplikace slouží jako rozhraní s koncovým uživatelem. Jedná se o responzivní webovou aplikaci slouženou z několika stránek, které jsou vzájemně propojeny odkazy. Každá stránka (komponenta) obsahuje navigační menu umístěné na levém panelu, které je pro každou komponentu stejné. Toto navigační menu dovoluje uživateli přecházet na jednotlivé stránky. Na obrázku 4.10 je zobrazena navigace mezi jednotlivými okny aplikace.

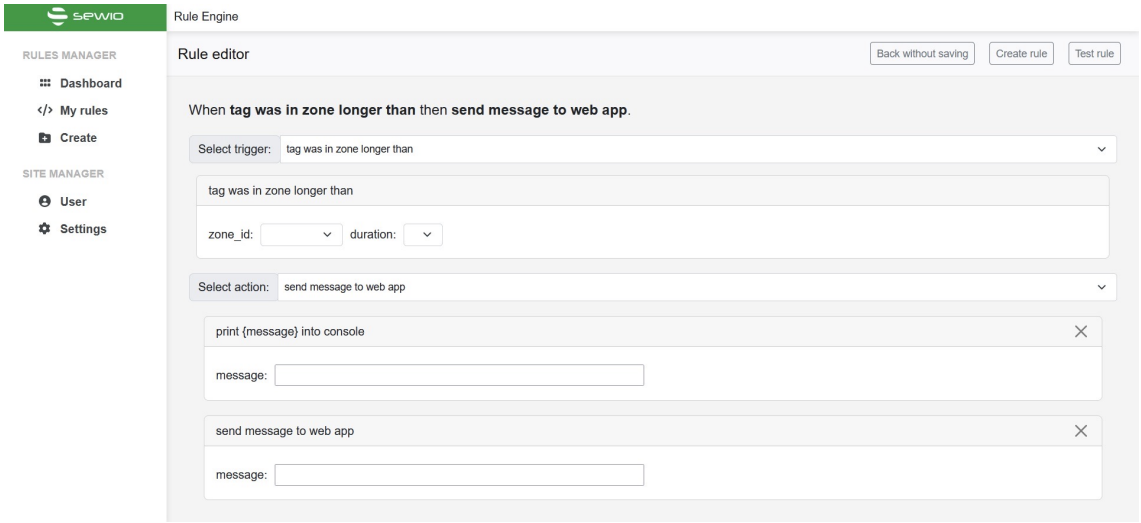


Obr. 4.10: Navigace mezi stránkami

Domovskou (úvodní) stránkou je komponenta `start`. Na stránce `rules` jsou zobrazena všechna existující pravidla, která uživatel může spravovat (například přepínat jejich stav do aktivního). Stránka `create` slouží k výběru šablony pro vytvoření pravidla. Koncový uživatel si může zvolit z předem vydefinovaných šablon nebo vytvořit vlastní pravidlo. V obou případech je uživatel přesměrován na stránku `edit`, v níž definuje vlastní pravidla, upravuje pravidla a nebo doplňuje informace k pravidlům vytvořených z šablon.

Vytváření pravidel v uživatelském rozhraní

K vytváření pravidel slouží stránka `editor`, která je navržena tak, aby v ní uživatel zvládl snadno a jednoduše vydefinovat pravidla. Stránka má pro koncového uživatele představovat low-code/no-code implementaci, která bude vhodnou grafickou formou nahrazovat část kódu, který rule-based engine vyhodnocuje. Vzhled stránky je zobrazen na obrázku 4.11.



Obr. 4.11: Low-code/no-code stránka pro vytváření pravidel

Na levém boku je navigační menu a napravo je hlavní část stránky. Každé vytvářené pravidlo se skládá ze dvou hlavních částí: **akcí** a **triggerů**. Obě tyto části jsou na stránce zobrazeny ve dvou blocích, do kterých uživatel vkládá dodatečné informace k jednotlivým objektům. Uživatel může vložit více akcí, ale pouze jeden trigger. I přes to, že samotný rule-based engine zvládne vyhodnocovat i náročnější logické operace, webová aplikace vytvářet zanořované podmínky neumožňuje.

Nad bloky je zobrazen název pravidla, který se vytváří automaticky v závislosti na jednotlivých akcích a triggerech, které uživatel zvolil. Díky tomuto řešení je předcházeno duplikování stejných pravidel, neboť každé pravidlo musí mít unikátní název. V případě, že již existuje pravidlo se stejným názvem, tak již bylo vytvořeno a znovu se do databáze neukládá.

Šablony akcí a triggerů uživatel nikdy nevytváří. Tyto bloky jsou předem definovány v rule-based enginu, který je vyhodnocuje. Uživatel pouze těmto předvytvořeným šablonám definuje parametry, které jsou ukládány separátně do jednotlivých entit akcí a triggerů. V případě, že již existuje kopie šablony se stejnými parametry (ať už se jedná o akci nebo trigger), není znovu ukládána, ale do pravidla se uloží odkaz na existující entitu. Tím je zamezeno duplikování stejných bloků.

Závěr

Cílem bakalářské práce bylo navrhnout softwarový systém včetně přívětivého uživatelského rozhraní určeného pro vytváření pravidel nad pozičními daty.

V první kapitole této práce byly popsány vybrané procesní problémy. Byly uvedeny jednotlivé typy a příklady procesních problémů, se kterými se lze setkat ve výrobních a logistických procesech. Z těchto problémů byly vyvozeny důsledky, z nichž následně vyplynula potřeba vytvořit systém umožňující nad těmito procesy vytvářet pravidla. Ve druhé části první kapitoly byla stručně popsána technologie UWB a RTLS systémy včetně lokalizačního systému společnosti Sewio.

Druhá kapitola práce popisovala prostředky pro vytváření webových aplikací. Zaměřovala se na popis základních technologií a na rozhraní aplikace.

Třetí kapitola této práce se zaměřovala analýzou dostupných low-code/no-code frameworků a platforem, pomocí kterých byl následně vytvářený celý systém. V této kapitole byly popsány vybrané low-code/no-code platformy a porovnány na základě několika parametrů, kterými byla funkcionalita platformy a uživatelské rozhraní. Na konci kapitoly byly uvedeny webové aplikace, které používají low-code/no-code implementaci.

V poslední části této práce byly popsány konkrétní požadavky na navrhovaný systém včetně uživatelského rozhraní. Byly zmíněny důležité parametry, které systém obsahuje a se kterými pracuje. V této části byla popsána navržená struktura pravidel, které systém uživateli umožňuje vytvářet a následně je vyhodnocovat. Dále byl popsán základní model systému, části systému a databáze, jak pro vývoj v low-code/no-code platformě Outsystems, tak pro vývoj ve frameworku Angular. Uživatelské rozhraní pro obě implementace bylo popsáno v poslední části této kapitoly.

Výstupem bakalářské práce je plně funkční rule-based engine, kompletně navržené uživatelské rozhraní v low-code/no-code platformě Outsystems a webová aplikace ve frameworku Angular využívající low-code/no-code prvků.

Dalšími kroky pro vývoj rule-based enginu je vytvoření nového přívětivějšího uživatelského rozhraní, které bude podporovat vydefinování pravidel se složitější strukturou. Dále vylepšení samotného systému vyhodnocujícího pravidla s důrazem na rychlost vykonávání pravidel a doplnění REST API o další metody a funkce, které vylepší komunikaci mezi jednotlivými částmi systému.

Literatura

[1] ALMADA-LOBO, Francisco. The Industry 4.0 revolution and the future of Manufacturing Execution Systems (MES). *Journal of Innovation Management* [online]. 2016, **3**(4), 16-21 [cit. 2022-11-14]. ISSN 2183-0606. Dostupné z: doi:10.24840/2183-0606_003.004_0003

[2] ZAFARI, Faheem, Athanasios GKELIAS a Kin K. LEUNG. *A Survey of Indoor Localization Systems and Technologies* [online]. 2019, **21**(3), 2568-2599 [cit. 2022-11-14]. ISSN 1553-877X. Dostupné z: doi:10.1109/COMST.2019.2911558

[3] BELLAVISTA, Paolo, Nicola BICOCCHI, Mattia FOGLI, Carlo GIANNELLI, Marco MAMEI a Marco PICONE. Requirements and design patterns for adaptive, autonomous, and context-aware digital twins in industry 4.0 digital factories. *Computers in Industry* [online]. 2023, **149**(1), 1 [cit. 2023-05-12]. ISSN 01663615. Dostupné z: doi:10.1016/j.compind.2023.103918

[4] *802.15.4z-2020 - IEEE Standard for Low-Rate Wireless Networks–Amendment 1: Enhanced Ultra Wideband (UWB) Physical Layers (PHYs) and Associated Ranging Techniques* [online]. 1. IEEE, 2020 [cit. 2022-12-08]. ISBN 978-1-5044-6798-8. Dostupné z: https://ieeexplore.ieee.org/document/9179124

[5] RAHAYU, Yusnita, Tharek Abd. RAHMAN, Razali NGAH a P.S. HALL. Ultra wideband technology and its applications. *2008 5th IFIP International Conference on Wireless and Optical Communications Networks (WOCN '08)*. IEEE, 2008, 2008, 1-5. ISBN 978-1-4244-1979-1. Dostupné z: doi:10.1109/WOCN.2008.4542537

[6] UWB Technology. In: *Kinexon* [online]. [cit. 2022-12-07]. Dostupné z: https://kinexon.com/uwb-technology/

[7] What's the deal with Ultra Wideband? *BMW* [online]. 2022 [cit. 2022-11-07]. Dostupné z: https://www.bmw.com/en/innovation/bmw-digital-key-plus-ultra-wideband.html

[8] Pozyx enables self-driving cars to park underground with UWB technology. *Agoria* [online]. 2021, 4 Jun 2021 [cit. 2022-11-07]. Dostupné z: https://www.agoria.be/en/business-clusters/mobility-vehicle-technology/smart-sustainable-mobility/mobility/pozyx-enables-self-driving-cars-to-park-underground-with-uwb-technology

[9] SIMATIC RTLS — Real-time locating system. *Siemens* [online]. [cit. 2022-11-14]. Dostupné z:

https://new.siemens.com/us/en/products/automation/industrial-identification/simatic-rtls.html

[10] THIEDE, Sebastian, Brendan SULLIVAN, Roy DAMGRAVE a Eric LUTTERS. Real-time locating systems (RTLS) in future factories: technology review, morphology and application potentials. *Procedia CIRP*. 2021, **104**, 671-676. ISSN 22128271. Dostupné z: doi:10.1016/j.procir.2021.11.113

[11] SLOVAK, Juraj, Pavol VASEK, Matej SIMOVEC, Markus MELICHER a Dana SISMISOVA. RTLS tracking of material flow in order to reveal weak spots in production process. *2019 22nd International Conference on Process Control (PC19)*. IEEE, 2019, 2019, 234-238. ISBN 978-1-7281-3758-2. Dostupné z: doi:10.1109/PC.2019.8815220

[12] GLADYSZ, Bartlomiej, Krzysztof SANTAREK a Cezary LYSIAK. Dynamic Spaghetti Diagrams. A Case Study of Pilot RTLS Implementation. *Intelligent Systems in Production Engineering and Maintenance — ISPEM 2017* [online]. Cham: Springer International Publishing, 2018, 2018-08-18, 238-248 [cit. 2022-11-14]. Advances in Intelligent Systems and Computing. ISBN 978-3-319-64464-6. Dostupné z: doi:10.1007/978-3-319-64465-3_24

[13] *Real-time location system (RTLS) for indoor tracking* [online]. Brno, Czech Republic, 2022 [cit. 2022-11-14]. Dostupné z: https://www.sewio.net

[14] UWB Tags for Indoor Location Tracking. *Real-time location system (RTLS) for indoor tracking* [online]. Brno, Czech Republic, 2022 [cit. 2022-11-14]. Dostupné z: https://www.sewio.net/uwb-tags/

[15] Sewio Public Documentation. *Real-time location system (RTLS) for indoor tracking* [online]. Brno, Česko, 2022 [cit. 2022-11-14]. Dostupné z: https://docs.sewio.net/docs/sewio-public-documentation-1015827.html

[16] Frontend vs Backend. *GeeksforGeeks* [online]. A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh: GeeksforGeeks, 2023, 2023 [cit. 2023-05-12]. Dostupné z: https://www.geeksforgeeks.org/frontend-vs-backend/

[17] TABARÉS, Raúl. HTML5 and the evolution of HTML; tracing the origins of digital platforms. *Technology in Society* [online]. 2021, **65**, 1 [cit. 2023-05-12]. ISSN 0160791X. Dostupné z: doi:10.1016/j.techsoc.2021.101529

[18] *Angular* [online]. USA: Google, 2023 [cit. 2023-05-12]. Dostupné z: https://angular.io

[19] Úvod do API. *Co je API?* [online], 2020 [cit. 2023-05-12]. Dostupné z: https://cojeapi.cz/01-uvod-do-api.html

[20] SANCHIS, Raquel, -scar GARCÍA-PERALES, Francisco FRAILE a Raul POLER. Low-Code as Enabler of Digital Transformation in Manufacturing Industry. *Applied Sciences* [online]. 2020, **10**(1), 2-4 [cit. 2022-11-14]. ISSN 2076-3417. Dostupné z: doi:10.3390/app10010012

[21] OVEREEM, Michiel a Slinger JANSEN. Proposing a Framework for Impact Analysis for Low-Code Development Platforms. *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)* [online]. IEEE, 2021, 2021, 88-97 [cit. 2022-11-14]. ISBN 978-1-6654-2484-4. Dostupné z: doi:10.1109/MODELS-C53483.2021.00020

[22] WASZKOWSKI, Robert. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine* [online]. 2019, **52**(10), 376-381 [cit. 2022-11-14]. ISSN 24058963. Dostupné z: doi:10.1016/j.ifacol.2019.10.060

[23] Co je CRM?. *EWay-CRM* [online]. 2022 [cit. 2022-12-07]. Dostupné z: https://www.eway-crm.com/cs/co-je-crm/

[24] Retool Docs. *Retool* [online]. Retool, 2022 [cit. 2022-11-16]. Dostupné z: https://docs.retool.com/docs

[25] Zoho Creator: Your Life's Work, Powered By Our Life's Work. *Zoho* [online]. 2022 [cit. 2022-11-16]. Dostupné z: https://www.zoho.com/creator/features/

[26] What is Power Apps?. *Microsoft* [online]. 2022 [cit. 2022-11-17]. Dostupné z: https://learn.microsoft.com/en-us/power-apps/powerapps-overview

[27] Proč jsou Power Apps a Microsoft Azure lepší dohromady. *Microsoft* [online]. 2022 [cit. 2022-11-17]. Dostupné z: https://powerapps.microsoft.com/cs-cz/power-apps-and-azure/

[28] *Webcon: The low-code application platform that will transform your business* [online]. 2022 [cit. 2022-11-18]. Dostupné z: https://webcon.com

[29] OutSystems Diagram. In: *OutSystems* [online]. [cit. 2022-12-07]. Dostupné z: https://www.outsystems.com/evaluation-guide/how-does-outsystems-fit-in-an-enterprise-architecture/

[30] MARTINS, Ricardo, Filipe CALDEIRA, Filipe SA, Maryam ABBASI a Pedro MARTINS. An overview on how to develop a low-code application using OutSystems. *2020 International Conference on Smart Technologies*

in Computing, Electrical and Electronics (ICSTCEE) [online]. IEEE, 2020, 2020-10-9, 395-401 [cit. 2022-11-18]. ISBN 978-1-7281-7213-2. Dostupné z: doi:10.1109/ICSTCEE49637.2020.9277404

[31] Waylay IO Documentation. *Waylay* [online]. 2022 [cit. 2022-11-27]. Dostupné z: https://docs-io.waylay.io//

[32] Blockly. *Google for Developers* [online]. -: Google, 2023 [cit. 2023-05-13]. Dostupné z: https://developers.google.com/blockly

[33] Drag and Drop. *Angular Material* [online]. -: Google, 2023, 2023 [cit. 2023-05-13]. Dostupné z: https://material.angular.io/cdk/drag-drop/overview

[34] Power Automate. *Microsoft.com* [online]. 2022 [cit. 2022-11-27]. Dostupné z: https://powerautomate.microsoft.com/cs-cz/

[35] VYCHODILOVÁ, N.; JURÁŇ, R. Rule-based engine for position data: low-code/no-code approach to process logic. *Proceedings I of the 29th Student EEICT 2023*. 2023. ISBN: 978-80-214-6153-6.

Seznam symbolů a zkratek

UWB	Ultra-wideband
OFDM	Orthogonal Frequency Division Multiplexing
GPS	Global Positioning System
Wi-Fi	Wireless Fidelity
RTLS	Real-time locating system
API	Application Programming Interface
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
SQL	Structured Query Language
URL	Uniform Resource Locator
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
CMR	Customer Relationship Management
IoT	Internet of Things

A Obsah elektronické přílohy

Bakalářská práce obsahuje elektronickou přílohu, která se skládá ze čtyř částí.

Ve složce `EngineCore` se nachází zdrojový kód jádra vytvořeného systému. Návod pro spuštění všech jeho částí systému včetně vytvoření databáze, je uveden v dokumentu `navod.md`. Aplikace byla vytvořena v jazyce *Python* pomocí frameworku *Django* (struktura složky není z důvodu složitosti ve výpisu níže uvedena).

Složka `WebAppAngular` obsahuje zdrojový kód webové aplikace vytvořené ve frameworku *Angular* vygenerované pomocí *Nx* (struktura složky není z důvodu složitosti ve výpisu níže uvedena).

Elektronická příloha dále obsahuje adresář `WebAppsScreenshots`, v němž jsou uloženy snímky stránek pro obě vytvořená uživatelská rozhraní.

Soubor `RuleEngine.oml` obsahuje zálohu aplikace vytvořené v platformě *Outsystems* a v souboru `navod.md` je popsán postup pro spuštění všech částí práce. V tomto souboru jsou taktéž uvedeny všechny aplikace ve verzích, které jsou nutné pro spuštění a správnou funkčnost navrženého systému a uživatelských rozhraní.

```
/ ..... kořenový adresář přiloženého archivu
├── EngineCore ..... jádro systému vyhodnocující pravidla
├── navod.md ..... dokument popisující spuštění všech částí systému
├── navod.pdf .... dokument popisující spuštění všech částí systému ve formátu .pdf
├── RuleEngine.oml ..... záloha systému platformy Outsystems
├── WebAppAngular ..... webová aplikace vytvořená ve frameworku Angular
└── WebAppsScreenshots ..... snímky uživatelských rozhraní
    ├── WebAppAngular ..... snímky uživatelského rozhraní Angular
    │   ├── CreatePage.jpeg
    │   ├── RuleEditorPage.jpeg
    │   ├── RulesPage.jpeg
    │   └── StartPage.jpeg
    └── WebAppOutsystems ..... snímky uživatelského rozhraní Outsystems
        ├── ActionCreate.jpeg
        ├── ActionsPage.jpeg
        ├── RuleCreate.jpeg
        ├── RulesPage.jpeg
        ├── TriggerCreate.jpeg
        └── TriggersPage.jpeg
```