UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2024 Petr Jeníček

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Systém pro monitoring senzoru rozsáhlých budov

Bakalářská práce

2024 Petr Jeníček

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

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Petr Jeníček**
Osobní číslo:	**I20305**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Systém pro monitoring senzoru rozsáhlých budov**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce je navrhnout a implementovat prototyp řešení pro monitoring většího množství rozsáhlejších budov (např. hotelové komplexy s lokací po celém světě apod.).
Jádrem bakalářské práce bude vhodně navržená databáze (např. MySQL, PostgreSQL apod.), pro sběr dat v rámci jednotlivých budov je předpokladem použití Raspberry Pi. Pro další přenos dat je předpokladem využití technologie REST API, případně jiného vhodného komunikačního prostředku.
Součástí bakalářské práce je také testování vybraných částí celého systému.

Rozsah pracovní zprávy: **min. 30 normostran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Hernandez M. J. Database Design for Mere Mortals: 25th Anniversary Edition. Addison-Wesley Professional; 4th edition, 2020, 640 pp. ISBN: 978-0136788041.

Richardson L. RESTful Web APIs. O'Reilly Media, 2013, 408 pp. ISBN 978-1449358068. 2. Burke B. RESTful Java with JAX-RS 2.0. O'Reilly Media, 2013, 392 pp. ISBN 978-1449361341.

Vedoucí bakalářské práce: **doc. Ing. Michael Bažant, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **16. prosince 2022**
Termín odevzdání bakalářské práce: **12. května 2023**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2023

Prohlašuji:

Práci s názvem Systém pro monitoring senzoru rozsáhlých budov jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

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

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 10.05.2024

Petr Jeníček v.r.

PODĚKOVÁNÍ

Rád bych vyjádřil své hluboké díky vedoucímu práce doc. Ing. Michaelovi Bažantovi, Ph.D., který mi poskytl cenné vedení, podporu a odborné rady během celého procesu psaní této závěrečné práce. Jeho trpělivost, odborné znalosti a schopnost motivovat mě byly klíčové pro dokončení tohoto projektu.

ANOTACE

Práce se zabývá návrhem a implementací systému pro monitorování hotelů, s důrazem na integraci IoT technologií pro sběr a analýzu dat z čidel umístěných v hotelových pokojích. Popisuje architektury webových služeb a komunikační protokoly vhodné pro IoT, které umožňují efektivní správu a automatizaci v rozsáhlých budovách. Analýza a praktická implementace systému jsou demonstrovány na příkladech databázových schémat, technologických stacků a uživatelských rozhraní, včetně bezpečnostních aspektů a správy uživatelů.

KLÍČOVÁ SLOVA

IoT, monitorování hotelů, webové služby, REST, GraphQL, SOAP, gRPC, databázová schémata, uživatelské rozhraní, bezpečnost dat

TITLE

System for monitoring sensors in large buildings

ANNOTATION

This thesis explores the design and implementation of a hotel monitoring system, emphasizing the integration of IoT technologies for data collection and analysis from sensors located in hotel rooms. It describes web service architectures and communication protocols suitable for IoT, facilitating efficient management and automation in extensive building environments. The analysis and practical implementation of the system are demonstrated through examples of database schemas, technology stacks, and user interfaces, including security aspects and user management.

KEYWORDS

IoT, hotel monitoring, web services, REST, GraphQL, SOAP, gRPC, database schemas, user interface, data security

OBSAH

SEZNAM ILUSTRACÍ

SEZNAM ZKRATEK A ZNAČEK

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CoAP	Constrained Application Protocol
CRUD	Create, Read, Update, Delete
gRPC	Google Remote Procedure Call
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
JPA	Java Persistence API
JDBC	Java Database Connectivity
MQTT	Message Queuing Telemetry Transport
OSI	Open Systems Interconnection model
QoS	Quality of Service
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol
UI	User Interface

ÚVOD

V dnešní době, kdy se technologie neustále vyvíjejí a komplexita systémů neustále roste, se stává klíčovým úkolem zajistit efektivní komunikaci mezi různými aplikacemi a zařízeními v rozsáhlých systémech. Takové systémy mohou zahrnovat věci od průmyslových řídicích systémů až po IoT (Internet věcí) zařízení a softwarové aplikace, které spolupracují na sledování, řízení a optimalizaci různých procesů.

Cílem této práce je proto zkoumat a řešit výzvy spojené se síťovým přenosem dat a komunikací mezi aplikacemi v kontextu rozsáhlých systémů, které vyžadují efektivní a spolehlivou komunikaci mezi svými částmi.

Zároveň se stále častěji využívají aplikace propojené pomocí rozhraní REST API (Representational State Transfer Application Programming Interface), které umožňují efektivní a standardizovanou komunikaci mezi různými softwarovými systémy a zařízeními. V kontextu rozsáhlých systémů mohou REST API sloužit k integraci různých komponent, automatizaci procesů a poskytování užitečných informací správcům a uživatelům.

V rámci této úlohy je také důležité poskytnout stručný popis rozsáhlých systémů, aby se lépe porozumělo kontextu a potřebám, na které se tato práce zaměřuje. Budou se zkoumat klíčové výzvy, které vyplývají z komunikace mezi různými aplikacemi a zařízeními v prostředí rozsáhlých systémů, a hledat efektivní řešení pro tyto problémy.

Následující kapitoly se budou postupně zabývat podrobnějším rozborem problematiky síťového přenosu dat, REST API komunikace mezi aplikacemi a implementačními aspekty s důrazem na optimalizaci komunikace a efektivitu systémů. Tato práce si klade za cíl přispět k lepšímu porozumění a efektivnímu využití moderních technologií v oblasti komunikace v rozsáhlých systémech.

1 PŘEDSTAVENÍ IOT

Moderní síťová komunikace a architektury webových služeb hrají klíčovou roli v integrovaných systémech pro monitorování rozsáhlých budov. Tato část práce se zaměřuje na poskytnutí základního porozumění principům a technologiím, které stojí za těmito koncepty a jejich aplikací v kontextu monitorování budov.

Internet věcí (IoT) popisuje síť fyzických objektů – „věcí“ –, které jsou vybaveny senzory, softwarem a dalšími technologiemi za účelem propojení a výměny dat s jinými zařízeními a systémy přes internet. Tato zařízení sahají od obyčejných domácích předmětů po sofistikované průmyslové nástroje.[1]

1.1 Základy síťové komunikace

Spojení dvou počítačů, z nichž každý může používat různý hardware a provozovat různý software, je jednou z hlavních výzev, které museli tvůrci internetu vyřešit. Bylo zapotřebí použití komunikačních technik, které jsou srozumitelné pro všechny připojené počítače, stejně jako dvě osoby, které vyrůstaly v různých částech světa, mohou potřebovat mluvit společným jazykem, aby se navzájem pochopily.[2]

Síťová komunikace je základem moderních informačních technologií a umožňuje výměnu dat mezi počítači a zařízeními přes různé typy sítí. Tato komunikace se opírá o soubor standardizovaných pravidel a protokolů, které definují, jak mají data cestovat od zdroje k cíli. Základní principy síťové komunikace zahrnují adresaci, směrování, přenos, segmentaci a kontrolu chyb, které jsou nezbytné pro spolehlivý a efektivní přenos informací.

1.1.1 Základní charakteristiky

- Použití protokolů jako je TCP/IP pro řízení přenosu dat.
- Síťové modely jako OSI (Open Systems Interconnection) a TCP/IP model, které definují vrstvy a funkce v síťové komunikaci.
- Mechanismy zabezpečení dat, jako jsou šifrování a autentizace.

1.2 Architektury webových služeb

Architektury webových služeb, jako jsou REST, GraphQL, SOAP, a gRPC, jsou zásadní pro moderní síťovou komunikaci. Každá z těchto architektur přináší specifické charakteristiky, výhody a nevýhody, které je třeba zvážit při jejich aplikaci v konkrétních kontextech. V této sekci se zaměříme na porovnání těchto architektur a prozkoumáme jejich vhodnost pro komunikaci v systémech monitorování budov. To nám umožní lépe pochopit, jaké mají jednotlivé architektury přednosti a omezení při zavádění komunikačních mechanismů pro tento specifický účel.

1.2.1 REST (Representational State Transfer)

REST je architektonický styl navržený pro distribuované systémy, zejména webové služby. Zformulován Royem Fieldingem ve své doktorské práci v roce 2000, REST definuje soubor omezení, které, pokud jsou dodrženy, vedou k vytvoření škálovatelného, spolehlivého a efektivního webového systému. REST používá standardní HTTP metody jako GET, POST,

PUT, DELETE atd. pro komunikaci a manipulaci s reprezentacemi zdrojů, které jsou často ve formátech jako XML nebo JSON.[3]

Základní principy:

- Bezstavovost
- Jednotné rozhraní
- Možnost mezipaměti
- Klient-server architektura

1.2.2 Výhody a nevýhody REST

REST jako architektonický styl přináší různé výhody a vyzývá k určitým kompromisům, které jsou důležité pro jeho implementaci a použití.[4]

Výhody REST:

- Jednoduchost a rozšiřitelnost
- Vysoká interoperabilita mezi různými síťovými systémy
- Efektivní využití síťových volání díky možnosti mezipaměti

Nevýhody REST:

- Potřeba pečlivě navrhnout zdroje a jejich reprezentace
- Může být méně efektivní pro některé typy operací, které vyžadují více komunikace mezi klientem a serverem
- Omezení na standardní HTTP metody může omezit flexibilitu

1.2.3 GraphQL

GraphQL je dotazovací jazyk pro API a běhové prostředí pro vykonávání těchto dotazů na vaše data. Byl vyvinut společností Facebook v roce 2012 a nyní je spravován GraphQL Foundation. GraphQL poskytuje kompletní a srozumitelné popisy dat v API, dává klientům moc požadovat přesně to, co potřebují, a nic víc. Tento přístup umožňuje efektivnější přenos dat a usnadňuje vývoj aplikací tím, že snižuje množství potřebného kódu pro přenos a manipulaci s daty. [5]

Základní charakteristiky:

- Umožňuje klientům specifikovat přesně, která data potřebují.
- Zabraňuje nadbytečnému přenosu dat (over-fetching) a neúplnému přenosu dat (under-fetching).
- Podporuje různé typy dotazů, včetně dotazů, mutací a odběrů.

Výhody a nevýhody GraphQL

GraphQL nabízí řadu výhod ve flexibilitě a efektivnosti, ale má také určité nevýhody. [6]

Výhody GraphQL:

- Vyšší efektivita v komunikaci dat díky dotazům na míru.
- Lepší zkušenosti pro vývojáře díky silné typovosti a introspekci.
- Snížení potřeby častých změn na serverové straně API.

Nevýhody GraphQL:

- Složitost v implementaci, zejména na serverové straně.
- Potenciální problémy s výkonem při špatně navržených dotazech.
- Vyšší náročnost na pochopení a správu oproti tradičním RESTful API.

1.2.4 SOAP (Simple Object Access Protocol)

SOAP je protokol pro výměnu strukturovaných informací v implementaci webových služeb v počítačových sítích. Jeho využití je především v komunikaci mezi aplikacemi přes internet. SOAP umožňuje aplikacím komunikovat mezi sebou bez ohledu na operační systémy a programovací jazyky, díky použití XML jako formátu pro zprávy. Protokol definuje pravidla pro strukturování zpráv, které mohou obsahovat volání procedur nebo zprávy. SOAP podporuje různé přenosové protokoly, ale nejčastěji se používá s HTTP.[7]

Základní charakteristiky:

- Nezávislost na přenosovém médiu
- Standardizovaný formát zpráv
- Podpora transakcí, bezpečnosti a spolehlivosti [8]

Výhody a nevýhody SOAP

SOAP je znám svou pevnou strukturou a spolehlivostí, které přinášejí jak výhody, tak i omezení.

Výhody SOAP:

- Vysoká úroveň bezpečnosti a spolehlivosti
- Standardizace umožňuje interoperabilitu mezi různými systémy
- Podpora komplexních transakcí a provozních procesů

Nevýhody SOAP:

- Vyšší režie a složitost zpráv v porovnání s jinými protokoly
- Může být náročnější na výkon a zdroje
- Složitější pro vývojáře k implementaci a debugování

1.2.5 gRPC

gRPC je moderní open source vysokovýkonný RPC (Remote Procedure Call) framework, který byl původně vyvinut společností Google. Je navržen pro efektivní komunikaci mezi mikroslužbami, podporuje více jazyků a je optimalizován pro nízkou latenci a vysoký propustný výkon. gRPC využívá protokol HTTP/2 pro transport, což umožňuje vylepšené streamování a kontrolu toku. Pro definici API a komunikačních rozhraní používá jazyk Protocol Buffers (ProtoBuf), což je mechanismus pro serializaci strukturovaných dat. [9]

Základní charakteristiky:

Využívá HTTP/2 pro efektivní komunikaci

Používá ProtoBuf pro definici rozhraní a serializaci dat

Podporuje čtyři typy volání: jednoduché RPC, server streaming, client streaming, a bidirectional streaming

Výhody a nevýhody gRPC

gRPC přináší výhody v rychlosti a efektivitě, ale může být náročnější v některých aspektech integrace. [10]

Výhody gRPC:

- Vysoká efektivita a nízká latence díky binární serializaci dat a HTTP/2
- Lepší podpora pro asynchronní komunikaci a streamování
- Jazyková nezávislost díky použití ProtoBuf

Nevýhody gRPC:

- Vyšší náročnost na počáteční nastavení a konfiguraci
- Menší podpora v prohlížečích a omezená kompatibilita s některými síťovými infrastrukturami
- Složitější pro debugování a sledování kvůli binárnímu formátu

1.2.6 Porovnání a použití v kontextu monitorování budov

V oblasti monitorování budov, kde je potřeba zajišťovat správu a koordinaci dat z mnoha senzorů a udržovat synchronizaci mezi různými uzly a centrální správou, je volba vhodné architektury webových služeb klíčová. Pro tento projekt byla vybrána architektura REST z důvodu jejích specifických vlastností, které nejlépe vyhovují požadavkům na jednoduchost, širokou podporu.

REST je známý svou jednoduchostí a snadnou škálovatelností. To je zvláště důležité v prostředích, kde systém musí efektivně komunikovat s velkým množstvím senzorů a různých zařízení. RESTful služby jsou bez stavové, což znamená, že každý požadavek od klienta na server musí obsahovat všechny informace potřebné k jeho zpracování. To redukuje potřebu udržování složitého stavu na serveru. Jelikož se hotel autorizuje pomocí svého unikátního ID nemusí se jednat o jeden server a je možnost škálovat do šířky.

Důvody zvolení REST pro tento projekt:

Snadná integrace a jednoduchost implementace: REST využívá standardní HTTP metody, což usnadňuje integraci s existujícími webovými technologiemi a nástroji. Tím se zjednodušuje vývoj a údržba systému.

Široká podpora a kompatibilita: Vzhledem k rozšířenému užívání RESTu ve webovém vývoji existuje mnoho knihoven a nástrojů, které podporují vývoj a testování RESTful API. Toto zajišťuje lepší kompatibilitu s různými klienty a serverovými platformami.

Efektivní pro základní operace: REST je velmi efektivní pro standardní operace jako získání stavu senzorů a aktualizace konfigurací, což jsou klíčové funkce pro systémy monitorování budov. Díky svému stateless charakteru je možné jednoduše škálovat RESTful služby podle potřeb projektu.

Nevýhody a jejich řešení: I když REST může být neefektivní pro komplexní operace nebo systémy vyžadující častou synchronizaci stavů v reálném čase, tyto nevýhody nejsou pro tento konkrétní projekt překážkou, jelikož prioritou je bezpečná bezztrátová komunikace.

1.3 Protokoly pro Internet věcí (IoT)

Internet věcí (IoT) transformuje způsob, jakým jsou propojena zařízení a aplikace, což má zásadní význam pro efektivní monitorování a řízení rozsáhlých budovních systémů. Díky IoT je možné sbírat data z různých senzorů umístěných po celé budově v reálném čase, což umožňuje okamžitou reakci na změny prostředí a optimalizaci provozních procesů. Klíčovou roli v této infrastruktuře hrají protokoly IoT, které definují pravidla pro komunikaci mezi zařízeními. Tyto protokoly musí být nejen efektivní a spolehlivé, ale také dostatečně bezpečné a škálovatelné, aby vyhověly požadavkům rozsáhlých budovních systémů. [11]

1.3.1 Nejčastěji používané protokoly IoT:

MQTT (Message Queuing Telemetry Transport)

- Charakteristika: Lehký protokol určený pro situace s omezenou šířkou pásma a nestabilním připojením. [12]
- Využití: Ideální pro systémy, kde je potřeba efektivně přenášet zprávy od zařízení k serveru (telemetrie).
- Výhody: Nízká šířka pásma a energetická náročnost; podpora různých úrovní kvality služeb (QoS).
- Nevýhody: Vyžaduje spojení s brokerem, což může vytvářet úzká místa v komunikaci.

CoAP (Constrained Application Protocol)

- Charakteristika: Protokol navržený speciálně pro jednoduché zařízení, podobně jako HTTP, ale optimalizovaný pro IoT. [13]
- Využití: Vhodný pro zařízení s velmi omezenými zdroji a pro situace vyžadující integraci s webovými technologiemi.

- Výhody: Podporuje obousměrnou komunikaci a objevování služeb; efektivní v manipulaci s daty XML a JSON.
- Nevýhody: Méně rozšířený než MQTT, může být složitější na implementaci.

HTTP/HTTPS

- Charakteristika: Univerzální protokol pro přenos hypertextu, využívaný v IoT pro jeho rozšířenost a podporu. [14][15]
- Využití: Používá se pro zasílání dat do cloudových aplikací, kde je nutná vyšší míra interakce nebo integrace s internetovými službami.
- Výhody: Snadná integrace, široká podpora a bezpečnostní mechanismy (HTTPS).
- Nevýhody: Vyšší režie, což může být problematické pro zařízení s omezenými zdroji.

AMQP (Advanced Message Queuing Protocol)

- Charakteristika: Komplexní, spolehlivý protokol vhodný pro korporátní prostředí. [16][17]
- Využití: Preferovaný ve velkých podnikových aplikacích, kde jsou důležité robustnost a zabezpečení přenosu.
- Výhody: Podporuje pokročilé funkce zpracování zpráv, jako jsou trvalost, transakce a potvrzení.
- Nevýhody: Vyšší náročnost na zdroje a složitost. Aplikace v monitorování rozsáhlých budov

1.3.2 Význam protokolů pro IoT v monitorování rozsáhlých budov

V kontextu rozsáhlých budovních systémů mohou být tyto protokoly aplikovány pro různé účely, od základního monitorování teploty a vlhkosti až po složitější aplikace, jako jsou bezpečnostní systémy a automatizace budov. Volba protokolu závisí na specifických požadavcích projektu, včetně požadované škálovatelnosti, bezpečnosti a interoperability s jinými systémy.

1.3.3 Využití protokolů pro IoT v monitorování rozsáhlých budov

V kontextu monitorování rozsáhlých budov jsou protokoly pro IoT využívány pro širokou škálu aplikací, včetně:

Sběr dat: Protokoly pro IoT umožňují senzorům a dalším zařízením v budovách pravidelně odesílat data o stavu a provozu různých systémů, jako jsou například teplota, vlhkost, osvětlení.

Řízení zařízení: Tyto protokoly umožňují vzdálené řízení a správu zařízení v budovách, což umožňuje automatizaci různých procesů a optimalizaci využití zdrojů.

Komunikace mezi zařízeními: Protokoly pro IoT poskytují mechanismy pro komunikaci mezi různými zařízeními v budovách, což umožňuje integraci a spolupráci mezi různými systémy a komponentami.

Využití protokolů pro IoT v monitorování rozsáhlých budov může přinášet značné výhody v podobě zlepšené efektivity, bezpečnosti a pohodlí uživatelů, a proto je důležité pečlivě vybírat a implementovat ty nejvhodnější protokoly pro konkrétní potřeby a požadavky prostředí.

2 PRAKTICKÁ ČÁST

V této praktické části bude kladen důraz na popis a implementaci konkrétních aplikací, které slouží k monitorování rozsáhlých budov, konkrétně v kontextu hotelového prostředí. Každá z těchto aplikací představuje klíčový krok k dosažení efektivního a spolehlivého monitorování a řízení budovních systémů, a to jak z technického, tak z uživatelského hlediska.

2.1 Seznam požadavků

2.1.1 Funkční požadavky

Funkční požadavky definují konkrétní chování nebo funkce systému. Tyto požadavky specifikují, co systém dělá – tedy jaké úkoly by měl být schopen vykonat, jaké operace umožňuje a jaké služby poskytuje. Tyto požadavky jsou obvykle formulovány jako akce, které může systém provádět na základě vstupů od uživatelů nebo jiných systémů. [18]

1. Čidlová aplikace (Emulace čidel v hotelu)

- FR1: Aplikace musí být schopna emulovat různé typy čidel (teplota, vlhkost, CO2).
- FR2: Musí periodicky odesílat data do ústředny hotelu.
- FR3: Aplikace musí umožnit konfiguraci cílové url, typu čidla, id sensoru a databázové url.
- FR4: Data musí být odeslána s časem naměření.

2. Ústředna hotelu

- FR5: Aplikace musí přijímat data z čidel a ukládat je do databáze.
- FR6: Musí periodicky agregovat přijatá data do stavů podlé místností.
- FR7: Aplikace bude periodicky získávat aktuální data o zařízeních z ústředny hotelů.

3. Ústředna sítě hotelů

- FR8: Aplikace má systém pro správu uživatelů.
- FR9: Uživatelé mohou mít role ADMIN a HOTEL_ADMIN.
- FR10: Aplikace musí umožnit CRUD operace nad hotely, místnostmi, sensory a uživateli.
- FR11: Poskytne uživatelské rozhraní pro zobrazení dat z více hotelů.
- FR12: UI musí umožnit zobrazování grafů teploty, vlhkosti a CO2 pro každý pokoj.

2.1.2 Nefunkční požadavky

Nefunkční požadavky, na rozdíl od funkčních, specifikují atributy systému, jako jsou výkon, bezpečnost, omezení, kvalita a omezení, která definují jak systém funguje. Nefunkční požadavky obvykle ovlivňují celkovou uživatelskou zkušenost, stabilitu, spolehlivost a systémovou efektivitu. [18]

1. Všeobecné

- NFR1: Aplikace musí být implementovány s důrazem na modularitu a rozšiřitelnost.
- NFR2: Data musí být uložena bezpečně.

2. Výkon a spolehlivost

- NFR3: Systém musí být navržen pro vysokou spolehlivost.
- NFR4: Systém musí být schopen škálovat počet čidel a uživatelů.

3. Technologie a rozhraní

- NFR5: Aplikace budou napsané v Java Spring Boot.
- NFR6: Uživatelské rozhraní bude implementováno pomocí Thymeleaf.
- NFR7: UI musí být intuitivní a přístupné pro různé typy uživatelů.

2.2 Přehled existujících řešení

Než se ponoříme do detailů jednotlivých řešení, je důležité mít přehled o celkové architektuře a fungování systému.

Vytvořené aplikace zahrnují:

Čidlo v hotelu: Tato aplikace slouží k emulaci reálných čidel umístěných v místnostech hotelu. Je navržena tak, aby generovala a odesílala data týkající se teploty, vlhkosti nebo osvětlení na základě vstupní konfigurace. Aplikace pravidelně odesílá shromážděná data do ústředny hotelu, kde jsou data zpracována.

Ústředna hotelu: Centrální jednotka hotelu, která zpracovává data z čidel, provádí zpracování dat a bezpečnou bezztrátovou komunikaci po internetu s ústřednou sítě hotelů.

Ústředna sítě hotelů: Centrální jednotka sítě hotelů slouží k zobrazení nashromážděných dat, poskytování informací hotelům a k práci s uživateli, grafy apod.

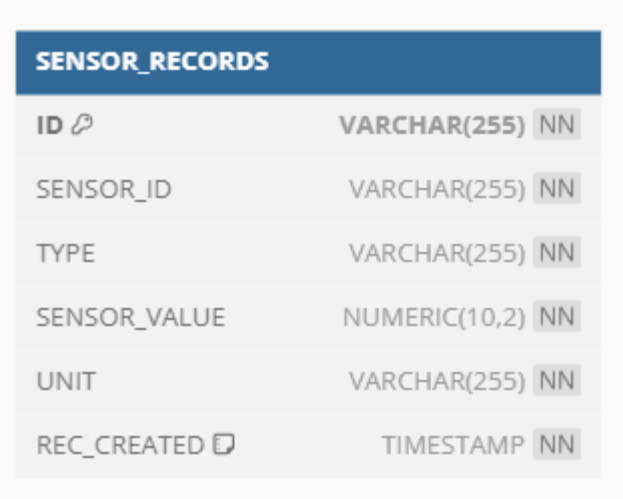
Každá z těchto aplikací má svou specifickou roli a přispívá k efektivnímu a spolehlivému monitorování a řízení hotelových prostorů, a to jak na úrovni jednotlivých pokojů a budov, tak na úrovni celé sítě hotelů.

V následujících sekcích se podrobněji zaměříme na každou z těchto aplikací a provedeme detailní rozbor jejich implementace, funkcí a procesů.

2.3 Čidlová aplikace

Aplikace, která emuluje funkci čidla v místnosti hotelu a je navržena pro generování a odesílání dat o teplotě, vlhkosti, osvětlení a dalších relevantních faktorech do ústředny hotelu. Tato aplikace simuluje reálné čidlo a pravidelně zasílá aktualizované informace, které pomáhají monitorovat a udržovat optimální podmínky pro pohodlí a bezpečnost hostů. Generovaná data jsou důležitá pro efektivní řízení interních systémů hotelu, jako je klimatizace a vytápění, a zajišťují, že všechny prostředí jsou přizpůsobeny aktuálním potřebám hostů a pracovníků. Kromě toho aplikace poskytuje cenné informace pro preventivní údržbu a rychlou reakci na potenciální problémy v rámci hotelového zařízení.

2.3.1 Databázové schéma



Obrázek 1 Databázové schéma čidla hotelu

Pro účely jednoduchosti a demonstrace bylo v tomto projektu implementováno čidlo pomocí databáze PostgreSQL. Toto řešení poskytuje robustní funkcionalitu a rozsáhlé možnosti pro správu dat, což je vhodné pro komplexnější testovací a vývojové scénáře.

Nicméně v reálném nasazení, zejména tam, kde jsou omezení na úložný prostor a výpočetní výkon, by bylo vhodnější použít SQLite. SQLite je lehká souborová databáze, která nevyžaduje běh žádných serverových procesů ani dalších souběžně běžících programů. Díky tomu je SQLite ideální pro zařízení s omezenými systémovými zdroji, protože spotřebovává méně systémových prostředků ve srovnání s PostgreSQL. Toto řešení je tedy vhodnější pro prostředí, kde je důležitá efektivní správa systémových zdrojů a minimalizace nákladů na údržbu. [19]

Relační model

V relačním modelu se zaměřuje na vztahy mezi entitami (tabulkami). V databázovém schématu čidla hotelu není žádný vztah mezi tabulkami, protože tabulka je pouze jedna.

Konceptuální model

Konceptuální model se zaměřuje na vysoce úrovňovou organizaci dat a jejich vztahy. V tomto modelu se nezaměřujeme na detaily implementace, ale spíše na to, jak data souvisí a jaká je mezi nimi logická spojitost.

ID: Jednoznačný identifikátor záznamu čidla, který slouží jako primární klíč.

SENSOR_ID: Identifikátor čidla, který by umožňoval spojit záznamy se senzor tabulkou. Ta v kontextu Čidla hotelu, neexistuje a nejde použít jako unikátní identifikátor, jelikož všechny záznamy mají stejný SENSOR_ID.

TYPE: Typ čidla, který určuje, jakého druhu je senzor (například teplota, vlhkost, osvětlení).

SENSOR_VALUE: Numerická hodnota, která reprezentuje měření provedené čidlem.

UNIT: Jednotka měření hodnoty, například stupně Celsius, procenta vlhkosti, luxy pro osvětlení atd.

REC_CREATED: Časové razítko, kdy byl záznam vytvořen. Toto pole automaticky generuje aktuální časové razítko při vytvoření záznamu.

Fyzický model

Fyzický model popisuje, jak jsou data skutečně uložena na disku nebo v databázovém systému. Tento model zahrnuje specifika, jako jsou datové typy, omezení a indexy použité pro optimalizaci výkonu databáze.

ID: VARCHAR (255), používá se jako primární klíč.

SENSOR_ID: VARCHAR (255), může být indexován pro rychlejší vyhledávání podle čidla.

TYPE: VARCHAR (255), potenciálně indexovaný sloupec pro rychlé filtrování podle typu čidla.

SENSOR_VALUE: NUMERIC (10,2), umožňuje přesné desetinné hodnoty.

UNIT: VARCHAR (255), standardní textový řetězec pro jednotky.

REC_CREATED: TIMESTAMP, s výchozí hodnotou CURRENT_TIMESTAMP, který zaznamenává čas vytvoření záznamu.

2.3.2 Technologie a programovací jazyky

Programovací jazyk

Java: Aplikace je napsána v programovacím jazyce Java, což umožňuje využít robustní ekosystém knihoven a nástrojů. Verze Java 21 je používána pro zajištění kompatibility s nejnovějšími vývojovými trendy a bezpečnostními standardy.

Nástroje pro sestavení

Maven: Pro správu projektu a závislostí je použit Maven, což je nástroj, který podporuje správu životního cyklu aplikace, od kompilace kódu přes jeho testování až po balení a distribuci.

Závislosti a knihovny

Spring Boot: Použití Spring Boot usnadňuje vývoj samostatně spustitelných aplikací na platformě Spring. Aplikace využívá několik modulů Spring Boot:

spring-boot-starter-data-jpa: Pro integraci s databázemi pomocí Java Persistence API, umožňuje jednoduché mapování objektů v Javě na databázové tabulky.

spring-boot-starter-web: Umožňuje vývoj webových aplikací, včetně RESTful služeb.

spring-boot-starter-webflux: Podporuje vývoj asynchronních a neblokujících webových aplikací využívajících model reaktivního programování.

spring-boot-starter-security: Nabízí rozsáhlé možnosti pro zabezpečení aplikace, včetně autentizace a autorizace.

spring-boot-starter-json: Poskytuje podporu pro práci s JSON daty, což je užitečné pro zpracování dat získaných z čidel.

PostgreSQL JDBC Driver: Pro komunikaci s PostgreSQL databází, která je používána pro ukládání dat z čidel.

Liquibase: Nástroj pro správu verzí databázových schémat, který umožňuje snadnou a bezpečnou migraci a úpravu databázových struktur.

Lombok: Knihovna usnadňující vývoj v Javě díky anotacím, které automatizují vytváření běžných metod, jako jsou getters/setters nebo toString.

Spring Boot Test a Spring Security Test: Tyto moduly umožňují efektivní testování komponent aplikace, včetně zabezpečení.

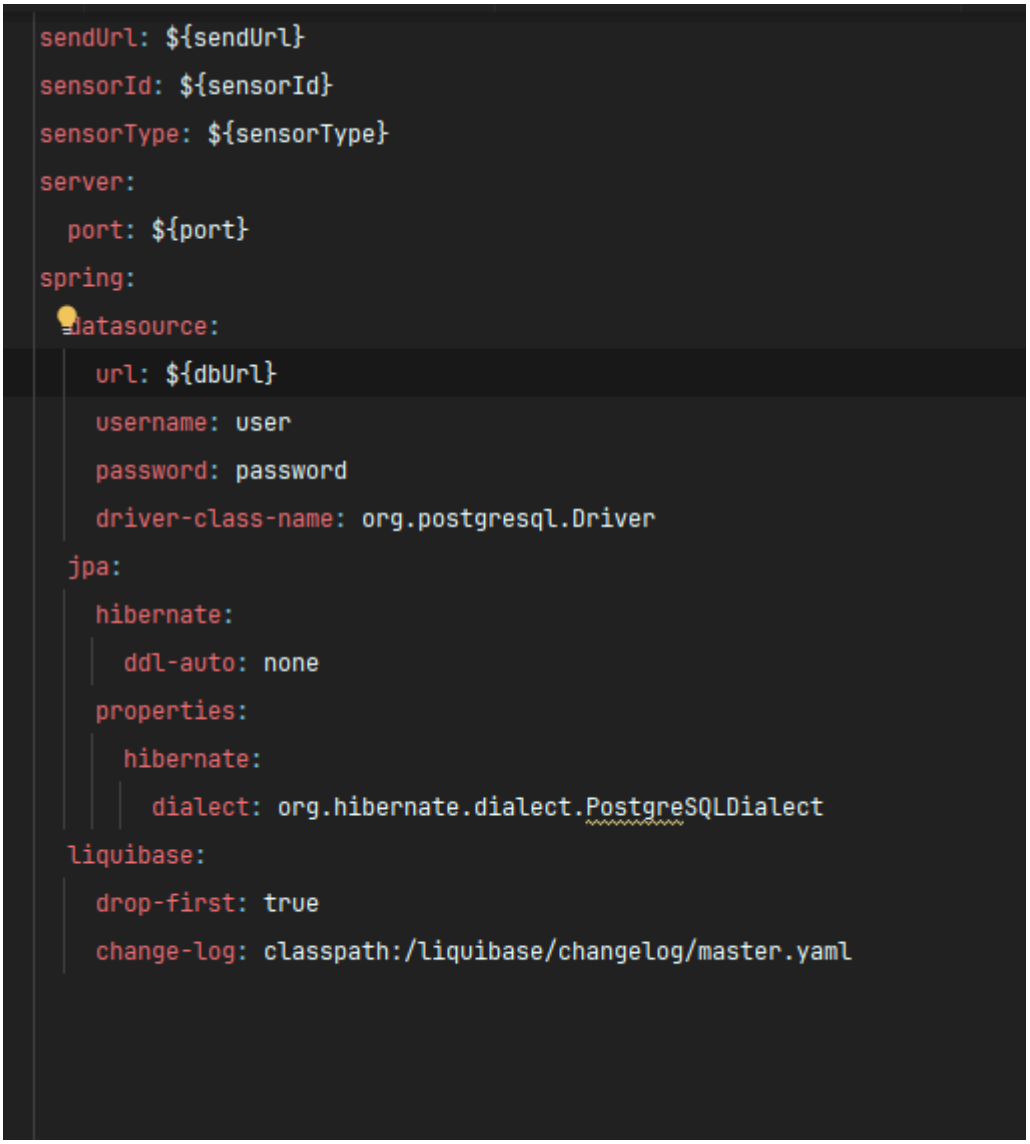
Konfigurace sestavení

Spring Boot Maven Plugin: Tento plugin je konfigurován pro vyřazení knihovny Lombok při finálním sestavení aplikace, aby se předešlo potenciálním problémům s kompilací a runtime.

Tato kombinace technologií a nástrojů poskytuje silný základ pro vývoj a provoz aplikace, která emuluje čidla v hotelu, zpracovává data a zajišťuje jejich přenos do ústřední jednotky hotelu pro další analýzu a využití.

2.3.3 Ukázky kódu s popisky

Konfigurační Soubor YAML pro Čidlovou Aplikaci



Obrázek 2 Application.yaml čidlového řešení

Sensor Konfigurace:

sendUrl: URL adresa, kam aplikace odesílá data z čidel. Hodnota je nastavena přes proměnnou prostředí ${sendUrl}, což umožňuje snadnou změnu bez potřeby zásahu do kódu.

sensorId: Identifikátor čidla, který lze dynamicky nastavit pomocí proměnné prostředí ${sensorId}.

sensorType: Typ čidla, nastavitelný skrze proměnnou prostředí ${sensorType}.

Server Konfigurace:

port: Port, na kterém server poslouchá, definovaný proměnnou prostředí ${port}.

Databázové Nastavení (Spring DataSource):

url: JDBC URL pro připojení k databázi PostgreSQL, nastavitelné přes proměnnou prostředí ${dbUrl}.

username a password: Přihlašovací údaje k databázi, což zvyšuje bezpečnost tím, že hesla a uživatelská jména nejsou hardkódována přímo v konfiguračním souboru.

driver-class-name: Nastavení PostgreSQL JDBC driveru.

JPA a Hibernate Konfigurace:

ddl-auto: Nastavení Hibernate, zda a jak automaticky spravovat databázové schéma. Zde je none, což znamená, že Hibernate nebude automaticky modifikovat schéma.

dialect: Specifikuje dialekt SQL používaný Hibernate, zde org.hibernate.dialect.PostgreSQLDialect, což umožňuje optimalizované SQL dotazy pro PostgreSQL.

Liquibase:

drop-first: Indikuje, zda Liquibase má při startu aplikace nejprve odstranit stávající databázové objekty. Hodnota true je vhodná pro vývojové prostředí.

change-log: Umístění souboru s definicemi změn databáze, které Liquibase používá k aktualizaci schématu.

Tento přístup ke konfiguraci prostřednictvím proměnných prostředí je standardním způsobem pro moderní cloudové a kontejnerizované aplikace, protože umožňuje snadné nasazení a škálování bez nutnosti měnit kód aplikace nebo zveřejňovat citlivé informace.

Metoda pro generování a odesílání dat

```
@Scheduled(cron = "*/10 * * * * *")
@Transactional
public void generateData() {
    SensorRecordEntity generatedData = SensorRecordEntity.builder()
        .sensorId(sensorId)
        .type(type)
        .unit(Arrays.stream(SensorRecordUnitEnum.values()).filter(value → value.getType().equals(type)).findFirst().get())
        .sensorValue(BigDecimal.valueOf(new Random().nextDouble( origin: 5,  bound: 35)))
        .recCreated(OffsetDateTime.now())
        .build();
    List<SensorRecordEntity> dataToSend = new ArrayList<>(sensorRecordRepository.findAll());
    dataToSend.add(generatedData);

    webClient.post() RequestBodyUriSpec
        .uri(ub → ub.path("/sensor-data").build()) RequestBodySpec
        .body(BodyInserters.fromValue(getMappedEntities(dataToSend))) RequestHeadersSpec<capture of ?>
        .retrieve() ResponseSpec
        .toEntity(Void.class) Mono<ResponseEntity<...>>
        .subscribe(
            result → {
                log.info("Data byla úspěšně odeslána. {}", dataToSend);
                sensorRecordRepository.deleteAll();
            },
            error → {
                log.error("Chyba při komunikaci se serverem... zálohuji data: {}", error);
                sensorRecordRepository.save(generatedData);
            }
        );
}
```

Obrázek 3 Generování a odesílání dat čidlového řešení

Na obrázku je zobrazen úryvek zdrojového kódu v jazyce Java, který ukazuje implementaci funkce pro generování a odesílání dat ze senzorů. Následuje podrobný popis kódu s odkazy na čísla řádků pro lepší orientaci:

Anotace a metoda

Řádek 41: Metoda generateData() je naplánována k automatickému spouštění každých 10 sekund pomocí anotace @Scheduled(cron = "*/10 * * * * *").

Řádek 42: Anotace @Transactional zajišťuje, že celý proces generování a odesílání dat bude prováděn v rámci databázové transakce.

Generování dat

Řádky 43-50: Vytvoření nové instance SensorRecordEntity s použitím Builder patternu. Nastavení atributů jako sensorId, type, unit, sensorValue a recCreated. Hodnota sensorValue je generována náhodně (řádek 48) a časové razítko je nastaveno na aktuální čas (řádek 49).

Řádky 51-53: Vytvoření seznamu dataToSend, do kterého je přidáno nově generované data.

Odeslání dat

Řádky 54-67: Využití WebClient pro asynchronní odeslání seznamu dat na server:

Řádek 55: Konfigurace URI pro odeslání dat.

Řádek 56: Přidání těla požadavku s daty k odeslání.

Řádek 57: Spuštění HTTP POST požadavku a očekávání odpovědi.

Řádek 58: Deklarace, že se neočekává žádná odpověď s tělem (Void.class).

Zpracování výsledků

Řádky 59-67: Zpracování výsledků požadavku pomocí metody .subscribe():

Řádky 60-63: V případě úspěchu (proměnná result) se loguje informace o úspěšném odeslání (řádek 61) a všechna data v databázi jsou smazána (řádek 62).

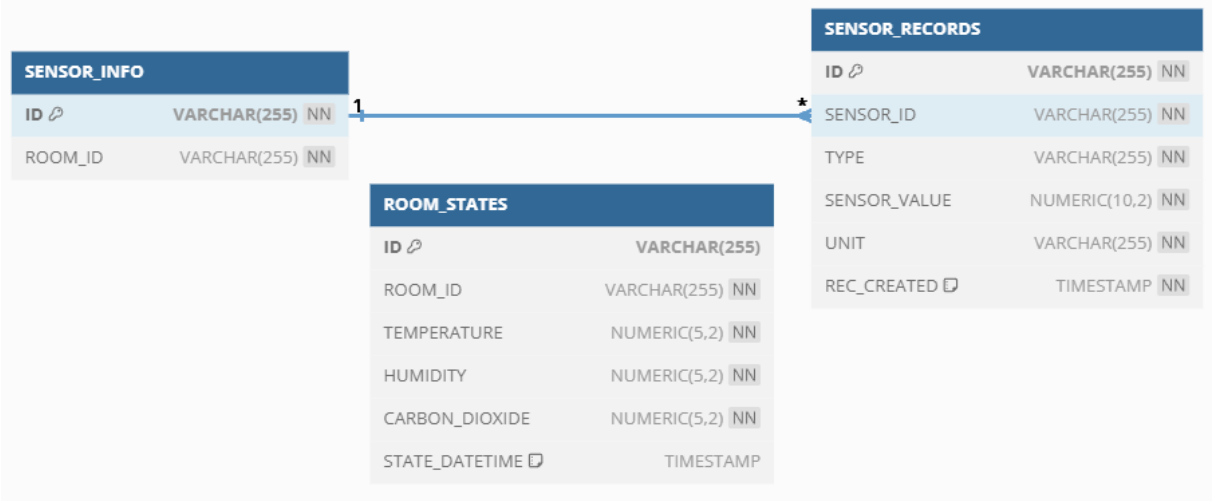
Řádky 64-66: V případě chyby (proměnná error) se loguje chyba (řádek 65) a generovaná data jsou uložena do databáze pro další zpracování (řádek 66).

Celkově tento kód ilustruje, jak aplikace může automaticky generovat data, odesílat je na server a reagovat na výsledky operace, a to vše v kontextu reaktivního programování s využitím Spring Framework a WebClient.

2.4 Ústředna hotelu

Ústředna hotelu je klíčovou součástí systému pro monitorování a řízení stavů místností v rozsáhlých budovách, jako jsou hotely. Tato kapitola se zabývá různými aspekty systému ústředny, včetně databázového schématu, technologického stacku a specifických implementačních detailů, které umožňují efektivní správu a agregaci dat z čidel rozmístěných po hotelu.

2.4.1 Databázové schéma



Obrázek 4 Databázové schéma ústředny hotelu

Na obrázku je zobrazeno schéma databáze sestávající z tří tabulek: SENSOR_INFO, ROOM_STATES, a SENSOR_RECORDS. Toto schéma je navrženo pro ukládání a správu dat souvisejících se senzory a stavy místností v kontextu systému pro monitorování rozsáhlých budov, jako jsou hotely. Tento návrh bude zobrazen ve třech různých pohledech: relačním, konceptuálním a fyzickém.

Relační pohled

V relačním pohledu se zaměříme na vztahy mezi tabulkami:

SENSOR_INFO:

Obsahuje informace o jednotlivých senzorech. Pro každý záznam v SENSOR_INFO může existovat několik záznamů v tabulce SENSOR_RECORDS

ROOM_STATES:

Sleduje aktuální stavy jednotlivých místností, jako jsou teplota, vlhkost a hladina oxidu uhličitého. Nejsou zde žádné vztahy.

SENSOR_RECORDS:

Uchovává historické záznamy dat získaných ze senzorů, včetně typu senzoru, naměřené hodnoty, jednotky a časového razítka vytvoření záznamu.

V relačním modelu vidíme, že může existovat relace mezi SENSOR_INFO a SENSOR_RECORDS přes SENSOR_ID, který slouží jako cizí klíč.

Konceptuální pohled

Konceptuální model se zaměřuje na vysokoúrovňový popis dat a jejich vztahů:

Senzory (SENSOR_INFO):

Reprezentuje fyzická zařízení rozmístěná v budově, každé s unikátním identifikátorem a přiřazením k místnosti.

Stavy místností (ROOM_STATES):

Udržuje klíčové ukazatele prostředí pro každou místnost, což umožňuje efektivní správu prostorů v budově.

Záznamy senzorů (SENSOR_RECORDS):

Detailní historická data od senzorů, která mohou být použita pro analýzy a optimalizace provozu.

Fyzický pohled

Fyzický model popisuje, jak jsou data uložena v databázi:

Typy dat:

Všechny ID jsou ukládána jako řetězce (VARCHAR (255)), což poskytuje flexibilitu v identifikaci.

Numerické hodnoty v ROOM_STATES (teplota, vlhkost, oxid uhličitý) jsou typu NUMERIC s přesností, což umožňuje zachovat přesnost měření.

Časová razítka (REC_CREATED v SENSOR_RECORDS) jsou ukládána jako TIMESTAMP, což zajišťuje přesné sledování času záznamu dat.

Indexace:

Primární klíče (ID v každé tabulce) jsou indexovány pro rychlý přístup a vyhledávání.

Možná by bylo vhodné indexovat i SENSOR_ID v SENSOR_RECORDS a ROOM_ID v ROOM_STATES a SENSOR_INFO pro efektivnější spojování dat.

Toto databázové schéma efektivně podporuje operace monitorování a řízení stavů místností v rozsáhlých budovách, jako jsou hotely, s ohledem na jejich provozní potřeby a analýzy.

2.4.2 Technologie a programovací jazyky

V kontextu ústředny hotelu byl projekt buildingNode vytvořen s použitím následujících technologií a nástrojů, jak je uvedeno v souboru pom.xml. Tento projekt využívá Java platformu a Spring Boot framework pro zpracování a analýzu dat z čidel umístěných po celém hotelu.

Spring Boot je projekt vytvořený nad frameworkem Spring, který usnadňuje vývoj a spuštění nových Spring aplikací. Hlavním cílem Spring Boot je minimalizovat množství konfigurace potřebné pro spuštění Spring aplikace. Umožňuje vývojářům rychlejší a efektivnější vývoj aplikací s méně konfiguracemi. [20]

Detailní popis konfigurace projektu v pom.xml je následující:

Struktura projektu

Parent Configuration: Projekt je postaven na spring-boot-starter-parent verze 3.2.2, což poskytuje přednastavené závislosti a pluginy vhodné pro rychlý vývoj Spring Boot aplikací.

Group ID: bachelor.practicalPart

Artifact ID: buildingNode

Version: 0.0.1-SNAPSHOT

Name: Building node

Description: Slouží k zpracování a analýze dat z čidel.

Použité technologie a knihovny

Java Version: Specifikována verze Java 21, která podporuje nejnovější jazykové funkce a optimalizace.

Dependencies:

Spring Boot Starter Data JPA: Pro integraci s databázemi a mapování objektů v Javě na databázové tabulky.

Spring Boot Starter Web: Umožňuje vývoj webových aplikací a RESTful služeb.

Spring Boot Starter Security: Zajišťuje zabezpečení aplikace, včetně autentizace a autorizace.

PostgreSQL Driver: JDBC driver pro připojení k PostgreSQL databázi.

Liquibase: Nástroj pro správu verzí databázových schémat a jejich migraci.

Spring Boot Starter Webflux: Podporuje vývoj asynchronních a neblokujících webových aplikací.

Lombok: Zjednodušuje psaní Java kódu pomocí anotací.

Spring Boot Starter Test a Spring Security Test: Poskytují podporu pro testování aplikace, včetně bezpečnostních aspektů.

Konfigurace sestavení

Spring Boot Maven Plugin: Nastavení tohoto pluginu zahrnuje vyřazení Lomboku při finálním sestavení, aby se předešlo potenciálním kompilačním problémům a problémům v runtime.

Tato konfigurace zajišťuje, že aplikace ústředny hotelu má robustní základ pro efektivní správu dat, která jsou klíčová pro monitoring a optimalizaci provozu hotelu. Díky modulárnímu přístupu Spring Bootu a integrovaným bezpečnostním funkcím je aplikace dobře připravena na různé operační výzvy a zabezpečení dat.

2.4.3 Ukázky kódu s popisky

Konfigurační Soubor YAML pro Ústřednu hotelu

```yaml
sendUrl: ${sendUrl}
buildingId: ${buildingId}
server:
  port: ${port}
spring:
  datasource:
    url: ${dbUrl}
    username: user
    password: password
    driver-class-name: org.postgresql.Driver
  jpa:
    hibernate:
      ddl-auto: none
    show-sql: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQLDialect
  liquibase:
    drop-first: true
    change-log: classpath:/liquibase/changelog/master.yaml
```

Obrázek 5 Applicaiton.yaml soubor ústředny hotelu

Na obrázku je zobrazen výřez z konfiguračního souboru ve formátu YAML, který se používá pro nastavení aplikace ústředny hotelu. Tento konfigurační soubor specifikuje různé parametry pro připojení a správu databáze a serveru, což umožňuje aplikaci efektivně fungovat v rámci hotelového systému. Detailněji popíšeme klíčové sekce:

Konfigurace serveru

buildingId: Tento parametr, nastavovaný proměnnou prostředí ${buildingId}, identifikuje specifický hotel nebo budovu, pro kterou je ústředna určena. To umožňuje aplikaci sloužit různým lokalitám, pokud je to potřeba.

port: Port, na kterém server naslouchá, definovaný proměnnou prostředí ${port}. To umožňuje flexibilní nasazení na různých portech v závislosti na potřebách infrastruktury.

Databázové nastavení

url: Adresa databáze je definována proměnnou ${dbUrl}, což umožňuje dynamické nastavení připojení k databázi.

username a password: Přihlašovací údaje k databázi, které zajišťují přístupová práva pro manipulaci s daty.

driver-class-name: Název třídy JDBC driveru pro PostgreSQL, který umožňuje komunikaci s databází.

Hibernate a JPA konfigurace

ddl-auto: Nastavení "none" znamená, že Hibernate automaticky nemodifikuje schéma databáze.

show-sql: Nastaveno na "true", což způsobí, že SQL příkazy generované Hibernate budou zobrazeny v logu. To je užitečné pro ladění a monitoring operací s databází.

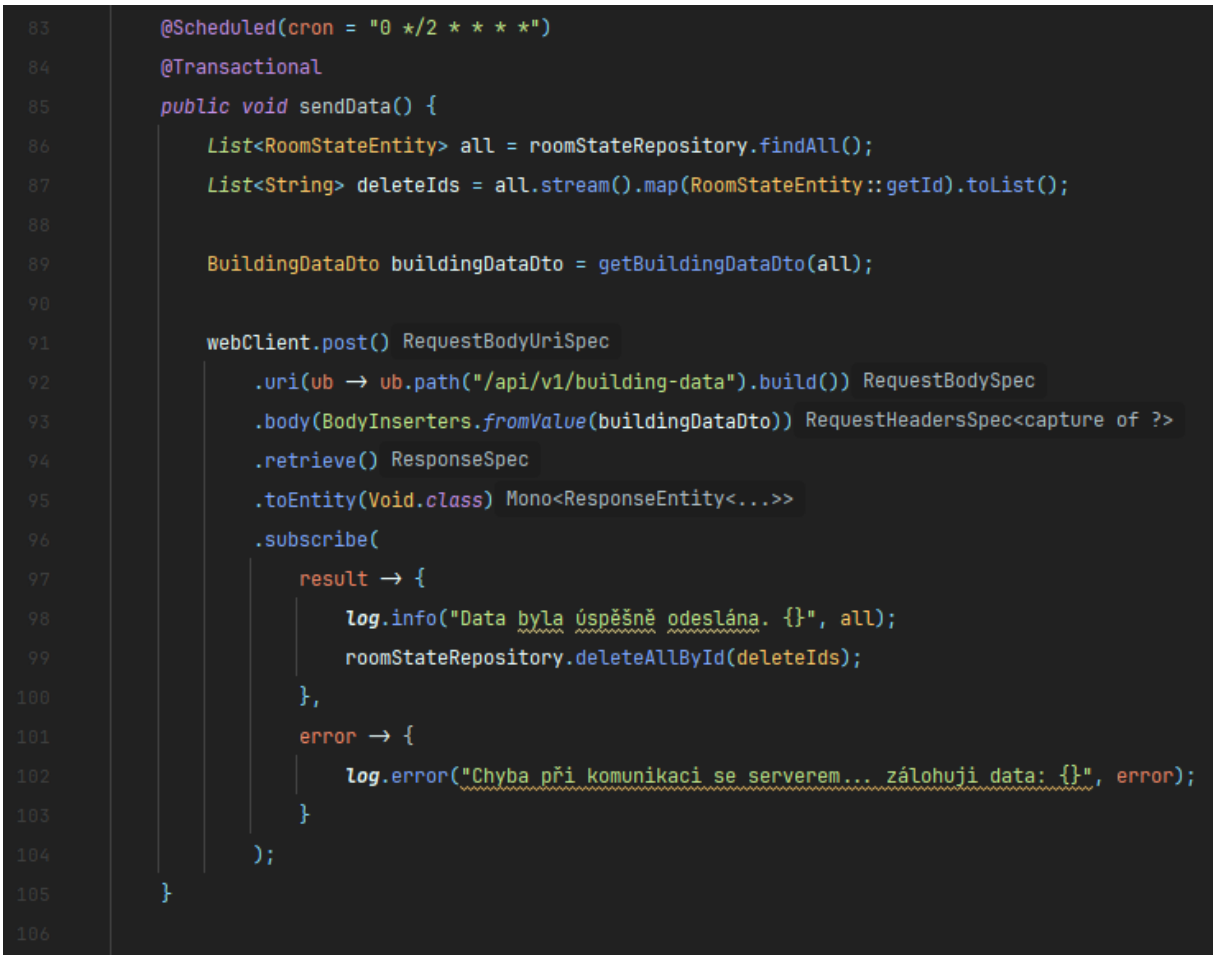
dialect: Specifikuje dialekt SQL používaný Hibernate pro databázi PostgreSQL, což umožňuje optimalizované SQL dotazy specifické pro danou databázi.

Liquibase

drop-first: Toto nastavení, když je nastaveno na "true", říká Liquibase, aby před aplikací změn nejprve odstranil stávající databázové objekty. Toto je obvykle užitečné v testovacím prostředí.

change-log: Cesta k souboru changelog Liquibase, který obsahuje definice změn databáze. Toto umožňuje správu verzí databázového schématu a jeho iterativní úpravy.

Tato konfigurace ilustruje, jak je možné flexibilně a bezpečně spravovat nastavení aplikace pro ústřednu hotelu, zajišťující tak její správnou funkčnost a integraci s ostatními systémy hotelu.



Obrázek 6 Ústředna hotelu metoda pro odesílání dat do ústředny sítě hotelů

Na obrázku je zobrazen kód v jazyce Java, který ilustruje implementaci funkce sendData(), sloužící k odeslání dat o stavu pokojů z ústředny hotelu na centrální server nebo do jiného systému. Popis kódu podle čísel řádků je následující:

Anotace a metoda

Řádek 84: Metoda sendData() je naplánována k periodickému spouštění každé dvě minuty, jak je specifikováno cron výrazem 0 */2 * * * *, což znamená spuštění na začátku každé druhé minuty.

Řádek 85: Anotace @Transactional zajišťuje, že celý proces běhu této metody proběhne v rámci jedné databázové transakce.

Získání a zpracování dat

Řádek 86-88: Načtení všech entit RoomStateEntity z repozitáře a extrakce jejich ID do seznamu deleteIds, které se později použije pro smazání záznamů.

Příprava dat k odeslání

Řádek 89: Vytvoření datového objektu BuildingDataDto z dat získaných z pokojů, což slouží jako payload pro odeslání.

Odeslání dat

Řádky 91-97: Použití WebClient pro asynchronní odeslání dat na definovanou URI. Metoda post() inicializuje POST požadavek, uri() specifikuje cílovou adresu, body() přidává data k odeslání, a retrieve() spouští požadavek.

Řádek 98: Zpracování odpovědi, kde toEntity(Void.class) naznačuje, že odpověď neobsahuje tělo.

Zpracování výsledků

Řádky 97-105: Metoda subscribe() zpracovává výsledky asynchronního požadavku:

Řádky 99-100: V případě úspěšného odeslání dat se loguje informace o úspěšném odeslání a provádí se mazání záznamů podle seznamu ID.

Řádky 102-103: V případě chyby se loguje chybové hlášení a provádění operace se zastaví.

Tento kód demonstruje, jak se v moderních aplikacích využívá Spring Framework k plánování úloh, asynchronní komunikaci s využitím WebClient a transakční správu databáze pro zpracování dat o stavu pokojů v hotelu.

```
43      @Scheduled(cron = "0 * * * * *")
44      @Transactional
45      public void processData() {
46          webClient.get() RequestHeadersUriSpec<capture of ?>
47              .uri(ub → ub.path("/api/v1/building-data/{hotelId}").build(buildingId)) capture of ?
48              .retrieve() ResponseSpec
49              .toEntityList(SensorRoomDto.class) Mono<ResponseEntity<...>>
50              .subscribe(
51                  result → {
52                      log.error("Aktualizace dat budovy");
53                      List<SensorRoomDto> dataFromSystem = result.getBody();
54                      if (!CollectionUtils.isEmpty(dataFromSystem)) {
55                          syncWithHotelSystem(dataFromSystem);
56                      }
57                      processDataInternal();
58                  },
59                  error → {
60                      log.error("Chyba při komunikaci se serverem... počítát se starými daty: {}", error);
61                      processDataInternal();
62                  }
63              );
64      }
```

Obrázek 7 Ústředna hotelu metoda pro agregaci dat

Z prvního obrázku vidíme, že se jedná o kód v jazyce Java, který obsahuje metodu processData(), sloužící k získání a zpracování dat o stavu hotelu z centrálního systému pomocí webového klienta. Metoda je plánovaná k pravidelnému spouštění každou minutu, jak je specifikováno v anotaci @Scheduled. Celý proces je zabaleno v transakci zajištěné anotací @Transactional.

Funkční popis kódu z prvního obrázku:

Řádky 45-63: Kód implementuje asynchronní dotazování na API pomocí webClient.get(), kde získává data z URI, která je konfigurovaná proměnnými prostředí a zahrnuje specifický identifikátor budovy (buildingId). Odpověď očekává ve formě seznamu objektů SensorRoomDto, které jsou poté zpracovány.

Řádek 51: Po úspěšném získání dat se loguje aktuální akce a data se dále zpracovávají. Pokud data nejsou prázdná, synchronizují se s interním systémem hotelu.

Řádek 57: Po zpracování dat se volá interní metoda processDataInternal(), která dále zpracovává získané informace a ukládá je do databáze.

Řádky 59-63: V případě chyby v komunikaci se loguje chyba a také se volá metoda processDataInternal().

Kontext s druhým obrázkem:

Druhý obrázek reprezentuje pokračování procesu, kde processDataInternal() agreguje získané údaje o stavu místností a aktualizuje databázi na základě nových dat. Toto byl kontext, který byl použitý pro návrh a implementaci první metody, zajišťující, že údaje získané z API jsou efektivně zpracovány a uloženy.

```
129     private void processDataInternal() {
130         OffsetDateTime now = OffsetDateTime.now();
131         List<SensorRecordEntity> allSensorRecords = sensorRecordRepository.findByRecCreatedBetween(now.minusMinutes(1), now);
132
133         List<String> sensorRecordIds = allSensorRecords.stream() Stream<SensorRecordEntity>
134             .map(SensorRecordEntity::getId) Stream<String>
135             .toList();
136         Map<String, List<SensorRecordEntity>> groupedBySensorId = allSensorRecords.stream()
137             .collect(Collectors.groupingByConcurrent(SensorRecordEntity::getSensorId));
138
139         sensorRepository.findAll().stream() Stream<SensorInfoEntity>
140             .collect(Collectors.groupingBy(SensorInfoEntity::getRoomId)) Map<String, List<...>>
141             .forEach((roomId, sensors) -> {
142                 Map<SensorValueTypeEnum, Double> averagedValuesForEachCategory = aggregateRoomState(sensors, groupedBySensorId);
143                 roomStateRepository.save(RoomStateEntity.builder()
144                     .roomId(roomId)
145                     .humidity(getAverageValueByType(averagedValuesForEachCategory, SensorValueTypeEnum.HUMIDITY))
146                     .temperature(getAverageValueByType(averagedValuesForEachCategory, SensorValueTypeEnum.TEMPERATURE))
147                     .carbonDioxide(getAverageValueByType(averagedValuesForEachCategory, SensorValueTypeEnum.CO))
148                     .stateDateTime(now)
149                     .build()
150                 );
151             });
152
153         if (CollectionUtils.isEmpty(allSensorRecords)) {
154             sensorRecordRepository.deleteAllById(sensorRecordIds);
155         }
156     }
```

Obrázek 8 Ústředna hotelu interní metoda pro agregaci dat

Druhý obrázek zobrazuje funkci processDataInternal(), která je navržena pro zpracování a aktualizaci dat o stavu místností hotelu založené na sensorových záznamech, které byly získány a předzpracovány v metodě processData(), jak bylo vidět na prvním obrázku.

Podrobný popis kódu z druhého obrázku:

Řádky 129-156: Metoda processDataInternal() zpracovává sensorová data uložená v databázi, agreguje je a aktualizuje stav místností v hotelu.

Řádek 130: Vytvoření proměnné pro zaznamenání aktuálního času.

Řádky 131-135: Získání všech sensorových záznamů vytvořených v poslední minutě z repozitáře a uložení jejich ID do seznamu pro možné následné smazání.

Řádek 136-138: Záznamy jsou seskupeny podle ID sensoru pro další zpracování.

Řádky 139-151: Iterace přes jednotlivé místnosti identifikované jejich ID. Pro každou místnost se vypočítají průměrné hodnoty pro jednotlivé typy měření (vlhkost, teplota, CO2) pomocí pomoci metody aggregateRoomState(), která zpracovává získaná data.

Řádek 142-148: Pro každou místnost se vytvoří nový záznam stavu místnosti s vypočítanými průměrnými hodnotami a aktuálním časovým razítkem. Tento záznam je poté uložen do repozitáře místností.

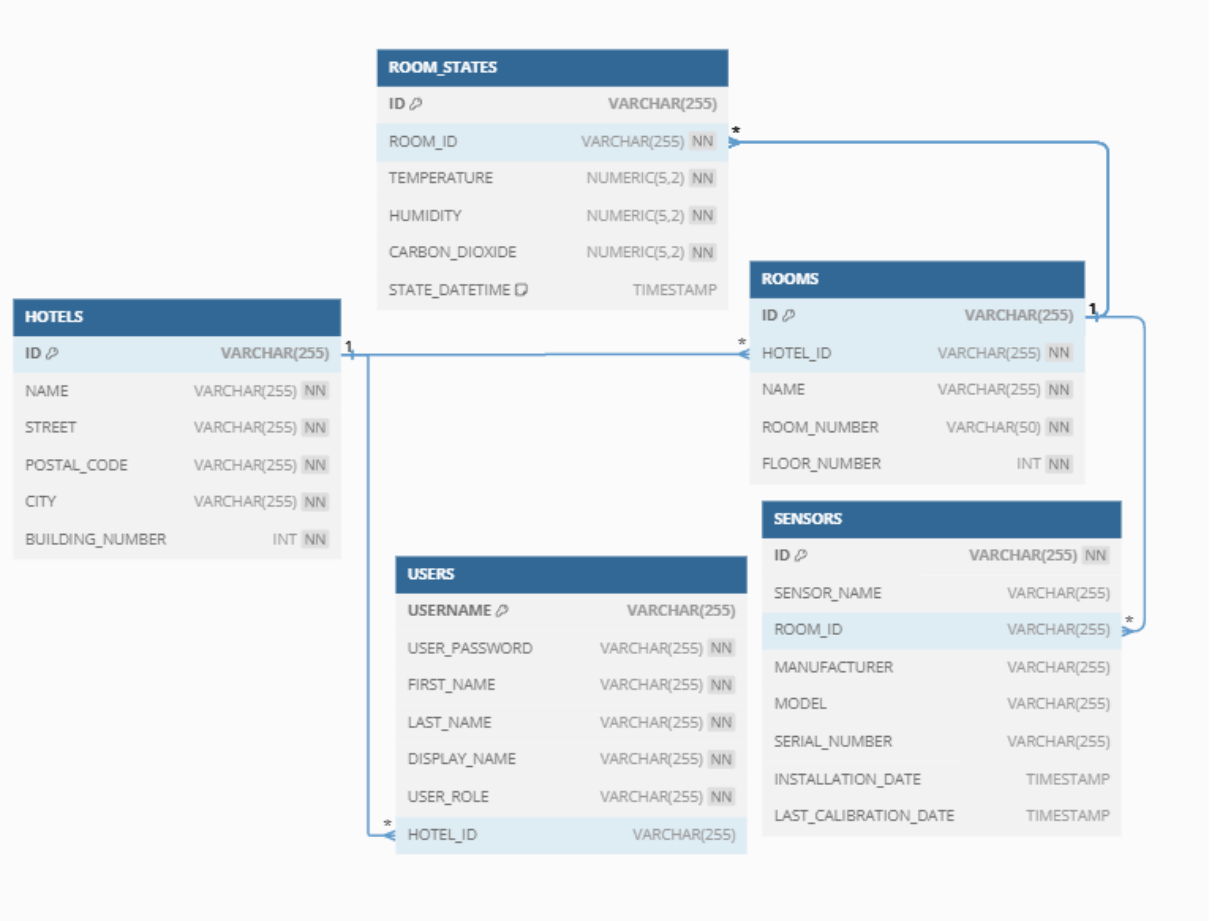
Řádky 153-155: Po úspěšném zpracování a uložení nových stavů místností, jsou staré sensorové záznamy smazány z databáze, čímž se udržuje databáze v čistotě a efektivitě.

Tato metoda je klíčovou součástí systému pro správu dat získaných z čidel, neboť zajišťuje, že data jsou nejen získávána, ale i efektivně zpracována a využita pro další provoz hotelu. Zpracování dat na úrovni místností umožňuje hotelu optimalizovat provoz a zlepšit pohodlí pro hosty tím, že reaguje na změny v prostředí v reálném čase.

2.5 Ústředna sítě hotelů

Ústředna sítě hotelů představuje komplexní systém pro správu a analýzu dat získaných z jednotlivých hotelů, které jsou součástí větší hotelové sítě. Tato kapitola se zaměřuje na technologické řešení, databázové schéma a uživatelské rozhraní, které společně poskytují nástroje pro efektivní správu dat na úrovni celé sítě.

2.5.1 Databázové schéma



Obrázek 9 Databázové schéma Ústředny hotelů

Na obrázku je zobrazeno databázové schéma ústředny hotelu, které obsahuje pět hlavních tabulek: HOTELS, USERS, ROOMS, SENSORS, a ROOM_STATES. Toto schéma umožňuje efektivní správu informací týkajících se hotelů, jejich pokojů, uživatelů, senzorů a měřených stavů pokojů.

Relační Pohled

HOTELS:

Uchovává informace o jednotlivých hotelech, včetně názvu, adresy a čísla budovy.

USERS:

Zaznamenává informace o uživatelích systému, včetně jejich rolí a příslušnosti k hotelům. Má vztah s tabulkou HOTELS přes HOTEL_ID.

ROOMS:

Obsahuje informace o pokojích v rámci hotelů, včetně umístění (podlaží, číslo pokoje). Je propojena s HOTELS skrze HOTEL_ID.

SENSORS:

Ukládá detaily o senzorech instalovaných v pokojích, včetně jména, výrobce, modelu, a dat instalace a poslední kalibrace. Je spojena s ROOMS přes ROOM_ID.

ROOM_STATES:

Zaznamenává stavy jednotlivých pokojů, jako jsou teplota, vlhkost a hladina CO2, včetně časového razítka stavu. Relace s ROOMS umožňuje sledování stavů konkrétních pokojů.

Konceptuální Pohled

Hotely:

Centrální entita, která organizuje všechny ostatní data kolem sebe, sloužící jako hub pro uživatele a pokoje.

Uživatelé:

Reprezentují osoby spravující nebo pracující v hotelu, s různými oprávněními založenými na jejich rolích.

Pokoje:

Základní jednotky hotelu, kde jsou instalovány senzory pro monitorování a zajištění pohodlí hostů.

Senzory:

Zařízení sbírající data o prostředí, jejichž údaje jsou klíčové pro operativní rozhodování a údržbu.

Stavy Pokojů:

Dynamické záznamy poskytující real-time data pro analýzu a reakci na měnící se podmínky v pokojích.

Fyzický Pohled

Datové Typy a Omezení:

ID všechny tabulek jsou VARCHAR(255), což zajišťuje univerzálnost a flexibilitu v identifikátorech.

Textová pole jako názvy a adresy jsou rovněž VARCHAR(255).

Numerické hodnoty pro teplotu, vlhkost a CO2 jsou typu NUMERIC(5,2), poskytující dostatečnou přesnost.

Časová razítka jsou typu TIMESTAMP, což umožňuje přesné sledování časových událostí.

Indexace a Klíče:

Každá tabulka má primární klíč definovaný na ID, což zajišťuje rychlé vyhledávání a jedinečnost záznamů.

Cizí klíče jsou definovány pro vytvoření relací mezi tabulkami, což umožňuje efektivní spojování dat při dotazech.

Toto databázové schéma a jeho implementace umožňují ústředně sítě hotelů efektivně spravovat a analyzovat velké množství dat, což vede ke zlepšení služeb a optimalizaci provozu hotelové sítě.

2.5.2 Technologie a programovací jazyky

Systém ústředny sítě hotelů je vyvíjen v jazyce Java, což je ideální pro vývoj robustních a škálovatelných aplikací. Java 21 je používána pro zajištění kompatibility s nejnovějšími vývojovými trendy a bezpečnostními standardy.

Nástroje pro Sestavení

Maven je zvolen pro správu projektu a závislostí. Tento nástroj podporuje správu životního cyklu aplikace od kompilace kódu, jeho testování, až po balení a distribuci, což zajišťuje efektivitu a konzistenci během vývoje.

Závislosti a Knihovny

Spring Boot: Použití různých modulů Spring Boot umožňuje efektivní vývoj a správu aplikací:

spring-boot-starter-web: Podporuje vývoj webových aplikací, včetně RESTful služeb.

spring-boot-starter-data-jpa: Facilituje integraci s databázemi a mapování objektů v Javě na databázové tabulky.

spring-boot-starter-security: Poskytuje nástroje pro zabezpečení aplikace, včetně autentizace a autorizace.

spring-boot-starter-validation: Umožňuje validaci dat přicházejících do aplikace.

spring-boot-starter-thymeleaf: Umožňuje vytváření server-side generovaných HTML šablon.

thymeleaf-extras-springsecurity6: Propojuje Thymeleaf s Spring Security, což umožňuje integraci bezpečnostních aspektů přímo do šablon.

PostgreSQL JDBC Driver: Zajišťuje komunikaci s PostgreSQL databází, která je používána pro ukládání a správu dat.

Liquibase: Nástroj pro správu verzí databázových schémat, který umožňuje snadné a bezpečné provedení změn v databázi.

Lombok: Zjednodušuje psaní Java kódu díky automatizaci běžných metod jako getters/setters.

Spring Boot Test a Spring Security Test: Poskytují podporu pro komplexní testování aplikace.

Konfigurace Sestavení

Konfigurace Maven pluginu pro Spring Boot zahrnuje vyloučení Lombok knihovny z finálního sestavení aplikace, což eliminuje potenciální komplikace při kompilaci a běhu aplikace.

Tato konfigurace zajišťuje, že aplikace ústředny sítě hotelů je robustní, bezpečná a efektivně spravovatelná. Systém umožňuje uživatelům pohodlně spravovat data a konfigurace napříč celou sítí hotelů, zatímco poskytuje nástroje pro hlubokou analýzu dat pro zlepšení operací a služeb.

2.5.3 Ukázky kódu s popisky

```
32      @PostMapping("/building-data")
33  @   public ResponseEntity<Void> getSensorData(@RequestBody() BuildingDataDto buildingDataDto) {
34          if (!hotelRepository.existsById(buildingDataDto.getBuildingId())) {
35              return ResponseEntity.badRequest().build();
36          }
37          roomStateRepository.saveAll(buildingDataDto.getRoomStateDtoList().stream() Stream<RoomStateDto>
38              .map(dto → RoomStateEntity.builder()
39                  .roomId(dto.getRoomId())
40                  .humidity(dto.getHumidity())
41                  .temperature(dto.getTemperature())
42                  .carbonDioxide(dto.getCarbonDioxide())
43                  .stateDateTime(dto.getStateDateTime())
44                  .build()
45              ) Stream<RoomStateEntity>
46              .toList()
47          );
48          return ResponseEntity.ok().build();
49      }
```

Obrázek 10 Ústředna hotelů metoda pro získání dat z budov

Na obrázku je zobrazen kód v jazyce Java, který definuje REST API endpoint v rámci systému pro monitoring rozsáhlých budov, specificky pro ústřednu sítě hotelů. Metoda getSensorData přijímá data z hotelů a zpracovává je. Zde je detailní popis toho, co se děje v metodě podle čísel řádků:

Metoda getSensorData

Řádek 32: Metoda getSensorData je definována jako POST endpoint na URL /building-data. Tato metoda očekává jako vstup objekt BuildingDataDto.

Řádek 33-36: První krok metody zkontroluje, zda hotel s daným ID existuje v databázi. Používá k tomu metodu existsById z repozitáře hotelRepository. Pokud hotel neexistuje, metoda okamžitě vrací HTTP stavový kód 400 (Bad Request).

Řádek 37-46: Pokud hotel existuje, metoda pokračuje zpracováním dat stavů místností, která jsou součástí přijatého BuildingDataDto. Data jsou získána z getRoomStateDtoList a poté jsou uložena do databáze pomocí roomStateRepository. Zde dochází k iteraci přes každý RoomStateDto objekt v seznamu:

Řádek 39-44: Pro každý objekt RoomStateDto je vytvořena nová entita RoomStateEntity pomocí builder patternu. Nastaví se hodnoty roomID, humidity, temperature, carbonDioxide, a stateDateTime z DTO do entity.

Výsledné entity jsou shromážděny do seznamu pomocí toList().

Řádek 48: Po úspěšném zpracování všech dat metoda vrací HTTP stavový kód 200 (OK) jako odpověď.

Tento kód ilustruje, jak aplikace přijímá a zpracovává data z čidel instalovaných v různých místnostech hotelů. Každý záznam o stavu místnosti je aktualizován v databázi s novými hodnotami přijatými od čidel, což umožňuje centrální monitorování a správu stavů prostředí v hotelové síti.

```
51      @GetMapping("/building-data/{hotelId}")
52      public ResponseEntity<List<SensorRoomDto>> getSensorData(@PathVariable String hotelId) {
53          if (!hotelRepository.existsById(hotelId)) {
54              return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
55          }
56
57          List<String> roomIds = roomRepository.findByHotelId(hotelId, Sort.by(Sort.Direction.ASC,  ...properties: "name")).stream().map(RoomEntity::getId).toList();
58          return ResponseEntity.ok(sensorRepository.findByRoomIdIn(roomIds).stream() Stream<SensorEntity>
59              .map(sensor → SensorRoomDto.builder().sensorId(sensor.getId()).roomId(sensor.getRoomId()).build()) Stream<SensorRoomDto>
60              .toList());
61      }
62  }
63
```

Obrázek 11 Ústředna hotelů metoda pro synchronizaci dat s budovami

Na obrázku je zobrazen Java kód, který definuje REST API endpoint pro systém pro monitoring rozsáhlých budov, konkrétně pro ústřednu sítě hotelů. Tento endpoint je určen pro synchronizaci dat o senzorech mezi ústřednou a jednotlivými hotely. Metoda getSensorData slouží k získávání dat o senzorech pro konkrétní hotel identifikovaný pomocí hotelId. Podrobný popis toho, co se děje v metodě podle čísel řádků je následující:

Metoda getSensorData

Řádek 51: Definuje se GET endpoint /building-data/{hotelId}, kde {hotelId} je proměnná cesty (path variable), která představuje ID hotelu.

Řádek 52-55: Metoda začíná kontrolou, zda hotel s daným ID existuje v databázi pomocí metody existsById z repozitáře hotelRepository. Pokud hotel neexistuje, metoda vrací HTTP status 401 (Unauthorized), což naznačuje, že dotaz na získání dat o senzorech pro neexistující hotel není povolen.

Řádek 56-60: Pokud hotel existuje, metoda pokračuje získáváním seznamu ID místností patřících k danému hotelu. To se děje pomocí metody findByHotelId z roomRepository, která vrací místnosti seřazené vzestupně podle názvu.

Řádek 58: Seznam ID místností se extrahuje z vrácených místností a uloží do seznamu roomIds.

Řádek 59-60: Pro každé získané ID místnosti se hledají senzory patřící do těchto místností pomocí findByRoomIdIn z sensorRepository. Výsledky se převádějí na seznam objektů SensorRoomDto pomocí stream() a map() operací, kde se nastavují ID senzoru a ID místnosti.

Řádek 61: Metoda vrací seznam objektů SensorRoomDto jako HTTP odpověď s status kódem 200 (OK).

Tento endpoint efektivně zajišťuje, že údaje o senzorech jsou dostupné pro synchronizaci mezi ústřednou a jednotlivými hotely, což umožňuje centralizované monitorování stavů a optimalizaci operací hotelu.

```java
@RequiredArgsConstructor
public class SecurityConfig {
    private final UserDetailsService userDetailsService;

    1 usage
    @Bean
    public static PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }

    no usages
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(httpSecurityCsrfConfigurer -> httpSecurityCsrfConfigurer.disable())
            .authorizeHttpRequests(requests -> requests
                .requestMatchers( ...patterns: "/css/**", "/js/**", "/api/v1/**", "/login").permitAll()
                .anyRequest().authenticated()
            )
            .httpBasic(Customizer.withDefaults())
            .formLogin(form -> form
                .loginPage("/login")
                .defaultSuccessUrl( defaultSuccessUrl: "/dashboard",  alwaysUse: true)
                .loginProcessingUrl("/login")
                .failureUrl( authenticationFailureUrl: "/login?error=true")
                .permitAll()
            )
            .logout((logout) -> logout.logoutUrl("/logout").logoutSuccessUrl("/login"));

        return http.build();
    }

    no usages
    public void configure(AuthenticationManagerBuilder builder) throws Exception {
        builder.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
    }
}
```

Obrázek 12 Ústředna hotelů nastavení Spring Security

Na obrázku je zobrazena část konfiguračního souboru pro Spring Security v aplikaci, která slouží jako ústředna sítě hotelů. Tato konfigurace zajišťuje zabezpečení aplikace nastavením různých bezpečnostních politik a pravidel pro autentizaci a autorizaci uživatelů. Podrobný popis klíčových aspektů konfigurace podle čísel řádků je následující:

Konfigurace Spring Security

Řádek 21: Deklarace závislosti na UserDetailsService, což je služba používaná k načítání uživatelských dat pro autentizaci.

Řádek 25-27: Definice PasswordEncoder, který je použit pro šifrování hesel uživatelů. Zde je použit BCryptPasswordEncoder, což je silný šifrovací algoritmus doporučovaný pro moderní aplikace.

Řádek 29-46: Nastavení SecurityFilterChain, které definuje, jak jsou zabezpečené HTTP požadavky:

Řádek 31-35: Vypnutí CSRF ochrany (csrf().disable()), což je běžné pro API, které jsou chráněné jinými mechanismy, například tokeny.

Řádek 33-34: Konfigurace pravidel pro přístup k různým URL. Statické zdroje (/css/**, /js/**) a API endpointy (/api/v1/**) jsou přístupné bez autentizace, zatímco všechny ostatní požadavky vyžadují autentizaci.

Řádek 36: Nastavení základní HTTP autentizace.

Řádek 37-43: Konfigurace formulářového přihlášení:

Řádek 38: Nastavení cesty k přihlašovací stránce.

Řádek 39: URL, na kterou se odešlou přihlašovací údaje.

Řádek 40: URL pro přesměrování po úspěšném přihlášení.

Řádek 41: URL pro přesměrování v případě selhání přihlášení.

Řádek 44-45: Nastavení odhlášení:

Řádek 45: URL pro proces odhlášení a URL pro přesměrování po úspěšném odhlášení.

Tato konfigurace zabezpečení je zásadní pro ochranu dat a operací v rámci ústředny sítě hotelů, zajišťuje kontrolu přístupu k citlivým informacím a umožňuje bezpečnou správu uživatelských účtů a operací.

2.5.4 Uživatelské rozhraní (UI)

Rozdíly v rolích

Admin (Globální Administrátor)

Přístup ke všem hotelům: Admin má přístup ke všem datům a funkcím v systému, což zahrnuje informace o všech hotelech v síti. Může prohlížet, upravovat a spravovat detaily každého hotelu bez omezení.

Správa uživatelů: Admin má schopnost spravovat všechny uživatelské účty, včetně přidávání nových uživatelů, úpravy jejich rolí a oprávnění, nebo mazání uživatelů z systému.

Kompletní dashboard: Dashboard admina poskytuje komplexní přehled o provozu a aktivitách napříč celou sítí hotelů, což zahrnuje analýzy, statistiky a hlášení.

Administrativní oprávnění: Admin má nejvyšší úroveň oprávnění, která zahrnuje nastavení systému, konfiguraci bezpečnostních politik a dalších systémových nastavení.

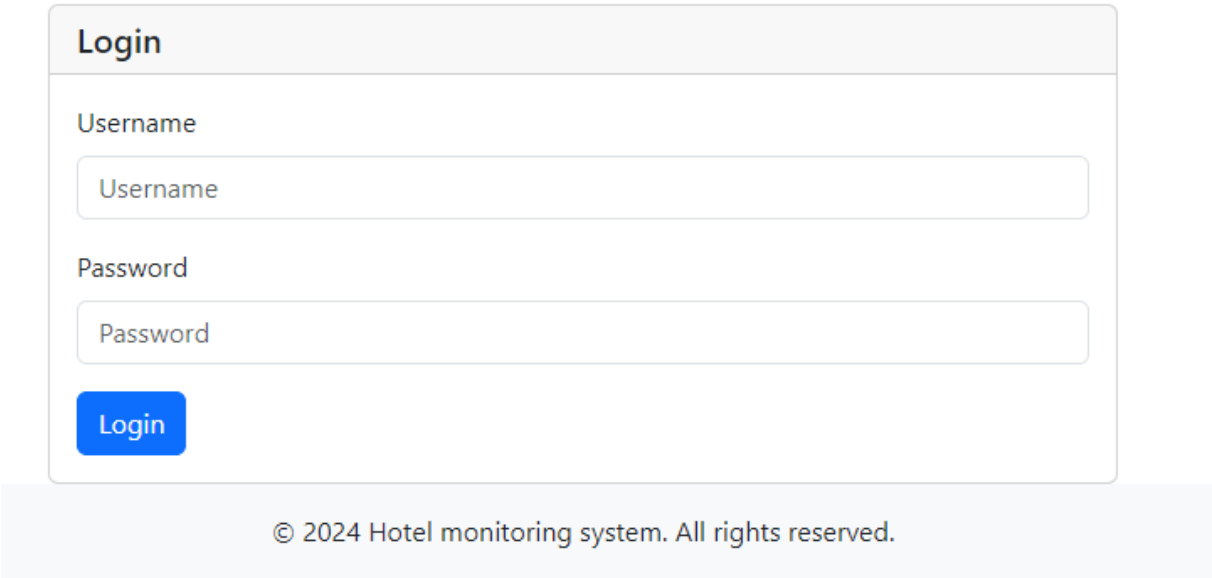
Admin Hotelu (Hotelový Administrátor)

Omezený přístup k hotelu: Admin hotelu má přístup pouze k datům a funkcím týkajícím se hotelu, který má přiřazený. Nemůže získat přístup k informacím nebo provádět operace mimo svůj přidělený hotel.

Správa místností a senzorů: Jeho hlavní odpovědností je správa místností a senzorů v jeho hotelu, včetně monitorování stavu místností a správy senzorů a jejich kalibrací.

Tato rozlišení v UI zajišťují, že každá administrativní role má přístup pouze k informacím a funkcím, které jsou relevantní pro její úkoly a odpovědnosti. Tímto způsobem systém udržuje bezpečnost a integritu dat, zatímco zajišťuje efektivní a cílenou správu jednotlivých segmentů hotelové sítě.

Přihlášení do systému



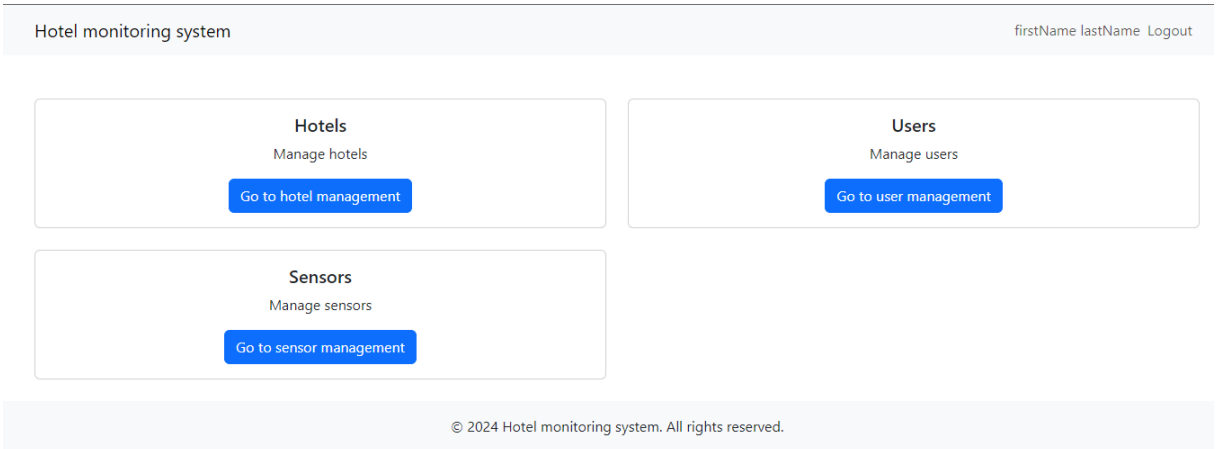
Obrázek 13 Ústředna hotelů přihlašovací stránka

Po otevření aplikace budete přesměrováni na přihlašovací obrazovku.

Vyplňte Username (přihlašovací jméno), Password (přihlašovací heslo) a stiskněte Login pro přihlášení.

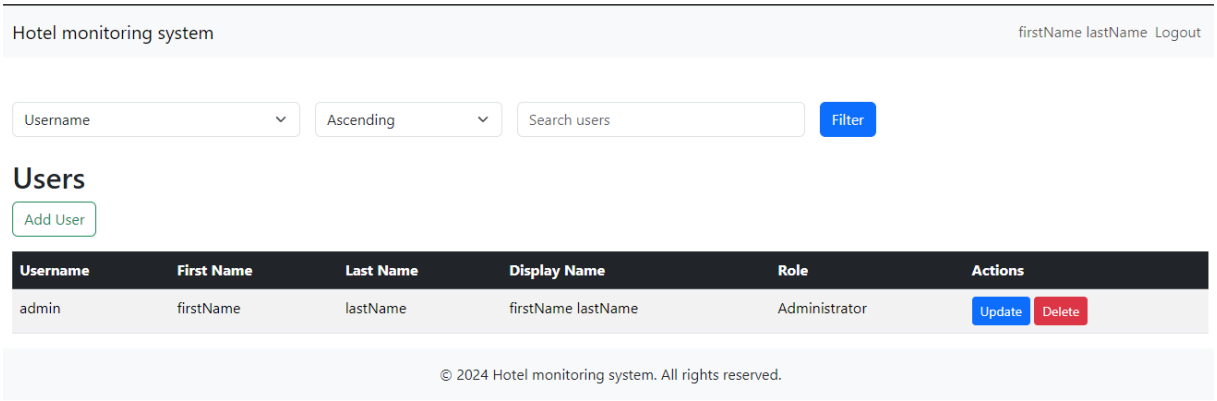
2.5.5 Průvodce s obrázky a popisky

Správa uživatelů



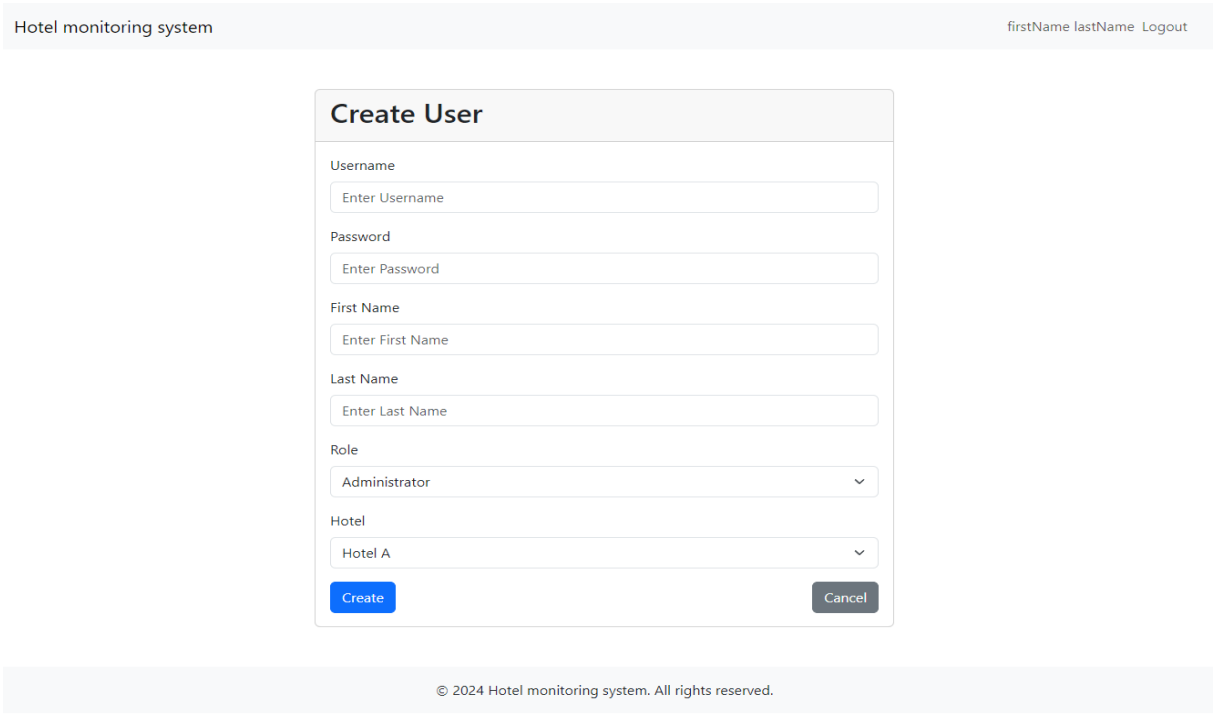
Obrázek 14 Ústředna hotelů dashboard

Po přihlášení klikněte na „Go to user management“



Obrázek 15 Ústředna hotelů správa uživatelů

Na stránce správy uživatelů se zobrazí seznam všech uživatelů s možnostmi akcí pro každého uživatele. Můžete zde filtrovat vyhledávacího pole a řadit podle různých sloupců.



Obrázek 16 Ústředna hotelů uživatelský formulář

Přidání uživatele:

Klikněte na „Add User“ tlačítko a vyplňte formulář.

Pozn.: Pole hotel ve formláři je ignorováno pokud se zakládá role „Administrator“.

Aktualizace uživatele:

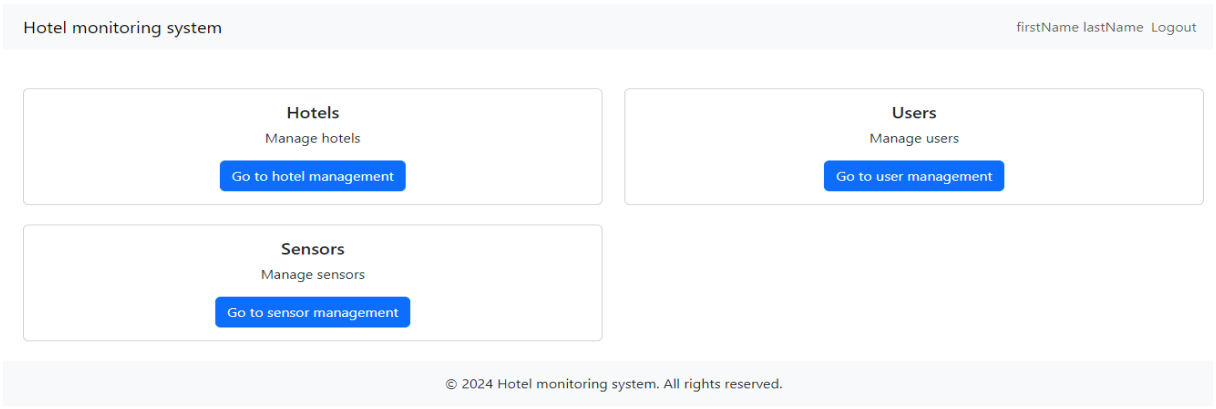
Klikněte na řádku uživatele, kterého chcete aktualizovat na modré tlačítko „Update“.

Pozn.: Pokud chcete změnit heslo musíte odstranit uživatele a vytvořit ho znovu.

Mazání uživatele:

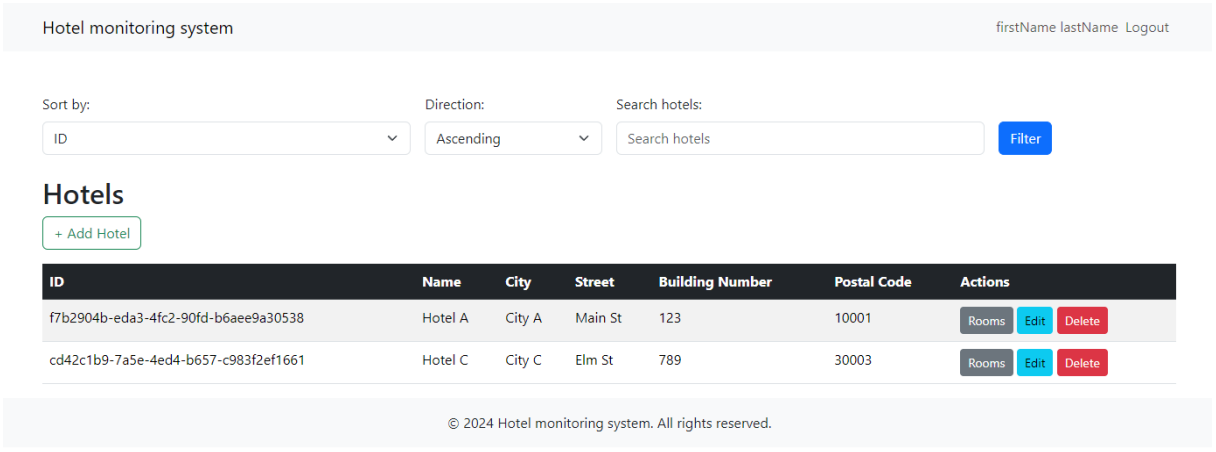
Klikněte na řádku uživatele, kterého chcete smazat na červené tlačítko „Delete“.

Správa hotelů

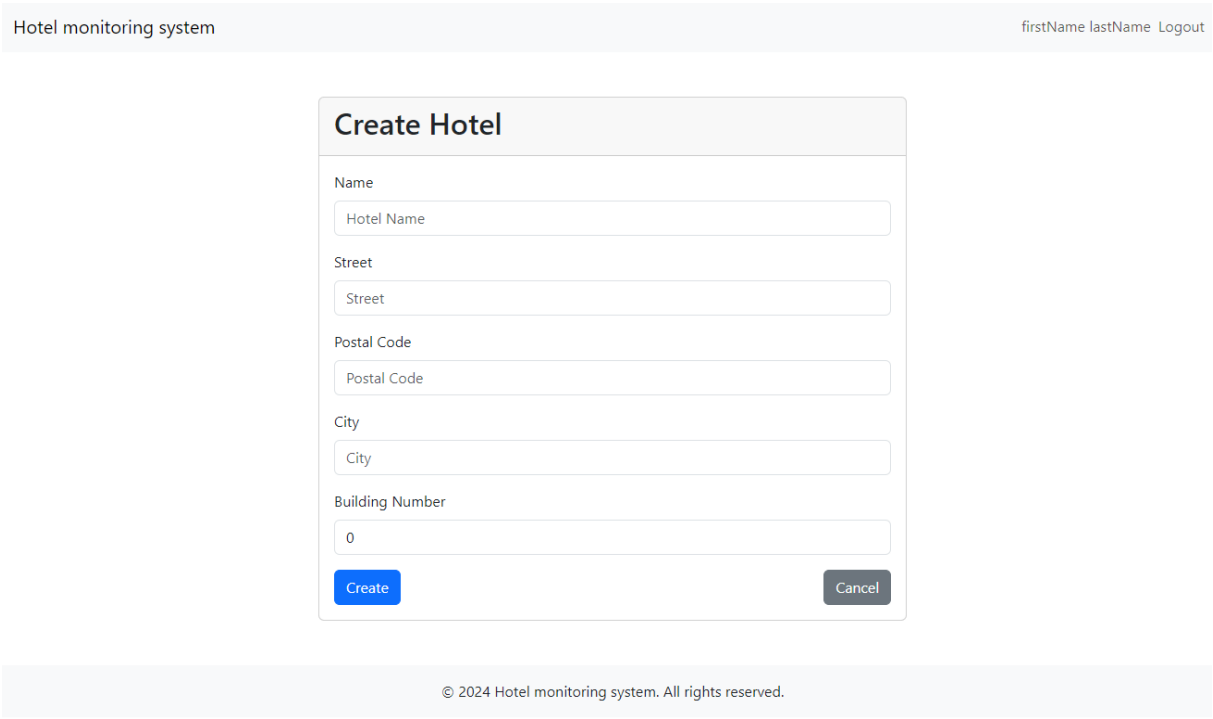


Obrázek 17 Ústředna hotelů dashboard pro hotely

Po přihlášení klikněte na „Go to hotel management“



Obrázek 18 Ústředna hotelů správa hotelů



Obrázek 19 Ústředna hotelů hotelový formulář

Na této stránce uživatelského rozhraní systému monitorování hotelů může uživatel provádět několik akcí spojených se správou hotelů. Stránka poskytuje funkce pro přidávání, prohlížení, úpravy a mazání informací o jednotlivých hotelech v systému. Detailní popis akcí, které může uživatel na této stránce provádět, zahrnuje:

Přidání nového hotelu:

Uživatel může kliknout na tlačítko "+ Add Hotel", což otevře formulář nebo novou stránku, kde lze zadat informace o novém hotelu, jako jsou název, ulice, číslo budovy, poštovní kód a město.

Filtrování a hledání hotelů:

Uživatel může vyhledávat hotely podle konkrétních kritérií pomocí textového pole "Search hotels".

K dispozici jsou také možnosti pro řazení hotelů podle různých kritérií (např. ID, název) a směru řazení (vzestupně/sestupně) pomocí rozbalovacích menu "Sort by:" a "Direction:".

Prohlížení detailů hotelu:

Seznam hotelů na stránce zahrnuje základní informace jako ID, název, ulici, číslo budovy, město a poštovní kód.

Úprava informací o hotelu:

Vedle každého záznamu hotelu je tlačítko "Edit", které umožňuje uživatelům upravit informace o hotelu. Kliknutím na toto tlačítko se uživatel dostane na stránku nebo formulář, kde může změnit existující informace o hotelu.

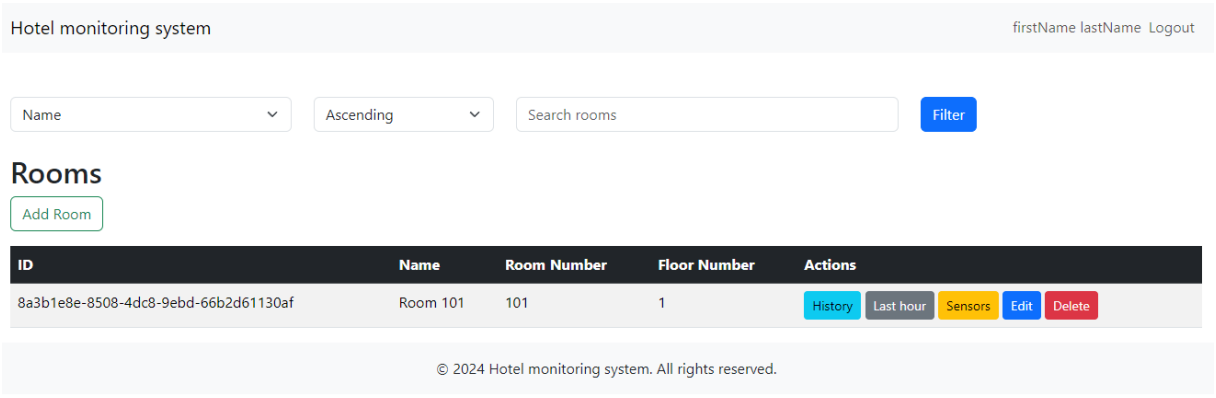
Mazání hotelu:

Tlačítko "Delete" umožňuje uživateli odstranit hotel z databáze. Toto je obvykle potvrzeno prostřednictvím dialogového okna, které žádá uživatele o potvrzení, zda skutečně chce hotel odstranit.

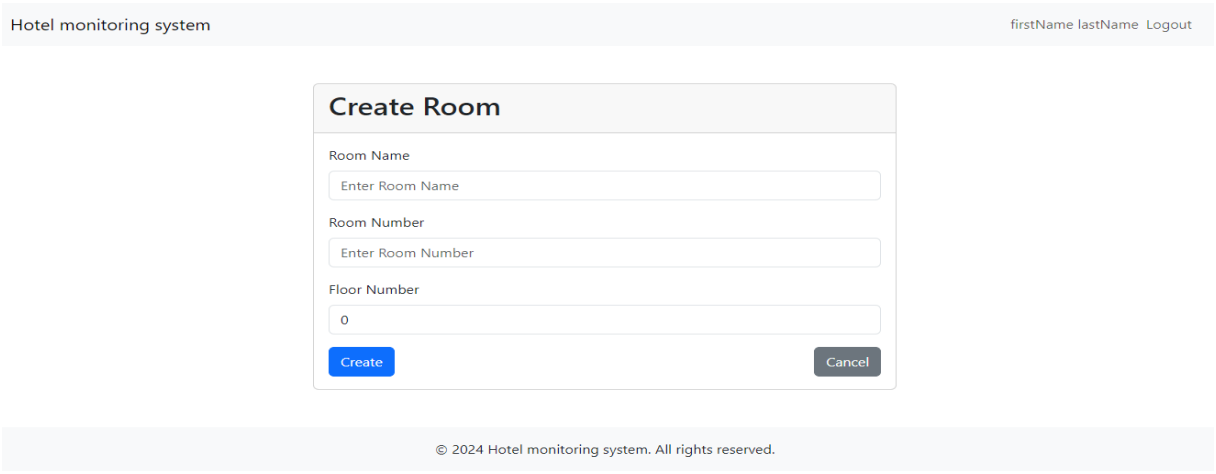
Prohlížení pokojů hotelu:

Tlačítko "Rooms" vedle každého hotelu umožňuje uživatelům zobrazit seznam pokojů, které jsou součástí daného hotelu. Toto tlačítko vede na stránku, kde jsou zobrazeny pokoje hotelu

Tato stránka tedy slouží jako centrální místo pro správu hotelů v rámci systému monitorování, poskytující uživatelům možnost spravovat klíčové informace týkající se jejich provozu a správy nemovitostí.



Obrázek 20 Ústředna hotelů správa místností hotelu



Obrázek 21 Ústředna hotelů formulář místnosti

Na této stránce uživatelského rozhraní pro systém monitorování hotelů má uživatel několik možností pro správu pokojů v rámci daného hotelu. Uživatel může přidávat nové pokoje, zobrazovat a aktualizovat existující informace, a mazat pokoje z databáze. Stránka poskytuje také nástroje pro filtraci a hledání specifických pokojů.

Možnosti, které uživatel může na této stránce provádět:

Přidání nového pokoje:

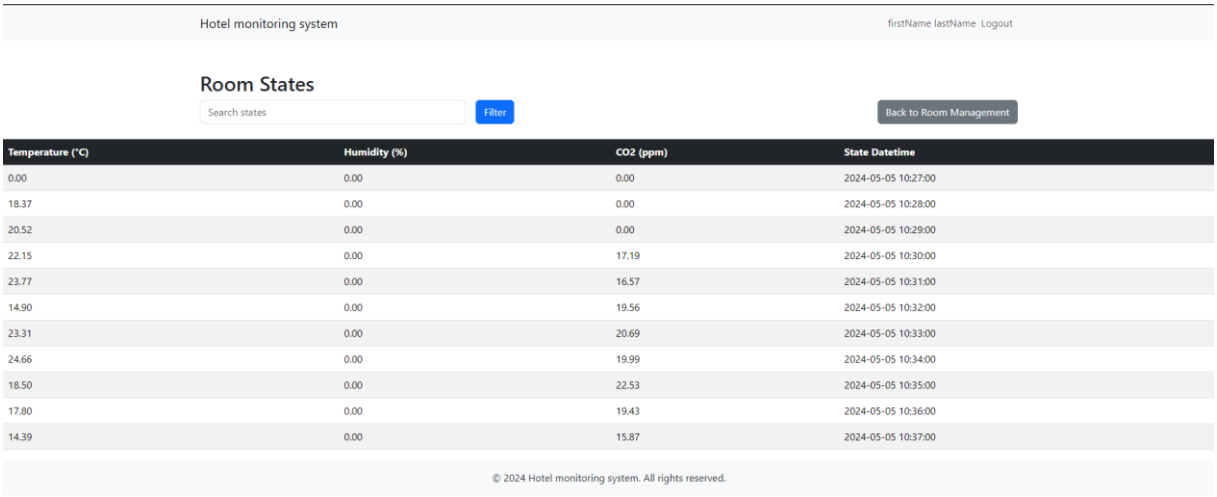
Uživatel může kliknout na tlačítko "Add Room", které otevře formulář pro zadání nového pokoje. Formulář vyžaduje vyplnění názvu pokoje, čísla pokoje a čísla podlaží.

Po vyplnění potřebných údajů může uživatel kliknout na "Create" pro přidání pokoje do systému nebo "Cancel" pro zrušení operace.

Prohlížení a správa existujících pokojů:

Stránka zobrazuje tabulku s existujícími pokoji, kde každý řádek reprezentuje jeden pokoj. U každého pokoje jsou zobrazeny informace jako ID, název pokoje, číslo pokoje a číslo podlaží.

U každého pokoje jsou dostupné akce pro zobrazení historie dat (tlačítko "History"), zobrazení dat za poslední hodinu (tlačítko "Last hour"), prohlížení senzorů v pokoji (tlačítko "Sensors"), úpravy informací o pokoji (tlačítko "Edit") a mazání pokoje (tlačítko "Delete").



Obrázek 22 Ústředna hotelů historie stavů místnosti

Na této stránce systému monitorování hotelu může uživatel provádět následující akce:

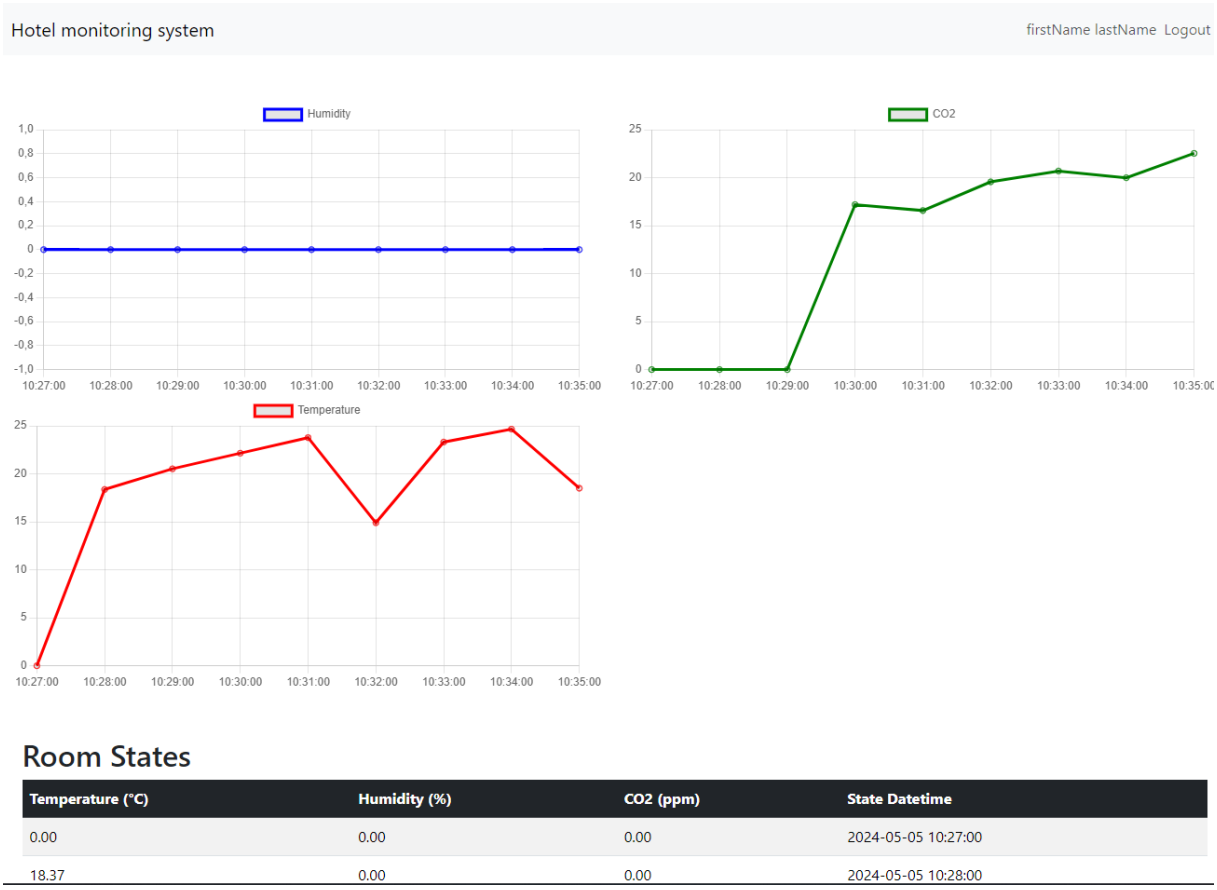
Filtrování stavů místností: Uživatel může použít pole "Search states" pro filtrování zobrazených dat na základě specifických kritérií.

Zobrazit data místností: Uživatel může prohlížet stavy jednotlivých místností, které zahrnují teplotu (°C), vlhkost (%) a koncentraci CO2 (ppm) spolu s datem a časem, kdy byla data zaznamenána.

Použít filtr: Po zadání hledané hodnoty do pole pro vyhledávání může uživatel kliknout na tlačítko "Filter" pro aplikaci filtru a zúžení zobrazených dat.

Návrat do správy místností: Kliknutím na odkaz "Back to Room Management" se uživatel může vrátit na stránku pro správu místností.

Stránka tedy umožňuje uživateli efektivně spravovat a přehlížet stavy místností v rámci hotelu, poskytuje možnost filtrování pro lepší orientaci v datech a nabízí snadný návrat do dalších částí systému pro správu místností.



Obrázek 23 Ústředna hotelů poslední hodina stavů místnosti

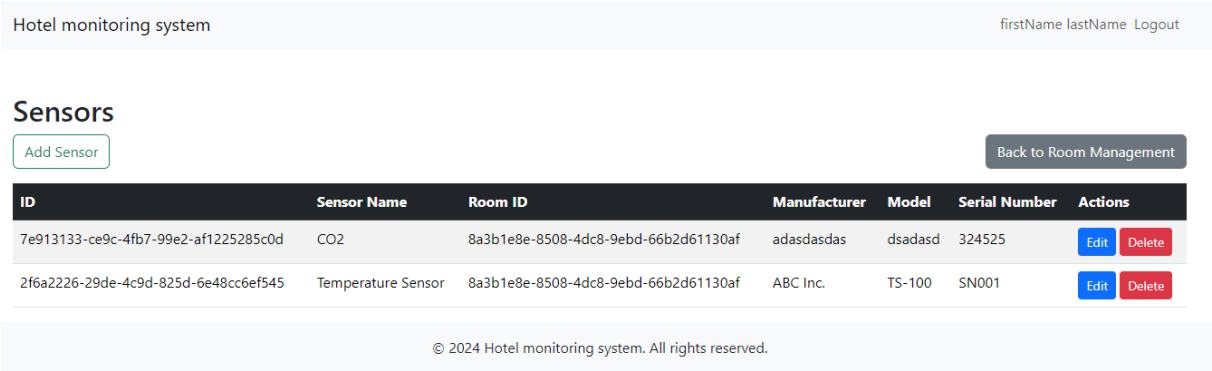
Filtrace a hledání pokojů:

Uživatel může použít filtrační a vyhledávací nástroje na začátku stránky pro rychlé vyhledání konkrétního pokoje. Je možné vyhledávat podle názvu a řadit výsledky podle různých kritérií (např. vzestupně/sestupně).

Aktualizace a mazání pokojů:

Uživatel může aktualizovat informace o existujících pokojích kliknutím na tlačítko "Edit" nebo může pokoje odstranit z systému kliknutím na tlačítko "Delete". Obě tyto akce jsou prováděny přímo z tabulky na hlavní stránce pro správu pokojů.

Tato stránka umožňuje uživatelům efektivně spravovat pokoje v rámci hotelu, zahrnující aktualizace stavu a monitorování prostředí skrze integrované senzory, což je klíčové pro správu moderního hotelového zařízení.



Obrázek 24 Ústředna hotelů správa sensorů místnosti

Na zobrazené stránce uživatel může provádět následující akce v rámci systému monitorování hotelu:

Přidat senzor do místnosti: Kliknutím na tlačítko "Add Sensor" může uživatel přidat nový senzor do systému.

Editovat senzor místnosti: U každého senzoru je tlačítko "Edit", které umožňuje uživateli upravit informace o daném senzoru.

Odstranit senzor místnosti: Tlačítko "Delete" umožňuje uživateli odstranit senzor ze systému.

Zobrazit a spravovat informace o senzorech: Uživatel může prohlížet seznam senzorů, kde každý řádek obsahuje informace jako ID, název senzoru, ID místnosti, výrobce, model a sériové číslo senzoru.

Návrat do správy místností: Kliknutím na odkaz "Back to Room Management" se uživatel může vrátit na stránku pro správu místností.

Celkově tato stránka umožňuje uživateli efektivně spravovat senzory v dané místnosti, což zahrnuje přidávání, úpravy a odstraňování senzorů, a také návrat do správy místností systému pro další správu.

ZÁVĚR

Tato práce úspěšně demonstruje návrh a implementaci pokročilého systému pro monitorování hotelů s využitím technologií Internetu věcí (IoT). Byly prozkoumány a implementovány klíčové webové služby jako REST, GraphQL, SOAP a gRPC, které podporují efektivní komunikaci a správu dat v rozsáhlých budovách. Praktická část práce ukazuje integraci těchto technologií v reálných aplikacích, což vede k lepší automatizaci, správě a bezpečnosti hotelového provozu.

Přestože byly dosaženy pozitivní výsledky, existují oblasti, kde by systém mohl být dále vylepšen:

Optimalizace výkonu - I když byly dosaženy dobré výsledky v rychlosti a spolehlivosti systému, další vylepšení a optimalizace infrastruktury mohou přinést lepší škálovatelnost a snížení latence, zejména při práci s velkým množstvím dat.

Uživatelské rozhraní - Ačkoliv uživatelské rozhraní poskytuje funkční a praktické řešení pro správu systému, další práce na jeho intuitivnosti a responzivitě by mohla zlepšit uživatelskou spokojenost a efektivitu práce.

Závěrem, tento systém pro monitoring hotelů s využitím IoT technologií nabízí robustní řešení pro zlepšení operací v hotelovém průmyslu. Budoucí vývoj by měl zahrnovat reakci na technologický pokrok a měnící se bezpečnostní požadavky, aby systém zůstal efektivní, bezpečný a na špičkové úrovni.

POUŽITÁ LITERATURA

[1] ORACLE.COM, c1995–2024. What is IoT? In: Oracle.com [online]. ORACLE CORPORATION [cit. 2024-04-05]. Dostupné z: https://www.oracle.com/internet-of-things/what-is-iot/

[2] CLOUDFLARE.COM, 2024. What is the Internet? In: Cloudflare.com [online]. Cloudflare, inc [cit. 2024-04-05]. Dostupné z: https://www.cloudflare.com/learning/network-layer/how-does-the-internet-work/

[3] TECHTARGET.COM, c2019–2024. REST API (RESTful API) In: TechTarget.com [online]. TechTarget [cit. 2024-04-05]. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API

[4] TECHTARGET.COM, c2019–2024. REST API (RESTful API) In: TechTarget.com [online]. TechTarget [cit. 2024-04-05]. Dostupné z: https://www.techtarget.com/searchapparchitecture/definition/REST-REpresentational-State-Transfer

[5] Lee Byron, 2015. GraphQL: A data query language In: Engineering.fb.com [online]. [cit. 2024-05-05]. Dostupné z: https://engineering.fb.com/2015/09/14/core-infra/graphql-a-data-query-language

[6] Riley Conrardy, 2023. GraphQL: Pros & Cons In: medium.com [online]. [cit. 2024-05-05]. Dostupné z: https://medium.com/@conrardy/graphql-pros-cons-23f8995752eb

[7] GEEKSFORGEEKS.ORG. Basics of SOAP – Simple Object Access Protocol. In: geeksforgeeks.org [online]. Omkarchalke [cit. 2024-05-05]. Dostupné z: https://www.geeksforgeeks.org/basics-of-soap-simple-object-access-protocol

[8] LINKEDIN.COM. What are the pros and cons of SOAP and REST for security and performance?. In: linkedin.com [online]. LinkedIn Corporation [cit. 2024-05-05]. Dostupné z: https://www.linkedin.com/advice/0/what-pros-cons-soap-rest-security-performance

[9] GRPC.IO. About gRPC. In: grpc.io [online]. gRPC [cit. 2024-05-05]. Dostupné z: https://grpc.io/about

[10] The Deca Dose, 2023. gRPC: A High-Performance, Remote Procedure Call (RPC) Framework In: medium.com [online]. [cit. 2024-05-05]. Dostupné z: https://medium.com/@vikramgyawali57/grpc-a-high-performance-remote-procedure-call-rpc-framework-dcbcdeb8dbd6

[11] TECHTARGET.COM, c2019–2024. An overview of IoT sensor types and challenges In: TechTarget.com [online]. TechTarget [cit. 2024-04-05]. Dostupné z: https://www.techtarget.com/iotagenda/tip/An-overview-of-IoT-sensor-types-and-challenges

[12] AWS.AMAZON.COM. What is MQTT? [online]. Amazon Web Services, Inc. [cit. 2024-04-05]. Dostupné z: https://aws.amazon.com/what-is/mqtt

[13] Harshvardhan Mishra, 2020. What is CoAP Protocol | CoAP Protocol Introduction | Overview In: medium.com [online]. [cit. 2024-05-05]. Dostupné z: https://medium.com/@harshhvm/what-is-coap-protocol-coap-protocol-introduction-overview-3e8bac4d7f8e

[14] CLOUDFLARE.COM, 2024. What is HTTP? In: Cloudflare.com [online]. Cloudflare, inc. [cit. 2024-05-05]. Dostupné z: https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http

[15] Mdn web docs. An overview of HTTP In: developer.mozilla.org [online]. Mdn web docs [cit. 2024-05-05]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[16] AMQP is the Internet Protocol for Business In: amqp.org [online]. AMQP [cit. 2024-05-05]. Dostupné z: https://www.amqp.org/about/what

[17] AMQP 0-9-1 Model Explained In: rabbitmq.com [online]. Rabbit MQ [cit. 2024-05-05]. Dostupné z: https://www.rabbitmq.com/tutorials/amqp-concepts

[18] GEEKSFORGEEKS.ORG. Functional vs Non Functional Requirements. In: geeksforgeeks.org [online]. chitrasingla2001 [cit. 2024-05-05]. Dostupné z: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements

[19] About SQLite In: sqlite.org [online]. SQLite [cit. 2024-05-05]. Dostupné z: https://www.sqlite.org/about.html

[20] SPRING.IO. Why Spring? In: sqlite.org [online]. Spring by VMware Tanzu [cit. 2024-05-05]. Dostupné z: https://spring.io/why-spring