Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Návrh a implementace webové aplikace pro správu a tvorbu revizí s využitím QR kódů

Jakub Joukl

Diplomová práce

2025

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jakub Joukl**
Osobní číslo:	**I23276**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Návrh a implementace webové aplikace pro správu a tvorbu revizí s využitím QR kódů**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

V teoretické části práce budou popsány současné trendy a nástroje v oblasti aplikací pro správu a tvorbu revizí a budou popsány výhody a nevýhody využívání webových aplikací. Dále pak budou popsány veškeré technologie, které budou v rámci realizace diplomové práce využity.
V rámci praktické části diplomové práce bude prvně třeba provést sběr a analýzu funkčních a nefunkčních požadavků. Po této části bude pozornost věnována návrhu vhodného databázového modelu a celkové koncepci webové aplikace včetně UI. V další kroku bude následovat vlastní implementace webové aplikace pro správu revizí, a to s využitím frameworku JAVA Spring a technologie Vaadin. Systém musí být řešen i z pohledu různých uživatelských rolí a více uživatelů a bude umožňovat skenování QR kódů, které budou provázány na jednotlivé revize.Webová aplikace bude nasazena pomocí kontejnerového systému Docker.

Rozsah pracovní zprávy: **50-60 stran**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

CARNELL, John. *Beginning spring boot 2: applications and microservices with the spring framework*. New York, NY: Springer Science Business Media, 2017. ISBN 978-148-4229-309.
Vaadin Team. *Book of Vaadin: Vaadin 14* Publisher: Independently published, 2019. 571 pages. ISBN 978-1692121440.

Vedoucí diplomové práce: **Ing. Jan Fikejz, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **31. října 2024**
Termín odevzdání diplomové práce: **23. května 2025**

L.S.

prof. Ing. Petr Doležel, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 29. listopadu 2024

Prohlašuji:

Práci s názvem „Návrh a implementace webové aplikace pro správu a tvorbu revizí s využitím QR kódů“ 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 16. 5. 2025

Jakub Joukl

PODĚKOVÁNÍ

Chtěl bych poděkovat Ing. Janu Fikejzovi, Ph.D. za odborné vedení práce, za veškerou pomoc, korekturu a veškeré konzultace v průběhu zpracování této práce. Také bych chtěl poděkovat rodině za podporu během celé doby studia.

ANOTACE

Cílem této diplomové práce je navrhnout a vyvinout webové aplikace pro správu a tvorbu revizí s využitím QR kódů. Za tímto účelem je nejprve provedena rešerše vybraných již existujících řešení. Po rešerši již existujících řešení následuje rešerše použitých technologií ve výsledné aplikaci. Další část práce se zabývá jak teoretickou rovinou analýzy aplikace a tvorby datového modelu, tak i analýzou a tvorbou modelu pro v této práci vytvořenou aplikaci. Na závěr následuje popis implementace aplikace se zaměřením na klíčové a problematické části vytvořené aplikace.

KLÍČOVÁ SLOVA

Systémy pro správu revizí, Java, Spring Boot, Vaadin Flow, Docker, datový model, Maven

TITLE

Design and implementation of a web application for managing and creating revisions using QR codes

ANNOTATION

The objective of this master's thesis is to design and develop a web application for managing and creating revisions using QR codes. To achieve this goal, the thesis first presents a review of selected existing solutions. The review is followed by an overview of the technologies used in the developed application. Following section focuses on both the theoretical foundations of application analysis and data model design, as well as the practical analysis and construction of a model used in the developed application. The thesis concludes with a detailed description of the application’s implementation, emphasizing the key components and addressing the most challenging aspects encountered during development.

KEYWORDS

Revision management systems, Java, Spring Boot, Vaadin Flow, Docker, data model, Maven

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM TABULEK

SEZNAM ZKRATEK A ZNAČEK

AOP – Aspect Oriented Programming

API – application programming interface

ARES – Administrativní registr ekonomických subjektů

Boilerplate – opakující se, standardizovaný kód, není specifický pro danou aplikaci

CI/CD – Continuous integration and continuous delivery

CLI – Command Line interface

CRUD – Create, Read, Update, Delete – základní operace v perzistentním úložišti dat

CSRF – Cross-Site Request Forgery

CSS – Cascading Style Sheets

ČSN – Česká technická norma

DB – Databáze

DevOps – Development and Operations

DevSecOps – Development and Security and Operations

DI – Dependency Injection

DIČ – Daňové identifikační číslo

DOM – Document Object Model

EPSG:4326 – Světový geodetický systém 1984

EPSG:5514 – Projected coordinate system for Czechia, S-JTSK / Krovak East North

HTML – HyperText Markup Language

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

IČO – Identifikační číslo osoby

IoC – Inversion of Control

IoT – Internet of Things

ISO – International Organization for Standardization

JAR – Java archive

JMS – Java Message Service

JMX – Java Management Extensions

JSON – JavaScript Object Notation

JVM – Java virtual machine

MVC – Model View Controller

ORM – Object Relational Mapping

OS – Operační systém

POJO – Plain Old Java Object

POM – Project object model

RDBMS – Relational database management system

REST – Representational state transfer

RÚIAN – Registr územní identifikace, adres a nemovitostí

SDK – Software development kit

SQL – Structured Query Language

SQL – Structured Query Language

SSL – Secure Sockets Layer

STN – Slovenská technická norma

UI – User Interface

URL – Uniform Resource Locator

WAR – Web Application Archive

WORA – Write once, run anywhere

XSS – Cross-Site Scripting

ÚVOD

V současné době s rychlým rozvojem digitálních technologií a širokým rozšířením chytrých mobilních zařízení roste poptávka po digitalizaci napříč všemi oblastmi každodenního profesního i soukromého života. Oblast správy revizí vyhrazených elektrických zařízení není výjimkou z tohoto trendu.

Bez využití digitálních technologií je běžnou praxí, že si revizní technik veškerou agendu v oblasti správy revizí a plánování jejich termínů hlídá a spravuje sám. Obdobný problém se týká i samotných majitelů elektrických spotřebičů v případě, že se rozhodnou nevyužít žádný ze softwarů pro správu revizí – zákonnou povinností klientů je zajistit, aby jejich elektrická zařízení měla platné revize. Majitelé spotřebičů by tedy měli mít přístup k seznam revizí svých elektrických zařízení.

S neustálým rozvojem technologií dochází i k růstu počtu vyhrazených elektrických zařízení, u nichž je nutné pravidelně provádět revize. Tento trend klade stále vyšší nároky na revizní techniky, kteří musí evidovat a sledovat velké množství termínů, řídit plánování revizí a spravovat rozsáhlé objemy dat souvisejících s jednotlivými zařízeními a jejich dokumentací. Vzhledem k rostoucímu objemu těchto činností je zřejmé, že pro zajištění efektivity práce techniků a zlepšení přehledu o revizích na straně klientů je velmi přínosné využít specializované softwarové řešení. Takový systém by měl umožňovat snadnou organizaci revizí a souvisejících dokumentů, včetně jejich jednoznačného přiřazení ke konkrétním organizacím a provedeným kontrolám.

Cílem této práce je návrh a implementace aplikace pro správu revizí vyhrazených elektrických zařízení, zahrnující i efektivní správu jednotlivých spotřebičů. Hlavními funkcionalitami vyvinuté aplikace by mělo být usnadnění správy revizí pro revizní techniky, organizace revizí dle klientů, organizace revizí, možnosti vyhledávání revizí, možnost generování QR kódů pro jednotlivé revidované spotřebiče, možnost generování revizních protokolů pro jednotlivé elektrické spotřebiče.

V teoretické části práce jsou nejprve popsány systémy pro správu revizí a 3 vybrané již existující aplikace pro správu revizí, poté následuje popis použitých technologií.

Následující část diplomové práce propojuje teoretické poznatky s jejich praktickým využitím. Nejprve je zde teoreticky popsána oblast analýzy vyvíjené aplikace, konkrétně sběr funkčních a nefunkčních požadavků, návrh business procesů a uživatelského rozhraní. Na každou z těchto

oblastí poté navazuje praktická ukázka jejich využití v rámci této diplomové práce. Poté je v práci uveden teoretický popis tvorby a návrhu datového modelu aplikace, za kterým následuje popis návrhu a tvorby datového a databázového modelu aplikace včetně popisu komunikačního schématu výsledného systému.

Praktická část práce se věnuje implementaci systému, konkrétně rozvržení aplikace a popisu jednotlivých implementovaných částí. Na implementaci navazuje část zaměřená na testování a nasazení aplikace.

1. SYSTÉMY A APLIKACE PRO SPRÁVU REVIZÍ

V této kapitole jsou nejprve definovány vlastnosti obecných systémů pro správu a poté jsou zde uvedeny existující komerční aplikace pro správu revizí.

1.1. Systémy pro správu

Jedná se o softwarové aplikace, které poskytují formuláře pro zadávání hodnot, významně zvyšují efektivitu práce a napomáhají k systematickému dodržování stanovených standardů. Implementace softwarového řešení umožňuje nejen automatizaci plánování revizí a s tím souvisejících činností, ale také standardizaci formulářů pro zaznamenávání údajů o revizích. Součástí těchto systémů bývají rovněž datové reporty a rozsáhlá datová základna vhodná pro účely datové analytiky. Další výhodou je možnost integrace s existujícími službami a aplikacemi. Celkově použití softwarového řešení přispívá k lepší organizaci a vyšší přehlednosti revizní dokumentace ve srovnání s tradičními papírovými formuláři. [21]

1.2. Existující aplikace pro správu revizí

1.2.1. Revelo

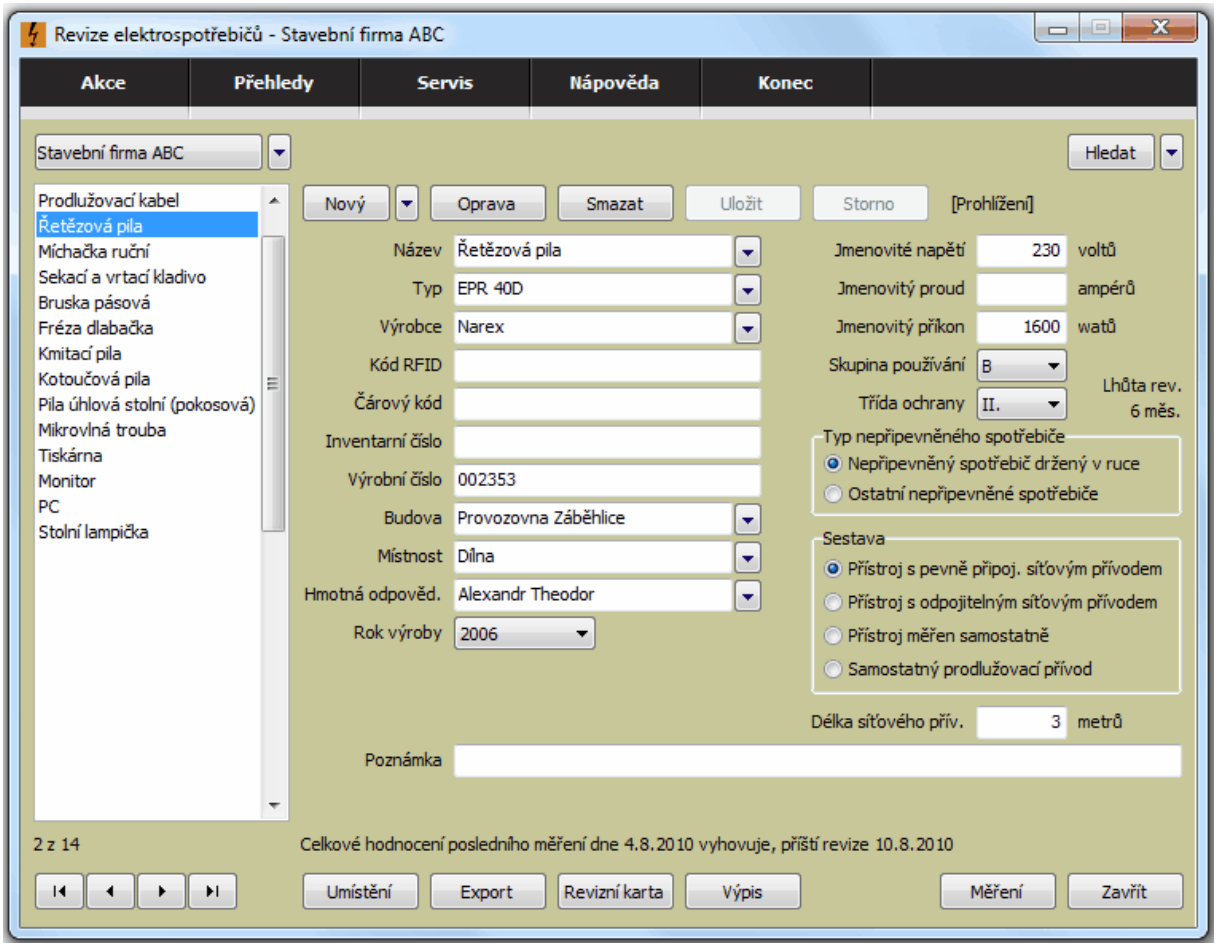
Jedná se o desktopovou aplikaci, který je určený pro platformu Windows, sloužící pro správu revizí elektrických spotřebičů. Aplikace splňuje jak ČSN 33 1600 ed.2, tak i STN 33 1600 a STN 33 1610. [23]

Funkcemi, který aplikace Revelo nabízí, jsou [23][24]:

- Ukládání spotřebičů do interní databáze, díky čemuž lze spotřebiče dohledávat na základě čárových kódů, RFID, identifikačního čísla, výrobního čísla a k dohledanému spotřebiči se pak doplní již uložené údaje
- Aplikace je schopna hlídat termíny revizí
- Možnost vytváření čárových kódů v běžných standardech
- Možnost generování a tisku sestav do PDF
- Možnost vytváření vlastních tiskových sestav nebo úpravy existujících pomocí návrháře
- Vygenerované dokumenty je možné elektronicky podepsat
- Optimalizace pro mobilní zařízení (myšleno netbooky)
- Existence odlehčené verze, „Revelo Expres“
- Aplikace funguje offline
- Rozsáhlá uživatelská dokumentace popisující práci s jednotlivými okny aplikace
- Číselník zákazníků pro přidání, úpravu, odebrání zákazníků

- Číselník techniků pro přidání, úpravu, odebrání techniků
- Číselník měřících přístrojů pro přidání, úpravu, odebrání měřících přístrojů
- Nastavení limitů pro jednotlivé třídy ochrany
- Import a export měření z/v CSV a XML

Na obrázku 1 je zobrazena karta spotřebiče v aplikaci Revelo.



Obrázek 1 - karta spotřebiče [24]

1.2.2. Aptien

Na webových stránkách je představen systém určený pro systematické sledování, plánování a evidenci revizí a kontrol v rámci organizací. Jedná se o submodul rozsáhlé webové aplikace, která, kromě správy revizí, slouží k administraci a řízení firmy, k řízení komunikace mezi týmy zaměstnanců i mezi samotnými zaměstnanci.

Aplikace není určena pouze pro realizaci revizí elektrických zařízení, ale podporuje také provádění dalších typů revizí, jako jsou například kontroly budov, místností či jiného majetku organizace. Uživatelské rozhraní aplikace je plně responzivní – nejenže se automaticky přizpůsobuje velikosti okna, ale uživatel má zároveň možnost explicitně přepínat mezi verzí

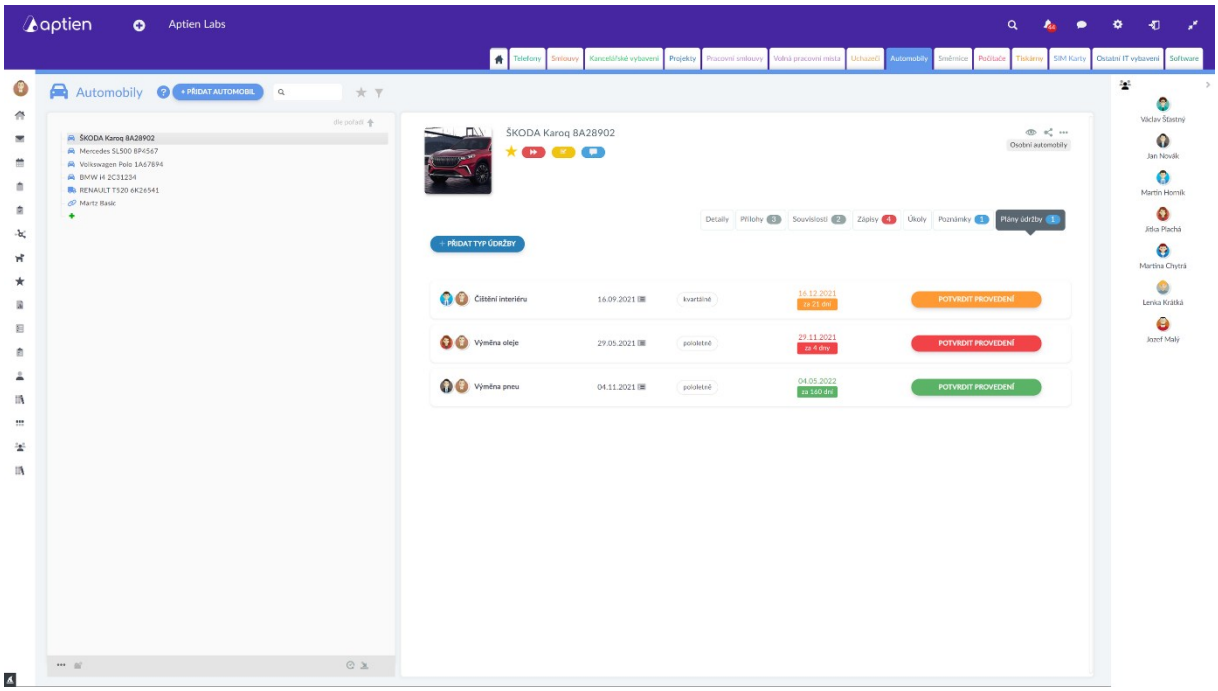
optimalizovanou pro kancelářské prostředí a mobilním zobrazením. Mezi další přednosti patří jazyková lokalizace do češtiny a angličtiny, díky níž je aplikace vhodná i pro mezinárodní uživatele.

V aplikaci je možné individuálně nastavit plán revizí pro každé jednotlivé zařízení. U každé revize lze specifikovat, zda se jedná o jednorázovou kontrolu, nebo o pravidelnou revizi v definovaných časových intervalech. Pro oba typy kontrol lze dále nastavit upozornění s časovým předstihem před blížícím se termínem revize. Přehled zařízení je k dispozici jak na kartách reprezentujících jednotlivé položky majetku společnosti, tak volitelně i na kartách s jemnějším členěním dle kategorií (např. výpočetní technika). V případě filtrování podle kategorie se zobrazí pouze zařízení odpovídající vybrané skupině.

Aplikace rovněž uchovává historii zařízení, čímž umožňuje zpětné dohledání provedených změn. Ke každému zařízení či položce majetku lze přikládat dokumentaci prostřednictvím karet konkrétních položek, což zajišťuje úplnost a dostupnost dokumentů souvisejících s revizemi.

Součástí modulu pro plánování revizí je také podpora hlášení závad a zadávání požadavků na opravu, což přispívá ke zrychlení procesu jejich řešení. V rámci aplikace je dále možné spravovat uživatelské účty, přiřazovat uživatelům role a definovat jejich oprávnění. Díky této granulární správě přístupových práv lze efektivně omezit přístup k určitým částem systému, například pouze pro revizní techniky nebo jiné specifické uživatelské skupiny. [25][26]

Níže se nachází obrázek číslo 2, na kterém je zobrazeno okno s revizemi v aplikaci Aptien.



Obrázek 2 - okno s revizemi v Aptien [25]

1.2.3. FACMAN

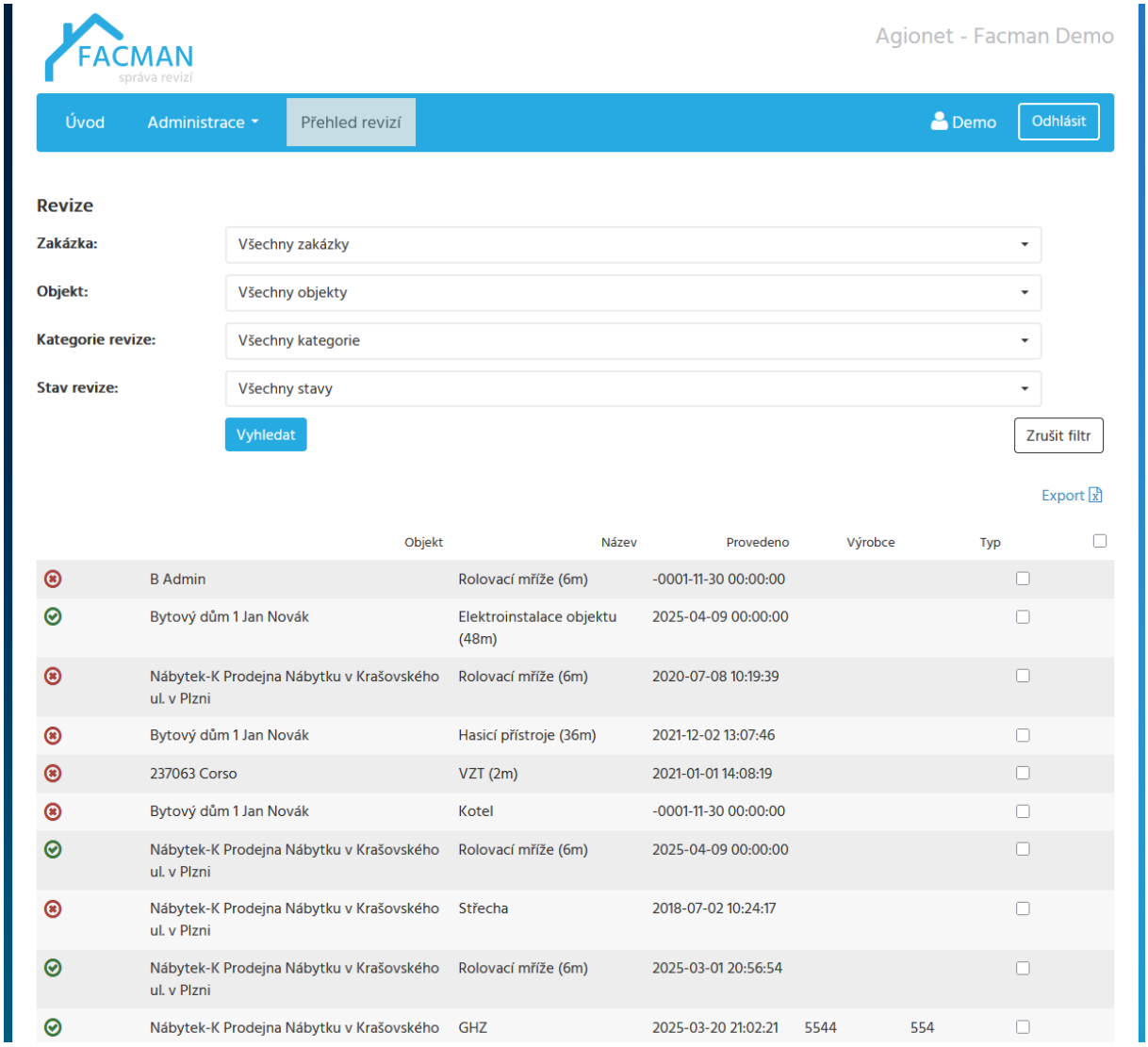
Aplikace FACMAN představuje komplexní informační systém určený pro správu a údržbu majetku, budov a dalších objektů. Je navržena tak, aby centralizovala evidenci a řízení provozních činností v oblasti facility managementu. Hlavním cílem je zefektivnit plánování, provádění a kontrolu revizí, údržby a oprav, což přispívá k bezpečnému a spolehlivému provozu zařízení a celkovému snížení provozních nákladů. [27]

Hlavní funkcionality aplikace FACMAN jsou [27]:

- Správa revizí a údržby – umožňuje plánovat, evidovat a monitorovat pravidelné revize elektrických zařízení, budov, místností a dalšího majetku. Systém automaticky generuje upozornění na blížící se termíny úkonů, čímž minimalizuje riziko opomenutí nezbytných úkonů.
- Dokumentace a archivace – poskytuje možnost centrální archivace dokumentace související s prováděnými revizemi, údržbou a opravami. To zahrnuje technické zprávy, certifikáty, fotky a další potřebné dokumenty, což usnadňuje následné kontroly a audity.
- Reportování a analýza – aplikace generuje přehledné reporty a statistiky, které podporují strategické rozhodování. Díky tomu lze efektivně vyhodnocovat stav majetku a optimalizovat provozní náklady.

- Uživatelské rozhraní a mobilita – intuitivní a uživatelsky přívětivé rozhraní umožňuje snadný přístup a obsluhu systému jak z desktopových, tak z mobilních zařízení. To zajišťuje flexibilitu a dostupnost informací kdykoli a odkudkoli.

Na následujícím obrázku 3 je zobrazen přehled revizí v aplikaci FACMAN.



Obrázek 3 – přehled revizí v aplikaci FACMAN [27]

2. POUŽITÉ TECHNOLOGIE

V této kapitole jsou popsány technologie, které byly využity pro tvorbu praktické části této diplomové práce.

2.1. Java

Java je programovací jazyk, jehož první verze byla vydána v roce 1995. Jedná se o populární celosvětově rozšířenou platformu pro vývoj aplikací a služeb. [1]

Java je objektově orientovaný, vysoce rozšířený programovací jazyk s podporou multiplatformnosti. V současnosti je používán miliardami zařízení. Je navržen tak, aby byl nezávislý na konkrétní hardwarové či softwarové platformě, což umožňuje spouštění téhož kódu na širokém spektru zařízení – od osobních počítačů s operačním systémem Windows až po mobilní zařízení. Tato vlastnost výrazně přispívá k přenositelnosti a opakované použitelnosti softwaru. V souvislosti s multiplatformností Javy se často používá slogan *„Write Once, Run Anywhere“* (zkráceně WORA), který vystihuje schopnost jazyka eliminovat nutnost přizpůsobování kódu pro jednotlivé platformy.

Multiplatformnost jazyka Java je zajištěna specifickým způsobem překladu zdrojového kódu. Při kompilaci není kód přeložen přímo do strojového kódu konkrétní platformy, ale do prostředního formátu zvaného *bytecode*. Tento *bytecode* je následně interpretován nebo spouštěn prostřednictvím Java Virtual Machine (JVM), která je dostupná pro různé operační systémy a hardwarové architektury. Díky tomu je možné program napsaný v Javě spustit na jakémkoliv zařízení, které JVM podporuje, bez nutnosti úprav zdrojového kódu.

Objektově orientovaný přístup Javy znamená, že zdrojový kód programu je rozdělen do tříd a objektů, a ne do procedur a příkazů.

Díky kombinaci objektově orientovaného přístupu a multiplatformnosti představuje Java velmi všestranný programovací jazyk. Tyto vlastnosti, spolu s relativní jednoduchostí při osvojování základních principů, z ní činí oblíbenou volbu jak pro začínající programátory, tak i pro technologické společnosti při vývoji softwaru. [2]

Java nachází své uplatnění zejména v těchto oborech [2]:

- Vývoj mobilních aplikací
- Vývoj webových aplikací
- Podnikový software
- Hry
- IoT aplikace

2.2. Spring

Jedná se o Framework určený k vývoji aplikací v Javě. Cílem Spring frameworku je zjednodušení vývoje podnikových aplikací a celkové zpřehlednění a zjednodušení zdrojového kódu.

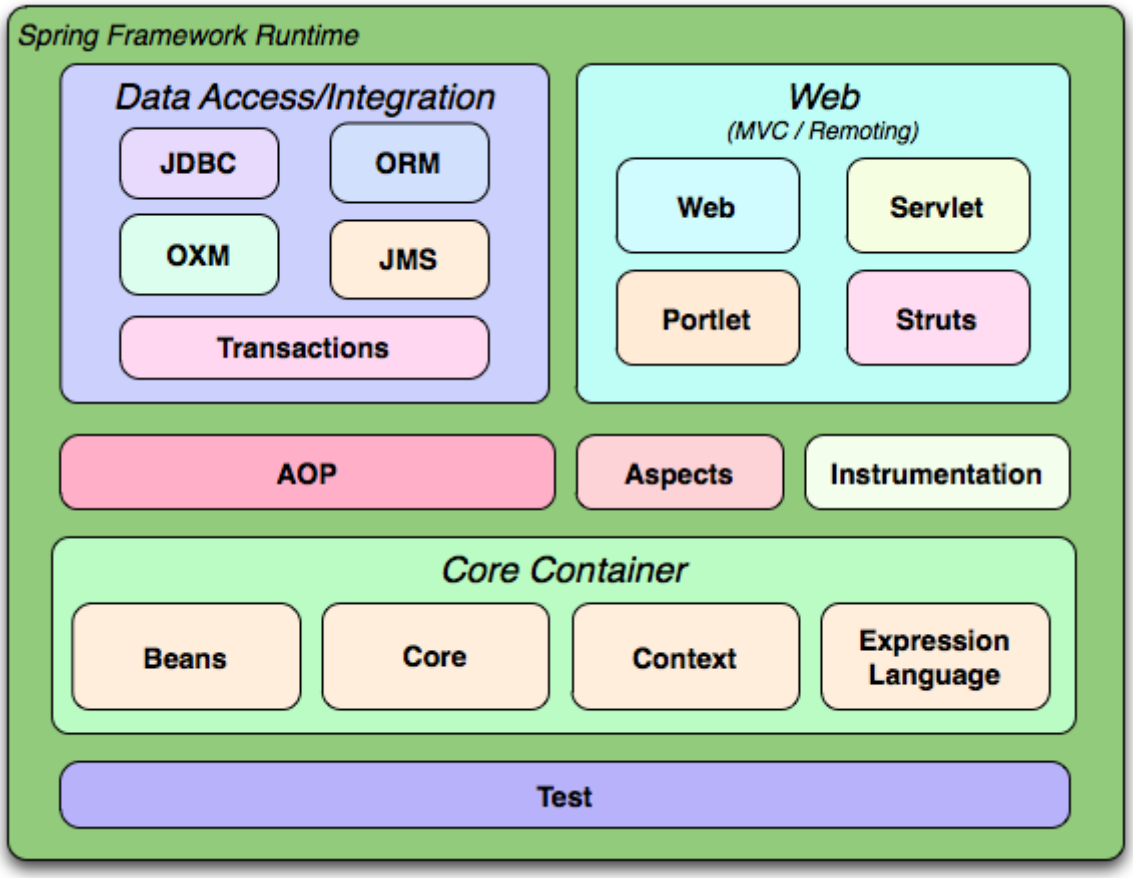
Mezi výhody, které zjednodušují vývoj aplikací, patří například [3]:

- Provedení Java metody v DB transakci bez nutnosti použití transakčního API.
- Převést místní metodu v Javě na vzdálenou proceduru, aniž by bylo nutné se zabývat vzdálenými API.
- Převést místní metodu v Javě na operaci pro správu, aniž by bylo nutné využít JMX API
- Převést místní metodu v Javě na zpracovatele zpráv, bez nutnosti se se zabývat JMS API

Jednou z klíčových vlastností frameworku Spring je podpora principů Dependency Injection (DI) a Inversion of Control (IoC). Tyto principy umožňují automatizovanou správu závislostí mezi objekty, zejména prostřednictvím vytváření a poskytování instancí tříd (typicky ve formě singletonů) komponentám, které je vyžadují. Díky tomu je možné oddělit samotnou aplikační logiku od logiky zajišťující správu konfigurace a závislostí.

Co se týká vnitřní struktury frameworku, tak framework je rozdělen do cca 20 modulů, tyto moduly jsou pak seskupené dle funkcionalit, které poskytují.

Na následujícím obrázku 4 se nachází seznam modulů frameworku:



Obrázek 4 - přehled seskupení Spring modulů [3]

Jak je vidět na předcházejícím obrázku, tak Spring se skládá z následujících skupin modulů [3]:

- Core Container – jedná se o skupinu modulů, které jsou klíčové k funkci frameworku, zejména o DI a IoC, načítání hodnot z konfigurace, přístup ke Spring kontextu
- Data Access/Integration – jedná se o skupinu modulů zajišťujících přístup k databázi, ORM mapování Java POJO tříd na databázové tabulky a přijímání a zasílání zpráv
- Web – tato skupina modulů zajišťuje webovou funkcionalitu, jako je MVC a správná práce s nahranými soubory
- AOP a Instrumentation – úkolem této skupiny modulů je zajistit podporu pro aspekty
- Test – Tento modul zajišťuje podporu pro tvorbu a provádění testů pomocí JUnit nebo TestNG a správnou funkci Spring context v testech

[3]

2.3. Spring Boot

Spring Boot je postaven na Spring framework. Poskytuje stejnou funkcionalitu jako Spring, ale mnoho často používaných funkcionalit je v něm předkonfigurovaných. Tato vlastnost je velmi důležitá pro vytváření mikroservisní architektury a REST API – při použití tradičního Springu by bylo nutné každou aplikaci manuálně a detailně konfigurovat. Naproti tomu Spring Boot díky své autokonfiguraci eliminuje značnou část často se opakujících konfiguračních kroků a anotací, čímž usnadňuje vývoj a zrychluje nasazení aplikací. Veškerou autokonfiguraci je možné přepsat vlastní konfigurací. [4]

Další významnou výhodou frameworku Spring Boot je integrace vestavěných aplikačních serverů, jako jsou Tomcat, Jetty nebo Undertow. Díky tomu není nutné vytvářet archivní soubory ve formátu WAR ani je nasazovat na externí aplikační server, což výrazně zjednodušuje nasazení a zkracuje čas potřebný k uvedení aplikace do provozu. [5]

Hlavní rozdíly mezi Spring a Spring Boot jsou shrnuty v následující tabulce 1 [4]:

Tabulka 1 - srovnání Spring a Spring Boot [4]

Spring	Spring Boot
Pro běh aplikace musíme explicitně nastavit server.	Server je vestavěn.
Je nutné psát mnoho konfigurace.	Díky autokonfiguraci není nutné psát tolik konfigurace.
Konfigurace zvyšuje počet řádků kódu.	Nižší množství řádků kódu díky autokonfiguraci.
Více boilerplate kódu	Redukce množství boilerplate kódu

2.4. Maven

Maven slouží k automatizaci sestavení aplikace do spustitelného kódu. Vývoj Maven začal díky potřebě standardizovat programátorské praktiky vedoucí k vytvoření sestavitelných projektů, mezi které patří standardizace jmenných konvencí, psaní testů, umístění testů v projektu, separátní proces sestavení testů bez nutnosti provádění úprav v projektu za účelem spuštění, provedení a vyhodnocení testů, dále se pak mezi další cíle řadí potřeba sjednotit a standardizovat proces sestavení projektu a jednotlivé kroky k sestavení projektu tak, aby bylo transparentnější, z jakých části a závislostí se projekt skládá. [6]

Hlavními cíli Maven jsou [6]:

- Zjednodušit a sjednotit proces sestavení projektu a správu závislostí
- Poskytnout informace o projektu
- Podpora dobrých praktik vývoje

Kromě toho, že Maven slouží k sestavení projektu, tak také poskytuje [6]:

- Seznam změn vytvořených přímo z verzovacího systému
- Křížově odkazované zdroje
- Seznamy e-mailových adres spravované projektem
- Závislosti používané projektem
- Reporty z unit testů včetně pokrytí
- Možnost použití pluginů třetích stran, které umožňují rozšířit standardní reporty

Jednou z důležitých vlastností nástroje Maven je standardizace struktury projektů, která usnadňuje spolupráci ve vývojových týmech a zajišťuje snadnou udržovatelnost kódu. Noví členové týmu se díky jednotnému způsobu správy závislostí a sestavení projektu rychle zorientují bez nutnosti složitého zaučování do specifik konkrétního nastavení. Další z vlastností je snadná a automatická správa závislostí, pluginů a případně i metadat – závislosti, pluginy a metadata je možné přidávat z centrálního repozitáře anebo ze soukromých firemních repozitářů. Maven také umožňuje na základě metadat projekt vyexportovat do JAR anebo WAR. Díky systému pro správu verzí lze také vytvářet různá sestavení označená konkrétními verzemi (tagy), které jednoznačně určují, z jakého stavu projektu dané sestavení vzniklo. Z metadat projektu je také možné vyexportovat reporty ohledně průběhu vývoje projektu, a to buď pomocí standardních Maven reportů anebo pomocí rozšířených reportů z pluginů. [7]

2.5. Vaadin Flow

Vaadin Flow je framework, jenž umožňuje napsat frontendové komponenty kompletně v jazyce Java a pomocí svých vnitřních funkcí je přeloží na odpovídající HTML, CSS, JavaScript. Backendové a frontendové reprezentace komponent jsou spolu provázané a synchronizované – provedení změn na backendu se odpovídajícím způsobem projeví i na frontendu.

2.5.1. Komunikace mezi frontendem a backendem

Provázání z frontendu na backend znamená, že poté, co uživatel na frontendu klikne na tlačítko, tak na backendu se pro dané tlačítko obslouží odpovídající událost, například zpracování odpovídajícího POST requestu.

Navázání backendové komponenty na frontendovou znamená, že poté, co se na backendu na základě definovaného chování provede definovaná akce, tak na frontendu se změny projeví odpovídajícím způsobem v DOM stránky prohlížeče.

Navigace na jiné URL adresy je ve Vaadin také řešena pomocí provázání backendu a frontendu. Při kliku na odkaz spravovaný Vaadinem je zabráněno defaultnímu chování v browseru a konečné chování pro navigaci je vyhodnoceno na backendu. Backend po provedení daného chování pošle na frontendu instrukce, jak má dále postupovat v navigaci, zejména jakým způsobem má překreslit zobrazený obsah a aktualizovat DOM. [8]

Navigaci je také možné spustit ze strany serveru pomocí příslušných metod Vaadin Flow frameworku, a to i s využitím parametrů v cestě URL adresy a parametrů dotazu, konkrétně se jedná například o metodu UI.navigate() a její další přípustné varianty dále popsané v odpovídajících dokumentacích. [9]

2.5.2. Zabezpečení

Vaadin Flow také poskytuje komplexní zabezpečení. Vzhledem k tomu, že se jedná o framework, který je na straně serveru, tak veškerá business logika a aplikační stavy a logika UI operací zůstávají na serveru a na frontend se posílají jen nezbytná data pro správnou funkci frontendu a zobrazení UI, díky tomu není z frontendu možné ovlivnit samotný průběh vnitřní logiky zpracování dat.

Uchovávání aplikačního stavu na backendu místo frontendu umožňuje backendu hlídat stav aplikace nezávisle na frontendu. Díky této vlastnosti backend uchovává informaci o stavu frontendových komponent, například zda má být komponenta pro uživatele viditelná nebo povolená. Pokud frontend pošle na backend data z neexistující komponenty anebo z prvku stránky, který by pro uživatele měl být v GUI prohlížeče neviditelný nebo vypnutý, tak backend je schopen nezávisle na frontendu tento stav rozpoznat, zabránit vykonání příchozího požadavku z této komponenty nebo prvku, zároveň je schopen takový požadavek zalogovat.

Kromě validace příchozích požadavků z frontendu na backend Vaadin Flow podporuje validaci příchozích dat, které směřují z frontendu na backend. Ačkoli se validace vstupů provádí jak na straně klienta, tak i na straně serveru, validace na frontendu není spolehlivá, protože ji lze z klientské strany snadno obejít nebo upravit přímo v prohlížeči. Z tohoto důvodu se Vaadin Flow na klientskou validaci nespoléhá a vždy provádí nezávislou kontrolu dat i na backendu. Kromě možnosti vlastní implementace, využití validace dat pomocí externích komponent nebo

validace na úrovni Java Bean je také možné použít předdefinované validátory dat, které Vaadin Flow poskytuje.

Vaadin Flow zabezpečuje komunikaci mezi frontendem a backendem s využitím standardního Java Servlet API pomocí jediného handleru HTTP požadavků, který se používá pro RPC, z toho důvodu není nutné vytvářet business logiku pro obsluhu webových služeb.

Další důležitou vlastností z oblasti bezpečnosti je nativní podpora pro HTTPS a SSL, takže ze strany vývojáře odpadá nutnost konfigurace pro HTTPS a SSL. [11]

Vzhledem k tomu, že Vaadin Flow je nezávislý na architektuře backendu, tak kromě validace vstupních dat neposkytuje žádnou dodatečnou ochranu proti útokům typu SQL Injections, ochrana proti těmto typům útoků musí být na backendu řešena jinými standardizovanými osvědčenými postupy a správnou manipulací se vstupními dat při práci s databázemi.

Pro komunikaci a požadavky spravované Vaadinem Vaadin Flow, bez nutnosti zásahu programátora, poskytuje ochranu před útoky typu CSRF, a to pomocí tzv. "Synchronizer Token Pattern". K tomu Vaadin Flow používá token Vaadin-Security-Key pro své interní UIDL requesty a pro WebSocket požadavky používá token Vaadin-Push-ID. Tyto tokeny jsou generovány pro každou instanci UI. Vaadin Flow podporuje vypnutí používání CSRF tokenům, ale vzhledem k problémům s bezpečností to není doporučeno.

Další ochranou, kterou Vaadin Flow poskytuje, je ochrana před útoky typu Cross-Site Scripting. Při správném použití ze strany programátora API na straně prohlížeče automaticky transformuje HTML značky na text, takže pokus o vložení například <script> tagu skončí transformací tohoto tagu na jeho textovou reprezentaci. Pokud se ze strany vývojáře objeví požadavek na přímou manipulaci s HTML jednotlivých Vaadin Flow komponent, které to umožňují, tak je doporučeno použít externí řešení na očištění a escapování těchto vstupů, jako je například Jsoup. Toto řešení není bohužel možné pro JavaScript skripty, jelikož escapování znaků by způsobilo nefunkčnost skriptu, Vaadin Flow však poskytuje řešení pro bezpečné předávání parametrů JavaScript skriptů. [12]

2.6. Docker

Jedná se o oper-source platformu pro vytváření, nasazení, spuštění, updatování a správu kontejnerů. Docker kontejnery jsou standardizovaná komponenta, která obsahuje zdrojový kód, knihovny operačního systému a závislosti potřebné ke spuštění kódu v jakémkoliv prostředí.

Docker, i přes to, že s rokem vydání první pro veřejnost dostupné verze 2013 není na poli kontejnerizace prvním nástrojem, je v současnosti nejvíce používaným nástrojem na kontejnerizaci, a to s podílem na trhu 82,84 %.

Kontejnerizace je možné provádět díky vlastnostem Linuxového kernelu, zejména díky izolaci procesů a možnosti virtualizace. Díky izolaci procesů a možnostem virtualizace v Linuxu je možné, aby kontejnery sdílely jen jednu instanci OS, tím se, v porovnání s virtuálními stroji, sníží nároky na úložiště na hardwaru, rychlost spuštění mikroslužeb, zvýší se přenositelnost mikroslužeb a díky rozhraní Docker, které umožňuje jednoduchou správu kontejnerů, se i sníží složitost ovládání stavu mikroslužeb.

Bezpečnost v Dockeru je zajištěna prostřednictvím izolace kontejnerů od hostitelského operačního systému a ostatních kontejnerů. Nicméně, protože tato izolace není absolutní, Docker také implementuje politiku *zero trust* a využívá osvědčené praktiky z oblasti DevSecOps, aby minimalizoval bezpečnostní rizika.

Fyzická a příkazová architektura Dockeru se skládá z následujících části [10]:

- Docker host – fyzický anebo virtuální hostitel na kterém běží Docker
- Docker Engine – klient/server aplikace sestávající s Docker deamon, Docker API, které komunikuje s deamonem, CLI pro komunikaci s deamonem
- Docker deamon – jedná se o službu, která na základě příkazů z klienta vytváří a spravuje Docker kontejnery
- Docker client – pomocí CLI komunikuje s deamonem
- Docker objects – komponenty Dockeru pro balíčkování a distribuci aplikací, patří mezi ně image, kontejnery, síťě, svazky, pluginy
- Docker images – obsahují spustitelný kód aplikace a veškeré potřebné nástroje a závistlosti ke spuštění kontejneru
- Docker containers – spustitelné instance Docker images, se kterými je možné pracovat, beží v nich spuštěný kód
- Docker build – příkaz k sestavení Docker images
- Dockerfile – textový soubor obsahující instrukce k sestavení Docker image
- Docker documentation – dokumentace k Dockeru
- Docker Hub – veřejný repozitář Docker images, kde je možné stáhnout a sdílet vytvořené docker images

- Docker Desktop – desktopová aplikace na Mac a Windows, která zahrnuje Docker Engine, Docker client, Docker Compose atd.
- Docker registry – open-source škálovatelné uložiště sloužící k distribuci Docker images, image je možné verzovat a tagovat
- Docker plugins – pluginy rozšiřující funkcionalitu Dockeru, kromě existujících přeinstalovaných oficiálních pluginů je možné použít i pluginy třetích stran
- Docker extensions – doplňky pro rozšíření Dockeru pomocí aplikací třetích stran
- Docker Compose – Za pomocí konfigurace napsané v YAML souboru je možné pomocí jednoho příkazu nasadit a spustit více kontejnerů naráz, v souboru je také možné definovat dokumentační kroky, proměnné a svazky, to vývojářům usnadňuje správu vícekontejnerových aplikací, Docker svazků a dokumentace

Z důvodu rostoucímu rozmachu mikroservísní architektury vzniká potřeba pro spouštění a provoz stále většího množství spolu na sobě volně spojených a navzájem nezávislých služeb. Díky způsobu, jakým je kontejnerizace realizována, představuje efektivní metodu pro přenositelnost, organizaci, provoz, škálování, nasazování nových verzí a údržbu těchto mikroslužeb. Kromě mikroslužeb najde Docker své uplatnění i v oblasti migrace aplikací, dat a workload, v oblasti CI/CD, pro hybridní multicloudový deployment, v oblasti strojového účení, Containers as a service. Docker kontejnery také vyhovují praktikám DevOps. [10]

2.7. MySQL

Jedná se open-source RDBMS k ukládání a správě dat. Databáze je spolehlivá a technologicky vyspělá, na trhu se používá už téměř 30 let. Poskytuje řadu různých edic, které se liší vlastnosti a cenou. Další vlastností databáze je škálovatelnost díky replikační architektuře. Neméně důležitou vlastností, kterou MySQL zajišťuje, je vysoká míra dostupnosti, která je zajištěná díky replikačním technikám, které tato databáze poskytuje v případě odhalení poruchy, díky těmto mechanismům je možné zabránit ztrátě dat s nulovým downtime. V oblasti bezpečnosti a ochrany dat databáze splňuje požadavky řady různých směrnic, mezi které se řadí GDPR, a poskytuje funkcionality, mezi které se řadí šifrování dat a auditování. Vzhledem k tomu, že se jedná o relační databázi, tak data jsou ukládána do schémat do řádků a sloupců. Schéma definuje způsob organizace a uložení dat, jejich typ a obsahuje informace o relacích mezi jednotlivými tabulkami. Pro práci s databází se používá jazyk SQL, což je standardizovaný jazyk pro práci s SQL databázemi. MySQL podporuje transakce a tzv. ACID princip – atomicita, konzistence, izolace a trvalost zápisu. [13]

2.8. Amazon S3

Amazon S3, plným názvem Amazon Simple Storage Service, slouží k uchovávání velkého množství různých druhů dat. Jedná se o komerčně dostupné cloudové úložiště, které je, díky své škálovatelnosti a konfigurovatelnosti, určené jak pro soukromé osoby, tak i pro velké korporace. Toto datové uložiště nachází své uplatnění v oblasti data lakes, webových služeb, mobilních aplikací, IoT zařízení, archivace dat, podnikových aplikací, datových analytik.

Data jsou ukládána do tzv. Buckets. Pro porovnání s OS by se svojí funkcí daly Buckety přirovnat ke složkám. Buckety jsou identifikovány dle geografického regionu, v kterém se nachází, a poté také pomocí jedinečných identifikátorů – klíčů. Množství Bucketů je u uživatele standardně omezeno na 10 000, v případě potřeby většího množství Bucketů je nutné požádat Amazon o rozšíření množství Bucketů.

Zabezpečení Bucketů je řešeno pomocí řízení přístupu, práva k práci s Buckety je nutné explicitně nastavovat, a to pomocí buď již zastaralých nedoporučovaných access control listů nebo politik řízení přístupu k Bucketům. Řízení přístupu k Bucketu a objektům uvnitř Bucketu může upravovat pouze vlastník Bucketu, maximální délka seznamu politik, psaných v jazyce založených na JSONu, Bucketu je 20 KB.

Data jsou do Bucketů ukládána ve formě objektů. Amazon S3 u Bucketů a objektů podporuje verzování, takže u každého objektu v Bucketu je možné se vrátit k jeho starší verzi anebo je možné zobrazit i starší verze objektů. Objekty se skládají ze samotných ukládaných dat a jejich příslušných metadat. Přístup k jednotlivým objektům uvnitř Bucketů je řízen pomocí názvu Bucketu, klíče a případně i čísla verze, pokud uživatel nechce přistupovat k nejnovější verzi.

Konzistence objektů v Bucketech je zaručena tím, že operace na zápis, aktualizaci a mazání dat jsou prováděny atomicky. Typ konzistence objektů v Bucketech je eventuální konzistence, to znamená, že v případě provedení změn se změny nepropíšou okamžitě do všech replikací v rámci datových center, ale propíšou se postupně. Dostupnost objektů je zaručena pomocí replikace objektů mezi více datových center Amazonu.

Přístup k Amazon S3 je možný pomocí 4 způsobů, a to jsou [14]:

- Konzole pro správu
- CLI
- SDK
- REST API

V této práci je využit přístup pomocí Java SDK.

Cenová politika Amazon S3 je, že se platí pouze využitá kapacita a dotazy, není nutné platit za fixní kapacitu a tu poté dodržovat. [14]

2.9. Mailgun API

Mailgun poskytuje různé služby na posílání emailů, mezi které patří i Mailgun API. Jedná se o RESTful API, které, ačkoliv ho je možné provolat i bez využití jakýchkoliv jiných technologií, je navrženo pro použití ve vyšších programovacích jazycích a podporuje řešení v následujících technologiích: Node.js, Go, PHP, Java, Ruby. Při zpracování této diplomové práce bylo využito řešení v Jazyce Java.

API poskytuje podporu pro validaci emailů, posílání předdefinovaných šablon emailů, pro hromadné posílání emailů, přijímání emailů a sběr dat pro tvorbu analytik a reportů, trasování emailů, pravidel pro posílání emailů a routing emailů.

Bezpečnost Mailgun API je zajištěna pomocí API klíče, který je možné na stránkách Mailgun vygenerovat a poté ho používat pro autorizaci jednotlivých požadavků.

Nastavení Mailgun a i samotného API je možné pomocí webového rozhraní, které Mailgun nabízí. [22] [15]

3. ANALÝZA TVOŘENÉ APLIKACE

Tato kapitola diplomové práce se nejprve zabývá vybranými teoretickými činnostmi prováděnými během vývoje informačních systému během analýzy a návrhu informačních systému a po teoretickém úvodu do vybrané problematiky je uvedeno její využití v rámci implementace praktické části této diplomové práce.

3.1. Analýza funkčních a nefunkčních požadavků

Jednou z klíčových oblastí návrhu systému je sběr požadavků. Jedná se o strukturovanou organizovanou metodologii, jejíž cílem je identifikovat a sesbírat požadavky na systém, požadavky je, v případě potřeby, také možné dekomponovat na dílčí části. Sběr požadavků pomáhá transformovat představy a požadavky zákazníka na systém na standardizované návrhové koncepty.

3.1.1. Význam technické disciplíny analýzy a sběru požadavků

Jedná se o důležitou fázi v projektu, pokud je správně provedena, tak vede k organizované struktuře řešení a specifikací, v nichž je jasně definovaný rozsah a cíle projektu. Pokud se nepodaří správně identifikovat požadavek nebo je identifikován nesprávně, tak může dojít k finančním škodám a škodám na zdraví.

Vznik této technické disciplíny byl podmíněn neustále se zvětšujícím rozsahem a komplexností softwarových projektů, která vedla k účasti většího množství softwarových inženýrů na projektu, a to se projevilo potřebou komunikace mezi inženýry. Tato úskalí vedla k potřebě standardizovat celý proces vývoje software, proces sběru a zpracování požadavků nevyjímaje.

Sesbírané požadavky by měly být měřitelné a ověřitelné, pokud nejsou měřitelné a jsou definované pouze kvalitativně, tak snadno vedou k prodražení a prodloužení projektu. U požadavků je také potřeba určit, zda jejich cenové a časové řešení odpovídá prováděnému projektu a problému, který tyto požadavky řeší. Pokud by například implementace ochranného mechanismu byla dle námi zvolených metrik dražší než škoda, které má zabránit, tak se takové řešení nevyplatí. [16]

Existuje mnoho zdrojů, z kterých lze sesbírat požadavky na systém, mezi ty nejčastěji používané patří [17]:

- Diskuse s uživateli systému a zainteresovanými stranami
- Sledování uživatelů, jakým způsobem vykonávají úkon, který bude modelovaný systém řešit
- Obchodní případ nebo návrh
- Koncepce provozu nebo vize projektu
- Příručky a návody
- Požadavky na vylepšení stávajícího systému
- Marketingový materiál a definice produktu
- Analýza obdobných produktů na trhu

Pro tvorbu aplikace pro správu revizí tvořené v rámci této práce byly požadavky na aplikaci získané zejména od vedoucího práce z konzultací, ze souboru se zadáním a ze zadání diplomové práce, dále pak z již existujících řešení a z obecných požadavků na webovou aplikaci.

3.1.2. Funkční požadavky aplikace

Obecnými požadavky na funkcionalitu aplikace jsou, že by se mělo jednat o aplikaci na správu revizí a revidovaných spotřebičů uchovávající soubory u revizí a spotřebičů, aplikace by měla umožňovat generovat QR kódy, ke kterým je možné přiřadit spotřebič. Aplikace by měla být schopná emailem upozornit na blížící se termíny revize. Každý revizní technik by měl být schopen nahrávat své klienty a vyplnit o nich údaje, a to nezávisle na ostatních revizních technicích. V oblasti typů uživatelů by měly existovat 3 druhy uživatelů, konkrétně se jedná o adminy, revizní techniky a zákazníky, kterých se revize týká. Uživatelé by měli mít nastavenou expiraci, po registraci by měla být 30 dní s možností uživatele okamžitě zablokovat, prodloužit platnost jejich registrace anebo je povolit natrvalo. Z předchozí uvedené vlastnosti vyplývá skutečnost, že součástí aplikace by také měla být obrazovka pro správu uživatelů, kde je možné povolovat a blokovat uživatele.

Funkčními požadavky na revizi jsou, že musí být vypracována revizním technikem pro jednoho klienta, u revize musí být možnost uvést adresu místa, kde byla revize vypracována, měla by existovat možnost nahrát GPS souřadnice adresy z RÚIAN a na základě GPS souřadnic, nezávisle na tom, zda byly vyplněny ručně nebo načteny z RÚIAN, by se zde měla zobrazovat mapa zobrazující adresu revize, k revizi musí být možné nahrát soubory revize a u revize musí existovat vazba na revidované spotřebiče. U revize by pak mělo být možné vyplnit pole týkající

se samotné revize, a to název revize, stav revize, datum revize, platnost revize, příznak fakturace, tagy revize, emaily revize, popis revize, závady revize. K revizi musí být přiřazen klient a adresa revize, u revize musí být zobrazen seznam spotřebičů. Revize musí obsahovat soubory revize, u souborů je nutné mít funkcionalitu pro omezení přístupu k souboru pouze pro revizního technika, který provedl revizi, a ne pro zákazníka, pro kterého byla revize vypracována. K revizi musí mít přístup pouze revizní technik, který revizi vytvořil a klient, pro kterého byla vytvořena. Právo k úpravě revize bude mít jen revizní technik, V aplikaci by měl být, pro potřeby hlídání termínů revizí, implementován kalendář revizí s přehledem revizí a jejich stavu, kalendář by měl poskytovat přehled revizí. Kromě kalendáře by se v systému také měl nacházet hlavní panel s kalendářem a přehledem revizí, které buď expirovaly nebo ke konci jejich expirace zbývá 30 a méně dní.

Funkčními požadavky kladenými na revidované spotřebiče jsou, že u spotřebičů by mělo být možné vyplnit informace dle excelového souboru v příloze práce. Spotřebiče by měly být svázány s QR kódem, který by mělo být možné v aplikaci generovat, pomocí skenování QR kódu by ke spotřebičům měl být přístup i pro nepřihlášené uživatele, spotřebič může upravovat jen revizní technik, který ho vytvořil. Revizní technik může generovat QR kódy pro své spotřebiče. Revizní technik, který spotřebič vytvořil, bude mít možnost vygenerovat ke spotřebičům revizní kartu spotřebiče, a to jak ve excelovském formátu dle šablony v zadání, tak i v obdobné struktuře v PDF. Do revizní karty by se, kromě informací o revidovaném zařízení, také měl doplňovat podpis technika, který daný dokument revize vytvořil. K revidovaným spotřebičům by technik, který spotřebič vytvořil, měl mít možnost nahrát jednotlivé soubory, u souborů, obdobně jako tomu je u revizí, by mělo být možné ovlivnit jejich viditelnost pro ostatní.

Funkční požadavky, které musí aplikace splňovat pro práci s klienty technika je, že údaje o klientovi může technik vyplňovat nezávisle na ostatních technicích. Údaji týkajícími se přímo klienta jsou jméno, IČO, DIČ, email, telefon, umístění spotřebičů, adresa klienta. Klienty by mělo být možné načítat z ARES na základě IČO, společně s klientem by mělo dojít k přečtení adresy a provolání RÚIAN pro zisk GPS souřadnic adresy, na základě GPS souřadnic, nezávisle na tom, zda byly zadány ručně nebo pocházejí z RÚIAN, by se zde měla zobrazovat mapa zobrazující adresu klienta.

U adres by aplikace měla ukládat adresu, město, psč, zemi, zeměpisnou šířku, zeměpisnou délku.

3.1.3. Nefunkční požadavky aplikace

Nefunkční požadavky kladené na tuto aplikaci se týkají zejména požadavků na samotný systém.

Vzhledem k tomu, že aplikace je primárně určena pro revizní techniky, tak jedním z požadovaných nefunkčních požadavků je responzivní design, který bude responsivní alespoň pro dva druhy zařízení – pro standardní mobilní telefony a pro standardní displeje počítačů s rozměry 1920 × 1080.

Vzhledem k funkčnímu požadavku mít možnost získat údaje o klientovi z ARES pomocí IČO je dalším nefunkčním požadavkem zajištění komunikace aplikace s ARES, z požadavků nutnosti načítání GPS souřadnic u adresy klienta a revize z RÚIAN také vyplývá nefunkční požadavek komunikace s RÚIAN.

Z oblasti technologií a ukládání dat by mezi vlastnostmi aplikace neměla chybět možnost kontejnerizace a nasazení aplikace pomocí Docker. Aplikace by měla být webovou aplikací napsanou v jazyce Java s použitím frameworků Spring s použitím Spring Boot a frameworku Vaadin. Vzhledem k tomu, že aplikace cílí zejména na použití revizními techniky, tak pro pohodlné používání v terénu a z domova nebo jiného pracoviště technika je nutné zajistit multiplatformnost. Vzhledem k tomu, že se jedná o webovou aplikaci, tak tento požadavek je implicitně splněn – díky tomu, že se jedná o webovou aplikaci, tak je k ní možné přistupovat pomocí jakéhokoliv zařízení s webovým prohlížečem. Další z požadavků se týká způsobu ukládání dat kromě souborů revizí a spotřebičů. Tato data by měla být uložena v relační databázi MySQL. Soubory revizí a zařízení nebudou ukládány do této databáze, ale do cloudového úložiště Amazon S3.

3.1.4. Shrnutí funkčních a nefunkčních požadavků

Výše popsané funkční požadavky jsou shrnuty v následující tabulce 2:

Tabulka 2 - funkční požadavky vyvíjené aplikace [zdroj vlastní]

Funkční požadavky	
FP01	Systém bude, kromě karty revize, dostupný pro přihlášené uživatele
FP02	Systém bude umožňovat registraci
FP03	Systém bude umožňovat autentizaci a autorizaci

FP04	Systém bude umožňovat ukládat, měnit a upravovat revize pro vybraného klienta, a to název revize, stav revize, datum revize, platnost revize, příznak fakturace, tagy revize, emaily revize, popis revize, závady revize, klienta revize, adresu revize
FP05	Systém bude uchovávat soubory revizí
FP06	Systém bude umožňovat změnu viditelnosti souboru revizí jen pro revizního technika, který revizi vytvořil
FP07	Systém bude umožňovat ukládat, měnit a upravovat spotřebiče
FP08	Systém bude uchovávat soubory spotřebičů
FP09	Systém bude umožňovat změnu viditelnosti souboru revize spotřebiče jen pro revizního technika, který spotřebič vytvořil
FP10	Systém bude emailem upozorňovat uživatele na blížící se konec revize
FP11	Systém bude umožňovat reviznímu technikovi nahrát své klienty a uchovávat o nich tyto informace: jméno, IČO, DIČ, email, telefon, umístění spotřebičů, adresa klienta
FP12	Systém bude poskytovat alespoň 3 druhy uživatelů – adminy, revizní techniky a klienty
FP13	Systém bude evidovat expiraci uživatelů
FP14	Systém bude nastavovat defaultní expiraci uživatelů na 30 dní
FP15	Systém bude umožňovat dočasné povolení, trvalé povolení a blokaci uživatelů
FP16	Systém bude umožňovat ukládat adresy revizí a klientů
FP17	Systém bude umožňovat načíst adresy revizí a klientů z ARES
FP18	Systém bude ukládat GPS souřadnice adres klientů a revizí
FP19	Systém bude umožňovat načíst GPS souřadnice adres klientů a revizí z RÚIAN
FP20	Systém bude omezovat přístup k revizi jen pro revizního technika, který ji založil a pro klienta, kterého se revize týká
FP21	Systém bude umožňovat upravit revizi jen reviznímu technikovi, který ji vytvořil

FP22	Systém bude umožňovat přístup ke spotřebičům přes QR kód
FP23	Systém bude umožňovat veřejný přístup ke spotřebičům
FP24	Systém bude umožňovat upravovat spotřebič pouze reviznímu technikovi, který ho vytvořil
FP25	Systém bude umožňovat reviznímu technikovi, který vytvořil spotřebič, vytvořit revizní kartu spotřebiče ve formátu PDF a excel, v kartě se nachází podpis technika
FP26	Systém bude umožňovat revizním technikům generovat QR kódy
FP27	Systém bude zobrazovat mapy adres klientů a revizí
FP28	Systém bude umožňovat načíst klienty z ARES na základě IČO
FP29	Systém bude určovat datum následující revize dle data provedení revize s přičtením konstantní hodnoty 3 měsíců, 6 měsíců, 1 rok, 2 roky, 3 roky. 4 roky, 5 let anebo neomezeně
FP30	Systém bude ukládat u adres adresu, město, psč, zemi, zeměpisnou šířku, zeměpisnou délku

Nefunkční požadavky popsané v předchozí podkapitole jsou shrnuty v následující tabulce 3:

Tabulka 3 - nefunkční požadavky vyvíjené aplikace [zdroj vlastní]

Nefunkční požadavky	
NP01	Systém bude mít responzivní design pro mobilní telefony a displeje 1920×1080
NP02	Systém bude využívat ARES pro načítání údajů o klientech a adres revizí
NP03	Systém bude využívat RÚIAN pro získání GPS souřadnic adres
NP04	Systém bude umožňovat kontejnerizaci a nasazení aplikace pomocí Dockeru
NP05	Systém bude realizován pomocí webové aplikace napsané v Javě, s využitím frameworků Spring, varianta Spring Boot, a Vaadin
NP06	Systém bude multiplatformní – aplikace dostupná přes webový prohlížeč
NP07	Systém bude používat pro ukládání dat relační databázi MySQL

NP08	Systém bude využívat pro ukládání souborů revizí a spotřebičů cloudového úložiště Amazon S3

3.2. Business procesy

3.2.1. Definice business procesů

Jedná se o definované procesy a aktivity, které se vykonávají v aplikaci, které slouží k vykonání určitého požadavku na systém nebo tvoří oporu k jeho vykonání. Procesy se řadí do tří skupin, které se liší využitím, a to jsou klíčové procesy zajišťující chod a činnost systému, podpůrné procesy podporující a umožňující správný průběh a funkci klíčových procesů a řídící procesy sloužící k orchestraci, plánování a sledování systému, stavů systému a k řízení ostatních business procesů. Business procesy se skládají z mnoha činností, systémů a uživatelů a jejich správné kompozice k zajištění správného chodu systému a vykonání požadovaných funkcionalit. Tyto činnosti mohou být na pozadí a vykonávat podpůrné akce zajišťující správný chod systému anebo se může jednat o klíčové procesy umožňující ovládání a používání systému uživateli pomocí uživatelského rozhraní.

Vzhledem k tomu, že business procesy zajišťují chod systému dle definovaných požadavků a dle vymezených cílů systému, tak jejich správná definice a vymezení jejich rozsahu je důležitou součástí návrhu informačního systému. V případě, že business procesy byly definovány správně, tak umožňují rychlé a efektivní reagování na změny na trhu čímž poskytují konkurenční výhodu při konkurenčním boji, zkvalitňují uživatelský komfort uživatelů při používání systému a díky tomu vedou k vyšší spokojenosti uživatelů se systémem, k retenci těchto uživatelů a zvyšují pravděpodobnost budoucího používání systému nebo dalších produktů od provozovatele tohoto systému, umožňují škálování systému v případě růstu velikosti systému, vedou k snadnější implementaci a přizpůsobení se novým technologiím a zjednodušují návrh inovativních řešení, a zvyšují kvalitu výsledného systému. V případě správné definice umožňují odhalit redundantní a neefektivní kroky, které prodražují systém a jeho provoz, a slouží jako opora k jejich nápravě. Správné vymezení business procesů zajišťuje, že systém bude vyhovovat regulacím, vyhláškám a směrnicím, kromě vyhovění těmto právním dokumentům také slouží k redukci porušování regulací vyplývajících z těchto dokumentů a tím dochází ke snížení operačních rizik. Pokud se jedná o procesy, s kterými přijdou do styku zaměstnanci používající informační systém, tak jim tyto procesy pomáhají zvyšovat jejich produktivitu a definují rozsah jejich očekávané činnosti a odpovědnosti. V neposlední řadě správná definice těchto procesů zjednodušuje práci s daty a usnadňuje jejich využívání. [19]

3.2.2. Hlavní klíčové business procesy v aplikaci

Hlavní klíčové procesy v aplikaci se věnují tvorbě a úpravě revizí včetně nahrávání souborů k revizím, sledování časové validity revizí a upozornění procházející a prošlé revize, evidence a přidávání klientů pro vybraného revizního technika, správě a evidenci revidovaných spotřebičů včetně správy a generování jejich QR kódů.

1. **Tvorba revize**

- Přihlášený revizní technik přejde na sekci „Správa revizí“.
- Použije tlačítko „Přidat novou revizi“, tím se přesune do formuláře pro přidání nové revize.
- Technik vyplní informací o revizi
- Technik vybere klienta z databáze nebo vytvoří nového klienta (možnost načtení z ARES podle IČO).
- Technik vyplní adresu revize, pokud nebyla vyplněna automaticky z adresy klienta.
- Technik nahraje soubory revize.
- Technik zkontroluje vyplněné údaje a pomocí tlačítka revizi uloží.

Obdobným způsobem funguje i scénář je i pro úpravu revize, z toho důvodu zde není uveden.

2. **Přidání klienta**

- Přihlášený revizní technik přejde do sekce „Klienti“.
- Kliknutím na tlačítko „Přidat nového klienta“ se přesune do formuláře pro přidání nového klienta.
- Zde buď vyplní IČO a pokusí se nahrát údaje o klientovi z ARES anebo, v případě, že technik nechce nahrávat údaje klienta z ARES nebo klient nebyl nalezen v ARES, tak vyplní údaje o klientovi ručně.
- Technik vyplní umístění spotřebičů v rámci klienta.
- Technik vyplní adresu klienta, pokud nebyla získána z ARES.
- Revizní technik zkontroluje údaje o klientovi a klienta uloží.

Obdobným způsobem funguje i scénář je i pro úpravu klienta, z toho důvodu zde není uveden.

3. **Notifikace na prošlou revizi**

- Systém každý den zkontroluje, zda se některým revizím blíží datum jejich expirace dle nastaveného intervalu připomínek u daného revizního technika.

- Pokud takové případy existují, tak na emailovou adresu revizního technika a emailové adresy uvedené u revize pošle email s upomínkou na blížící se termín revize.

4. **Vytvoření revize spotřebiče**

- Technik buď na stránce „Generování QR kódů“ vytvoří nový QR kód, nebo zde vybere stávající QR kód, nebo oskenuje stávající QR kód, ke kterému není připojen spotřebič
- Technik je poté přesměrován na detail QR kódu, kde vyplní údaje o revizi zařízení
- Kromě údajů týkajících se samotného spotřebiče vybere klienta a revizi, v níž došlo k revizi spotřebiče
- Poté může vygenerovat dokument revize, případně i PDF s QR kódem spotřebiče
- K revizi zařízení může nahrát soubory
- Poté zkontroluje vyplněné údaje a nahrané soubory a po kontrole a korekci údajů uloží novou revizi spotřebiče

Obdobným způsobem probíhá proces úpravy údajů o spotřebiči, který se liší v prvním kroku – zde technik buď na stránce „Spotřebiče“ vybere klienta a spotřebič a nebo oskenuje QR kód již revidovaného spotřebiče.

5. **Registrace uživatele (technika)**

- Nepřihlášený technik přejde na registrační formulář
- Vyplní uživatelské jméno, heslo, email, křestní jméno a příjmení
- Zkontroluje vyplněné údaje
- Pomocí tlačítka „Registrovat“ dokončí registraci

3.3. Návrh UI aplikace

3.3.1. Postup při návrhu UI

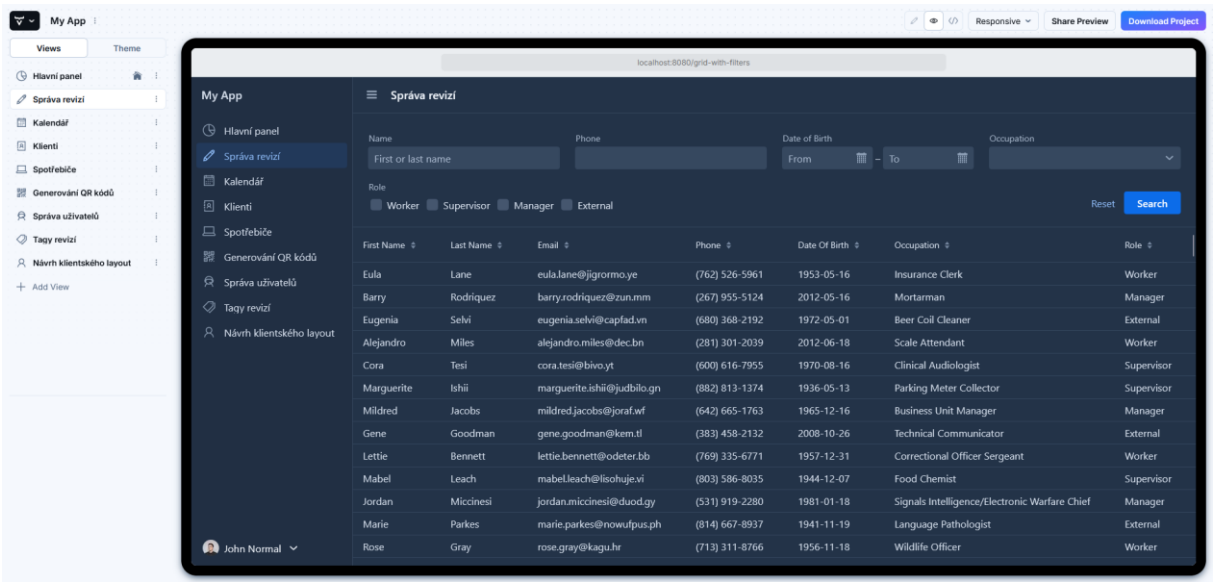
Návrh UI aplikace probíhá na základě požadavků na základě požadavků na systém, a to se zohledněním potřeb veškerých aktérů systému včetně systému samotného. Při samotném designu návrhu systému se zpravidla zohledňují uživatelé systému, jejich přání, požadavky a zpětná vazba, takovému principu se říká „user-centric“. Prvky UI systému by měly být navrhovány se zohledněním cílové skupiny uživatelů, a to například se zohledněním jejich věku, pohlaví, vzdělání, vykonávané profese. Ovládání systému by mělo být intuitivní, a to takovým způsobem, aby nebylo nutné uživatele do systému zaškolovat nebo aby k běžnému provozu systému nebylo nutné používat uživatelské příručky a manuály. Při pojmenovávání stránek, ovládacích prvků a procesů v aplikaci by se designér měl snažit o co nejvýstižnější názvy, které uživatelům napoví, za kterou činnost je daný prvek nebo oblast systému

odpovědný. Design systému je iterativní proces, kde na základě analýzy již navrhnutého UI a zpětné vazby k UI je možné dle přání uživatelů a požadavků na systém UI upravit. [20]

3.3.2. Návrh UI v aplikaci

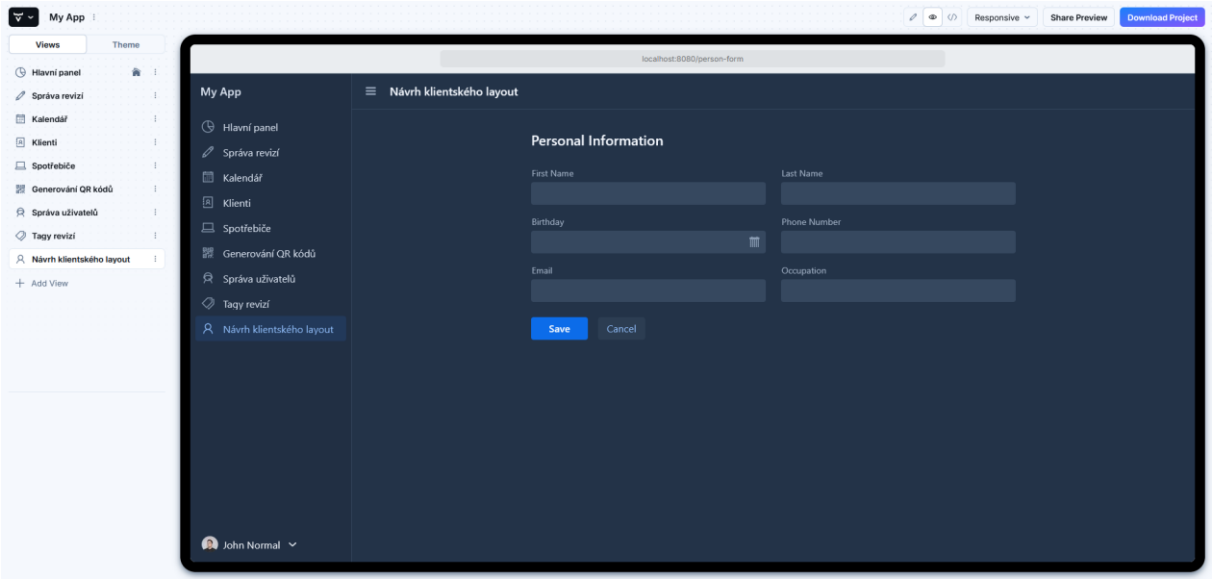
Po sesbírání požadavků a tvorby business procesů byl vytvořen základní návrh UI. Základní návrh UI systému byl vytvořen pomocí Vaadin Start, což je webový nástroj Vaadinu k vytvoření základní kostry projektu. Pomocí webového rozhraní bylo vytvořeno rozvržení aplikace včetně seznamu veškerých stránek aplikace, které budou dostupné z hlavní lišty programu. Vzhledem k absenci pokročilé konfigurace zejména datových zdrojů a k nutné úpravě vygenerovaného kódu byl v této webové aplikaci byl proveden pouze návrh předběžného layoutu jak aplikace, tak i podoby formulářů a tento layout byl vyexportován pro další zpracování. K samotnému rozvržení polí entit v aplikaci a přizpůsobení responzivity bylo přikročeno až během implementace funkcionalit a UI aplikace.

Níže, na obrázku 5, je ukázka layoutu aplikace a také rozvržení filtračního gridu. V layoutu aplikace byl dočasně pro vizualizaci přidán i návrh klientského layoutu, který byl před exportem projektu do zdrojového kódu odebrán.



Obrázek 5 - návrh UI systému [zdroj vlastní]

Pro úvodní návrh rozvržení UI formulářových prvků aplikace bylo využito rozvržení formulářových prvků, které Vaadin nabízí již během generování projektu, layout formulářových prvků je vidět na následujícím obrázku 6:



Obrázek 6 - návrh layoutu formuláře [zdroj vlastní]

Pro návrh hlavního panelu byl vybrán dashboard, ze kterého ve výsledné aplikaci bylo využito pouze horní rozvržení s počtem hodnot, graf v aplikaci využit nebyl. Layout je vidět níže na obrázku 7:

Obrázek 7 - návrh layoutu hlavního panelu [zdroj vlastní]

4. DATOVÝ MODEL

4.1. Návrh datového modelu

Návrh datového modelu je jedním z prvních kroků transformace sesbíraných požadavků do prvků vytvářené aplikace. Jedná se o komplexní proces transformace sesbíraných požadavků na koncept organizace, propojení a způsob uložení dat, z hoho vyplývá, že datový model představuje abstraktní pohled na interakci mezi daty a strukturu dat. Na rozdíl od databázového schématu, které se typicky zabývá pouze daty z jednoho zdroje, se schéma zabývá pohledem na celý modelovaný systém.

Datový model představuje přehledný abstraktní pohled na modelovaný informační systém z pohledu dat. Tento model by měl obsahovat strukturu dat a popsat způsob a metody jejich zpracování. Pokud je model správně navržený, tak usnadňuje a urychluje tvorbu zabezpečení, vývin funkcionalit, škálovatelnost, použitelnost systému a usnadňuje interkomunikaci systému s dalšími systémy. [18]

4.2. Tvorba datového modelu

Tvorba datového modelu se skládá z několika kroků, dle [18] se jedná o následujících 9 kroků:

1. Sběr business požadavků – zdrojem těchto požadavků jsou zainteresované strany, uživatelé systému, klienti, odborníci z oboru, business požadavky představují funkcionalitu aplikace
2. Definice business procesů – jedná se o funkční logiku systému, o probíhající procesy, a to jak systémové vzniklé z událostí, tak i těch zahájených uživateli. Je zde definován pouze obecný průběh procesů, a ne implementační detaily
3. Tvorba konceptuálního modelu – v tomto kroku se tvoří konceptuální model, kde jsou zastoupeny nejdůležitější entity aplikace a vazby mezi těmito entitami, model je tvořen z pohledu businessu
4. Tvorba entit a atributů – jedná se o transformaci konceptuálního modelu na databázový model a entity aplikace
5. Identifikace datových zdrojů – tato část procesu se zabývá identifikací zdroje dat, z kterých bude aplikace přebírat data, může se jednat o vnitřní databáze, vnější databáze, API a webové služby, soubory atd., na základě datových zdrojů je možné učinit rozhodnutí vytvořit tabulky v modelu tvořené aplikace
6. Tvorba vazeb mezi entitami – na základě analýzy modelu je nutné určit typy vazeb, typy vazeb jsou nejčastěji jedna-ku-jedné, mnoho-ku-jedné, mnoho-ku-mnoho.

7. Tvorba fyzického modelu – s použitím výstupů předchozích kroků se vytvoří interní a externí databázový model, to znamená tvorba tabulek a atributů vnitřních DB a volba způsobu připojeni a realizace připojení k externím DB
8. Normalizace dat a zajištění jejich integrity – atomizace, omezení redundance dat a zajištění referenční integrity v modelu
9. Údržba datového modelu – po implementaci datového modelu je nutné při dalším vývoji brát ohled na stávající struktury a vazby mezi nimi

4.3. Datový model použitý v aplikaci

Datový model použitý v této diplomové práci vznikl na základě analýzy požadavků a business procesů s ohledem na použité externí služby, s kterými tato aplikace komunikuje.

Níže následuje výčet datových entit aplikace, které byly v této práci použity:

4.3.1. Abstraktní historizovaná entita

Nejedná se o standardní entitu, ale o předka všech entit, u kterých je nutné uchovávat tvůrce a čas vytvoření, tato entita bude sloužit jako pomocná. Vzhledem k tomu, že se jedná o důležitou vlastnost systému ovlivňující strukturu velkého množství entit, tak je vhodné ji uvést ve výčtu použitých tříd a výčtů modelovaného informačního systému.

4.3.2. Adresa

Jedná se o entitu, která představuje adresu, a jako taková musí ukládat informace o adrese, městě, zemi, poštovní směrovací číslo, zeměpisnou šířku a délku adresy. Také je zde, dle typu adresy, ukládán buď zákazník anebo revize. U adresy je nutné ukládat informace o technikovi, který ji přidal, a čas jejího přidání, jedná se o historizovanou entitu.

4.3.3. Spotřebič

Spotřebič představuje zařízení, pro které byla provedena revize. Jedná se o historizovanou entitu, která umožňuje vyplnění informací o názvu spotřebiče, výrobci, roku výroby, umístění spotřebiče v rámci klienta, sériové číslo, typové označení, skupinu používání, jmenovité napětí, jmenovitý proud, jmenovitý výkon, třídě ochrany, datu revize, typu prohlídky, zjištěných závadách, výsledku prohlídky, sestavě zařízení, délce síťového přívodu, odporu ochrany vodiče u spotřebiče, izolační odpor, metodě měření, naměřeném proudu, zkoušce chodu, celkovém vyhodnocení kontroly, termínu další revize, jménu a příjmení technika a obsahuje podpis technika. Kromě vlastních atributů zde musí existovat vazba na klienta, revizi a QR kód.

4.3.4. Klient

Tato entita představuje zákazníka, pro kterého je prováděna revize. U této entity je nutné udržovat historizaci. Entita klienta obsahuje název, email, telefonní číslo, IČO, DIČ, seznam umístění spotřebičů klienta. Klient také obsahuje informaci o své adrese a je nutné u něj evidovat seznam jeho revizí a spotřebičů.

4.3.5. Umístění spotřebičů

Tato entita slouží k reprezentací umístění spotřebičů u klienta, jejími atributy jsou název umístění. U umístění spotřebiče musí být možné zjistit seznam spotřebičů, které se na něm nacházejí, a klienta, ke kterému patří.

4.3.6. Soubor

Jedná se o entitu představující buď soubor revize nebo soubor spotřebiče, z toho důvodu zde musí být evidována informace o vazbě na tyto entity. Tuto entitu je nutné historizovat. Atributy této entity jsou název souboru, cesta k souboru na externím zařízení, zda je nebo není přístupný pro klienta, typ souboru a velikost souboru. Data souboru zde není nutné uchovávat, ta jsou ukládána v cloudovém úložišti Amazon S3.

4.3.7. QR kód

Jedná se o entitu reprezentující QR kód. U této entity stačí evidovat vazbu na spotřebič, pokud existuje, tak se jedná o použitý QR kód, jinak se jedná o volný QR kód.

4.3.8. Revize

Tato entita je stěžejní historizovanou entitou aplikace, reprezentuje jednu instanci provedené revize. U revize je nezbytné udržovat přehled o souborech revize, klientovi, pro kterého byla provedena, seznamu spotřebičů, které se v rámci revize revidovaly, a adrese, na které revize proběhla a seznam tagů revize, které jsou k revizi přiřazeny. Atributy přímo patřící revizi jsou jméno, status revize, datum provedení revize, příznak, zda byla fakturována, popis revize, seznam kontaktních emailů revize, popis zjištěných závad revize, platnost revize.

4.3.9. Tag revize

Jedná se o pomocnou entitu představující tag revize, která obsahuje název tagu a vazbu na revize, na kterých byl tag použit. Tag je historizovaná entita.

4.3.10. Role

Tato entita přiřazuje oprávnění uživateli. V aplikaci se nachází role user, admin, client, technician. Role je historizovanou entitou obsahující informaci o názvu role a uživatelích, ke kterým byla přiřazena.

4.3.11. Uživatel

Jedná se o entitu představující uživatele aplikace, kterým je klient, technik nebo administrátor. Atributy, které se se musí na této entitě nacházet, jsou uživatelské jméno, podpis uživatele, název souboru s podpisem uživatele, email, křestní jméno, příjmení, heslo uživatele uložené v bezpečné formě v podobě hashe se solí, validita uživatelského účtu, čas upomínky revize před expirací a vlastní text upomínky revize. Pro potřeby evidence klientských účtů se zde pak dále musí nacházet informace o IČO, pro potřeby resetování hesla pak token pro reset hesla. Dále by u uživatele mělo být možné dohledat seznam jeho klientů, revizí, spotřebičů, rolí a QR kódů.

Zde se nachází seznam použitých výčtových typů v aplikaci, které jsou použity pro uchovávání dat:

4.3.12. Country

Vzhledem k relativní neměnnosti počtu států světa není nutné informace o zemích uchovávat v DB, ale ukládají se v číselníku. V této aplikaci je použit číselník zemí dle standardu ISO 3166-1.

4.3.13. Sestava zařízení

Počet sestav zařízení je pevně omezen na následující 4 hodnoty:

- S - přístroj měřen samostatně (nevyplňuje se délka přívodu),
- P - přístroj s pevně připojitelným síťovým přívodem
- O - přístroj s odpojitelným síťovým přívodem
- PP - pohyblivý přívod

4.3.14. Formát dokumentu

Vzhledem k tomu, že aplikace umožňuje generovat dokumenty ve formátu PDF a Excel (xlsx), tak tyto pro tyto hodnoty je vhodné použít číselník.

4.3.15. Metoda měření

Pro metodu měření, podobně jako pro sestavu zařízení, byl použit číselník se 4 následujícími hodnotami reprezentující metodu měření:

- V - měření proudu protékající ochranným vodičem
- D - měření dotykového proudu
- U - měření unikajícího proudu
- R - měření rozdílového proudu

4.3.16. Stav revize

Z analýzy požadavků na správu revizí bylo zjištěno, že revize se může nacházet ve stavech založená, rozpracovaná nebo hotová.

4.3.17. Platnost revize

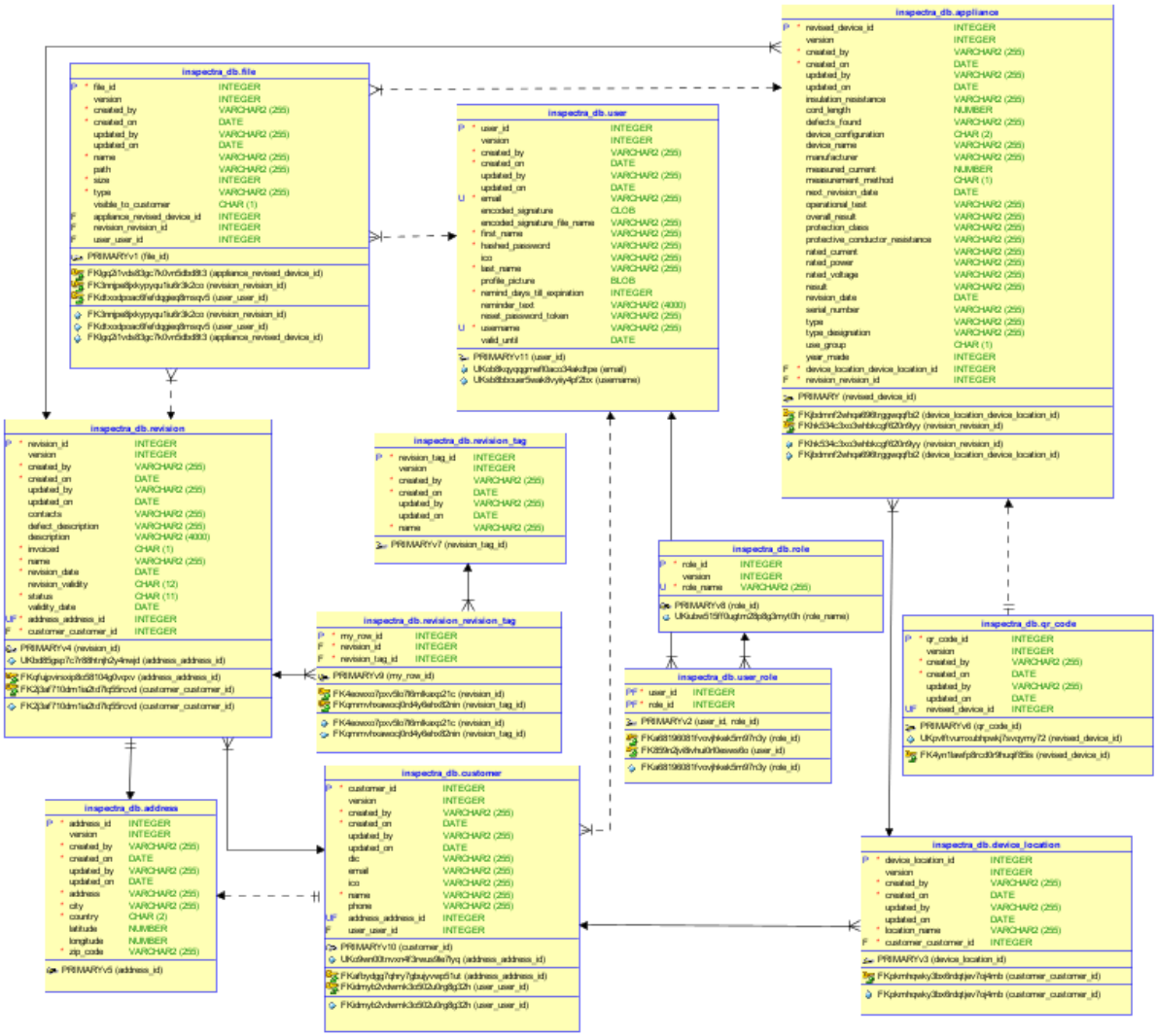
Z definice požadavků vyplývá, že platnost revize je neměnný číselník, který nabývá hodnot 3 měsíce, 6 měsíců, 1 rok, 2 roky. 3 roky, 4 roky, 5 let a neomezeně.

4.3.18. Třída použití

V aplikaci jsou použity hodnoty A, B, C, D, E reprezentující třídy použití spotřebiče.

4.4. Databázový model v aplikaci

Transformací logického datového modelu na relační databázový model bylo vytvořeno následující schéma databáze, viditelné na obrázku 8, které je v této diplomové práci použito:

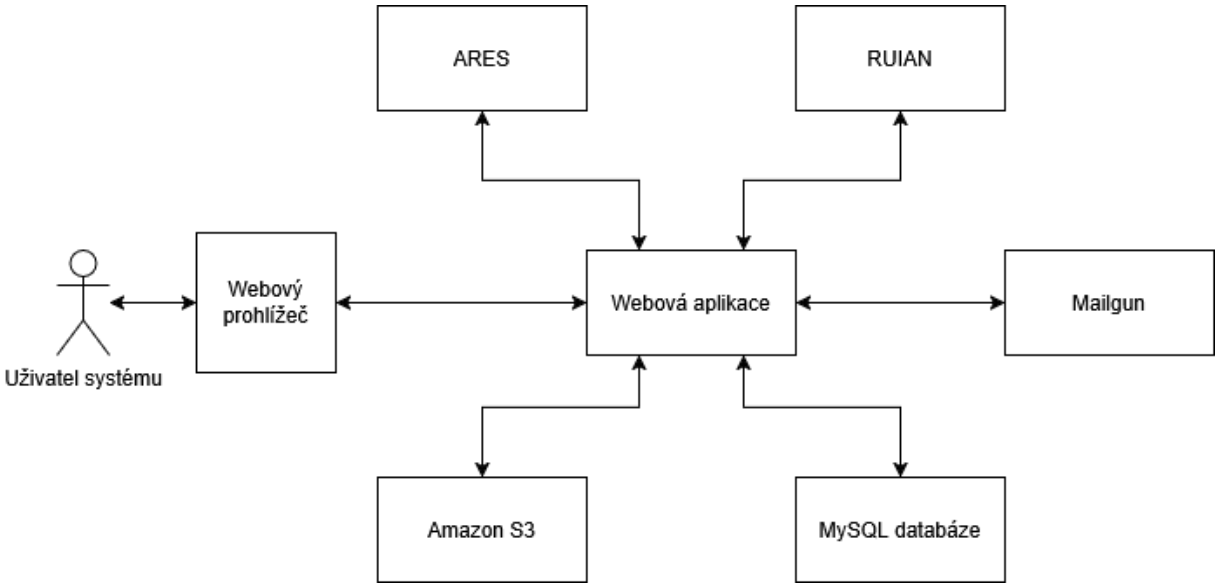


Obrázek 8 - schéma DB aplikace [zdroj vlastní]

4.5. Schéma informačního systému implementované aplikace

Na základě sesbíraných požadavků, navrhnutých business procesů v aplikaci, vytvořeného datového a databázového modelu je možné vyvodit schéma informačního systému. Komunikace uživatele se systémem je zajištěna pomocí webového prohlížeče, kde přes UI posílá příkazy webové aplikaci. Většina dat aplikace, kromě souborů revizí a spotřebičů, je uložena v databázi, která podle dotazu aplikace vrátí příslušná data. Dále pak aplikace pro načítání dat o klientech a jejich adresách používá volání ARES pro data klientů a jejich adres a RÚIAN pro získání GPS souřadnic adres. Pokud uživatel po aplikaci požaduje načtení GPS souřadnic adresy z RÚIAN, tak komunikace aplikace a RÚIAN probíhá i v tomto případě.

Volání Mailgun API aplikace provádí v případě, že by mělo dojít k zaslání emailu, a to například při blížícím se termínu revize, jelikož API vrací odpověď, zda se podařilo zaslat email, tak komunikace je obousměrná. Pro správu a ukládání souborů revizí a spotřebičů, které uživatel nahraje, se využívá volání API Amazon S3, které jako odpověď vrací, zda se požadavek podařilo úspěšně vykonat. Výše popsaná komunikace mezi jednotlivými komponentami je zobrazena na následujícím obrázku 9:



Obrázek 9 - architektura aplikace [zdroj vlastní]

5. IMPLEMENTACE

Cílem této kapitoly je seznámit čtenáře s procesem vývoje aplikace, zejména popsat strukturu zdrojového kódu a projektu, jeho rozdělení do balíčků a tříd, zdůraznit klíčové třídy a přiblížit finální podobu vyvinuté aplikace.

Pro zachování konzistentního designu byla aplikace inspirována předchozí prací pana Michaela Léva. Z této práce byly převzaty některé komponenty, zejména dlaždice pro responzivní zobrazení gridu revizí a také vybrané CSS styly a vzhled stavu revize. [30]

5.1. Struktura projektu

Struktura projektu a rozdělení aplikace do balíčků do značné míry vychází z využití startovacího projektu dostupného na oficiálních stránkách Vaadin. Pro vývoj aplikace byl zvolen startovací projekt ve Vaadin Flow, který je postaven na technologii Spring Boot, díky čemuž výsledný vygenerovaný projekt svojí strukturou odpovídá klasické Spring Boot aplikaci s částmi specifickými pro Vaadin Flow, mezi které patří složka frontend sloužící pro uložení jak uživatelsky vytvořených, tak i Vaadinem vygenerovaných CSS, HTML a Javascript souborů potřebných pro běh aplikace.

Výsledná struktura projektu, očištěná od souborů generovaných vývojovým prostředím anebo přidaných do projektu za účelem verzování, je následující:

- Src
    - main
        - bundles – adresář pro Vaadinem předkompilované soubory
        - frontend – adresář pro jak ručně vytvořené, tak i Vaadinem vygenerované soubory CSS, HTML a Javascript
        - java
            - com.example.application
                - components – v této složce jsou uloženy formuláře, komponenty nacházející se na pohledech a další komponenty pro validaci, výpis chyb a internacionalizaci
                - database – složka obsahující třídu zajišťující úvodní naplnění databáze
                - dataholders – složka obsahující přepravky dat
                - enitites – složka obsahující třídy a rozhraní mapované na databázové entity aplikace

        - enums – složka obsahující vlastní výčtové typy
        - exceptions – složka pro definici vlastních vyjímek
        - repositories – složka s rozhraními repozitářů použitých v aplikaci
        - security – v této složce se nachází třídy s konfigurací bezpečnosti a kontroly práv v aplikaci
        - services – jedná se o složku se servisními třídami
        - staticConstants – tato složka obsahuje konstanty použité v aplikaci
        - views – složka pro uložení pohledů aplikace, které jsou viditelné pro uživatele dle jejich rolí
        - Application – spustitelná třída aplikace
    - resources – soubory zdrojů, jako je například vlastní font, šablona dokumentu pro generování souborů revizí, Spring konfigurace aplikace
- Dockerfile – jednoduchý Dockerfile s instrukcemi k sestavení Docker kontejneru
- docker-compose.yml – soubor ke spuštění projektu pomocí docker compose up s konfigurací dle .env souboru
- pom.xml – Maven konfigurace projektu, obsahuje závislosti
- .env – soubor s defaultní konfigurací pro spuštění projektu, obsahuje proměnnou QR_CODE_BASE_URL pro definici základní URL adresy generovaných QR kódů a VAADIN_OFFLINE_KEY pro předání potřebného Vaadin Pro klíče, který je potřeba pro spuštění aplikace

5.2. Pohledy

Jedná se o view třídy aplikace, které řídí zobrazení správného obsahu okna v prohlížeči. Tyto třídy nejenže zajišťují správné zobrazení obsahu, ale také ve spolupráci se Spring Security omezují přístup k jednotlivým oknům aplikace pomocí Vaadin anotací. Omezení přístupu se řídí uživatelskou rolí nebo tím, zda je uživatel vůbec přihlášen. Pohledy zároveň umožňují zpracování navigačních událostí a předávání parametrů z navigačního řádku. V aplikaci je jakákoliv viditelná obrazovka pohled, a to buď v aplikaci třídy pojmenované jako ObrazovkaDetailView s formulářem pro úpravu údajů, anebo pohled v aplikaci pojmenované třídy view pro navigaci.

Použité pohledy v sobě obsahují další komponenty, které jsou zodpovědné za zobrazení obsahu, v této aplikaci se jedná zejména o filtrační menu, formuláře a gridy. Všechny pohledy aplikace,

kromě pohledů pro registraci, přihlášení a obnovu hesla, dědí z třídy MainLayout. Použití MainLayout zajišťuje správné rozvržení a základní responzivní zobrazení aplikace, přítomnost navigačního menu v aplikaci a zobrazení obsahu do svého vnitřního okna. Kromě snížení duplicity kódu přináší použití layoutu také výkonnostní výhodu – prohlížeč nemusí znovu vykreslovat jeho statickou část, což urychluje načítání stránky a šetří přenesená data. MainLayout je do aplikace přidáno použitím anotace @Route s parametrem layout = MainLayout.class. Následující řádek ukazuje příklad použití anotace u pohledu spotřebičů:

```
@Route(value = "appliances", layout = MainLayout.class)
```

V implementované aplikaci byly implementovány následující navigační pohledy pro položky menu:

- Hlavní panel – tento pohled zobrazuje revize dle stavu, kalendář s naplánovanými revizemi a prošlé revize nebo revize procházející v následujících 30 dnech, tento pohled je viditelný pro přihlášené uživatele
- Správa revizí – jedná se o pohled na revize, pohled obsahuje filtrační menu s možností založení nové revize, grid s revizemi a tlačítko „Hromadné akce“ pro provádění hromadných akcí s vybranými revizemi v gridu, tento pohled je viditelný pro přihlášené uživatele
- Kalendář – pohled obsahuje kalendář s revizemi a seznam revizí ve vybraném měsíci, tento pohled je viditelný pro přihlášené uživatele
- Klienti – pohled s filtrem klientů revizního technika s možností přidání nového klienta a grid se seznamem klientů, tento pohled je viditelný pro revizní techniky a adminy
- Spotřebiče – pohled s filtrem spotřebičů a gridem se spotřebiči, tento pohled je viditelný pro revizní techniky a adminy
- Generování QR kódů – pohled s možností generování nových QR kódů, generování dokumentu s QR kódy z gridu se seznamem QR kódů, které ještě nebyly přiděleny k žádnému spotřebiči, tento pohled je viditelný pro revizní techniky a adminy
- Správa uživatelů – pohled s filtrem uživatelů a gridem se seznamem uživatelů v aplikaci, tento pohled je viditelný pro adminy
- Tagy revizí – pohled s možností přidání nového tagu a s gridem se seznamem vytvořených tagů revizí, tento pohled je viditelný pro revizní techniky a adminy

Navigační pohledy byly implementovány následujícím způsobem, viditelném na příkladu zdrojového kódu 1:

```
@PageTitle("Klienti")
@Route(value = "clients", layout = MainLayout.class)
@Menu(order = 4, icon = LineAwesomeIconUrl.ADDRESS_BOOK)
@RolesAllowed({ADMIN, TECHNICIAN})
@Uses(Icon.class)
public class CustomerView extends Div {

    private Grid<Customer> customerGrid;
    private CustomerUpperMenu customerUpperMenu;

    private final CustomerService customerService;
    private final UserService userService;
    private User user;

    private Registration widthListener;
```

Zdrojový kód 1 - pohled CustomerView [zdroj vlastní]

Z výše uvedené ukázky zdrojového kódu může čtenář vyčíst následující informace:

- Anotace @PageTitle slouží nastavení názvu stránky v panelu prohlížeče.
- Anotace @Route mapuje cestu clients, pohled je obalen pomocí layoutu MainLayout
- Anotací @Menu je pohled zařazen na 4 pozici s ikonkou adresáře představující klienty
- Anotace @RolesAllowed omezuje přistup pro adminy a revizní techniky, dalšími použitými anotacemi v aplikaci pro omezení přístupů jsou @AnonymousAllowed pro povolení přístupu pro přihlášené i nepřihlášené uživatele a @PermitAll pro povolení přístupu pro všechny přihlášené uživatele
- Anotace @Uses(Icon.class) zajišťuje správné načtení Vaadin ikonek
- Položky v pohledu jsou realizovány pomocí komponent (zde customerGrid a customerUpperMenu)

Pohledy detailView budou uvedeny v podkapitole zabývající se formuláři.

5.3. Filtrační komponenty

Filtrační komponenty umožňují uživatelům aplikace provádět dynamickou filtraci na základě vybraných filtrů. GUI filtrů je implementováno pomocí formulářových prvků, které do backendové části ukládají dle vyplněných nebo vybraných hodnot v GUI jejich logickou reprezentaci, která se pak použije při samotném filtrování. Filtrování pomocí filtrů je v aplikaci realizováno na backendu. Filtry jsou v implementované aplikaci v pohledech vždy umístěny v horní části stránky.

Jak je vidět na ukázce zdrojového kódu 2, filtry jsou implementovány následovně:

```
@CssImport(value = "./themes/sprava-revizi/components/wrap-
button.css", themeFor ="vaadin-button")
public class ApplianceUpperMenu extends VerticalLayout
implements BeforeEnterObserver {
    private ComboBox<Customer> customerComboBox;
    private ComboBox<DeviceLocation> deviceLocationComboBox;
    private FormLayout formLayout;

    private final UserService userService;
    private final CustomerService customerService;
    private final ApplianceService applianceService;
    private final DeviceLocationService deviceLocationService;
    private Grid<Appliance> applianceGrid;

    private ApplianceFilter applianceFilter;
    private Binder<ApplianceFilter> applianceFilterBinder;
    private User user;
```

Zdrojový kód 2 - horní menu spotřebiče s filtrem [zdroj vlastní]

Při pohledu na předcházející zdrojový kód lze usoudit, že:

- customerComboBox a deviceLocationComboBox jsou formulářová pole sloužící k filtraci
- applianceGrid je filtrovaný grid
- applianceFilter je backendový objekt obsahující informace z customerComboBox a deviceLocationComboBox
- applianceFilterBinder zajišťuje propsání dat z formulářových polí do backendového objektu a opačně

5.4. Gridy

Gridy použité v aplikaci představují hlavní přístup k uloženým datům v aplikaci, díky tomu představují jednu z klíčových komponent v aplikaci. Uchovávají seznam entit načtených z databáze, jejich proklikem, případně i tlačítky, jsou-li u nich dostupné, je možné se přesměrovat na detail entity. Jedná se o tabulky, které zobrazují data na základě zvolené entity a jejích vybraných atributů. Vaadin gridy umožňují řazení dat, řazení sloupců, měnit šířku sloupců, vybírat řádky, proklikávat na položky v nich zobrazené. Sloupce v těchto gridech mohou být buď jednoduché, vytvořené na základě atributů entit, anebo složitější, vlastní komponenty. Vaadin gridy podporují lazy loading, díky použití této techniky není nutné do paměti načíst veškerá data naráz, ale grid se podle pozice na stránce postará o správné načtení dat. Šířka sloupců je v gridu v základním nastavení responzivní, ale jejich počet se dynamicky

nemění, takže responzivitu pro menší displeje je nutné naprogramovat, v této aplikaci například zavedením vlastní komponenty, přičemž pod definovanou velikost displeje dojde k nahrazení sloupců vlastní komponentou, která je responzivní.

Níže, na ukázce zdrojového kódu 3, je uveden zdrojový kód gridu pro zobrazení klientů technika, počet sloupců na uvedeném příkladu je zkrácen:

```
private Grid<Customer> createGrid() {
    customerGrid = new Grid<>(Customer.class, false);
    customerGrid.setSelectionMode(Grid.SelectionMode.NONE);
    customerGrid.addComponentColumn(customer -> new
CustomerResponsiveGridItem(customer, customerUpperMenu,
customerService))
            .setHeader("Klient")
            .setAutoWidth(true)
            .setKey("customer")
            .setVisible(false);
    customerGrid.addColumn(Customer::getName)
            .setHeader("Jméno")
            .setKey("name")
            .setAutoWidth(true)
            .setSortable(true);
    customerGrid.addColumn(
            new LocalDateTimeRenderer<>(customer ->
                    customer.getCreatedOn() == null? null
                            :

LocalDateTime.ofInstant(customer.getCreatedOn(),
APPLICATION_ZONE_OFFSET), "dd. MM. yyyy HH:mm", CZECH_LOCALE,
"")
            )
            .setHeader("Vytvořen")
            .setSortable(true)
            .setAutoWidth(true)
            .setKey("createdOn");

customerGrid.addComponentColumn(this::getActionButtons).setTex
tAlign(ColumnTextAlign.END)
            .setSortable(false)
            .setAutoWidth(true)
            .setKey("actions")
            .setFrozenToEnd(true)
            .setFlexGrow(0);

    customerGrid.addItemDoubleClickListener(
            e ->
UI.getCurrent().navigate(CustomerDetailView.class, new
RouteParameters("customerId",
String.valueOf(e.getItem().getCustomerId())))
```

```
    );
    customerGrid.setEmptyStateText("Na základě zvolených
filtrů nebyli nalezeni žádní klienti");
    customerGrid.setColumnReorderingAllowed(true);
    customerGrid.getColumns().forEach(customerColumn ->
customerColumn.setResizable(true));

    return customerGrid;
}
```

Zdrojový kód 3 - Customer grid [zdroj vlastní]

Výše uvedený zdrojový kód obsahuje:

- customerGrid – jedná se o samotný grid
- customerGrid.setSelectionMode(NONE) – v gridu není možné vybírat položky pomocí zaškrtávacího políčka, dalším použitým Selection Mode v aplikaci je MULTI pro umožnění výběru více položek, a to například u gridu QR kódů
- customerGrid.addColumn – přidání jednoduchého sloupce do gridu
- customerGrid.addComponentColumn – přidání komponenty jako sloupce do gridu, zde se jedná jak o CustomerResponsiveGridItem pro responzivní zobrazení sloupců gridu, tak i o tlačítka pro přechod na zákazníka a smazání zákazníka
- customerGrid.addItemDoubleClickListener – ovládání pro přesměrování na detail klienta při dvojkliku na odpovídající položku gridu
- customerGrid.setEmptyStateText – text, který se v gridu zobrazí místo hodnot, pokud nejsou dostupné
- customerGrid.setColumnReorderingAllowed(true) – pořadí sloupců je možné přetažením změnit
- customerGrid.getColumns().forEach(customerColumn-> customerColumn.setResizable(true)) – smyčka pro nastavení povolení změny velikosti pro všechny sloupce

Responzivní zobrazení sloupců v gridu je řešeno následující funkcí, uvedenou na příkladě zdrojového kódu 4, která je registrována na událost změny velikosti stránky:

```
private void adjustVisibleGridColumns(int width) {
    // Change which columns are visible depending on browser
width
    int WIDTH_BREAKPOIN = 1100;
    customerGrid.getColumnByKey("customer").setVisible(width
<= WIDTH_BREAKPOIN);
    customerGrid.getColumnByKey("name").setVisible(width >
WIDTH_BREAKPOIN);
```

```
    customerGrid.getColumnByKey("email").setVisible(width >
WIDTH_BREAKPOIN);
    customerGrid.getColumnByKey("phone").setVisible(width >
WIDTH_BREAKPOIN);
    customerGrid.getColumnByKey("ico").setVisible(width >
WIDTH_BREAKPOIN);
    customerGrid.getColumnByKey("createdOn").setVisible(width
> WIDTH_BREAKPOIN);
    customerGrid.getColumnByKey("actions").setVisible(width >
WIDTH_BREAKPOIN);
}
```

Zdrojový kód 4 - funkce na zajištění responzivity sloupců [zdroj vlastní]

Ve výše uvedeném zdrojovém kódu lze vyčíst, že pokud je šířka stránky menší než 1100 pixelů, tak se zobrazí responzivní komponenta gridu (mapováno pod klíčem customerGrid.getColumnByKey("customer"), viz příklad gridu výše), jinak se zobrazí ostatní sloupce gridu.

Responzivní komponenta v gridech jsou řešený pomocí přidání sloupce do gridu, který je, jak je patrné na předešlé ukázce kódu, skrytý do zvolené šířky, výše se jedná o 1100 pixelů. Responzivita komponenty je zajištěna pomocí dědění z HorizontalLayout, což je Vaadin komponenta pro Flexbox layout s komponentami zarovnanými vedle sebe. V tomto layoutu jsou dále vložené další prvky, které jsou vhodně umístěny tak, aby výsledná podoba komponenty byla použitelná i na displejích mobilních zařízení s menší zobrazovací plochou.

Níže, na ukázce zdrojového kódu 5, je ukázka části kódu komponenty pro responzivní zobrazení údajů o klientovi, komponenta je pojmenována CustomerResponsiveGridItem:

```
public class CustomerResponsiveGridItem extends
HorizontalLayout {

    private final Customer customer;
    private final CustomerService customerService;

    private CustomerUpperMenu customerUpperMenu;

    public CustomerResponsiveGridItem(Customer customer,
CustomerUpperMenu customerUpperMenu, CustomerService
customerService) {
        this.customer = customer;
        this.customerService = customerService;
        this.customerUpperMenu = customerUpperMenu;
        createLayout();
    }

    private void createLayout() {
```

```
        setClassName(LumoUtility.Padding.Vertical.SMALL);
        setSpacing(false);

        VerticalLayout verticalLayout = new VerticalLayout();
        verticalLayout.setPadding(false);
        verticalLayout.setSpacing(false);
        verticalLayout.addClassName(LumoUtility.Gap.XSMALL);
        verticalLayout.add(getName(), getEmail(), getPhone(),
getIco(), getCreatedOn());

        HorizontalLayout buttons = getButtons(customer);
        add(verticalLayout, buttons);
    }

    private Component getCreatedOn() {
        HorizontalLayout customerCreatedOnRow = new
HorizontalLayout();

customerCreatedOnRow.setClassName(LumoUtility.FontSize.XXSMALL
);
        Icon calendarIcon = new Icon(VaadinIcon.CALENDAR);
        calendarIcon.setTooltipText("Vytvořen");
        calendarIcon.addClassName(LumoUtility.IconSize.SMALL);
        customerCreatedOnRow.add(calendarIcon, new
Text(customer.getCreatedOn() == null? "" :
customer.getCreatedOn().atOffset(APPLICATION_ZONE_OFFSET).form
at(DateTimeFormatter.ofPattern("dd.MM.yyyy"))));
        return customerCreatedOnRow;
    }
```

Zdrojový kód 5 - ukázka responzivní komponenty CustomerResponsiveGridItem [zdroj vlastní]

Z výše uvedeného kódu je patrné, že layout je vytvořen v Javě s použitím instancí tříd Vaadinu. Layout obsahuje komponenty vytvořené metodami getName, getEmail, getPhone, getIco, getCreatedOn, které představují jednotlivé vlastnosti entity klienta. Komponenty je možné skládat z libovolného množství subkomponent, jak je patrné na příkladu komponenty pro datum vytvoření entity klienta – výše se jedná o kód metody getCreatedOn. Tato komponenta obsahuje ikonku kalendáře a časový údaj, kdy byl daný klient vytvořen.

5.5. Formuláře

Dalšími klíčovými komponentami použitými v aplikaci jsou formuláře, které slouží k zobrazení a úpravě jednotlivých záznamů. Umožňují editaci konkrétních entit i jejich logických celků, napojení jednotlivých záznamů na sebe. Klíčovými formuláři použitými v aplikaci jsou formuláře pro tvorbu a úpravu revizí, tvorbu a úpravu klientů, tvorbu a úpravu spotřebičů a jejich přiřazení ke QR kódu. Použité formuláře v aplikaci, kromě formulářů pro registraci, přihlášení a resetování hesla, dědí z vlastní generické třídy AbstractDetailForm<V>, která je

založená na třídě VerticalLayout z toho důvodu, aby měla zajištěnou responzivita, součástí takto zděděné responzivity je i zarovnání child komponent pod sebe. Kromě responzivity také dědění z této komponenty zajišťuje, že vlastní komponenta je potomek třídy Component, bez toho, aniž by vlastní třída byla potomek této třídy, tak ji není možné vložit na stránku. Výhodou využití vlastní třídy AbstractDetailForm je deduplikace kódu díky zajištění dědění Vaadin binderu pro provázání formulářových prvků s entitou, vlastní ErrorComp, která slouží pro výpis chybových hlášek, a dědění formLayout, ve kterém jsou uspořádány vlastní pole formulářů. AbstractDetailFrom také pomocí anotace @CssImport zajišťuje import pro styl tlačítek, který vylepšuje jejich zalamování textu.

Na dále uvedené ukázce zdrojového kódu 6 je vidět implementace AbstractDetailForm:

```
@CssImport(value = "./themes/sprava-revizi/components/wrap-
button.css", themeFor ="vaadin-button")
public abstract class AbstractDetailForm <V> extends
VerticalLayout {
    protected Binder<V> binder;
    protected ErrorComp errorComp;
    protected FormLayout formLayout;

    protected abstract void createLayout();
    protected abstract void initFormLayout();
    protected abstract void bindBinderToFields();

    public void refreshBinding() {
        binder.refreshFields();
    }

    public boolean isValidationOk() {
        return binder.validate().isOk();
    }

    public V getEntity() {
        return this.binder.getBean();
    }
}
```

Zdrojový kód 6 – AbstractDetailForm [zdroj vlastní]

Samotné formulářové prvky jsou napsané takovým způsobem, aby podporovaly možnost zanoření, například pro Adresu je použita komponenta AddressDetailForm, která je použita jednak u klienta, tak i u revizí.

5.6. FormLayout

Navigační menu v aplikaci a veškerá pole zobrazená ve formulářích, která nejsou zděděna z vnořeného formuláře, jsou umístěna do komponenty FormLayout. Tato Vaadin komponenta

umožňuje responzivní uspořádání nejen polí, ale i libovolných jiných komponent, do řádků a sloupců s přihlédnutím k šířce okna prohlížeče. Díky možnosti definování responzivních kroků a počtu zobrazených sloupců v jednotlivých krocích se rozložení dynamicky přizpůsobuje aktuální velikosti zobrazeného okna. U FormLayout je také možné, podobně jako u ostatních typů Vaadin layoutů založených na FlexboxLayout, nastavit u komponenty možnost zobrazení na více sloupců než jeden, v takovém případě zabere komponenta vybraný počet sloupců, maximálně však celý řádek.

Níže, ve zdrojovém kódu 7, se nachází ukázka využití FormLayout ve formuláři AddressFormDetailForm:

```
@Override
protected void initFormLayout() {
    formLayout = new FormLayout();
    formLayout.setResponsiveSteps(
            // Use one column by default
            new FormLayout.ResponsiveStep("0", 1),
            // Use two columns, if layout's width exceeds
400px
            new FormLayout.ResponsiveStep("400px", 2),
            new FormLayout.ResponsiveStep("600px", 3)
    );

    formLayout.add(addressTextField, 3);
    formLayout.add(cityTextField, zipTextField,
countryComboBox, latitudeBigDecimalField,
longitudeBigDecimalField, findAddressInRuian);
}
```

Zdrojový kód 7 - příklad použití formLayout ve třídě AddressFormLayout [zdroj vlastní]

Na výše uvedeném příkladu je vidět, že AddressFormDetailForm používá tři různé styly zarovnání sloupců podle šířky obrazovky. Pokud je šířka stránky menší než 400 px, zobrazí se jeden sloupec na řádek. Při šířce 400–600 px se zobrazí dva sloupce a při šířce 600 px a více se uspořádání rozšíří na tři sloupce. Také je zde uveden příklad komponenty, která se rozšíří na více sloupců – konkrétně addressTextField. Toto pole zůstane roztažené na celou šířku stránky, bez ohledu na její velikost.

Implementace některých formulářových prvků v aplikaci umožňuje jejich zanořování, což je obdobné jako u běžných formulářových prvků. Níže, ve zdrojovém kódu 8, je uveden příklad, jak lze přidat formulář adresy do formuláře klienta:

```
private void initAddressDetailForm() {
    addressHeaderText = "Adresa klienta";
    addressDetailForm = new AddressDetailForm(readOnly,
```

```
addressHeaderText, customer.getAddress(), errorComp,
addressService, ruianService);
}
```

Zdrojový kód 8 - ukázka přidáni vnořeného layoutu [zdroj vlastní]

5.7. Vyhledávání klientů v ARES

Možnost načítání klientů a jejich informací je klíčovou funkcionalitou aplikace. Klienty lze vyhledávat podle IČO, přičemž načítání údajů z ARES výrazně usnadňuje práci revizním technikům, kteří tak nemusí manuálně dohledávat a vyplňovat informace o klientech a jejich adresách do formulářů. Komponenta pro vyhledávání klientů v ARES je tvořena polem pro zadání IČO a tlačítkem „Vyhledat klienta v ARESu“, které pomocí přístupu k ARES REST api na endpointu „Ekonomické subjekty“ umožňuje dle IČO vyhledání klienta a jeho adresy v ARES. Nalezené údaje o klientovi jsou z JSON formátu transformovány pomocí business logiky do údajů o entitě. Klíčovou metodou pro extrakci a transformaci dat získaných z ARES je metoda, která převádí atribut z obdrženého JSON objektu na String. Tato metoda je implementována následujícím způsobem, viditelném ve zdrojovém kódu 9:

```
private String getNodeAttributeValue(JsonNode jsonNode, String
attributeName) {
    JsonNode attributeNode = jsonNode.get(attributeName);
    if(attributeNode != null) {
        return attributeNode.asText("");
    } else {
        return "";
    }
}
```

Zdrojový kód 9 - metoda pro transformaci JSON node na hodnotu atributu [zdroj vlastní]

Parametr jsonNode je vybraný uzel, z kterého se mají přečíst data, attributeName je hledaný atribut.

Po kliknutí na tlačítko „Vyhledat klienta v ARESu“ se, kromě pokusu o nalezení klienta v ARES, také automaticky zavolá volání RÚIAN pro dohledání zeměpisné šířky a délky načtené adresy klienta.

5.8. Vyhledávání adres v RÚIAN

Pomocí integrace s RÚIAN REST Api je v aplikaci řešeno hledání zeměpisné šířky a zeměpisné délky adres revizí a klientů. Hledání v RÚIAN je možné pouze pro české adresy. Vstupem do vyhledávání jsou adresní řádka adresy, PSČ, město. Údaje o zeměpisné šířce a délce jsou přebírány ve formátu JSON a jsou obdobným způsobem, jako při zpracování obdržených dat z ARES, transformovány na zeměpisnou délku a šířku adresy, ale vzhledem k tomu, že

souřadnice získané z RUIANU jsou ve formátu EPSG:5514, ale v aplikaci se používají souřadnice GPS dle standardu EPSG:4326, tak je nutné zajistit konverzi souřadnicových systémů, v aplikaci je zajištěna následujícím způsobem, uvedeným níže ve zdrojovém kódu 10:

```
public static double[] convertRuianCartographicToGps(double x,
double y) {
    CRSFactory factory = new CRSFactory();
    CoordinateReferenceSystem srcCrs =
factory.createFromName("EPSG:5514");
    CoordinateReferenceSystem dstCrs =
factory.createFromName("EPSG:4326");

    BasicCoordinateTransform transform = new
BasicCoordinateTransform(srcCrs, dstCrs);

    // Note these are x, y so lng, lat
    ProjCoordinate srcCoord = new ProjCoordinate(x, y);
    ProjCoordinate dstCoord = new ProjCoordinate();

    // Writes result into dstCoord
    transform.transform(srcCoord, dstCoord);

    double[] xy = new double[2];
    xy[0] = dstCoord.x;
    xy[1] = dstCoord.y;
    return xy;
}
```

Zdrojový kód 10 - převod EOSG:5514 na EPSG:4326 [zdroj vlastní]

Vzhledem ke komentářům u zdrojového kódu ho není nutné blíže vysvětlovat.

Pokud se na základě vyplněných údajů podaří nalézt adresu, tak na základě převzaté a transformované zeměpisné šířky a délky dojde k zobrazení mapy dle obdržených souřadnic.

5.9. Mapa

Mapa pro zobrazení adresy na základě zeměpisné šířky a délky se využívá jak u klientů, tak u revizí. Aplikace využívá komponentu Vaadin Map, která je součástí Vaadin Pro. Tato mapa podporuje posun, centrování, přiblížení a oddálení, stejně jako přidávání značek souřadnic ve formátu GPS dle standardu EPSG:4326. Pro implementaci do této aplikace byla vybrána zejména díky nativní integraci s ostatními Vaadin komponentami a jednoduchému přidání značek na mapě, kde pro přidání značky stačí předat pouze zeměpisnou šířku a délku bodu.

5.10. Kalendář

Další klíčovou komponentou v aplikaci je kalendář, který je v aplikaci použit pro zobrazení termínu revizí. Zobrazené revize se liší podle typu přihlášeného uživatele – revizní technik vidí

revize, které provedl, zatímco klient pouze ty, které se ho týkají. Kalendář pro revize byl implementován s využitím kalendáře z oficiálně doplňku rozšíření pro Vaadin FullCalendar for Flow. Aplikace vyvinutá v této práci využívá kalendář především pro plánování událostí a pro zachycení a zpracování události při kliknutí na naplánované položky. Dále byla využita podpora responzivity v případě proměnlivého počtu týdnů v měsíci a částečná podpora internacionalizace, která umožnila lokalizaci kalendáře do češtiny. Kalendář je využit jak na obrazovce „Hlavní panel“, tak i na obrazovce „Kalendář“. V kalendáři je možné měnit zobrazený měsíc a klikat na jednotlivé zobrazené revize. Zobrazené položky revizí jsou v kalendáři barevně odlišeny na základě toho, jaká událost revizí v daném dni nastala. Žlutě jsou vyznačeny dny, kdy byla alespoň jedna revize založena a alespoň jedna revize v daném dni vyprší, zeleně jsou označeny dny, kde byla alespoň jedna revize vytvořena, červeně pak dny, kdy alespoň jedna revize končí. Dále pak je s kalendářem propojen i seznam revizí, ve kterém se, v případě použití na obrazovce „Hlavní panel“ zobrazují prošlé revize a revize procházející v následujících 30 dnech, v případě kalendáře na své vlastní obrazovce „Kalendář“ se seznamem revizí v daném měsíci filtrované dle výběru jedné z následujících možností: „Všechny revize“, „Dokončené revize“, „Nedokončené revize“. Při implementaci kalendáře nastal problém s nemožností zabránit přesouvání naplánovaných revizí. Ačkoli to neovlivňovalo funkcionalitu ani data aplikace, negativně to dopadalo na uživatelský zážitek. Pokus o omezení přesouvání položek pomocí doporučeného postupu z dokumentace však nefungoval podle očekávání, proto bylo problém nutné vyřešit jiným způsobem. Tento problém byl vyřešen workaroundem pomocí přidání JavaScript funkce na stránku, které Vaadin díky své nativní podpoře pro provádění JavaScript skriptů podporuje, výsledné řešení, spolu se způsoby, které pravděpodobně měly zafungovat, ale nezafungovaly, je uvedeno ve zdrojovém kódu 11:

```
calendar.setEditable(false);
calendar.setEntryDurationEditable(false);
calendar.setTimeslotsSelectable(false);
calendar.setEntryResizableFromStart(false);
//Konec toho, co by asi mělo fungovat, ale nefunguje, pod tím
js hack
UI.getCurrent().getPage().executeJs(
"setInterval(() => {" +
        "    document.querySelectorAll('.fc-event-draggable,
.fc-event-resizable, .fc-event-resizer, .fc-event-resizer-
end').forEach(el => {" +
        "        el.removeAttribute('draggable');" +
        "        el.classList.remove('fc-event-draggable',
'fc-event-resizable', 'fc-event-resizer', 'fc-event-resizer-
end');" +
```

```
        "    });" +
        "}, 100);"
);
```

Zdrojový kód 11 - ukázka workaroundu pro zamezení přesouvání položek kalendáře [zdroj vlastní]

Z výše uvedeného kódu by mělo teoreticky stačit nastavit calendar.setEditable(false), kromě toho by měla fungovat kombinace následujících 3 řádků pod tím příkazem, toto řešení však nezabránilo přesouvání a změně velikosti slotů, výše uvedená JavaScript funkce proto na stránce najde prvky s třídami .fc-event-draggable, .fc-event-resizable, .fc-event-resizer, .fc-event-resizer-end a tyto třídy z těchto prvků odstraní. Funkce je vytvořena jako setInterval z toho důvodu, aby při přesunu kalendáře na jiný měsíc došlo k odstranění tříd i na nově načteném měsíci.

5.11. Generování QR kódů

Generování QR kódů pro spotřebiče je další výhodou aplikace. Generování QR kódů se v aplikaci skládá ze dvou částí, těmi jsou vytvoření databázových entit, v této fázi je pouze vytvořena databázová entita, a ze samotného vygenerování PDF dokumentu, který je naplněn vybranými QR kódy.

Generování entit QR kódů probíhá na stránce „Generování QR kódů“, kde uživatel zadá počet kódů (celé číslo od 1 do 100) a stisknutím tlačítka „Vygenerovat nové QR kódy“ spustí jejich vytvoření.

Dokument s QR kódy lze vygenerovat dvěma způsoby:

1. Na stránce generování QR kódů – uživatel vybere požadované QR kódy v gridu, zvolí jejich velikost a klikne na tlačítko „Vygenerovat PDF dokument s vybranými QR kódy“.
2. Na kartě spotřebiče – po výběru velikosti QR kódu lze pomocí tlačítka „Vygenerovat PDF dokument s QR kódem spotřebiče“ vytvořit dokument obsahující QR kód daného spotřebiče.

Obrázky QR kódů jsou v aplikaci generovány použitím knihovny Spire.Barcode, nad QR kódy se také nachází ID QR kódu, které je zde přidáno pomocí kreslení do Graphics Javy.

Kód pro vygenerování obrázku QR kódu, očištěný od poznámek, vypadá dle zdrojového kódu 12, který je implementován následovně:

```
public ByteArrayOutputStream generateQrCodeImage(QrCode
qrCode, int size) {
    BarcodeSettings settings = new BarcodeSettings();
```

```
    settings.setType(BarCodeType.QR_Code);
    String data = qrCodeBaseUrl + "/" + getQrSpecificUrl + "/"
+ qrCode.getQrCodeId();
    settings.setData(data);
    settings.setX(2);
    settings.setQRCodeECL(QRCodeECL.M);
    settings.setShowText(false);
    settings.setShowTextOnBottom(false);
    settings.hasBorder(false);
    settings.setTopMargin(25);
    settings.setLeftMargin(10);
    settings.setRightMargin(10);
    BarCodeGenerator barCodeGenerator = new
BarCodeGenerator(settings);
    BufferedImage qrCodeImageReadOnly =
barCodeGenerator.generateImage(new Dimension(size, size));
    Graphics g = qrCodeImageReadOnly.getGraphics();
    drawCenteredString(g, qrCode.getQrCodeId().toString(),
qrCodeImageReadOnly, new Font("Verdana", Font.BOLD, 50));
    g.dispose();
    try(ByteArrayOutputStream qrCodeImageOutputStream = new
ByteArrayOutputStream()) {
        ImageIO.write(qrCodeImageReadOnly, "PNG",
qrCodeImageOutputStream);
        return qrCodeImageOutputStream;
    } catch (Exception ex) {
        throw new RuntimeException("Nepodařilo se vygenerovat
QR kód.");
    }
}
```

Zdrojový kód 12 - generování obrázku QR kódu [zdroj vlastní]

Nejzajímavějšími částmi ve výše uvedeném zdrojovém kódu jsou části settings.setQRCodeECL(QRCodeECL.M), která nastavuje míru samoopravitelnosti QR kódu na střední, metoda drawCenteredString, která je vlastní metodou vykreslující číslo s ID daného QR kódu nad vygenerovaným QR kódem. Z návratové hodnoty funkce lze poznat, že obrázek je vracen jako pole bajtů.

5.12. Generování dokumentu revize spotřebiče

Jedná se významnou část při práci se spotřebiči. Generování protokolu o revizích a kontrolách elektrického spotřebiče přímo v aplikaci usnadňuje reviznímu technikovi tvorbu revizních dokumentů. Na základě údajů vyplněných na kartě spotřebiče může technik vygenerovat dokument ve formátu Excel nebo PDF a stisknutím tlačítka „Vygenerovat dokument revize“ tento dokument vytvořit, přičemž k tomu využije i svůj vložený podpis na stránce „Můj profil“. Při generování dokumentu ve formátu Excel se používá šablona, do které se vyberou příslušná

políčka a do těchto polí se doplní hodnoty z obrazovky spotřebiče. V případě generování dokumentu ve formátu PDF se celý dokument sestaví včetně hodnot z karty prohlížeče a statických textů na protokolu, přičemž struktura PDF dokumentu vychází z excelového podkladu.

V aplikaci je za generování dokumentů odpovědná služba DocumentService, která na základě zvoleného typu dokumentu volá buď ExcelService nebo PdfService.

Vygenerování dokumentu ve formátu excel je zajištěno následujícím způsobem, uvedeném na příkladu zdrojového kódu 13:

```
public byte[]
generateElectronicDeviceRevisionFileExcel(Appliance device) {

    try(InputStream in =
classLoader.getResourceAsStream(excelResourcesPath);
        XSSFWorkbook wb = new XSSFWorkbook(in, true);
        ByteArrayOutputStream bos = new
ByteArrayOutputStream();
        BufferedOutputStream excelOutputStream = new
BufferedOutputStream(bos);
    ) {
        XSSFSheet sheet = wb.getSheet("List1");

        fillFirstRow(device, sheet);
        fillSecondRow(device, sheet);
        fillThirdRow(device, sheet);
        fillFourthRow(device, sheet, wb);

        wb.write(excelOutputStream);
        excelOutputStream.flush();
        wb.close();
        return bos.toByteArray();
    } catch (Exception ex) {
        throw new RuntimeException(ex.getMessage());
    }
}
```

Zdrojový kód 13 - generování excel dokumentu [zdroj vlastní]

Ve výše uvedeném kódu se v try bloku nejprve otevře zdrojová šablona, na základě šablony se vytvoří nový excel dokument, poté se v dokumentu vybere List1, pomocí metod fillXRow se do dokumentu na příslušné pozice zapíšou data spotřebiče a vloží se podpis revizního technika, poté příkazy:

- wb.write(excelOutputStream);
- excelOutputStream.flush();

- wb.close();
- return bos.toByteArray();

je do dokumentu zapsán obsah a ten je vrácen ve formě pole bajtů.

Metody fillXRow slouží k doplnění konkrétního řádku pomocí absolutních adres buněk v Excelu. Jejich implementace je podobná, zde, na ukázce zdrojového kódu 14, je uveden příklad metody pro vyplnění prvního řádku – metody fillFirstRow:

```
private void fillFirstRow(Appliance device, XSSFSheet sheet) {
    Row row1 = sheet.getRow(4); //5 radek
    Cell revisedDeviceCell = row1.getCell(0);
    revisedDeviceCell.setCellValue(device.getDeviceName());
    Cell inventoryNumberCell = row1.getCell(3);

inventoryNumberCell.setCellValue(device.getDisplayedInventoryN
umber());
    Cell manufacturerCell = row1.getCell(6);
    manufacturerCell.setCellValue(device.getManufacturer());
    Cell yearMadeCell = row1.getCell(11);
    yearMadeCell.setCellValue(device.getYearMade());
    Cell serialNumberCell = row1.getCell(14);
    serialNumberCell.setCellValue(device.getSerialNumber());
}
```

Zdrojový kód 14 - metoda fillFirstRow pro vyplnění 1. řádku excel dokumentu [zdroj vlastní]

V výše uvedené ukázky kódu je vidět, že se jedná jen o propis hodnot z databázové entity do vybrané buňky generovaného dokumentu.

6. TESTOVÁNÍ A NASAZENÍ

Cílem této kapitoly je seznámit čtenáře s vytvořenými testy použitými v aplikaci a s průběhem nasazením aplikace.

6.1. Metoda testování použitá v aplikaci

Vzhledem k tomu, že vyvinutá aplikace neobsahuje žádnou složitou logiku a jedná se zejména o aplikaci realizující operace CRUD a testování správné funkce samotných komponent není rámcem této aplikace, ale mělo by být provedeno vývojáři Vaadin, tak v aplikaci nebylo použito jednotkové testování, ale bylo rovnou přikročeno k použití integračních testů.

Integrační testy aplikace se zaměřují na ověření přítomnosti a viditelnosti správných prvků na pohledových obrazovkách a správnosti navigace při pokusu o vytvoření nové entity. U pohledových tříd obsahujících komponentu typu grid je testována její přítomnost a správné zobrazení. Pro pohledy umožňující vytváření nových entit se ověřuje funkčnost tlačítka a přesměrování na formulář pro vytvoření nové entity. Naopak u pohledových tříd, kde není možné vytvářet nové entity, je testováno, že tato možnost není dostupná.

Pro otestování aplikace pomocí integračních testů byly použity integrační testy z Vaadin Test Bench pro JUnit5, jedná se o testy prováděné třídami dědícími z třídy BrowserTestBase. Výhodou využití této třídy je její přímá podpora pro Vaadin komponenty, která významně usnadňuje tvorbu testů díky přímé podpoře parametrizovaného vyhledávání frontendových komponent vytvořených pomocí Vaadin backendu. Třída mimo jiné obsahuje i Driver a metodu pro změnu Driveru pro spuštění a provedení testů ve vybraném prohlížeči. Testy mohou být jak bezparametrické, tak i parametrické, v této práci jsou využité pouze bezparametrické testy. [28]

Pro zpřehlednění výsledných testů aplikace a snížení množství opakovaného kódu byly v aplikaci využity dvě konfigurační třídy, TestConfig a TestConfigLogin. Metoda TestConfig dědí z BrowserTestBase a dále obsahuje metody pro nastavení testového prostředí a driveru, tato třída je vytvořena způsobem uvedeným ve zdrojovém kódu 15:

```
@SpringBootTest
//DriverSupplier should not be needed -> but it is required
https://vaadin.com/forum/t/testbench-selenium-failing-on-
linux/168156/2
//Otherwise broken chrome driver that has a broken session and
can only be closed launches
public abstract class TestConfig extends BrowserTestBase
implements DriverSupplier {
    @BeforeEach
```

```
    public void setup() {
        testBench().resizeViewPortTo(1600, 900);
        // Wait for frontend compilation complete before
testing
        waitForDevServer();
    }

    @AfterEach
    public void teardown() {
        getDriver().close();
    }

    @Override
    public WebDriver createDriver() {
        FirefoxOptions firefoxOptions = new FirefoxOptions();
        firefoxOptions.addArguments("--headless");
        return new FirefoxDriver(firefoxOptions);
    }
}
```

Zdrojový kód 15 - třída TestConfig [zdroj vlastní]

Ve zdrojovém kódu třídy TestConfig se nachází metoda setup() realizující nastavení velikosti okna prohlížeče a čekání na běh aplikace. Metoda teardown je odpovědná za správné zavření Driveru po provedení testů. Metoda CreateDriver zajišťuje spuštění testů v prohlížeči Mozilla Firefox, možnost –headless umožňuje provedení integračních testů bez spuštění UI okna prohlížeče.

Třida TestConfigLogin rozšiřuje třídu TestConfig o metodu login, která zajišťuje přihlášení uživatele xHeslo123! s heslem xHeslo123! s rolí admin. Pro správné provádění testů s tímto uživatelem se předpokládá jeho existence v DB. Kód této metody zde vzhledem k jeho jednoduchosti a nezajímavosti nebude uveden.

Vybranými obrazovkami, které jsou testovány, jsou View třídy dostupné z hlavního navigačního menu a pak také detailView třídy s formuláři pro práci s entitami v aplikaci, konkrétně se jedná o testy pro třídy:

- ApplianceView
- CalendarView
- CustomerDetailView
- CustomerView
- DashBoardView
- QrCodeDetailView
- QrCodeView

- RevisionDetailView
- RevisionView
- RevisionTagDetailView
- RevisionTagView
- UserManagementDetailView
- UserManagementView

Níže, na ukázce zdrojového kódu 16, je zdrojový kód třídy CustomerViewTest, která realizuje test pro navigační pohled CustomerView:

```
class CustomerViewTest extends TestConfigLogin {
    private String baseUrl = "http://localhost:8080/clients/";

    @BrowserTest
    public void gridPresent() {
        getDriver().get(baseUrl);
        login();

assertTrue($(GridElement.class).single().isDisplayed());
    }

    @BrowserTest
    public void upperMenuIsPresent() {
        getDriver().get(baseUrl);
        login();

assertTrue($(TextFieldElement.class).withCaption("Hledat dle
názvu klienta").single().isDisplayed());
        assertTrue($(ButtonElement.class).withText("Přidat
nového klienta").single().isDisplayed());

assertTrue($(TextFieldElement.class).withCaption("Email").sing
le().isDisplayed());

assertTrue($(TextFieldElement.class).withCaption("Telefon").si
ngle().isDisplayed());

assertTrue($(TextFieldElement.class).withCaption("IČO").single
().isDisplayed());

assertTrue($(TextFieldElement.class).withCaption("DIČ").single
().isDisplayed());
    }

    @BrowserTest
    public void clickOnAddNewClientRedirects() {
        getDriver().get(baseUrl);
```

```
        login();
        $(ButtonElement.class).withText("Přidat nového
klienta").single().click();

Assertions.assertEquals("http://localhost:8080/clients/novy",
getDriver().getCurrentUrl());
        assertTrue($(H1Element.class).withText("Nový
klient").single().isDisplayed());
    }
}
```

Zdrojový kód 16 - testovací třída CustomerViewTest [zdroj vlastní]

Ve výše uvedeném kódu se nachází metoda gridPresent, ve které je kontrolována přítomnost gridu na stránce, dále se zde nachází metoda upperMenuIsPresent, která ověřuje přítomnost navigačního menu na stránce, poslední použitou testovací metodou je metoda clickOnAddNewClientRedirects, v které se nachází metoda provádějící kontrolu správného provedení přesměrování na stránku vytvoření nového klienta. Za povšimnutí stojí zejména metody $(), například assertTrue($(GridElement.class).single().isDisplayed()); – jedná se o metody Vaadin Test Bench, pomocí kterých je možné na stránce vybírat elementy, a to i pomocí různých parametrů, například pomocí withCaption.

6.2. Nasazení

Pro správné fungování Vaadin aplikace je nutné vytvořit JAR nebo WAR soubor s použitím přepínače -Pproduction, který, kromě optimalizace buildu pro běh na serveru díky předkompilovaní frontendu a odebrání závislostí používaných pouze pro debugging aplikace, zajistí správné vložení závislostí do vytvořeného souboru JAR nebo WAR. Jeden z možných způsobů, za použití CLI pro sestavení souboru JAR nebo WAR (dle konfigurace souboru POM) pomocí nástroje Maven vypadá následovně:

- mvn clean package -Pproduction

Takto sestavený JAR nebo WAR soubor je pak možné způsobem běžným pro Spring Boot aplikace nasadit na produkční server. [29]

6.2.1. Nasazení pomocí Docker

Jedním z požadavků této diplomové práce je, aby výsledná aplikace byla kontejnerizovatelná. Pro účely kontejnerizace aplikace byla vytvořen DockerFile, který slouží k tvorbě vlastní image. Kromě souboru Dockerfile jsou v zdrojových souborech aplikace pro usnadnění sestavení Docker image se správnými parametry vytvořeny soubory .env a docker-

compose.yml. Soubor .env slouží k definici proměnných prostředí systému, které je pak možné použít buď v příkazu:

- Docker build

anebo při sestavení aplikace s použitím docker-compose.yaml pomocí příkazu:

- docker compose up

Níže se nachází zdrojový kód 17, který je obsažen v souboru Dockerfile:

```
FROM openjdk:21-jdk-bookworm
COPY target/*.jar app.jar

RUN echo "deb http://deb.debian.org/debian stable main contrib
non-free" > /etc/apt/sources.list && \
    apt update && \
    apt install -y ttf-mscorefonts-installer fontconfig && \
    fc-cache -fv && \
    rm -rf /var/lib/apt/lists/*

EXPOSE 8080
ENV QR_CODE_BASE_URL=http://localhost:8080
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Zdrojový kód 17 - Dockerfile pro kontejnerizaci aplikace [zdroj vlastní]

Výše je uveden zdrojový kód pro vytvoření Docker image založeného na linuxové distribuci Debian s OpenJDK 21 (image openjdk:21-jdk-bookworm). Předpokladem pro vytvoření Docker image je vytvoření JAR souboru pomocí nástroje Maven, například pomocí příkazu uvedeného na začátku této kapitoly. Jedná se o jednoduchý Dockerfile – za pozornost stojí zejména část s příkazem RUN, která zajišťuje instalaci Microsoft fontů potřebných pro generování revizních dokumentů. Proměnná prostředí QR_CODE_BASE_URL=http://localhost:8080 je zde uvedena pouze jako pojistka pro případy neuvedení této hodnoty při sestavování Docker image.

Zde je uveden zdrojový kód 18 s obsahem souboru .env (skutečné hodnoty jsou nahrazeny):

```
QR_CODE_BASE_URL=https://server.cz
VAADIN_OFFLINE_KEY={"proKey":"klic","username":"email"}
```

Zdrojový kód 18 - obsah souboru .env [zdroj vlastní]

Proměnná QR_CODE_BASE_URL slouží ke konfiguraci základní URL adresy serveru, která se používá pro nastavení adresy generovaných QR kódu. Proměnná VAADIN_OFFLINE_KEY se používá k předání klíče potřebného k předání produkční licence do výsledného produkčního buildu aplikace.

Níže, ve zdrojovém kódu zdrojový kód 19, je představen zdrojový kód docker-compose.yml:

```
version: '3.8'
services:
  sprava-revizi-production:
    image: sprava-revizi-production
    build: .
    ports:
      - "8080:8080"
    env_file:
      - .env
```

Zdrojový kód 19 - docker-compose.yml [zdroj vlastní]

Jedná se o jednoduchý docker-compose.yaml, ve kterém je pouze popsáno vytvoření kontejner sprava-revizi-production z Dockerfile umístěného na stejné úrovni jako je tento soubor, prováže hostitelský port 8080 s portem kontejneru 8080 a do proměnného prostředí systému převezme hodnoty ze souboru .env.

ZÁVĚR

Tato diplomová práce reaguje na rostoucí potřebu digitalizace v oblasti správy revizí vyhrazených elektrických zařízení. S ohledem na zvyšující se nároky kladené na revizní techniky a majitele spotřebičů byl navržen a implementován softwarový systém, který usnadňuje správu revizí, zefektivňuje evidenci elektrických zařízení a zajišťuje přehled nad provedenými revizemi, spotřebiči a evidovanými klienty.

V teoretické části práce byly zanalyzovány existující systémy pro správu revizí, zhodnoceny jejich funkcionality a popsány technologie využité při vývoji řešení. Na tuto část navazovala analýza požadavků, návrh business procesů a uživatelského rozhraní, stejně jako návrh datového a databázového modelu, doplněný o komunikační schéma systému.

Praktická část se zaměřila na samotnou implementaci aplikace, popis jednotlivých pohledů v aplikaci a popis formulářů. Výsledný systém poskytuje funkcionality jako je správa revizí dle klientů, správa spotřebičů, správa dokumentů revizí a spotřebičů včetně možnosti omezení jejich viditelnosti jen pro klienty, generování QR kódů pro jednotlivé spotřebiče, tvorba revizních protokolů, emailové upozornění na blížící se termíny revizí jak pro technika, tak i pro výčet uvedených adres u revize, sledování revizí pomocí dashboardu, správa uživatelů, kalendář revizí. Závěrečná část práce se věnovala testování a nasazení aplikace. Výsledný systém omezuje oprávnění uživatelů na základě rolí.

Vyvinutá aplikace představuje efektivní nástroj pro správu revizí, který přináší přidanou hodnotu jak revizním technikům, tak koncovým uživatelům. Zároveň vytváří prostor pro další rozvoj systému v souladu s aktuálními i budoucími požadavky praxe.

POUŽITÁ LITERATURA

[1] *What is Java technology and why do I need it?* [online]. Oracle, [cit. 8. března 2025]. Dostupné z: https://www.java.com/en/download/help/whatis_java.html

[2] What is java?-beginner's guide to java: Microsoft azure. *-Beginner's Guide to Java | Microsoft Azure* [online]. Microsoft Azure, [cit. 8. března 2025]. Dostupné z: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-java-programming-language

[3] *Spring Framework Reference Documentation* [online]. Version 3.2.x. Pivotal Software, Inc., [cit. 8. března 2025]. Dostupné z: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html

[4] *Difference Between Spring and Spring Boot* [online]. GeeksforGeeks, Sanchhaya Education Private Limited, 6. září 2024 [cit. 8. března 2025]. Dostupné z: https://www.geeksforgeeks.org/difference-between-spring-and-spring-boot/

[5] *Spring Boot* [online]. Pivotal Software, ©2025 [cit. 8. 3. 2025]. Dostupné z: https://spring.io/projects/spring-boot

[6] *What is Maven?* [online]. The Apache Software Foundation, ©2025, 6. března 2025 [cit. 8. 3. 2025]. Dostupné z: https://maven.apache.org/what-is-maven.html

[7] *Maven Features* [online]. The Apache Software Foundation, ©2025, 6. března 2025 [cit. 8. 3. 2025]. Dostupné z: https://maven.apache.org/maven-features.html

[8] *What is Flow?* [online]. Vaadin, 19. prosince 2024 [cit. 8. 3. 2025]. Dostupné z: https://vaadin.com/docs/latest/flow/what-is-flow

[9] *Routing and Navigation in Vaadin Flow* [online]. Vaadin, [cit. 8. 3. 2025]. Dostupné z: https://vaadin.com/docs/latest/flow/routing/navigation

[10] Susnjara, S.; Smalley, I. *What is Docker?* [online]. IBM, 6. 6. 2024 [cit. 9. 3. 2025]. Dostupné z: https://www.ibm.com/think/topics/docker

[11] *Advanced Security Architecture in Vaadin Flow* [online]. Vaadin, 19. prosince 2024 [cit. 9. 3. 2025]. Dostupné z: https://vaadin.com/docs/latest/flow/security/advanced-topics/architecture

[12] *Security Vulnerabilities in Vaadin Flow* [online]. Vaadin, 19. prosince 2024 [cit. 9. 3. 2025]. Dostupné z: https://vaadin.com/docs/latest/flow/security/advanced-topics/vulnerabilities

[13] Erickson, J. *MySQL: Understanding What It Is and How It's Used* [online]. Oracle, 29. 8. 2024 [cit. 9. 3. 2025]. Dostupné z: https://www.oracle.com/mysql/what-is-mysql/

[14] *What is Amazon S3* [online]. © 2024, Amazon Web Services, Inc. [cit. 9. 3. 2025]. Dostupné z: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

[15] *Getting Started: Sending Email* [online]. Mailgun, [cit. 9. 3. 2025]. Dostupné z: https://documentation.mailgun.com/docs/mailgun/user-manual/get-started/

[16] GRADY, Jeffrey O. *System Requirements Analysis*. 2nd Edition. Amsterdam: Elsevier, 2013, 834 s. ISBN 9780124171077.

[17] Requirement Sources [online]. Sparx Systems Pty Ltd., [cit. 10. 3. 2025]. Dostupné z: https://sparxsystems.com/enterprise_architect_user_guide/17.0/modeling_domains/requirement_sources.html

[18] McQuillan, R. *How to Create a Data Model in 9 Steps* [online]. Budibase, 22. 5. 2024 [cit. 13. 3. 2025]. Dostupné z: https://budibase.com/blog/data/how-to-create-a-data-model/

[19] DOGLIO F., *What is a Business Process and Why Should You Care* [online]. Camunda, 29. 2. 2024 [cit. 15. 3. 2025]. Dostupné z: https://camunda.com/blog/2024/02/what-is-a-business-process-why-important/

[20] Pressman, R. S., *Software Engineering: A Practitioner's Approach*. 7. vyd. New York: McGraw-Hill Education, 2009. ISBN 978-0073375977.

[21] Francis O., *Inspection Management System* [online]. © 2025 Lumiform, 5. 11. 2024 [cit. 17. 3. 2025]. Dostupné z: https://lumiformapp.com/guides/inspection-management-system

[22] Dolan, M. *When Should You Use an Email API?* [online]. Mailgun, [cit. 9. 3. 2025]. Dostupné z: https://www.mailgun.com/blog/email/when-should-you-use-email-api/

[23] *Revelo* [online]. Studio Datales, Evžen Šrámek [cit. 8. 3. 2025]. Dostupné z: https://www.datales.cz/revelo.html

[24] *Revelo - Nápověda* [online]. Studio Datales, Evžen Šrámek [cit. 8. 3. 2025]. Dostupné z: https://datales.cz/help/revelo/

[25] *Hlídání revizí, údržby a kontrol majetku* [online]. Aptien.com, [cit. 18. 3. 2025]. Dostupné z: https://aptien.com/cs/hlidani-revizi-a-kontrol

[26] *Usnadněte si řízení provozu a administrativy firmy* [online]. Aptien, [cit. 8. 3. 2025]. Dostupné z: https://aptien.com/cs/

[27] *Systém pro REVIZE* [online]. Agionet s.r.o., [cit. 18. 3. 2025]. Dostupné z: https://www.facman.cz/

[28] *Advanced Testing Methods* [online]. Vaadin, 19. prosince 2024 [cit. 7. 4. 2025]. Dostupné z: https://vaadin.com/docs/latest/flow/testing/end-to-end/advanced-concepts

[29] *Deploying Spring Boot-Based Applications,* Vaadin, 19. prosince 2024 [online]. [cit. 13. 4. 2025]. Dostupné z: https://vaadin.com/docs/latest/flow/production

[30] LÉV, Michael. *Návrh a implementace webové aplikace pro správu revizí*. Online. Diplomová práce. Pardubice: Univerzita Pardubice, Fakulta elektrotechniky a informatiky. 2023. Dostupné z: https://theses.cz/id/u7k01s/