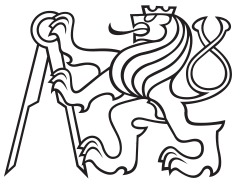
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Systém pro správu revizí a údržby průmyslových kotelen

Rostislav Kvarda

Vedoucí: Ing. Martin Ledvinka, Ph.D.
Studijní program: Softwarové inženýrství a technologie
Specializace: Enterprise systémy
Květen 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kvarda** Jméno: **Rostislav** Osobní číslo: **507416**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

Specializace: **Enterprise systémy**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Systém pro správu revizí a údržby průmyslových kotelen

Název bakalářské práce anglicky:

System for Managing Industrial Boiler Revisions and Maintenance

Pokyny pro vypracování:

1. Seznamte se se současným stavem správy revizí a údržby průmyslových kotelen v cílové společnosti a proveďte sběr požadavků na systém nový.
2. Prozkoumejte možná existující řešení a srovnejte jejich vhodnost/složitost úprav pro účely cílového systému. Srovnejte vhodné technologie pro případnou implementaci systému nového.
3. Navrhněte systém správy revizí a údržby tak, aby vyhovoval sesbíraným požadavkům.
4. Naimplementujte navržený systém. V úvahu vezměte též prostředí, ve kterém bude nasazen.
5. Ověřte použitelnost systému uživatelským testováním s alespoň třemi uživateli.

Seznam doporučené literatury:

[1] Fielding R.: Architectural Styles and the Design of Network-based Software Architectures, 2000.
[2] Fowler M.: Patterns of Enterprise Application Architecture, 2002.
[3] Walls C.: Spring Boot in Action, 2015.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Martin Ledvinka, Ph.D. skupina znalostních softwarových systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **25.01.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

Ing. Martin Ledvinka, Ph.D.		prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Rád bych zde primárně poděkoval mému vedoucímu práce Ing. Martinu Ledvinkovi za odborný dohled, cenné rady a podporu při psaní této práce.

Dále chci poděkovat všem zástupcům společnosti, kteří ochotně věnovali čas při konzultacích k mé bakalářské práci.

Prohlášení

Tímto prohlašuji, že jsem tuto bakalářskou práci, se všemi jejími přílohami, vypracoval samostatně. Zároveň jsem uvedl veškeré zdroje, ze kterých jsem čerpal. Při psaní kódu aplikace jsem využíval nástroj umělé inteligence GitHub Copilot.

V Praze dne 25. května 2024

Abstrakt

Cílem této bakalářské práce je zpracovat analýzu, návrh a následně implementovat informační systém pro teplárenskou společnost provozující průmyslové kotelny. Práce obsahuje podrobnou analýzu současné situace ve společnosti, včetně identifikace potřeb a požadavků na informační systém. Následně navržený a vyhotovený systém poskytuje nástroj pro správu kotelen, revizí, pracovníků a jejich kvalifikací.

Klíčová slova: Informační systém, Spring Boot, Java, GraphQL, PostgreSQL, Svelte, Docker

Vedoucí: Ing. Martin Ledvinka, Ph.D.

Abstract

The aim of this bachelor thesis is to analyse, design and subsequently implement an information system for a company operating industrial boiler rooms. The thesis contains a detailed analysis of the current situation in the company, including the identification of needs and requirements for the information system. The subsequently designed and implemented system provides a tool for managing boiler rooms, revisions, personnel and their qualifications.

Keywords: Information system, Spring Boot, Java, GraphQL, PostreSQL, Svelte, Docker

Title translation: System for Managing Industrial Boiler Revisions and Maintenance

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Průmyslové kotelny, vytápějící bytové domy a další větší nemovitosti, podléhají pravidelným revizím a údržbám. Společnosti, spravující chod těchto kotelen, jsou povinny zajistit, aby revize probíhaly v zákonem daných intervalech a zároveň, že obsluha kotelen má příslušné kvalifikace k obsluze těchto zařízení. Případná chyba ve správě těchto náležitostí a následné zanedbání provedení revizí či přidělení nekvalifikované obsluhy vede k legislativním sankcím vůči společnostem provozujícím kotelny [1, 2, 3, 4, 5].

V dnešní době sahají společnosti po takových informačních systémech, které jim poskytnou důležité nástroje pro sběr a zpracování firemních dat. Tato data poté informační systémy prezentují uživateli a poskytují mu tak cenné informace, na základě kterých dále rozhoduje. Tím, že informační systémy automatizují zpracování dat, dokáží předejít chybám ve zpracování a mohou i celkový proces zrychlit [6]. Jedna teplárenská společnost se tedy rozhodla pomocí informačního systému zautomatizovat správu kotelen, jejich revizí a kvalifikací obsluhy a tím primárně odstranit chyby a následné sankce vyplývající ze stávajícího manuálního zpracování dat.

1.1 Motivace

Po setkáních s představiteli zmíněné společnosti bylo zjištěno, že v současné době je správa revizí a obsluhy kotelen realizována pomocí dat v tabulkách Excel a na sdíleném úložišti, což ztěžuje vyhledávání a neposkytuje žádné kontrolní či notifikační mechanismy. Zdrojové dokumenty, jako jsou revizní protokoly a certifikáty osvědčení obsluhy, se ukládají v elektronické podobě na sdíleném úložišti NAS (Network Attached Storage)[1]. Dohledat tyto dokumenty lze pouze na základě jmenné konvence, jejíž nedodržení vede prakticky k nedohledatelnosti potřebného dokumentu.

Zástupci společnosti potvrdili, že tento způsob hospodaření s daty nutnými ke správě kotelen je časově náročný a v celkovém procesu zpracování dat dochází k chybám. To způsobuje, že pracovníci společnosti tráví nadbytečný čas vyhledáváním informací a další administrativní činností. Podle představitelů

[1] NAS - Network Attached Storage je typ úložiště datových souborů, které je připojeno k síti a umožňuje uživatelům sdílet a ukládat data z centrálního úložiště [7].

společnosti ke zmíněným chybám ve zpracování dochází několikrát měsíčně. V lepším případě je odpovědná osoba upozorněna kolegy. Avšak čím déle chyba uniká pozornosti, tím se zvyšuje pravděpodobnost, že chybu odhalí až státní orgán a příslušně ji potrestá.

Dalším specifickým problémem momentálního stavu je podle představitelů společnosti situace, kdy účetní zaúčtovává přijaté faktury za již vykonané revize. Účetní musí nejprve dohledat, k jakým revizím má hledat faktury. Avšak účetní nemá přístup do tabulek správy kotelen, tudíž musí požádat odpovědnou osobu, aby jí sdělila, jaké revize může zaúčtovat a poté pokračovat k hledání odpovídající faktury v účetním systému. Nový systém, který by účetní poskytl přehled revizí, za které byly dodány faktury, by velmi urychlil celý proces.

Představitelé společnosti a autor této práce se shodují, že informační systém má potenciál dramaticky snížit chybovost procesu správy revizí a obsluhy kotelen. Zároveň by systém měl zrychlit činnosti spjaté se zmíněnou správou revizí a obsluhy kotelen, jelikož poskytne jednotný a přehledný přístup ke všem potřebným informacím. Ušetřený čas zefektivní práci zaměstnanců na správě kotelen. Společnost dále předpokládá snížení nákladů na provoz ve zmiňované oblasti.

1.2 Cíle práce

Cílem práce je navrhnout a vytvořit nový informační systém pro vybranou společnost, který nahradí současné řešení založené na tabulkách Excel a zároveň umožní uživatelům efektivně spravovat plán revizí a údržeb. Jádrem navrhovaného systému bude uživatelsky přívětivý nástroj, který odstraní chyby ve zpracování dat správy kotelen, akcentované zástupci společnosti a současně ušetří čas zaměstnanců společnosti.

Struktura práce

Kapitola 2 se zabývá analýzou současného stavu společnosti a sběrem požadavků na funkcionality systému. Zároveň se věnuje analýze již existující řešení na trhu a jejich srovnání. V kapitole 3 je zpracovaný návrh informačního systému, který splňuje funkcionality, požadované představiteli společnosti. Dále v kapitole 4 popisuji způsob a technologie vybrané pro vytvoření nového systému. Testování vytvořené aplikace je následně popsáno v kapitole 5.

Kapitola 2
Analýza

V rámci této kapitoly je zpracována analýza vytvářeného informačního systému na základě konzultací s představiteli společnosti. Analýza systému je tvořena tak, aby co nejpřesněji dokumentovala požadavky společnosti na výsledné řešení.

Kapitola obsahuje stručný popis domény a analýzu cílů, kterých chce společnost tímto projektem dosáhnout. Dále se zde věnuji identifikaci rolí a požadovaných funkcionalit, které mají být implementovány. Představy a požadavky společnosti jsou následně formálně zaneseny a zpracovány do diagramů případů užití, které poskytují jednotný přehled fungování budoucího systému. Dále jsou zde zobrazeny stavové diagramy důležitých entit v systému. Posledním bodem kapitoly je průzkum existujících řešení k definovaným požadavkům.

2.1 Popis domény

Kotelny společnosti se skládají z jednotlivých provozních jednotek, které jsou funkčně provázány a tvoří nedílný funkční celek. Zejména se jedná o vyhrazené technické zařízení u kterého legislativa předepisuje provádění pravidelných kontrol, zkoušek a revizí. O těchto činnostech se pořizují písemné protokoly, které je nutno archivovat po dobu několika let. Stejně tak i pracovníci, kteří zajišťují provoz kotelen musí mít pro tuto práci příslušné zkoušky a oprávnění předepsaná pro daný typ zařízení. Tyto záležitosti jsou kontrolovány Státní energetickou inspekcí, Energetickým regulačním úřadem a Inspektorátem práce.

Kromě technické stránky je nezbytně nutné vést i evidenci nákladů spojených se zajištěním provozu kotelny. Výrobní účetní proto potřebuje vědět, zda již byla provedena zkouška či revize, zda společnost obdržela daňový doklad za tento výkon a následně na jaké účetní středisko má být náklad zaúčtován.

2.2 Cíle společnosti

Ze schůzek se zástupci společnosti vyplynuly na základě hlavních požadavků a nedostatků stávajícího řešení následující primární cíle, kterých chce společnost za pomoci informačního systému dosáhnout:

- Snížení počtu propadlých revizí kotelen v důsledku opomenutí.
- Snížení počtu propadlých kvalifikací obsluhy kotelen v důsledku opomenutí.
- Eliminace legislativních postihů v důsledku chybné správy revizí a odborné kvalifikace obsluhy kotelen.
- Zefektivnění práce zaměstnanců, týkající se správy kotelen, jejich revizí a kvalifikací obsluhy.

2.3 Požadavky systému

Požadavky na systém vychází z konzultací s představiteli společnosti a požadovaných cílů, kterých má systém dosáhnout.

2.3.1 Role

Na základě analýzy současného stavu byly identifikovány role, které se zúčastní procesu správy revizí a obsluhy kotelen. S představiteli společnosti bylo dohodnuto, že role současného stavu budou ponechány, avšak jejich jednotlivé činnosti se interpretují do funkcionalit nadcházejícího systému. Jednotlivé role jsou popsány podle toho, jaký budou mít účel a s jakými funkcionalitami budou interagovat.

Manažer

Role manažer je nejvyšší rolí v systému a má největší práva. Má přístup ke všem kotelnám a uživatelům evidovaným v systému. Této roli je určen manažerský dashboard, ve kterém vidí informace ke všem kotelnám, technikům a obsluze. Informace zahrnují výčet všech úkonů, které jsou potřeba pro danou kotelnu udělat v určitém období. Úkonem může být neplatná revize kotelny, končící platnost kvalifikace technika, atd..

Další výsadou role je zakládání nových kotelen a nových uživatelských účtů systému. Následně manažer přiřadí kotelnu technikovi provozu, který za ni bude odpovědný a je určen k její správě.

Technik provozu

Technik má na starost přidělené kotelny a je zodpovědný za jejich správu v systému. V rámci jemu přiřazené kotelny vytváří a dále spravuje provozní

jednotky, ze kterých se kotelna skládá. Dále může v systému zadávat revize a úkoly pro jednotlivé kotelny nebo vybrat a přiřadit obsluhu ke svým kotelnám. Technik tedy může zobrazit všechny osoby typu Obsluha vedené v systému a také jim může přiřazovat potřebné kvalifikace. Avšak nevidí další techniky provozu.

V kotelně může nastat situace, že technik provozu sám sebe přiřadí jako obsluhu. Technik tedy může být přiřazen jako obsluha provozní jednotky, avšak pouze v rámci té kotelny, za kterou je jako technik provozu zodpovědný. Musí však sám mít příslušné kvalifikace, aby mohl danou provozní jednotku kotelny obsluhovat. Technik má k dispozici také dashboard podobný manažerskému, ale omezený pouze na jemu přiřazené kotelny.

Obsluha

Obsluha kotelny je v systému pouze evidována s příslušnými kvalifikacemi, ale nemá do systému přístup a nebude mít tedy uživatelský účet. Obsluha pracuje na kotelně a stará se o její provoz. Je přímo podřízena technikovi provozu. Jeden člověk obsluhy může být přiřazen k více kotelnám, na kterých bude pracovat.

Kvalifikace obsluhy jsou kontrolovány a při končící platnosti systém upozorní technika provozu, že daná obsluha musí kvalifikaci obnovit. Pokud kvalifikace není obnovena, obsluha nesmí na provozních jednotkách, kde je tato kvalifikace vyžadována, pracovat.

Účetní

Účetní v systému potřebuje zobrazit, za jaké revize již přišla faktura. Zjistí tak, jaké faktury k jakým revizím má přiřadit a následně je zpracovává v účetním systému.

V systému tedy s ničím nemanipuluje, jen zobrazuje informace o fakturaci revizí.

2.3.2 Funkční požadavky

Požadavky na funkcionalitu systému vychází z konzultací se zástupci společnosti. Jednotlivé funkční požadavky jsou vypsány níže a slouží jako podklad pro výčet funkcionalit, které má navrhované řešení poskytovat.

FRQ-1 Přístup do systému bude zabezpečen uživatelskými účty, které budou mít roli manažer, technik provozu nebo účetní.

FRQ-2 Systém poskytne evidenci kotelen s podrobnostmi o nich.

FRQ-3 Systém umožní provoznímu technikovi přidat ke kotelně provozní jednotky, ze kterých se skládá.

FRQ-4 Systém umožní provoznímu technikovi nastavit kotelně a provozní jednotce požadované revize.

FRQ-5 Systém poskytne evidenci provozních techniků a obsluhy.

FRQ-6 Systém bude ukládat získané kvalifikace provozních technika a obsluhy, společně s PDF skenem této kvalifikace.

FRQ-7 Systém umožní manažerovi přiřadit technika provozu ke kotelně. Ten je poté za konkrétní kotelnu zodpovědný.

FRQ-8 Systém umožní technikovi provozu přiřadit obsluhu na jednotlivé kotelny, za které je daný provozní technik zodpovědný.

FRQ-9 Systém umožní manažerovi vylistovat všechny kotelny, za které zodpovídá daný technik provozu.

FRQ-10 Systém upozorní, pokud je kotelně přiřazen technik provozu nebo obsluha, která nesplňuje předepsané kvalifikace.

FRQ-11 Systém upozorní, pokud se blíží konec platnosti potřebné kvalifikace provozního technika či obsluhy.

FRQ-12 Systém umožní manažerovi přidat úkol pro kotelnu. Úkol se skládá z nadpisu, popisu a deadline. Úkol má indikaci, zda byl splněn. Za úkol je zodpovědná ta osoba, která je zodpovědná za danou kotelnu.

FRQ-13 Systém umožní provoznímu technikovi provozu ozmačit úkol jako splněný.

FRQ-14 Systém umožní manažerovi a provoznímu technikovi vylistovat všechny úkoly podle splněnosti a příslušnosti ke kotelně.

FRQ-15 Systém umožní ukládat jednotlivé revize kotelny či provozní jednotky s protokolem ve formátu PDF.

FRQ-16 Systém umožní provoznímu technikovi zadat informaci, zda za revizi již přišla faktura.

FRQ-17 Systém umožní provoznímu technikovi vylistovat všechny úkony, které jsou potřeba udělat v následujícím časovém úseku pro jemu svěřené kotelny. Např. jaké revize se musí udělat, jaké kvalifikace je nutno obnovit, atd.

FRQ-18 Systém umožní účetní zobrazit seznam a revizí podle toho, zda již dorazila faktura.

FRQ-19 Systém umožní managementu a technikovi vytvořit novou osobu obsluhy.

FRQ-20 Přístup k funkcím systému bude omezen podle oprávnění uživatelské role, která daný uživatel má.

2.3.3 Nefunkční požadavky

Nefunkční požadavky popisují kritéria na kvalitu systému. Popisují, jaké má mít systém vlastnosti [8].

Následující kvalitativní požadavky opět vychází ze schůzek se zástupci společnosti:

NFRQ-1 Systém je schopen obsloužit alespoň 20 uživatelů současně bez ztráty výkonu.

NFRQ-2 Systém je dostupný 99% času po celý rok.

NFRQ-3 Systém zajistí, že uživatelská data jsou chráněna před neoprávněným přístupem (GDPR).

NFRQ-4 Systém poskytuje uživatelské rozhraní v českém jazyce.

NFRQ-5 Systém bude nasazen ve virtualizačním nástroji Docker.

2.4 Případy užití

V této kapitole jsou zpracovány případy užití[1]. Plně pokrývají funkční požadavky systému a zohledňují identifikované role, jež byly konzultovány se zástupci společnosti.

Aktéři

Obrázek 2.1 zobrazuje aktéry, kteří se vyskytují v systému a interagují s jeho funkcionalitami. Jednotliví aktéři odpovídají uživatelským rolím popsaným v sekci 2.3.1.

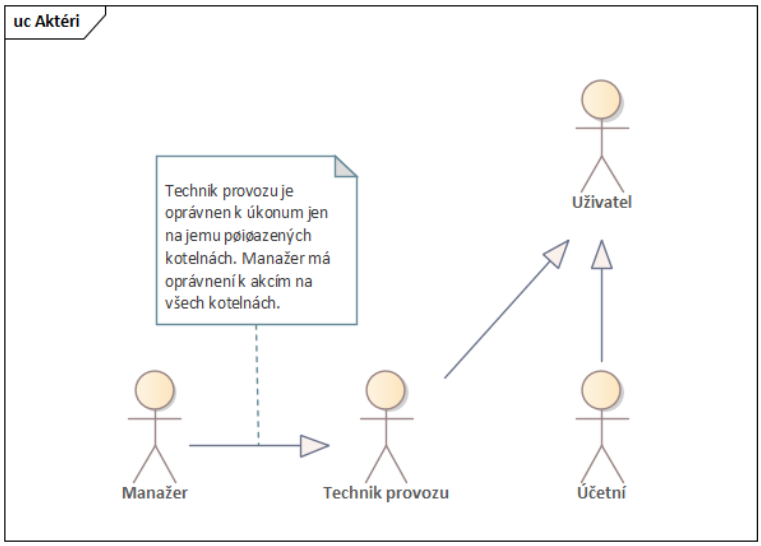
V obrázku je zanesena dědičnost funkcionalit jednotlivých rolí. Specifická dědičnost nastává mezi aktérem „Manažer“ a „Technik provozu“. Technik provozu je oprávněn k úkonům jen na jemu přiřazených kotelnách. Manažer má oprávnění k akcím na všech kotelnách vedených v systému.

2.4.1 Diagramy případů užití

Případy užití jsou zpracovány do diagramů případů užití, které představují celistvý grafický soupis funkcionalit, jež systém poskytuje. Vyobrazují, ke kterým konkrétním funkcionalitám mají jednotliví aktéři, uživatelé systému, přístup a mohou je využívat [8, 10].

Jednotlivé diagramy případů užití jsou pro větší přehlednost členěny do tematických celků. V této kapitole je vyobrazen pouze diagram ke Správě kotelen. Ostatní diagramy případů užití se nachází v příloze A.

[1] Případy užití je metoda pro identifikaci a organizaci požadavků na systém z pohledu uživatele. Jedná se o popis, k jakým funkcionalitám má uživatel přístup a jakým způsobem s funkcionalitami interaguje [9].



Obrázek 2.1: Katalog aktérů a dědičnosti

Správa uživatelů

Diagram v příloze na obrázku A.1 popisuje funkcionality spojené se správou uživatelských účtů. Registrace do systému je v režii aktéra „Manažer“, který má na starost vytváření uživatelských účtů v systému. Samostatná registrace uživatele v systému umožněna není.

Správa obsluhy provozních jednotek

Na obrázku A.2 v příloze je diagram popisující funkcionality ohledně obsluhy jednotlivých provozních jednotek. Dále zobrazuje funkcionality pro správu jednotlivých kvalifikací a jejich ukládání k jednotlivým pracovníkům. Kvalifikace může ukládat i k sobě samému.

Správa revizí

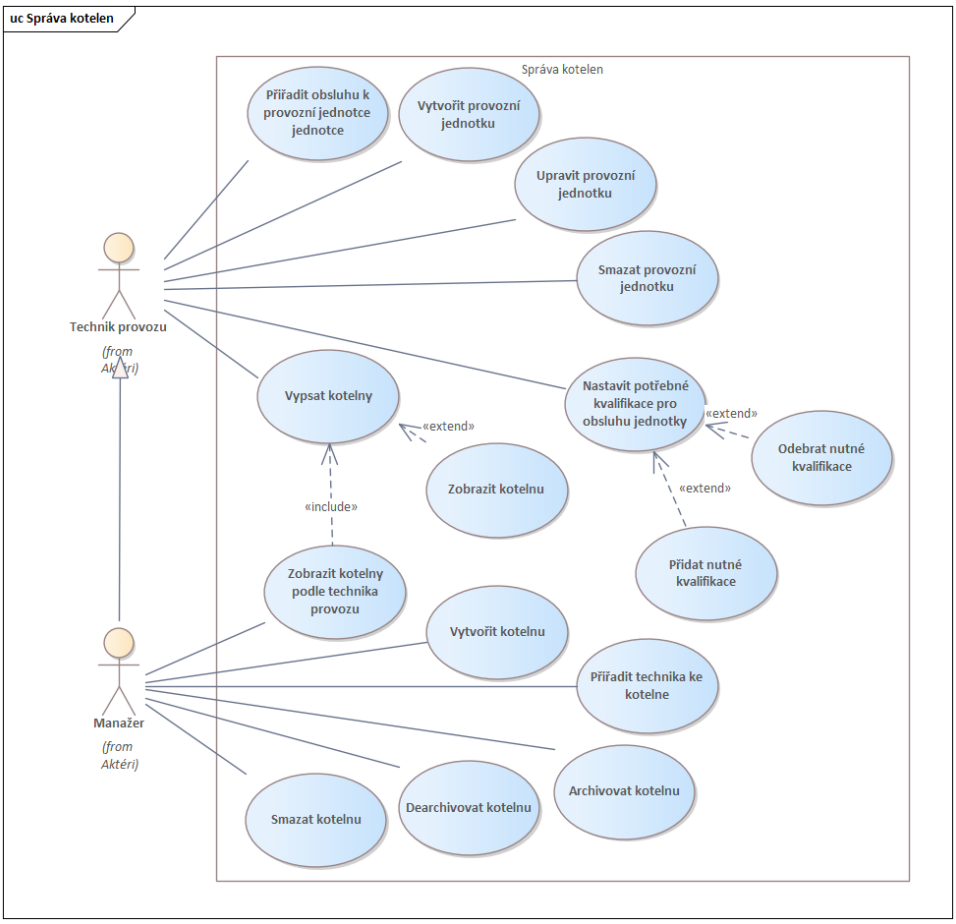
Diagram v příloze na obrázku A.3 zobrazuje funkcionality pokrývající revize a úkoly kotelny. Aktér „Účetní“ má přístup k funkcionalitě, která mu umožňuje zobrazit revize podle toho, zda již byly fakturovány.

Správa kotelen

Na obrázku 2.2 jsou zobrazeny funkcionality pokrývající správu kotelen. Kotelna je vytvořena aktérem „Manažer“, který jí přiřadí odpovědného technika provozu. Aktér „Technik provozu“ dále samostatně spravuje jemu svěřené kotelny. Technik rovněž spravuje životní cyklus provozních jednotek, ze kterých se kotelna skládá.

Manažer je oprávněn k funkcionalitám aktéra „Technik provozu“, ale s tím rozdílem, že Manažer má oprávnění spravovat všechny kotelny, avšak Technik

provozu má oprávnění jen pro ty kotelny, jež mu jsou svěřeny.



Obrázek 2.2: UML Diagram případů užití pro správu kotelen

Správa účetních středisek

Poslední diagram na obrázku A.4 popisuje funkcionality spojené se správou účetních středisek. Účetní střediska představují administrativní členění a shlukují kotelny do administrativních celků.

2.4.2 Scénáře případů užití

Scénáře případů užití jsou konkrétní sekvence kroků popisujících interakci mezi aktéry a systémem pro dosažení požadovaného cíle v rámci případů užití [11].

V této sekci se nachází pouze jeden ukázkový scénář pro případ užití „Přiřadit technika ke kotelně“. Další scénáře případů užití se nachází v příloze B.

Scénář „Přiřadit technika ke kotelně“

Popis: Umožňuje ke kotelně přiřadit technika, který bude za danou kotelnu zodpovědný.
Předpoklad:

- Uživatel je přihlášen do systému a má oprávnění spravovat danou kotelnu.
- Uživatel se nachází na stránce příslušné kotelny.

Hlavní scénář

1. Uživatel u příslušné kotelny zvolí možnost přidání technika ke kotelně.
2. Systém zobrazí dialogové okno s nabídkou dostupných techniků.
3. Uživatel vybere požadovaného technika.
4. Systém uloží provozního technika ke kotelně a nastaví se tím danému technikovi oprávnění spravovat kotelnu v systému.

Alternativní scénář

- **2a** Systém nezobrazí požadovaného technika.
- **2a2** Systém nabídne uživateli přesměrování na UC „Vytvořit uživatele“.
- **2a3** Uživatel pokračuje na «extend» UC „Vytvořit uživatele“.

Koncové podmínky:

- Vybraný provozní technik je přiřazen dané kotelně.
- Provozní technik je oprávněn ke správě dané kotelny.

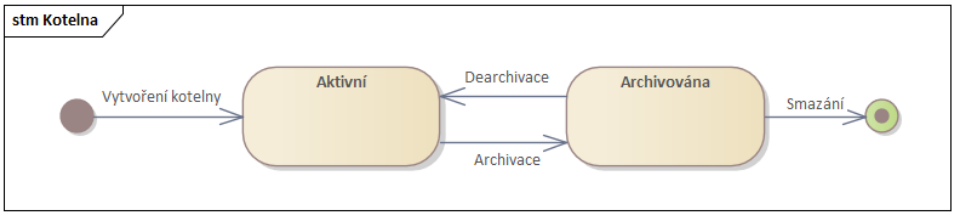
2.5 Stavové diagramy

Na základě jednání se zástupci společnosti byly identifikovány entity „Kotelna“, „Úkol“ a „Revize“, které které v rámci procesu správy kotelen a revizí procházejí určitými stavy. „Úkol“ a „Revize“ mají pouze dva stavy: buď je úkol splněný, nebo ne, a revize je buď fakturovaná, nebo ne. Mezi těmito stavy lze libovolně přecházet bez omezujících podmínek, a tyto entity lze smazat z jakéhokoli stavu. Pro tyto jednoduché případy není nutné detailně ilustrovat logiku přechodů mezi stavy v diagramu. Nicméně entita „Kotelna“ má specifickou podmínku, že může být smazána pouze v určitém stavu. Je tedy vhodné tento životní cyklus entity „Kotelna“ znázornit pomocí stavového diagramu.

Stavový diagram kotelny

Diagram stavů kotelny je na obrázku 2.3. Kotelna se po svém vytvoření může nacházet ve dvou stavech - „Aktivní“ a „Archivována“. Kotelna přechází mezi jednotlivými stavy na základě činnosti uživatele, který volí, v jakém stavu se bude kotelna nacházet. Kotelna ve stavu „Archivována“ nebude oproti stavu „Aktivní“ generovat upozornění o končících datech platnosti revize pro danou kotelnu a nebude se zobrazovat ve standardních výpisech.

Ze stavu „Archivována“ je možné kotelnu úplně smazat ze systému. Toto může nastat v případě, že společnost kotelnu přestane spravovat a uplyne zákonná lhůta, po kterou je nutné držet informace o provozovaných kotelnách [4].



Obrázek 2.3: UML Stavový diagram kotelny zobrazující jednotlivé stavy kotelny a přechody mezi nimi.

2.6 Existující řešení

Informační systémy, které poskytují funkcionality požadované zástupci společnosti ohledně správy kotelen a revizí, se nazývají Facility Management Software. V překladu „Systémy správy majetku“. Jedná se o systémy, které poskytují nástroje pro správu majetku, údržby a revize. Zároveň jsou schopny evidovat dokumentaci a hlídat termíny údržby [12]. Těchto systémů na trhu existuje několik.

EMA+

Jednou z variant řešení je systém EMA+. Z propagačních materiálů se jedná o

> „*Komplexní nástroj pro facility management - efektivní online evidence a správa majetku prostřednictvím efektivního cloudového řešení. Poskytuje data o majetku online, na jednom místě a v reálném čase*“ [13].

Systém poskytuje přístup jednak pomocí webového rozhraní, jednak pomocí mobilní aplikace. Oba přístupy jsou uživatelsky přívětivé a přehledné. Nabízené funkcionality jsou členěny do několika modulů. Např: Evidence majetku, Opakované činnosti, Kalendář a další. Funkce systému EMA+ odpovídají

požadavkům společnosti, avšak jsou moduly, které by společnost nevyužila např. BIM[2] vizualizace, Katastr nemovitostí a Investiční žádanky.

Právě kvůli vysokému počtu nevyužitých modulů tento systém není vhodný. Nevyužité moduly způsobí nadbytečnou komplexitu a obtížnou správu, což vede ke ztrátě přehledu nad funkcionalitou nástroje. Zároveň větší počet modulů zatěžuje školení uživatelů a může vést k nižší akceptaci nové technologie.

Cenu licence nebo bližší informace nebylo možné zjistit bez přímého kontaktování výrobce.

EVROIS

EVROIS je informační systém pro technickou podporu agend provozu. Jedná se o modulární řešení určené k zachycení procesů a dat souvisejících s činnostmi při provozování CNG[3] stanic a energetických zdrojů. Poskytuje funkcionality ohledně evidence zařízení, plány revizí, evidence osob, kalkulace ceny tepla a další. Stejně jako u předchozího zmíněného systému i EVROIS naplňuje požadavky společnosti, avšak má další funkcionality, které by nebyly využívány, jako jsou přehledy parcel, katastrální informace a měřidla energetických zdrojů.

Systém je přístupný pomocí proprietární desktopové aplikace. Mobilní zařízení nejsou nijak podporována. To společnosti nevyhovuje, jelikož chce mít systém dostupný z širokého množství zařízení bez nutnosti instalovat klientské aplikace na koncová zařízení. Desktopová aplikace neposkytuje přívětivé uživatelské rozhraní, které by umožnilo jednoduchý přehled v nástroji.

Další nevýhodou je skutečnost, že EVROIS již není aktivně vyvíjen ani podporován. Zástupci společnosti se obávají, že v případě kritické chyby nebude možné provést opravu a znovu uvést systém do provozu. Tím pádem společnost nemá dostatečnou jistotu ohledně dlouhodobé udržitelnosti a spolehlivosti tohoto řešení. Kromě toho společnost vyjadřuje potřebu možnosti rozšiřování funkcí dle vlastních potřeb. To v případě EVROIS není možné, jelikož neumožňuje přizpůsobení budoucím požadavkům.

Informace o systému EVROIS nejsou veřejně dohledatelné. Zástupce společnosti mi poskytl dokumentaci, ze které jsem mohl čerpat. O systém EVROIS měla společnost dříve zájem, avšak ze zmíněných důvodů jej nezvolila.

Zhodnocení

Nabízené systémy na trhu pro správu kotelen a revizí naplňují požadavky společnosti, avšak vždy se jedná o příliš komplexní řešení. Kombinují požadované funkcionality s kalkulací ceny tepla, odečty měřidel, vyhodnocení účinnosti zdrojů a rozvodů, katastr nemovitostí a další. Tyto nadbytečné funkcionality společnost již řeší prostřednictvím existujícího a osvědčeného řešení.

Podle zástupců společnosti žádný z výrobců systémů nebyl ochoten, či schopen, poskytnout pouze požadovaný dílčí segment týkající se správy kotelen a revizí bez nutnosti přijmout celý komplexní systém. Navíc žádný

[2] Building Information Modeling - digitální model stavby

[3] Compressed Natural Gas - stlačený zemní plyn

z výrobců nebyl schopen zajistit bezproblémovou migraci všech nezbytných dat (smlouvy, účetnictví, agenda fakturačních měřidel, systém vedení cen tepla, personalistika) do nového, souhrnného systému.

Proto se společnost rozhodla vytvořit vlastní systém na míru, který bude plně pokrývat požadavky bez nevyužitých nadbytečných funkcionalit. Tímto způsobem bude schopna dosáhnout optimálního a přesně vymezeného úseku funkcionalit pro správu kotelen a revizí, přizpůsobitelného přesně podle potřeb společnosti.

Kapitola 3
Návrh

Tato kapitola popisuje, jak bude na základě analýzy cílů a požadavků společnosti navržena konkrétní architektura a struktura řešení. Cílem je vytvořit systém, který přesně odpovídá potřebám společnosti a současně je rozšířitelný pro budoucí požadavky.

Kapitola zahrne popis zvolené architektury systému, podle níž bude systém implementován. Dále je v ní zpracován diagram komponent, který zobrazuje jednotlivé části systému a jejich vzájemné vztahy. Následně uvedu, jakým způsobem bude řešení nasazeno a spuštěno do provozu. V této kapitole také představím jednotlivé datové objekty vyskytující se v systému.

3.1 Architektura

Architektura informačního systému je klíčovým prvkem, který ovlivňuje jeho flexibilitu, škálovatelnost a celkovou účinnost. Pro vyvíjené řešení jsem zvolil klient-server architekturu [14, 15]. Tato architektura odpovídá plánovanému užívání systému. Uživatelé budou skrze webového klienta posílat požadavky na server a interagovat tak s uloženými daty. Zároveň tato architektura je v případě budoucího zvýšení nároků na výkon aplikace dobře škálovatelná díky vyvažování zátěže (load balancing), kdy se zátěž distribuuje mezi více serverů [16].

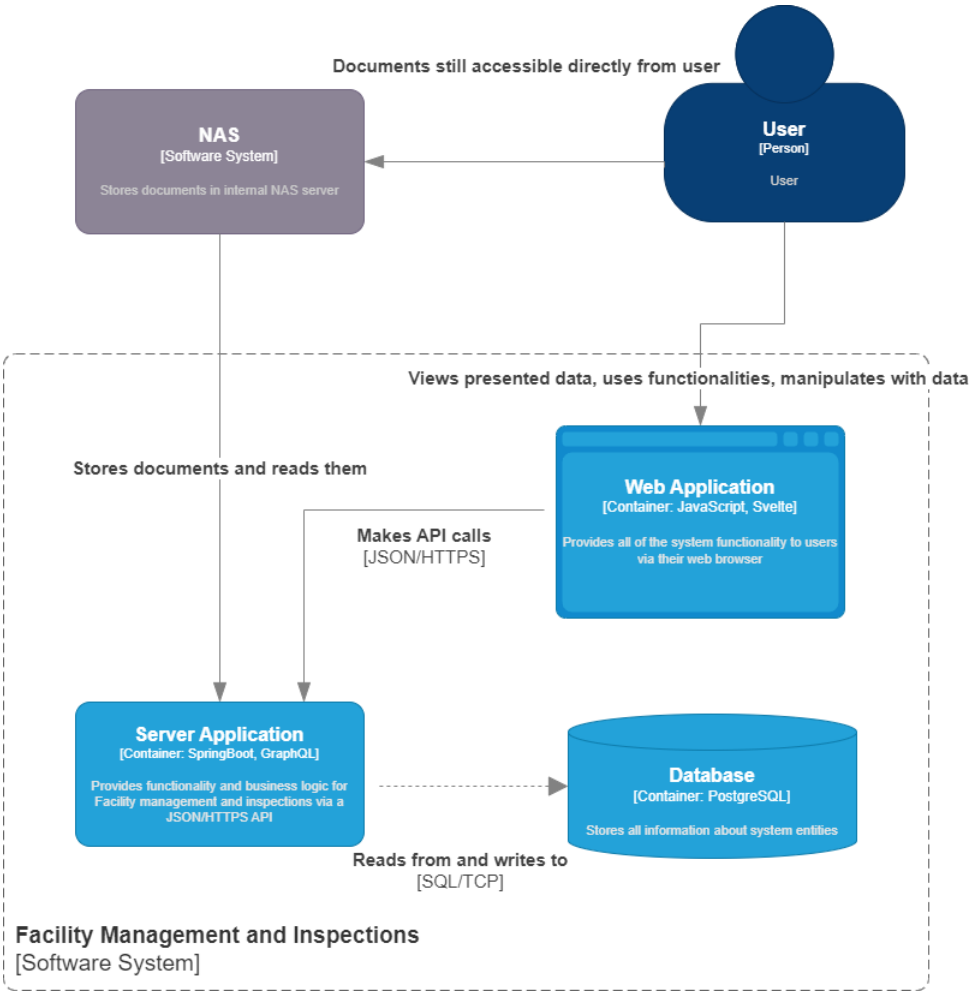
Pro samotný server jsem zvolil vícevrstvou architekturu. Tato architektura dělí aplikaci na jednotlivé vrstvy, z nichž každá zajišťuje specifickou funkcionalitu. Vrstvy mezi sebou spolupracují, ale komunikace je možná pouze mezi komponentami na stejné vrstvě nebo s komponentami přímo sousedící nižší vrstvy [14].

Konkrétně se bude jednat o perzistentní vrstvu starající se o ukládání a přístup k datům uložených v databázi. Dále servisní vrstvu, která implementuje byznys logiku[1]. Zpracování požadavků od klienta bude zajišťovat prezentační vrstva ve formě aplikačního rozhraní GraphQL.

[1]Byznys logika je aplikační logika, která se zabývá hlavním účelem aplikace, co má dělat. Jedná se o jádro aplikace, které obsahuje logiku, výpočty a zpracování dat [17].

3.2 Nasazení systému

Nasazení jednotlivých součástí systému je znázorněno na obrázku 3.1. Vyobrazuje zvolenou Klient-server architekturu. Uživatel přistupuje k systému pomocí klientské webové aplikace, přes kterou informační systém poskytuje svou funkcionalitu. Webová aplikace na základě akcí uživatelé volá požadavky na aplikační rozhraní serveru. Serverová aplikace poskytuje veškerou funkcionalitu a logiku správy kotelen a revizí. Pro ukládání dat používá server databázi. Dokumenty jsou ukládány a načítány z NAS v interní síti společnosti. Uživateli zůstane možnost přistupovat k souborům dokumentů napřímo.



Obrázek 3.1: Container diagram nasazení dle modelu C4. Zobrazuje jednotlivé části systému a komunikaci mezi nimi.

3.3 Datový model

Datový model představuje strukturu a vztahy mezi daty, která jsou uchovávána v informačním systému [18]. Model je tvořen tak, aby reflektoval požadavky na funkcionalitu systému.

3.3.1 Doménový model

Doménový model popisuje entity v systému a vztahy mezi nimi. Souhrn všech tříd a jejich význam se nachází v tabulce 3.1. Model na obrázku 3.2 zobrazuje kompletní strukturu entit, se kterými systém pracuje, a jejich atributů.

Název	Význam
AccountingCenter	Účetní středisko, pod které spadá daná kotelna
Document	Generická třída pro uchování údajů o dokumentu
Facility	Kotelna vedená v systému
FacilityState	Stav, ve kterém se v systému nachází kotelna či provozní jednotka
Inspection	Konkrétní revize provozní jednotky
InspectionState	Stav, ve kterém se nachází revize
InspectionType	Druh revize nutný pro provozní jednotku
Person	Uživatel v systému
Qualification	Konkrétní kvalifikace obsluhy
QualificationType	Druh kvalifikace pro obsluhu provozní jednotky
Roles	Uživatelské role
Serviceable	Mateřská třída pro Facility a Unit, které evidují revize a přiřazené úkoly.
Task	Úkol zadaný pro danou kotelnu
Unit	Provozní jednotka, ze kterých je složena kotelna
UnitType	Druh provozní jednotky
User	Uživatelský účet s přihlašovacími údaji

Tabulka 3.1: Tabulka výčtu entit v systému se slovním popisem jejich účelu.

Entita Facility se skládá kompozitní vazbou z Unit. V momentě, kdy přestane existovat Facility, přestanou existovat i všechny Unit, ze kterých se skládala. Facility i Unit mají stejného rodiče Serviceable, jelikož oba mají předepsáno, jaké revize musí pravidelně probíhat - InspectionType a

evidují přiřazené úkoly . Facility navíc eviduje jakou kvalifikaci musí mít její obsluha - QualificationType a jaký provozní technik je za ni odpovědný. Adresa u Facility je pro zjednodušení vedena jako pouhý textový řetězec bez členění na dílčí části (město, ulice), jelikož z požadavků společnosti se neočekává filtrování kotelen podle například města nebo jiné části adresy a slouží tak pouze pro evidenci a identifikaci. Systém kontroluje, zda obsluha Facility má platnou kvalifikaci daného typu. Stejným způsobem kontroluje, zda Facility a Unit mají platnou revizi požadovaného typu. Všechny revize a kvalifikace musí být podloženy příslušným dokumentem. Proto dědí od třídy Document, která eviduje platnost dokumentu a cestu k souboru dokumentu.

Unit zahrnuje vlastnost „description“ představující popis dané provozní jednotky. Tento popis a údaje v něm obsažené se mění na základě druhu provozní jednotky - UnitType. Například u druhu „Plynový kotel“ se očekává, že bude mít v popisu tepelný výkon, ale „Oběhové čerpadlo“ bude evidovat nominální průtok vody. Původní myšlenka byla separátně ukládat jednotlivé atributy jednotky daného druhu. Avšak s dynamickým počtem UnitType s velkou variabilitou ukládaných dat v popisu a faktickou nepotřebou s těmito daty pracovat (slouží jen pro evidenci), jsem se po dohodě se zástupci společnosti rozhodl popis nechat pouze jako textový řetězec.

Problém nestrukturovaného textu v popisu jednotky lze vyřešit na straně klientské aplikace. Klientská strana může místo pouhého textového řetězce vytvořit strukturovaný text pomocí JSON nebo XML. Poté podle příslušných značek strukturovaného textu může jednoduše pracovat s konkrétními hodnotami daného druhu provozní jednotky. Toto řešení nevyžaduje další změny na serverové části ani v databázovém úložišti.

Po konzultaci se zástupci společnosti se nepočítá s přidáváním nových hodnot Roles, FacilityState a InspectionState za běhu aplikace. Proto jsem zvolil výčtový typ. Oproti tomu u QualificationType, InspectionType a UnitType se očekává přidávání nových hodnot za běhu aplikace. Zde jsem zvolil přístup výčet hodnot pomocí plnohodnotné třídy.

Pro ukládání osob v systému je použita entita Person. Uživatelský účet osoby s přihlašovacími údaji se ukládá v navazující entitě User, jelikož ne všechny osoby vedené v systému musí mít uživatelský účet.

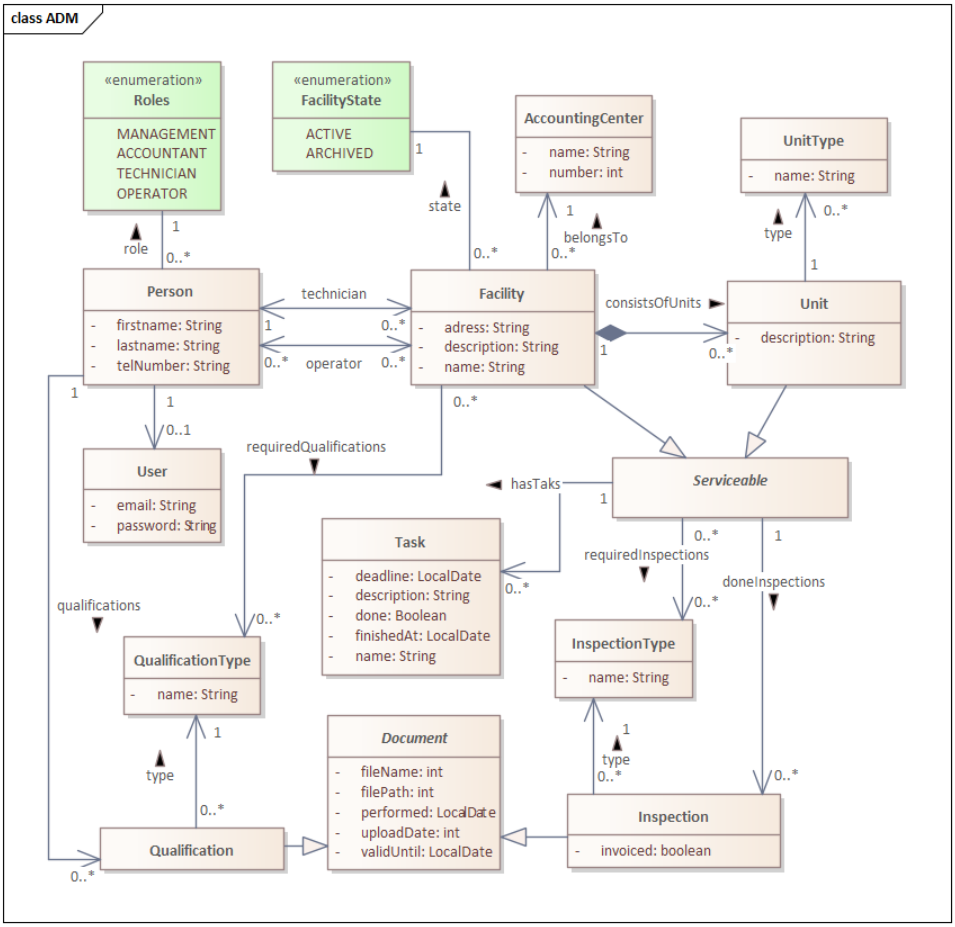
3.3.2 Databázový model

K ukládání dat aplikace bude sloužit relační databáze. Jedná se o druh databáze, kde jsou data organizována do tabulek a vazby mezi nimi jsou založeny na relacích. Pro práci s uloženými daty se používá SQL (Search Query Language [19]), což je standardizovaný jazyk, který umožňuje definovat, manipulovat a řídit data uložená v relačních databázích.

Relační databáze zajišťují zachování integrity a konzistence dat, jelikož splňují ACID[2] podmínky transakcí [21, 22].

Nevýhodou je horší škálovatelnost, jelikož relační databáze jsou určeny

[2]ACID - Atomicity (Atomicita), Consistency (Konzistence), Isolation (Izolace), Durability (Trvanlivost) [20]



Obrázek 3.2: Analytický doménový model UML zobrazující entity vytvářeného systému s příslušnými atributy, vazbami a násobnostmi jednotlivých vazeb mezi entitami.

pro běh na jednom zařízení. V případě složitých vazeb mezi tabulkami se začíná zhoršovat celkový výkon databáze [22]. Pro vyvíjenou aplikaci však výhody zachování integrity dat převyšují nad nevýhodami. Očekávaný počet uživatelů je v nižších desítkách, a tudíž se neočekává razantní zpomalení relační databáze.

3.4 Komponenty systému

Funkcionality informačního systému jsou rozděleny do jednotlivých komponent podle domény, za kterou jsou zodpovědné. Tyto komponenty jsou navrženy tak, aby implementovaly svou logiku nezávisle na ostatních, ale současně využívají služby poskytované jinými komponentami. Každá komponenta je odpovědná za implementaci logiky související s konkrétní doménou.

Na obrázku (3.3) jsou jednotlivé komponenty vyobrazeny společně s vazbami na ostatní komponenty. Jednotlivá propojení ilustrují, jaká rozhraní jsou po-

skytována danou komponentou a zároveň jaká rozhraní ostatních komponent jsou vyžadována.

Soupis a určení jednotlivých komponent je následující:

Person (Osoba) Osoby vedené v systému s jejich přihlašovacími údaji a rolí, požaduje komponentu správy kvalifikací.

Facility (Kotelna) Správa kotelen, požaduje komponentu správy osob, přidělených úkolů a jednotlivé provozní jednotky, ze kterých se kotelna skládá.

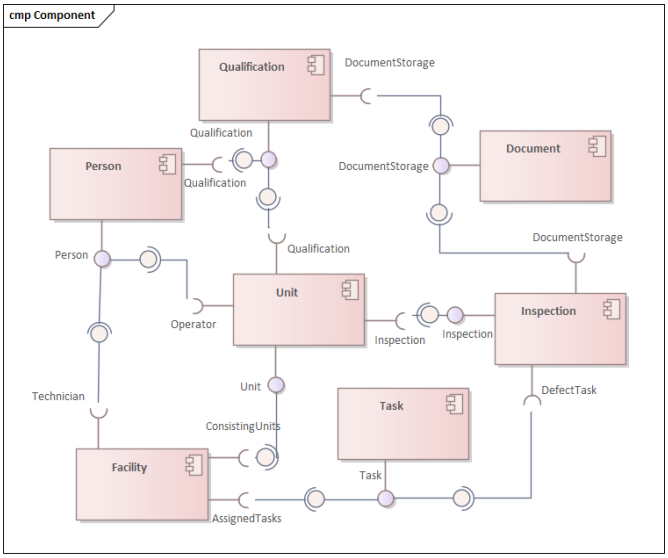
Unit (Provozní jednotka) Správa provozních jednotek, ze kterých je složena kotelna, požaduje funkcionality osob, revizí a kvalifikace.

Task (Úkol) Správa úkolů zadaných pro kotelnu.

Inspection (Revize) Druhy a jednotlivé revize provedené a požadované na provozní jednotce.

Qualification (Kvalifikace) Druhy a jednotlivé kvalifikace nutných pro obsluhu provozních jednotek, požaduje funkcionalitu správy souborů.

Document (Dokument) Poskytuje nástroje pro správa a ukládání souborů.



Obrázek 3.3: UML Diagram komponent popisující jednotlivé komponenty systému a jejich závislosti prostřednictvím poskytovaných a požadovaných rozhraní.

3.5 Prototyp

Návrh uživatelského rozhraní, pomocí něhož bude systém poskytovat své funkcionality, je zpracován do prototypu[3] v nástroji Figma.com [24]. Pomocí tohoto prototypu jsem znázornil důležité funkcionality výsledného systému společně s grafickým návrhem.

Prototyp byl představen zástupcům společnosti, kteří prototyp procházeli a doplňovali podněty pro změny, které vylepšovaly uživatelskou přívětivost a grafický návrh pro potřeby zaměstnanců společnosti. Konzultace prototypu probíhala iterativně a umožnila co nejvíce sjednotit chápání požadovaných funkcionalit vyvíjeného řešení a poskytnout lepší příležitost, aby finální produkt nejlépe odpovídal představám klienta.

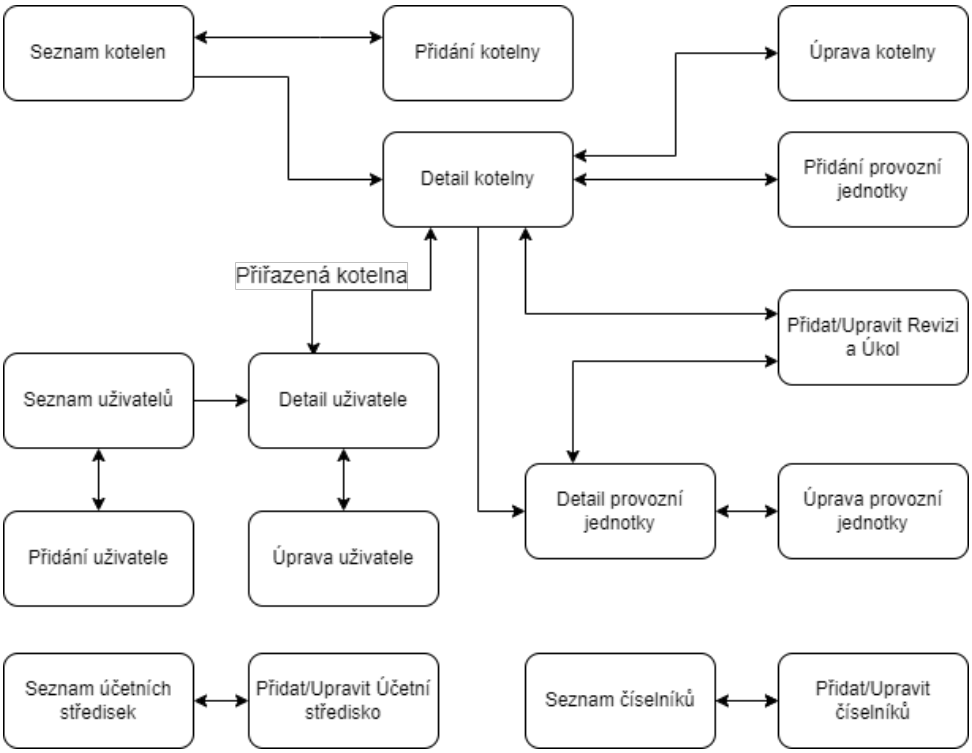
Díky této technice jsem před započetím implementace sesbíral důležitou zpětnou vazbu. Ta snižuje pravděpodobnost rizika spojeného s pozdějšími změnami nebo úpravami v průběhu implementace z důvodu nesrovnalostí se zadáním klienta. Ve fázi návrhu/prototypování je totiž snazší učinit změnu, nežli až v průběhu implementace.

3.5.1 Navigace aplikace

Navigační menu, skrze které se lze navigovat napříč funkcionalitami aplikace, je navrženo formou vždy viditelného postranního panelu umístěném v levé části obrazovky. V tomto panelu se nachází navigační tlačítka pro přechod na stránku událostí, pracovníků, kotelen, účetních středisek, číselníků a seznam fakturovaných revizí. Pravá část obrazovky je rezervována pro zobrazování samotného obsahu.

Navigace z jednotlivých obrazovek na další je schématicky znázorněna na obrázku 3.4. Každý obdélník představuje pojmenovanou obrazovku. Orientované šipky znázorňují možnou navigaci mezi obrazovkami. Z obrazovky „Detail uživatele“ se lze navigovat na „Detail kotelny“ v případě, že daný uživatel je odpovědný za konkrétní kotelnu. Z jakékoliv obrazovky se lze dodatečně za pomoci navigačního panelu přesunout na vybranou stránku z nabídky navigace.

[3]Prototyp je model nebo vzor, který simuluje vzhled a chování konečného systému. Umožňuje se zákazníkem zvalidovat grafický návrh a funkcionalitu před započetím vlastního vývoje [23].



Obrázek 3.4: Diagram schématického zobrazení navigace v prototypu aplikace.

Kapitola 4
Implementace

Kapitola Implementace popisuje fázi tvoření samotného informačního systému podle navrženého řešení v kapitole 3 a funkcionalit požadovaných společností, definovaných v kapitole 2.

Zahrnuje rovněž výběr technologií, pomocí nichž je řešení zhotoveno. Výběr je důležitým aspektem realizace systému, jelikož výběr vymezuje, s jakými možnostmi a omezeními je nutno při implementaci počítat. Samotný výběr je tématicky rozdělen na serverovou a klientskou část.

4.1 Server

K implementaci serverové strany formou webové aplikace existuje široké množství technologií napříč různými jazyky. Mezi technologie, mezi kterými jsem si vybíral je framework Spring Boot v jazyce Java, Django v jazyku Python a framework ASP.NET Core.

Spring Boot

Spring Boot je open-source framework poskytující jednoduchý a rychlý způsob vytváření webových aplikací v jazyce Java. Jeho klíčovým filozofií je konvence nad konfigurací, což znamená, že aplikace je přednastavena s výchozími hodnotami, které umožňují okamžitý start bez nutnosti manuálního nastavování. To umožňuje vývojářům soustředit se přímo na psaní kódu a implementaci logiky aplikace, místo aby ztráceli čas ruční konfigurací [25].

Hlavní charakteristickou vlastností Spring Boot je jeho využívání principu Inversion of Control (IoC). To přesouvá zodpovědnost za řízení životního cyklu objektů a jejich konfiguraci na kontejner spravovaný Spring Boot frameworkem, místo aby tento životní cyklus manuálně spravoval samotný vývojář.

Další vlastností, kterou tento framework využívá, je Dependency Injection (DI). To umožňuje, že závislosti objektu jsou (tj. jiné objekty, které objekt potřebuje k práci) poskytnuty zvenčí, obvykle prostřednictvím konstruktorů, metod nebo vlastností. Tento přístup umožňuje vytvářet objekty, které jsou méně závislé na konkrétních implementacích a je snazší je testovat a znovu používat [26].

Spring Boot rovněž nabízí vestavěnou podporu pro mnoho běžně používaných technologií v ekosystému Javy, jako je Spring Framework, Spring Data, Spring Security a mnoho dalších. Díky tomu vývojáři mohou snadno integrovat tyto nástroje do svých aplikací bez nutnosti složité konfigurace a instalace [25].

Django

Django je vysoko úrovňový open-source webový framework napsaný v jazyce Python, jehož cílem je urychlit vývoj webových aplikací. Jeho architektura zdůrazňuje koncept „batteries-included“, což znamená, že poskytuje komplexní sadu nástrojů a funkcí přímo v jádře frameworku, potlačující potřebu závislosti na externích knihovnách [27].

Jedním z klíčových prvků Django jsou tzv. „templates“ (šablony), které umožňují vývojářům jednoduše generovat prezentační vrstvu pomocí souborů ve formátu jako je HTML nebo CSV. Tyto šablony obsahují proměnné a ovládací struktury, které jsou vyhodnoceny v době generování finálního obsahu, který je následně poskytnut klientovi [28].

Dalším užitečným prvkem je vestavěné administrační rozhraní umožňující přímou správu datového modelu z webového prohlížeče. Tento nástroj poskytuje uživatelsky přívětivé prostředí pro přidávání, úpravu a mazání záznamů v databázi, což usnadňuje administrátorské úkoly bez nutnosti vlastní implementace administrátorského rozhraní. Administrátoři tak mohou efektivně spravovat obsah aplikace přímo pomocí tohoto administračního rozhraní a provádět různé operace bez zásahu do kódu aplikace [29].

ASP.NET Core

ASP.NET Core je open-source framework pro tvorbu webových aplikací vyvinutý společností Microsoft. Jedná se o evoluci původního ASP.NET frameworku, který nabízí mnoho vylepšení a nových funkcí. Další vlastností je přenositelnost, díky níž může být aplikace vyvíjena a provozována na různých platformách, včetně Windows, Linuxu a macOS [30].

Dalším klíčovým rysem ASP.NET Core je jeho vysoká výkonnost a škálovatelnost. Podle testu TechEmpower, který se zaměřuje na výkonnost webových frameworků [31], se ASP.NET Core umístil jako třetí nejvýkonnější framework s počtem odbavených 7 006 142 textových požadavků za sekundu. Pro srovnání, Django dosáhlo pouze 300 170 požadavků za sekundu a Spring Boot 506 087 požadavků za sekundu.

4.1.1 Vybraná technologie

Pro serverovou aplikaci jsem zvolil Spring Boot [25] aplikaci v jazyce Java, jelikož je pro vyvíjený typ informačních systémů oblíbený a je široce podporovaný s dostupnými knihovnami, které ho rozšiřují [32]. Zároveň jsem se v rámci studia na ČVUT FEL dobře seznámil, jak se Spring Boot, tak

s jazykem Java, tudíž s ohledem na předchozí zkušenosti jsem opět vybral tuto technologii pro implementaci serverové strany.

4.1.2 Persistentní vrstva

Pro relační databázi, jako úložiště aplikačních dat, jsem zvolil open-source PostgreSQL. V posledních letech nabírá na popularitě díky svému výkonu, možnostem a skutečnosti, že se jedná o otevřený software [33].

Napojení Spring Boot aplikace na databázi řeším pomocí Objektově Relačního Mapování (ORM) s Java Persistence API (JPA) [34]. ORM mapuje objekty v aplikaci na tabulky v relační databázi. JPA poskytuje rozhraní, pomocí něhož lze s daty pracovat jako s objekty jazyka Java pomocí metod a atributů, bez nutnosti ručního psaní SQL dotazů. Ve Spring Boot je JPA realizována pomocí knihovny Hibernate.

Přístup k databázovým operacím zajišťuje Spring Data JPA. Tato knihovna je abstrakční vrstva nad JPA, která poskytuje předpřipravené metody pro získávání, ukládání a mazání dat z databáze. Díky této knihovně odpadá nutnost manuálně psát opakující se metody. Dále umožňuje psát databázové dotazy ve formě signatury Java metody [35].

V aplikaci jsem konkrétně používal interface implementující JpaRepository, který poskytuje zmíněné předpřipravené metody pro práci s databázovými objekty. Příklad použití JpaRepository je k vidění v ukázce 4.1.

```
1 public interface InspectionRepository extends JpaRepository<
      Inspection, Long> {
2
3     List<InspectionEntity> findInspectionsByValidUntilBetween(
      LocalDate from, LocalDate to);
4 }
```

Ukázka 4.1: Definice interface implementující JpaRepository pro přístup k databázovým objektům. Metoda „findInspectionByValidUntilBetween“ představuje vygenerovaný na základě signatury Java metody, kde pomocí signatury určím, že chci nalézt všechny entity revize, které mají datum platnosti mezi uvedenými daty.

4.1.3 GraphQL

Jako aplikační rozhraní pro komunikaci s klientskou aplikací jsem zvolil GraphQL. Jedná se o relativně nový způsob tvorby rozhraní na bázi dotazovacího jazyka. V principu funguje tím stylem, že serverová strana poskytuje seznam objektů a operace, se kterými nad objekty pracuje.

Klientská strana vytvoří dotaz a zvolí si ty atributy objektů, které potřebuje. Daný dotaz se pošle na server, který jej zpracuje a klientovi zašle zpět jen žádané atributy vráceného objektu. Je zde tudíž možnost ušetřit výkon serveru i datové linky. Server nemusí zbytečně zpracovávat a posílat složitá data, která klient ani nepotřebuje a tím by se jen plýtvalo zdroji [36].

Dva základní typy operací používaných v GraphQL pro čtení a zápis dat do serveru je query a mutation. Query se používá pro čtení dat a nevyvolává

žádné změny na serveru, zatímco mutation se používá pro zápis, aktualizaci nebo mazání dat a může měnit stav serveru. To znamená, že pomocí query můžete získávat informace, zatímco pomocí mutation můžete měnit stav dat, ačkoliv obě operace mají stejnou syntaxi a strukturu v dotazech.

V ukázce 4.2 je příklad GraphQL dotazu, který se dotazuje na všechny kotelny vedené v systému.

```
query {
    facilities {
        name
        address
        technician {
            firstname
            lastname
        }
    }
}
```

Ukázka 4.2: GraphQL dotaz pro získání všech kotelen v systému. V návratovém poli kotelen specifikuji, že potřebuji jméno, adresu a technika, který je za danou kotelnu zodpovědný. Tuto projekci lze propagovat i na vnořené typy jako je kupříkladu atribut „technician“, kde požaduji křestní jméno a příjmení technika.

Schémata

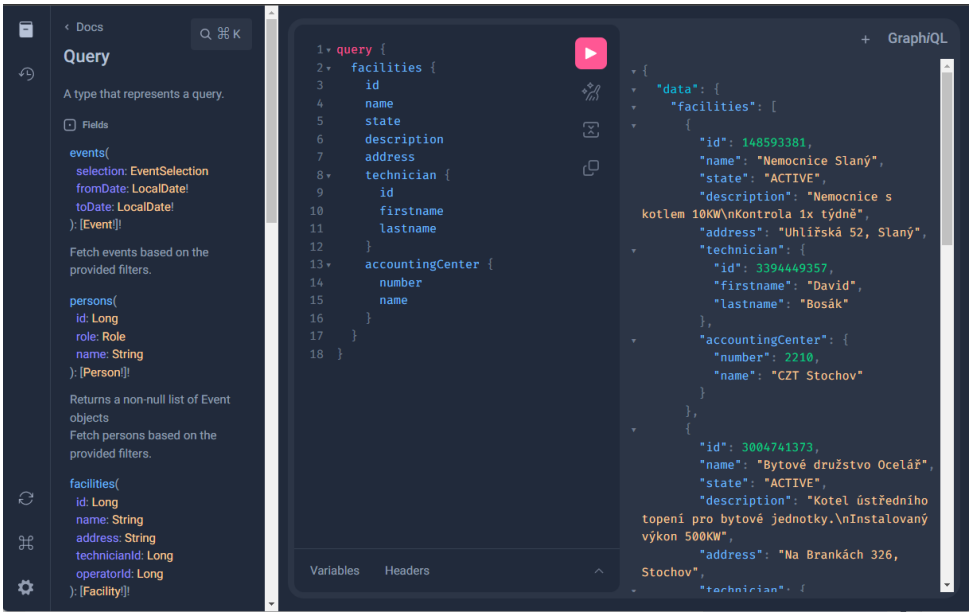
Operace dostupné v GraphQL rozhraní se popisují v souborech nazývaných GraphQL schémata. V těchto souborech jsou definovány objekty, se kterými rozhraní pracuje a jednotlivé query a mutation, kterými s rozhraním interagujeme. Po spuštění aplikace je na adrese „/graphiql“ dostupný interaktivní nástroj, který zpracovává schémata do dokumentace a grafického uživatelského rozhraní, ve kterém umožňuje vytvářet a testovat dotazy přímo v prohlížeči. Tento nástroj jsem hojně využíval při implementaci, jelikož se velmi osvědčil pro odesílání testovacích dotazů a sledování odpovědí od serveru.

Ukázka rozhraní GraphiQL je vidět na obrázku 4.1. V levé části je dokumentace rozhraní, která poskytuje přehled o dostupných objektech a dotazech. Prostřední část nabízí editor pro tvorbu dotazu, po jehož provedení se zobrazí adekvátní odpověď v pravé části.

4.1.4 Netflix DGS

K implementaci GraphQL rozhraní využívám framework DGS (Domain Graph Service) vyvinutý společností Netflix. DGS poskytuje potřebné nástroje pro implementaci rozhraní. Užitečným nástrojem frameworku je generování Java objektů přímo z GraphQL schématu, díky němuž je zaručena synchronizace změn schématu a skutečných datových objektů v aplikaci [37]. Konkrétní zpracování endpointu[1] rozhraní je demonstrována v ukázce 4.3.

[1] Endpoint je koncový bod, ke který se klient může připojit, aby komunikoval se serverem.



Obrázek 4.1: Ukázka rozhraní GraphiQL s dokumentací, dotazem a odpovědí.

```
@DgsQuery
public List<AccountingCenter> accountingCenters() {
    return accountingService.getAccountingCenters();
}
```

Ukázka 4.3: Implementace metody GraphQL rozhraní pomocí frameworku DGS ve Spring Boot. Pomocí anotace @DgsQuery označím tuto metodu jako endpoint GraphQL rozhraní pod názvem „accountingCenters“. DGS zajistí zpřístupnění tohoto endpointu i datovou projekci na návratovém typu. Tato metoda poté na příslušný dotaz navrátí všechna účetní centra.

4.1.5 Mapstruct

Rozhraní aplikace vrací uživateli tzv. Data Transfer Object (DTO)[2]. Pro mapování mezi objekty aplikace a DTO využívám knihovnu MapStruct. Tato knihovna automaticky generuje mapovací kód na základě definovaných mapovacích rozhraní a anotací, což eliminuje potřebu psát ruční mapovací logiku. Pokud zachováme jmennou konvenci mezi DTO a aplikačnimi objekty, knihovna dokáže sama tyto atributy párovat. Pokud se názvy liší, musíme definovat, jaký atribut se mapuje na jaký.

Další užitečnou vlastností MapStruct je, že dokáže využívat ostatní komponenty pro mapování. Kupříkladu, pokud chceme mapovat ID osoby na konkrétní objekt osoby, označíme mapovacímu rozhraní, ať využívá komponentu zodpovědnou za vyhledání osoby podle ID. Vygenerovaný mapovací kód

[2]DTO je datový objekt pro komunikaci, aby obsahoval pouze potřebná data pro konkrétní přenos, což může vést ke snížení objemu přenášených dat. Zároveň minimalizují závislost na interním datovém modelu [38]

poté využívá příslušnou metodu komponenty, která na vstupu požaduje ID a její návratovou hodnotou je objekt osoby. Ukázka 4.4 demonstruje mapovací interface pro entitu účetních středisek.

```
@Mapper(componentModel = "spring", uses = {PersonService.class})
public interface AccountingCenterMapper {

    @Mapping(target = "number", source = "id")
    AccountingCenter toDto(AccountingCenterEntity
    accountingCenterEntity);
}
```

Ukázka 4.4: Definice mapovacího interface knihovny MapStruct pro objekt AccountingCenter. Anotace „@Mapper“ označuje mapovací interface, „uses = PersonService.class“ označuje komponentu, kterou má využívat. Pomocí anotace „@Mapping“ určujeme, které atributy se mají mezi sebou převádět. Při kompilaci aplikace MapStruct automaticky vygeneruje implementaci tohoto rozhraní s příslušnými mapovacími metodami.

4.1.6 Autentizace

Autentizace uživatelů je zajištěna pomocí knihovny Spring Security 6 [39] integrované ve Spring Boot. Využívám standardní způsob pomocí relace pro uživatele (session). Při úspěšném přihlášení si server vytvoří novou relaci, která identifikuje daného uživatele a pošle klientovi identifikátor relace zvaný sessionId, pomocí něhož klient prokazuje svou identitu. Klient pro komunikaci se serverem musí vždy předkládat tento identifikátor (kromě přihlášení), jinak je odmítnut stavovým kódem 401 - Unauthorized [40], který oznamuje nepřihlášeného uživatele.

4.1.7 Autorizace

Autorizace aplikace je stavěna na řízení přístupu na základě uživatelských rolí, které mají definovány své pravomoce. Pro kontrolu autorizačních oprávnění opět využívám knihovnu Spring Security 6. Autorizaci k jednotlivým operacím kontroluji na jednotlivých endpointech GraphQL rozhraní. Jednotlivé metody obhospodařující dané endpointy jsou ošetřeny anotací @PreAuthorize, ve které se kontrolují podmínky přístupu jako např. role, id upravované entity. Anotace @PreAuthorize kontroluje podmínky před vstupem k metodě a při nesplnění podmínek se nepokračuje dále do těla metody a na požadavek odpoví stavovým kódem 403 - Forbidden [40], který oznamuje neoprávněný přístup.

Jednotlivé podmínky se zapisují pomocí jazyka vyhodnocování výrazů Spring Expression Language (SpEL) [39]. Pomocí tohoto jazyka lze přistupovat i k parametrům volání metody, pomocí nichž mohu rozhodovat o autorizaci.

V bezpečnostní politice aplikace kontroluji i složitější autorizační pravidla, jako je kupříkladu pravidlo, že technik provozu je oprávněn manipulovat pouze s tou kotelnou, která mu je přiřazena. Tyto autorizační podmínky kontroluji v příslušné komponentě, která je odpovědná k operacím na dané

doméně. Abych zachoval kontrolu autorizačních pravidel na úrovni metod rozhraní, využívám vkládání závislosti dostupnou v SpEL, abych delegoval kontrolu na odpovídající komponentu.

V ukázce 4.5 kontroluji, zda uživatel má autoritu „MANAGEMENT“ (pomocí „hasAuthority“) nebo zda je odpovědný za danou kotelnu. Právě ke kontrole odpovědnosti za kotelnu využívám vkládání závislosti pomocí „@“ a názvu příslušné komponenty s voláním metody. Pouze pokud je kontrola „@PreAuthorize“ úspěšná, je uživatel oprávněn k provedení těla metody a přidání provozní jednotky ke kotelně. Přihlášeného uživatele získám pomocí výrazu „authentication.principle.personEntity“. Objekt „authentication.principle“ je objekt spravovaný Spring Security, který obsahuje informace o aktuálně přihlášeném uživateli. „personEntity“ je poté mnou vytvořený objekt, který obsahuje všechny mnou potřebné informace o uživateli, který je vložen do objektu spravovaného Spring Security.

```
1 @DgsMutation
2 @PreAuthorize("hasAuthority('MANAGEMENT') || @facilityService.
      isresponsibleForFacility(authentication.principle.personEntity
      ,#facilityId)")
3 public Unit addUnitToFacility(@InputArgument Long facilityId,
      @InputArgument InputUnit unit) {
4     return facilityService.addUnitToFacility(facilityId, unit);
5 }
```

Ukázka 4.5: Zabezpečení metody pomocí Spring Security 6.

4.2 Klient

Klientská aplikace bude implementována jazykem TypeScript využívající framework Svelte. Jedná se o open-source framework vzniklý roku 2016. Syntaxe využívá kombinace HTML, CSS a TypeScript. Díky tomu má snadnou křivku učení, jelikož se syntaxí podobá zmíněným technologiím a nepoužívají se zde složité konstrukty [41]. Motivací pro výběr této technologie je jeho výkonnost, narůstající popularita a touha naučit se pracovat s tímto frameworkem.

Svelte se vyznačuje tím, že při kompilaci převádí svůj zdrojový kód na čistý JavaScript. Ten je pak přímo spouštěn v prohlížeči klienta, aniž by bylo potřeba jakéhokoliv běhového prostředí k zajištění funkčnosti kódu. [42, 43].

Na rozdíl od frameworků jako Angular, React a Vue, které běhové prostředí vyžadují, Svelte dosahuje nižšího objemu dat, které musí být přeneseny klientovi, při zachování ekvivalentní funkcionality aplikace. Další klíčovou vlastností Svelte je jeho zaměření na výkon. Ten byl testován v porovnání s uvedenými frameworky vyžadujícími běhové prostředí pomocí testu známého jako „JS framework benchmark“, který generuje rozsáhlou tabulku s náhodnými záznamy a měří čas a využitou paměť potřebné k provedení různých operací [44]. Z výsledků tohoto testu [45], prezentovaných v tabulce 4.1, je zřejmé, že Svelte dosahuje nejlepších výsledků v rychlosti provedení operací a zároveň přenáší na klienta nejmenší objem dat ve srovnání s ostatními

testovanými frameworky.

Název testu	Svelte v5.0.0	Vue v3.4.23	React v18.2.0	Angular v17.3.1
Vytvoření 1000 řádků [ms]	39,7	46,7	49,6	51,2
Prohození 1000 dvojic řádků [ms]	21,7	22,5	176,6	22,9
Použitá paměť po načtení stránky [MB]	0,5	0,7	0,9	1,3
Použitá paměť po přidání 10 000 řádků [MB]	19,4	28,2	32,1	29,6
Objem dat zaslaných klientovi [kB]	5,8	20,5	40,3	41,4

Tabulka 4.1: Porovnání výsledků testů JavaScript frameworků příslušné verze

4.2.1 Flowbite, TailwindCSS

Pro vývoj klientské části jsem využil open-source knihovnu Flowbite, která nabízí desítky interaktivních UI (User Interface) komponent. Použití komponent umožnilo urychlit vývoj a udržet jednotný vzhled aplikace.

Jednotlivé komponenty Flowbite jsou stylovány pomocí TailwindCSS, což je knihovna předpřipravených CSS tříd, které se napřímo zadávají k jednotlivým komponentám. Není tak nutno tvořit vlastní CSS třídy s jednotlivými atributy. Tailwind zároveň ve svých třídách zohledňuje responzivitu, pseudotřídy a pseudoelementy.

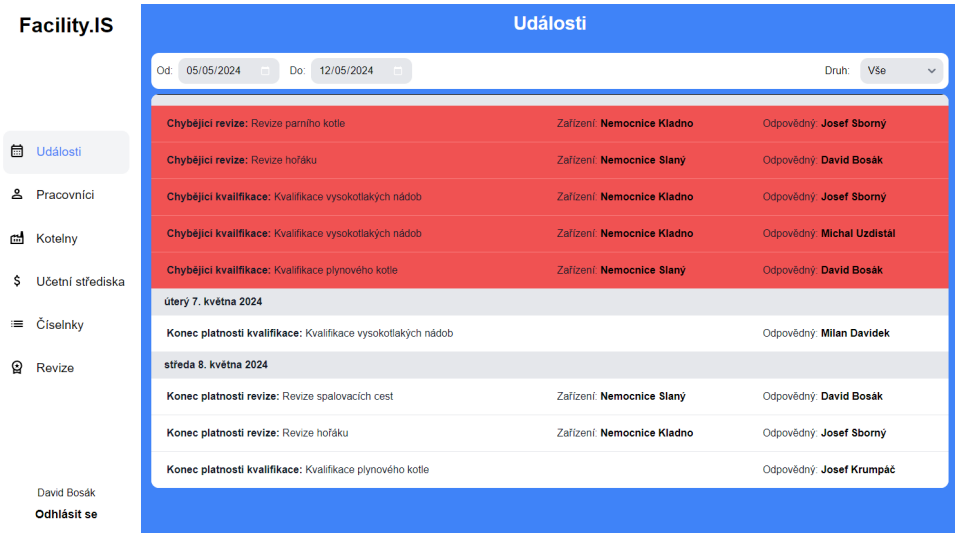
4.2.2 Snímky obrazovek

Implementace jednotlivých obrazovek se řídila podle návrhu prototypu 3.5. Avšak finální vzhled obrazovek se liší v detailech, jelikož jsem používal komponenty z knihovny Flowbite, které mají svůj přednastavený vzhled.

Celá webová aplikace si udržuje jednotný vzhled pro snazší orientaci. Po levé straně máme vždy viditelný boční panel s navigací pro dobrou orientaci napříč systémem. Barevná paleta modré a bílé byla zvolena, aby odpovídala barvám společnosti, pro kterou je aplikace vyvíjena.

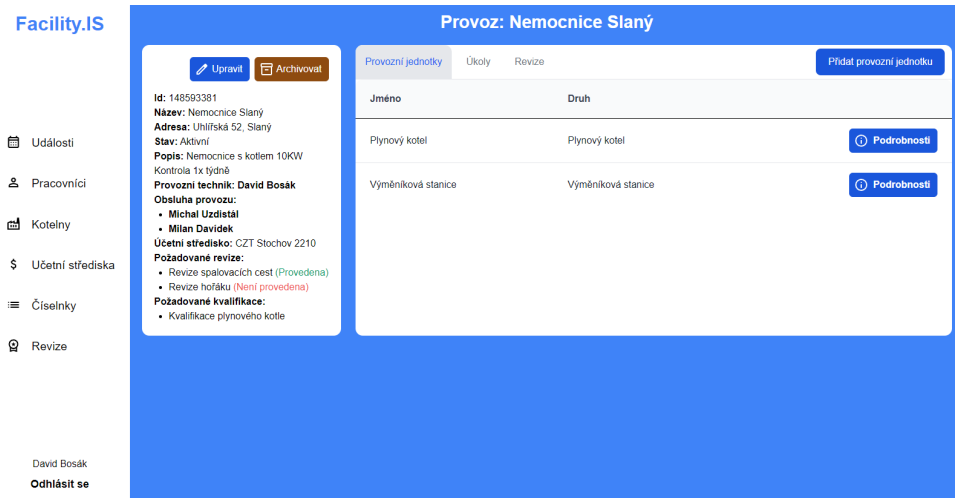
Na obrázku 4.2 je zobrazena hlavní stránka přehledu událostí. Zde jsou zobrazeny nejprve revize a kvalifikace, kterým vypršela platnost. Zároveň jsou zvýrazněny červeně, aby na sebe přitáhly pozornost, jelikož upozorňují na stav, který může být penalizován kontrolním úřadem. Pod těmito nejdůležitějšími událostmi jsou nadcházející události, které informují, k jakému dni končí jejich platnost, avšak jsou stále platné. Jednotlivé události mají na sobě proklik na odpovědnou osobu nebo zařízení, ke kterému se událost vztahuje.

Obrázek 4.3 představuje detail kotelny. V levé části je okno informací o dané kotelně. Zde podotknu, že je zde zvýrazněno, zda jsou provedeny požadované



Obrázek 4.2: Hlavní obrazovka událostí z pohledu role Manažer

revize, pro lepší přehlednost, jelikož se jedná o důležitou informaci. V pravé části lze vybírat v kartách další detail kotelny.



Obrázek 4.3: Ukázka obrazovky detailu kotelny

4.2.3 Autentizace

V klientské části aplikace se po vyplnění přihlašovacího formuláře zašle autentizační dotaz na serverovou stranu. V případě úspěchu server vytvoří relaci pro uživatele (tím ho přihlásí) a klientovi zašle identifikátor relace, společně s uživatelskými daty právě přihlášeného uživatele. Klientská aplikace si uloží identifikátor relace, který je následně zasílán spolu s každým požadavkem na server. Tímto způsobem dokáže serverová strana rozpoznat a ověřit přihlášeného uživatele.

Uživatelská data obsahují jméno, identifikátor a roli uživatele v systému a aplikace si je uloží do úložiště prohlížeče nazývaného „local storage“[3]. Díky vlastnostem tohoto úložiště umožňuje aplikace zachovat přihlášeného uživatele mezi jednotlivými otevřeními aplikace, což zvyšuje uživatelský komfort a umožňuje mu plynule pokračovat v práci bez nutnosti opakovaného přihlašování.

4.2.4 Autorizace

Klientská aplikace implementuje mechanismus řízení přístupu na základě uživatelských rolí, což znamená, že zobrazuje a umožňuje přístup pouze k funkcionalitám a datům, která jsou přístupna dané roli. Všechny ostatní funkcionality a data jsou uživateli skryta. K tomuto účelu jsem vytvořil vlastní autorizační komponentu, která kontroluje, zda jsou splněny autorizační podmínky. Pokud podmínky splněný jsou, tak je uživatel oprávněn zobrazit obsah komponenty, v opačném případě je zobrazena chybová hláška.

Například uživateli s rolí „Provozní technik“ aplikace poskytne přístup pouze k detailům kotelny, za kterou je zodpovědný. To znamená, že navigační prvky aplikace budou dynamicky upraveny tak, aby nezobrazovaly cesty k detailům ostatních kotelen, na které uživatel nemá oprávnění. Tímto způsobem je uživateli prezentována pouze relevantní a povolená funkcionalita.

Pokud by se uživatel pokusil se o přístup k neautorizovanému zdroji, aplikace provede kontrolu a pokud není autorizován, zobrazí chybovou hlášku a uživatele přesměruje zpět na domovskou stránku, aniž by mu umožnila přístup k nedovoleným datům.

4.3 Nasazení

Pro účely nasazení aplikace využívám virtualizační platformu Docker. Ta umožňuje kompaktní zabalení všech potřebných komponent aplikace, včetně souborů, knihoven a dalších důležitých prvků, do samostatného kontejneru. Díky této kontejnerizaci je aplikace izolována od okolního prostředí a může být nasazena na jakýkoliv stroj, na kterém je dostupná platforma Docker, bez ohledu na jeho konkrétní konfiguraci. Tímto způsobem je dosaženo zjednodušení procesu nasazení aplikací, neboť virtualizační platforma abstrahuje od rozdílů v infrastruktuře a zajišťuje konzistentní chování napříč různými prostředími [47].

Serverová i klientská část aplikace mají svůj konfigurační Docker soubor nazývaný Dockerfile. V tomto souboru jsou příkazy potřebné k vybudování spustitelné aplikace a následné spuštění v rámci kontejneru. Pro efektivní nasazení celkové aplikace, složené ze tří samostatných částí (klient, server,

[3]Local Storage je datové úložiště v prohlížeči, které umožňuje webovým aplikacím ukládat data přímo na zařízení uživatele. Tím aplikace může ukládat údaje o stavu aplikace nebo jiná data, která mají být dostupná i po opětovném načtení stránky nebo restartu prohlížeče. Data jsou ukládána principem klíč-hodnota a velikost úložiště je limitována na jednotky MB, nejčastěji 5MB pro jednu aplikaci [46].

databáze), se využívá Docker Compose. Tento nástroj umožňuje konfigurovat a spouštět více kontejnerů současně. Docker Compose poskytuje prostředí, kde lze specifikovat potřebné příkazy pro orchestraci a zajišťuje automatické spuštění celé aplikace. Tímto způsobem je dosaženo automatizace nasazení bez nutnosti provádět dodatečnou konfiguraci aplikace na cílových zařízeních [48].

Pro serverovou část bylo nezbytné v Docker Compose definovat tzv. „Volume“ [4] pro adresář s dokumenty a pro službu databáze. To zajistí uchování stavu databáze a uložených souborů s dokumenty i po vypnutí kontejnerizované aplikace.

Po nasazení celé aplikace je klíčové, aby v systému existoval manažerský účet, který má oprávnění zakládat nové uživatelské účty a tím zpřístupnit systém ostatním uživatelům. Bez tohoto prvotního účtu by neexistoval jiný způsob, jak nového uživatele přidat a do aplikace by se nikdo nemohl přihlásit. Proto při spuštění aplikace systém automaticky ověřuje, zda již existuje manažerský účet v databázi. Pokud ne, aplikace ho vytvoří s předem definovanými přihlašovacími údaji v konfiguraci aplikace. Tímto způsobem je zajištěno, že systém je připraven k používání již od okamžiku nasazení.

[4] Volume je mechanismus pro uchovávání dat, které jsou používány Docker kontejnery. Definují se jako cesty k hostitelskému adresáři, ve kterém data zůstanou zachována mezi jednotlivými restarty kontejneru. [47]

Kapitola 5
Testování

V této kapitole se věnuji testování vytvořené aplikace. Testování software umožňuje odhalit chyby v systému a kontrolovat, zda aplikace splňuje požadovanou funkcionalitu [49]. Proces testování probíhal ve dvou fázích a to vývojových testů prováděných během samotného vývoje a uživatelského testování, které se konalo před finálním dokončením aplikace. Pravidelné konzultace a revize s představiteli společnosti umožnily průběžnou validaci analýzy a návrhu, čímž bylo minimalizováno riziko nesouladu mezi požadavky zákazníka a výsledným řešením.

5.1 Vývojové testy

Aplikační logiku serverové strany jsem testoval pomocí automatizovaných jednotkových testů (unit tests), které ověřují jednu dílčí funkci (metodu) izolovaně od zbytku systému. Pro každý test je připraven scénář se vstupními daty a očekávaným výsledkem, který je po provedení testu vyhodnocen vůči skutečnému výsledku. Izolované testování přináší klíčovou výhodu v tom, že umožňuje testovat jednotlivé dílčí funkce nebo metody bez ohledu na ostatní části systému. To znamená, že výsledek testu není ovlivněn stavem nebo chováním jiných komponent a závisí pouze na konkrétní testované funkci. [50, 49].

Pro testování spolupráce mezi jednotlivými komponentami serverové aplikace jsem používal integrační testování. Tento typ testů slouží k ověření správného fungování aplikace jako celku, zejména pokud jde o vzájemnou interakci komponent. [51]. Ústředním cílem bylo detekovat případné nedostatky v interakcích mezi jednotlivými komponentami. Konkrétně se většina integračních testů zaměřovala na validaci správnosti zpracování požadavků a na korektnost uložení dat do databáze.

K implementaci testů jsem využíval knihovnu JUnit 5, což je framework pro psaní a automatizované spouštění testů v Javě. Pro snížení závislosti testovaných komponent jsem používal knihovnu Mockito. Ta umožňuje vytvářet falešné objekty, nazýváme mock objekty, jež simulují chování skutečných objektů v testovaném prostředí. Tyto mock objekty jsou následně použity k nahrazení skutečných objektů, na kterých testované komponenty závisejí. Tímto způsobem můžeme vytvořit izolované testovací prostředí, které

umožňuje testovat danou komponentu bez nutnosti spouštění ostatních částí aplikace. Vzhledem k integraci Mockito s JUnit5 můžeme definovat chování mock objektů v rámci testovacího scénáře [50].

Rozsah vývojových testů je uveden v tabulce 5.1. Výsledky jsou rozděleny podle sledované metriky pokrytí testů na celou aplikaci a servisní vrstvu. V závorce je uveden poměr otestovaných jednotek k celkovému počtu jednotek. Celkové pokrytí aplikace není vysoké, jelikož jsem se zaměřil především na testování aplikační logiky, která se nachází ve vrstvě nazývané „Servisní vrstva“. Tato vrstva má výrazně vyšší pokrytí testy ve srovnání se zbytkem aplikace. Ostatní části aplikace zahrnují převážně generovaný kód, jako jsou mapovací rozhraní a DTO, které neobsahují logiku vyžadující detailní testování.

	Celá aplikace	**Servisní vrstva**
Třídy	49% (108/220)	100% (16/16)
Metody	47% (538/1124)	94% (182/192)
Řádky kódu	45% (1359/2806)	92% (594/641)

Tabulka 5.1: Přehled pokrytí serverové aplikace automatizovanými testy.

5.2 Uživatelské testy

Po dokončení vývoje aplikace jsem předložil systém k otestování přímo zástupcům společnosti. Tomuto druhu testování se říká User Acceptance Tests (UAT), což jsou testy prováděné koncovými uživateli. Testovacím uživatelům se připraví testovací scénáře či úkoly, který mají dokončit a sledují se metriky testování, jako například úspěšnost, počet zaznamenaných chyb nebo doba trvání scénáře.

Cílem bylo zjistit, zda aplikace plně vyhovuje stanoveným požadavkům a očekáváním ohledně funkcionalit. Současně je prostřednictvím UAT prověřována uživatelská přívětivost, neboť tuto stránku aplikace testují jedinci, kteří budou systém aktivně využívat v praxi [52].

Testovací úkoly

Celkově jsem vypracoval čtyři úkoly, které pokrývají klíčové funkcionality pro jednotlivé uživatelské role. Zástupci společnosti poskytli testovací pracovníky, kteří převzali odpovědnost za dané role a prošli příslušnými úkoly. S cílem co nejvěrněji otestovat uživatelskou přívětivost, nebyli testovací pracovníci instruováni či zaškolováni v používání systému. Tím jsem mohl efektivněji posoudit přehlednost a logické uspořádání systému z pohledu uživatelů. Vyhodnocení úspěšnosti dokončení úkolů zahrnovalo identifikované obtíže, dobu trvání jednotlivých úkonů a zhodnocení uživatelského pohodlí a přívětivosti.

1. **Úkol**: Nalézt revize, ke kterým již dorazila faktura
   Role: Účetní
   Výsledek: Úspěch bez potíží
   Doba trvání: 30 sekund
   Poznámka: Uživatel chválí, že potřebná obrazovka pro zjištění fakturovaných revizí je vidět hned po přihlášení role Účetní. Potřebné revize se zobrazí po výběru filtru „Stav fakturace: Faktura“. Zástupce společnosti zmiňuje, že při tomto úkolu sytém velmi šetří čas, jelikož při původním způsobu dohledávání se revize hledají v čase kolem 10 minut.

2. **Úkol**: Vytvořit účet technika, vytvořit novou kotelnu a přiřadit jí nového technika
   Role: Manažer
   Výsledek: Úspěch s mírnou potíží
   Doba trvání: 3 minuty
   Poznámka: Při vytváření účtu technika byla zjištěna chyba, že pokud je zadáno heslo kratší než 8 znaků, formulář nereaguje na odeslání ani nezobrazuje chybovou hlášku. Při zadání hesla delšího než 8 znaků vše funguje správně. Proces založení kotelny a přiřazení technika proběhl bez potíží. Uživatel se orientoval v systému snadno a ocenil jeho přehlednost, jelikož vždy našel, co hledal. Nicméně uvedl, že při vytváření kotelny, pokud není v nabídce požadovaný technik, revize nebo kvalifikace, musí zavřít formulář a přidat požadovanou entitu na jiné stránce. Uvítal by možnost přidat tyto entity přímo ve formuláři tvorby kotelny. Přesto to nepovažuje za kritickou záležitost, jelikož si uvědomil, že pokud bude mít vše předem připravené, tomuto problému se vyhne.

3. **Úkol**: Zjistit chybějící revizi a zadat novou tam, kde chybí
   Role: Technik provozu
   Výsledek: Úspěch s výtkou
   Doba trvání: 1 minuta
   Poznámka: Systém uživatele okamžitě upozornil na chybějící revizi na úvodní stránce. Jedním kliknutím se dostal na příslušnou kotelnu a bez problémů zadal novou revizi. Uživatel měl výhrady u výběru data, avšak a po vysvětlení, jak se v menu výběru efektivně orientuje, je vše v pořádku. Vzhled výběru data je nyní ponechán na implementaci v prohlížeči, ale do budoucna by mohl být nahrazen vlastní, přehlednější komponentou.

4. **Úkol**: Zadat úkol pro kotelnu
   Role: Manažer
   Výsledek: Úspěch bez potíží
   Doba trvání: 1 minuta
   Poznámka: Uživatel se samostatně navigoval na potřebnou kotelnu, nalezl formulář pro přidání úkolu a úspěšně ho přidal. Uživatel ocenil

možnost dát lhůty splnění do minulosti pro případ, kdy úkol měl být hotový, ale ještě nebyl v systému evidovaný. Tím se takový úkol zobrazí v událostech jako upozornění a bude se připomínat pro urgentní zhotovení.

5.2.1 Vyhodnocení

Uživatelské testy dopadly přívětivě Zástupci společnosti, kteří aplikaci testovali, úspěšně dokončili všechny úkoly. Zároveň poskytovali cennou zpětnou vazbu, zvláště v oblasti uživatelské přívětivosti. Celkově aplikaci označili za uživatelsky přívětivou, čemuž svědčí, že všichni testovací uživatelé byli schopni se samostatně orientovat v systému.

Jediným větším úskalím uživatelské přívětivosti byla situace, kdy při vytváření nové kotelny musel uživatel zrušit formulář pro vytvoření, jelikož chyběla požadovaná entita typu revize, kvalifikace či technika ve výběru. Následně se musel uživatel přesunout na příslušnou stránku s formulářem pro přidání chybějící entity. Tento postup se mohl opakovat vícekrát, pokud se postupně objevovaly další chybějící entity.

Po konzultaci se zástupci společnosti se došlo k závěru, že důkladné zaškolení uživatelů na systém by mohlo minimalizovat tento problém. Bude se tak dbát na to, aby před samotným procesem vytváření nové kotelny byly předem vytvořeny potřebné entity. Navrhlo se také možné budoucí vylepšení, které by umožnilo přidávat entity přímo při vytváření kotelny, což by snížilo počet kroků a zjednodušilo proces pro uživatele.

Z uživatelského testování vyplynuly následující potencionální úpravy:

- Zobrazit chybovou hlášku při nesplnění podmínek na heslo. (implementováno)
- Přidat možnost vytvářet druh revize, kvalifikace a vytvářet uživatele přímo u formulářů, kde se tyto entity zadávají (např. vytváření kotelny).
- Vylepšit uživatelskou přívětivost výběru data vlastní komponentou.

Kapitola 6

Závěr

V této práci jsem popsal proces tvorby zakázkového informačního systému zahrnující seznámení se problematikou, analýzou požadavků, návrh řešení a následnou implementaci aplikace. Mým cílem bylo zužitkovat v práci široké spektrum dovedností, které mě studium na fakultě naučilo.

Spolupráce se společností při tvorbě požadovaného informačního systému byla neocenitelnou zkušeností. Pravidelné setkání se zástupci společnosti mi poskytovalo cenný vhled do způsobů komunikace s klienty mimo oblast informačních technologií. Tato interakce byla klíčová, neboť představovala jediný zdroj informací pro identifikaci požadavků na systém, které jsem následně transformoval do navrženého a později implementovaného řešení.

Přístup společnosti byl všeobecně velmi přívětivý, což se projevilo zejména v aktivní spolupráci a ochotě vést konzultace. Bylo pro mě obzvláště ocenitelné, že zástupci společnosti věnovali čas ze své pracovní doby na pravidelná setkání se mnou. Díky tomu jsem mohl lépe proniknout a porozumět požadavkům společnosti a následně validovat průběžné výsledky tvorby systému. Bez této komunikace by hrozilo vytvoření informačního systému, který by nedokázal dostatečně reflektovat skutečné požadavky a představy společnosti, což by mohlo vést k jeho neúspěchu.

V rámci této práce jsem implementoval funkční informační systém, který plní všechny funkční i nefunkční požadavky ze strany zástupců společnosti. Systém je implementován formou webové aplikace a dělí se na klientskou a serverovou část.

Serverová část je implementována v jazyce Java s pomocí frameworku Spring Boot. Zabezpečení serverové aplikace je zajištěno s využitím Spring Security 6. Datové úložiště aplikace tvoří relační databáze PostgreSQL, jejíž data jsou na serveru reprezentována a manipulována pomocí JPA. Server poskytuje aplikační rozhraní GraphQL pro komunikaci s klientem. Klientská strana je tvořena jazykem TypeScript používající framework Svelte. Uživatelské rozhraní aplikace je budováno z komponent knihovny Flowbite. Správné nasazení obou částí aplikace je zajištěno platformou Docker.

Informační systém se do společnosti bude nasazovat v průběhu léta 2024, během něhož bude probíhat i celková migrace potřebných dat. Samotnou aplikaci chci dále rozvíjet a zdokonalovat podle potřeb společnosti, aby co nejlépe plnila svůj účel. Prvním zdokonalením se do budoucna nabízí vylepšení

uživatelské přívětivosti ohledně přidávání potřebných entit u přidání kotelny. Je pravděpodobné, že při ostrém provozu aplikace vyvstane více požadavků na změnu a vylepšení, které bude potřeba implementovat.

Bibliografie

1. *Nařízení vlády č. 191/2022 Sb. o vyhrazených technických plynových zařízeních a požadavcích na zajištění jejich bezpečnosti* [online]. Zákony pro lidi, 2022-07 [cit. 2023-11-06]. Dostupné z: `https://www.zakonyprolidi.cz/cs/2022-191`.
2. *Nařízení vlády č. 192/2022 Sb. o vyhrazených technických tlakových zařízeních a požadavcích na zajištění jejich bezpečnosti* [online]. Zákony pro lidi, 2022-07 [cit. 2023-11-06]. Dostupné z: `https://www.zakonyprolidi.cz/cs/2022-192`.
3. *Nařízení vlády č. 194/2022 Sb. o požadavcích na odbornou způsobilost k výkonu činnosti na elektrických zařízeních a na odbornou způsobilost v elektrotechnice* [online]. Zákony pro lidi, 2022-07 [cit. 2023-11-06]. Dostupné z: `https://www.zakonyprolidi.cz/cs/2022-194`.
4. *Vyhláška č. 91/1993 Sb. Vyhláška Českého úřadu bezpečnosti práce k zajištění bezpečnosti práce v nízkotlakých kotelnách* [online]. Zákony pro lidi, 1993-04 [cit. 2023-11-06]. Dostupné z: `https://www.zakonyprolidi.cz/cs/2005-251`.
5. ČSN, 690012. *Tlakové nádoby stabilní. Provozní požadavky.* ÚNMZ, 1986.
6. ZWASS, Vladimir. *Information system* [online]. Encyclopædia Britannica, inc., 2023-11 [cit. 2023-11-12]. Dostupné z: `https://www.britannica.com/topic/information-system`.
7. BIGELOW, Stephen J.; LUTKEVICH, Ben; KRANZ, Garry. *What is network-attached storage (NAS)? A complete guide* [online]. TechTarget, 2022-09 [cit. 2023-11-12]. Dostupné z: `https://www.techtarget.com/searchstorage/definition/network-attached-storage`.
8. UNHELKAR, Bhuvan. Software Engineering with UML. In: 1. vyd. United Kingdom: CRC Press, 2017. ISBN 9781138297432.
9. BRUSH, Kate. *DEFINITION: use case* [online]. TechTarget, 2022-10 [cit. 2023-11-20]. Dostupné z: `https://www.techtarget.com/searchsoftwarequality/definition/use-case`.

10. BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. *The unified modeling language user guide*. 2. vyd. Boston, MA: Addison-Wesley Educational, 2005. ISBN 9780321267979.

11. *Use Cases and Scenarios: Their Importance, How to Write Them, & More* [online]. Inflectra, 2023-09 [cit. 2023-11-20]. Dostupné z: `https://www.inflectra.com/Ideas/Topic/Use-Cases.aspx`.

12. *What is Facilities Management?: A Complete Guide* [online]. ServiceChannel [cit. 2023-12-05]. Dostupné z: `https://servicechannel.com/what-is-facilities-management-software/`.

13. *Facility management EMA+* [online]. [cit. 2023-12-05]. Dostupné z: `https://cafm.emaplus.cz/`.

14. KHALID, Lina. *Software Architecture for Business*. 1;1st 2020.; Cham: Springer International Publishing AG, 2019. ISBN 9783030136314.

15. TERRA, John. *What is Client-Server Architecture? Everything You Should Know* [online]. Simplilearn, 2023-08 [cit. 2023-12-19]. Dostupné z: `https://www.simplilearn.com/what-is-client-server-architecture-article`.

16. ALI, S; ALAULDEEN, R; RUAA, A. What is Client-Server System: Architecture, Issues and Challenge of Client-Server System. *HBRP Publication*. 2020, roč. 2, č. 1, s. 1–6.

17. *Implementace obchodní logiky (LINQ to SQL)* [online]. Microsoft, 2023-04 [cit. 2023-12-19]. Dostupné z: `https://learn.microsoft.com/cs-cz/dotnet/framework/data/adonet/sql/linq/implementing-business-logic-linq-to-sql`.

18. CRAIG STEDMAN, Jack Vaughan. *data modeling: DEFINITION* [online]. TechTarget, 2021-12 [cit. 2023-12-19]. Dostupné z: `https://www.techtarget.com/searchdatamanagement/definition/data-modeling`.

19. MELTON, Jim; SIMON, Alan R. *Understanding the new SQL: a complete guide*. Morgan Kaufmann, 1993.

20. *What are ACID Properties in Database Management Systems?* [online]. MongoDB [cit. 2023-12-18]. Dostupné z: `https://www.mongodb.com/basics/acid-transactions`.

21. CHITTAYASOTHORN, Suphamit. The Misconception of Relational Database and the ACID Properties. In: *2022 8th International Conference on Engineering, Applied Sciences, and Technology (ICEAST)*. IEEE, 2022, s. 30–33.

22. *Relational vs. Non-Relational Databases* [online]. MongoDB [cit. 2023-12-18]. Dostupné z: `https://www.mongodb.com/compare/relational-vs-non-relational-databases`.

23. MISTRY, Pratik. *Why Software Prototyping Should Be a Crucial Part of Your Development Process* [online]. Radix, 2021-08 [cit. 2024-01-08]. Dostupné z: `https://radixweb.com/blog/what-is-software-prototyping`.
24. KVARDA, Rostislav. *Prototyp Informačního systému pro správu kotelen a revizí* [online]. Figma [cit. 2024-05-15]. Dostupné z: `https://www.figma.com/design/rw8yn35Bt3vvFZ0GljTKHN/Kotelny-IS?node-id=0-1&t=8TuztKOQDORtAbba-0`.
25. WALLS, Craig. *Spring Boot in action*. Simon a Schuster, 2015.
26. CRUSOVEANU, Loredana. *Intro to Inversion of Control and Dependency Injection with Spring* [online]. Baeldung, 2024-04 [cit. 2024-05-15]. Dostupné z: `https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring`.
27. *Django introduction* [online]. Mozilla [cit. 2024-05-15]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction`.
28. *The Django template language* [online]. Django, 2024-04 [cit. 2024-05-15]. Dostupné z: `https://docs.djangoproject.com/en/5.0/ref/templates/language/#templates`.
29. *Django introduction* [online]. Mozilla [cit. 2024-05-15]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Admin_site`.
30. LOCK, Andrew. *ASP. NET core in Action*. Simon a Schuster, 2023.
31. *Web Framework Benchmarks* [online]. TechEmpower, 2024-04 [cit. 2024-05-15]. Dostupné z: `https://www.techempower.com/benchmarks/#hw=ph&test=plaintext&section=data-r22`.
32. MYTHILY, M; RAJ, A Samson Arun; JOSEPH, Iwin Thanakumar. An analysis of the significance of spring boot in the market. In: *2022 International Conference on Inventive Computation Technologies (ICICT)*. IEEE, 2022, s. 1277–1281.
33. LINSTER, Marc. *Postgres is the Undisputed Most Admired and Desired Database in Stack Overflow's 2023 Developer Survey* [online]. EnterpriseDB, 2023-06 [cit. 2024-01-29]. Dostupné z: `https://www.enterprisedb.com/blog/postgres-most-admired-database-in-stack-overflow-2023`.
34. TUDOSE, Cătălin; ODUBĂŞTEANU, Carmen. Object-relational mapping using JPA, Hibernate and Spring Data JPA. In: *2021 23rd International Conference on Control Systems and Computer Science (CSCS)*. IEEE, 2021, s. 424–431.
35. CAMBI, Luca. *Difference Between JPA and Spring Data JPA* [online]. Baeldung, 2024-01 [cit. 2024-05-05]. Dostupné z: `https://www.baeldung.com/spring-data-jpa-vs-jpa`.

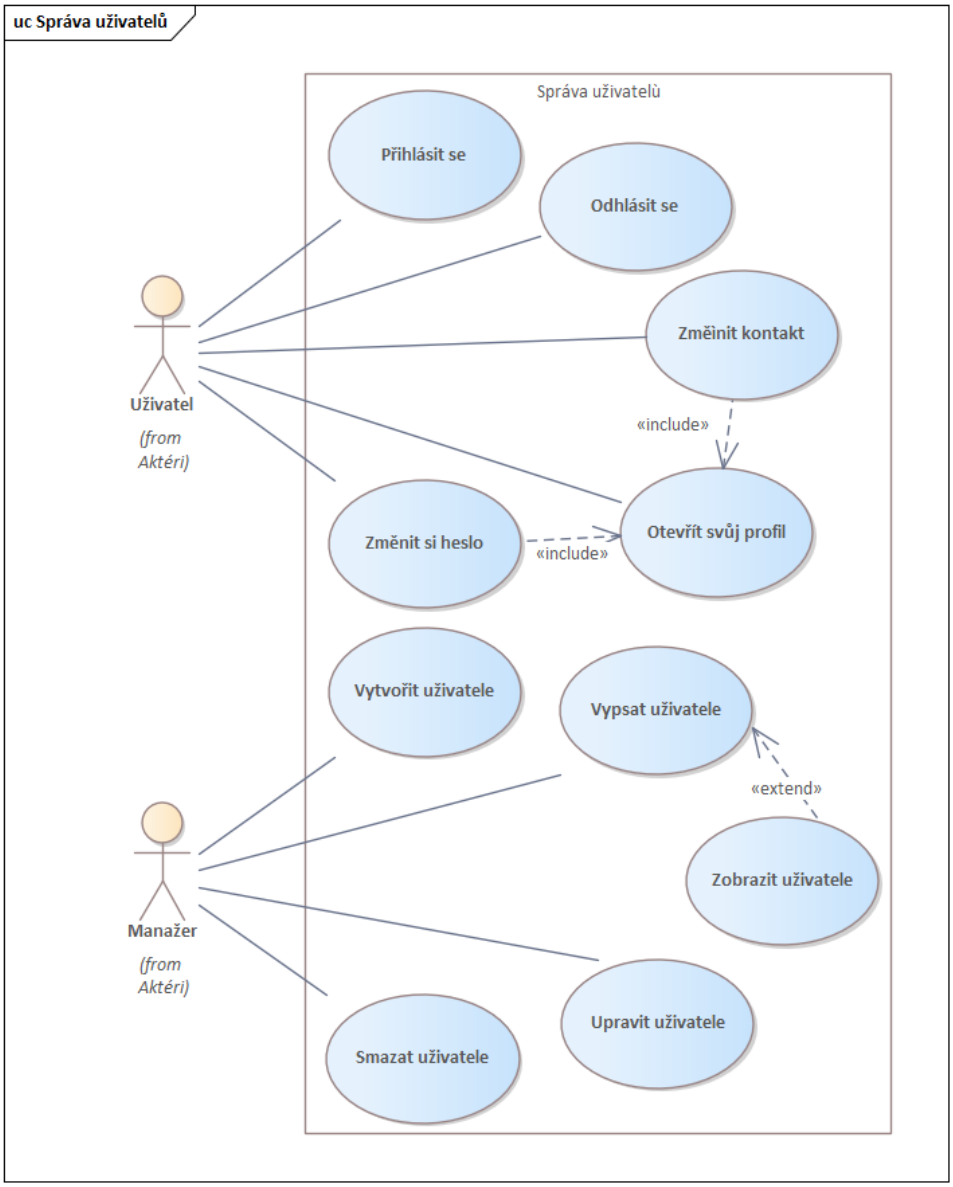
36. QASIM, Awais. *Deep dive into GraphQL: Benefits and applications* [online]. educative, 2023-05 [cit. 2024-01-29]. Dostupné z: `https://www.educative.io/blog/graphql-benefits-and-applications`.
37. *Netflix DGS documentation* [online]. [cit. 2024-05-05]. Dostupné z: `https://netflix.github.io/dgs/`.
38. FOWLER, Martin. *Patterns of enterprise application architecture*. Addison-Wesley, 2012.
39. ALEX, Ben; TAYLOR, Luke. *Spring Security*. 2022.
40. *HTTP response status codes* [online]. Mozilla [cit. 2024-05-15]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Status`.
41. BHUSHAN, Ashirvad. *Svelte vs React: Features, Performance, and More* [online]. Kinsta, 2023-11 [cit. 2024-01-29]. Dostupné z: `https://kinsta.com/blog/svelte-vs-react/`.
42. VOLKMANN, Mark. *Svelte and Sapper in Action*. Simon a Schuster, 2020.
43. *Getting started with Svelte* [online]. Mozilla [cit. 2024-05-11]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Svelte_getting_started#svelte_a_new_approach_to_building_rich_user_interfaces`.
44. KRAUSE, Stefan. *js-framework-benchmark* [online]. GitHub [cit. 2024-05-15]. Dostupné z: `https://github.com/krausest/js-framework-benchmark`.
45. KRAUSE, Stefan. *Results for js web frameworks benchmark - official run* [online]. GitHub [cit. 2024-05-15]. Dostupné z: `https://krausest.github.io/js-framework-benchmark/2024/table_chrome_124.0.6367.91.html`.
46. WEST, William; PULIMOOD, S Monisha. Analysis of privacy and security in HTML5 web storage. *Journal of Computing Sciences in Colleges*. 2012, roč. 27, č. 3, s. 80–87.
47. MIELL, Ian; SAYERS, Aidan. *Docker in practice*. Simon a Schuster, 2019.
48. LIST, Markus. Using docker compose for the simple deployment of an integrated drug target screening platform. *Journal of integrative bioinformatics*. 2017, roč. 14, č. 2, s. 20170016.
49. COSMINA, Iuliana. Testing Spring Applications. In: *Pivotal Certified Professional Core Spring 5 Developer Exam: A Study Guide Using Spring Framework 5*. Berkeley, CA: Apress, 2020, s. 201–276. ISBN 978-1-4842-5136-2. Dostupné z DOI: `10.1007/978-1-4842-5136-2_3`.
50. GULATI, Shekhar; SHARMA, Rahul. Java unit testing with JUnit 5. *Java unit testing with junit*. 2017.

51. ORSO, Alessandro. Integration testing of object-oriented software. *Dottorato di Ricerca in Ingegneria Informatica e Automatica, Politecnico di Milano*. 1998.

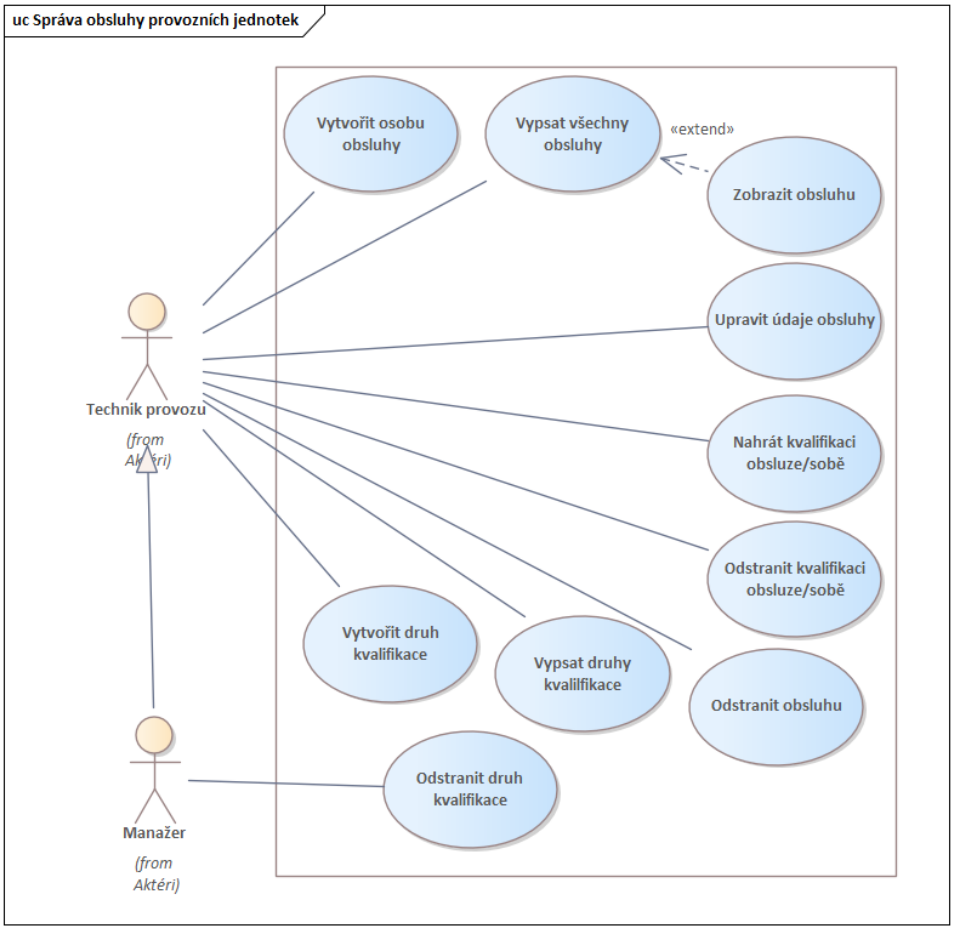
52. PANDIT, Pallavi; TAHILIANI, Swati. AgileUAT: A framework for user acceptance testing based on user stories and acceptance criteria. *International Journal of Computer Applications*. 2015, roč. 120, č. 10.

Příloha A

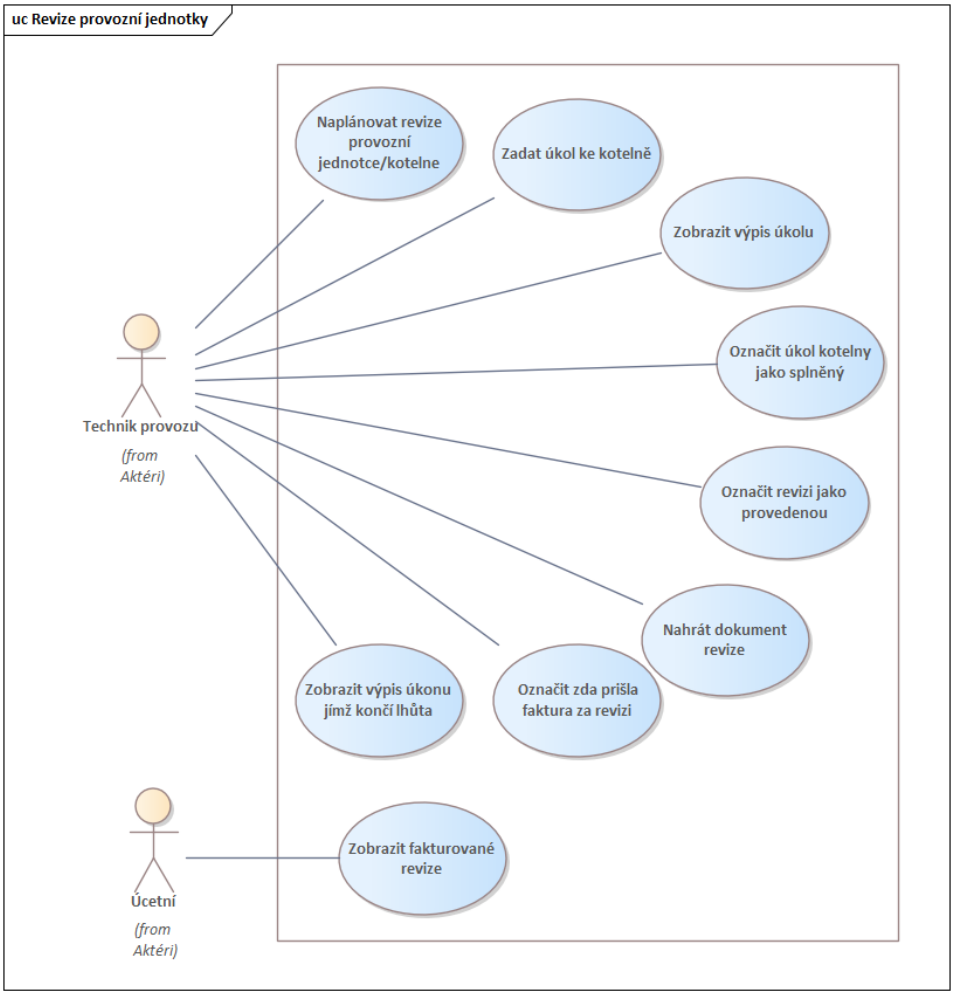
Diagramy scénářů užití



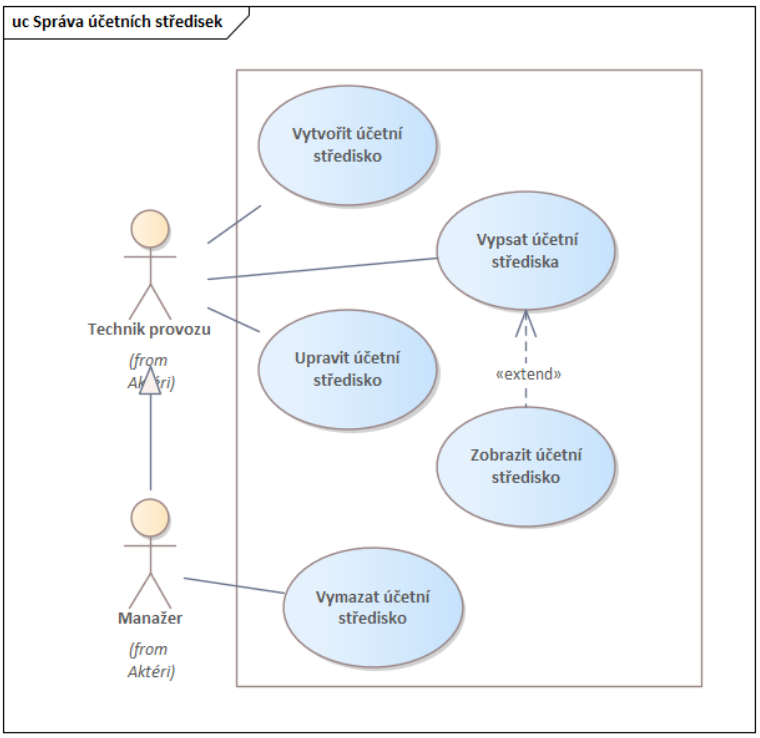
Obrázek A.1: Diagram případu užití pro správu uživatelů



Obrázek A.2: Diagram případu užití pro správu obsluhy provozních jednotek



Obrázek A.3: Diagram případu užití pro správu revizí



Obrázek A.4: Diagram případu užití pro správu účetních středisek

Příloha B

Scénáře případů užití

Archivovat kotelnu

Umožňuje archivovat kotelnu, která poté bude stále vedena v systému, avšak pouze jako neaktivní. Nebude se poté vyskytovat ve standardním výpisu kotelen a budou potlačeny jakékoliv upozornění týkající se této kotelny.

SCENARIOS

Basic Path. Hlavní scénář

Hlavní scénář:
1. Uživatel na příslušné kotelně zvolí možnost "Archivovat"
2. Systém zobrazí dialogové okno s potvrzením akce
3. Uživatel potvrdí akci archivování kotelny
4. Systém změní stav kotelny na "Archivováno". Kotelna se již nadále nebude vyskytovat ve standardním výpisu kotelen a budou potlačena všechna upozornění týkající se této kotelny

CONSTRAINTS

Invariant. Aktivní kotelna

Kotelna se nachází ve stavu "Aktivní"

[Approved, weight is 0.]

Archivovat provozní jednotku

Umožňuje archivovat provozní jednotku, která poté bude stále vedena v systému, avšak pouze jako neaktivní. Budou potlačeny jakékoliv upozornění týkající se této provozní jednotky.

SCENARIOS

Basic Path. Hlavní scénář

1. Uživatel na příslušné provozní jednotce zvolí možnost "Archivovat"
2. Systém zobrazí dialogové okno s potvrzením akce
3. Uživatel potvrdí akci archivování provozní jednotky
4. Systém změní stav provozní jednotky na "Archivováno". Budou potlačena všechna upozornění týkající se této provozní jednotky.

CONSTRAINTS

Invariant. Aktivní provozní jednotka

Provozní jednotka se nachází ve stavu "Aktivní"

[Approved, weight is 0.]

Dearchivovat kotelnu

Umožňuje převést archivovanou kotelnu do stavu "Aktivní"

SCENARIOS

SCENARIOS

Basic Path. Hlavní scénář

Hlavní scénář:
1. Uživatel si zobrazí archivovanou kotelnu a zvolí možnost "Dearchivovat"
2. Systém zobrazí dialogové okno s potvrzením akce
3. Uživatel potvrdí akci dearchivace
4. Systém uvede kotelnu zpět do stavu "Aktivní". Bude se tak objevovat ve standardním výpisu kotelen a všechna upozornění, týkající se této kotelny, budou aktivní.

CONSTRAINTS

Invariant. Archivovaná kotelna

Kotelna je ve stavu "Archivováno"

[Approved, weight is 0.]

Dearchivovat provozní jednotku

Umožňuje převést archivovanou provozní jednotku do stavu "Aktivní"

SCENARIOS

Basic Path. Hlavní scénář

1. Uživatel si zobrazí archivovanou provozní jednotku a zvolí možnost "Dearchivovat"
2. Systém zobrazí dialogové okno s potvrzením akce
3. Uživatel potvrdí akci dearchivace
4. Systém uvede provozní jednotku zpět do stavu "Aktivní". Všechna upozornění, týkající se této provozní jednotky, budou aktivní.

CONSTRAINTS

Invariant. Archivovaná provozní jednotka

Provozní jednotka se nachází ve stavu "Archivováno"

[Approved, weight is 0.]

Nastavit potřebné kvalifikace pro obsluhu jednotky

Umožňuje uživateli nastavit potřebné kvalifikace, které jsou nutné k obsluze dané jednotky.

SCENARIOS

Basic Path. Hlavní scénář

1. Uživatel zvolí možnost "Nastavit kvalifikace" u příslušné provozní jednotky
2. Systém vypíše stávající kvalifikace uložené v systému k provozní jednotce
3. IF Uživatel chce odebrat kvalifikace
   3.1 <<extend>> Přidat nutné kvalifikace
4. IF Uživatel chce přidat kvalifikace

SCENARIOS

4.1 <<extend>> Odebrat nutné kvalifikace

Odebrat nutné kvalifikace

Umožňuje uživateli odebrat kvalifikace nutné pro obsluhu provozní jednotky.

SCENARIOS

Basic Path. Hlavní scénář

1. Uživatel zvolí možnost "Odebrat kvalifikaci" u dané kvalifikace, kterou chce odebrat
2. Systém odebere požadovanou kvalifikaci z kotelny a uloží změnu

Přidat nutné kvalifikace

Umožňuje uživateli přidat kvalifikace nutné pro obsluhu provozní jednotky.

SCENARIOS

Basic Path. Hlavní scénář

1. Uživatel zvolí možnost "Přidat kvalifikaci"
2. Systém vypíše nabídku dostupných kvalifikací v systému
3. Uživatel vybere z navídky příslušnou kvalifikaci
4. Systém uloží kvalifikaci ke koteně

CONSTRAINTS

Invariant. Kvalifikace v systému

Kvalifikace, která má být přidána ke kotelně, je uvedena v systému

[Approved, weight is 0.]

Přiřadit obsluhu k provozní jednotce jednotce

SCENARIOS

Basic Path. Hlavní scénář

Hlavní scénář:
1. Uživatel u zvolené provozní jednotky zvolí možnost "Přiřadit obsluhu"
2. Systém zobrazí dostupné obsluhy, které splňují kvalifikace pro obsluhu dané provozní jednotky
3. Uživatel zvolí z nabídky příslušnou obsluhu
4. Systém uloží k provozní jednotce informaci o přiřazené obsluze

SCENARIOS

CONSTRAINTS

Invariant. Existující obsluha

V systému je vedena osoba obsluhy s příslušnými kvalifikacemi pro danou kotelnu.

[Approved, weight is 0.]

Přiřadit technika ke kotelně

Umožňuje příslušné kotelně přiřadit technika

SCENARIOS

Basic Path. Hlavní scénář

Hlavní scénář
1. Uživatel u příslušné kotelny zvolí možnost přidání technika ke kotelně
2. Systém zobrazí dialogové okno s nabídkou dostupných techniků, které splňují požadavky na kvalifikaci zadané kotelny
3. Uživatel vybere požadovaného technika
4. Systém uloží provozního technika ke kotelně a nastaví danému technikovi oprávnění spravovat kotelnu v systému

Alternate. Alternativní

Vedlejší scénář:
2a Systém nenalezne žádného technika, který splňuje požadavky
2a1 Systém zobrazí chybovou hlášku
2a2 Systém nabídne uživateli přesměrování na UC "Vytvořit uživatele"
2a3 Uživatel pokračuje na <<extend>> UC "Vytvořit uživatele"

CONSTRAINTS

Invariant. Účet technika

V systému je vytvořen účet provozního technika, který má být přiřazen ke kotelně

[Approved, weight is 0.]

Smazat kotelnu

Umožňuje vymazat ze systému archivovanou kotelnu

SCENARIOS

Basic Path. Hlavní scénář

1. Uživatel zobrazí archivovanou kotelnu a zvolí možnost "Smazat"
2. Systém zobrazí dialogové okno a bude žádat potvrzení akce
3. Uživatel potvrdí smazání kotelny
4. Systém vymaže danou kotelnu ze systému.

SCENARIOS

CONSTRAINTS

Invariant. Kotelna je archivována

Kotelna, která má být smazána je ve stavu "Archivováno".

[Approved, weight is 0.]

Smazat provozní jednotku

Umožňuje vymazat ze systému archivovanou provozní jednotku

SCENARIOS

Basic Path. Hlavní scénář

1. Uživatel zobrazí archivovanou provozní jednotku a zvolí možnost "Smazat"
2. Systém zobrazí dialogové okno a bude žádat potvrzení akce
3. Uživatel potvrdí smazání kotelny
4. Systém odebere z kotelny danou provozní jednotku a smaže jednotku ze systému.

CONSTRAINTS

Invariant. Archivovaná provozní jednotka

Provozní jednotka, která má být vymazána je v systému evidována jako "Archivována"

[Approved, weight is 0.]

Upravit provozní jednotku

Umožňuje upravit údaje u provozní jednotky

SCENARIOS

Basic Path. Hlavní scénář

1. Uživatel zvolí možnost "Upravit provozní jednotku" u příslušené provozní jednotky
2. Systém poskytne formulář pro zadání nových údajů provozní jednotky
3. Uživatel vyplní požadované údaje v potvrdí změnu
4. Systém vytvoří uloží nové údaje provozní jednotky

Vypsat kotelny

Umožňuje vypsat veškeré kotelny v systému podle zvolených filtrů, ke kterým má uživatel oprávnění

SCENARIOS

Basic Path. Hlavní scénář

SCENARIOS

Hlavní scénář:
1. Uživatel vybere v systému možnost vypsání kotelen
2. Systém zobrazí seznam všech kotelen, ke kterým má daný uživatel oprávnění
3. Systém poskytne formulář pro filtrování kotelen (např. město, ulice, název kotelny,..)
4. IF Uživatel chce filtrovat výsledky
   4.1 Uživatel vyplní formulář pro filtrování kotelen a potvrdí volbu
   4.2 Systém zobrazí kotelny vyhovující danému filtrování
1. IF Uživatel chce zobrazit konkrétní kotelnu a její detaily
   5.1 <<extend>> UC - Zobrazit kotelnu

Vytvořit kotelnu

Umožňuje zadat do systému novou kotelnu

SCENARIOS

Basic Path. Hlavní scénář

Hlavní scénář:
1. Uživatel zvolí možnost "Vytvořit novou kotelnu"
2. Systém zobrazí formulář pro vytvoření kotelny.
3. Uživatel vyplní formulář požadovanými daty a potvrdí zadané informace
4. Systém zkontroluje zadané hodnoty
5. Systém vytvoří novou kotelnu v systému, přiřadí ji jedinečný identifikátor a zadané údaje

Alternate. Alternativní

Vedlejší scénář
4a Systém vyhodnotí zadaná data jako nevalidní
   4a1 Uživatel se vrací zpět do kroku 3) s příslušnou chybovou hláškou

Vytvořit provozní jednotku

Umožní uživateli v rámci kotelny vytvořit provozní jednotku

SCENARIOS

Basic Path. Hlavní scénář

1. Uživatel zvolí možnost "Vytvořit provozní jednotku" u příslušené kotelny
2. Systém poskytne formulář pro zadání údajů pro novou provozní jednotku
3. Uživatel vyplní požadované údaje v potvrdí vytvoření
4. Systém vytvoří novou provozní jednotku a přiřadí ji ke kotelně

Zobrazit kotelnu

SCENARIOS

Basic Path. Hlavní scénář

Hlavní scénář:
1. Uživatel ze seznamu kotelen vybere příslušnou kotelnu
2. Systém zobrazí stránku dané kotelny s jejími informace
3. Systém vypíše provozní jednotky, ze který se kotelna skládá

Zobrazit kotelny podle technika provozu

Umožňuje vypsat kotelny na základě odpovědného technika provozu.

SCENARIOS

Basic Path. Hlavní scénář

1. <<include>> Vypsat kotelny
2. Uživatel vybere možnost "Zobrazit kotelny podle technika"
3. Systém poskytne nabídku dostupných techniků provozu
4. Uživatel vybere požadovaného technika
5. Systém zobrazí ty kotelny, za které je daný technik zodpovědný