UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2023 Bc. Michael Lév

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Návrh a implementace webové aplikace pro správu revizí

Diplomová práce

2023 Bc. Michael Lév

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

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Michael Lév**
Osobní číslo:	**I21281**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Návrh a implementace webové aplikace pro správu revizí**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

V teoretické části práce budou popsány současné trendy a nástroje v oblasti aplikací pro správu revizí a budou popsány výhody a nevýhody využívání webových aplikací. Dále pak budou popsány veškeré technologie, které budou pro realizace páce využity.

V rámci praktické části diplomové práce bude prvně třeba provést sběr a analýzu funkčních a nefunkčních požadavků. Po této části bude pozornost věnována návrhu vhodného databázového modelu a celkové koncepci webové aplikace včetně UI. V další kroku bude následovat vlastní implementaci webové aplikace pro správu revizí, a to s využitím frameworku JAVA Spring a technologie Vaadin. Systém musí být řešen i z pohledu různých uživatelských rolí a více uživatelů.

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

Seznam doporučené literatury:

CARNELL, John. *Spring microservices in action: a multiplatform approach to building chatbots*. Shelter, Island, NY: Manning Publications Co., 2017. ISBN 16-172-9398-9.

CARNELL, John. *Beginning spring boot 2: applications and microservices with the spring framework*. New York, NY: Springer Science Business Media, 2017. ISBN 978-148-4229-309.

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

Datum zadání diplomové práce: **8. listopadu 2022**
Termín odevzdání diplomové práce: **19. května 2023**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

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

V Pardubicích dne 30. listopadu 2022

Prohlašuji:

Práci s názvem Návrh a implementace webové aplikace pro správu revizí jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

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

V Pardubicích dne 17. 5. 2023

Bc. Michael Lév

PODĚKOVÁNÍ

Rád bych poděkoval Ing. Janu Fikejzovi, Ph.D. za odborné vedení, cenné rady a pomoc při zpracovávání této diplomové práce. Děkuji také své rodině, která mě po celou dobu studia na vysoké škole výrazně podporovala.

ANOTACE

Diplomová práce se věnuje problematice systémů pro správu revizí. V teoretické části jsou představeny základní informace týkající se revizí, dále jsou popsány současné nástroje pro práci s revizemi a následně je provedeno jejich porovnání. Poté jsou popsány výhody a nevýhody využití webových aplikací. Praktická část práce se věnuje popisu využitých technologií a následně pak vlastnímu návrhu, implementaci, testování a nasazení webové aplikace pro správu revizí, která je postavena na technologiích Java, Spring Boot a Vaadin.

KLÍČOVÁ SLOVA

správa revizí, webová aplikace, Vaadin, Java, Spring Framework, Spring Boot

TITLE

Design and implementation of a revision management web application

ANNOTATION

This diploma thesis deals with the problem of revision management systems. In the theoretical part, basic information about revisions is presented, then the current tools for working with revisions are described and then their comparison is made. Then the advantages and disadvantages of using web-based applications are described. The practical part of the thesis is devoted to the description of the technologies used and then the actual design, implementation, testing and deployment of a web application for revision management, which is based on Java, Spring Boot, and Vaadin technologies.

KEYWORDS

revision management, web application, Vaadin, Java, Spring Framework, Spring Boot

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK A ZNAČEK

API	Application Programming Interface
AWS	Amazon Web Services
CLI	Command Line Interface
CRM	Customer relationship management
CSS	Cascading Style Sheets
ČSN	Česká Technická Norma
DAO	Data Access Object
DDoS	Distributed Denial of Service
HTML	HyperText Markup Language
IČO	Identifikační číslo osoby
IDE	Integrated Development Environment
JAR	Java Archive
JDBC	Java Database Connectivity
JDK	Java Development Kit
JPA	Java Persistent API
JVM	Java Virtual Machine
MVC	Model-View-Controller
ORM	Object-Relational Mapping
OS	Operační systém
PaaS	Platform as a Service
REST	Representational state transfer
SDK	Software Development Kit
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
UX	User Experience
VM	Virtual machine
W3C	World Wide Web Consortium
WAR	Web Application Archive
WYSIWYG	What You See Is What You Get
XML	Extensible Markup Language

ÚVOD

Dnes ve svém okolí máme celou řadu různých zařízení, strojů či spotřebičů, které při poruše, nesprávné funkčnosti nebo při nesprávném zacházení mohou způsobit škody na majetku či mohou ohrozit zdraví osob či zvířat. Mezi takovéto zařízení, které mohou mít lidé doma patří např. různé elektrospotřebiče, plynové spotřebiče, plynové kotelny nebo komíny. Ale na takovéto zařízení lze také narazit i ve veřejném prostoru, jedná se například o výtahy, hromosvody, různé tlakové nádoby, hasící přístroje nebo posilovny a venkovní dětská hřiště a další.

Všechna tato zařízení mají často společného jmenovatele, a tím je zákonná povinnost provádět pravidelné kontroly neboli revize za účelem ověření technického stavu a jejich bezpečnosti. Tyto revize vždy musí provádět revizní technik, který je k takovým to účelům řádně proškolen a má platné osvědčení a oprávnění od Technická inspekce České republiky. Pokud by takovéto zařízení zůstalo v provozu i nadále bez platné revize, mohlo by dojít nejen k ohrožení zdraví a majetku, ale provozovatel se také vystavujete různým postihům.

V dnešní době s narůstajícím objemem dat a informací, které nás každý den zahlcují pak není jednoduché mít přehled o takovýchto povinných revizích a kontrolách, obzvláště pokud se opakují v delších časových intervalech např. v řádu let. Obdobný problém pak mohou mít i různí správci budov či přímo revizní technici, kteří musí takovýchto termínů hlídat daleko více. Z tohoto důvodu vznikla tato práce, která má za cíl navrhnout a implementovat vlastní aplikaci pro správu revizí. Tato aplikace by měla umět automaticky hlídat a upozorňovat na termíny a zároveň shromažďovat veškerá potřebná data na jednom místě, včetně revizních zpráv a další průvodní dokumentace.

I přestože dnes existuje na trhu několik podobných aplikací, tak ne všechny jsou uživatelsky přívětivé, dostatečně jednoduché či snadno dostupné. Cílem této práce by mělo být se poučit z chyb těchto aplikací a přinést novou jednoduchou aplikaci, která však bude mít všechny klíčové funkcionality doplněné o moderní design a intuitivní uživatelské rozhraní. Důraz by měl být také kladen na responzivitu z důvodu, že dnes již valná většina potencionálních uživatelů vlastní mobilní telefon či tablet. Tyto zařízení se jeví jako ideální řešení pro využití aplikace v terénu, a proto je důležité, aby byla aplikace snadno ovladatelná i na těchto přístrojích.

V teoretické části je cílem nejprve vysvětlit základní pojmy a další nezbytné informace spojené s revizemi. Dále pak budou představeny současné aplikace, kde budou popsány především jejich klady a zápory, a následně jejich celkové porovnání a zhodnocení. V závěru této části se pak práce bude věnovat tématu webových aplikací, jejich výhod a nevýhod a také jejich samotnému vývoji.

V praktické části bude popsán celkový vývoj vlastní aplikace pro správu revizí. V prvním kroku budou představeny použité technologie. Následně bude popsána provedená analýza a návrh celé aplikace. Základem tohoto kroku je sběr požadavků, který je pro celý vývoj stěžejní. Na jeho základě pak bude vypracován datový model, skládající se z jednotlivých datových entit a v návaznosti na to vytvořen návrh uživatelského rozhraní. Další část se bude věnovat samotné implementaci, kde bude nejprve představen celkový koncept a následně budou vybrány a popsány ukázky funkcionalit spolu se zdrojovými kódy. Poslední část se bude věnovat testování a nasazení aplikace do produkčního prostředí.

1 REVIZE

Jako revizi můžeme v širším slova smyslu chápat kontrolu vybraného zařízení či systému. Cílem revize je ověřit technický stav a zajistit tak bezpečnost při používání kontrolovaného zařízení. Pravidla pro revize a revizní techniky se řídí různými zákony, vyhláškami a normami, především pak zákonem č. 250/2021 Sb., Zákon o bezpečnosti práce v souvislosti s provozem vyhrazených technických zařízení.
Revize se provádí především u zařízení, u kterých hrozí ohrožení života, zdraví a bezpečnosti osob, zvířat či majetku. Jedná se například o:

- elektrická zařízení – elektroinstalace, hromosvody, elektrické spotřebiče,
- plynová zařízení – plynové spotřebiče, plynové kotelny,
- zdvihací zařízení – výtahy, jeřáby,
- tlaková zařízení – talkové nádoby,
- protipožární zařízení – protipožární systémy, hasící přístroje, hlásiče požáru,
- revize komínů a spalinových cest.

Každá revize má jasně určenou lhůtu její platnosti. Před jejím vypršením je vždy nutné provést revizi novou, pokud se plánuje zařízení či systém i nadále používat. V opačném případě pak hrozí postihy. Lhůty pro revize jsou určeny normami ČSN nebo vyhláškami a liší se na základě typu daného zařízení a míry rizika, které představuje.

Revize vždy smí provádět pouze revizní technik s příslušným oprávněním. Tyto oprávnění uděluje Technická inspekce České republiky na základě zkoušky z odborné způsobilosti. Na jejich webu lze najít informace k získání osvědčení, vzorové otázky i veřejnou evidenci revizních techniků [3].

Revize lze také rozdělit dle doby, kdy se revize provádí. Jedná se o výchozí revize, které jsou prováděny před uvedením zařízení do provozu. Následně to mohou být revize pravidelné, které jsou prováděny v pravidelných intervalech dle zákonných lhůt. Posledním typem pak jsou revize mimořádné, které jsou prováděny na základě mimořádného požadavku. Může se jednat např. o právní předpis, technickou normu nebo na popud státní správy či majitele zařízení.

Výsledkem každé revize by měl být zpráva o revizi, která především říká, zda je zařízení schopné provozu s ohledem na jeho bezpečnost nebo je nutné provést úpravy a opravy pro jeho bezpečný provoz. Dále by měla zpráva obsahovat informace o druhu revize, informace o kontrolovaném zařízení, popis provedených úkonů, údaje revizního technika, datum zahájení, ukončení, vypracování a předání revizní zprávy či soupis zjištěných závad.

1.1 Systémy pro správu revizí

Systémy pro správu revizí představují informační systémy, uchovávající informace o revizích a zařízeních a systémech, u kterých je potřeba tyto revize provádět. U provedených revizí jsou vždy uchovávány základní informace, jejich výsledků, lhůty revizí nebo také informace plánovaných revizích. Systém by měl hlídat termíny a lhůty a automaticky uživatele upozornit na blížící se expirace. Mezi další funkce, které může nabízet lze také zařadit možnost ukládání a prohlížení dokumentů, jako jsou např. revizní zprávy, různé protokoly či fotodokumentace.

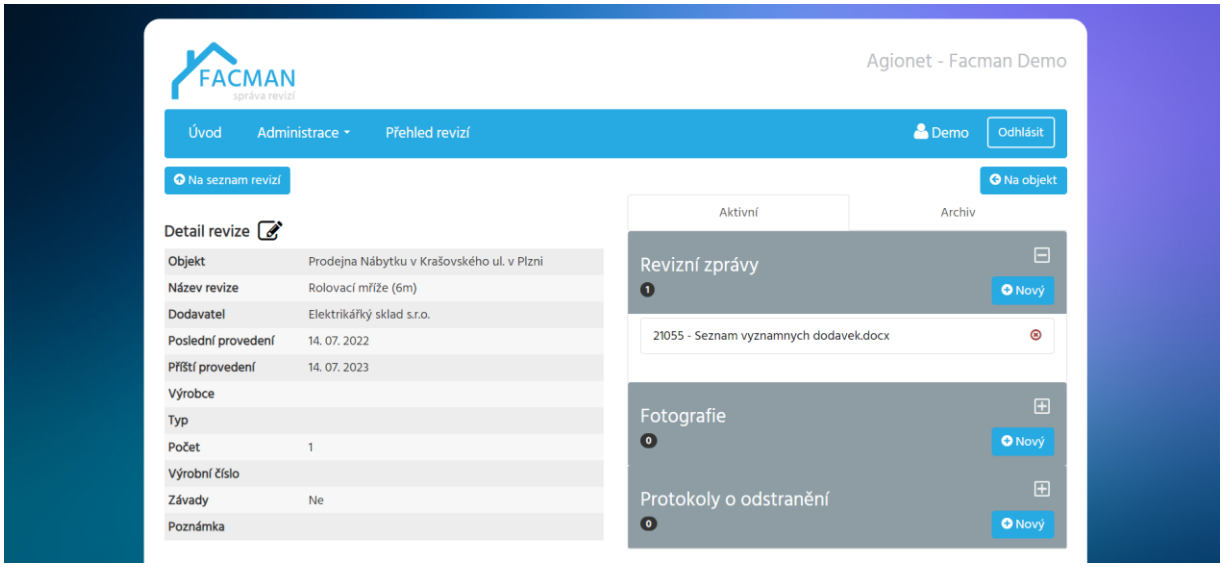
Tyto správcovské systémy mohou být určeny jak pro revizní techniky, tak pro různé správce budov či majetku. Revizní technici mohou tyto systémy využít jako evidenci svých zakázek, mohou své zákazníky upozornit na blížící se lhůty a opět jim tak nabídnout své služby. Správci mohou tyto systémy využít pro evidenci všech svých zařízení či budov, ve kterých je potřeba provádět pravidelně revize. Aplikace může správce upozornit na blížící se datum expirace a také usnadní uchovávání všech dokumentů spojených s prováděním revizí. Ty budou uloženy na jednom místě a budou dostupné on-line odkudkoliv.

Cílem těchto systémů pro správu revizí by mělo být ušetřit práci s papírováním, centralizovat všechny informace o revizích, které tak budou dostupné on-line. S tím by mělo být i spojené zálohování všech uložených informací i dokumentů. V neposlední řadě také použití těchto systémů předchází zapomenutí na znovu provedení revize, díky nimž se majitelé vyhnou případným postihům.

1.1.1 Facman

První aplikací je webová aplikace Facman vytvořená firmou Agionet s.r.o., pro jednoduchou správu revizí a veškerou administraci související s nimi.

Pro správu revizí je k dispozici vytváření revizí, jejich detail viz Obrázek 1 a seznam revizí, ve kterém lze vyhledávat pomocí různých kritérií jako je stav, kategorie revize nebo objekt. U jednotlivých záznamů je pak možné ukládat veškeré potřebné informace včetně revizních zpráv a dalších dokumentů.



Obrázek 1: Detail revize z demo aplikace Facman

Aplikace umožnuje evidenci jednotlivých objektů a budov pro jednotlivé revize. U objektů je možno ukládat adresu, kontaktní údaje, mapu. Z objektů lze také vytvářet různé skupiny a podskupiny. Aplikace také automaticky hlídá datum expirace podle přednastavených termínů dle ČSN, což velmi usnadňuje práci s dohledáváním lhůt, a uživatele umí upozornit v aplikaci i v e-mailu.

Pro revizní pracovníky aplikace nabízí záložky zakázky a klienti, ve kterých si mohou vést záznamy o svých minulých i budoucích zakázkách a všech zákaznících.

Přístup do aplikace je možný na základě jména a hesla. Jednotlivým uživatelům lze přiřazovat role, které umožnují buďto pouze čtení, zápis nebo administraci. Veškerá uložená data jsou dle uvedených informací zálohována každý den. Poskytovatel také tvrdí, že aplikace nabízí responzivní zobrazování, a tak by neměl problém ji používat na mobilním telefonu nebo tabletu, což ale není úplná pravda.

K dispozici je v rámci měsíčního předplatného ve třech variantách, a to Basic, Standard a Pro. Funkce aplikace by měli být ve všech variantách stejné, liší se pouze množstvím revizí, objektů a dat, které lze ukládat, počtem uživatelů a cenou. Pro vyzkoušení je také k dispozici demo aplikace, ve kterém si lze zdarma vyzkoušet základní funkčnost aplikace. Srovnání jednotlivých verzí najdete v následující tabulce Tabulka 1.

Tabulka 1: Srovnání verzí aplikace Facman

	Basic	Standart	Pro
Počet revizí	100	1 000	3 000
Počet objektů	10	20	30
Počet uživatelů	4	10	30
Velikost archivu	10 GB	20 GB	75 GB
Cena	450 Kč/měsíc	950 Kč/měsíc	1 950 Kč/měsíc

Zdroj: [4]

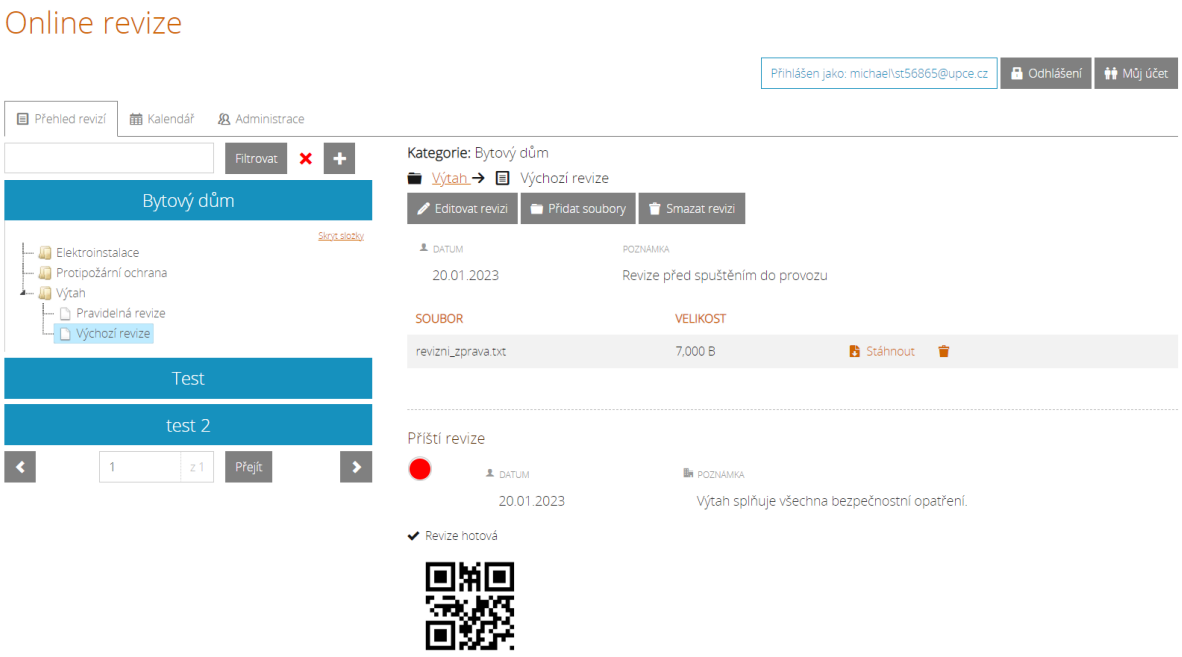
1.1.2 Online Revize

Dalším zástupcem je jednoduchá webová aplikace Online Revize, určená k evidencí revizí, která je vytvořená společností ACONTE s.r.o. Nabízí rychlý přehled o všech revizích, hlídání termínů, automatické upomínky a ukládání veškeré dokumentace na jednom místě.

Pro přístup do aplikace je využita autentizace na základě e-mailové adresy a hesla.
K dispozici jsou dva typy účtů. Uživatelské účty slouží pouze k prohlížení uložených dat, zatímco administrátorské účty umožňují editovat jednotlivé záznamy o revizích a zakládat další administrátorské a uživatelské účty.

Jednotlivé revize jsou rozděleny do tzv. kategorií, které mohou představovat rozdělení např. dle lokality, typu revize nebo zákazníka. Jejich určení závisí pouze na uživateli. U kategorie si lze uložit název, popisek, adresu, kontaktní osoby a jejich telefonní čísla a e-mail a počet dnů před expirací, kdy má přijít upozornění. K jednotlivým kategoriím lze přidávat uživatele, kteří budou následně moci nahlížet do složek a revizí v dané kategorii.

Jednotlivé kategorie pak představují stromovou strukturu podobnou souborovému systému. Po vytvoření kořenové složky je k dispozici vytváření dalších podsložek a zakládání jednotlivých revizí. Jak si uživatelé roztřídí jednotlivé revize je tedy čistě na nich. U revizí se ukládá vždy jejich název, datum revize, datum následující revize, poznámka či příznak, zda je revize hotová. K dispozici jsou také dynamická pole, u kterých si může uživatel určit jak jejich název, tak hodnotu, což umožňuje další možnosti vlastního přizpůsobení. K revizím lze také přikládat soubory nebo si nechat vygenerovat QR kód. Příklad takovéto revize je vidět na Obrázek 2.



Obrázek 2: Přehled revize v aplikace Online revize

Dále je v aplikaci k dispozici kalendář, ve kterém si lze přehledně prohlédnou data jednotlivých revizí. Pro administrátory je také k dispozici nastavení, ve kterém si může založit další administrátorské účty, nastavit si v jakém předstihu budou chodit notifikační e-maily a také podrobnosti těchto e-mailů jako je odesílatel, předmět či vlastní text zprávy. K dispozici jsou takto tři zprávy, které lze různě nastavit.

Pro používání aplikace je nutné zakoupit jeden ze tří výchozích balíčků – Start, Medium nebo Pro. Jejich základní porovnání je v následující tabulce *Tabulka* 2. Pro vyzkoušení je k dispozici balíček Pro zdarma na 30 dní, během kterých si uživatelé mají možnost se seznámit s aplikací a prozkoušet všechny nabízené funkce.

Tabulka 2: Srovnání verzí aplikace Online Revize

	Start	Medium	Pro
Počet revizí	100	500	2 000
Počet uživatelských účtů	5	10	Neomezen
Počet administrátorských účtů	1	3	10
Cena	450 Kč/měsíc	1 000 Kč/měsíc	2 000 Kč/měsíc

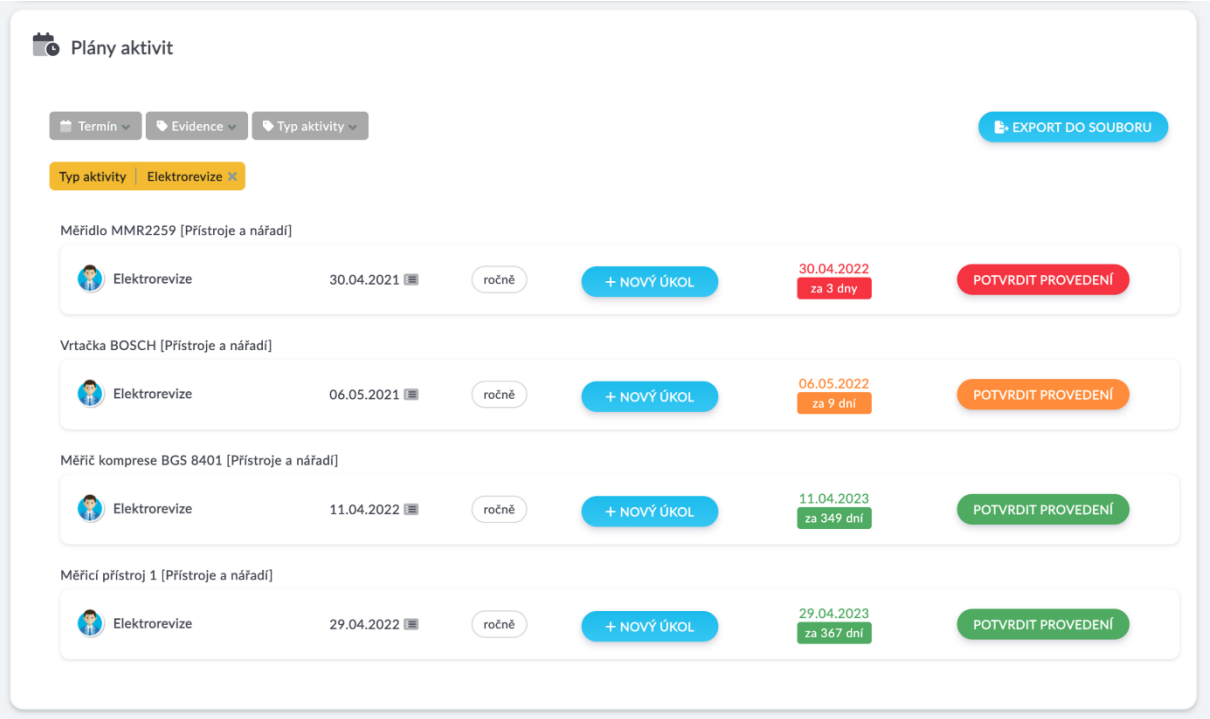
Zdroj: [4]

1.1.3 Aptien

Poslední vybranou aplikací je Aptien, respektive jedna z jejích částí. Celková aplikace se skládá z různých samostatných modulů, které nabízí širokou škálu řešení, mezi které patří řízení lidských zdrojů, správu majetku, správu smluv a dokumentů, řízení vztahu se zákazníky (tzv. CRM), projektové řízení a další. Celkově se tak jedná o komplexní informační systém, který si lze složit na míru. Dále se však budeme zabývat pouze řešením pro správu majetku, jehož součástí je také správa revizí. Mimo to tato část nabízí také evidenci majetku, vybavení a nástrojů, řešení pro inventury či preventivní údržbu.

Webová aplikace poskytuje možnost přehledu o všech revizích a kontrolách. Jednotlivé zařízení lze třídit do tzv. evidencí, které v aplikaci představují šanony, ve kterých se uchovávají informace a lze k nim individuálně řídit přístup. U jednotlivých zařízení či objektů lze zadávat jejich základní a provozní informace, kompletní dokumentaci, evidovat různé zápisy nebo přidávat úkoly a balíčky úkolů, které je následně možno přiřadit dalším uživatelům. Do systému je tak možné přímo zapojit i revizní techniky či jiné externí pracovníky.

Další funkcí je přehledný plán údržby, který je možné vidět na Obrázek 3. Zde aplikace umožňuje zadat veškeré plánované aktivity údržby, revize nebo kontroly. Ty lze poté snadno zobrazit v kalendáři díky ročnímu plánu revizí. Systém nabízí přednastavené typy zařízení a majetku i různé typy revizí. Tyto typy si lze libovolně rozšířit, případně zúžit pouze na ty, které jsou reálně potřeba.



Obrázek 3: Plán aktivit v aplikaci Aptien

Aplikace nabízí široké možnosti customizace jako například možnost definovat si vlastní uživatelské role a práva, nastavení vlastních hlídačů, vlastní tiskové formuláře, vlastní přizpůsobení jednotek atd. Do aplikace si lze také nahrát vlastní směrnice, normy a další firemní dokumenty. Nabízena je podpora češtiny i angličtiny. K dispozici je také verze přizpůsobená pro zobrazení na mobilních zařízeních.

Webová aplikace je k dispozici ve formě měsíčního předplatného dvou verzích, a to Company Premium a Enterprise Premium. Oproti předešlým aplikacím se platí měsíční poplatek vždy za uživatele, kde s rostoucím počtem uživatelů se snižuje cena za uživatele/měsíc. Porovnání obou dostupných verzí je v následující tabulce Tabulka 3. Pro vyzkoušení je k dispozici jak přístup do aplikace zdarma na 30 dní, tak demoverze aplikace pomocí testovacích přístupových údajů.

Tabulka 3: Srovnání verzí aplikace Aptien

	Company Premium	Enterprise Premium
Počet uživatelů	až 300	až 10 000
Počet evidencí	50	100
Počet záznamů	50 000	100 000
Velikost uložiště	30 GB	100 GB
Cena při 10 uživatelských účtech	490 Kč/měsíc/uživatel	950 Kč/měsíc/uživatel
Cena při 100 uživatelských účtech	300 Kč/měsíc/uživatel	420 Kč/měsíc uživatel

Zdroj: [6]

1.1.4 Porovnání aplikací

Všechny zmíněně aplikace jsou implementovány jako webové aplikace, což sebou přináší určité výhody i nevýhody. Všechna řešení jsou dostupná skrze měsíční předplatné. To je ze začátku výhodnější, protože zákazník nemusí kupovat drahou licenci a po krátké době používání se můžeme rozhodnout aplikaci dále již nepředplácet, pokud mu nevyhovuje. Nevýhody tohoto modelu se mohou objevit časem, když se poskytovatel rozhodne zvýšit ceny a zákazník se bude muset rozhodnout, zda se vyplatí zůstat u současného dodavatele či přejít na nový systém, do kterého je však nutné přenést všechna data a naučit se s ním pracovat.

Pro vybrání nejvhodnější aplikace je nutné si nejprve určit požadavky a rozpočet. Pokud je požadavek pouze na jednoduchý systém určený pouze pro správu revizí a žádné dodatečné funkce jsou vhodnou volbou první dvě aplikace Facman a Online revize. Ty jsou také určeny spíše pro jednotlivce jako revizní techniky či malé firmy. Ceny těchto dvou produktů jsou takřka srovnatelné, liší se pouze v detailech.

Správa revizí Facman může sloužit také jako jednoduchá správa budov. Výhodou jsou také předem nastavené typy revizí spolu s povinnými termíny dle normy ČSN. Uživatel se tak nemusí starat o žádné lhůty, systém ho vždy automaticky včas upozorní. Mezi další výhody můžeme zařadit ukládání souřadnic polohy a jejich interaktivní zobrazení na mapě, možnost filtrování revizí na základě různých kritérií či export dat. Nevýhodou aplikace je podpora pouze 30 uživatelů v nejvyšší variantě, oproti 10 administrátorským a neomezeném počtu uživatelských účtů u aplikace Online Revize. Další nevýhodou může být také starší design a s ním spojená lehká nepřehlednost pro nové uživatele. S designem je spojená také responzivita, která ne vždy úplně funguje, i přesto že je na stránkách tato vlastnost uvedena.

Oproti tomu systém pro správu revizí Online Revize nabízí větší volnost při práci. Nenabízí žádné přednastavené kategorie či typy. Vše si lze nastavit a roztřídit podle vlastních preferencí, díky nabízené struktuře kategorií a stromové struktuře složek. Nastavit si lze také samostatně např. znění e-mailů, které budou chodit jako upozornění. Tato volnost může být pro některé uživatele, kteří preferují již zavedenou strukturu a nechtějí ztrácet čas jejím vytvářením nevýhodou. Další nevýhodou může být také nižší počet revizí, které lze ukládat, oproti konkurenčnímu řešení Facman. U střední varianty je limit 500 revizí oproti 1 000 revizí, u nejvyšší varianty je pak limit 2 000 revizí na rozdíl od konkurenčního limitu 3 000.

Pokud se jedná o větší firmu, která požaduje také správu majetku a další rozšiřující funkce, včetně širokých možností vlastního přizpůsobení je nejvhodnější volbou aplikace Aptien. Výhodou také může být, že nabízí krom správy revizí, také další hotová řešení a systémy, která je možné dokoupit. Mezi další přednosti lze zařadit také moderní design či možnost vybrat anglickou lokalizaci. Nevýhodou aplikace je její menší přehlednost pro nezkušené uživatele díky její komplexnosti. Problémem může být také vyšší cena, která je účtována měsíčně za každého uživatele. Tato aplikace je tak spíše vhodná pro střední a větší podniky než pro jednotlivce či revizní techniky.

2 WEBOVÉ APLIKACE

Webová aplikace je počítačový program, který je uložený a spuštěný na vzdáleném serveru. Dostupná je skrze počítačovou síť Internet, případně její privátní variantu intranet, a spouští se pomocí webového prohlížeče. Webové aplikace mohou být jak jednoduché, jako např. kalkulačka nebo převodník měn, tak velmi složité systémy jako např. e-shopy, informační systémy nebo e-mailový klient.

Předností webových aplikací je, že nemusí být instalovány a jsou dostupné on-line. Není tak třeba řešit distribuci novinek či oprav ke každému klientovi zvlášť. Nejsou také závislé na konkrétní platformě, nezáleží na přístroji ani nainstalovaném OS, stačí pouze webový prohlížeč a připojení k internetu. Díky tomu k nim lze přistupovat i z mobilních telefonů či tabletů. Nevyžadují ani žádný nadbytečný výkon nebo prostor na disku, protože všechna data jsou ukládána na straně serveru.

Mezi hlavní nevýhody webových aplikací se řadí nutnost neustálého připojení k internetu. Tato nevýhoda se, ale s postupem času a rozšiřování dostupností internetového připojení stává menší a menší. V České republice mělo v roce 2022 přístup k internetu 85 % domácností [8]. Navíc již dnes existují řešení pro off-line provoz webových aplikací. Další nevýhodou může být, když se poskytovatel rozhodne přestat aplikaci či službu provozovat, ta se tak stává okamžitě nedostupná a nadále už jí nelze používat. Zde je rozdíl oproti desktopovým aplikacím, které zůstávají nainstalovány na systému uživatele. Nevýhodou také může být zpomalení aplikace, případně její nedostupnost při velkém náporu uživatelů v jeden okamžik, pokud na to není server dostatečně připraven. Stejné problémy mohou také nastat při DoS a DDoS útocích, kdy se útočník snaží server zahltit požadavky.

Problémem může být také zobrazování vzhledu aplikace na různých zařízeních a v různých internetových prohlížečích. Vzhledem k multiplatformnosti webových aplikací je mohou uživatelé zobrazovat na různě velkých obrazovkách od širokoúhlého monitoru, přes displej telefonu až po Smart TV. Vývojáři tak na toto musí při vývoji myslet a přizpůsobit design aplikace různým velikostem zobrazení. Zároveň může také nastat problém při používání různých webových prohlížečů, a to z důvodů odlišné výkonosti, různé rychlosti implementace nových funkcí a vlastností nebo různých renderovacích jader (např. Blink, Gecko, WebKit atd.), která mohou mít odlišnou implementaci webových standardů. Vždy je tedy dobré zjistit si na jakých zařízeních a prohlížečích budou uživatelé webovou aplikaci nejčastěji používat a řádně ji na nich otestovat.

2.1 Vývoj webových aplikací

Webové aplikace jsou většinou rozděleny na dvě části – frontend a backend.
Frontendová část aplikace, je ta část, která je spuštěna v prohlížeči na straně klienta a se stará o prezentační vrstvu. Při vývoji frontendu se využívají značkovací jazyk HTML pro definici struktury stránek, jazyk CSS pro stylování webových stránek a pro interaktivní a dynamické chování skriptovací jazyk JavaScript. V dnešní době jsou velmi populární různé frameworky a knihovny, které programátorům ulehčují práci při vývoji frontendu. Mezi nejpoužívanější se řadí např. Angular, React nebo VueJS.

Oproti tomu backendová část běží na straně serveru. Skládá se z databáze, která slouží pro ukládání dat a serverové aplikace, která přijímá uživatelské požadavky tzv. requesty, stará se o logiku celé aplikace a komunikaci s databází a případně i dalšími aplikacemi.

Pro naprogramování backendu lze použít různé programovací jazyky. Mezi ty nejpoužívanější se řadí Java, JavaScript, PHP, Python nebo C#. Pro usnadnění vývoje serverové části aplikace existují také různé frameworky jako např. Spring, Express, Django nebo Ruby on Rails.

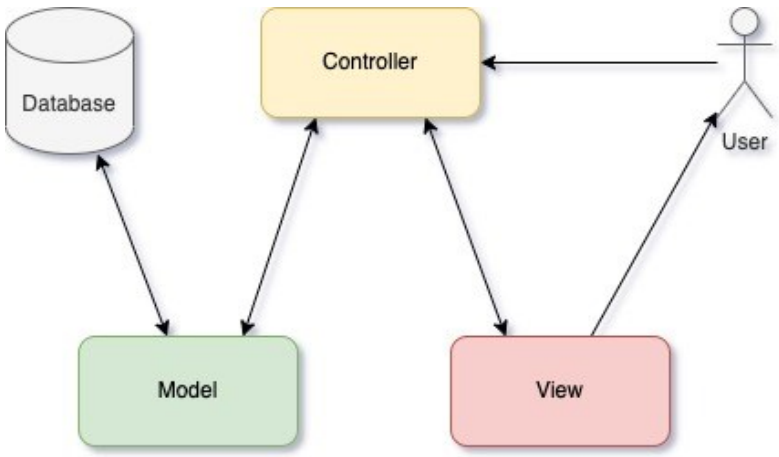
2.1.1 Model-View-Controller

Při vývoji webových aplikací se často využívá architektonický vzor Model-View-Controller (MVC). Skládá se ze tří oddělených vrstev, které spolu navzájem spolupracují viz Obrázek 4.

Vrstva model představuje datovou část, ve které se pracuje s daty a stará se o byznys logiku celé aplikace. Zodpovídá za zpracování dat a jejich následné uchovávání, modifikaci, mazání a zpřístupňování.

Další část nazvaná view, slouží k vytváření uživatelského rozhraní na základě dat získaných z modelu. Tato vrstva umožňuje uživatelům interaktivně využívat webovou aplikace. Samotná vrstva by se měla starat pouze o prezentaci a neobsahovat žádnou logiku aplikace.

Poslední částí je controller (česky řadič), který řídí veškerou komunikaci mezi předešlými dvěma vrstvami model a view. Řídí, jak potečou data z view vrstvy směrem do modelu a nazpátek. Controller je zodpovědný za správnou reakci na uživatelské změny ve view a následné správné volání funkce modelu. Následně poté pošle výsledná data z modelu zpátky do view, které aktualizuje zobrazení na základě těchto dat.



Obrázek 4: Diagram MVC

Mezi výhody MVC se řadí modularita a oddělení logické části od prezenčních. Díky tomuto rozdělení se tak snižuje komplexnost aplikace, na jednotlivých část tak lze pracovat samostatně a zvyšuje se tím také přehlednost kódu. Vývojářům také umožňuje snadnější úpravy či výměny jednotlivých části případně rozšiřování celé aplikace. Jednodušším se také stává testování jednotlivých komponent. Nevýhodou může být pro nezkušené programátory složitost jak pochopení celé architektury MVC, tak i její následná implementace. Kvůli její složitosti se tak příliš nehodí pro menší projekty.

3 VYUŽITÉ TECHNOLOGIE

V následující kapitole budou popsány technologie, které byly použity při návrhu a implementaci webové aplikace pro správu revizí. Nejpodstatnější technologií byly zcela určitě programovací jazyk Java a frameworky Spring Boot a Vaadin, které jsou na tomto jazyku postavené a díky nimž je celá aplikace naprogramovaná. Následně jsou představeny doplňkové technologie jako databáze, cloudová služba pro ukládání souborů či služba pro odesílání e-mailů.

3.1 Java

Je jeden z nejpopulárnějších open-source programovacích jazyků na světě. Mezi jeho hlavní vlastnosti patří objektově orientovaný přístup, multiplatformnost, spolehlivost a bezpečnost. Funguje na principu *„Write once and run anywhere“* neboli naprogramuj jednou, spusť kdekoliv. Lze je použít na různé druhy projektů od desktop aplikací, přes mobilní aplikace až po webové aplikací či serverové aplikace.

Díky popularitě a rozšířenosti tohoto jazyka je k dispozici velké množství výukových materiálů či různých návodů a velká a aktivní komunita. Pro práci lze využít široké spektrum nástrojů jak pro vývoj, tak pro debugging, testování či nasazování. Vývojáři pak při programování mohou ocenit velké množství vestavěných funkcí a knihoven, které umožňují rychle a efektivně vyvíjet software.

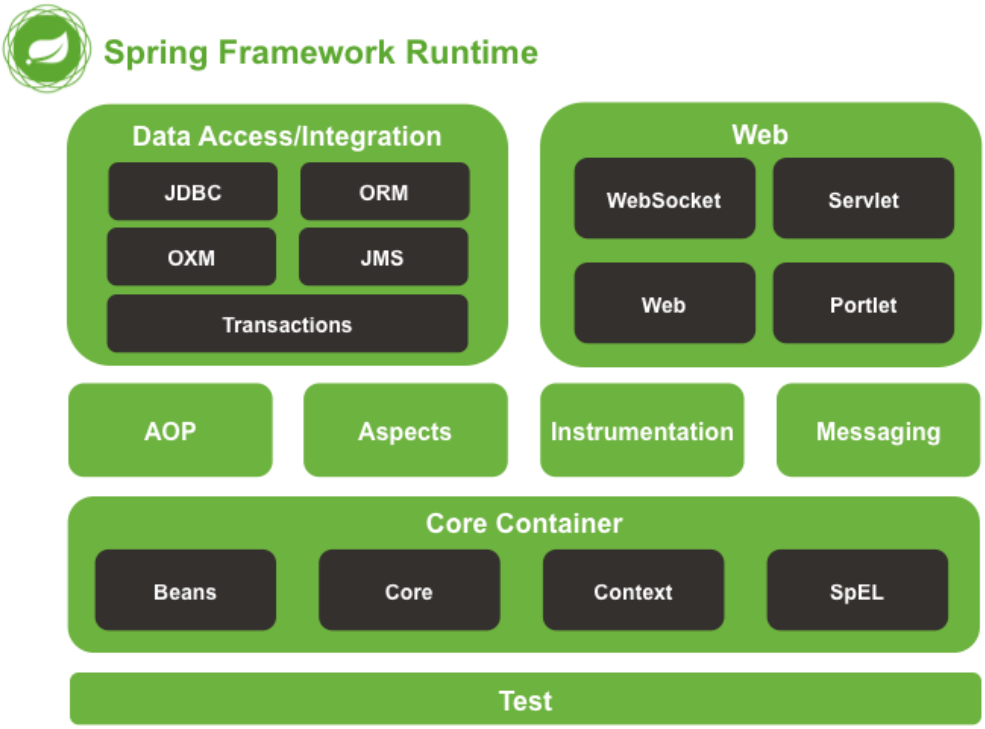
Kód v Javě je nezávislý na platformě, na které běží díky tomu, že pro svůj běh využívá Java Virtual Machine. Ta představuje přidanou vrstvu mezi programem a hardwarem. Program je vždy nejprve zkompilován do bytecode, který může běžet pouze v JVM. Ta následně tento bytecode interpretuje do strojového kódu cílové platformy. Díky tomu tak mohou programy běžet v nezměněné podobě na Windows, Linux, macOS, iOS, Android a dalších OS.
Při implementaci se jedná o hlavní programovací jazyk, který je použit a na kterém jsou postaveny dále popsané technologie.

3.2 Spring

Spring Framework je populární open-source framework určený pro vývoj rozsáhlých podnikových aplikací nebo informačních systémů především v jazyce Java. Podporuje ale i další programovací jazyky např. Kotlin nebo Groovy jako alternativu Javy.

Spring framework je rozdělen do modulů, ze kterých si vývojáři mohou vybrat ty, které zrovna pro svůj projekt potřebují. Jediným povinným modulem je Core Container. Ten tvoří srdce celého frameworku a zodpovídá za konfiguraci, řízení životního cyklu objektů a implementaci návrhových vzorů Inversion of Control a Dependency of Injection, které tvoří základ celého frameworku.

Mezi další volitelné moduly patří např. ty ze skupiny Web, zajišťující podporu funkcím pro tvorbu webu, moduly Test usnadňující testování, dále moduly AOP podporující aspektově orientované programování nebo skupina modulů Data Access díky které lze přidat podporu transakcí, objektově orientovaného mapování (ORM) či Data Access Objektů (DAO) jako jsou JDBC, Hibernate nebo JPA. Ukázka těchto modulů je vidět na Obrázek 5.



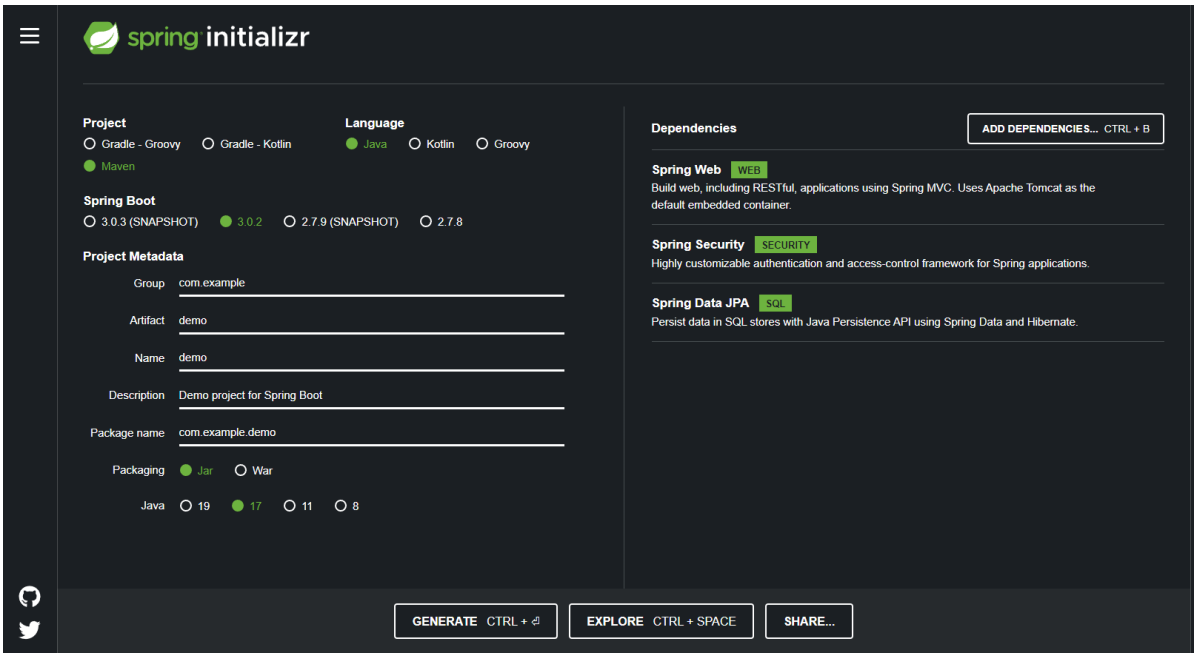
Obrázek 5: Moduly Spring Framework

Ze samotného Spring Framework vychází dále mnoho dalších projektů. Mezi ty nejznámější se řadí Spring Boot, pro rychlé vytváření aplikacích založených na tomto frameworku, Spring Cloud nabízející nástroje na vývoj pro distribuované systémy, Spring Cloud Data Flow, který slouží pro správu, zpracování a automatizaci toku dat v cloudu, nebo Spring for Android obsahující komponenty pro vývoj mobilních aplikací na Android.

3.3 Spring Boot

Spring Boot vychází ze Spring Frameworku, oproti němu však nabízí jednoduší a rychlejší vývoj samostatných aplikací. Zjednodušuje především prvotní konfiguraci a nastavení závislostí celého projektu o které se stará automaticky. Snazší je také případné upravování konfigurace, která je v Spring Boot založená na anotacích, na rozdíl od čistého Springu, kde se konfigurace prováděla skrze XML konfigurační soubory. Součástí Spring Boot je také vestavěné webové servery Tomcat, Jetty nebo Undertow. Není tak potřeba vlastnoručně nasazovat WAR soubory.

Snadná inicializace projektu je možná díky webovému rozhraní Spring Initializr, jehož ukázka je na Obrázek 6. Ten poskytuje možnost jednoduchého nastavení nového projektu, pomocí výběru programovacího jazyka a jeho verze, zvolení nástroje pro správu závislostí aplikace, verze Spring Boot a popis metadat projektu jako jsou jeho název či hlavní balíček. Poslední možností je pak výběr závislostí, které se mají k projektu připojit, tak aby aplikace obsahovala závislosti takové, které opravdu využije.



Obrázek 6: Spring Initializr

Tyto závislosti se poté připojí k aplikaci pomocí tzv. Starters. Jedná se o popisovače závislostí, které v sobě sdružují veškeré potřebné závislosti. Základním popisovačem je *spring-boot-starter*, který do projektu zahrne podporu autokonfigurace, logování a YAML. Pokud je pak cílem např. vývoje webová aplikace stačí použít *spring-boot-starter-web*. Mezi dalšími můžeme nalézt *spring-boot-starter-jpa, spring-boot-starter-security, spring-boot-starter-test, spring-boot-starter-data-elasticsearch* nebo *spring-boot-starter-data-mongodb*. Využít lze takto i starters třetích stran.

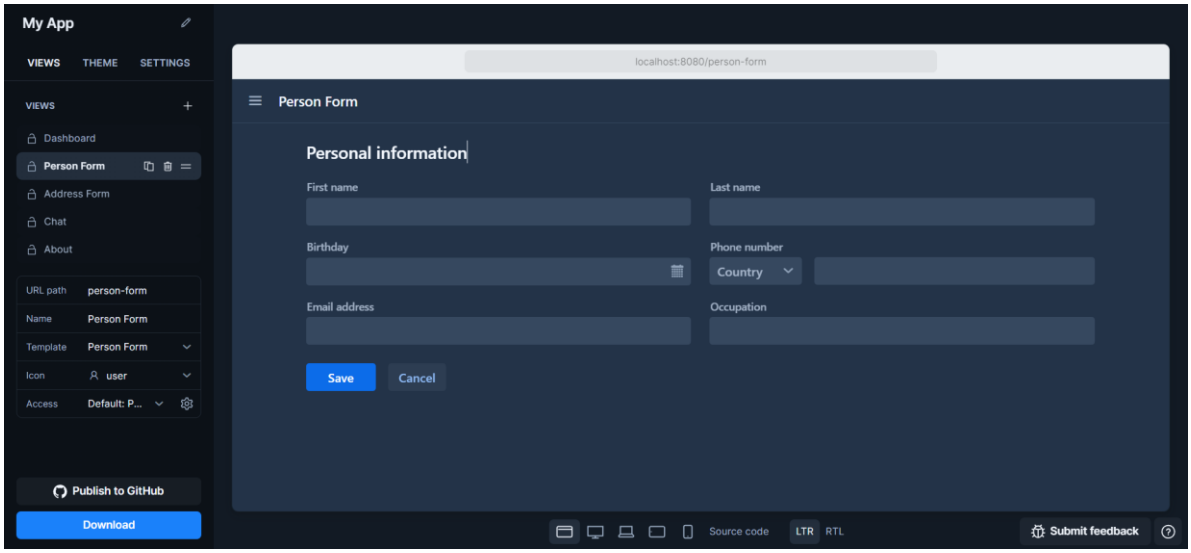
3.4 Vaadin

Vaadin je open-source webový framework pro vytváření moderních webových aplikací v jazyce Java bez použití HTML a Javascriptu. Jeho první verze byla vydána v roce 2002, velkou revolucí pak prošel v prosinci roku 2021, kdy byla z původní verze Vaadin 14 přeskočeno na verzi Vaadin 22.

Jedná se tzv. full-stack framework, což znamená, že je pomocí něj lze vyvinout jak frontendovou část, tak backend. Tento framework v základu využívá kombinace Spring Boot pro serverovou část a Vaadin Flow pro klientskou část. Umožňuje tak vyvíjet webové aplikace stejným způsobem jako jsou vývojáři zvyklí vyvíjet desktopové aplikace. Celý framework je postaven na komponentách. Aplikace vyvinuté pomocí tohoto frameworku, jsou následně vygenerovány běžným způsobem do standardního HTML a běží ve všech moderních prohlížečích i zařízeních.

Vaadin je naprogramován se zabezpečenou architekturou. Veškerá logika uživatelského rozhraní je umístěna na straně serveru. Veřejnosti je vystaven pouze jeden zabezpečený endpoint, což omezuje prostor útoku na minimum. Framework je schopen zajistit validaci vstupních dat na straně serveru, a to jak pomocí předpřipravených validátorů, tak těch naprogramovaných, což zabraňuje útokům ze strany klienta. Vaadin také nabízí nástroj Vaadin TestBench, který umožňuje snadnou implementaci a automatizaci UI unit testů a end-to-end testů v jazyce Java.

Pro vytváření nových projektů lze použít jednoduché přesto komplexní webové rozhraní Vaadin Start. Zde se po otevření objeví možnosti nastavení celé aplikace spolu s jejím živým náhledem, jak lze vidět na Obrázek 7. Lze si zde před vytvořit všechny stránky webové aplikace, určit jejich název, cestu, možnost přístupu dle uživatelských rolí nebo obsah stránek díky předpřipraveným šablonám. Dále je k dispozici nastavení tématu celé aplikace, kde si je možno vybrat základní styly, barevnou paletu, typografii, velikost textu a řádkování. V poslední záložce se nachází konfigurace projektu. Vybrat si jde layout celé aplikace, verzi Vaadinu, verzi Javy, databázi, kterou budeme používat, případně nechat si vygenerovat soubory pro nasazování aplikace. Takto připravenou aplikaci si lze stáhnout nebo nechat nahrát na GitHub.



Obrázek 7: Vytvoření nové aplikace pomocí Vaadin Start

Ve své základní verzi nazvané Core je Vaadin volně dostupný. K dispozici je ale také verze Pro, která oproti základní verzi nabízí nástroj TestBench pro automatizované UI testování, přidává pokročilé UI komponenty, jako např. grafy nebo mapy, a také UI designer umožňující vytváření stránek pomocí techniky drag and drop. Tato verze je nabízena formou měsíčního předplatného za cenu 139 $/měsíc/uživatel. Pro studenty je k dispozici Student Developer program skrze přihlášení přes GitHub, který poskytuje studentům verzi Pro zdarma pro studijní účely.

Dále jsou nabízeny verze Prime a Ultimate obsahující komplexní nástroje, přizpůsobitelné služby a podporu pro organizací či firem. Cena těchto verzí je individuální na základě dohody a rozsahu zakoupených služeb.

3.4.1 Vaadin Design System

Vaadin Design System poskytuje UI komponenty, UX návrhové vzory a komplexní dokumentaci, které dohromady pomáhají vytvářet uživatelsky přívětivé aplikace. Skládá se z více než 45 moderních UI komponent, kde jsou například různé inputy, combo boxy, checkboxy nebo date pickery. V Pro verzi jsou k dispozici pokročilé komponenty, jako např. grafy, tabulky nebo rich text editor. Všechny komponenty jsou založeny na W3C Web Components standardu, splňují regulace EU a USA pro přístupnost spolu se standardem WCAG 2.1 AA. To umožňuje využívat webové aplikace uživatelům bez ohledu na jakékoli postižení, které mohou mít. Zároveň mají také responzivní vzhled, který je optimalizovaný jak pro desktop, tak pro dotykové obrazovky.

K dispozici jsou dvě základní témata Lumo a Material. Ty poskytují základní nastavení stylů celé aplikace, jako jsou typografie, základní barvy, velikosti textu nebo ikony. Zaručují tak jednotný vzhled celé aplikace. Tyto základní témata jsou snadno přizpůsobitelné bez širších znalostí CSS, díky jejich parametrizaci. Vybrat si lze také mezi světlou a tmavou variantou, popřípadě si lze doprogramovat přepínač mezi denním a nočním režimem.

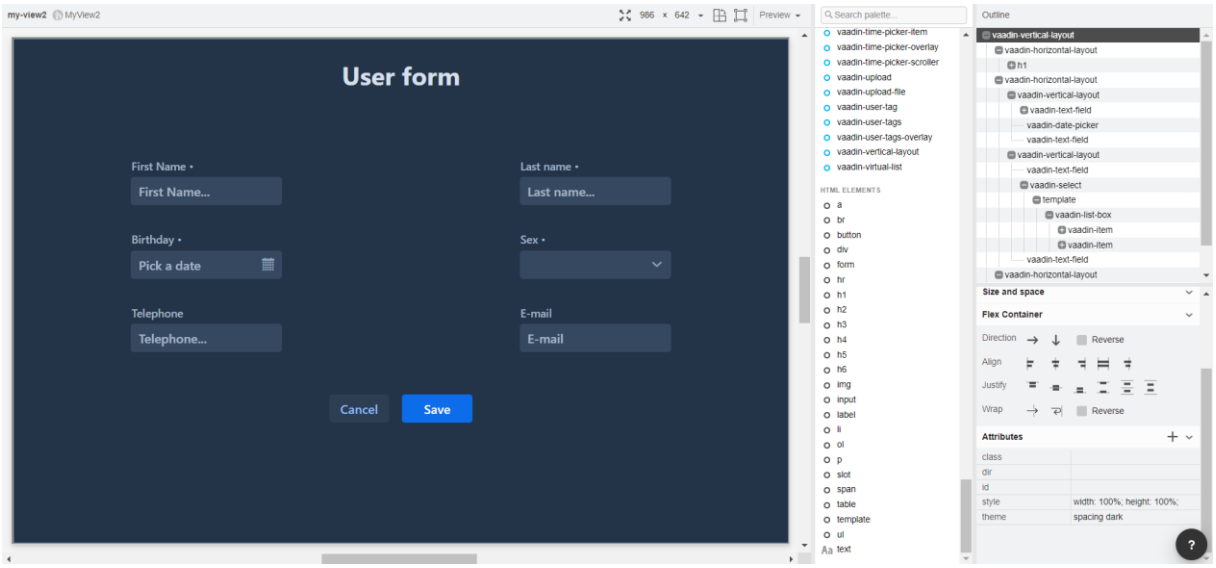
Pro designéry webových stránek je připraven soubor všech komponent a stylů v populárním nástroji pro návrh webových stránek Figma. To zaručuje, že jak návrháři, tak i vývojáři budou pracovat se stejným základem.

3.4.2 Vaadin Designer

Vizuální nástroj pro interaktivní vytváření jednotlivých stránek pomocí jednoduchého drag and drop rozhraní. K dispozici je vždy canvas (plátno), na které lze přetahovat jednotlivé komponenty, upravovat jejich vlastnosti a celkové rozložení na stránce. Výsledkem je HTML šablona, která je navázaná na třídu v Javě, kterou lze snadno přidat do aplikace a dále s ní pracovat.

Komponenty jsou uspořádány do stromové struktury a ke každé komponentě lze přiřadit ID, díky kterému s ní lze následně pracovat v kódu a naplnit ji například daty z databáze. Vše funguje na principu WYSIWYG neboli „*Co vidíš, to dostaneš*“. Veškeré změny, které se v designeru provedou jsou okamžitě promítnuty do kódu. K dispozici je také okamžitý náhled, jak bude vzhled vypadat na mobilních zařízeních.

Vaadin Designer je dostupný jako doplněk do IDE pro InteliJ IDEA a Eclipse. Je možné si jej zobrazit přímo v IDE, tak i ve webovém prohlížeči. K dispozici je pouze v placené verzi Pro, která byla popsána dříve. Ukázka tohoto doplňku je na následujícím Obrázek 8.



Obrázek 8: Vaadin Designer

Tento doplněk nakonec při vývoji aplikace využit nebyl z důvodu jeho nestability, kdy se při testování často dlouho načítal či padal. Celá aplikace je tak postavena na třídách rozšiřujících různé předdefinované layouty a využívání různých Vaadin komponent.

3.5 MySQL

Pro primární ukládání dat se jako databáze bude pro aplikaci využita relační databáze MySQL. Jedná se o jednu z nejpopulárnějších open-source databází na trhu. Nabízí jednoduchost, rychlost, spolehlivost a také vysokou dostupnost. Díky těmto vlastnostem, její popularitě, a ne příliš složitému navrhovanému datovému modelu se tento databázový systém hodí pro vyvíjenou webovou aplikaci.

Jak již z názvu vyplývá, pro přístup do databáze se využívá dotazovací jazyk SQL. Ten ale v aplikaci není třeba vůbec používat, díky frameworku Spring Boot a Hibernate, které zajišťují objektově relační mapování. Data jsou pak ukládána ve formě tabulek, které jsou vytvořeny na základě vytvořených datových entit.

3.6 Amazon S3 Cloud

Amazon S3 neboli Simple Storage Service je veřejné cloudové uložiště pro ukládání různého druhu dat. Poskytuje objektové úložiště, kde jsou data ukládána v objektech, které vždy uchovávají krom samotných dat, také jejich metadata a jedinečný identifikátor. Každý objekt musí být uložen v tzv. bucketu. Ty se podobají složkám a lze taky soubory jednoduše třídit a řádit k nim přístup.

S3 nabízí vysokou dostupnost a možnost škálování výkonu, bezpečnost díky automatickému šifrování a možnostem určovat jednotlivá oprávnění jak pro uživatele, objekty i buckety a také vysokou trvanlivost dat, která zaručuje, že data nebudou ztracena.

Využití této služby je možné např. pro ukládání a zálohování dat, archivaci dat za nízkou cenu, vytvoření tzv. Data lake či pro cloud-native aplikace. Základní verze s omezenými funkcemi je dostupná zdarma, plná verze se všemi funkcemi je následně placená na základě měsíčního předplatného. To není vždy stejné, ale odvíjí se od objemu využívaných dat.

3.7 Mailgun

Mailgun je služba, která nabízí různé služby v oblasti e-mailů, především pak odesílání a příjímání e-mailových zpráv. Je dostupné skrze jednoduše přístupné programové API rozhraní, díky němuž ji snadno mohou využívat vývojáři ve svých projektech a aplikacích. Krom API jsou také dostupné oficiální knihovny pro práci s Mailgunem pro vybrané jazyky jako jsou např. Java, JavaScript, PHP nebo Ruby.

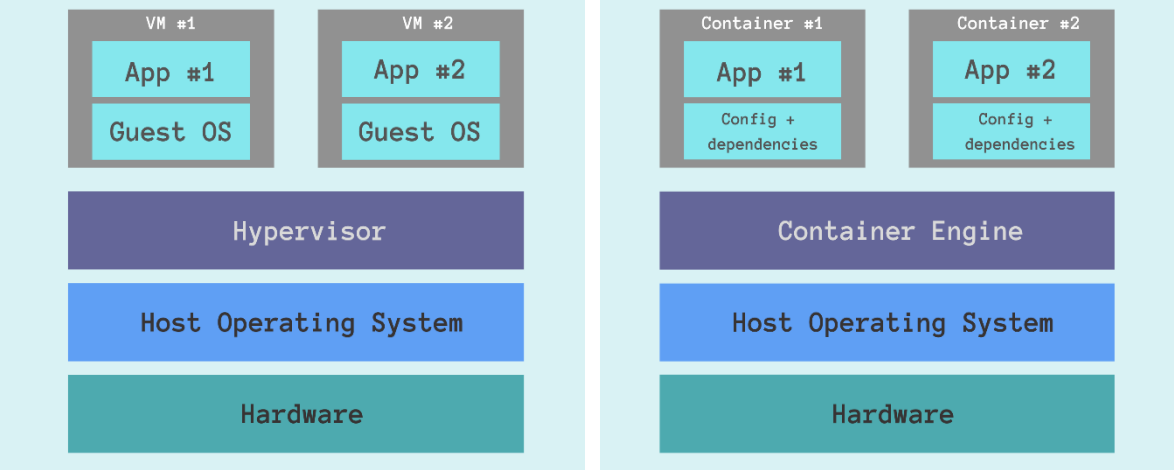
Mimo to také nabízí sledování doručení jednotlivých e-mailů, a to včetně toho kde zpráva skončí v klientově schránce, jestli ve složce doručených zpráv či ve spamu. Na tomto základě pak dokáže zobrazit statistiky a analýzu. Dále pak zasílání hromadných e-mailových zpráv a s tím spojená správa e-mailových front. Mailgun také dokáže ověřit platnost e-mailové adresy a další podrobnější informace, jako např. zda není daná adresa nebezpečná či se nepoužívá k podvodným účelům.

3.8 Docker

Docker je open-source platforma, určená pro vývoj, distribuci a běh aplikací. Poskytuje možnost zabalení aplikaci a její následné spuštění v izolovaném prostředí, které se označuje jako kontejner. Díky této izolaci a oddělení lze na cílovém systému spouštět více kontejnerů současně.

Kontejner je samostatně stojící, odlehčená softwarová jednotka, která obsahuje pouze věci skutečně potřebné pro jeho běh aplikace, jako jsou její kód, běhové prostředí, závislosti či knihovny. Díky tomu jsou tak kontejnery zcela odstíněny od hostitelského systému, nezáleží na tom, zda se jedná o lokální počítač, fyzický či virtuální stroj nebo u poskytovatele cloudových služeb. Kontejnery tak lze snadno sdílet s jistotou, že vždy a všude budou fungovat a pracovat stejným způsobem.

Docker kontejnery tak nabízí výhody izolace a alokace zdrojů, stejně jako virtuální stroje, fungují ale na odlišném principu. VM jsou abstrakcí na fyzické vrstvě a každá taková VM pak obsahuje svůj vlastní OS, binární soubory, knihovny atd., které pak mohou zabírat zbytečně místo. Kontejnerizace funguje pouze nad aplikační vrstvou, kde se virtualizuje operační systém. Kontejnery pak běží jako izolované procesy a sdílejí mezi sebou jádro operačního systému. Díky tomu zabírají daleko méně prostoru na disku, jsou efektivnější a snadno přenositelnější. Porovnání lze vidět na Obrázek 9.



Obrázek 9: Porovnání VM (vlevo) a kontejneru

Docker má architekturu postavenou na principu klient-server. Uživatel vždy komunikuje skrze klienta s tzv. *Docker deamon*. Ten může běžet na lokálním systému, případně se lze připojit k vzdálenému systému skrze REST API, přes UNIX sockety nebo webové rozhraní.

Docker démon se stará o správu, sestavování, běh a distribuci jednotlivých kontejnerů. Na starost má ale také další objekty, jako např. images, volumes nebo networks. Images představují šablony pro vytváření kontejnerů. Tyto image je možno stahovat i vytvářet vlastní, pomocí *Dockerfile,* který pomocí jednoduché syntaxe popisuje, jak mají vznikat. Kontejner je poté spustitelná instance tohoto image. Díky sítím spolu mohou jednotlivé kontejnery bezpečně komunikovat. Volumes zase slouží k uchovávání dat mimo kontejnery, což zajišťuje perzistenci dat a jejich možné sdílení mezi více kontejnery.

Součástí celé architektury je také uložiště Docker registry, které umožňuje ukládat a stahovat Docker images. Veřejný registr se nazývá Docker Hub a je v něm volně k dispozici velké množství již vytvořených obrazů, jako např. různé operační systémy, databáze nebo vývojová prostředí. Lze si založit i své privátní uložiště, kde se budou ukládat soukromé obrazy.

4 ANALÝZA A NÁVRH

Tato část diplomové práce se zabývá analýzou a návrhem vlastní webové aplikace, která bude poskytovat správu revizí. Tato část vývoje je klíčovým krokem při vývoji jakéhokoli softwarového projektu a není radno ji podceňovat. Součástí této fáze je především identifikace a sběr požadavků, které slouží pro správné porozumění potřeb požadovaných vlastností. Dalším krokem je vytvoření návrhu datových entit a vazeb mezi nimi, které dohromady tvoří datový model. Dalším krokem je navrhnout uživatelské rozhraní, které bude intuitivní a snadno použitelné pro uživatele a bude splňovat všechny požadavky.

Cílem této fáze vývoje je získat jasný obraz o tom, jak bude výsledná webová aplikace fungovat, jaké hlavní funkcionality bude poskytovat a jak bude vypadat.

4.1 Sběr Požadavků

Sběr požadavků by měl být první fází vývoje softwaru. Jedná se o důležitou fázi, která by nám měla poskytnout pohled na to, co by všechno měla aplikace umět, jakým způsobem by měla fungovat a jak se bude chovat při různých situacích. Důležité je tuto fázi nepodcenit, protože při nedostatečné definici požadavků či jejich změně v pozdějších fázích vývoje může vést k prodloužení, prodražení, nedodržení termínu či dokonce nezdaru celého projektu.

Mezi základní techniky sběru požadavků můžeme zařadit analýzu zadání, osobní konzultace se zadavatelem a budoucími uživateli, modelování případů užití případně vytváření prototypů. Při vytváření požadavků by se mělo dbát na jejich jasnost a srozumitelnost, díky použití jednoduchého jazyka srozumitelného pro všechny strany. Při popisu by měly být požadavky strukturovány do tabulky či stromové struktury, každý požadavek by měl být opatřen unikátním identifikátorem a obsahovat klíčové slovo *bude*. Požadavky by měly být také proveditelné a jejich výsledek by mělo být možné otestovat. Požadavky můžeme také rozdělit na dva základní typy. Funkční požadavky popisují vždy jaké věci by měl systém umět. Nefunkční požadavky naopak popisují vlastnosti systému či jeho omezení.

Sběr požadavků pro implementaci vlastní aplikace nejprve probíhal ze zadání diplomové práce a následně konzultací s vedoucím práce. Díky této kombinace tak bylo možné si udělat obrázek o požadovaných funkcích a vlastnostech systému, které bude potřeba následně implementovat. Výstupem by měl být přehledný seznam jednotlivých požadavků. Během vývoje pak po konzultacích docházelo k lehkým úpravám případně k doplnění dalších požadavků. Ale díky dostatečně zpracovaným základní struktuře požadavků nebyl žádný problém některé funkcionality upravit případně doplnit.

4.1.1 Funkční požadavky

Mezi základní funkční požadavky se řadí autentizace a autorizace uživatele, tak aby každý uživatel měl přístup pouze ke svým datům a nemohl upravovat data cizí. Každý uživatel také bude mít možnost upravovat si své údaje a nastavení aplikace. Platnost běžného uživatelského účtu po registraci by měla být 30 dní, případné prodloužení poté může nastavit administrátor.

Dále se požadavky vztahují především k revizím, které jsou základem celého systému. Uživatel by měl krom základních operací jako jsou vytvoření, úprava a mazání, mít možnost v revizích vyhledávat a filtrovat si je, ukládat různé typy souborů či přiřazovat revize klientům. Aplikace by také měla umět upozornit uživatele na blížící se termín revize, a to v předstihu, jaký si uživatel nastaví ve svém profilu. Termíny revizí by mělo být také možné přehledně zobrazit v kalendáři.

Mezi další požadavky se pak řadí vytváření a ukládání klientů, pro které budou revize vyřizovány. Při vytváření klientů také přišlo na požadavek načítání dat klientů z veřejného registru ARES dle zadaného IČO klienta. Seznam všech funkčních požadavků je v následující Tabulka 4.

Tabulka 4: Seznam funkčních požadavků

FR01	Systém bude umožňovat přístup pouze autentizovaným uživatelům
FR02	Systém bude umožňovat autorizaci uživatelů na základě přidělených rolí
FR03	Systém bude umožňovat registraci uživatele
FR04	Systém bude umožňovat obnovu zapomenutého hesla
FR05	Systém bude umožňovat vlastní nastavení uživatele
FR06	Systém bude umožňovat zadat datum expirace uživatelského účtu
FR07	Systém bude umožňovat spravovat uživatele
FR08	Systém bude umožňovat zablokovat uživatele
FR09	Systém bude umožňovat vytvářet a ukládat klienty pro revize
FR10	Systém bude umožňovat upravovat klienty
FR11	Systém bude umět načítat klienty dle IČO z veřejně dostupných dat
FR12	Systém bude umožňovat vytvářet a ukládat revize
FR13	Systém bude umožňovat vyhledávání a filtraci revizí na základě různých kritérií
FR14	Systém bude umožňovat upravovat revize
FR15	Systém bude umožňovat hromadné úpravy a mazání revizí
FR16	Systém bude upozorňovat předem na blížící se vypršení platnosti revize dle vlastního nastavení uživatele
FR17	Systém bude upozorňovat e-mailem uživatele na blížící se termíny revize
FR18	Systém bude upozorňovat e-mailem uživatele na vytvoření nové revize
FR19	Systém bude umožňovat ukládat soubory k revizím
FR20	Systém bude umožňovat stahovat uložené soubory
FR21	Systém bude umožňovat vytvářet tagy k revizím
FR22	Systém bude umožňovat přiřazovat více tagů k revizím
FR23	Systém bude umožňovat zobrazovat termíny revizí v kalendáři
FR24	Systém bude umožňovat uchovávat soubory k revizím
FR25	Systém bude umožňovat administrátorům správu uživatelů
FR26	Systém bude umožňovat administrátorům správu revizních tagů

4.1.2 Nefunkční požadavky

Nefunkční požadavky vycházejí především ze zadání diplomové práce. Z něj vyplývá, že vlastní implementace aplikace pro správu revizí by měla být ve formě webové aplikace. Pro implementaci by měly být využity technologie Java, Spring Boot a Vaadin. Z dalších konzultací poté vznikly na požadavky pro ukládání dat s využitím relační databáze MySQL a využití cloudové služby Amazon S3 Cloud pro ukládání souborů. Dále by také aplikace měla mít responzivní design pro pohodlné používání. Aplikace by měla být připravená na nasazení na cloudovou platformu Heroku a vytvoření kontejneru pro Docker. Seznam všech nefunkčních požadavků je v Tabulka 5.

Tabulka 5 Seznam nefunkčních požadavků

NR01	Systém bude dostupný ve formě webové aplikace
NR02	Systém bude multiplatformní
NR03	Systém bude umožňovat přístup více uživatelů v jeden moment
NR04	Systém bude naprogramován v jazyce Java
NR05	Systém bude využívat framework Vaadin
NR06	Systém bude využívat pro ukládání dat databázi MySQL
NR07	Systém bude využívat službu Amazon S3 Cloud pro ukládání souborů
NR08	Systém bude využívat Administrativní registr ekonomických subjektů (ARES)
NR09	Systém bude mít responzivní design
NR10	Systém bude nasazen na cloudové platformě Heroku
NR11	Systém bude možno nasadit jako kontejner na platformu Docker

4.2 Entity a Datový model

Po dokončení sběru požadavků přichází na řadu vytvoření datového modelu, který představuje datovou strukturu aplikace a bude následně využit pro tvorbu databáze. Nejdříve je nutné nalézt všechny datové entity a jejich atributy, které je potřeba uchovávat. Dalším krokem by mělo být následně nalezení vztahů mezi entitami. Základní návrh je důležité mít ještě před začátkem vývoje z důvodu lepší přehlednosti o fungování aplikace a omezení výskytu případných problémů a chyb.

Po vytvoření návrhu modelu jsou poté jednotlivé entity vytvořeny pomocí rozhraní JPA a technologie Hibernate, které zajistí objektově relační mapování. Díky tomu tak při vývoji nejsme závislí na konkrétní databázi a případné malé úpravy modelu, lze provádět i během vývoje.

4.2.1 Revize

Základní entitou celé aplikace je Revision. Ta by měla představovat jednotlivé revize, které jsou základem celé aplikace. Každá revize by měla uchovávat základní údaje jako název, popis, stav, datum revize, datum vypršení platnosti, seznam kontaktních e-mailů, příznak, zda byla revize vyfakturována a případný popis nalezených pochybení. Dále revize, jako i další entity, ukládá důležité informace o datumech vytvoření a poslední úpravy a uživatelích, kteří tyto akce provedly pomocí vazby na entitu User.

Každá Revize má vazbu na entitu File. Ta ukládá metadata o souborech, které jsou uložené u revizí. Skutečný soubor je ukládán mimo databázi, z důvodu vyšší rychlosti, škálovatelnosti

a snadnější správě. Jelikož soubor je uložen vždy k jediné revize je tato vazba realizována jako 1:N. Další důležitou vazbou je vazba M:N na entitu RevisionTag, který je realizován pomocí vazební entity. Ta zajišťuje ukládání jednotlivých tagů, které označují revize. Mezi další vazby patří vazby na entity Client a Address, které jsou představeny později.

4.2.2 Uživatel a role

Druhou klíčovou entitou je entita User, která uchovává informace o jednotlivých uživatelských účtech. Ke každému uživateli se ukládají jeho základní informace jako jméno, příjmení, e-mail, uživatelské jméno a hash z uživatelova hesla. Další doplňující informace, které se ukládají je datum expirace účtu, po jehož vypršení nebude mít dále uživatel přístup k aplikaci, a informace o tom kolik dní, před vypršením revize chce být uživatel informován a případný vlastní text e-mailu, kterým jsou uživatelé upozorněni.

Entita uživatel je dále navázána na role, ve které se ukládají jednotlivé role, které jsou přiřazeny k uživatelům. Každému uživateli lze přidělit více než jednu roli. Na základě přidělených rolí je poté uživatelům omezen nebo povolen přístup k některým funkcionalitám.

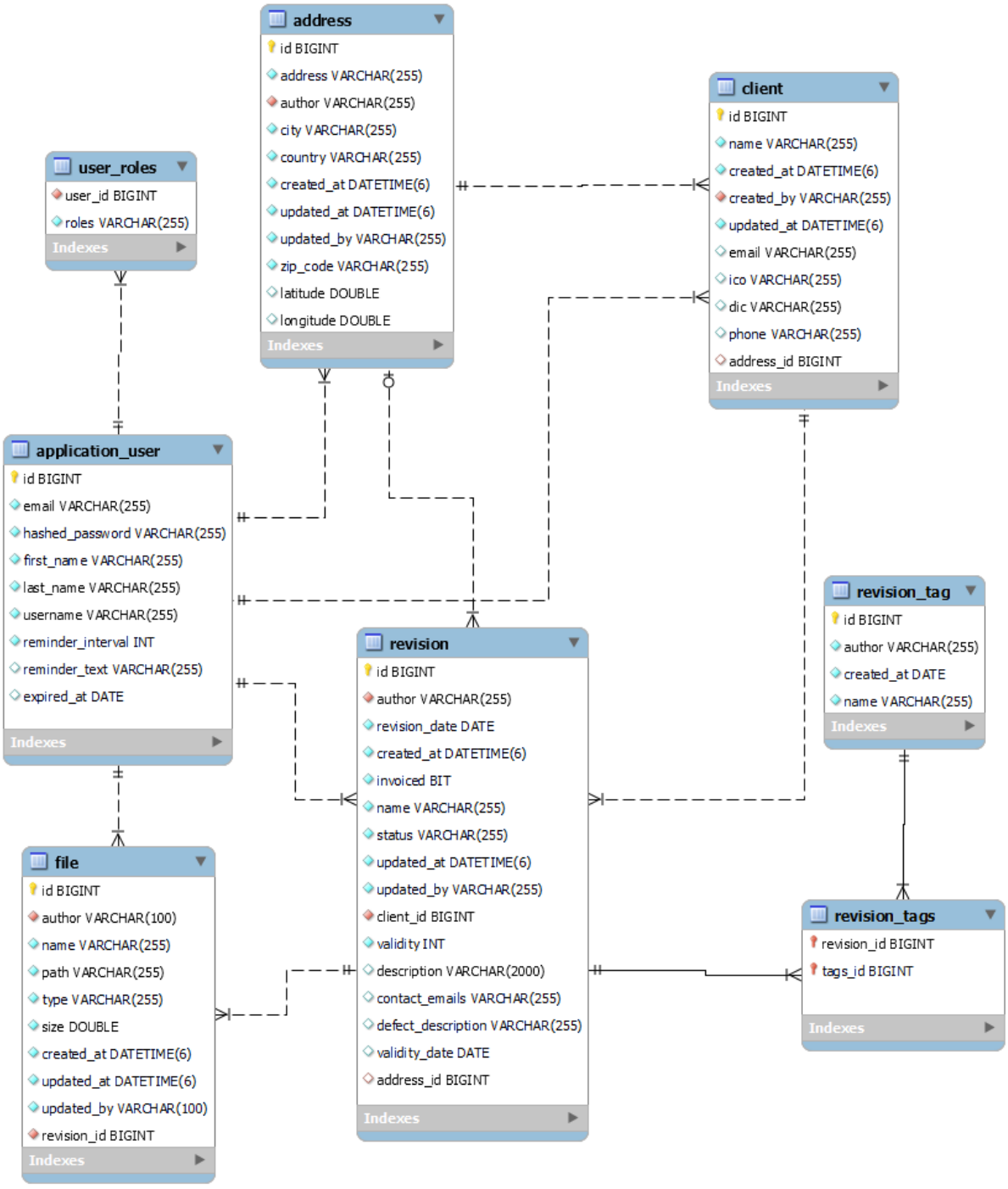
4.2.3 Klient a adresa

Další vybranou entitou je Client. Ta představuje klienty, pro které jsou revize prováděny. U klientů se ukládá jejich název, IČO, DIČ, adresa, e-mail a telefon. Při vytváření klientů se počítá s pozdějším napojení na ARES, které bude automaticky tyto údaje načítat z veřejných zdrojů pomocí zadaného IČO. Další doplňkové informace, které se ukládají jsou informace o uživatelích a datumech vytvoření a poslední úpravy.

Jelikož se adresa využívá také v revizích je pro ni vytvořena samostatná entita, která ukládá informace o adrese a zeměpisných souřadnicích, které se využívají pro zobrazovaní bodu na mapě. Tato entita je následně napojena vazbou na entity Client i Revision.

4.2.4 Konečný model

Finální model vychází, z již výše zmíněných entit a všech vazeb mezi nimi a nabízí celkový pohled na data, které bude aplikace využívat a ukládat. Celý jej lze viděn na Obrázek 10: Datový model.



Obrázek 10: Datový model

4.3 Návrh UI

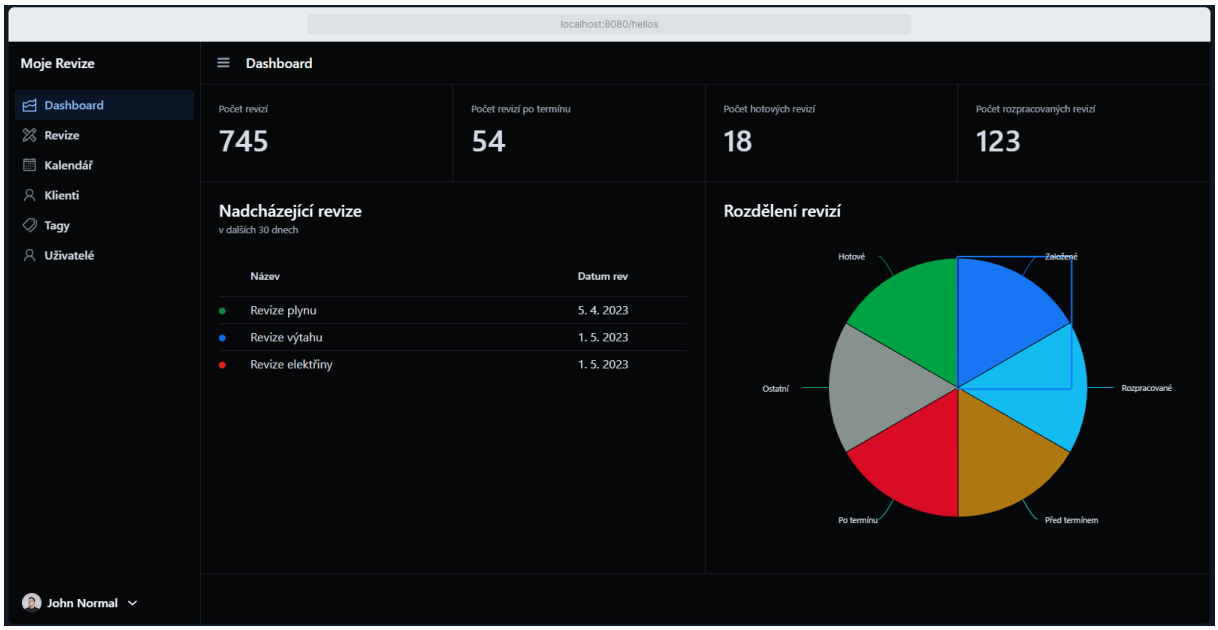
Poslední fází návrhu, bylo navržení uživatelského rozhraní. Výhodou frameworku Vaadin pro vývoj aplikace je že nabízí knihovnu se svými komponentami pro nástroj Figma, který je široce využíván mezi UX a UI návrháři. Díky ní lze vytvořit návrh, který pak vývojáři nebudou mít problém implementovat a mohou ušetřit spoustu času při převzetí navržených stylů.

Další výhodou použití frameworku Vaadin je, že nabízí široké množství vlastních komponent a před připravených stylů na kterých lze stavět. Případně není ani problém si již existující styly upravit podle sebe pomocí kaskádových stylů. Důležitou výhodou je využití proměnných v CSS. Díky nim lze jednoduše měnit na jednom místě např. základní barvy a odstíny, různé velikosti odsazení, fontů apod. Tyto změny se následně propíšou do celé aplikace.

Z důvodu nepřílišné složitosti aplikace a úspoře času při návrhu nebyl využit nástroj Figma, ale pouze nástroj Vaadin Start, který umožnuje základní návrh struktury celé aplikace, jeho stylů a nastavení. Další výhodou je, že po dokončení návrhu lze následně vygenerovat předpřipravený projekt na kterém lze začít ihned pracovat.

Při samotném návrhu byl nejprve vybrán název aplikace a její základní styl s bočním vysouvacím menu. Dále byl nastaven základní barevný styl aplikace a lehce upravena paleta používaných barvy. V aplikaci se počítá s přepínačem mezi tmavým a světlým režimem. Poté se postupně nastavila typografie, velikost ovládacích prvků a jejich odsazení, a nakonec zobrazení zakulacených okrajů a styl tlačítek a inputů.

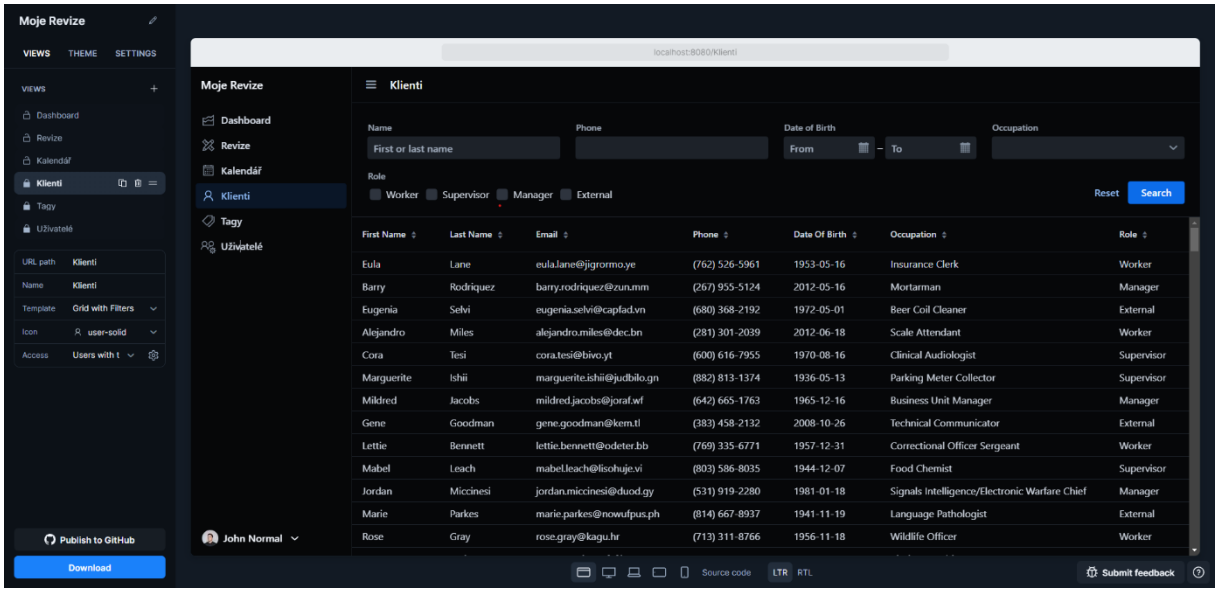
Následně byly vytvořeny jednotlivé položky v menu s ikonami, které budou představovat jednotlivé stránky aplikace. Úvodní stránkou po přihlášení je dashboard viz Obrázek 11, který bude nabízet základní informace o počtech jednotlivých revizí dle stavu. Dále by zde měl být seznam s nadcházejícími revizemi případně kalendář s aktuálním měsícem a přehledem revizí.



Obrázek 11: Návrh dashboardu

Druhou stěžejní stránkou budou revize. Zde bude k dispozici seznam všech revizí spolu s jejich nejdůležitějších údajů. K dispozici by měl také vyhledávací pole, různé inputy pro filtrování a hromadné akce. Ze seznamu se bude také možné prokliknout do detailu revize, který bude tvořen formulářem s možností revize upravovat. Z důvodu většího množství uchovávaných informací by měl být formulář rozdělen do více částí pro lepší přehlednost uživatelů.

Další stránkou bude dostupnou všem uživatelům jsou Klienti. Zde by se měl opět stejně jako u revizí zobrazovat seznam klientů i vyhledávací pole. Detail klienta je opět řešen pomocí formuláře. Součástí by měla také funkce automatického načítání klientů na základě IČO. V následujícím návrhu Obrázek 12 je pak vidět předpřipravený podrobný seznam, spolu s řádkem pro vyhledávání a filtraci. Ten bude ve výsledné aplikaci zredukován, jelikož nebude potřeba uchovávat ani zobrazovat všechny tyto informace.



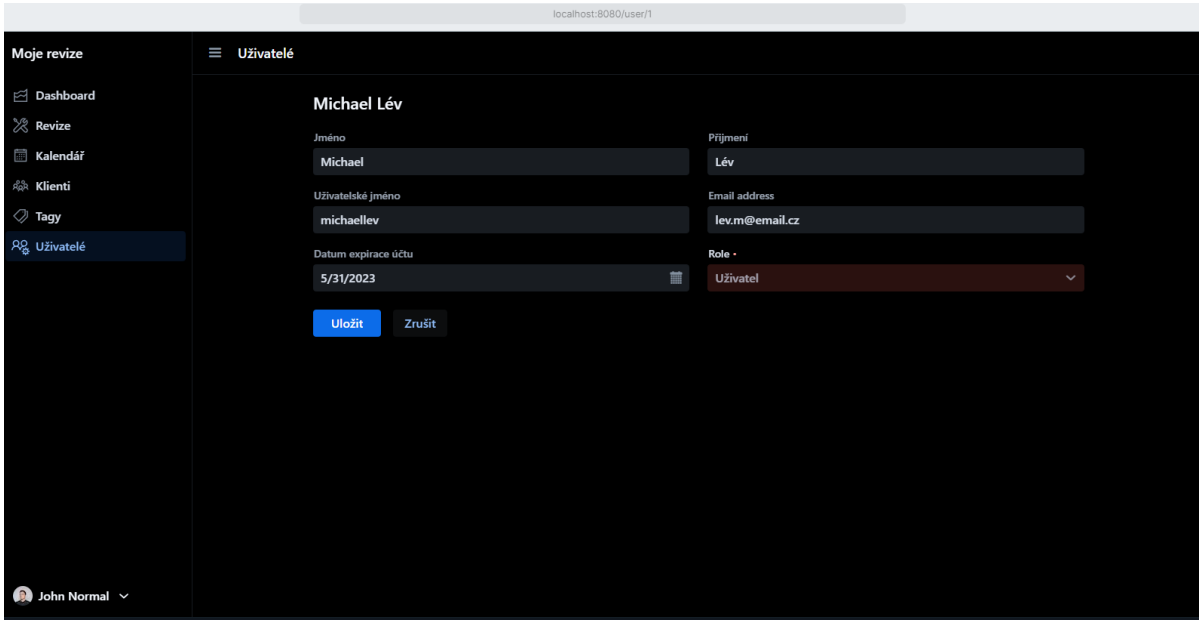
Obrázek 12: Návrh seznamu

Pro každého uživatele také bude k dispozici stránka se s vlastním profilem, kde si budou uživatele moct nastavit osobní údaje, interval pro upozorňování na revize a vlastní text upozornění. Pro text upozornění bude k dispozici tzv. rich text editor, díky kterému si uživatelé budou moct sami nastylovat e-mail.

Poslední dostupnou stránkou běžným uživatelům je kalendář. Ten bude sloužit pro přehledné zobrazení termínů revizí v čase. Budou se zobrazovat jak konání revizí, tak datumy konce platnosti revizí. Vaadin nenabízí vlastní komponentu pro kalendář, proto bude muset být vytvořena vlastní komponenta, případně využita komponenta třetích stran, které Vaadin nabízí ve vlastním adresáři na webu a umožňuje je přidávat do projektu.

Další stránky jsou již pouze pro administrátory. Výhodou této webové aplikace je, že lze omezení nastavit již přímo při návrhu, a to pomocí jednoduchého výběru. První takovou stránkou bude stránka Tagy, která nabízí jednoduchou správu Tagů, které bude možno vytvářet, upravovat či mazat. Druhou stránkou je stránka uživatelé, sloužící pro správu uživatelů. Opět zde bude seznam uživatelů s nejdůležitějšími informacemi. Navíc zde bude možnost jednotlivé uživatele zablokovat, případně jim prodloužit platnost účtu.

V následujícím návrhu Obrázek 13 je pak vidět formulář pro zobrazení a úpravu detailu uživatele, pomocí něj se budou dát upravit jednotlivá data.

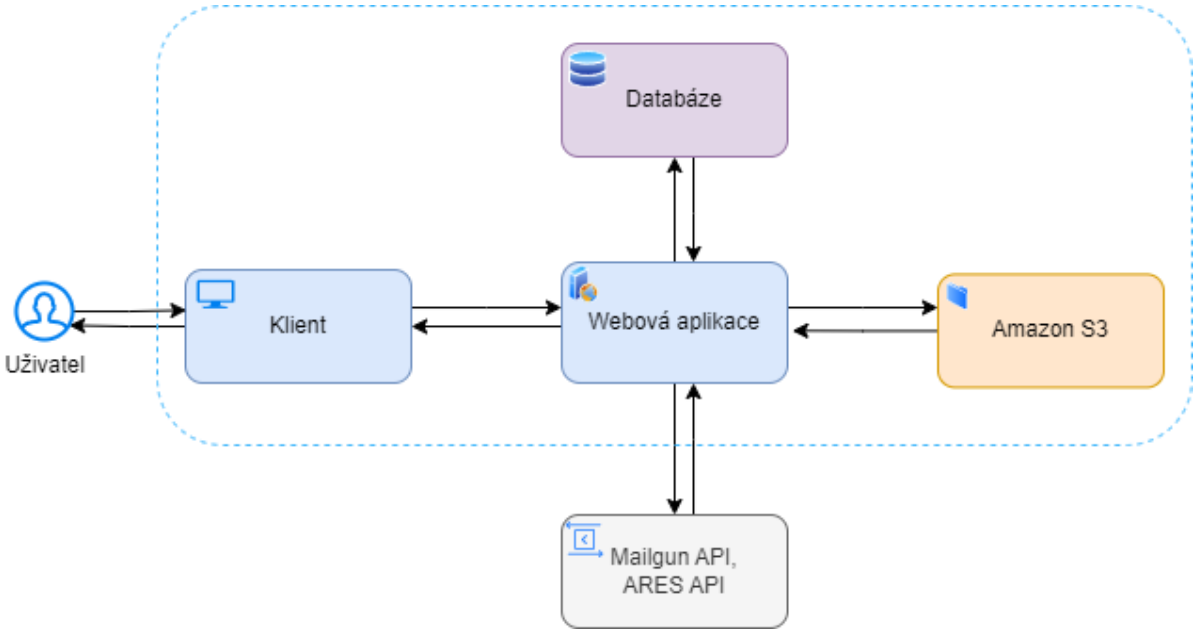


Obrázek 13: Návrh detailu uživatele

5 IMPLEMENTACE

Tato kapitola bude věnována samotné implementaci aplikace pro správu revizí. Ta je vytvořena především pomocí jazyka Java a frameworku Vaadin ve verzi 23.3.6. Jako balíčkovací nástroj, který zajišťuje správu a řízení sestavování aplikace, byl vybrán Maven.

Z následujícího diagramu na Obrázek 14 je pak patrné jednoduché schéma aplikace. Základem celého programu je webová aplikace, která obsahuje veškerou logiku a řídí komunikaci v celé aplikaci. Pro ukládání a načítání dat je tato aplikace napojena na relační databázi, kde jsou všechny data uchovávána. K ukládání a stahovaní souborů poté slouží služba Amazon S3, kde jsou jednotlivé soubory uchovávány v cloudu a lze k nim snadno řídit přístup. Dále webová aplikace také komunikuje skrze API s dalšími aplikacemi a službami třetích stran, jako jsou např. Mailgun pro zasílání e-mailů nebo ARES pro načítání dat o klientech. Poslední částí, se kterou webová aplikace komunikuje, je pak samotný klient, který poskytuje uživatelské rozhraní. To je následně k dispozici běžným uživatelům, kteří s ním mohou dále pracovat a zadávat skrze něj své vstupy a požadavky.



Obrázek 14: Architektura aplikace

5.1 Struktura projektu

Díky návrhu uživatelského rozhraní pomocí webové aplikace Vaadin Start je již k dispozici projekt s předpřipravenou základní strukturou. Ta v sobě obsahuje všechny potřebné konfigurační soubory a nastavení pro běh aplikace, dále také balíček *frontend*, ve které se nachází všechny použité CSS styly a případně také vlastní designy vytvořené ve Vaadin Designeru, a balíček *src*, obsahující všechny zdrojové kódy a veškeré funkce a logiku celé aplikace.

Vývoj následně probíhá v tomto projektu v balíčku *src* a jeho podřízených balíčcích, které se řídí standardní strukturou projektu Java. Základní rozdělení je určeno čtyřmi balíčky, které se následně uvnitř dělí podrobnějším způsobem. Toto rozdělení bylo vytvořeno vždy na základě části aplikace, se kterou se pracuje.

Obsah kořenového balíčku, obsahující také hlavní třídu Application, je pak následující:

- components – obsahuje sdílené komponenty,
- data – obsahuje další balíčky pro práci s daty,
    - dto – obsahuje objekty pro přenos dat,
    - entity – obsahuje datové entity,
    - enums – obsahuje výčtové typy,
    - I18N – obsahuje třídy pro českou lokalizaci komponent,
    - service – obsahuje servisy a rozhraní pro komunikaci s databází,
- security – obsahuje třídy zajišťující zabezpečení aplikace a konfiguraci,
- views – obsahuje jednotlivé stránky uživatelského rozhraní,
    - calendar – stránka s kalendářem,
    - client – stránky se seznamem klientů a jejich detail,
    - dashboard – úvodní stránka se základními informacemi,
    - error – chybová stránka při nenalezení cesty či záznamu,
    - login – přihlašovací stránka,
    - profile – stránka s osobním nastavením uživatele,
    - register – stránka pro registraci uživatele,
    - revision – stránky se seznamem revizí a jejich detail,
    - revisiontag – stránky se seznamem tagů a jejich detail,
    - user – stránky se seznamem uživatelů a jejich detail.

5.2 Přístup k jednotlivým stránkám

Framework Vaadin nabízí zabezpečení aplikace ve spolupráci se Spring Boot a Spring Security. Díky vestavěným doplňkům v tomto frameworku se usnadňuje, zrychluje a zvyšuje zabezpečení aplikace v porovnání s přímým používáním Spring Security. Základním jsou anotace, které lze přiřazovat jednotlivým view a na jejich základě je poté uživatel přístup na stránku povolen nebo zamítnut. Použít lze anotace:

- @AnonymousAllowed – přístup umožněn všem uživatelům,
- @PermitAll – přístup pouze přihlášených uživatelů,
- @RolesAllowed – přístup povolen pouze určeným rolím,
- @DenyAll – přístup zamezen všem. Toto nastavení je defaultní pro všechny view, které neobsahují žádnou anotaci.

V příkladu Zdrojový kód 1 obsahujícím definici třídy s formulářem pro zobrazování a úpravu detailu uživatele, lze vidět využití anotace, která omezuje přístup pouze uživatelům s rolí Admin. Dále jsou také vidět anotace určující titulek stránky a cestu na které je daná stránka dostupná.

```
@PageTitle("Uživatel")
@Route(value = "user/:userId", layout = MainLayout.class)
@RolesAllowed("ADMIN")
public class UserDetailView extends Main implements BeforeEnterObserver {
```

Zdrojový kód 1: Ukázka anotací u třídy UserDetailView

5.3 Grid

Pro implementaci různých seznamů v aplikaci byla zvolena komponenta z knihovny Vaadin nazvaná Grid. Ta slouží k zobrazování tabulkových dat, podporuje různé řazení, filtraci, různé nastavení jednotlivých sloupců nebo lazy-loading. Její nevýhodou může být, že se automaticky vzhled nepřizpůsobuje automaticky velikosti displeje, a tak je nutné si dát pozor na zobrazení na tabletech a mobilních telefonech.

Při implementaci byla využita vlastnost lazy-loading, která umožňuje načítání položek seznamu až když jsou skutečně potřeba. Při velkém počtu položek tak není zpomaleno načítání a renderování celé komponenty a není tak třeba řešit případné stránkování, protože se položky načítají a zobrazují postupně jak uživatel scrolluje seznamem postupně níže a níže.

Pro jeho implementaci se využívají třídy *DataProvider*, které dědí ze třídy *AbstractBackEndDataProvider* s dvěma generickými parametry, a to vždy entitou a třídou, která zajišťuje filtrování položek. V ní je následně přepisována metoda *fetchFromBeckEnd()*, která zajišťuje načtení seznamu dle uživatele, následné vyfiltrování a seřazení položek podle uživatelského nastavení a navrácení pouze požadovaných položek. Při běhu aplikace pak komponenta Grid automaticky detekuje, že je potřeba načíst další data a v callbacku volá tuto metodu s parametrem typu Query, který obsahuje všechny potřebné informace pro načtení pouze potřebných dat. Ukázka implementace této třídy je vidět v příkladu Zdrojový kód 2.

```
public class ClientDataProvider
            extends AbstractBackEndDataProvider<Client, ClientFilter> {

private final ClientService clientService;

@Override
protected Stream<Client> fetchFromBackEnd(Query<Client, ClientFilter>
  query) {
    Stream<Revision> stream =
          revisionService.findAll(PageRequest.of(
          query.getOffset() / RevisionGridHelper.PAGE_SIZE,
          RevisionGridHelper.PAGE_SIZE))
          .stream();

    // Filtering
    if (query.getFilter().isPresent()) {
        stream = stream.filter(revision ->
                          query.getFilter().get().test(revision));
    }
    // Sorting
    if (query.getSortOrders().size() > 0) {
        return stream.sorted(sortComparator(query.getSortOrders()));
    }
    return stream.sorted(sortComparator(List.of(
        new QuerySortOrder("revisionDate", SortDirection.DESCENDING))));
}
```

Zdrojový kód 2: Ukázka ClienzDataProvider a lazy-loading

Dále v příkladu Zdrojový kód 3 se seznamem klientů je nejprve inicializována komponenta Grid, následně je v konstruktoru vytvořen objekt *ClientDataProvider*, z něj následně vytvořen *ConfigurableFilterDataProvider* pro možnost nastavování filtrů a v posledním kroku nastavení tohoto vytvořeného objektu do gridu pomocí metody *setItems()*.

Dále je pak v příkladu vidět nastavení samotného gridu. Nejprve je nastaven režim výběru a styl. Dále pak jsou nastaveny jednotlivé sloupečky. Pro přidání sloupečku jsou k dispozici dvě metody, a to *addColumn()*, která umí automaticky vytvořit sloupeček pomocí předání atributu, popřípadě může přebírat tzv. Lit Renderer. Ten umožňuje rychlé renderování obsahu, ale vyžaduje napsaní komponenty pomocí HTML. Druhou metodou je pak *addComponentColumn()*, která přijímá na vstupu Vaadin komponentu. Její napsání je tak snazší, ale renderování je naopak pomalejší, proto není příliš vhodné mít takto vytvořeno mnoho sloupců, pokud očekáváme spousty záznamů. V příkladu jsou použity oba způsoby pro vytvoření sloupců a dále jsou jim nastaveny další parametry jako název, klíč, tříditelnost či šířka. Po nastavení sloupečků je pak ještě nastaven listener na dvojklik, který uživatele přesune na stránku s vybraným klientem.

```
private final Grid<Client> grid = new Grid<>(Client.class, false);
private final ConfigurableFilterDataProvider<Client, Void, ClientFilter>
filterDataProvider;

public ClientGridView(ClientService clientService) {
 ClientDataProvider dataProvider = new ClientDataProvider(clientService);
 filterDataProvider = dataProvider.withConfigurableFilter();
 grid.setItems(filterDataProvider);
 setupGrid();
 ...
}

private void setupGrid() {
    grid.setSelectionMode(Grid.SelectionMode.NONE);
    grid.addThemeVariants(GridVariant.LUMO_NO_BORDER);

    grid.addColumn(createClientnNameRenderer())
        .setHeader("Klient")
        .setSortable(true)
        .setAutoWidth(true)
        .setFlexGrow(1)
        .setKey("name");
    grid.addColumn(Client::getIco)
        .setHeader("IČO")
        .setAutoWidth(true)
        .setSortable(true)
        .setKey("ico");
    grid.addColumn(new LocalDateTimeRenderer<>(
      Client::getUpdatedAt, "dd. MM. yyyy HH:mm"))
        .setHeader("Vytvořeno")
        .setSortable(true)
        .setAutoWidth(true)
        .setKey("createdAt");
    grid.addComponentColumn(this::getButtons)
        .setTextAlign(ColumnTextAlign.END)
        .setSortable(false)
        .setAutoWidth(true)
        .setFlexGrow(0);

    grid.addItemDoubleClickListener(e ->
              UI.getCurrent().navigate("client/" + e.getItem().getId()));

}
```

Zdrojový kód 3: Ukázka nastavení komponenty Grid

5.4 Formuláře

Další stěžejní částí aplikace jsou formuláře, které budou představeny v této kapitole. Pro vytváření formulářů je ve Vaadinu připravena speciální komponenta *FormLayout*, která umožnuje tvořit responzivní formuláře s různým rozložením inputů či popisků.

Do takto vytvořeného layoutu pak lze vkládat jednotlivé druhy inputů. Vaadin nabízí široké množství inputů. Mezi základní komponenty pro zadávání textu uživatelem jsou *TextField* a jeho variace *NumberField*, *TextArea*, *PasswordField* či *EmailField*, které poskytují další funkce nebo automaticky kontrolující zadávaný obsah. Dále je k dispozici např. komponenty *Select*, umožňující vybrat uživateli hodnotu ze seznamu, nebo *ComboBox* plnící podobnou funkci, ale s rozšiřujícími možnostmi jako jsou vybírání více hodnot, zadání vlastní hodnoty,

vyhledávání v hodnotách či lazy-loading. Pro zadávání datumu a času jsou k dispozici *DatePicker* a *TimePicker*. Základní inputy pak doplňují ještě *RadioButton* určený pro vybírání jedné z více možností a *Checkbox* reprezentující volba ano/ne.

V příkladu Zdrojový kód 4 jsou pak ukázány použití těchto inputů ve formuláři pro vytváření a úpravu revizí. Jednotlivé inputy jsou nejprve vytvořeny jako lokální proměnné, pro jejich pozdější použití v rámci data bindingu.

```
private final TextField name = new TextField("Název revize");
private final TextArea description = new TextArea("Popis");
private final DatePicker revisionDate = new DatePicker("Datum revize");
private final Select<RevisionValidity> revisionValidity = new Select<>();
private final RadioButtonGroup<String> invoiced =
                                        new RadioButtonGroup<>();
private final MultiSelectComboBox<RevisionTag> tags =
                                        new MultiSelectComboBox<>("Tagy");
private final RadioButtonGroup<RevisionStatus> status =
                                        new RadioButtonGroup<>("Stav revize");
```

Zdrojový kód 4: Ukázka inicializace formulářových polí

Následně jsou jednotlivé inputy vždy nastaveny dle potřeby. V následující ukázce kódu Zdrojový kód 5 jsou předvedeny metody, které se o toto nastavení starají. Jako první je nastavení *DatePickeru* pro zadávání datumu revize, kterému je nastavena česká lokalizace a příznak povinného pole. V další metodě je nastaven *Select* pro vybrání intervalu platnosti revize. Tomu je přiřazen popisek, následně jsou nastaveny hodnoty z enumu, ze kterých si bude uživatel vybírat a nastaven jakým způsobem se budou zobrazovat popisky těchto hodnot. V poslední ukázané metodě je pak nastavován *RadioButton* pro vybírání stavu revize. Možné stavy jsou nastaveny opět z enumu, podobně pak i generování popisků a indikátor povinnosti.

```
private void prepareDueDateField() {
    revisionDate.setI18n(new CalendarCzechI18N());
    revisionDate.setRequiredIndicatorVisible(true);
    revisionValidity.setRequired(true);
}

private void prepareRevisionValidityField() {
    revisionValidity.setLabel("Platnost revize");
    revisionValidity.setItems(RevisionValidity.values());
    revisionValidity.setItemLabelGenerator(RevisionValidity::getName);
    revisionValidity.setRequiredIndicatorVisible(true);
    revisionValidity.setRequired(true);
}

private void prepareStatusTextField() {
    status.setItems(RevisionStatus.values());
    status.setItemLabelGenerator(RevisionStatus::getStatus);
    status.setRequiredIndicatorVisible(true);
    status.setRequired(true);
}
```

Zdrojový kód 5: Ukázka nastavení formulářových prvků

V posledním příkladě Zdrojový kód 6 je pak vytvořen nový *FormLayout*, do kterého jsou přiřazeny všechny potřebné inputy. Nakonec je ještě nastaveno, při jaké šířce se má změnit rozložení ze dvou sloupců na jeden.

```
private FormLayout createDetailFormLayout() {
    FormLayout formLayout = new FormLayout();
    formLayout.setClassName("client-form");
    formLayout.add(name, status, revisionDate, revisionValidity,
                description, invoiced, tags, contactEmails, hasDefects);

    formLayout.setResponsiveSteps(
        new FormLayout.ResponsiveStep("0", 1),
        new FormLayout.ResponsiveStep("21em", 2));
    return formLayout;
}
```

Zdrojový kód 6: Ukázka vytvoření FormControl

5.5 Data Binding

Po vytvoření formulářů je následné tyto formulář napojit na business objekty, tak aby data mohla být ukládána a zpětně také načtena do formulářů. Tato technika se nazývá data binding a pro její implementaci je ve Vaadinu k dispozici třída *Binder*, která umožňuje určit, jak jsou jednotlivé atributy objektu navázány na zobrazovaná pole.

Dále jsou ve třídě *Binder* také k dispozici předpřipravené validátory, které automaticky kontrolují uživatelské vstupy. Definovat si lze snadno ale také vlastní validátory pomocí metody *withValidator()*. Jednotlivé validátory lze za sebou řetězit a samozřejmostí jsou také vlastní chybové hlášky.

Další funkcí, kterou *Binder* také nabízí jsou převod datových typů z formátu, který je uložen v příslušných objektech do formátu, který očekávají jednotlivé inputy a zpět. K tomu slouží metodu *withConverter()*, pomocí níž lze určit jak jednotlivé vstupy a výstupy převádět.

V následující ukázce Zdrojový kód 7 je předvedena metoda *addBindingAndValidation()*, která slouží k nabindovaní formuláře pro úpravu profilu uživatele na entitu User. Nejprve je vytvořen binder na základě User entity a následně jsou již napojovány jednotlivá pole. U polí je nejprve provedena validace, zda je v poli zadána hodnota, pomocí *asRequired()* s parametrem chybové hlášky. Dále u polí jako je e-mail nebo interval připomínky jsou využity vlastní validátory, které dále kontrolují zadaný vstup. U pole připomínky je také využit zmínění konvertor, který převádí celá čísla na desetinné a obráceně, z důvodu, že pole typu *NumberField* pracuje s typem Double, zatímco interval je ukládán jako celočíselný Integer. Následně je pak pro každé pole nabindované pomocí metody *bind()* a zadáním getterů a setterů z daného atributu entity.

Na konci ukázky pak můžeme vidět načtení dat do binderu, pomocí metody *readBean()* a nastavení listeneru na kliknutí na tlačítko uložit, při kterém se nejprve provede validace všech vstupů a poté je buď objekt uložen a zobrazena notifikace o uložení nebo jsou uživateli zobrazeny nalezené chyby a notifikace.

```
public void addBindingAndValidation(User user) {
    BeanValidationBinder<User> binder =
                                new BeanValidationBinder<>(User.class);

    binder.forField(profileView.getFirstName())
            .asRequired("Jméno musí být vyplněno.")
            .bind(User::getFirstName, User::setFirstName);

    binder.forField(profileView.getLastName())
            .asRequired("Příjmení musí být vyplněno.")
            .bind(User::getLastName, User::setLastName);

    binder.forField(profileView.getEmail())
            .asRequired("Email musí být vyplněn.")
            .withValidator(
                  new EmailValidator("Email není ve správném formátu."))
            .bind(User::getEmail, User::setEmail);

    binder.forField(profileView.getReminderInterval())
            .asRequired(
                  "Připomínka musí být nejpozději 1 den před termínem.")
            .withValidator(reminderInterval -> reminderInterval < 90,
                  "Připomínka musí být dříve než 90 dní před revizí.")
            .withConverter(Double::intValue, Integer::doubleValue)
            .bind(User::getReminderInterval, User::setReminderInterval);

    binder.forField(profileView.getReminderTextEditor().asHtml())
            .bind(User::getReminderText, User::setReminderText);

    binder.readBean(user);
    profileView.getSaveButton().addClickListener(event -> {
        try {
            binder.writeBean(user);
            userService.update(user);
            showSuccess();
        } catch (ValidationException exception) {
            showError();
        }
    });
}
```

Zdrojový kód 7: Ukázka data bindingu

5.6 Odesílání e-mailů

Další funkcionalitou, kterou aplikace disponuje automatickým odesíláním e-mailů při různých příležitostech. Mezi tyto události patří registrace uživatele, obnovení hesla, vytvoření nové revize a připomenutí blížícího se termínu platnosti revize.

Odesílání zpráv je zajišťováno komponenty *MailService* a *MailConfig*, která je vidět v příkladu Zdrojový kód 8. V konfiguračním souboru se vytváří připojení k Mailgun API, pomocí soukromého klíče a URL adresy, které jsou předány pomocí proměnných prostředí. Po vytvoření této beany lze volat její metody pro odesílání zpráv atp.

```
@Configuration
public class MailConfig {
    @Value("${mailgun.apiUrlBase}")
    private String API_URL_BASE;

    @Value("${mailgun.apiKey}")
    private String API_KEY;

    @Bean
    public MailgunMessagesApi mailgunMessagesApi() {
        return MailgunClient.config(API_URL_BASE, API_KEY)
                .createApi(MailgunMessagesApi.class);
    }
}
```

Zdrojový kód 8: Třída MailConfig

Následně lze tuto *MailgunMessagesApi* využít v *MailService*. Nejprve je ale potřeba vytvořit samotnou zprávu skrze třídu *Message* a statickou metodu *builder()*. V příkladu Zdrojový kód 9 odesílání registračního e-mailu je vidět, že se nejprve zadává odesílatel, dále příjemce, předmět a samotná zpráva ve formátu HTML. Nastavit lze i další možnosti jako kopie, skrytá kopie, odpověď, přílohy, možnost sledování interakce s e-mailem a další. Po vytvoření zprávy je e-mail odeslán prostřednictvím již zmiňovaného objektu, představující připojení na Mailgun API.

```
public void sendRegistrationEmail(User user) {
    Message message = Message.builder()
            .from(mailSender)
            .to(user.getEmail())
            .subject("Registarce")
            .html(getRegistrationEmail(user))
            .build();
    mailgunMessagesApi.sendMessage(domain, message);
}
```

Zdrojový kód 9: Ukázka odeslání e-mailu

5.7 Ukládání souborů v Amazon S3

Pro ukládání souborů se v aplikaci využívá již dříve zmíněná služba Amazon S3. Pro práci s ní je využita AWS SDK, která dokáže usnadnit práci se soubory a buckety. V první ukázce je zobrazena část *FileService*, která se stará nejen o práci s Amazon S3, ale také o ukládání metadat souborů v relační databázi. Konstruktor v ukázce Zdrojový kód 10: Ukázka vytvoření připojení k Amazon S3 je označen anotaci *@Autowired* z důvodu vložení environmentálních proměnných obsahujících přístupových klíčů. V těle konstruktoru je následně z těchto proměnných vytvořen objekt typu *AmazonS3*, pomocí něhož lze vytvářet, upravovat či mazat soubory i buckety.

```
private final AmazonS3 s3client;

@Autowired
public FileService(@Value("${aws.s3.access-key}") String aws_access_key,
                   @Value("${aws.s3.secret-key}") String aws_secret_key
) {
 BasicAWSCredentials credentials =
                new BasicAWSCredentials(aws_access_key, aws_secret_key);
 s3client = AmazonS3Client.builder()
        .withRegion(Regions.EU_CENTRAL_1)
        .withCredentials(new AWSStaticCredentialsProvider(credentials))
        .build();
}
```

Zdrojový kód 10: Ukázka vytvoření připojení k Amazon S3

V další ukázce Zdrojový kód 11 je zobrazena metoda, která má na starost ukládání jednotlivých souborů. Soubory jsou vždy ukládané do jednoho bucketu, kde se následně dělí nejprve dle uživatelského ID a dále podle ID revize. Po uložení souboru pomocí metody *putObject()*, je následně vytvořena URL cesta k objektu, kterou tuto metoda vrací a ta je následně uložena do relační databáze spolu s dalšími metadaty.

```
public String saveFileToS3StorageAndGetPath(File file, String userId,
 String revisionId) {
    String fileName = file.getName();
    String path = getFilePath(userId, revisionId, fileName);
    String bucketPath = bucketName + "/" + userId + "/" + revisionId;
    s3client.putObject(bucketName, path, file);
    URL url = s3client.getUrl(bucketPath, fileName);
    return url.toString();
}
```

Zdrojový kód 11: Ukázka uložení souboru do Amazon S3

Poslední ukázka Zdrojový kód 12 je věnována stahování souborů z Amazon S3. Pro stažení souboru je opět třeba znát ID uživatele, ID revize a název souboru. Pro soubor je nejprve vytvořen dočasný soubor se stejnou příponou jakou má stahovaný soubor, kterému je dále nastaveno automatické smazání pomocí metody *deleteOnExit()*. Poté už je stažen soubor pomocí metody *getObject()* a jeho stažený obsah je poté načten do dočasného souboru.

```
public File getFileFromS3Cloud(String file, String revisionId) {
    User user = VaadinSession.getCurrent().getAttribute(User.class);
    File tempFile = null;
    String[] splitStr = file.split("\\.");
    tempFile = File.createTempFile("tmp", "." + splitStr[1]);
    tempFile.deleteOnExit();
    String key =
      getFilePath(user.getId().toString(), revisionId, file);

    GetObjectRequest request = new GetObjectRequest(bucketName, key);
    S3Object object = s3client.getObject(request);
    InputStream in = object.getObjectContent();
    Files.copy(in, tempFile.toPath(),
             StandardCopyOption.REPLACE_EXISTING );

    return tempFile;
}
```

Zdrojový kód 12: Ukázka stažení souboru z Amazon S3

5.8 Načítání dat klientů

V zadání jednoho z požadavků bylo automatické načítání dat klientů na základě zadaného IČO. Pro tuto funkci byla využit administrativní registr ekonomických subjektů neboli zkráceně ARES. Jedná se o webovou aplikaci Ministerstva financí, která zdarma zpřístupňuje údaje o ekonomických subjektech z veřejných informačních systémů jako jsou např. veřejný rejstřík, registr živnostenského podnikání nebo registr ekonomických subjektů.

Pro účely aplikace je využit základní dotaz na API pro výpis identifikačních údajů. Ten je dostupný pomocí volání metody GET s query parametrem udávajícím hledané IČO. Odpověď na tento dotaz je následně ve formě XML.

Pro účely volání API je využita třída *RestTemplate* pomocí níž je sestaven dotaz a následně odeslán pomocí metody *exchange()* jak lze vidět v Zdrojový kód 13.

```
RestTemplate restTemplate = new RestTemplate();
HttpHeaders headers = new HttpHeaders();
headers.set("Content-Type", "text/xml");
headers.set("Accept-Charset", "utf-8");
HttpEntity<String> entity = new HttpEntity<>("parameters", headers);
String aresApiUrl =
            "https://wwwinfo.mfcr.cz/cgi-bin/ares/darv_bas.cgi?ico="+ico;
ResponseEntity<String> response =
 restTemplate.exchange(aresApiUrl, HttpMethod.GET, entity, String.class);
```

Zdrojový kód 13: Ukázka volání ARES

Pro následnou práci s odpovědí ve formátu XML, je využita třída *JAXBContext*, která poskytuje abstrakci pro správu XML/Java bindingu. Dále byly také vytvořena třída *AresResponse*, která představuje strukturu, do které se mapují výsledky získané z API.

V další ukázce Zdrojový kód 14 je předvedena část této třídy, kde je vidět využití anotací pro označení kořenového elementu *@XmlRootElement* nebo běžného elementu *@XmlElement* ze kterého se již načítají data do atributů.

```
@XmlRootElement(name = "VBAS", namespace = "http://wwwinfo.mfcr.cz/...")
@XmlAccessorType(XmlAccessType.FIELD)
class AresDetail {

    @XmlElement(name = "ICO", namespace = "http://wwwinfo.mfcr.cz/...")
    private String ico;

    @XmlElement(name = "OF", namespace = "http://wwwinfo.mfcr.cz/...")
    private String name;

    @XmlElement(name = "DIC", namespace = "http://wwwinfo.mfcr.cz/...")
    private String dic;

    @XmlElement(name = "AA", namespace = "http://wwwinfo.mfcr.cz/...")
    private AresAddress address;

}
```

Zdrojový kód 14: Ukázka třídy AresDetail

V následujícím zdrojovém kódu Zdrojový kód 15 je pak ukázáno, jak se z načtené XML ve formě stringu převádí do třídy *AresResponse*. Nejprve je vytvořena instance *JAXBContext* podle dané třídy a následně do ní převedeny data pomocí metod *createUnmarshaller()* a *unmarshal()*. Následně již lze pracovat s objektem a z načtených dat vytvořit nového klienta.

```
StringReader reader = new StringReader(convertedXml);
JAXBContext context = JAXBContext.newInstance(AresResponse.class);
AresResponse aresResponse = (AresResponse)
                    context.createUnmarshaller().unmarshal(reader);

Client newClient = new Client();
newClient.setIco(aresResponse.getResult().getDetail().getIco());
newClient.setDic(aresResponse.getResult().getDetail().getDic());
```

Zdrojový kód 15: Ukázka načtení XML pomocí JAXBContext

Jako poslední krok byl následně ve formuláři pro vytváření klientů vytvořeno pole pro zadávání IČO spolu s tlačítkem pro vyhledání klienta. Tomu byl nastaven listener, kdy se zavolá metoda pro načtení klienta dle identifikačního čísla občana. V metodě pro hledání klientů se nejprve kontroluje, zda je zadáno platné číslo a následně je volána *ClientService*, která vyhledá klienta pomocí ARES API a pokud je nalezen jsou jeho údaje automaticky vyplněny do ostatních polí formuláře.

```
private FlexLayout getFindClientByIcoAndGetClientIcoWithSearchButton() {
    FlexLayout clientIcoWithSearchButton = new FlexLayout();
    ico.setTooltipText("Po zadání IČO se vyplní ostatní dostupné údaje");
    ico.setPattern("[0-9]{8}");
    ico.setErrorMessage("IČO musí být 8 číslic");
    clientIcoWithSearchButton.add(ico, findCllientButton);
    clientIcoWithSearchButton.setAlignItems(
                                        FlexComponent.Alignment.BASELINE);
    clientIcoWithSearchButton.setFlexGrow(1, ico);
    findCllientButton.addClickListener(e ->
                                        findClientByIco(ico.getValue()));
    findCllientButton.addClassName(LumoUtility.Margin.Left.SMALL);
    findCllientButton.setTooltipText("Vyhledat klienta podle IČO.");
    return clientIcoWithSearchButton;
}


private void findClientByIco(String ico) {
    if (ico == null || ico.trim().length() != 8)
        return;
    Client client = clientService.getByIco(ico.trim());
    if (client != null) {
        setClient(client);
    }
}
```

Zdrojový kód 16: Ukázka formulářového prvku s automatickým načítáním klienta

5.9 Kalendář

Aplikace nabízí také přehledné zobrazení datumů revizí a konec jejich platností v kalendáři. Jelikož framework Vaadin nenabízí komponentu pro zobrazení kalendáře je využita možnost přidat si aplikace komponenty třetích stran. Pro tento účel byla vybrána komponenta Year Month Calendar [43], která nabízí zobrazení kalendáře v různých formách.

Pro použití tohoto je nejprve nutné zahrnou závislosti. V projektu je využit Maven, proto je v příkladu Zdrojový kód 17: Přídání komponenty třetích stran do pom.xmlZdrojový kód 17 zobrazena část souboru *pom.xml*, kde je nutné přidat nejen závislost komponenty, ale také přidat repozitář, obsahující Vaadin komponenty.

```
<repository>
   <id>Vaadin Directory</id>
   <url>https://maven.vaadin.com/vaadin-addons</url>
</repository>

<dependency>
   <groupId>org.vaadin.addons.flowingcode</groupId>
   <artifactId>year-month-calendar</artifactId>
   <version>3.0.1</version>
</dependency>
```

Zdrojový kód 17: Přídání komponenty třetích stran do pom.xml

Dále je také nutné vložit balíček ve kterém se nachází tato komponenta, na tzv. whitelist, povolených zdrojů balíčků, ve kterých se vyhledávají komponenty pro použití v uživatelském rozhraní. Toto nastavení se provádí v souboru *application.properties*

```
vaadin.whitelisted-packages=com.vaadin,org.vaadin,dev.hilla,
                                    cz.upce.fei.dp,com.flowingcode
```

Zdrojový kód 18: Přidání do balíčku do seznamu určených pro UI komponenty

V aplikaci je pak využita komponenta *MonthCalendar*, která slouží k zobrazování vždy jednoho měsíce v roce. Z této komponenty je následně vytvořena vlastní komponenta s dodatečným ovládáním a přizpůsobenými styly, která je použita na více místech v aplikaci. Komponenta pro svůj běh využívá třídu *YearMonth* pro zobrazování různých měsíců a let.

```
private YearMonth date = YearMonth.now();
private final MonthCalendar calendar = new MonthCalendar(date);
```

Zdrojový kód 19: Inicializace komponenty MonthCalendar

V dalším příkladu Zdrojový kód 20 je ukázáno nastavení této komponenty. Kalendáři je nejprve nastaveno vlastní téma a česká lokalizace. Následně je pomocí funkce *setClassNameGenerator()* nastaven styl jednotlivých polí v kalendáři na základě toho, zda je v daný naplánována či má termín nějaká revize. Přidán je také listener na vybrání některého datumu, po kterém se zobrazí dialogové okno s revizemi souvisejících s tímto dnem.

```
calendar.getElement().getThemeList().add("custom-calendar");
calendar.setI18n(new CalendarCzechI18N());
calendar.setClassNameGenerator(date -> {
    boolean hasDateRevisionDate = false;
    boolean hasDateRevisionExpired = false;
    for (Revision revision : revisions) {
        if (revision.getRevisionDate().equals(date)) {
            hasDateRevisionDate = true;
        } else if (revision.getValidityDate() != null &&
                        revision.getValidityDate().equals(date)) {
            hasDateRevisionExpired = true;
        }
    }
    if (hasDateRevisionDate && hasDateRevisionExpired) {
        return "has-revision-date-and-expired";
    } else if (hasDateRevisionDate) {
        return "has-revision-date";
    } else if (hasDateRevisionExpired) {
        return "has-revision-expired";
    }
    return null;
});
calendar.addDateSelectedListener(date -> {
    List<Revision> revisionInDate = getRevisionInDate(date);
    Dialog revisionsDialog = new RevisionGridDialogView(
                                    revisionInDate, date.getDate());
    revisionsDialog.open();
});
```

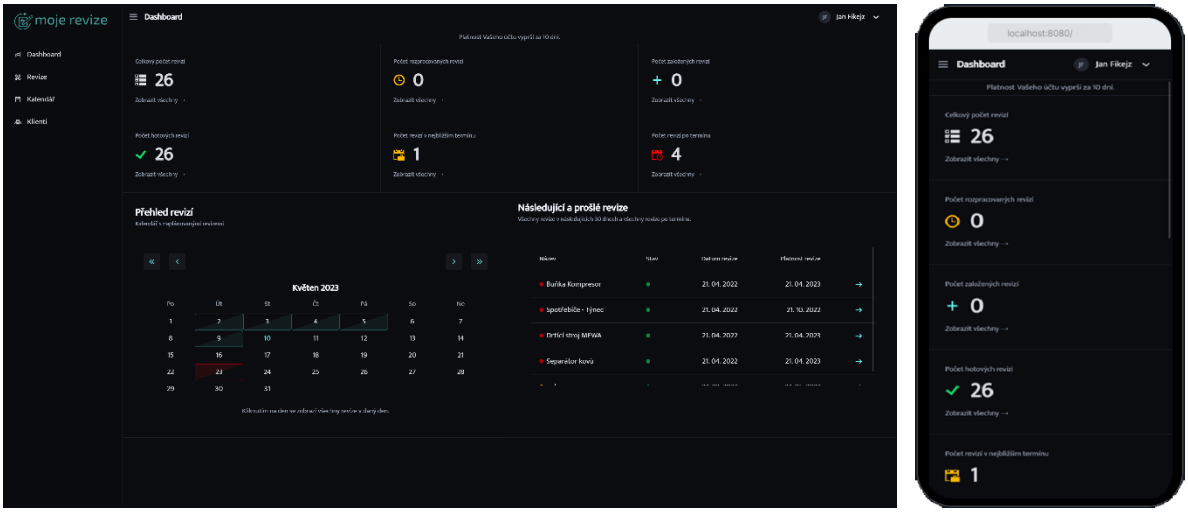
Zdrojový kód 20: Ukázka použití komponenty MonthCalendar

5.10 Responzivní design

V dnešní době již většina lidí vlastní mobilní telefon s přístupem k internetu, a ten se tak stal pro většinu populace primárním zařízením, pomocí něhož přistupují na internet. Na základě výzkumu o využívání informačních a komunikačních technologií z roku 2022 [65], je známo, že z údajů z roku 2021 v České republice ve věkové kategorii 16-74 let využívá mobilní telefon k přístupu na internet 79% obyvatel. Průměr v EU je pak ještě o 81% vyšší a tento trend stále stoupá.

Proto je nutné na toto myslet při vývoji, tak aby výsledná aplikace se dala pohodlně používat i v mobilním telefonu či tabletu. Navíc je velmi pravděpodobné, že uživatelé budou tuto aplikace používat také v terénu přímo na místě konání revize, a tak bude daleko víc pohodlné využít nějaké mobilní zařízení, než sebou nosit notebook či PC. Tato funkce by se tedy mohla stát rozdílovou výhodou při výběru aplikace, oproti svým konkurentům.

Framework Vaadin v základu již nabízí komponenty, které v sobě mají zakomponovaný responzivní design. Jedná se především o třídu *AppLayout*, která představuje základní rozdělení celé stránky na tři hlavní části – horní panel, boční menu (tzv. drawer), které je možno skrýt pomocí tlačítka a centrální část ve které se nachází vždy samotný obsah. Tato komponenta se dokáže automaticky přizpůsobit velikosti displeje používaného zařízení ať už jde o desktop, tablet či mobilního telefonu. Příklad takovéhoto rozložení lze vidět na následujícím Obrázek 15, kde jsou porovnáno zobrazení aplikace na desktopu a na mobilním zařízení.



Obrázek 15: Porovnání zobrazení na desktopu a mobilním zařízení

Dalším komponentou, která nabízí responzivní design je *FormLayout*, který slouží k snadnému vytváření responzivních formulářů. Pro vytváření jsou k dispozici různé rozvržení pomocí sloupců či úprava umisťování popisků vedle či nad jednotlivými inputy. V následující ukázce Zdrojový kód 21: Ukázka využití responzivního chování FormLayout lze vidět nastavení responzivního chování pomocí metody *setResponsiveSteps()*. Formulář reaguje na šířku okna, na jejímž základě následně upravuje počet sloupců, do kterých se inputy řadí.

```
FormLayout formLayout = new FormLayout();
formLayout.add(firstName, lastName, email);
formLayout.setResponsiveSteps(
     // Use one column by default
     new ResponsiveStep("0", 1),
     // Use two columns, if the layout's width exceeds 320px
     new ResponsiveStep("320px", 2),
     // Use three columns, if the layout's width exceeds 500px
     new ResponsiveStep("500px", 3)
);
```

Zdrojový kód 21: Ukázka využití responzivního chování FormLayout

Další na řadě je komponenta *Board*, která na slouží pro tvorbu responzivního stránek. Tato komponenta sama automaticky uspořádává svůj vnitřní obsah podle různých velikostí obrazovek a zároveň maximalizuje využití dostupného prostoru bez nutnosti dalších zásahů programátora. *Board* se vždy skládá z řádků a každý řádek může být dále rozdělen až na čtyři sloupce. Tyto sloupce se dokážou sami zalamovat při změně velikosti zobrazení. Komponenta má tři předdefinované breakpointy, na kterých se vždy upravuje vzhled celé komponenty. Na tyto breakpointy lze také navázat různé styly např. změnu velikosti písma či mezery mezi vnitřními komponenty. V aplikaci je tato komponenta využita např. při tvorbě dashboardu, jak lze vidět zpětně na Obrázek 15: Porovnání zobrazení na desktopu a mobilním zařízení.

Problémem při tvorbě responzivního designu nastal při implementací seznamu. Komponenta *Grid* se automaticky nepřizpůsobuje velikosti displeje. Pro seznamy, které obsahují více sloupců s informacemi o jednotlivých záznamech poté nastává problém, kde všechny sloupce nejsou na menších displejích zobrazeny a uživatel musí scrollovat, aby se k informacím dostal. Toto řešení není příliš uživatelsky pohodlné, proto je nutné jej vyřešit programově. V view ve kterém se tyto seznamy nachází, jsou vytvořeny listenery na změnu šířky zobrazení a na základě těchto změn se následně upravuje zobrazení jednotlivých sloupců. V ukázce Zdrojový kód 22: Ukázka změny zobrazovaných sloupců na základě šířky zobrazení lze vidět použití v seznamu revizí. Jelikož je u každé revize zobrazováno běžně velké množství údajů, tak se pro menší displeje zobrazuje pouze speciálně vytvořený sloupec a ostatní sloupce jsou skryty.

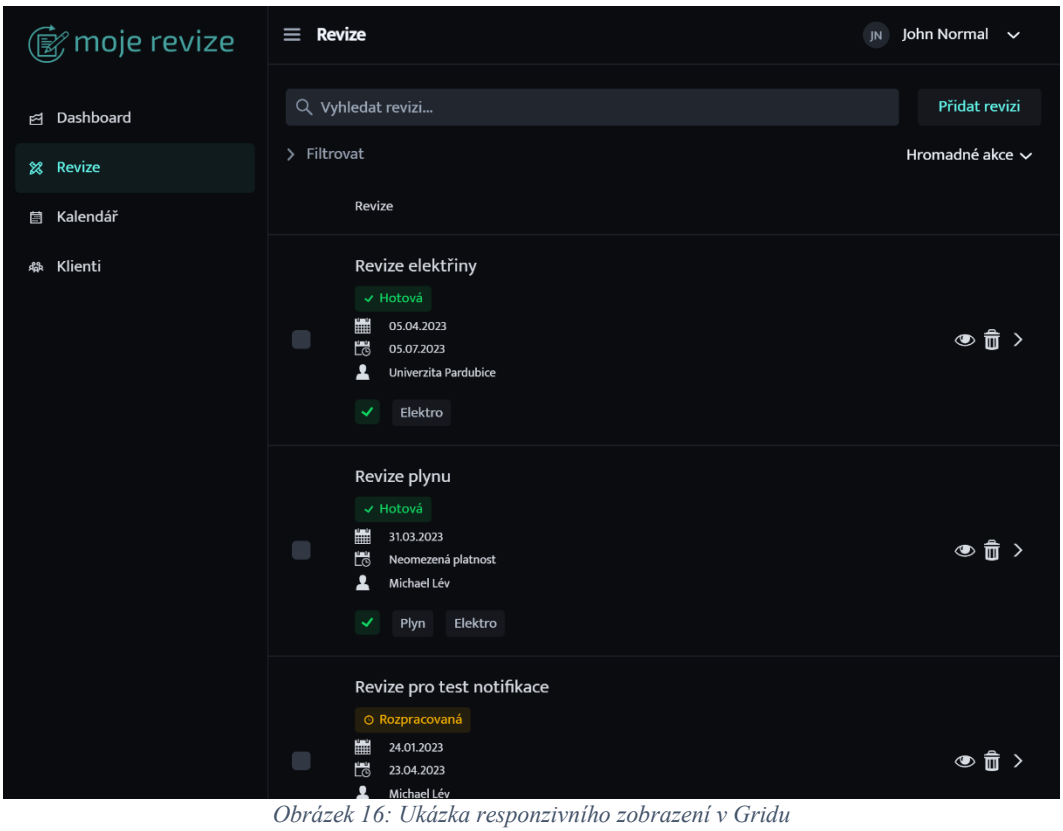
```
@Override
protected void onAttach(AttachEvent attachEvent) {
    super.onAttach(attachEvent);
    getUI().ifPresent(ui -> widthListener =
      ui.getPage().addBrowserWindowResizeListener(event ->
         adjustVisibleGridColumns(event.getWidth())));
    getUI().ifPresent(ui ->
      ui.getPage().retrieveExtendedClientDetails(receiver -> {
        int browserWidth = receiver.getBodyClientWidth();
        adjustVisibleGridColumns(browserWidth);
    }));
}

@Override
protected void onDetach(DetachEvent detachEvent) {
    widthListener.remove();
    super.onDetach(detachEvent);
}
```

```
private void adjustVisibleGridColumns(int width) {
 grid.getColumnByKey("revision").setVisible(width <= WIDTH_BREAKPOIN);
 grid.getColumnByKey("name").setVisible(width > WIDTH_BREAKPOIN);
 grid.getColumnByKey("status").setVisible(width > WIDTH_BREAKPOIN);
 grid.getColumnByKey("revisionDate").setVisible(width > WIDTH_BREAKPOIN);
 grid.getColumnByKey("validityDate").setVisible(width > WIDTH_BREAKPOIN);
 grid.getColumnByKey("actions").setVisible(width > WIDTH_BREAKPOIN);
 grid.getColumnByKey("invoiced").setVisible(width > WIDTH_BREAKPOIN);
 grid.getColumnByKey("tags").setVisible(width > WIDTH_BREAKPOIN);
 grid.getColumnByKey("client").setVisible(width > WIDTH_BREAKPOIN);
}
```

Zdrojový kód 22: Ukázka změny zobrazovaných sloupců na základě šířky zobrazení

Ve speciálním sloupci jsou pak zobrazeny všechny potřebné informace, tak aby je bylo možno zobrazit na menších obrazovkách. Ukázka tohoto zobrazení je pak na následujícím Obrázek 16: Ukázka responzivního zobrazení v Gridu.



Obrázek 16: Ukázka responzivního zobrazení v Gridu

6 TESTOVÁNÍ A NASAZENÍ APLIKACE

Poslední fázi vývoje softwaru je testování aplikace, následné nasazení hotové aplikace na produkční prostředí a její údržba. Tato kapitola tak bude věnována především testování uživatelského rozhraní a následného nastavení a nasazení celé aplikace.

6.1 Testování aplikace

Cílem testování by mělo být ověření funkčnosti aplikace a nalezení chyb či problémů. Testování aplikace by mělo probíhat během celého vývoje, kdy pro každou fázi vývoje je vhodný jiný druh testů. Jelikož daný projekt není příliš komplexní v backendové části, bylo přistoupeno až k testování uživatelského rozhraní hotové aplikace za pomocí end-to-end testování.

Pro testování aplikace se využívá nástroj TestBench, který je určen pro vytváření UI unit testů a end-to-end testů pro Vaadin aplikace. Tento nástroj poskytuje speciální podporu pro Vaadin aplikace, nabízí tak snadnější a robustnější testování ve srovnání s obecnými řešeními, jako je například nástroj Selenium.

Pomocí jednotkového UI testování lze psát jednoduché testy, které ke svému běhu nepotřebují prohlížeč ani servlet kontejner. Tyto testy běží pouze na straně serveru a díky tomu jsou tak rychlejší a stabilnější. Hodí se tak spíše na testování během vývoje aplikace, kdy je třeba psát více testů a ty pak často opakovaně spouštět pro ověření správné funkčnosti.

Oproti tomu end-to-end testy jsou blíže reálnému uživatelskému prostředí, mohou simulovat různé uživatelské interakce s aplikací a ověřit že je aplikace vizuálně v pořádku ve všech testovaných prohlížečích. End-to-end testy jsou tak vhodnější pro testování kritických částí aplikace, jako je např. přihlašování, protože simulují skutečné využití aplikace uživatelem. Pomocí tohoto testování lze navíc také otestovat vlastní knihovny napsané pomocí knihovny Lit.

Pro použití tohoto nástroje je nejprve nutné jej zahrnout do seznamu závislostí dle ukázky Zdrojový kód 23, s přidáním pouze do testovacího scopu. Dále je také nutné stáhnout si webové ovladače všech prohlížečů, na kterých se aplikace bude testovat.

```
<dependency>
    <groupId>com.vaadin</groupId>
    <artifactId>vaadin-testbench</artifactId>
    <scope>test</scope>
</dependency>
```

Zdrojový kód 23: Zahrnutí Vaadin TestBench do projektu

V následujícím příkladu Zdrojový kód 24 lze vidět inicializace třídy pro testování přihlašovací obrazovky. Tato třída je označena anotací *@SpringBootTest* a dědí z třídy *TestBenchTestCase* a jako první jsou uvedeny metody s anotacemi *@Before* a *@After*, které se provedou vždy před, respektive po každém provedeném testu. V metodě *setup()* se nejprve nastavuje cesta k ovladači a nastavení pro prohlížeč Google Chrome a následně se tento ovladač nastaví a otevře se úvodní stránka aplikace. V metodě *tearDown()* se následně tento prohlížeč zavírá.

```
@SpringBootTest
public class LoginViewTest extends TestBenchTestCase {

    @Before
    public void setup() {
        System.setProperty("webdriver.chrome.driver",
                    "src/test/resources/chromedriver.exe");
        ChromeOptions options = new ChromeOptions();
        options.addArguments("--remote-allow-origins=*");
        setDriver(new ChromeDriver(options));
        getDriver().get("http://localhost:8080/");
    }

    @After
    public void tearDown() {
        getDriver().quit();
    }
```

Zdrojový kód 24: Ukázka třídy LoginViewTest

Jako příklad testu v Zdrojový kód 25, byl vybrány jednoduché testy, které testují správné zobrazení přihlašovací obrazovky a následně pokus o neúspěšné přihlášení. Pro nalezení potřebných komponent se využívá *ComponentQuery* za pomocí volání *$()* se zadáním specifické komponenty, kterou je potřeba najít. Následně je nad tímto objektem volat metody jako např. *first()*, *last()*, *all()*, *atIndex()* či *id()* pro nalezení hledané komponenty. Následně pak lze s těmito komponentami pracovat, získat jejich data či zadávat vstupy. Na konci jsou ověřeny výstupy, zda se shoduje hodnota v uživatelském rozhraní s a předem danými popisy.

```
@Test
public void isLoginViewShownTest() {
   LoginFormElement loginFormElement = $(LoginFormElement.class).first();
   TextFieldElement usernameFieldElement =
                                    loginFormElement.getUsernameField();
   PasswordFieldElement passwordFieldElement =
                                    loginFormElement.getPasswordField();
   ButtonElement submitButton = loginFormElement.getSubmitButton();

   Assert.assertTrue(loginFormElement.isDisplayed());
   Assert.assertEquals("Uživatelské jméno",
                                    usernameFieldElement.getLabel());
   Assert.assertEquals("Heslo", passwordFieldElement.getLabel());
   Assert.assertEquals("Přihlásit se", submitButton.getText());
}
```

Zdrojový kód 25: Ukázka testu pro ověření zobrazení přihlašovací obrazovky

V následující ukázce Zdrojový kód 26 se testuje přihlášení s neplatnými údaji. Při provádění jsou do příslušných inputů zadány neplatné přihlašovací údaje a poté pomocí metody *click()* na přihlašovací tlačítko odeslán pokus o přihlášení. Následně je ověřeno zobrazení správné chybové hlášky.

```
@Test
public void isLoginViewShownTest() {
    LoginFormElement loginFormElement = $(LoginFormElement.class).first();
    TextFieldElement usernameFieldElement =
                                        loginFormElement.getUsernameField();
    PasswordFieldElement passwordFieldElement =
                                        loginFormElement.getPasswordField();
    ButtonElement submitButton = loginFormElement.getSubmitButton();

    usernameFieldElement.setValue("user");
    passwordFieldElement.setValue("wrongPassword");
    button.click();

    LoginCzechI18n expected = new LoginCzechI18n();
    LoginFormElement result = $(LoginFormElement.class).first();
    Assert.assertTrue(result.isDisplayed());
    Assert.assertEquals(expected.getErrorMessage().getTitle(),
                                            result.getErrorMessageTitle());
}
```

Zdrojový kód 26: Ukázka testu pro neúspěšné přihlašování

6.2 Nasazení aplikace

Následné nasazení aplikace na produkční server v sobě zahrnuje kompilaci a zabalení aplikace do balíčku tak, aby byla spustitelná a optimalizovaná pro daný server. Pro její nasazení musí být vytvořen jediný soubor, do kterého se zahrnou všechny soubory projektu zkomprimují se a zabalí. Vaadin nabízí dva typy souborů, do kterých lze aplikaci vygenerovat v závislosti na tom kde webovou aplikaci nasazujeme.

Pro nasazení na běžně používané webové servery jako jsou Jetty, Tomcat, WildFly nebo GlassFish je nutné vygenerovat aplikaci ve formátu WAR. Do tohoto formátu jsou standartně zabalovány webové aplikace napsané v Javě. Druhou možností je zvolit typ souboru JAR, který již zahrnuje aplikační server.

Při spuštění aplikace lokálně se Vaadin projekt standartně spouští v tzv. development módu, který je upraven pro pohodlný vývoj. Díky němu získáme snadné debuggování za cenu vyššího vytížení CPU a paměti. Navíc je v tomto módu automatická detekce při změně CSS a JavaScript souborů, díky níž není třeba celou aplikaci restartovat. V produkčním prostředí tyto vlastnosti již nejsou potřeba proto je vhodné při sestavování aplikace pro produkci zvolit produkční mód. Ten zajistí lepší optimalizaci a výkon celé aplikace v produkčním prostředí. Pro vytvoření produkčního buildu pomocí nástroje Maven se používá následující příkaz.

```
$ mvn clean package -Pproduction
```

Zdrojový kód 27: Vytvoření produkčního buildu

6.3 Nasazení na Heroku

Heroku je cloudová platforma založená na kontejnerech, která slouží k nasazování a provozu moderních aplikací. Jedná se o tzv. PaaS službu neboli platforma jako služba. Uživatelé zde nemusí řešit nastavení ani jakoukoliv správu serveru. Jediné, co je potřeba řešit je nahrání

samotné aplikace a o zbytek se již postará platforma sama. Heroku nabízí široké množství podporovaných technologií a doplňkových služeb jako jsou třeba databáze, monitoring nebo automatické testování.

Tato platforma byla vybrána především díky jednoduchosti a rychlosti s jakou lze aplikaci nasadit a začít používat. Navíc je pro studenty k dispozici studentský program skrze GitHub Student Developer PackHeroku lze následně ovládat buďto skrze příkazový řádek Heroku CLI případně pomocí webové aplikace. Pro nasazení pomocí webu stačí propojit účet na Heroku s účtem na platformě GitHub a následně zvolit repozitář a vývojovou větev ve kterém se nachází aktuální verze aplikace. O zbytek nahrání a nasazení aplikace se již postará Heroku samo.

Nasazení aplikace pomocí CLI je velmi snadné, jak lze vidět na příkladu Zdrojový kód 28. Před prvním použitím je nejprve nutné se přihlásit. To lze jednoduše pomocí příkazu z následující ukázky. Po jeho zadání je uživatel přesměrován na stránky Heroku, kde si přihlásí pomocí svého jména a hesla a může dále pokračovat.

Následně je nutné nainstalovat si plugin pro běh Java aplikaci. Po jeho instalaci je možné nahrávat na platformu jak sobory typu WAR, tak i soubory ve formátu JAR. Dalším krokem je vytvoření samotné aplikace v Heroku. To je nutné pouze při prvním nahrávání aplikace, případně lze aplikaci vytvořit pomocí webové aplikace. Dalším krokem je zbuildění samotné aplikace, jak bylo představeno již na začátku této kapitoly. Po úspěšném sestavení aplikace do souboru JAR, lze pak jednoduše tento soubor nahrát na server a samotná aplikace je během chvíle nasazena.

```
$ heroku login
$ heroku plugins:install java
$ heroku create moje-revize
$ heroku deploy:jar target/mojerevize-1.0-SNAPSHOT.jar -a moje-revize
```

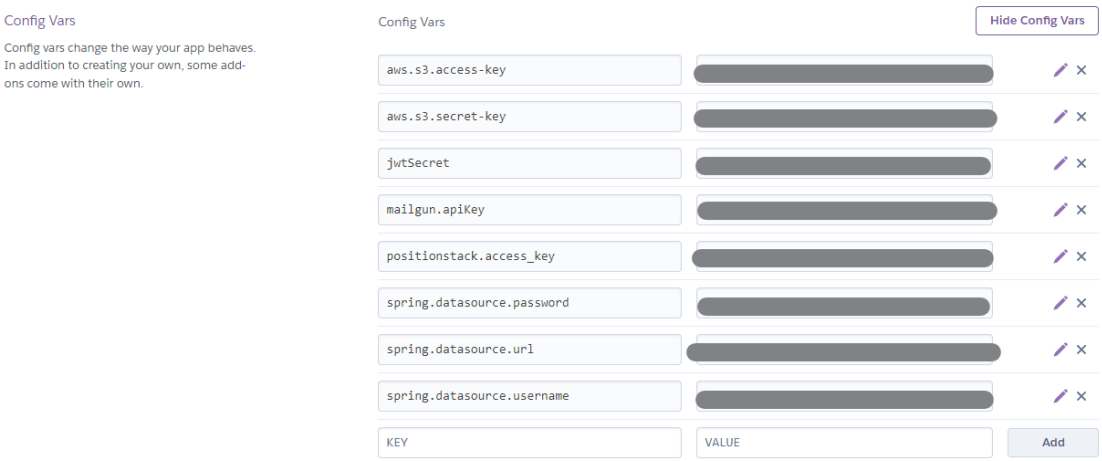
Zdrojový kód 28: Postup nasazení aplikace na Heroku pomocí Heroku CLI

Pro správný běh aplikace je ještě nutné nastavit správné proměnné prostředí. Tyto proměnné se využívají zejména pro data, které se mohou měnit nebo pro citlivé údaje jako jsou přihlašovací údaje či API klíče. Tyto citlivá data by neměla být uložena v zdrojovém kódu z důvodu bezpečnosti. V příkladě aplikace pro správu revizí se může jednat například použití různých Amazon S3 účtů či různých účtů pro odesílání e-mailů. Tyto proměnné lze opět jednoduše nastavit pomocí příkazové řádky a příkazu ze Zdrojový kód 29.

```
$ heroku config:set mailgun.apiKey=apiKey123456
```

Zdrojový kód 29: Nastavení proměných prostředí pomocí Heroku CLI

Případně lze tyto proměnné nastavit opět ve webové aplikaci Heroku v nastavení dané aplikaci v sekci Config Vars jak lze vidět na Obrázek 17. Jelikož se jedná o screenshot z nasazené aplikace jsou jednotlivé hodnoty z důvodu bezpečnosti skryty.



Obrázek 17: Nastavení Config Vars v Heroku

6.4 Nasazení pomocí Dockeru

Jedním z požadavků při zadání bylo také možnost nasazení aplikace ve formě kontejneru na platformu Docker. Ta umožňuje vytvořit kontejner, který obsahuje vše potřebné pro běh samotné aplikace, její kód, běhové prostředí, systémové knihovny atp. Výsledný kontejner pak není závislý na systému ani platformě, na kterém je spouštěn, lze jej spustit kdekoliv.

Pro vytvoření takovéhoto kontejneru z aplikace je třeba mít připravený *Dockerfile*. Jedná se o textový soubor, který obsahuje všechny příkazy pro sestavení a následný jeho běh. Tento *Dockerfile* si je možno nechat vygenerovat při vytváření projektu pomocí webové aplikace Vaadin Start nebo si jej lze vytvořit samostatně.

V ukázce Zdrojový kód 30, lze vidět, že se výsledný *Dockerfile* skládá pouze z pěti příkazů. První příkaz určuje základní obraz, na kterém bude kontejner postaven. Jedná se o obraz s jádrem OpenJDK 20 a s minimálními balíčky potřebnými pro běh Java aplikací. Druhý příkaz následně kopíruje do kontejneru a přejmenovává všechny soubory s příponou *.jar* z adresáře *target* pro nakopírování aplikace a dále se také kopíruje soubor *env.conf* s enviromentálními proměnnými. V dalším kroku je definován port 8080, na kterém bude kontejner naslouchat. Posledním příkaz pak určuje, jaký příkaz a s jakými parametry se bude spouštět po spuštění kontejneru.

```
FROM openjdk:20-jdk-slim
COPY target/*.jar app.jar
COPY env.conf /env.conf
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Zdrojový kód 30: Dockerfile

Pro následné vytvoření a spuštění pak lze použít běžně používané příkazy z Zdrojový kód 31. Nejprve je nutné mít vytvořený produkční build. Následně lze vytvořit kontejner pomocí příkazu *docker build*. Po jeho vytvoření se pak spouští příkazem *docker run* s parametrem *-p*, který určuje přesměrování komunikace z portu 8080 na hostitelském systému na port 8080 uvnitř kontejneru a parametrem *--env-file*, pomocí nějž se kontejneru předávají proměnné prostředí.

```
$ mvn clean install -Pproduction
$ docker build . -t mojerevize:latest
$ docker run -p 8080:8080 –env-file env.conf mojerevize:latest
```

Zdrojový kód 31: Vytvoření kontejneru a jeho spuštění v Dockeru

ZÁVĚR

Cílem této diplomové práce bylo navrhnout a vyvinout webovou aplikaci pro správu revizí s využitím technologií Spring a Vaadin, která umožní efektivní a spolehlivé řízení revizních procesů.

Revize představují důležitou kontrolu funkčnosti a bezpečnosti různých zařízení a systému, které by při poruše či nesprávném fungování mohli způsobit škody na majetku či zdraví zvířat a osob. Proto je nutné zákona tyto revize provádět v předem daných intervalech podle platných zákonů a norem. Aplikace pro správu revizí se tak zaměřuje především na usnadnění a automatizaci tohoto procesu, včetně plánování, sledování, připomínání a vyhodnocování výsledků revizí.

V rámci práce bylo provedeno srovnání existujících aplikací pro správu revizí, přičemž byly analyzovány jejich klíčové funkce, vlastnosti, komplexnost, cena či uživatelská přívětivost. Tato analýza poskytla důležitý základ pro návrh a implementaci vlastní aplikace, která byla přizpůsobena konkrétním potřebám a požadavkům ze zadání. Dále byly také představeny klíčové vlastnosti webových aplikací, jejich hlavní výhody a nevýhody oproti běžným aplikacím a byl krátce nastíněn, jak probíhá jejich vývoj.

Během návrhu vlastní aplikace byly nejprve specifikovány funkční a nefunkční požadavky, na jejichž základě byl vytvořen datový model a návrh uživatelského prostředí. Následně byly proveden výběr vhodných technologií a architektury, následovaný samotným vývojem aplikace. Pro implementaci byl zvolen framework Vaadin, který je určen pro jednoduchý vývoj webových aplikacích v jazyce Java ve spojení s frameworkem Spring. Mezi dalšími popsanými a využitými technologiemi jsou MySQL, Amazon S3 či Mailgun.

Po dokončení fáze návrhu následovala již samotná implementace aplikace. Během této fáze byla vyvinuta celá aplikace na základě zadaných požadavků. Během vývoje nastalo několik komplikací, především při návrhu intuitivního uživatelského rozhraní, u responzivního zobrazení na různě velkých zařízení či doplnění a upřesnění některých požadavků během vývoje.

Po dokončení implementace následovalo otestování aplikace a její nasazení na produkční prostředí. Pro testování byl využit nástroj TestBench, pomocí něhož byly vytvořeny end-to-end testy pro ověření správné funkčnosti uživatelského rozhraní. Po ukončení testování a opravení nalezených chyb byla následně aplikace nasazena na cloudovou platformu Heroku. Zároveň byla také projekt připraven pro případnou kontejnerizaci na platformě Docker.

Finálním výsledkem této diplomové práce je tak webová aplikace pro správu revizí s názvem Moje revize. Tato aplikace splňuje všechny zadané požadavky a oproti svým stávajícím konkurentům nabízí spojení všech nabízených funkcionalit doplněné o moderní design. Cíle této diplomové práce tak byly splněny v plném rozsahu.

POUŽITÁ LITERATURA

[1] PRO-IDEA / WWW.PRO-IDEA.CZ. Informace | Revizáci.cz: Co je revize [online]. © 2011 [cit. 2023-02-05]. Dostupné z: https://www.revizaci.cz/revize/revize

[2] ČESKO. Zákon č. 250/2021 Sb., o bezpečnosti práce v souvislosti s provozem vyhrazených technických zařízení a o změně souvisejících zákonů. In: *Zákony pro lidi.cz* [online]. © AION CS 2010-2023 [cit. 2023-02-05]. Dostupné z: https://www.zakonyprolidi.cz/cs/2021-250

[3] TECHNICKÁ INSPEKCE ČESKÉ REPUBLIKY. TIČR - Technická inspekce České republiky [online]. 2021 [cit. 2023-02-05]. Dostupné z: https://www.ticr.eu/

[4] AGIONET S.R.O. Správa revizí - FACMAN - online řešení [online]. © 1999–2022 [cit. 2023-02-05]. Dostupné z: https://www.facman.cz/

[5] ACONTE, S.R.O. Online Revize | Systém pro správu revizí - online a jednoduše [online]. © 2017 [cit. 2023-02-05]. Dostupné z: https://online-revize.cz/

[6] APTIEN.COM. Servis a revize strojů a zařízení | Aptien [online]. © 2021 [cit. 2023-02-05]. Dostupné z: https://aptien.com/cs/servis-a-revize-stroju-a-zarizeni

[7] VOLLE, Adam. Web application | Definition, History, Development, Examples, Uses, & Facts [online]. 6. 10. 2022 [cit. 2023-02-05]. Dostupné z: https://www.britannica.com/topic/Web-application

[8] ČSÚ [Český statistický úřad]. *Využívání informačních a komunikačních technologií v domácnostech a mezi osobami – 2022*: *1. Počítače a internet v domácnostech* [online]. 22. 11. 2022 [cit. 2023-02-05]. Dostupné z: https://www.czso.cz/documents/10180/164606768/0620042201.pdf/5699654d-a722-44c9-a5e8-80443c89be18?version=1.1

[9] SINGH, Siddharth. MVC Framework: A Modern Web Application Development Approach and Working. *International Research Journal of Engineering and Technology* [online]. 2020, **7**(1), 51-55 [cit. 2023-02-11]. ISSN 2395-0056. Dostupné z: https://www.irjet.net/archives/V7/i1/IRJET-V7I111.pdf

[10] What Is Java? [online]. Amazon Web Services, © 2023 [cit. 2023-02-11]. Dostupné z: https://aws.amazon.com/what-is/java/

[11] Introduction to Java. In: W3Schools: Online Web Tutorials [online]. © 1999-2023 [cit. 2023-02-11]. Dostupné z: https://www.w3schools.com/java/java_intro.asp

[12] CARNELL, John. Spring microservices in action: a multiplatform approach to building chatbots. Shelter, Island, NY: Manning Publications Co., 2017. [cit. 2023-02-11]. ISBN 16-172-9398-9.

[13] CARNELL, John. Beginning spring boot 2: applications and microservices with the spring framework. New York, NY: Springer Science Business Media, 2017. [cit. 2023-02-11]. ISBN 978-148-4229-309.

[14] JOHNSON, Rod, aj. *Spring Framework Overview* [online]. 11. 1. 2023 [cit. 2023-02-11]. Dostupné z: https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html#overview

[15] JOHNSON, Rod, aj. *2. Introduction to Spring Framework* [online]. 2014 [cit. 2023-02-11]. Dostupné z: https://docs.spring.io/spring-framework/docs/4.3.x/spring-framework-reference/html/overview.html#overview-modules

[16] Spring.io. In: Projects [online]. © Pivotal Software, Inc. 2023 [cit. 2023-02-14]. Dostupné z: https://spring.io/projects

[17] Spring.io. In: Spring Boot [online]. © Pivotal Software, Inc. 2023 [cit. 2023-02-14]. Dostupné z: https://spring.io/projects/spring-boot

[18] WEBB, Phillip, Dave SYER, Josh LONG aj. *Developing with Spring Boot. Spring Initializr Reference Guide* [online]. 20. 1. 2023 [cit. 2023-02-12]. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/html/using.html#using.build-systems.dependency-management

[19] NICOLL, Stéphane, Dave SYER a Madhura BHAVE. *Spring Initializr Reference Guide* [online]. 21. 10. 2022 [cit. 2023-02-12]. Dostupné z: https://docs.spring.io/initializr/docs/current/reference/html/#introduction

[20] ORABY, Tarek, Jouni KOIVUVIITA, Doug JEWSBURY, aj. *Security | Vaadin Docs: Starting a Project* [online]. 1. 7. 2022 [cit. 2023-02-14]. Dostupné z: https://vaadin.com/docs/v23/guide/start

[21] Vaadin Ltd. *Vaadin Flow | The Full-stack Java Web Framework* [online]. © 2023 [cit. 2023-02-14]. Dostupné z: https://vaadin.com/flow

[22] KOIVUVIITA, Jouni, Marco COLLOVATI, aj. *Starting a Project | Get Started | Vaadin Docs: Securing Spring Boot Applications* [online]. 14. 10. 2022 [cit. 2023-02-14]. Dostupné z: https://vaadin.com/docs/v23/security/enabling-security

[23] KOIVUVIITA, Jouni, Mikhail SHABAROV, aj. *Testing | Vaadin Docs: Testing Vaadin Applications Using TestBench* [online]. 10. 1. 2023 [cit. 2023-02-14]. Dostupné z: https://vaadin.com/docs/v23/testing

[24] Vaadin Ltd. *UI Component Library for Your Java Web Apps | Vaadin* [online]. © 2023 [cit. 2023-02-14]. Dostupné z: https://vaadin.com/components

[25] Vaadin Ltd. *Design system | Vaadin* [online]. © 2023 [cit. 2023-02-14]. Dostupné z: https://vaadin.com/design-system

[26] Vaadin Ltd. *Pricing | Vaadin: Pricing* [online]. © 2023 [cit. 2023-02-14]. Dostupné z: https://vaadin.com/pricing

[27] KOIVUVIITA, Jouni. *Designer | Tools | Vaadin Docs: Designer* [online]. 25. 1. 2023 [cit. 2023-02-14]. Dostupné z: https://vaadin.com/docs/latest/tools/designer

[28] HELLBERG, Marcus. *Spring Boot Web App Tutorial (Java) | Full Course. In: Youtube* [online]. 30. 9. 2021 [cit. 2023-02-15]. Dostupné z: https://www.youtube.com/watch?v=bxy2JgqqKDU

[29] VAN STIJN, Sebastiaan, Allie SADLER, Stephen TURNER a aj. *Docker overview | Docker Documentation: Docker overview* [online]. 2020, 8. 2. 2023 [cit. 2023-02-15]. Dostupné z: https://docs.docker.com/get-started/overview/

[30] DOCKER. *What is a Container? | Docker* [online]. 2021, 9. 12. 2022 [cit. 2023-02-15]. Dostupné z: https://www.docker.com/resources/what-container/

[31] BALA, Priya C. *Docker vs Virtual Machine (VM) – Key Differences You Should Know* [online]. 8. 10. 2022 [cit. 2023-02-15]. Dostupné z: https://www.freecodecamp.org/news/docker-vs-vm-key-differences-you-should-know/

[32] ORACLE. What Is MySQL? | Oracle [online]. 27. 8. 2021 [cit. 2023-04-20]. Dostupné z: https://www.oracle.com/mysql/what-is-mysql/

[33] SHAH, Hardik. The 7 Best Reasons to Choose MySQL for Organizing Your Web Database [online]. 15. 10. 2021 [cit. 2023-04-20]. Dostupné z: https://devm.io/databases/mysql-7-best-reasons-175591

[34] AMAZON. Amazon S3 Features - Amazon Web Services [online]. 2023 [cit. 2023-04-20]. Dostupné z: https://aws.amazon.com/s3/features/

[35] MAILGUN INC. Mailgun REST API Documentation [online]. 9. 5. 2023. [cit. 2023-04-21]. Dostupné z: https://documentation.mailgun.com/_/downloads/en/latest/pdf/

[36] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. [cit. 2023-04-20]. ISBN 978-80-251-1503-9.

[37] KOIVUVIITA, Jouni, Marco COLLOVATI, aj. *Grid | Components | Vaadin Docs* [online]. 14. 10. 2022 [cit. 2023-04-20]. Dostupné z: https://vaadin.com/docs/v23/components/grid

[38] KOIVUVIITA, Jouni, Marco COLLOVATI, aj. *Form Layout | Components | Vaadin Docs* [online]. 14. 10. 2022 [cit. 2023-04-20]. Dostupné z: https://vaadin.com/docs/v23/components/form-layout

[39] KOIVUVIITA, Jouni, Marco COLLOVATI, aj. *Data Binding | Vaadin Docs* [online]. 14. 10. 2022 [cit. 2023-04-20]. Dostupné z: https://vaadin.com/docs/v23/binding-data

[40] ZAGRYVYI, Sergii. Mailgun/mailgun-java: Java SDK for integrating with Mailgun [online]. 28. 8. 2022, 11. 2. 2023 [cit. 2023-04-20]. Dostupné z: https://github.com/mailgun/mailgun-java

[41] AMAZON. Working with Amazon S3 objects - AWS SDK for Java 2.x [online]. 2023 [cit. 2023-04-20]. Dostupné z: https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/examples-s3-objects.html

[42] MINISTERSTVO FINANCÍ ČR. ARES - Administrativní registr ekonomických subjektů: ARES - Popis [online]. 2013 [cit. 2023-04-20]. Dostupné z: https://wwwinfo.mfcr.cz/ares/

[43] GODOY, Javier a Felipe LANG. Year Month Calendar Add-on | Vaadin Add-on Directory [online]. 3. 4. 2023. 17. 8. 2022 [cit. 2023-04-20]. Dostupné z: https://vaadin.com/directory/component/year-month-calendar-add-on

[44] WEICHETOVÁ, Lenka. Využívání informačních a komunikačních technologií v domácnostech a mezi jednotlivci v roce 2022. Praha: Český statistický úřad, 2022. Služby. ISBN 978-80-250-3289-3. Dostupné také z: https://www.czso.cz/documents/10180/164606768/06200422.pdf/1c5c22c0-8941-4670-9698-e949482b0c35?version=1.3

[45] KOIVUVIITA, Jouni, Marco COLLOVATI, aj. *Board | Components | Vaadin Docs* [online]. 14. 10. 2022 [cit. 2023-04-20]. Dostupné z: https://vaadin.com/docs/v23/components/board

[46] KOIVUVIITA, Jouni, Marco COLLOVATI, aj. *App Layout | Components | Vaadin Docs* [online]. 14. 10. 2022 [cit. 2023-04-20]. Dostupné z: https://vaadin.com/docs/v23/components/app-layout

[47] LUND, Tatu. How do I make a responsive Grid that has a different set of columns depending on the browser width. - Vaadin Cookbook [online]. 3. 8. 2020. [cit. 2023-04-21]. Dostupné z: https://cookbook.vaadin.com/responsive-grid

[48] KOIVUVIITA, Jouni, Marco COLLOVATI, aj. *Deploying to Production | Components | Vaadin Docs* [online]. 14. 10. 2022 [cit. 2023-04-20]. Dostupné z: https://vaadin.com/docs/v23/production

[49] KOIVUVIITA, Jouni, Marco COLLOVATI, aj. *Teseting | Vaadin Docs* [online]. 14. 10. 2022 [cit. 2023-04-20]. Dostupné z: https://vaadin.com/docs/v23/testing

[50] KOIVUVIITA, Jouni, Marco COLLOVATI, aj. *End-to-end testing | Testing | Vaadin Docs* [online]. 14. 10. 2022 [cit. 2023-04-20]. Dostupné z: https://vaadin.com/docs/v23/testing/end-to-end

[51] KOIVUVIITA, Jouni, Marco COLLOVATI, aj. *Deploying a Vaadin Application to Heroku | Components | Vaadin Docs* [online]. 14. 10. 2022 [cit. 2023-04-20]. Dostupné z: https://vaadin.com/docs/v23/production/cloud-providers/heroku

[52] KOIVUVIITA, Jouni, Marco COLLOVATI, aj. *Deploying Using Docker | Components | Vaadin Docs* [online]. 14. 10. 2022 [cit. 2023-04-20]. Dostupné z: https://vaadin.com/docs/v23/production/docker