Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Návrh a implementace webového rozhraní pro správu a dohled chytrých zařízení

Bc. Ondřej Burda

Diplomová práce

2021

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Ondřej Burda**
Osobní číslo:	**I18236**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Návrh a implementace webového rozhraní pro správu a dohled chytrých zařízení**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

V teoretické části práce budou popsány současné trendy v oblasti chytrých zařízení tzv. IoT a to jak z pohledu technologií tak aplikací.
Dále budou popsány veškeré technologie, které budou finálně pro realizace práce vybrány.
V rámci vlastní implementace bude třeba navrhnout vhodný databázový model a navrhnout samotný software pro správu a dohled chytrých zařízení s využitím frameworku JAVA Spring a technologie Vaadin. Systém bude vybudován jako webová aplikace a bude umožňovat správu jednotlivých zařízení, a to včetně dohledového panelu (tzv. dashboard). Finální aplikace bude nasazena na vhodném webovém serveru.

Rozsah pracovní zprávy: **50-60 stran**
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.
Vaadin Team. *Book of Vaadin: Vaadin 14* Publisher: Independently published, 2019. 571 pages. ISBN 978-1692121440.

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

Datum zadání diplomové práce: **6. listopadu 2020**
Termín odevzdání diplomové práce: **15. května 2021**

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 2020

Prohlašuji:

Tuto práci 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., autorský zákon, 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. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 20. srpna 2021

Bc. Ondřej Burda

PODĚKOVÁNÍ

Chtěl bych poděkovat Ing. Janu Fikejzovi, Ph.D. za možnost zpracování této práce, cenné rady a odborné vedení. Děkuji také své rodině, která mě po dobu studia na vysoké škole výrazně podporovala.

ANOTACE

Diplomová práce se zaměřuje na proces vývoje webové aplikace s orientací na správu a dohled zařízení pro monitorování spotřeby energií v rámci sledovaných objektů. Práce je rozdělena na dva logické celky. V první části se zabývá představením a moderními trendy internetu věcí. Dále je zde provedena analýza použitých technologií a jejich důkladný popis. Ve druhé části se práce soustředí na návrh a implementaci konkrétní webové aplikace. Jsou zde předvedeny ukázky uživatelského rozhraní, celkového návrhu i technologických řešení stěžejních funkcionalit.

KLÍČOVÁ SLOVA

Vaadin, internet věcí, webová aplikace, Java, Spring, uživatelské rozhraní, návrh software, Apache Tomcat, Heroku

TITLE

Design and implementation of a web application for the management and supervision of smart devices

ANNOTATION

This diploma thesis focuses on the process of developing a web application focusing on the management and supervision of energy monitoring devices within monitored facilities. The thesis is divided into two logical units. The first part deals with the introduction and modern trends of the Internet of Things. Furthermore, an analysis of the technologies used and their thorough description is presented. In the second part, the thesis focuses on the design and implementation of a specific web application. Examples of the user interface, overall design and technological solutions of core functionalities are presented.

KEY WORDS

Vaadin, Internet of Things, web application, Java, Spring, user interface, software design, Apache Tomcat, Heroku

Obsah

Seznam obrázků **9**

Seznam zdrojových kódů **10**

Seznam zkratek **11**

Úvod **12**

1 Internet věcí **13**

1.1 IoT obecně 13

1.2 Současné moderní trendy 14

1.2.1 Chytrá domácnost 14

1.2.2 IoT ve zdravotnictví 15

1.2.3 Blockchain a bezpečnost 16

1.2.4 Chytrá města 16

1.2.5 Umělá inteligence společně s IoT 17

2 Použité technologie **18**

2.1 Spring 18

2.1.1 Hlavní přednosti Spring 19

2.2 Spring Boot 22

2.3 MySQL 23

2.4 Docker 25

2.4.1 Kontejnerizace vs. Virtualizace 26

2.4.2 Architektura Docker platformy 27

2.5 IntelliJ Idea 29

3 Vaadin framework **30**

3.1 Obecně o Vaadin 30

3.2 Historie a verze projektu 33

3.3 Data binding 34

3.4 Vaadin Designer 35

4 Návrh a implementace **37**

4.1 Požadavky na systém 37

4.1.1 Funkční požadavky 38
4.1.2 Nefunkční požadavky 40
4.1.3 Přehled a správa sensorů a hardware 41
4.1.4 Správa uživatelů 41
4.1.5 Dohledový panel 41
4.2 Návrh architektury a struktury 41
4.3 Datový model 42
4.3.1 Sensory 42
4.3.2 Hardware 44
4.3.3 Uživatel 44
4.4 Uživatelské rozhraní 45
4.4.1 Přihlašovací obrazovka 46
4.4.2 Správa senzorů 47
4.4.3 Vytváření nových senzorů 50
4.4.4 Dashboard 51
4.4.5 Správa uživatelů 53
4.5 Technické řešení implementace 55
4.5.1 Diagram tříd 55
4.5.2 Ukázky stěžejních funkcí a technických řešení 56
5 Nasazení aplikace 63
5.1 Způsoby zabalení aplikace 63
5.2 Nasazení na webový server 63
5.3 Nasazení na platformu Heroku 64
Závěr 66
Použitá literatura 67
Přílohy 72

Seznam obrázků

Obrázek 1 – IoT produkty Google Nest 15
Obrázek 2 – Spring Initializr 23
Obrázek 3 – Model klient-server 25
Obrázek 4 – Srovnání kontejnerizace a virtualizace 26
Obrázek 5 – Architektura Docker platformy 28
Obrázek 6 – Vývojové prostředí IntelliJ Idea 29
Obrázek 7 – Vaadin Designer 36
Obrázek 8 – Architektura aplikace 42
Obrázek 9 – Model definující senzor a jeho data 43
Obrázek 10 – Model hardware zařízení 44
Obrázek 11 – Model uživatelských účtů 45
Obrázek 12 – Přihlašovací obrazovka 46
Obrázek 13 – Správa senzorů 48
Obrázek 14 – Detail senzoru snímajícího vodní tok 49
Obrázek 15 – Potvrzovací dialog mazání senzoru 49
Obrázek 16 – Export dat do souboru 50
Obrázek 17 – Vytvoření nového senzoru 51
Obrázek 18 – Panely dashboardu v horní části stránky 52
Obrázek 19 – Graf v dolní části stránky dashboardu 53
Obrázek 20 – Správa uživatelů 54
Obrázek 21 – Můj profil v režimu editace 54
Obrázek 22 – Diagram komponentních tříd a jejich nástrojů 56
Obrázek 23 – Ukázka nasazení aplikace na Heroku pomocí GitHub 65

Seznam zdrojových kódů

Zdrojový kód 1 – Ukázkový SQL dotaz 24
Zdrojový kód 2 – Ukázka data binding objektu User na formulář 34
Zdrojový kód 3 – Nastavení komponenty Grid 57
Zdrojový kód 4 – Vytvoření sloupce tabulky 57
Zdrojový kód 5 – Automatická aktualizace dat v tabulce senzorů 58
Zdrojový kód 6 – Vytvoření Excel dokumentu 59
Zdrojový kód 7 – Naplnění buněk tabulky 59
Zdrojový kód 8 – Tvorba grafu 60
Zdrojový kód 9 – Vytvoření dialogového okna 61
Zdrojový kód 10 – Ukázka data binding 61
Zdrojový kód 11 – Ukázka odeslání emailu 62
Zdrojový kód 12 – Nastavení použití Gmail 62

Seznam zkratek

AIoT	Artificial Intelligence of Things
AJAX	Asynchronous JavaScript and XML
AOP	Aspect Oriented Programming
API	Application Programming Interface
AWT	Abstract Window Toolkit
CLI	Command Line Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DAO	Data Access Object
DI	Dependency Injection
DNS	Domain Name Systém
DOM	Document Object Model
EJB	Enterprise Java Beans
GB	Gigabyte
GWT	Google Web Toolkit
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoC	Inversion of Control
IoT	Internet of Things
IT	Information Technology
JDK	Java Development Kit
JEE	Java Enterprise Edition
JPA	Java Persistence Api
JSF	JavaServer Faces
JVM	Java Virtual Machine
LTS	Long Term Support
LXC	LinuX Containers
MB	Megabyte
MVC	Model View Controller
ORM	Object-Relational Mapping
PaaS	Platform as a Service
POJO	Plain Old Java Object
REST	Representational State Transfer
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
SWT	Standard Widget Toolkit
UI	User Interface
UX	User Experience
XML	Extensible Markup Language

Úvod

Moderní svět se neustále vyvíjí a co bylo včera považováno za revoluční může být zítra nahrazeno něčím novým a lepším. S příchodem průmyslu 4.0 se ve světě spustil nezastavitelný trend digitalizace a téměř každá technologická firma má ambice stát se jeho nedílnou součástí. Jedním z dynamicky se rozvíjejících odvětví tohoto novodobého průmyslu je Internet of Things neboli Internet věcí. Internet věcí bývá často označován jako jedna z nejdůležitějších a nejprogresivnějších technologií 21. století a skýtá obrovský potenciál v mnoha různých oborech. Mezi ně patří například výroba, zdravotnictví, doprava, bezpečnost a mnoho dalších.

Zařízení internetu věcí často generují velké množství dat. Tato data lze následně dále vyhodnocovat či v nich hledat určité závislosti. K tomu mohou sloužit specializované aplikace, které zároveň dávají možnost tyto přístroje centrálně spravovat a na základě jimi poskytovaných informací činit důležitá rozhodnutí. Z toho důvodu je stěžejní, aby tyto aplikace dokázaly zpracovat data a prezentovat je v přehledné formě, například pomocí grafů, obrázků a dalších grafických komponent. Návrh a implementace těchto aplikací je zásadní pro maximalizaci přínosu internetu věcí.

Cílem této práce je proto návrh a implementace takové webové aplikace, která bude uživateli umožňovat správu svých zařízení a bude poskytovat důležité graficky přívětivé informace v přehledné a moderní formě. Vývoj webových aplikací byl historicky poměrně složitým úkolem. Nyní se však pro tento proces dají využít mnohé nástroje a technologie, které práci značně usnadní. Přesto se ale vývojář musí potýkat s problémy v oblastech návrhu software, programování, grafiky, testování, nasazení, podpory a dalších. Část této práce je zaměřena na využití technologie Vaadin, která poskytuje vývojářům možnost vybudování webových aplikací, a to bez nutné znalosti standardních front-endových frameworků.

1 Internet věcí

Tato kapitola se zabývá hlavním teoretickým tématem diplomové práce, pro který byla následně vytvářena webová aplikace. Internet věcí (dále jen IoT) představuje velice moderní odvětví, kterému se díky obrovskému nárůstu chytrých zařízení vyplatí věnovat pozornost. Podle předpovědi společnosti Statista by počet IoT zařízení připojených ke globální internetové síti mohl v roce 2025 přesáhnout hranici 75 miliard, což by znamenalo téměř desetinásobek celosvětové lidské populace. V kapitole je vysvětlen přínos IoT. Velký důraz je kladen na představení moderních trendů a směrů v tomto odvětví a jejich konkrétních příkladů a aplikací ve společnosti.

Zdroje [1], [2].

1.1 IoT obecně

IoT lze definovat jako „*Síť propojených objektů (věcí), které jsou jednoznačně adresovatelné s tím, že tato síť je založena na standardizovaných komunikačních protokolech umožňující výměnu a sdílení dat a informací, jejichž analýzou bude možné docílit vyšší přidané hodnoty.*“ [3]. Síť v této definici představuje způsob, jakým jsou mezi sebou objekty propojeny. Nemusí se tedy jednat výhradně o internet, ale například o lokální vnitřní síť. Jako objekt se uvádí zařízení, které poskytuje data, například skrze senzor, který snímá určitou veličinu.

K čemu tedy IoT může sloužit si lze vysvětlit na následujícím příkladu z běžného života. Zaměstnanec, který se zrovna chystá na týdenní firemní cestu do zahraničí, si těsně před odjezdem vzpomene, že možná zapomněl vypnout klimatizaci v celém domě. Za normálních okolností, pokud by nechtěl utrácet peníze za zbytečně klimatizovaný dům ve své nepřítomnosti, by se musel vrátit domů a klimatizaci vypnout ručně. S příchodem IoT už se však do takové situace nikdo dostat nemusí. S rostoucím počtem zařízení připojených k internetu tak lze například jednoduše využít svůj mobilní telefon a skrze specializovanou aplikaci status klimatizace zjistit a změnit. Stejně tak lze například centrálně řídit a kontrolovat různé sensory ve výrobním procesu. Tímto způsobem se IoT výrazně podílí na moderním formování světa a technologické budoucnosti.

Zdroje [1], [3].

1.2 Současné moderní trendy

Až na několik hlavních směrů využití IoT se predikce pro moderní trendy v této oblasti mění každým rokem. Z pohledu technologie se jeví zcela zásadní příchod nejnovější bezdrátové sítě 5G. Ta umožňuje rychlejší, stabilnější a bezpečnější připojení, které bude mít zásadní dopad na IoT zařízení od autonomních vozidel přes inteligentní sítě pro obnovitelné zdroje energie až po roboty s umělou inteligencí využívané ve výrobních procesech.

Zdroje [1], [4].

1.2.1 Chytrá domácnost

Díky chytrým domácím řešením získává uživatel zcela novou úroveň kontroly nad vlastním domovem. První generace IoT zařízení se v tomto odvětví soustředila hlavně na přístup k zařízením a možnost je vzdáleně upravovat, vypínat a zapínat. Moderní chytrá domácnost tento koncept posouvá o několik kroků dál. Pomocí senzorů jednotlivá zařízení, spotřebiče a celé prostory v domě neustále shromažďují údaje o tom, jak je uživatel používá. Učí se o konkrétních zvycích a pomocí složitých algoritmů určují vzorce spotřeby, které následně dokážou značným způsobem zredukovat spotřebu energií v domácnosti.

Ideální příklady zařízení a technologií zaměřených přímo na IoT v domácnosti jsou Google Nest a TP-Link Smart Home. Smart Home od společnosti TP-Link nabízí chytré kamery, wifi systémy, žárovky, a hlavně chytré zásuvky, které tak umožňují vypínat a zapínat i zařízení, která v zásadě nejsou IoT zařízeními. V případě Nest se jedná o řadu produktů pro chytrou domácnost, která zahrnuje reproduktory, chytré displeje, streamovací zařízení, termostaty, detektory kouře, bezpečnostní zařízení, směrovače, zvonky a kamery, které jsou k vidění na následujícím obrázku 1. Obě tyto technologie lze ovládat skrze dedikované mobilní aplikace. Nest používá vlastní nativní aplikaci, Smart Home využívá bezplatnou aplikaci Kasa Smart.

Zdroje [5], [6].

Obrázek 1 – IoT produkty Google Nest [1]

1.2.2 IoT ve zdravotnictví

Odvětví zdravotnictví představuje jednu z nejvýznamnějších možností, jak mohou technologické podniky pozitivně ovlivnit úroveň a kvalitu života skrze využití IoT. Existuje mnoho aplikací, které přinášejí prospěch jak pacientům, tak i lékařům, nemocnicím a pojišťovnám. Vzdálené monitorování umožňuje například neustálé sledování zdravotního stavu zejména starších pacientů a jakoukoliv změnu nebo vychýlení od běžných činností okamžitě hlásit zodpovědným osobám (rodině, lékaři atd.). Dalším moderním trendem v této oblasti jsou nositelná zařízení, jako například fitness náramky nebo glukometry pro diabetiky.

Jako příklad lze uvést platformu Quio, která propojuje různá terapeutická zařízení související s léky, aktivitou a zdravím pacientů s chronickými onemocněními. Systém funguje zcela automatizovaně s podporou týmu odborníků na monitorování, kteří shromažďují a vyhodnocují zdravotní data získávaná ze speciálních senzorů, na jejichž základě může k uživateli vyslat tým zdravotníků.

Zdroje [7], [8].

[1] Obrázek převzat ze zdroje [6].

1.2.3 Blockchain a bezpečnost

Všeobecné přínosy IoT jsou značné a nezvratné. Přesto však existuje množství případů napadení konkrétních společností i jednotlivých zařízení a spolu s nejistotou ohledně osvědčených bezpečnostních postupech a spolehlivosti poskytovaných dat se může řada podniků zdráhat začlenění IoT technologií do svých procesů. V takových případech je nutné provést kompletní analýzu a vyhodnocení bezpečnostních rizik, které prověří zranitelnosti zařízení a síťových systémů. Potenciálních bezpečnostních problémů je mnoho, od špatné implementace obslužné aplikace, přes chyby v návrhu zařízení, nezabezpečenou komunikaci, až k samotnému ukládání a manipulaci s daty. Společnost Siemens předpokládá, že jedním z možných řešení problémů bezpečnosti souvisejících s daty se stane využití blockchain. Jedná se o speciální druh databáze, ve které jsou záznamy chráněny proti neoprávněnému zásahu. Svým jedinečným způsobem sdílení, který zajišťuje spolehlivost a dohledatelnost informací, se předpokládá zásadní vylepšení technologií IoT, protože zdroje dat lze v rámci blockchain snadno identifikovat a data zůstávají neměnná, což výrazně zvýší důvěru v jejich využívání.

Zdroje [7], [9], [10].

1.2.4 Chytrá města

V současnosti již existují IoT technologie, přístroje a kompletní software, které slouží k monitorování dopravy, provozu veřejných zařízení a správě budov v rozsáhlých metropolích. Chytrá města tak již rozhodně nejsou věcí budoucnosti. Existuje nespočet využití IoT v automatizaci měst, což je koncept, kterým se zabývá například firma Siemens se svým projektem Digital City Solutions, který v současnosti funguje v Hong Kongu, Singapuru a dalších velkoměstech. Dvě oblasti, kde se očekává zásadní inovace a pozornost technologických gigantů, jsou chytrá veřejná doprava a správa energií. Očekává se například, že technologie internetu věcí umožní lépe porozumět potřebám občanů a jejich vzorcům pohybu, což v případě přelidněných měst se zatíženou dopravou dokáže výrazně zlepšit životní úroveň občanů a výrazně snížit například čas potřebný na dojíždění do zaměstnání.

Příklad šetření energie lze nalézt například v Amsterdamu, kde se intenzita osvětlení pouličních lamp v různých ulicích automaticky mění podle vytíženosti chodci, čímž se snižuje energetická náročnost. Koncepci chytrého města se snaží rozvíjet i Praha, a to hned několika různými projekty. Patří mezi ně například chytrý svoz odpadu, který využívá speciálních ultrazvukových

IoT senzorů umístěných v konkrétních odpadních kontejnerech, jejichž svoz je finančně i časově náročný.

Zdroje [7], [11], [12].

1.2.5 Umělá inteligence společně s IoT

Umělá inteligence věcí (dále jen AIoT) spojuje technologie umělé inteligence s IoT. Ačkoliv se jedná o dvě nezávislá technologická odvětví, při správném využití a propojení dokážou vzájemně akcelerovat své schopnosti a výkon. „*Inteligence AIoT umožňuje analýzu dat, která se pak používají k optimalizaci systému a generování vyššího výkonu a obchodních poznatků a vytváření dat, která pomáhají činit lepší rozhodnutí a ze kterých se systém může učit.*“ [13]. Jako jeden z příkladů lze uvést využití AIoT v rámci chytré domácnosti. Mezi dostupnými produkty řady Google Nest se nachází i chytrý samoučící se termostat Nest Learning Thermostat (viditelný na obrázku 1). Prvek umělé inteligence v tomto zařízení slouží k naučení termostatu, aby se dokázal sám regulovat podle chování a preferencí obyvatel domácnosti a výrazně tak přispíval k úsporám na energii. Za pomoci dalších senzorů a například mobilní aplikace dokáže poznat, zda obyvatel opustil dům a vyrazil do práce, a následně se přepnout do úsporného režimu. Po několika dnech manuálního používání je termostat úplně soběstačný a dokáže řídit teplotu v celé domácnosti tak, aby vyhovovala jejím obyvatelům, a to o poznání efektivněji a úsporněji. Zároveň poskytuje přehledné hlášení o spotřebě energie a přichází s návrhy, jak tuto spotřebu ještě snížit a ušetřit.

Zdroje [7], [13].

2 Použité technologie

V následujících podkapitolách jsou popsána a vysvětlena témata, na kterých stojí základy technologického řešení této diplomové práce. Při výběru vhodných technologií byla uvažována následující kritéria:

- **Použitelnost** – technologie musejí být schopné v plném rozsahu podporovat a umožnit vyřešení všech požadavků práce.
- **Bezplatný software** – vzhledem k povaze diplomové práce bylo nezbytné vybrat takové technologie a software, které budou disponovat minimálně určitou bezplatnou verzí, jež nebude omezena natolik, aby bránila uskutečnění všech předem stanovených cílů práce.
- **Popularita a podpora** – zvolené technologie by měly být moderní, podporované a aktualizované tvůrcem a zároveň perspektivní z hlediska budoucího rozvoje IT.

Jako nejvhodnější technologie byl vybrán Java framework Spring v kombinaci s Vaadin, kterému je jakožto hlavnímu stavebnímu kamenu celé práce věnovaná samostatná kapitola 3. Dále byla při práci použita databáze MySQL, vývojové prostředí IntelliJ Idea a kontejnerizační platforma Docker.

2.1 Spring

Vývoj webových aplikací v jazyce Java byl historicky vždy poměrně velkým programátorským oříškem. V období před tím, než Sun Microsystems, tvůrci programovacího jazyka Java, přišli s novinkou EJB (Enterprise Java Beans), se museli vývojáři spoléhat na použití JavaBeans. Byť jde o užitečný nástroj, který pomáhal zejména při tvorbě komponent uživatelského rozhraní, neposkytoval služby, jako je například řízení transakcí a zabezpečení, bez kterých se vývoj rozsáhlých podnikových aplikací neobejde. Konkrétní řešení těchto problémů se objevilo příchodem právě zmiňovaného Enterprise JavaBeans API. Toto rozhraní poskytlo metody a služby, které vývoj robustních podnikových aplikací dokázaly do určité míry usnadnit. Stále se ovšem jednalo z hlediska vývoje o poměrně komplikovaný proces.

Jako řešení komplikací spojených s vývojem pomocí EJB se v červnu roku 2003 objevila první verze nyní již extrémně populárního open source frameworku s názvem Spring. Tato platforma poskytuje nástroje umožňující vytvářet rozsáhlé podnikové aplikace v jazyce Java, a to velice rychle a poměrně jednoduše. Spring představil nové techniky, jako například POJO (Plain Old

Java Object), AOP (Aspect Oriented Programming) a hlavně DI (Dependency Injection), které odstranily složitosti spojené s používáním EJB a nabídly alternativní styl vývoje podnikových aplikací. Jádro frameworku je v základní verzi značně odlehčené a velikostně zabírá přibližně 2 MB. Spring jako celek lze proto považovat za určitý soubor dílčích frameworků, které lze při vývoji seskupovat dohromady a zajistit si tak lepší funkčnost webové aplikace a rychlejší a pohodlnější proces tvorby aplikace. Těmto dílčím frameworkům se říká vrstvy a patří mezi ně například Spring Web MVC, Spring AOP, Spring ORM (Spring Object-Relational Mapping) nebo Spring Web Flow. Nejen tyto vlastnosti a funkce se zasloužily o to, že: „*Spring je nejoblíbenějším frameworkem pro vývoj aplikací v podnikové Javě. Miliony vývojářů po celém světě používají framework Spring k vytváření vysoce výkonného, snadno testovatelného a opakovaně použitelného kódu.*“ [15].

Aktuálně nejnovější finální verze páté generace Spring frameworku 5.3.x slibuje dlouhodobou podporu pro jednotlivé LTS verze Java Development Kit, a to dokonce pro ještě nevydané JDK 17, jehož vydání je plánované na září letošního roku 2021.

Zdroje [14], [15].

2.1.1 Hlavní přednosti Spring

Charakter frameworku Spring, jako jsou správa transakcí, IoC (Inversion of Control) a AoP jej na poli Java frameworků činí naprosto unikátním. Některé z nejdůležitějších vlastností a možností frameworku Spring jsou následující:

- **Plain Old Java Object** – Spring umožňuje vývojářům vyvíjet robustní podnikové aplikace pomocí POJO, což znamená, že vývojář v podstatě popisuje jednoduchý typ bez odkazů na konkrétní frameworky a zároveň nemusí dodržovat žádné konvence pro pojmenování vlastností a metod implementované třídy. Zásadní výhodou použití pouze POJO je, že k vývoji takové aplikace není zapotřebí žádný EJB kontejner, což by mohl být například konkrétní aplikační server, ale Spring umožnuje použít pouze robustnější servlet kontejner, jako je Tomcat, Undertow, Jetty nebo jiný komerční produkt.
- **IoC container** – jedná se o základní kontejner jádra frameworku, který využívá návrhového vzoru IoC, konkrétně implementaci DI. Inversion of Control, česky označováno jako „obrácené řízení“, funguje tak, že přenáší odpovědnost za vytváření a propojování objektů z aplikace přímo na konkrétní framework. Přesně o to se stará IoC container, vytváří objekty, propojuje je mezi sebou, konfiguruje a stará se o řízení jejich

životního cyklu od vytvoření až po jejich zánik. Dependency Injection, do češtiny často překládané jako „vkládání závislostí“, poté řeší konkrétní způsob vkládání objektů. Jako základní způsoby této metody se označuje trojice Setter (property) Injection, Constructor Injection a Interface Injection. Kontejner získává pokyny, jaké objekty má instancovat, konfigurovat a sestavit čtením poskytnutých konfiguračních metadat. Ta mohou být reprezentována třemi způsoby, a to buď pomocí XML, anotacemi v jazyce Java, nebo kódem v jazyce Java. Tyto objekty se nazývají Beans a tvoří páteř ve Spring frameworku vytvořených aplikací.

- **Modularita** – Spring se skládá z částí uspořádaných do několika různých modulů (sub-frameworků), kterých poskytuje opravdu spoustu a pokrývají všechna možná odvětví. Při vývoji se ovšem stačí zaměřit na využití pouze těch modulů potřebných k řešení konkrétních úloh a povaze aplikace.
- **Integrace s existujícími frameworky** – jednou ze základních myšlenek Spring frameworku je možnost volby. V obecném smyslu tedy nenutí používat nebo kupovat nějakou konkrétní architekturu, technologii nebo metodiku (i když z různých důvodů doporučuje některé před jinými) a zároveň když něco od konkurence funguje, není třeba to vymýšlet znovu a jinak. Tímto heslem se Spring řídí a umožňuje integraci s mnoha dalšími frameworky a technologiemi, jako jsou různé ORM frameworky, logovací frameworky (například Log4J od firmy Apache), JEE, Quartz a další. Ukázkový příklad se prezentuje v oblasti webu, kde Spring poskytuje vlastní webový modul Web MVC, ale zároveň umožňuje integraci s řadou dalších populárních webových frameworků třetích stran, jako například Struts nebo JSF (JavaServer Faces), čímž vývojáře nesvazuje, na rozdíl od některých konkurenčních produktů.
- **Web MVC Framework –** velká a silná zbraň, která umožňuje vytvářet webové aplikace založené na populární MVC (Model View Controller) architektuře. Disponuje mnoha hotovými komponenty, které lze použít k vývoji flexibilních a volně provázaných webových aplikací. Návrhový Vzor MVC slouží k oddělení různých aspektů aplikace na tři samostatné sekce, které mezi sebou spolupracují. Model představuje datovou logiku a reprezentaci informací, View je zodpovědný za převod dat z modelu do podoby prezentované uživateli v podobě UI a Controller řeší příchozí události (typicky zapříčiněné uživatelovou interakcí v UI) a zajišťuje změnu dat v modelu.

- **Správa transakcí** – jedná se o velice důležitou součást jakékoliv podnikové aplikace orientované na relační databáze, která zajišťuje integritu a konzistenci dat. Spring poskytuje spolehlivé transakční rozhraní, které pomáhá při správě transakcí aplikace bez zásahu do jejího kódu. Pro globální transakce spravované aplikačním serverem se využívá Java Transaction API (JTA) a pro lokální transakce lze použít JDBC, Hibernate, Java Data Objects (JDO) nebo některý z jiných API či frameworků určených pro řešení přístupu k datům.
- **Testování** – jakožto nedílnou součást vývoje robustních podnikových aplikací se Spring výrazně podílí i na zjednodušení procesu testování. Vývojářem napsané POJO třídy by měly být testovatelné v nyní už klasických JUnit nebo TestNG testech. Díky funkci Dependency Injection by měl být kód méně závislý na kontejneru, než by tomu bylo při tradičním vývoji v prostředí Java EE. K izolovanému testování kódu se dají použít mock objekty (ve spojení s dalšími hodnotnými technikami testování). Metoda mockingu se dá využít například při testování rozhraní DAO nebo úložiště, aniž by bylo zapotřebí při provádění jednotkových testů přistupovat do konkrétní databáze. Součástí Spring je hned několik balíčků určených pro podporu mockování a využit může být i v kombinaci s jiným frameworkem třetí strany, například populární open source projekt Mockito. Při vývoji software je důležité, aby mohl vývojový tým provádět všechny potřebné integrační testy, aniž by musel nasazovat produkt do svého aplikačního serveru nebo řešit připojení k podnikové infrastruktuře. Kromě jednotkových testů proto Spring poskytuje i prvotřídní podporu pro integrační testování skrze spring-test modul.

Zdroje [14], [15], [16].

2.2 Spring Boot

Spring je široce používán pro vytváření škálovatelných aplikací a dokáže usnadnit spoustu práce a ušetřit vývojáři cenný čas, přesto však disponuje i některými značnými nevýhodami. Hlavní nevýhodou v tomto frameworku vytvořených projektů je fakt, že jejich konfigurace je opravdu časově náročná a pro začínající vývojáře může být poněkud obtížná a odrazující. Pro nováčka, který se Spring frameworkem začíná, zabere zpracování aplikace do stavu připraveného k nasazení na produkci nějaký ten čas navíc, který by jinak mohl investovat do jiných důležitých částí produktu.

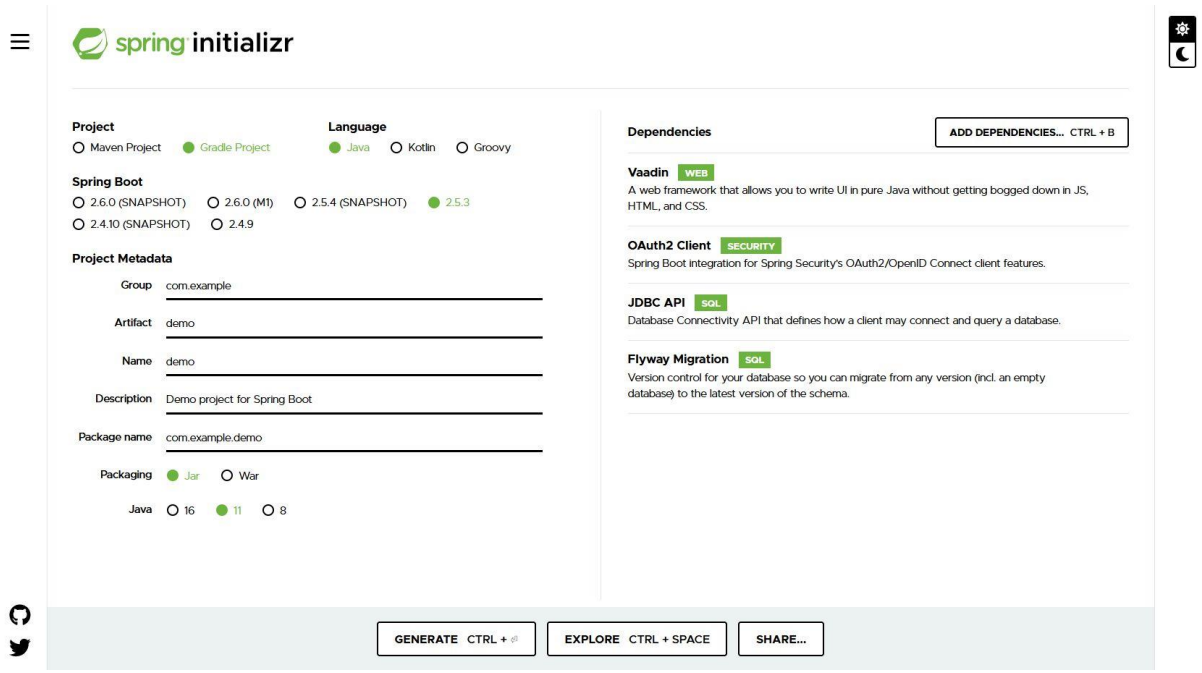
Jako řešení tohoto problému byl vytvořen Spring Boot. Jedná se o nástroj, který je postaven nad Spring frameworkem a obsahuje tak všechny jeho funkce a vlastnosti a slouží zejména k co nejjednoduššímu vytváření webových aplikací či mikroslužeb připravených přímo k rychlému a snadnému uvedení do produkce. S trochou nadsázky se dá říct, že Spring Boot je vlastně jakýsi inicializátor projektu založený na platformě Spring, který je připraven k nasazení na produkční prostředí. Díky funkcím, jako je automatická konfigurace, ušetří vývojáři psaní zdlouhavého kódu a pomůže mu vyhnout se zbytečnému nastavování. Jako hlavní výhody Spring Boot se označují následující vlastnosti:

- Umožňuje vyhnout se náročné konfiguraci XML, která je v čistém Spring Frameworku nutností.
- Poskytuje snadnou údržbu a vytváření REST API, a to zejména pomocí jednoduchých anotací.
- Obsahuje vestavěný Tomcat server.
- Pomáhá snižovat počet řádků v kódu a celkovou velikost projektu.
- Nabízí funkce připravené k uvedení aplikace do produkčního prostředí na aplikační server či do cloudu.
- Snadnější přizpůsobení požadavkům vývojáře a následná správa.
- Podporuje typ architektury založený na mikroslužbách.

Spring Boot spatřil světlo světa v roce 2014 a okamžitě se mezi vývojáři stal velice oblíbeným nástrojem. Dle výzkumu[2] společnosti Snyk z roku 2020, kterého se zúčastnilo více než 2000 respondentů z různých částí světa, je přesně 60% Java vývojářů ve svých projektech závislých na Spring Frameworku a v 50 % případů využívají Spring Boot na straně serveru.

Projekt v tomto frameworku se dá založit přímo ve vývojovém prostředí nebo na oficiální stránce https://start.spring.io (obrázek 2 níže), která umožňuje vytvořit předpřipravený projekt s konkrétním nastavením, přidanými závislostmi a následně si jej jednoduše stáhnout a začít pracovat.

Zdroje [17], [18], [19].



Obrázek 2 – Spring Initializr

2.3 MySQL

MySQL je open source systém pro správu relačních databází s modelem klient-server, který je založen na SQL (strukturovaném dotazovacím jazyku). Předchozí věta disponuje hned dvěma důležitými termíny, jejichž pochopení a znalost je pro ovládnutí MySQL klíčová. Jedná se o relační databáze a strukturovaný dotazovací jazyk. Databázi je možné jednoduše definovat jako místo, ve kterém jsou uložena a organizována data. Slovo "relační" v tomto případě znamená,

[2] Dostupné ze zdroje [17].

že data uložená v databázi jsou uspořádána ve formě tabulek. SQL je doménově specifický jazyk používaný a určený pro správu dat uložených v databázích. Jednoduše řečeno, SQL poskytuje způsob, jak dávat serveru pokyny k provedení určitých operací týkajících se databáze. Jako reálný příklad uvažujme situaci, kdy vlastník firmy nabízející určité produkty a služby chce zjistit jména, příjmení a emailové adresy všech svých zákazníků z města Pardubice, kteří jsou starší 40 let, za účelem rozeslání cílené nabídky svých nových produktů a služeb. Veškerá zákaznická data má skrze firemní informační systém pečlivě uložena v MySQL databázi. SQL dotaz pro zjištění těchto údajů by vypadal následovně:

```
SELECT name, surname, email FROM customers

WHERE city = 'Pardubice' AND age > 40;
```

Zdrojový kód 1 – Ukázkový SQL dotaz

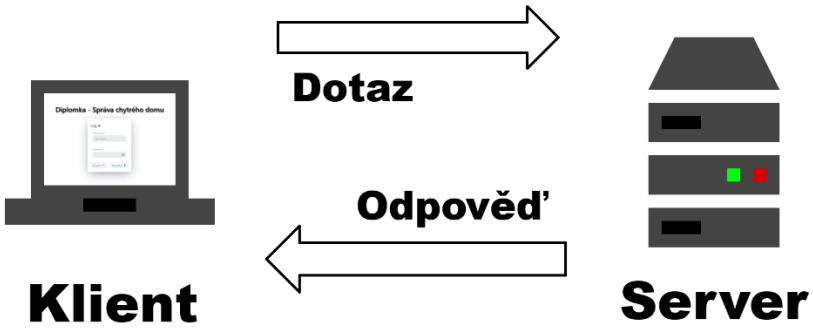
Na pohled je zřejmé, že jazyk SQL je velice intuitivní a jeho syntaxe připomíná běžnou anglickou větu. MySQL vyvíjí, distribuuje a podporuje společnost Oracle Corporation.

„*MySQL je jednou z nejznámějších technologií v moderním ekosystému velkých dat. Často bývá označována za nejoblíbenější databázi a v současné době se těší širokému a efektivnímu využití bez ohledu na odvětví, je tedy zřejmé, že každý, kdo se zabývá podnikovými daty nebo obecně IT, by měl usilovat alespoň o základní znalost MySQL.*“ [20].

Velkou výhodou tohoto systému je zejména to, že i vývojáři, kteří mají s relačními databázemi malou nebo úplně nulovou zkušenost, mohou téměř okamžitě vytvářet rychlé, výkonné a bezpečné systémy pro ukládání dat. Programová syntaxe a rozhraní MySQL jsou svojí jednoduchostí dokonalou vstupní branou do širokého světa dalších populárních dotazovacích jazyků a strukturovaných datových úložišť. Začínající vývojář jistě ocení i značné množství literatury a přehlednou dokumentaci. Mezi firmy využívající MySQL ve svých aplikacích a projektech patří i takoví giganti jako Facebook, YouTube, Twitter, NASA, BBC, Spotify, Tesla, a dokonce i Námořnictvo Spojených států amerických.

Model klient-server (na obrázku 3), který MySQL využívá, je v jádru věci opravdu jednoduchý. Jedná se o vztah mezi dvěma účastníky síťového provozu, kteří zastávají různé role. První, nazýván jako klient, žádá ten druhý, zvaný server, o poskytnutí určitých služeb. Klient tedy zašle žádost jednomu nebo případně i více připojeným serverům a ty se rozhodnou, zda žádost akceptují, zpracují a vrátí požadovanou službu nebo data. Na tomto typu síťové architektury je založeno velké procento obchodních či firemních aplikací, některé internetové protokoly jako například HTTP, SMTP, DNS, přístup k databázi a také dnes už velice častá komunikace mezi

webovým prohlížečem a webovým serverem. Mezi hlavní výhody tohoto modelu patří jednoduchost, snadnější údržba, zabezpečení a rychlost při aktualizování dat uložených na serveru. Jako hlavní nevýhody se uvádí problém přetěžování sítě a závislost na konkrétním serveru. Pokud dojde k jeho přetížení či výpadku, žádosti klientů tak zůstanou bez odpovědi a nemohou být vyslyšeny.



Obrázek 3 – Model klient-server

Opakem architektury klient-server je model peer-to-peer (někdy také nazývaný jako klient-klient), ve kterém mezi sebou komunikují jednotliví klienti a zastávají jak roli klienta, tak roli serveru, což znamená, že všechny uzly této sítě jsou si tak rovnocenné.

2.4 **Docker**

Docker je open source platforma pro vývoj, distribuci a provoz aplikací. Dokáže oddělit aplikace od infrastruktury za účelem zkrácení prodlevy mezi napsáním kódu a jeho následným spuštěním v produkci.

„*Docker umožňuje vývojářům balit aplikace do kontejnerů – standardizovaných spustitelných komponent, které kombinují zdrojový kód aplikace s knihovnami operačního systému a závislostmi potřebnými ke spuštění tohoto kódu v libovolném prostředí. Kontejnery zjednodušují poskytování distribuovaných aplikací a stávají se stále populárnějšími s tím, jak organizace přecházejí na vývoj v cloudu a hybridní multicloudová prostředí.*“ [24].

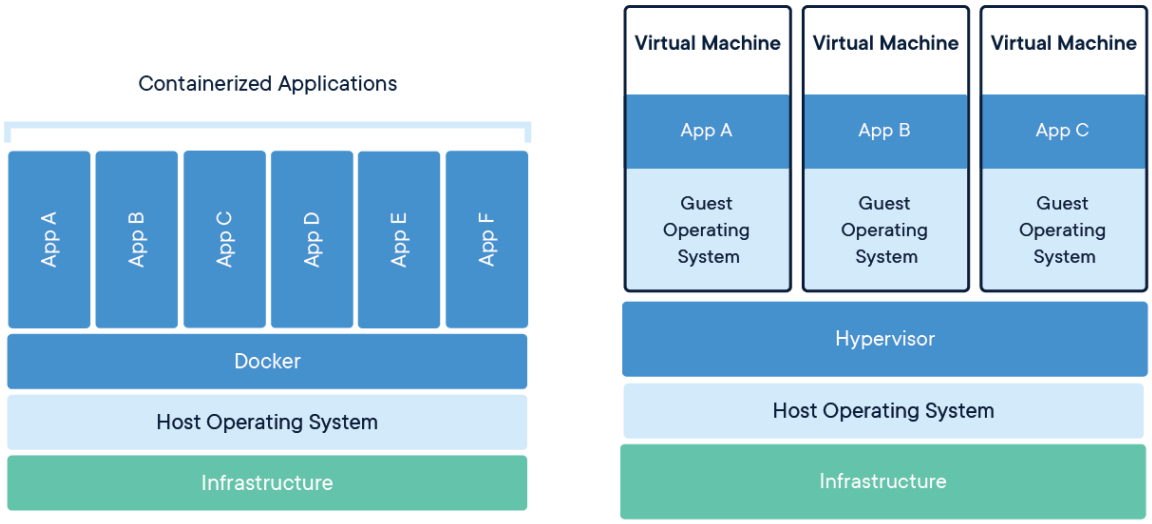
Kontejnery jsou odlehčené a obsahují vše potřebné pro běh aplikace, takže vývojář nemusí vůbec řešit, co je na hostiteli aktuálně nainstalováno a jakým disponuje hardware. Jednotlivé kontejnery lze snadno sdílet a mít jistotu, že každý dostane ten samý kontejner, který funguje stejným způsobem. Při vývoji se takto dá například simulovat produkční prostředí nebo zajistit,

aby měl každý vývojář k dispozici tu samou předem navrženou testovací databázi naplněnou stejnými daty. Kontejnery lze použít lokálně na konkrétním počítači, na fyzických nebo virtuálních strojích v datových centrech, u poskytovatelů různých cloudových služeb nebo dokonce v kombinaci těchto prostředí. V podnikovém prostředí usnadňuje přenositelnost a odlehčená povaha Docker platformy lepší správu pracovní zátěže a škálování podle aktuálních potřeb. Nutno podotknout, že kontejnerizace je technologie používaná ještě před příchodem Docker platformy. V roce 2008 se objevila první nejucelenější implementace správce kontejnerů v Linuxu označovaná jako LXC (LinuX Containers) a někteří vývojáři ji stále využívají, stejně jako jiné konkurenční produkty, například rkt nebo Podman.

Zdroje [24], [25].

2.4.1 Kontejnerizace vs. Virtualizace

Je velice důležité nezaměňovat pojmy kontejnerizace a virtualizace. Kontejnery a virtuální stroje mají totiž dost podobné výhody izolace a přidělování prostředků, ale fungují velice rozdílně, protože kontejnery virtualizují operační systém namísto hardwaru. Tento rozdíl je patrný z obrázku 4. Přesně z tohoto důvodu jsou kontejnery přenosnější a efektivnější.



Obrázek 4 – Srovnání kontejnerizace a virtualizace [3]

[3] Obrázek převzat ze zdroje [26].

Kontejnery jsou abstrakcí na aplikační vrstvě, která balí kód aplikace a další její závislosti dohromady. Oproti tomu virtualizace poskytuje virtuální stroje, které jsou abstrakcí fyzického hardwaru. Na jednom počítači dokáže současně běžet více kontejnerů, které mezi sebou sdílí jádro operačního systému a běží jako izolované procesy v konkrétním uživatelském prostoru. Podobně umožňuje Hypervizor, ať už nativní (například VMWare ESXi nebo Hyper-V), nebo hostovaný (VirtualBox, VMWare Player a další), provozovat více virtuálních počítačů na jednom fyzickém stroji. Nevýhodou se zde v případě virtualizace stává fakt, že každý z těchto virtuálních počítačů obsahuje kopii určitého operačního systému se všemi aplikacemi, binárními soubory a knihovnami, a to i těmi, které by k běhu vyvíjené aplikace nebyly zapotřebí. Takový virtuální počítač dokáže zabírat až desítky GB místa. Oproti tomu obrazy kontejnerů zabírají obvykle několik desítek MB, a to z důvodu, že neobsahují zbytečné soubory a knihovny, ale pouze ty, které jsou nezbytně nutné ke správnému běhu aplikace. Doba spouštění hraje taktéž ve prospěch kontejnerů, a to poměrně razantním rozdílem.

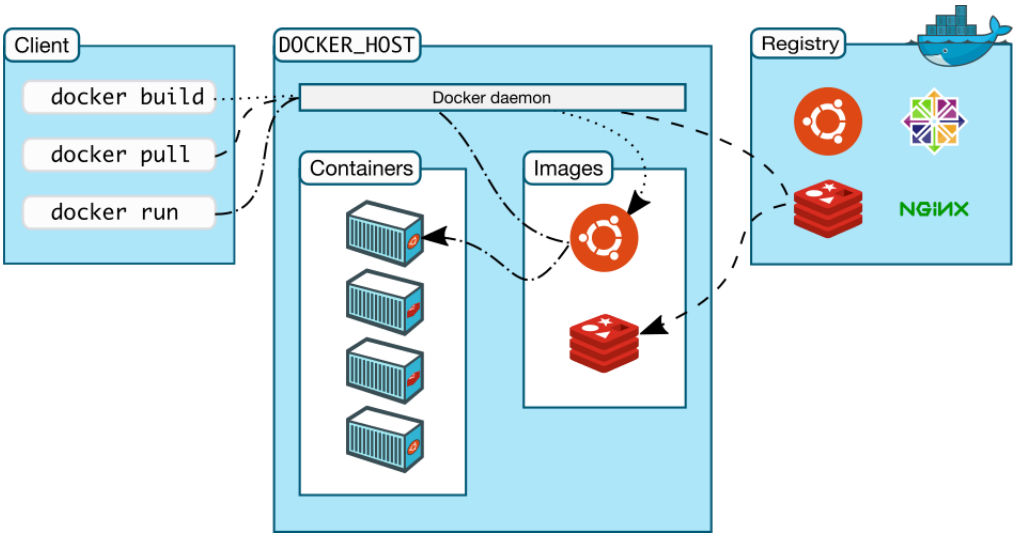
Zdroje [26].

2.4.2 Architektura Docker platformy

Docker používá architekturu typu klient-server. Klient pomocí jednoduchých příkazů komunikuje s Docker démonem, který navenek zastupuje serverový celek Docker Host (k vidění na obrázku 5). Ten poskytuje kompletní prostředí pro spouštění a provoz aplikací. Skládá se z Docker démona, obrazů, kontejnerů, sítí a úložiště.

Docker klient umožňuje uživatelům komunikovat s nástrojem Docker skrze démona. Nacházet se může na stejném hostitelském stroji jako démon, nebo se může připojit k jinému démonovi na vzdáleném hostiteli a může komunikovat s více než jedním. Při použití příkazu, jako je například *docker run*, klient skrze Docker API odešle tento příkaz démonovi dockerd, který ho následně vykoná. Rozhraní, které klient poskytuje, je ve formě klasického příkazového řádku. Hlavním účelem je poskytnout prostředky pro přímé stažení obrazů z registrů a jejich spuštění na hostiteli Docker. Nejpoužívanější příkazy, se kterými by se měl vývojář pracující s Docker platformou seznámit, je trojice *docker build*, *docker pull* a *docker run*.

Docker démon naslouchá požadavkům rozhraní Docker API a spravuje objekty, jako jsou obrazy, kontejnery, sítě a svazky. Démon může také komunikovat s jinými démony.



Obrázek 5 – Architektura Docker platformy [4]

Docker obrazy jsou šablony s přesnými pokyny pro vytvoření kontejneru. Obrazy lze získat z oficiálních registrů, kam je publikují konkrétní uživatelé a firmy, nebo je možné si vytvořit vlastní. Příkladem může být třeba oficiální obraz Ubuntu, který lze následně rozšířit o obraz některého webového serveru a přidat další konfigurace a vlastní aplikace.

Docker kontejner je spustitelná instance konkrétního obrazu. Ve výchozím stavu je kontejner izolovaný od hostitelského systému.

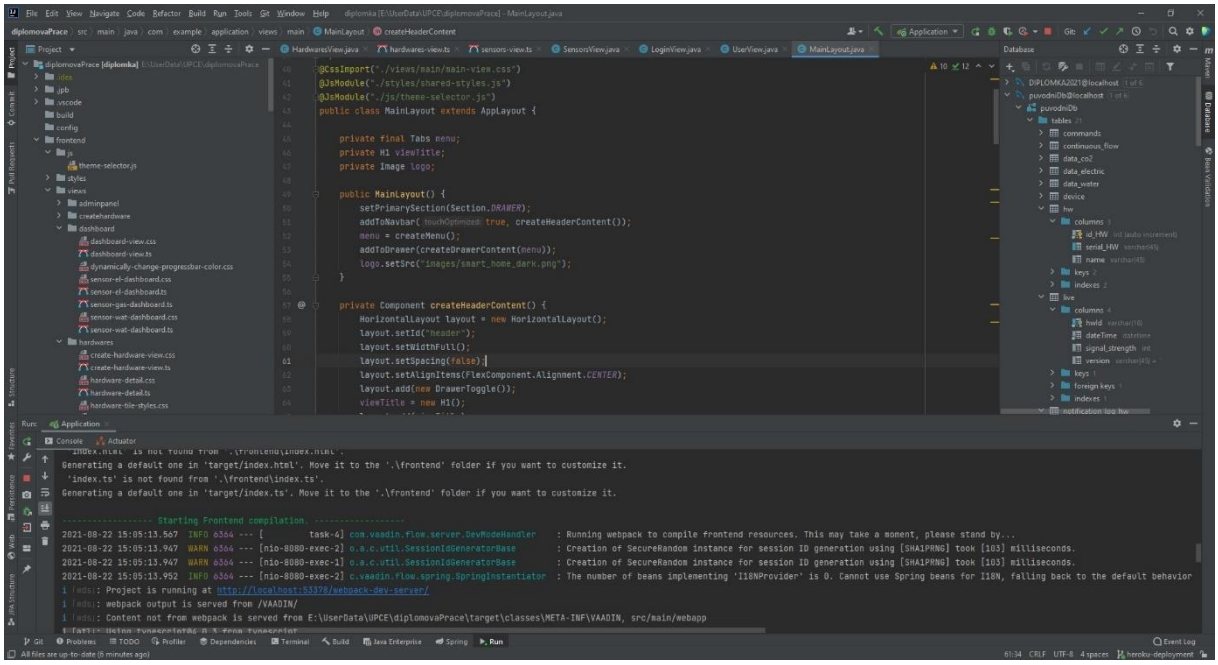
O Docker se tak dá říct, že je to v podstatě sada nástrojů, která umožňuje sestavovat, nasazovat, aktualizovat, spouštět a zastavovat kontejnery pomocí jednoduchých příkazů.

Zdroje [24], [25].

[4] Obrázek převzat ze zdroje [25].

2.5 **IntelliJ Idea**

IntelliJ Idea, k vidění na obrázku 6, je integrované vývojové prostředí (dále jen IDE) od společnosti JetBrains, navržené tak, aby maximalizovalo produktivitu vývojářů a usnadňovalo proces vývoje. Jedná se o multiplatformní IDE dostupné na operačních systémech Windows, macOS a Linux. Chytré doplňování kódu, statická analýza a refaktorizace jsou jen jedny z mnoha možností, které toto IDE poskytuje.



Obrázek 6 – Vývojové prostředí IntelliJ Idea

Mezi hlavní podporované jazyky patří Java, Kotlin, Groovy a Scala, ale skrze zásuvné moduly lze IDE rozšířit o podporu pro mnoho dalších populárních moderních jazyků, jako například Ruby, Python nebo TypeScript.

Zdroj [27].

3 Vaadin framework

Tato kapitola se zabývá Vaadin frameworkem, a to zejména jeho architekturou, výhodami, nevýhodami, nástroji, možnostmi využití, porovnáním s konkurencí, historií a komunitou, která se kolem frameworku utvořila. Většina informací uvedena v následujících podkapitolách vychází ze skvěle zpracované učebnice Book of Vaadin: Vaadin 14 Edition a obsáhlé internetové dokumentace tvůrců frameworku. Vaadin Docs, jak se dokumentace nazývá, slouží primárně k důkladnému popisu všech komponent, nástrojů a možností frameworku, ale zároveň i jako vzdělávací místo s několika užitečnými výukovými příklady a video návody, které neznalému uživateli komplexně představí práci s Vaadin a provedou ho prvními kroky na cestě k ovládnutí tohoto frameworku. Webová verze dokumentace disponuje oproti elektronické učebnici dalšími příklady kódu a technickými záležitostmi, které mohou programátorovi usnadnit práci při implementaci zamýšlených funkcionalit.

Jak učebnice, tak její internetová verze předpokládají alespoň základní zkušenost s programováním v jazyce Java. Znalosti týkající se vývoje uživatelského rozhraní a dalších frameworků často používaných při vývoji grafického uživatelského rozhraní v jazyce Java jako například AWT (Abstract Window Toolkit), Swing, SWT (Standard Widget Toolkit) mohou být čtenáři k dobru, ovšem nejsou předpokládány ani vyžadovány.

Vaadin nabízí kromě bezplatné verze i dvě komerční placená řešení Pro, Prime a Enterprise. Ta obsahují množství moderních a práci ulehčujících komponent a doplňků. Licence Pro je dostupná i pro studijní účely skrze platformu GitHub a jejich projekt GitHub Student Developer Pack, který provede ověření studentského emailu a poskytuje přístup k velkému množství placených licencí.

3.1 Obecně o Vaadin

„*Vaadin je v jádru open source framework sloužící zejména pro vývoj moderních webových aplikací v jazyce Java. Obsahuje velice rozsáhlou sadu komponent uživatelského rozhraní a nástrojů navržených tak, aby co nejvíce usnadnily a urychlily vývoj a následnou údržbu fungujících aplikací. Vaadin podporuje různé programovací modely. Lze využít buďto rozhraní Java API na straně serveru, HTML Web Components na straně klienta nebo kombinaci obou.*“ [28]

Web Components je sada různých technologií, které umožňují vývojářům vytvářet vlastní opakovaně použitelné HTML značky a komponenty, jejichž funkčnost je izolována a zapouzdřena mimo zbytek kódu. Tato sada poskytuje široké využití, neboť s ní vyvinuté komponenty následně fungují ve všech moderních internetových prohlížečích a také se dají využít v jakékoliv JavaScript knihovně nebo frameworku, který pracuje s jazykem HTML, což dokáže vývoj webové stránky či aplikace značně urychlit a usnadnit. Skládá se ze čtyř hlavních technologií, které společně slouží k tomu, že následně lze vytvořit univerzální vlastní prvky se zapouzdřenou funkčností a opakovaně je použít kdekoli bez obav z kolizí kódu.

- *Vlastní prvky* – jedná se o sadu API JavaScriptu, která je při vývoji moderních webových aplikací opravdu hodně důležitá. Umožňuje totiž definovat vlastní prvky (značky) HTML stránky a jejich chování, jež lze následně v uživatelském rozhraní používat dle potřeby a opakovaně.
- *Stínový DOM* – druhá důležitá sada API jazyka JavaScript, která slouží k připojení zapouzdřeného „stínového“ DOM (Document Object Model) stromu k prvku a ovládání s ním souvisejících funkcí. Důležitou součástí tohoto způsobu je fakt, že stínový DOM je vykreslován zvlášť stranou mimo hlavní DOM. Programátor tedy ocení zejména možnost zachování funkcí prvku privátních, takže je lze skriptovat a stylizovat, aniž by bylo potřeba řešit možné kolize s jinými částmi dokumentu.
- *HTML šablony* – jako třetí stavební kámen slouží prvky <template> a <slot>, které umožňují psát šablony značek, jež se ve vykreslené stránce nezobrazují. Ty pak lze opakovaně použít jako základ struktury vlastního prvku, což se projevuje jako obrovská úspora a zjednodušení kódu.
- *ES moduly* – závěrem se zde nachází systém modulů pro JavaScript, který se po téměř deseti letech standardizační práce dočkal oficiálního zveřejnění. Slouží zejména ke zrychlení v JavaScriptu psaného kódu a to tím, že umožňují lépe a systematicky uspořádávat proměnné a funkce. Myšlenkou je seskupovat právě takové funkce a proměnné, které jsou používány společně.

Tento koncept tak dopomáhá zejména k naplnění jedné ze základních zásad dobrého programování, a to neopakování stejného kódu. To v souvislosti s komplexnějšími HTML stránkami a s nimi spojenými styly a skripty nebylo v historii vždycky tak jednoduché.

„*Model HTML Web Components na klientské straně umožňuje používat knihovnu komponent uživatelského rozhraní Vaadin s jakýmkoli jiným webovým frameworkem kompatibilním s Web Components nebo dokonce v jakémkoli HTML dokumentu, aniž by vůbec bylo potřeba používat webový framework.*“ [29]

Zdroje [28], [29], [30], [31].

Model programování na straně serveru je v tomto případě jednou z velkých výhod a základních myšlenek celého Vaadin projektu. Umožňuje totiž programovat webová uživatelská rozhraní netradičně zcela v jazyce Java, nebo v jakémkoli jiném jazyce použitelném pro JVM (Java Virtual Machine). Tento princip tak funguje podobně jako v případě programování desktopové aplikace pomocí sad nástrojů, jako jsou AWT a Swing, ovšem o poznání jednodušeji, rychleji a soustředí se zejména na webové aplikace. S trochou nadsázky lze říct, že s použitím Vaadin dokáže webovou aplikaci s moderním uživatelským rozhraním vytvořit i programátor, který je zcela bez znalostí jazyků HTML a JavaScript.

Každá Vaadin komponenta disponuje API pro oba modely a při vývoji je možné tyto způsoby kombinovat. Komunikace klient-server je v tomto případě zcela automatizovaná a Vaadin se o ní stará sám.

Mezi hlavní výhody Vaadin patří zejména vysoká a komfortní rychlost vývoje, rozsáhlá a propracovaná sada grafických komponent, mechanismus datových vazeb, využití jazyka Java, integrace s dalšími frameworky a opravdu kvalitně zpracovaná dokumentace a aktivní komunita.

Jako nevýhody se uvádí například nutnost dynamického překladu do JavaScript souborů, zaměření na produktivitu před efektností a hodně špatná možnost škálovatelnosti. Vaadin není vhodný pro masivní veřejně přístupné webové stránky, ale spíše pro podnikové webové aplikace.

Zdroje [28], [32].

3.2 Historie a verze projektu

Když v roce 2000 Joonas Lehtinen zakládal tento projekt, tehdy ještě jako nadstavbu nad open source webovým frameworkem Millstone 3, jeho cílem bylo vytvoření technologie, která by výrazně zjednodušila proces vývoje interaktivních webových aplikací, přičemž hned od začátku čelil mnoha výzvám. Jako první komerční produkt byl tento adaptér uveden v roce 2006 a obsahoval komunikaci s klientem a vykreslovací jádro založené na AJAX (Asynchronous JavaScript and XML), což je technologie umožňující měnit obsah webové stránky bez nutnosti kompletního znovunačtení a vykreslování v ní se nacházejících komponent. Jednou ze zásadních změn se v roce 2007 stalo opuštění vlastní AJAX implementace pro vykreslování na straně klienta a místo toho byl integrován GWT (Google Web Toolkit) nad komponenty na straně serveru. Za zmínku stojí i fakt, že v té době do společnosti investoval hlavní autor původní verze MySQL Michael Widenius, čímž výrazně pomohl k následujícímu růstu firmy, a to nejen finančně, ale i pozvednutím určité míry důvěryhodnosti projektu, když s ním veřejně spojil své jméno. Další zlomový okamžik se odehrál 20. května roku 2009, kdy se projekt přejmenoval na Vaadin Framework a záhy i společnost změnila svůj název na Vaadin Ltd. O rok později spatřil světlo světa Vaadin Directory, webový kanál určený k distribuci doplňkových komponent a nástrojů vytvořených komunitou nebo oficiálním Vaadin Component Factory týmem. V současné době disponuje téměř 2000 komponenty pro různé verze frameworku a při vývoji složitějších systémů dokáže ušetřit spoustu času s implementováním vlastních komponent. V únoru 2017 byl představen a vydán Vaadin 8, jenž nově disponoval vylepšeným a přepracovaným API pro data binding využívající moderní prvky jazyka Java, jako například typové parametry a lambda výrazy. Vylepšení a zefektivnění se dočkalo také využívání paměti a výkonu CPU. Jako nejnovější LTS (Long Term Support) verze je v současnosti Vaadin 14 vydaný v srpnu 2019. Společnost se zavázala udržovat a podporovat tuto verzi na minimálně pět let ode dne vydání ve všech dostupných verzích předplatného. Předchozí LTS verze Vaadin 8 končí svoji podporu na začátku roku 2022 a společnost výrazně doporučuje migrovat aplikace běžící na této verzi na novější dlouhodobě podporovanou. V případě komerčních licencí Prime a Enterprise je jejich součástí i rozšířená dlouhodobá podpora pro předchozí i nynější LTS, a to až do roku 2032. Aktuálně nejnovější stabilní verze Vaadin 20 byla zveřejněna v létě roku 2021 a jedná se o mezikrok směrem k další LTS verzi.

Zdroje [33], [34].

3.3 Data binding

Ve většině aplikací vytvořených platformou Vaadin musí programátor pracovat s uživatelskými vstupy ve formě strukturovaných dat vkládaných do polí v rozsáhlých formulářích a zároveň tato data zpětně do formulářů zobrazit. Tato data jsou v kódu obvykle reprezentována jako instance Java objektu. Jako příklad si lze představit třeba uživatele aplikace, přičemž všechny jeho atributy (jméno, adresa, telefon, email a další) je třeba vytvořit, zobrazit a následně i editovat. Vaadin pro tyto případy poskytuje třídu Binder, která umožňuje definovat, jak jsou atributy třídy svázány s poli v uživatelském rozhraní.

Binder čte hodnoty v business objektu a automaticky je převádí z typu definovaného atributem v Java třídě do formátu očekávaného polem ve formuláři uživatelského rozhraní a naopak. Lze tedy například číselný datový typ nebo datum skrze Binder převést do textové komponenty TextField a zobrazit v různém formátu. Naopak lze tento vstupní formulář využít k naplnění dat do business objektu. Zároveň je možné validovat uživatelský vstup vůči předem stanovému vzoru a uživateli prezentovat stav validace několika různými způsoby.

Jako příklad lze uvažovat atribut *email* typu String, který je z uživatelského vstupu zadáván skrze Binder napojený na komponentu TextField. Takový Binder by měl logicky obsahovat i validaci na vzor emailu pomocí jednoduchého regulárního výrazu a v případě špatně vyplněného pole formuláře i výstižně nastavený parametr pro chybovou zprávu, který uživateli vizuálně zprostředkuje důležitou zpětnou vazbu. Následující ukázka zdrojového kódu tento příklad ukazuje z hlediska konkrétní implementace. Jakmile je jednou spojení mezi Binderem a příslušnou komponentou provázáno, stačí už vývojáři z Binderu pouze číst data konkrétního objektu, nebo je do něj zapisovat.

```
TextField emailField = new TextField();
Binder<User> userBinder = new Binder<>();

userBinder.forField(emailField)
        .asRequired("Required field.")
        .withValidator(new EmailValidator(
                "Please enter a valid email format."))
        .bind(User::getEmail, User::setEmail);

userBinder.readBean(user);
```

Zdrojový kód 2 – Ukázka data binding objektu User na formulář

Pro tento konkrétní příklad se dá k validaci využít třída EmailValidator, která kontroluje zadaný textový řetězec proti předem danému regulárnímu výrazu. Ten bohužel není stoprocentně kompatibilní s normou RFC 5322, ale zvládá validovat převážnou většinu platných emailových adres. Velkou předností je ovšem možnost vytvoření vlastního validátoru, ať už se jedná o validaci textu pomocí regulárního výrazu, či například porovnání dvou časových hodnot, z nichž první nesmí být větší než druhá. Validacím se v tomto ohledu meze nekladou a při vývoji uživatelského rozhraní dokáže tento nástroj ušetřit spoustu cenného času.

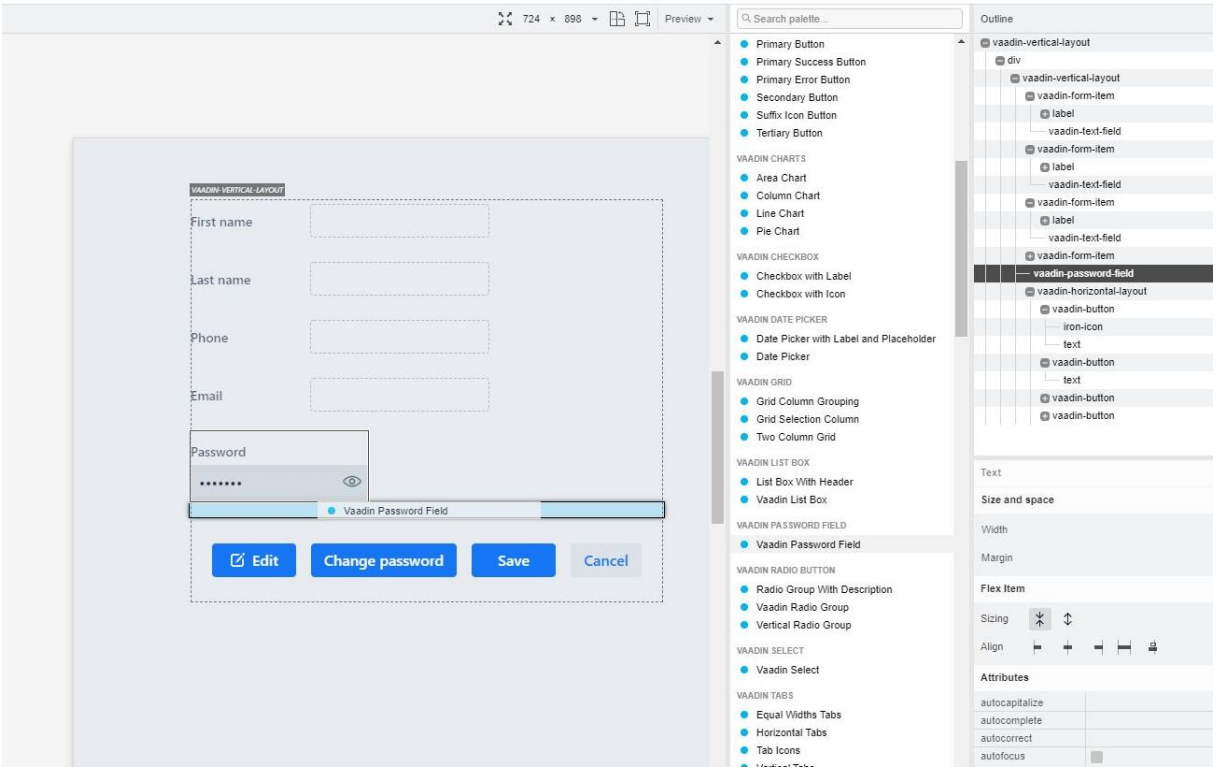
Zdroj [28].

3.4 Vaadin Designer

Oproti klasickým metodám vývoje UI pro webové aplikace disponuje Vaadin obrovskou výhodou v podobě vlastního vizuálního nástroje, který umožňuje rychle a efektivně tvořit rozsáhlé návrhy uživatelského rozhraní. V případě, že chce vývojář tvořit UI přímo v kódu a nemá k tomu potřebné předchozí zkušenosti, může se jednoduše zamotat do pozicování layoutů, komponent a následném stylování. Bez okamžité vizuální zpětné vazby se tento proces může časově protáhnout a potenciálního nezkušeného vývojáře vystresovat a odradit. Vaadin Designer poskytuje rychlý „drag and drop“ systém, možnost stylování a nastavení komponent na plátno pomocí jednoduchého klikání. Zároveň umožňuje okamžitý náhled na vytvářené UI, seznam použitelných layoutů, Vaadin komponent, HTML elementů a projektově specifických komponent, což při vývoji ušetří spoustu cenného času. Pro každé takto vytvořené uživatelské rozhraní nebo dílčí komponentu je zhotoven Vaadin 10+ Design TypeScript soubor obsahující HTML šablonu, na kterou se následně odkazuje ze souvisejícího Java souboru. Skrze anotace (odkazy) na jednotlivé komponenty či jiné části návrhu je možné vytvoření datové vazby mezi front-endem a back-endem.

Tento přístup jde ruku v ruce s principem oddělení zájmů, protože návrhový soubor s HTML šablonou definuje rozvržení a vizuální vzhled, zatímco doprovodný Java soubor obsahuje datovou logiku. V některých případech je ovšem tento princip příhodné porušit, protože určité atributy/chování komponent je snazší nastavovat skrze Java kód. Na druhou stranu není příliš vhodné zahlcovat back-endovou část velkým množstvím kódu věnujícímu se vizuální stránce věci. Do jaké míry je vhodné kombinovat tyto dvě možnosti záleží na rozhodnutí vývojáře a zejména na velikosti a povaze projektu.

Vaadin Designer (k vidění na obrázku 7) je součástí placených komerčních licencí Pro, Prime a Enterprise. V aktuální verzi je distribuován jako zásuvný modul pro dvě nejpopulárnější vývojová prostředí, Eclipse a IntelliJ IDEA.



Obrázek 7 – Vaadin Designer

Struktura a rozložení Vaadin Designer je rozdělena do tří hlavních částí. Uprostřed figuruje náhled na návrh konkrétní stránky, layoutu nebo komponenty. Na pravé straně se nachází seznam použitelných objektů, které lze pouhým přetažením myší vkládat do návrhu. Vedle seznamu komponent lze operovat se stromovou strukturou návrhu. Zároveň zde lze nastavovat a měnit některé atributy a styly konkrétní komponenty. Designer umožňuje i testovat náhled stránky na různých mobilních zařízeních a velikostech.

Zdroj [35].

4 Návrh a implementace

Tato kapitola se zabývá návrhem webové aplikace a následnou implementací konkrétního řešení. Proces návrhu je v rámci tvorby jakéhokoliv software jednou z nejdůležitějších částí projektu, jelikož se zde pokládají pomyslné základy celého produktu. Špatně navržený software může obsahovat kritické chyby, nedostatečnou kvalitu řešení, absenci funkcionalit požadovaných zákazníkem a mnoho dalších hypotetických problémů. Proto je zapotřebí věnovat přípravě projektu značné množství času a provést důkladnou rešerši vhodných technologií (kapitoly 2 a 3), možných způsobů implementace, nasazení, údržby, potenciálních uživatelů a zejména požadavků zadavatele projektu.

Zadání diplomové práce staví na základech konkrétního již existujícího a fungujícího ekosystému IoT zařízení, které však postrádá možnost jednoduché správy a dohledu pomocí webové aplikace. V současnosti je k přístupu k zařízením využívána mobilní aplikace, která však neposkytuje dostatečný komfort, požadovanou informační hodnotu a její funkce jsou značně omezeny. Z tohoto důvodu je třeba navrhnout a naimplementovat webovou aplikaci, která posune funkcionality zmiňované mobilní aplikace o level výše a poskytne uživateli příjemnější přístup k zařízením.

V rámci implementace technického řešení jsou v této kapitole představeny ukázky důležitých částí aplikace, a to jak z pohledu uživatelského rozhraní, tak z pohledu kódu.

Celá kapitola je protkána autorovými osobními pracovními zkušenostmi z firmy zabývající se vývojem rozsáhlých softwarových řešení pro energetické společnosti a znalostmi získanými při studiu na Univerzitě Pardubice.

4.1 Požadavky na systém

Na začátku návrhu software by měl proběhnout sběr požadavků na fungování systému, aby bylo zajištěno správné směřování projektu od samého počátku a byly přesně stanoveny cíle, kterých je třeba dosáhnout a jaké kritické body musí aplikace splňovat. Požadavky na systém se v reálném procesu vývoje software získávají hlavně ve spolupráci se zákazníkem a provedením vlastní analýzy. V rámci tohoto projektu diplomové práce se konkrétně jedná o vlastní analytický průzkum oficiálního zadání a komunikaci s vedoucím práce. Požadavek musí být konkrétně definovaný, proveditelný a testovatelný.

4.1.1 Funkční požadavky

Funkční požadavky určují, co musí být systém schopný dělat, jak reagovat, jaké akce provádět a jaké problémy by měl řešit. Při sběru takových požadavků by měly být pokládány různé analytické otázky, například:

- Proč je nutné systém vytvořit?
- K čemu má systém sloužit?
- Kdo a jak bude se systémem pracovat?
- Jaké budou vstupy a výstupy systému?
- Jaké funkce bude systém plnit?

Odpovědi na tyto analytické otázky by měly být informačně obsáhlé a následně by měly být detailně rozpracovány do jednotlivých konkrétních požadavků. Ve firmách soustředících se na vývoj softwaru se těmito úkoly zabývají speciálně proškolení analytičtí pracovníci, kteří následně úzce spolupracují s vývojovým týmem, zejména se softwarovými architekty.

Seznam požadavků by měl být strukturovaný a dobře organizovaný, například ve formě tabulky nebo stromové struktury. Přehled funkčních požadavků webové aplikace této práce je následující:

- FP 1 – Požadavky týkající se správy a dohledu senzorů.
    - FP 1.1 – Systém musí umožňovat přehled existujících senzorů.
        - FP 1.1.1 – Systém musí zajistit zobrazení pouze senzorů, kterých je uživatel vlastníkem, pokud nemá administrátorská práva.
        - FP 1.1.2 – Systém musí stav senzorů pravidelně aktualizovat.
    - FP 1.2 – Systém musí umožňovat editaci senzoru.
        - FP 1.2.1 – Systém musí zamezit editaci senzoru uživateli, který není jeho vlastníkem, nebo nemá administrátorská práva.
        - FP 1.2.2 – Systém musí umožňovat změnit vybrané atributy senzoru.
        - FP 1.2.3 – Systém musí umožňovat přeřadit senzor na jiný hardware.
        - FP 1.2.4 – Systém musí validovat změny.

        - FP 1.2.5 – Systém musí zamezit změnu určitých atributů pro uživatele bez administrátorských práv.
    - FP 1.3 – Systém musí umožňovat smazat senzor.
        - FP 1.3.1 – Systém musí smazání senzoru povolit pouze administrátorovi.
    - FP 1.4 – Systém musí umožňovat exportovat data senzoru za určité období.
- FP 2 – Požadavky týkající se správy a dohledu hardware zařízení.
    - FP 2.1 – Systém musí umožňovat přehled existujících zařízení.
        - FP 2.1.1 – Systém musí zajistit zobrazení pouze hardware zařízení, kterých je uživatel vlastníkem, pokud nemá administrátorská práva.
        - FP 2.1.2 – Systém musí stav hardware pravidelně aktualizovat.
    - FP 2.2 – Systém musí umožňovat editaci hardware zařízení.
        - FP 2.2.1 – Systém musí zamezit editaci hardware uživateli, který není jeho vlastníkem, nebo nemá administrátorská práva.
        - FP 2.2.2 – Systém musí umožňovat změnit vybrané atributy hardware.
    - FP 2.3 – Systém musí umožňovat smazat hardware.
        - FP 2.3.1 – Systém musí smazání hardware povolit pouze administrátorovi.
        - FP 2.3.2 – Systém musí zamezit smazání hardware na který je připojený nějaký senzor.
- FP 3 – Požadavky týkající se dohledového panelu.
    - FP 3.1 – Systém musí poskytovat dohledový panel pro každý senzor.
    - FP 3.1 – Systém musí dohledový panel pravidelně aktualizovat.
- FP 4 – Požadavky týkající se uživatelských účtů.
    - FP 4.1 – Systém musí autorizovat a autentizovat vstupujícího uživatele.
    - FP 4.2 – Systém musí umožňovat editaci uživatelského účtu.
    - FP 4.3 – Systém musí poskytovat správu uživatelských účtů pro administrátory.

        - FP 4.3.1 – Systém musí umožňovat vytvoření nového uživatele administrátorem.
        - FP 4.3.2 – Systém musí umožňovat smazání uživatele administrátorem.
        - FP 4.3.3 – Systém musí umožňovat možnost změny práv uživatele administrátorem.
    - FP 4.4 – Systém musí umožňovat změnu hesla bez nutnosti zásahu administrátora v případě zapomenutí přihlašovacích údajů.

4.1.2 Nefunkční požadavky

Druhým obecným typem požadavků jsou nefunkční požadavky. Kromě funkční náplně systému je také nutné definovat pravidla a omezení týkající se technologického řešení. Mezi nejčastější požadavky tohoto typu patří výkon, spolehlivost, rozšiřitelnost, údržba, škálovatelnost a bezpečnost. Při tvorbě těchto požadavků je třeba klást velký důraz na analýzu celkového systému, protože se jedná o výlučné požadavky a například velký důraz na optimalizaci výkonu musí být vykoupen budoucí možností rozšíření. V rámci tohoto projektu jsou nefunkční požadavky předem definovány zvolenými technologiemi v zadání práce. Přehled nejdůležitějších nefunkčních požadavků webové aplikace této práce je následující:

- NP 1 – Požadavky týkající se technického řešení.
    - NP 1.1 – Vývoj v jazyce Java.
        - NP 1.1.1 – Využití frameworku Spring.
        - NP 1.1.2 – Využití frameworku Vaadin.
    - NP 1.2 – Využití MySQL databáze.
- NP 2 – Požadavky týkající se produkce.
    - NP 1.1 – Nasazení na webový server Apache Tomcat.
    - NP 1.2 – Nasazení na cloudovou platformu Heroku.

4.1.3 Přehled a správa sensorů a hardware

Alfou a omegou navrhované webové aplikace je správa a dohled chytrých zařízení, konkrétně senzorů snímajících elektrickou a vodní spotřebu. Za tímto účelem je třeba vytvořit přehledný pohled (tabulku), kde bude možné senzory sledovat a přistupovat k detailu konkrétního senzoru a dohledového panelu. U každého senzoru by měly být vidět následující údaje: indikátor stavu, síla signálu, název senzoru, typ (elektrika/voda), informace pramenící z typu, informace, na kterém hardwaru a konkrétním pinu se senzor nachází. Každý senzor by mělo být možné rozkliknout a přejít k detailnímu přehledu všech atributů s možností editace. Pro každý senzor bude připravena také možnost vyexportovat a stáhnout agregovaná data za určité období. Musí být k dispozici i možnost vytvořit nový senzor a stávající senzor případně smazat. Jelikož jsou senzory spojené s hardware zařízeními, stejný přehled a správu je nutné vytvořit i pro hardware.

4.1.4 Správa uživatelů

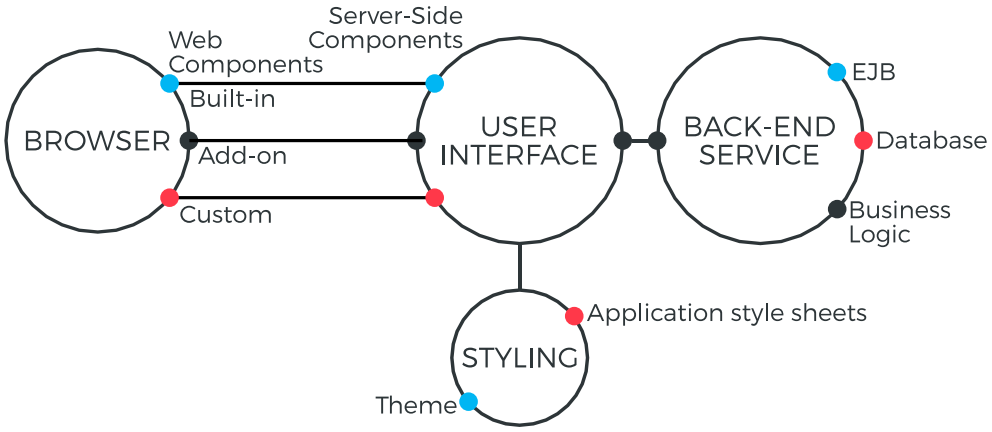
Aplikace by měla disponovat dvěma uživatelskými rolemi. Obyčejným uživatelem a uživatelem s administrátorskými právy. Uživatel by měl mít možnost změnit informace o vlastním účtu jako jméno, telefon nebo heslo. Administrátoři budou mít k dispozici přehled všech účtů v systémů s možností smazání a změny práv.

4.1.5 Dohledový panel

Další extrémně důležitou částí aplikace je dohledový panel, tzv. dashboard, pro každý senzor. Jelikož senzory produkují surová data, je třeba je smysluplně agregovat a prezentovat v podobě grafů a užitečných číselných hodnot. Každý dashboard by měl obsahovat informace o sensoru, hardwaru, na kterém je připojen, aktuální spotřebu, graf dnešní a měsíční spotřeby a velký sloupcový graf s možnostmi přepínáním mezi různými časovými obdobími a mezi spotřebou a cenou.

4.2 Návrh architektury a struktury

Pro webovou aplikaci není třeba vymýšlet žádnou složitou architekturu, ale lze využít některý ze zavedených a osvědčených postupů. Vzhledem k použitým technologiím je architektura navržena stejným způsobem, jako v základu všechny aplikace vytvářené Vaadin frameworkem v kombinaci se Spring. Tuto strukturu vystihuje obrázek 8 převzatý z dokumentace Vaadin.



Obrázek 8 – Architektura aplikace [5]

Uživatel k aplikaci přistupuje skrze prohlížeč, ve kterém jsou prezentovány webové komponenty. O tvorbu této části se Vaadin stará sám. Webové komponenty jsou svázány s UI komponentami aplikace na straně serveru. Na serveru spolu komunikují a vyměňují si data pohledy uživatelského rozhraní a backendové servisy připojené k MySQL databázi. Toto spojení obstará rozhraní Spring Data JPA (Java Persistence API) skrze technologii Hibernate.

4.3 Datový model

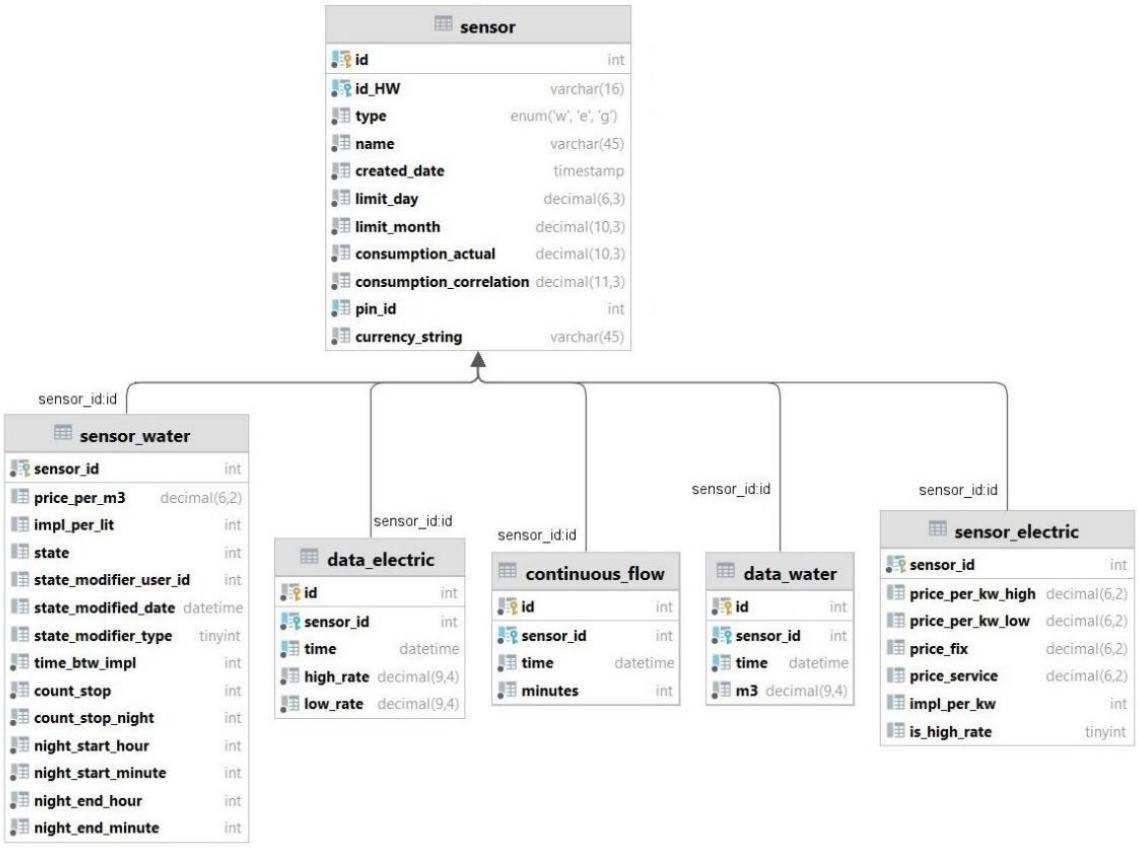
Návrh datového modelu je další disciplína, kterou není radno podceňovat a vyžaduje důkladnou analýzu celého projektu. Špatně navržený model dat povede k nutnosti přepracování během vývojového procesu, z čehož můžou vyplynout další nedostatky a problémy. Z podstaty zadání diplomové práce a již existujících senzorů produkujících data do databázových tabulek je tvorba nového datového modelu nežádoucí. Ze stávajícího datového modelu, který je poměrně rozsáhlý je nutné vybrat takové tabulky, se kterými bude potřeba pracovat v pozdější fázi implementace a která obsahují relevantní data pro potřeby aplikace a dokážou úspěšně saturovat všechny výše zmiňované požadavky.

4.3.1 Sensory

Nejdůležitějším prvkem aplikace jsou IoT senzory, které v datovém modelu reprezentuje hned několik tabulek viditelných na obrázku 9. Zaštiťující rodičovská tabulka *sensor* se společnými

[5] Obrázek převzatý ze zdroje [36].

atributy a dále dvě tabulky pro konkrétní typ senzoru a tabulky s daty jednotlivých měření. Všechny entity mají svoje unikátní identifikátory v podobě atributu *id*, případně *sensor_id* odkazující na *id* tabulky *sensor*. V tabulce *sensor* lze nalézt například cizí klíč *id_HW* ukazující na konkrétní záznam v tabulce hardware zařízení, ke kterému je senzor připojen. Důležitými atributy jsou *type* definující typ senzoru (w – voda, e – elektrika, g – plyn), *consumption_actual*, který udává aktuální průtok senzorem a denní a měsíční limity definovány atributy. Rozšiřující tabulky dle typu senzoru obsahují množství užitečných konfiguračních atributů týkajících se například výpočtu ceny nebo fungování senzoru. V případě vodního senzoru je významný atribut *state* určující aktuální stav ventilu (otevřený / zavřený).



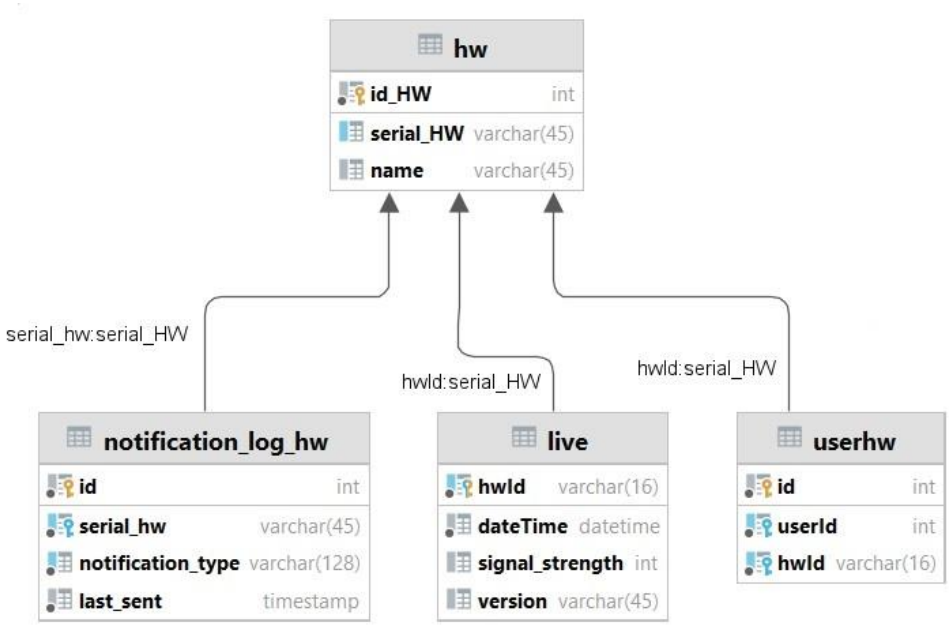
Obrázek 9 – Model definující senzor a jeho data

Ke každému typu senzoru je přiřazena i tabulka obsahující data měření v určitém čase, konkrétně se jedná o *data_electric* a *data_water*. Tato data jsou každou hodinu příslušným senzorem nahrána do databáze. U typu měřícího elektriku se data rozlišují na vysoký *high_rate* a nízký *low_rate* tarif.

Datový model je připraven pro budoucí možnost rozšíření například o plynové senzory, které v aktuálním provozu nejsou připojeny a ve vyvíjené verzi aplikace tak nejsou vyžadovány. Při návrhu je ovšem zvyklostí zohledňovat i možné či plánované budoucí rozšiřování.

4.3.2 Hardware

Jak již bylo několikrát zmiňováno, každý ze senzorů musí být připojen na konkrétní hardware zařízení. Proto je tento datový model taktéž velice důležitý. Všechny tabulky spojené s tímto modelem lze vidět na obrázku 10. Samotná tabulka *hw* obsahující informace o hardware zařízení je velice jednoduchá a obsahuje pouze tři atributy, název *name*, unikátní sériový kód zařízení *serial_hw*, který následně další tabulky využívají jako identifikátor a unikátní číselný identifikátor *id_hw* generovaný databází.



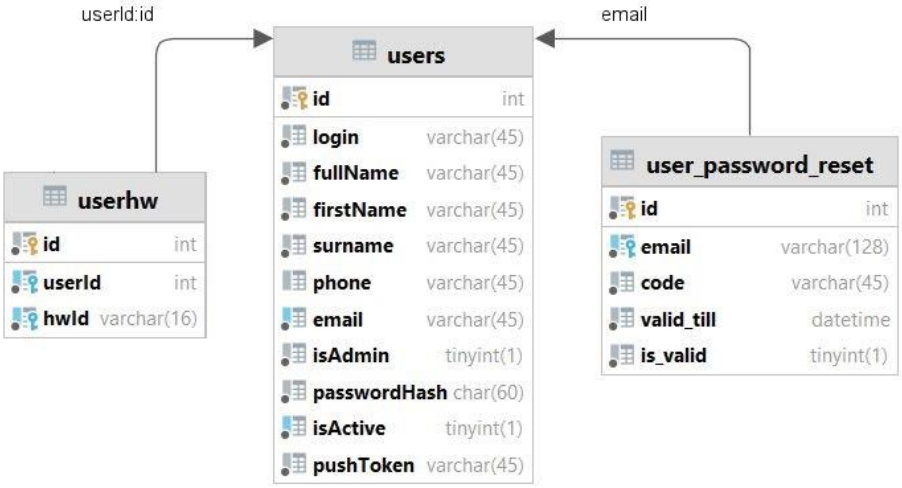
Obrázek 10 – Model hardware zařízení

Tabulka *userhw* slouží k propojení zařízení k jeho vlastníkům (tvůrcům). Tabulka *live* už je o něco zajímavější a obsahuje aktuální údaje o síle spojení IoT senzoru s připojenou sítí. Atribut *signal_strength* vyjadřuje tuto sílu spojení celým číslem od nuly do sta. Čas, kdy proběhlo aktuálně poslední měření tohoto záznamu, je uložen v atributu *dateTime*. Poslední tabulka týkající se hardware zařízení je *notification_log_hw*, ve které jsou uloženy informace o hlášení stavu zařízení *notification_type* (online/offline) s časovým údajem *last_sent*, kdy ke hlášení došlo.

4.3.3 Uživatel

Uživatelé, kteří s webovou aplikací pracují, jsou reprezentováni záznamem v tabulce *users*, jež je k vidění na obrázku 11. Zde se nachází všechny důležité atributy potřebné k autorizaci a autentizaci uživatele při přihlášení. Atribut *login* slouží jako přihlašovací jméno a atribut

passwordHash reprezentuje zakódované heslo pomocí hashovací funkce BCrypt. Lze zde také vidět konfiguraci práv uživatele, a to atributem *isAdmin*, který značí, zdali má uživatel administrátorská práva. Propojení s tabulkou *userhw* stejně jako v předchozím datovém modelu (kapitola 4.3.2) udává vlastnictví konkrétního zařízení určitými uživateli.



Obrázek 11 – Model uživatelských účtů

Tabulka *user_password_reset* slouží k možnosti obnovení zapomenutého hesla. Uživatel si může nechat vygenerovat unikátní kód, který je odeslán na zadanou emailovou adresu. V takovém případě se vytvoří záznam v tabulce, jako *code* se nastaví vygenerovaný kód, *valid_till* značí čas po který je kód platný a atribut *is_valid* udává, zdali kód ještě nebyl použit.

4.4 Uživatelské rozhraní

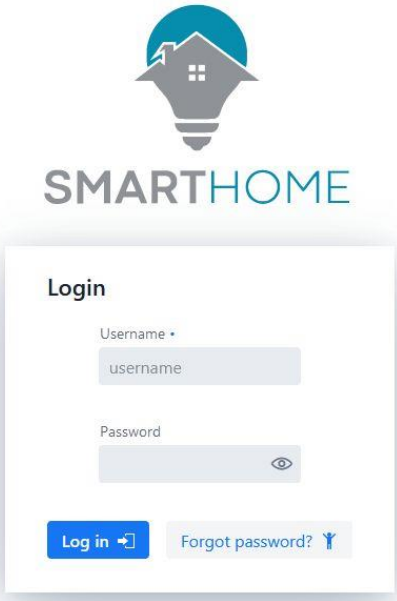
Návrh uživatelského rozhraní (dále jen UI) by měl vycházet z předem zpracovaných požadavků a ověřených návrhových vzorů. Musí poskytovat intuitivní ovládání webové aplikace a představuje všechny prvky, které uživatel vidí a se kterými může interagovat. Velkou pozornost je třeba věnovat také User Experience (dále jen UX), tedy uživatelskému zážitku. Jedná se o to, jak se uživatel při užívání aplikace cítí a jakou má výslednou zkušenost s jejím používáním. Například použití moderních grafických komponent frameworku Vaadin dokáže UX zásadně vylepšit v porovnání se zastaralejšími konkurenčními frameworky a službami.

Implementace webového uživatelského rozhraní by měla probíhat souběžně s vývojem backendové části aplikace. Funkčnost UI je také nutno důkladně testovat a konzultovat se zákazníkem, aby se předešlo konfliktu a nutnosti předělávat větší část aplikace. V době psaní

této kapitoly bylo již UI kompletně vytvořeno, proto jsou v následujících podkapitolách konkrétní ukázky z webové aplikace, namísto původních návrhů a náčrtů, které v momentě dokončení implementace pozbývají většího významu. Proces vývoje UI procházel několika etapami prototypování a v průběhu implementace se různě měnil podle výsledků testování funkčnosti a UX. Občas se totiž stane, že ne všechno, co se v návrhu jeví jako dobrý nápad se následně prokáže jako užitečná a lehce implementovatelná funkcionalita. Mezi návrhem a implementací je v takových případech nutné hledat adekvátní kompromisy, které následně odsouhlasí všechny zainteresované strany a dojde k všeobecné spokojenosti.

4.4.1 Přihlašovací obrazovka

Přihlašovací obrazovka je obecně první věc, která je uživateli prezentována při návštěvě webové aplikace. Jelikož není žádoucí vpustit do systému každého, kdo se do něj chce dostat, tato stránka slouží jako autorizační a autentizační vrátnice. Metaforický vrátný v podobě okna pro zadání přístupových údajů (k vidění na obrázku 12) vyhodnotí, zdali je uživatel ten, za koho se vydává a jestli ho smí vpustit do aplikace. Pokud ano, uživatele přesměruje na konkrétní stránku v systému.



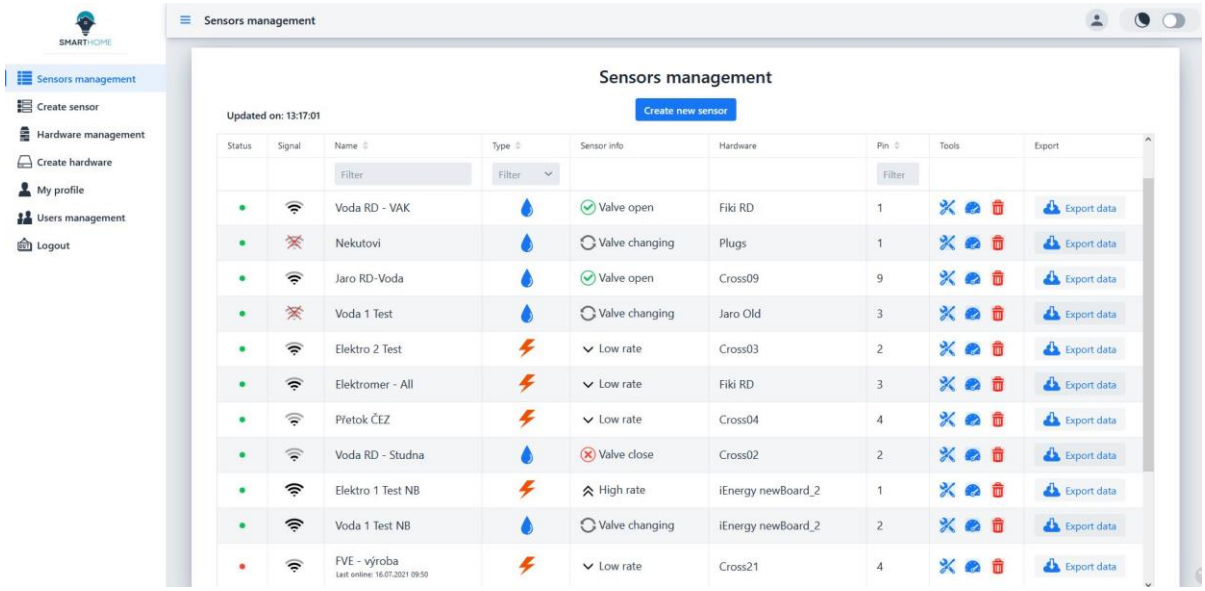
Obrázek 12 – Přihlašovací obrazovka

Přihlášení se skládá z loga aplikace a okna pro zadání přihlašovacích údajů, které je zvýrazněno černým stínem a vytváří příjemný efekt vznášejícího se formuláře. Dvě textové pole *username* a *password* slouží k zadání přihlašovacího jména a hesla. V případě hesla byla použita komponenta PasswordField, která ve výchozím stavu zobrazuje místo zadaného vstupu jen černé tečky z důvodu bezpečnosti. Tento stav lze změnit kliknutím na ikonu oka v pravé části textového pole, která zadaný text odhalí. Zvýrazněné tlačítko pro login provede ověření zadaných údajů a v případě špatně zadaného vstupu zobrazí chybovou hlášku. V opačném případě vpustí uživatele do aplikace. Pro případ, že by někdo zapomněl svoje přihlašovací údaje, je zde možnost obnovit své heslo pomocí tlačítka v pravé dolní části formuláře. Po kliknutí se objeví modální okno, ve kterém lze v několika jednoduchých krocích a vyplněním požadovaných informací vygenerovat email pro obnovu hesla.

4.4.2 Správa senzorů

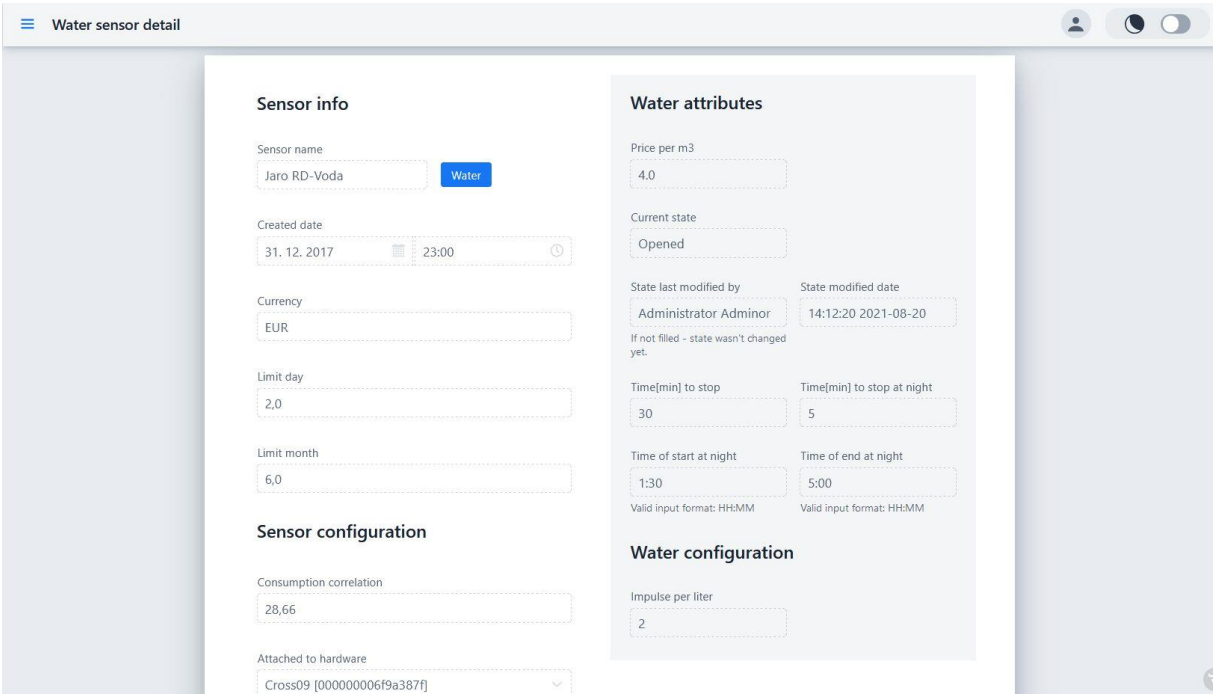
Nejdůležitější stránkou je bezpochyby správa a přehled senzorů. Této obrazovce nazvané „Sensors management“, která je k vidění na obrázku 13 níže, byla v rámci návrhu a implementace věnována největší pozornost. Každá stránka v aplikaci je potomkem hlavního layoutu, který v sobě obsahuje přehledné menu v levé části obrazovky. Tento seznam dostupných stránek je možné zabalit a obsah konkrétní stránky se roztáhne na celou šířku. Otevřená záložka je vždy barevně zvýrazněna. V horní části se nachází lišta s názvem aktuální stránky a v pravé části se nachází přepínač grafického zobrazení aplikace. K dispozici jsou dva motivy, světlý a tmavý. Barvy všech komponent a pohledů jsou optimalizovány pro oba motivy tak, aby zbytečně nedráždily oči a poskytovaly co nejlepší možné UX.

V horní části správy se nachází tlačítko pro vytvoření nového senzoru, které uživatele přesměruje na příslušnou stránku. Důležitým faktorem při návrhu bylo udržení přehlednosti a jednoduchosti v rámci množství informací, které je potřeba zde zobrazit. Dostupné senzory jsou zarovnány do tabulkové komponenty Grid. Řádky následně reprezentují konkrétní zařízení.



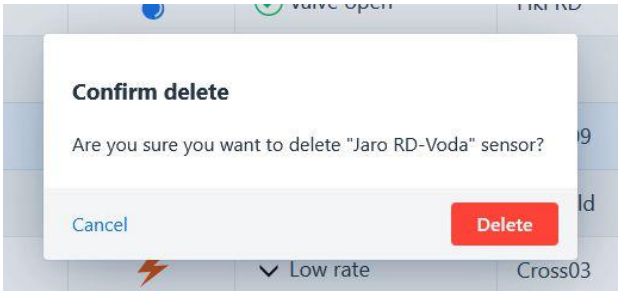
Obrázek 13 – Správa senzorů

V prvním sloupci tabulky je vyobrazen stav hardwaru, ke kterému je senzor připojen. Zelená znamená, že senzor je online, naopak červená signalizuje offline stav. V dalším sloupci je pomocí ikony prezentována síla signálu zařízení. Následuje název senzoru, podle kterého je možné záznamy v tabulce i filtrovat. Pokud je senzor offline, je zde zobrazen i časový údaj, kdy bylo zařízení naposledy online. Typ senzoru je v dalším sloupci prezentován všeříkající ikonou a je zde umožněno tabulku filtrovat stejně jako u názvu. Následující sloupec je ovlivněn typem a prezentuje dodatečné aktuální informace o senzoru. V případě vodního senzoru se jedná o stav uzávěru, který může být nastaven na otevřený a zavřený, případně stav, kdy se zrovna mění z jednoho na druhý. V případě elektrického senzoru je zobrazen tarif. Možnosti jsou zde dvě, a to vysoký a nízký. V sloupci Hardware se vyskytuje název zařízení, ke kterému je senzor připojen a následuje pin, na němž se nachází. Důležitý je sloupec pro možnosti správy senzorů, který je tvořen třemi ikonami. První ikona značí editaci a přesměruje uživatele do detailního pohledu na senzor, viditelný na následujícím obrázku.



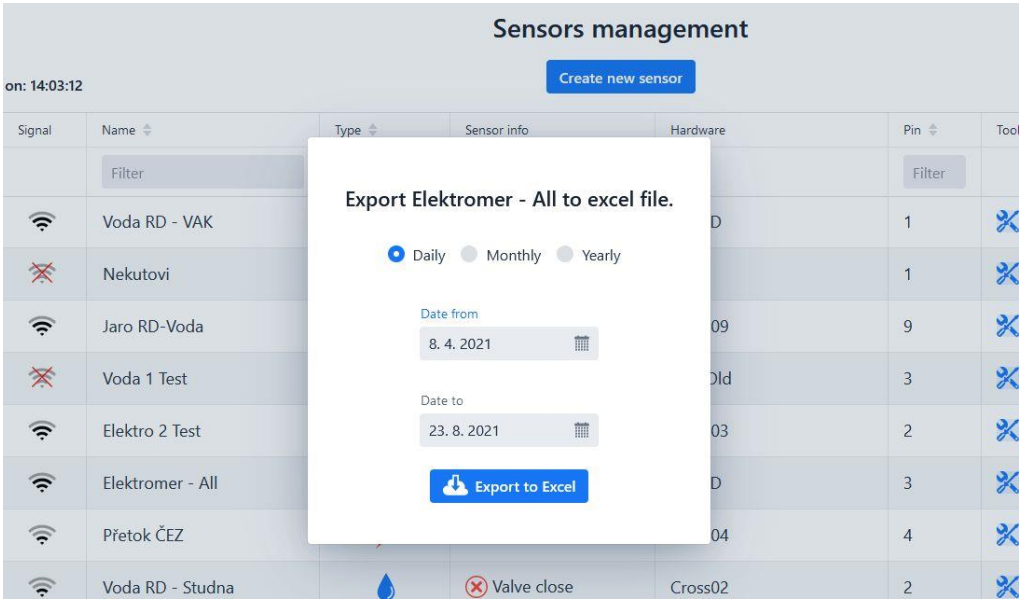
Obrázek 14 – Detail senzoru snímajícího vodní tok

Druhá ikona představuje dashboard a taktéž přesměruje uživatele na jinou stránku, v tomto případě na dohledový panel. Poslední možností, kterou sloupec poskytuje, je senzor smazat. Mechanismus odstranění je opatřen komponentou ConfirmDialog, k vidění na obrázku 15, aby nedošlo k nechtěnému smazání.



Obrázek 15 – Potvrzovací dialog mazání senzoru

Poslední sloupec disponuje možností exportovat data konkrétního senzoru do přehledné Excel tabulky a stáhnout soubor. Pro tyto potřeby je zobrazeno modální okno s potřebnými parametry pro následující agregaci dat a export, tak jak je vidět na obrázku 16.

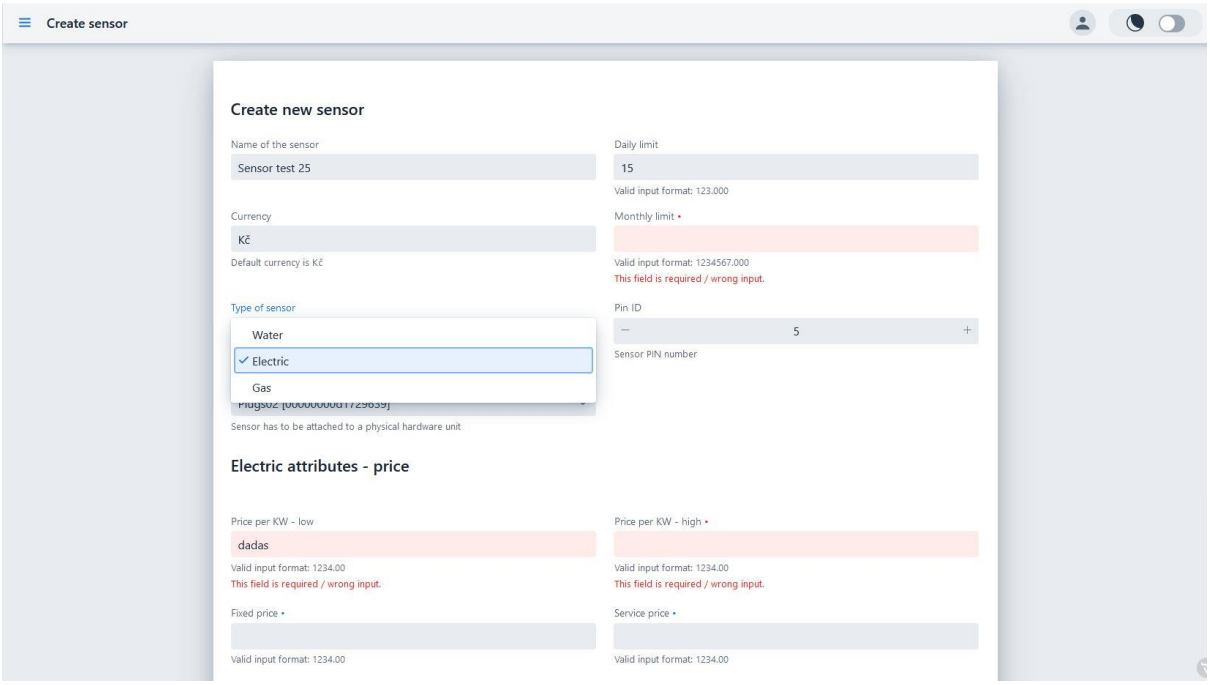


Obrázek 16 – Export dat do souboru

Tabulka je vybavena ještě pomocnou funkcionalitou, kdy se při kliknutí pravým tlačítkem myši na některý z řádků otevře kontextové menu se stejnými možnostmi, jaké nabízí sloupec *Tools*.

4.4.3 Vytváření nových senzorů

Samotná správa senzorů by byla neúplná, pokud by nebylo umožněno přidávat nové senzory. Tento požadavek splňuje stránka pojmenovaná „Create sensor“, která je k vidění na obrázku 17. Jak vyplývá z již ukázaného datového modelu senzoru (kapitola 4.3.1), při jeho vytváření je zapotřebí vyplnit značné množství informací, které se liší podle zvoleného typu. Většina těchto dat je ve formuláři zadávána skrze komponenty TextField, BigDecimalField a Select. Důležitým aspektem je výběr typu senzoru, podle kterého se v dolní části formuláře zobrazí buď atributy související s elektrickým senzorem, nebo atributy vodního senzoru. Všechny vstupy jsou validovány vzhledem k datovým typům uvedeným v databázovém modelu.

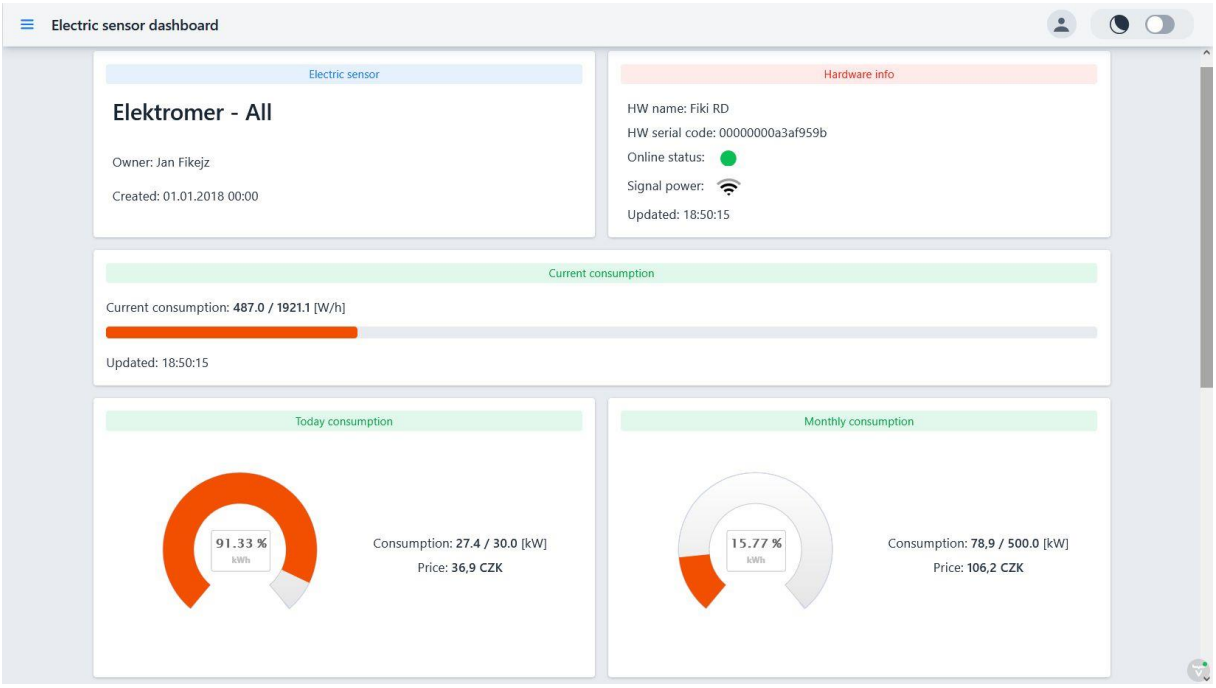


Obrázek 17 – Vytvoření nového senzoru

Významným prvkem je také napojení senzoru na existující hardware. V příslušné komponentě Select jsou zobrazena všechna dostupná zařízení, ke kterým má uživatel přístup. Účet s administrátorskými právy má přístup ke všem zařízením. Obyčejný uživatel pouze k těm, jichž je vlastníkem. Další validace spojená s připojením k hardware se týká pinu, na kterém se bude senzor nacházet. V případě, že je pin na zařízení již obsazen, objeví se modální okno s příslušnou hláškou a senzor není vytvořen. Pokud jsou všechna nutná formulářová pole správně vyplněna, po kliknutí na tlačítko *create* umístěné v dolní části stránky je senzor vytvořen, uložen do databáze a data jsou z komponent odstraněna, čímž se formulář uvede do původního prázdného stavu.

4.4.4 Dashboard

Další ze stránek, které si již od začátku projektu žádaly zvýšenou pozornost a musely být důkladně testovány, je dashboard konkrétního senzoru. Celkový pohled obsahuje několik panelů, z nichž každý zastává odlišnou funkci a poskytuje odlišnou informační hodnotu. V horní části na obrázku 18 se vlevo nachází panel s obecnými informacemi o senzoru. Vpravo nahoře je umístěn panel týkající se hardware zařízení, jehož informace jsou periodicky aktualizovány. Prostřední panel poskytuje vysokou informační hodnotu skrze komponentu ProgressBar, ve které graficky znázorňuje aktuální spotřebu naměřenou senzorem. Tento údaj je aktualizován každých 10 vteřin a při změně dojde k překreslení komponenty.

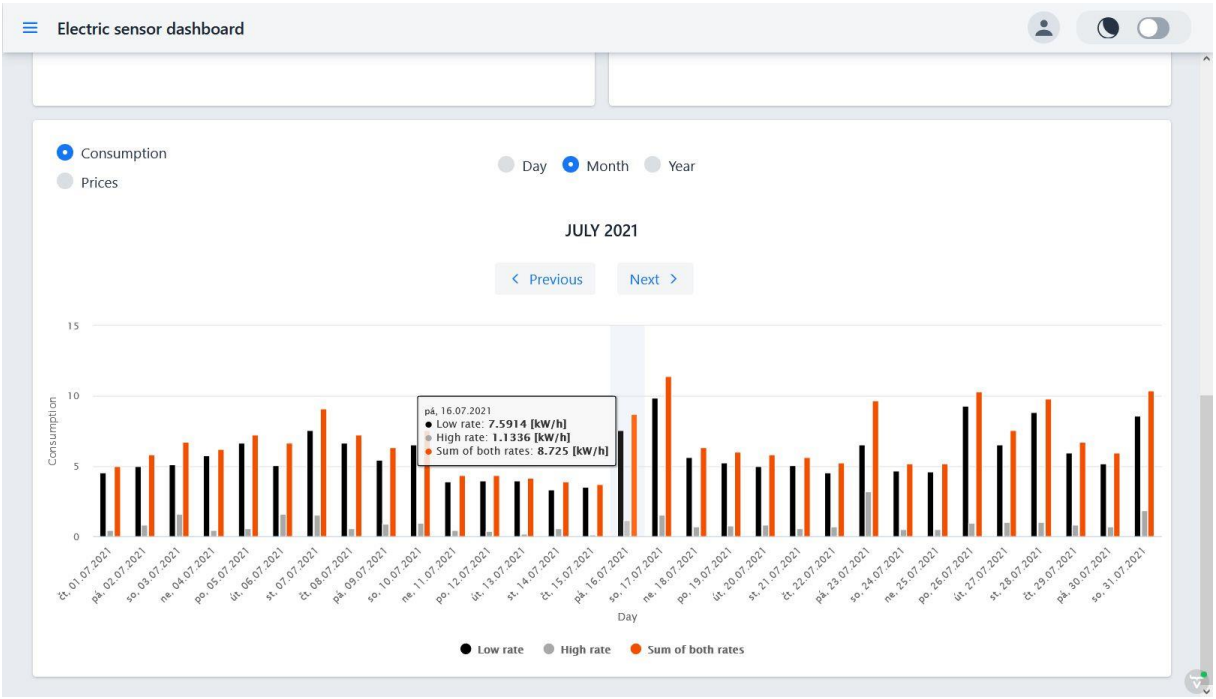


Obrázek 18 – Panely dashboardu v horní části stránky

V dolní části předchozího obrázku jsou vidět dva velké panely, jejichž obsah je vyplněn měřícími grafy ve tvaru podkovy a které se v celkovém náhledu stránky nachází uprostřed. Jedná se o agregovaná data týkající se denní a měsíční spotřeby. V grafu je spotřeba vyobrazena vzhledem k nastavenému dennímu či měsíčnímu limitu v procentech, proto jsou vedle k vidění i konkrétní číselné hodnoty. Jako bonus je zde zobrazena i vypočtená cena.

Vespod stránky se nachází sloupcový graf. Ten je odlišný pro různé typy senzorů, které se liší povahou veličiny. V praxi to znamená, že u senzoru snímajícího vodu není vysoký a nízký tarif, tudíž je zde v dashboardu pouze jeden sloupec, na rozdíl od tří sloupců u elektrického dashboardu. Součástí panelu je i několik ovládacích prvků. V levém horním rohu panelu je umístěn přepínač, který určuje, zda má být v grafu zobrazena spotřeba senzoru, nebo cena spotřeby. Nahoře uprostřed se nachází další přepínač, tentokrát definující časové období, pro které mají být data v grafu agregována. V případě zaškrtnutí *Day* se v pravém horním rohu objeví komponenta DatePicker, která umožňuje pohodlně vybrat konkrétní kalendářní den. Graf se následně změní na spotřebu nebo cenu za den, zobrazenou v sloupcích reprezentujících hodinu, v níž došlo k měření. Pokud je zaškrtnuta možnost *Month*, graf se zachová přesně jako na obrázku 19 a agreguje data jednotlivých dnů v měsíci do jedné hodnoty. V případě možnosti *Year* se graf chová obdobně, ale data agreguje do jednotlivých měsíců. Pod nadpisem grafu se uprostřed nachází dvě obslužná tlačítka, která umožní posouvat se na časové ose dopředu a dozadu. Na spodku grafu se nachází vysvětlující legenda. Zároveň umožňuje kliknutím myši

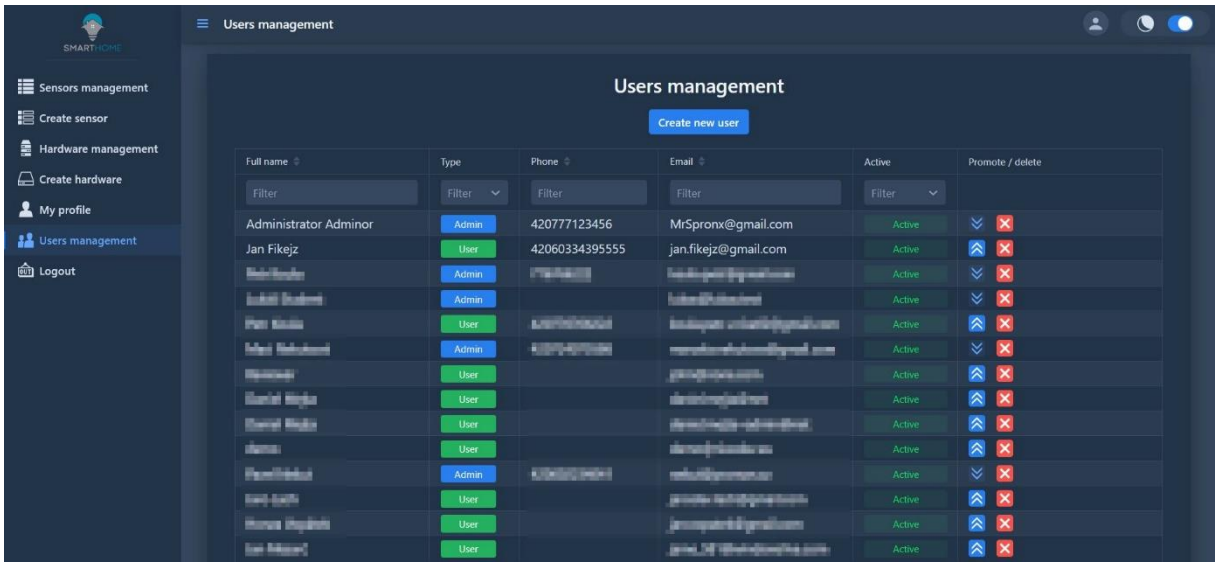
vypnout zobrazení některého z druhu sloupců. Všechny grafy použité v dashboardu jsou určitou variací komponenty Chart.



Obrázek 19 – Graf v dolní části stránky dashboardu

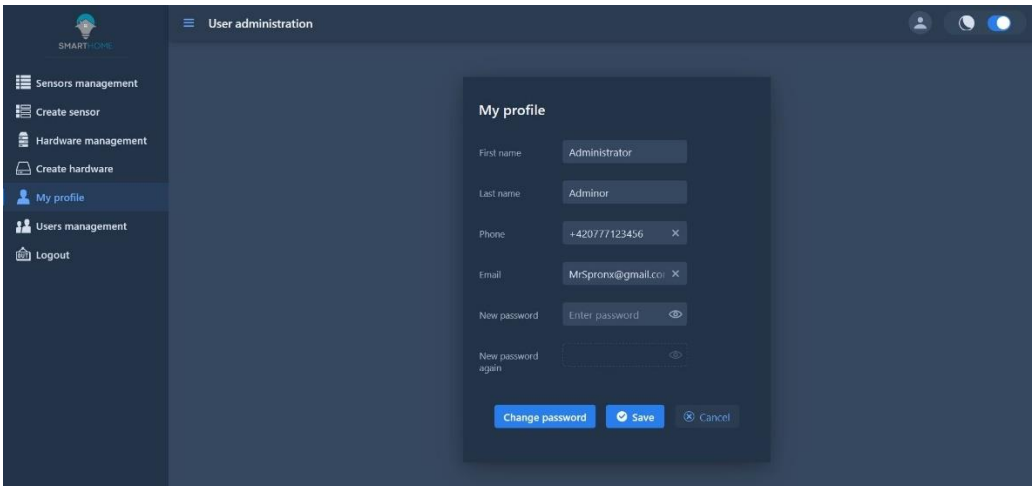
4.4.5 Správa uživatelů

Pro uživatele s administrátorskými právy je k dispozici i přehled a správa ostatních uživatelů. Stránka disponuje podobným návrhem jako přehled senzorů, a to zejména z důvodu zachování jednotného stylu stránek. „Users management“, jak se správa nazývá, je k vidění na obrázku 20, kde je zároveň aplikace přepnuta do tmavého režimu. V prvním sloupci tabulky uživatelů je uvedeno celé jméno. V dalším sloupci je pak vyobrazeno, jaká má uživatel práva. Zelená barva reprezentuje obyčejného uživatele a modrá administrátorský účet. Následuje nepovinný atribut telefonní číslo a hned poté povinný unikátní atribut emailové adresy. Pátý sloupec udává, zdali je uživatelský účet aktivní. Poslední část tabulky obsahuje možnost povýšit obyčejného uživatele do role administrátora nebo případně smazat jeho účet. Snížení práv z administrátora na obyčejného uživatele je povoleno pouze předem určenému správci aplikace. Stejně tak mazání administrátorských účtů. Komponenta pro filtrování poskytuje fulltextové vyhledávání, takže lze například zobrazit pouze uživatele, jejichž celé jméno obsahuje slovo „Petr“, nebo mají emailovou adresu s koncovkou „gmail.com“.



Obrázek 20 – Správa uživatelů [6]

Každý přihlášený uživatel má možnost spravovat svůj vlastní uživatelský profil. Tato stránka obsahuje prostý formulář s informacemi a možností editovat. Mód editace je vidět na obrázku 21. Za zmínku stojí možnost změny hesla, která se objeví po rozkliknutí tlačítka *edit*. Vstupy do formulářů jsou validovány podle stanovených pravidel. Například telefonní číslo a emailová adresa musí mít určitou formu. Heslo lze z bezpečnostních důvodů nastavit pouze takové, které je minimálně 8 znaků dlouhé a obsahuje nejméně jedno velké písmo a jeden speciální znak. Při změně emailu, stejně jako při vytváření uživatele, nelze použít email již používaný jiným uživatelem.



Obrázek 21 – Můj profil v režimu editace

[6] Obrázek byl pořízen z produkční verze aplikace, ve které se nachází kontaktní informace reálných uživatelů. Některé řádky jsou proto z bezpečnostních důvodů rozmazány.

4.5 Technické řešení implementace

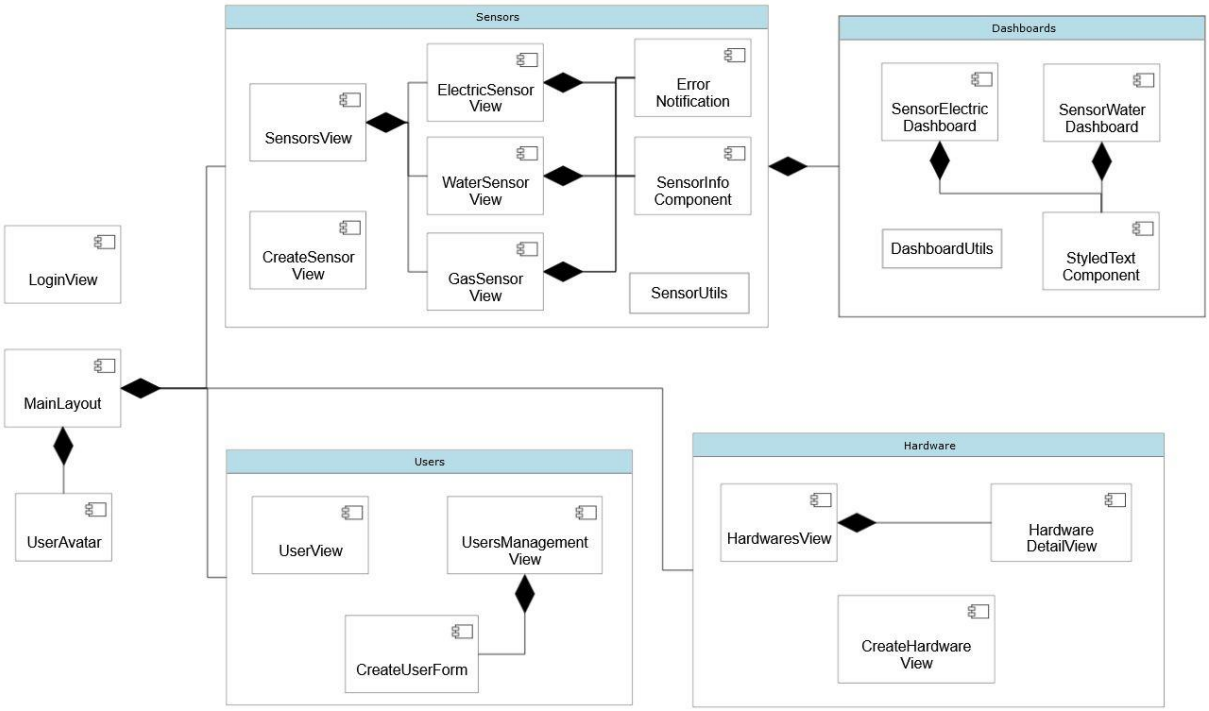
Implementace aplikace je bezpochyby nejdůležitější a technicky nejnáročnější částí vývoje softwaru z hlediska úspěšného dokončení produktu. Její součástí je i nutná příprava a seznámení se s technologiemi v případě, že vývojář zvolil některý z nástrojů, frameworků či programovacích jazyků, se kterými do té chvíle nepracoval. Kvalitně zpracovaný návrh dokáže proces implementace značně usnadnit. Vývojář by měl pracovat efektivně, využívat osvědčených postupů a nepokoušet se takzvaně vymýšlet kolo. Mnoho problémů, se kterými se setká, už pravděpodobně někdy někdo vyřešil a z těchto zkušeností je dobré čerpat i při vlastním řešení.

Tato kapitola předpokládá alespoň základní znalost jazyka Java, ve kterém byla aplikace vytvořena. Projekt se skládá z téměř 10 000 řádků kódu, které z 83 % tvoří Java, 12 % TypeScript a 5 % CSS (viz. Příloha B – Rozsah implementace projektu).

4.5.1 Diagram tříd

Strukturu implementace je zvykem ukazovat pomocí diagramu tříd, který představuje grafický návrh toho, jak jsou na sobě určité třídy závislé a jak spolupracují. Vzhledem k velikosti projektu by bylo neefektivní zobrazit všechny třídy. Obecně by také měly být pro každou třídu zobrazeny všechny její metody a atributy. Datové entity není z podstaty věci důležité v tomto diagramu zahrnovat, jelikož by měly jedna ku jedné odpovídat datovému modelu představenému v kapitole 4.3. Servisní třídy, které se starají o komunikaci s databází a přenos a přípravu dat by tento diagram také zbytečně zaplňovaly, jelikož je jich opravdu velké množství. Pro každou entitu je připravena servisní třída, která implementuje vlastní rozhraní. Takový diagram by na čtenáře mohl působit jako přeplněný a jelikož tato diplomová práce neslouží jako absolutní dokumentace aplikace, budou v následujícím diagramu vyobrazeny pouze důležité třídy komponent, které tvoří jádro aplikace.

V diagramu tříd, který je vyobrazen obrázkem 22, lze vidět rozdělení do čtyř logických celků. Mimo tato uskupení vyčnívá zaštiťující komponenta *MainLayout*, která odkazuje na všechny ostatní třídy krom jedné. Je tomu tak z prostého důvodu. Jedná se totiž o třídu představující layout, tedy rozložení, aplikace. Obsahuje v sobě rolovací menu na pravé straně a horní panel, které jsou součástí všech dalších stránek. Vazbu mezi touto třídou a všemi ostatními zajišťuje anotace @ParentLayout, která je součástí Vaadin Flow.



Obrázek 22 – Diagram komponentních tříd a jejich nástrojů

Stránka *LoginView* je z logických důvodů mimo ekosystém všech ostatních komponent, protože představuje přihlašovací obrazovku, která by měla být od systému určitým způsobem odtržená. Není žádoucí, aby uživatel, který ještě úspěšně nepodstoupil proces autorizace, viděl, jak aplikace vypadá a jaké poskytuje možnosti v menu. Jednotlivé skupiny stránek odpovídají logickým celkům plynoucím z návrhu aplikace. Všechny třídy obsahují různé množství dalších komponent Vaadin frameworku a některé využívají modální okna, která jsou v určitých případech natolik jednoduchá, že se je nevyplatilo vytvářet jako samostatné formulářové třídy. Jedná se například o potvrzující dialogy nebo modální okna pro zadání určité hodnoty.

4.5.2 Ukázky stěžejních funkcí a technických řešení

Mezi nejdůležitější třídy projektu patří bezpochyby *SensorsView*, která poskytuje ucelený přehled senzorů. V následující ukázce zdrojového kódu 3 je vyobrazena metoda *createGridComponent*, ve které se nastavují různé atributy této komponenty. Skrze Vaadin Designer, ve kterém byly stránky vytvářeny, se anotací @Id("grid") automaticky utvořilo spojení s konkrétní instancí objektu Grid ve frontendové části. Nejdůležitější je obsah tabulky, tedy seznam senzorů, který zajišťuje metoda *getSensors* a následně se vloží do objektu Grid. Tato metoda vyhodnotí práva přihlášeného uživatele a ve spolupráci se servisní třídou načte z databáze potřebné informace o senzorech. Objem přenášených dat je žádoucí co nejvíce

redukovat, proto třída *SensorGridRepresentation* obsahuje pouze ty atributy, které jsou následně zobrazeny v tabulce.

```
@Id("grid")
private Grid<SensorGridRepresentation> grid;

private void createGridComponent() {
    grid.setSelectionMode(SelectionMode.SINGLE);
    grid.addThemeVariants(GridVariant.LUMO_ROW_STRIPES);
    grid.setHeight("100%");

    List<SensorGridRepresentation> sensors = getSensors();
    grid.addItemClickListener(event -> {
        grid.select(event.getItem());
    });

    gridListDataView = grid.setItems(sensors);
}
```

Zdrojový kód 3 – Nastavení komponenty Grid

Ve zdrojovém kódu 3 je zobrazen způsob, kterým se vytváří obsah jednotlivých sloupců tabulky. Komponenta Grid disponuje stěžejními metodami *addColumn* a *addComponentColumn*, které se liší očekávaným obsahem sloupce. Pokud se očekává atribut třídy *SensorGridRepresentation*, lze použít *addColumn* a parametrem navázat konkrétní atribut. V případě, že by obsahem sloupce měla být komponenta, použije se druhá metoda. V ukázce 4 se jedná o sloupec představující sílu signálu hardware zařízení. K tomuto zobrazení je využit obrázek. Pomocná třída *SensorsUtil* a její metoda *setSignalImage* zvolí, podle číselného údaje získaného z databáze, jaký konkrétní obrázek by měl být zobrazen a následně ho vloží do obalující komponenty Div. Tento proces je opakován pro každý senzor, který představuje řádek v tabulce. Sloupci se dále nastaví další parametry, jako je titulek, šířka a možnost automatického roztáhnutí.

```
private void createSignalColumn() {
    signalColumn = grid.addComponentColumn(sensor -> {
        Integer signalStrenght = hardwareLiveService
                .getSignalStrenght(sensor.getIdHw());
        Div span = new Div();
        span.setWidthFull();
        span.addClassName("signalImage");
        Image image = new Image();
        SensorsUtil.setSignalImage(signalStrenght, span, image);
        return span;
    }).setHeader("Signal").setWidth("90px").setFlexGrow(0);
}
```

Zdrojový kód 4 – Vytvoření sloupce tabulky

Data v tabulce senzorů jsou pravidelně aktualizována a stará se o to metoda *refreshGrid*. Pomocí anotace @Scheduled (k vidění v ukázce zdrojového kódu 5), kterou Spring framework poskytuje, se dá na straně serveru nastavit periodické opakování jakékoliv metody s návratovým typem void. Jelikož se jedná o funkcionalitu prováděnou na serveru, je potřeba na změny dat v komponentách upozornit prohlížeč. Ve Vaadin se to řeší způsobem předvedeným na následující ukázce zdrojového kódu. Nejdříve se zjistí možnost spojení s UI a následně se metodou *push* pošlou změněná data přímo prohlížeči. Nemusí se tak čekat, až si o data z nějakého důvodu zažádá prohlížeč, který tak ve většině případech činí až na podnět uživatele.

```
@Scheduled(fixedDelay = 60000)
public void refreshGrid() {
    try {
        getUI().ifPresent(ui -> ui.access(() -> {
            grid.setItems(getSensors());
            gridListDataView.refreshAll();
            lastUpdateText.setText("Updated on: " +
                    LocalDateTime.now().format(DateTimeFormatter
                            .ofPattern("HH:mm:ss")));
            ui.push();
        }));
    } catch (UIDetachedException e) {
       ...
    }
}
```

Zdrojový kód 5 – Automatická aktualizace dat v tabulce senzorů

Technologicky zajímavý je způsob exportu dat do souboru. Spring ani Vaadin neposkytují řešení tohoto problému, proto je třeba využít specializovanou knihovnu. Apache POI je ideální nástroj, který poskytuje API pro čtení a zápis souborů ve formátech Microsoft Office, jako jsou Word nebo Excel. Jeho využití je předvedeno na následující ukázce zdrojového kódu 6. Vaadin umožňuje stáhnout dynamicky generovaná data uložená v objektu třídy StreamResource. Slouží k tomu komponenta Anchor, která funguje jako odkaz. Pokud se do ní následně vloží komponenta Button, zafunguje jako tlačítko ke stažení dat. Třída XSSFWorkbook knihovny Apache POI slouží k vytvoření tabulkového souboru. Do něj je následně vložen list s názvem podle senzoru, jehož data se generují. Vlastní privátní metody *getHeaderRow* a *getSecondHeaderRow* následně v listu vytvoří první dva řádky. Na prvním řádku je v několika sloučených buňkách uveden nadpis a na druhém jsou nadpisy konkrétních sloupců.

```
StreamResource streamResource = new StreamResource(
        „Export-„ + sensor.getName() + „.xlsx“, () -> {
    XSSFWorkbook workbook = new XSSFWorkbook();

    XSSFSheet sheet = workbook
            .createSheet(„Data of „ + sensor.getName());

    getHeaderRow(sensor, sheet, workbook);
    getSecondHeaderRow(sensor, sheet, workbook,
            pickerFrom.getValue(), pickerTo.getValue());

    createWaterExcelWorkBook(sensor, sheet,
radioButtonGroup.getValue(),
            pickerFrom.getValue(), pickerTo.getValue());
    return getByteArrayInputStream(workbook);
});
download = new Anchor(streamResource, „“);
download.add(exportBtn);
```

Zdrojový kód 6 – Vytvoření Excel dokumentu

Důležitá je metoda *createWaterExcelWorkbook* v ukázce zdrojového kódu 7, která následující řádky v listu naplní konkrétními daty. Ta jsou seřazena od nejnovější po nejstarší. V ukázce se jedná o data z vodních senzorů sloučená do měsíců ve vybraném časovém úseku.

```
int rowNum = 2;
for (LocalDate month : dataMonthlyMap.keySet()
        .stream().sorted().collect(Collectors.toList())) {
    XSSFRow row = sheet.createRow(rowNum++);
    Cell cellDate = row.createCell(0);
    cellDate.setCellValue(month.getMonth().name()
            .toLowerCase(Locale.ROOT) + "|" + month.getYear());
    Cell cellValue = row.createCell(1);
    cellValue.setCellValue(
            MathUtils.round(dataMonthlyMap.get(month), 3));
}
```

Zdrojový kód 7 – Naplnění buněk tabulky

Jak již bylo v rámci této práce zdůrazňováno v několika kapitolách, důležitou součástí aplikace jsou dohledové panely jednotlivých senzorů. Stěžejní komponentou, která je v dashboardech využita hned několikrát, je Vaadin Chart. Na ukázce zdrojového kódu 8 je vidět tvorba takového grafu a nastavení důležitých atributů. Jelikož jsou v implementaci tato nastavení poměrně rozsáhlá a jsou rozdělena do několika různých metod a v kódu se nachází na odlišných místech, pro větší přehled jsou zde v ukázce uspořádána pospolu tak, aby čtenáři dávala smysl. Nejdůležitějším prvkem je určení typu, který se zadává v konstruktoru a určuje následnou

grafickou podobu grafu. Dále se s grafem pracuje zejména skrze objekt třídy Configuration, která umožňuje specifikaci mnoha parametrů.

```
private Chart mainChart = new Chart(ChartType.COLUMN);
Configuration configuration = mainChart.getConfiguration();

XAxis x = new Xaxis();
x.setTitle(„Hour“);
configuration.addxAxis(x);

Yaxis y = new Yaxis();
y.setMin(0);
y.setTitle(„Consumption“);
configuration.addyAxis(y);

configuration.getChart().setBackgroundColor(
        new SolidColor(255,255,255,0));
.
.
.
ListSeries series = getColoredListSeries(
            new ArrayList<>(values),
            "m3",
            new SolidColor(Colors.BLUE.getRgb()));
configuration.setSeries(series);
```

Zdrojový kód 8 – Tvorba grafu

Do grafu se vloží obě osy s příslušným popisem a pro y osu se nastaví minimální hodnota. Pozadí je nastaveno na průhlednou barvu, protože Vaadin nedisponuje tmavým barevným tématem pro komponenty Chart. Jelikož je tento mód v současné době poměrně populární a v aplikaci dostupný, byl v rámci implementace tímto způsobem vytvořen určitý kompromis. Data se do grafu vloží jako objekt třídy ListSeries. Metoda *getColoredListSeries* vrací instanci tohoto objektu vytvořeného ze zadaných parametrů. V tomto případě se jedná o data měření průtoku vody uchovaná v proměnné *values*. Sloupce grafu jsou zbarveny do modra a datová řada je pojmenována „m3“. V ukázce je vynecháno získávání dat a jejich agregace.

Často využívaným nástrojem je komponenta Dialog v modálním režimu, která poskytuje libovolně naplněné okno, po jehož otevření nelze vykonávat jiné operace, dokud uživatel okno nezavře. Ideální příklad lze nalézt ve třídě *UserView*. Jak je vidět na ukázce zdrojového kódu 9, dialogové okno zde plní roli upozornění na určitý nežádoucí stav. Pokud si chce uživatel změnit heslo, musí do obou polí typu PasswordField zadat stejný text. Když tak neučiní, vytvoří se komponenta typu Dialog s nastavením jako modální a odpovídající hláškou o chybně zadaném vstupu. Po otevření okna lze zavřít jednoduchým kliknutím myši mimo okno.

```
String pw1 = passwordField1.getValue();
String pw2 = passwordField2.getValue();
if (!pw1.equals(pw2)) {
    Dialog dialog = new Dialog();
    dialog.setModal(true);
    dialog.add("New passwords do not match. Try again.");
    dialog.open();
} else {
    user.setPasswordHash(passwordEncoder.encode(pw1));
}
```

Zdrojový kód 9 – Vytvoření dialogového okna

Ze třídy *UserView* vychází i další ukázka zdrojového kódu 10. Celá aplikace je protkána značným množstvím různých druhů formulářů. Většinou se jedná o komponenty TextField. Pro ně nabízí Vaadin speciální možnost, jak propojit atribut konkrétního objektu s UI komponentou. Jedná se o Binder představený v kapitole 3.3. V následujícím zdrojovém kódu jsou atributy *name* a *phone* napojeny na TextFieldy *firstNameField* a *phoneField*. U prvního příkladu je textové pole nastavené jako požadované. U druhého dochází k validaci pomocí regulárního výrazu. Zavoláním metody *readBean* objektu *userBinder* dojde k načtení dat z objektu *user* zadaného v parametru do připojených textových polí.

```
userBinder.forField(firstNameField)
        .asRequired("Required field.")
        .bind(User::getFirstName, User::setFirstName);

userBinder.forField(phoneField)
        .withValidator(s -> s.matches(PatternStringUtils.phoneRegex),
                PatternStringUtils.phoneErrorMessage)
        .bind(User::getPhone, User::setPhone);

userBinder.readBean(user);
```

Zdrojový kód 10 – Ukázka data binding

Stejně tak lze opačným způsobem metodou *writeBean* načíst data z textových polí do atributů objektu. To je využito například u editace a následném ukládání změněného objektu do databáze.

Dalším zajímavým příkladem implementace je zajištění možnosti odesílání emailových zpráv. To je využíváno, pokud uživatel zapomene vlastní heslo a potřebuje ho bezpečně změnit. Využit je balíček mail, který je součástí Spring frameworku. Servisní třída *EmailServiceImpl*, implementující rozhraní *EmailService*, poskytuje metodu, která se o odeslání emailu postará

způsobem, jenž je vidět ve zdrojovém kódu 11. Jako parametr je nutné uvést cílovou adresu, předmět a text zprávy. O všechno se následně postará instance třídy JavaMailSender.

```
public void sendSimpleMessage(String to, String subject, String
text) {
    SimpleMailMessage message = new SimpleMailMessage();
    message.setFrom("smart.home.upce@gmail.com");
    message.setTo(to);
    message.setSubject(subject);
    message.setText(text);
    emailSender.send(message);
}
```

Zdrojový kód 11 – Ukázka odeslání emailu

Tato funkcionalita by se neobešla bez předešlé konfigurace zobrazené v ukázce 12. V rámci konfiguračního souboru *application.properties* je nutné uvést následující parametry, které zajistí využití účtu Gmail jako prostředníka pro odesílání emailových zpráv. Z bezpečnostních důvodů je přihlašovací jméno i heslo zobrazeno jako skryté.

```
#EMAIL CONFIGURATIONS
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=sm*******ce@gmail.com
spring.mail.password=***********

# Other properties
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.connectiontimeout=5000
spring.mail.properties.mail.smtp.timeout=5000
spring.mail.properties.mail.smtp.writetimeout=5000
```

Zdrojový kód 12 – Nastavení použití Gmail

5 Nasazení aplikace

Tato kapitola se zabývá závěrečným tématem praktické části této diplomové práce a tím je nasazení projektu na běžící prostředí. Aplikaci nelze spouštět pokaždé skrze vývojové prostředí, a i kdyby ano, v takovém případě by stejně nebyla přístupná pro další uživatele, což je u webové aplikace naprosto nezbytné. Vývojář se tedy po dokončení vývoje musí poprat ještě se závěrečnou výzvou, pokud nebereme v potaz údržbu a podporu aplikace, kterou je nasazení na připravené prostředí. Musí tak zajistit například běžící databázi, případně další služby potřebné k provozu a zvolit si způsob, jakým aplikaci nasadí a kam.

5.1 Způsoby zabalení aplikace

Aplikace se na server nedá nahrát ve stejné formě, v jaké je vytvářená skrze projekt ve vývojovém prostředí, ale musí být sestavena do speciálního výsledného souboru. Ten se vytváří způsobem, kdy se všechny soubory, které tvoří aplikaci Java, zkomprimují a zabalí do jediného souboru. Jazyk Java používá příponu *.ear* pro podnikové aplikace založené na Java EE, *.war* pro webové aplikace a *.jar* pro samostatné aplikace a knihovny.

- Soubor JAR je soubor s třídami jazyka Java, přidruženými metadaty a zdroji, jako je text a obrázky, sdruženými do jednoho souboru.
- Soubor WAR je soubor, který se používá k distribuci kolekce souborů JAR, Servlet, souborů XML, statických webových stránek, jako je HTML, a dalších zdrojů, které tvoří webovou aplikaci.
- Soubor EAR je standardní soubor JAR, který představuje moduly aplikace, a adresář s metadaty zvaný META-INT, který obsahuje jeden nebo více deskriptorů nasazení. Umožňuje tak současné nasazení různých modulů na aplikační server.

Zdroj [40].

5.2 Nasazení na webový server

Po dokončení vývoje je aplikace připravena k nasazení na webový server. Vzhledem k povaze diplomové práce a snaze vyhnout se placeným licencím produktů byl jako vhodný server v tomto případě zvolen Apache Tomcat. Jedná se o bezplatný webový server, na který je možné nasadit soubory typu WAR. Za pomoci balíčkovacího nástroje Maven a správně nastavené konfigurace souboru *pom.xml* lze pomocí jednoduchého příkazu `mvn clean package` zabalit

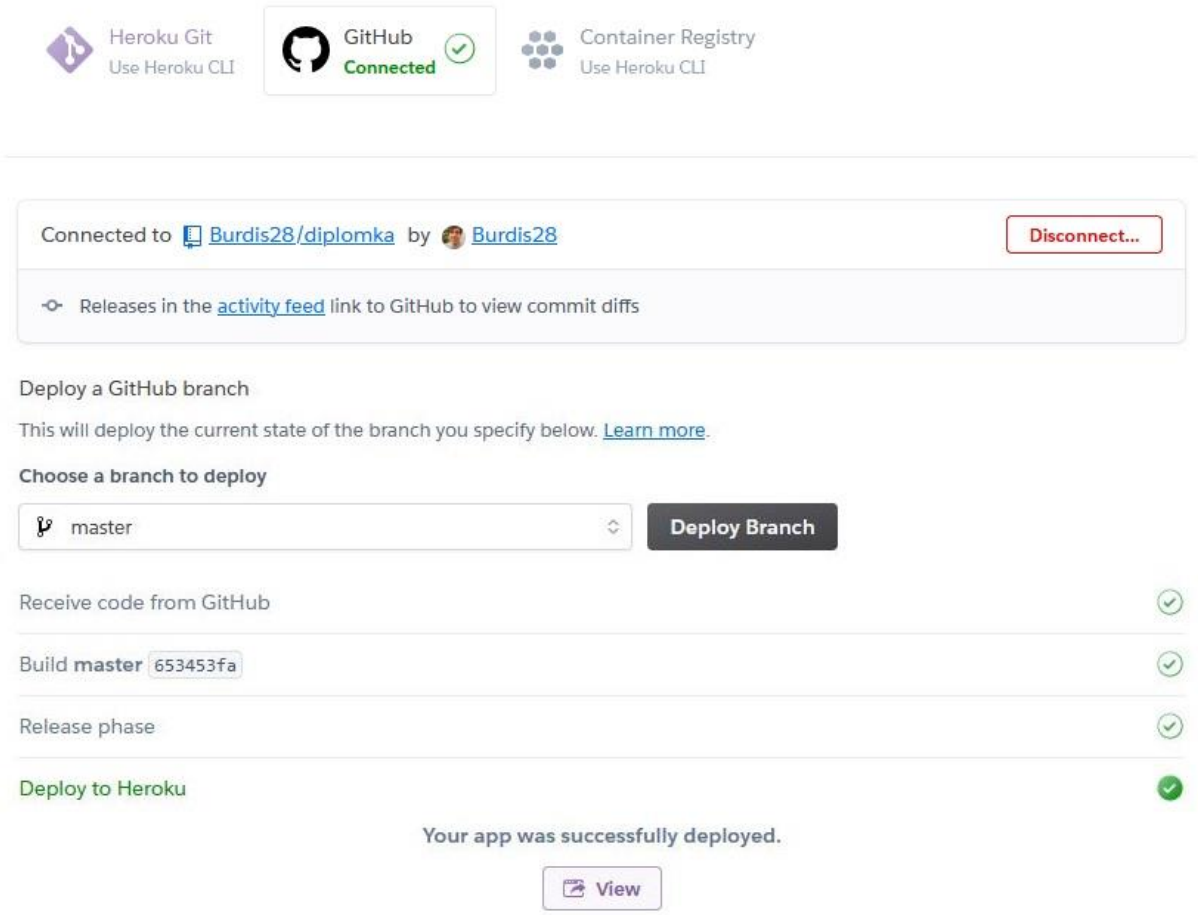
celý projekt a vytvořit tak soubor připravený pro nasazení na webový server. Samotný proces nasazení je velice jednoduchý. Stačí zkopírovat připravený WAR soubor do *$CATALINA_HOME/webapps* kde proměnná *$CATALINA_HOME* ukazuje na složku, v níž je server nainstalován. Spuštěním serveru se zároveň spustí i webová aplikace.

Zdroj [39].

5.3 Nasazení na platformu Heroku

V případě, že vývojář nechce nebo nemá možnost provozovat webový server ve vlastní režii, může využít možností některé z platforem orientovaných na cloud computing. Konkrétně se pro případ webové aplikace hodí model PaaS (Platform as a Service), který poskytuje možnost rychlého vytváření řešení na míru. Uživatel se zde nemusí starat o infrastrukturu, konkrétně správu serveru, operačního systému ani dalšího software. Jediné, co řešit musí, je aplikace a její data. Jednou z prvních služeb tohoto typu a průkopníkem v oblasti cloudových platforem je Heroku. Poskytuje podporu pro všechny technologie použité při vývoji webové aplikace v rámci této diplomové práce, a navíc disponuje možností nasazení aplikace i v bezplatném režimu. Samozřejmě se jedná o výkonnostně poměrně omezenou možnost, ale pokud se vývojář rozhodne, že potřebuje větší výkon, bonusové služby a databázové místo navíc, lze si jednoduše koupit jiný plán a bezproblémově tak aplikaci škálovat.

Platforma se ovládá skrze vlastní Heroku CLI (Command Line Interface) a umožňuje nasadit připravený a zabalený soubor webové aplikace. Druhou možností je integrace skrze GitHub, kde lze nastavit automatické nasazení na určitou vývojovou větev (k vidění na obrázku 21). Třetí možností je nahrání a nasazení připraveného Docker kontejneru. Samotné nasazení je poměrně jednoduchý proces, který lze provést pomocí několika málo příkazů.



Obrázek 23 – Ukázka nasazení aplikace na Heroku pomocí GitHub

Kromě nasazení vlastních aplikací poskytuje Heroku i značné množství doplňků, které dokážou aplikaci rozšířit a usnadnit její provoz. Od mnoha různých typů datových úložišť, přes monitoring a analýzu, až po automatické testování a zabezpečení. V rámci testování procesu nasazení webové aplikace (před uvedením do produkce) byl využit doplněk ClearDB MySQL.

Zdroj [38].

Závěr

Cílem této diplomové práce bylo navrhnout a implementovat webovou aplikaci pro správu a dohled chytrých zařízení. Vytvořený produkt bylo následně nutné nasadit na webový server Apache Tomcat a do cloud prostředí skrze platformu Heroku.

Internet věcí je tématem této diplomové práce. Jedná se o převratnou technologii, kterou bylo nutné důkladně představit a popsat moderní trendy tohoto odvětví. Jelikož se jedná o opravdu rozsáhlý obor, byly zde vybrány pouze ty nejdůležitější trendy. V rámci této části práce byl kladen důraz na představení reálných příkladů a konkrétních IoT aplikací.

K implementaci webové aplikace byla vybrána technologie Vaadin, která umožňuje komplexní tvorbu webových aplikací v jazyce Java a je dobře provázána s frameworkem Spring. Před samotným návrhem a implementací bylo proto zapotřebí se s těmito technologiemi seznámit. Z toho důvodu práce rozebírá i použití těchto technologií, jejich koncepci, přínosy, výhody a nevýhody. Popsány jsou i další použité nástroje a systém řízení báze dat MySQL.

Práce dále představuje přehledný návrh nové webové aplikace, která uživateli poskytuje možnost svá chytrá zařízení spravovat a získávat důležité provozní a statistické informace pomocí takzvaných dashboardů. Návrh slouží zároveň jako určitá forma business dokumentace projektu, proto jsou zde popsány funkční i nefunkční požadavky na systém, datové modely, architektura a uživatelské rozhraní.

Po fázi návrhu následovala implementace finálního řešení. V rámci této kapitoly byly představeny stěžejní funkcionality z hlediska technického zpracování. Důraz je zde kladen na ukázku použití Vaadin frameworku a komponent, které poskytuje. Proces implementace doprovázela řada problémů. Řešení se však vždy dalo dohledat v oficiální dokumentaci Vaadin Docs.

Poslední částí práce bylo nasazení vytvořené aplikace na dvě různá prostředí. Zdokumentován byl proces nasazení na webový server Apache Tomcat a cloud platformu Heroku.

Aplikace je navržena tak, aby ji bylo možné v budoucnu jednoduše rozšiřovat, například o nové druhy chytrých zařízení nebo další grafy v rámci dohledových panelů.

Finálním výsledkem diplomové práce je webová aplikace sloužící ke správě a dohledu IoT zařízení. Aplikace je zcela funkční a implementuje všechny požadavky uvedené v návrhu. Cíle této diplomové práce tak byly splněny v plném rozsahu.

Použitá literatura

[1] EVANS, Dave. *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*. San Jose: Cisco, 2011. Dostupné také z: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[2] STEWARD, Jack. The Ultimate List of Internet of Things Statistics for 2021. *Findstack* [online]. London, 2021 [cit. 2021-8-19]. Dostupné z: https://findstack.com/internet-of-things-statistics/

[3] POHANKA, Pavel. Internet věcí. *Data Distribution Service - Pavel Pohanka* [online]. Brno, 2020 [cit. 2021-8-19]. Dostupné z: https://pavelpohanka.cz/internet-of-things/

[4] 5G and IoT in 2021. *Thales - building a future we can all trust* [online]. Paříž: Thales Group, 2021 [cit. 2021-8-19]. Dostupné z: https://www.thalesgroup.com/en/markets/digital-identity-and-security/iot/resources/innovation-technology/5G-iot

[5] DIGITEUM TEAM, ed. Smart Homes: Guide to Home Automation Using IoT (Internet of Things). *Digiteum* [online]. Gdansk, 17.4.2021 [cit. 2021-8-19]. Dostupné z: https://www.digiteum.com/iot-smart-home-automation/

[6] AMADEO, Ron. Google commits to supporting Nest smart home devices for 5 years. *Ars Technica* [online]. 30.6.2021 [cit. 2021-8-19]. Dostupné z: https://arstechnica.com/gadgets/2021/06/google-commits-to-supporting-nest-smart-home-devices-for-5-years/

[7] BURBACH, Björn. What are the IoT Trends in 2021? *Siemens Advanta* [online]. 14.1.2021 [cit. 2021-8-19]. Dostupné z: https://www.siemens-advanta.com/blog/what-are-expected-iot-trends-2021

[8] SCHWARTZ, David. IoT Applications in Healthcare. *Wireless Watchdogs* [online]. Los Angeles [cit. 2021-8-19]. Dostupné z: https://www.wirelesswatchdogs.com/blog/iot-applications-in-healthcare

[9] Build trust in your IoT data with blockchain. *IBM* [online]. Armonk, New York [cit. 2021-8-19]. Dostupné z: https://www.ibm.com/topics/blockchain-iot

[10] Building trust in IoT devices with powerful IoT security solutions. *Thales - building a future we can all trust* [online]. Paříž: Thales Group, 2021 [cit. 2021-8-19]. Dostupné z: https://www.thalesgroup.com/en/markets/digital-identity-and-security/iot/iot-security

[11] Chytré město. *Wikipedie: otevřená encyklopedie* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 6. 8. 2021 [cit. 2021-8-19]. Dostupné z: https://cs.wikipedia.org/wiki/Chytré_město

[12] Chytrý odvoz odpadu. *Smart Prague - inovace pro lepší život v praze* [online]. Praha: Smart Prague, 2019 [cit. 2021-8-19]. Dostupné z: https://smartprague.eu/projekty/chytry-svoz-odpadu

[13] MARR, Bernard. What Is The Artificial Intelligence Of Things? When AI Meets IoT. *Forbes* [online]. New Jersey: Integrated Whale Media Investments, 2019, 20.12.2019 [cit. 2021-8-19]. Dostupné z: https://www.forbes.com/sites/bernardmarr/2019/12/20/what-is-the-artificial-intelligence-of-things-when-ai-meets-iot/?sh=73e199abb1fd

[14] Spring Framework Documentation. *Spring* [online]. Palo Alto: Spring, 2021, 14.7.2021 [cit. 2021-8-19]. Dostupné z: https://docs.spring.io/spring-framework/docs/5.3.9/reference/pdf/index.pdf

[15] Spring Tutorial. *The Biggest Online Tutorials Library* [online]. Haidarábád, India, 2016 [cit. 2021-8-19]. Dostupné z: https://www.tutorialspoint.com/spring/index.htm

[16] VAGHANI, Romin, PAVITHRAN, Anand, ed. Introduction to Spring Framework. *GeeksforGeeks | A computer science portal for geeks* [online]. Noida, 5.12.2019 [cit. 2021-8-19]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-spring-framework/

[17] VERMEER, Brian. Spring dominates the Java ecosystem with 60% using it for their main applications. *Snyk* [online]. Spencers Wood (Reading): Snyk Limited, 5.2.2020 [cit. 2021-8-19]. Dostupné z: https://snyk.io/blog/spring-dominates-the-java-ecosystem-with-60-using-it-for-their-main-applications/

[18] IBM CLOUD EDUCATION. Java Spring Boot. *IBM* [online]. Armonk, New York, 25.3.2020 [cit. 2021-8-20]. Dostupné z: https://www.ibm.com/cloud/learn/java-spring-boot

[19] Spring Boot Reference Documentation. *Spring* [online]. Palo Alto: Spring, 19.8.2021 [cit. 2021-8-20]. Dostupné z: https://docs.spring.io/spring-boot/docs/2.5.4/reference/htmlsingle/

[20] What is MySQL? Everything You Need to Know. *Talend - A Cloud Data Integration Leader* [online]. Redwood City (Kalifornie), 12.10.2020 [cit. 2021-8-20]. Dostupné z: https://www.talend.com/resources/what-is-mysql/

[21] MySQL 8.0 Reference Manual. *MySQL :: Developer Zone* [online]. Austin (Texas): Oracle Corporation [cit. 2021-8-25]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

[22] MySQL Customers. *MySQL* [online]. Austin (Texas): Oracle Corporation [cit. 2021-8-20]. Dostupné z: https://www.mysql.com/customers/

[23] Klient-server. *Wikipedie: otevřená encyklopedie* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 1.6.2021 [cit. 2021-8-20]. Dostupné z: https://cs.wikipedia.org/wiki/Klient-server

[24] IBM CLOUD EDUCATION. Docker: What is docker? *IBM* [online]. Armonk, New York, 23.6.2021 [cit. 2021-8-20]. Dostupné z: https://www.ibm.com/cloud/learn/docker

[25] Docker overview. *Docker Documentation* [online]. Palo Alto: Docker [cit. 2021-8-20]. Dostupné z: https://docs.docker.com/get-started/overview/

[26] What is a Container? | App Conteinerization | Docker. *Docker: Empowering App Development for Developers* [online]. Palo Alto: Docker [cit. 2021-8-20]. Dostupné z: https://www.docker.com/resources/what-container

[27] IntelliJ IDEA overview. *JetBrains: Essential tools for software developers and teams* [online]. Praha: JetBrains, 16.7.2021 [cit. 2021-8-20]. Dostupné z: https://www.jetbrains.com/help/idea/discover-intellij-idea.html

[28] VAADIN TEAM. *Book of Vaadin: Vaadin 14 edition*. Turku (Finsko): Vaadin, 2019. ISBN 978-1692121440.

[29] Introduction: What are web components? *Webcomponents* [online]. [cit. 2021-8-20]. Dostupné z: https://www.webcomponents.org/introduction

[30] Web Components: Concepts and usage. *MDN Web Docs* [online]. San Francisco: Mozilla Foundation, 29.6.2021 [cit. 2021-8-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Web_Components

[31] CLARK, Lin. ES modules: A cartoon deep-dive. *Mozilla Hacks* [online]. San Francisco: Mozilla Foundation, 28.3.2018 [cit. 2021-8-20]. Dostupné z: https://hacks.mozilla.org/2018/03/es-modules-a-cartoon-deep-dive/

[32] Vaadin - Overview. *The Biggest Online Tutorials Library* [online]. Haidarábád, India, 2016 [cit. 2021-8-20]. Dostupné z: https://www.tutorialspoint.com/vaadin/vaadin_overview.htm

[33] Vaadin. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2021-8-20]. Dostupné z: https://en.wikipedia.org/wiki/Vaadin

[34] *Vaadin - the company behind the framework* [online]. Turku (Finsko): Vaadin [cit. 2021-8-20]. Dostupné z: https://vaadin.com/company

[35] Vaadin Designer: An intuitive visual Vaadin UI builder for Eclipse & IntelliJ IDEA. *Vaadin* [online]. Turku (Finsko): Vaadin, 21.4.2020 [cit. 2021-8-20]. Prezentace ve formátu PDF. Dostupné z: https://v.vaadin.com/hubfs/Pdfs/Vaadin-designer-factsheet.pdf

[36] Introduction. *Vaadin 14 Docs* [online]. Turku (Finsko): Vaadin, 3.6.2021 [cit. 2021-8-20]. Dostupné z: https://vaadin.com/docs/v14/guide/introduction

[37] wesome24. Smart Home Logo. *Vecteezy* [online]. Bowling Green (Kentucky): Eezy [cit. 2021-8-20]. Dostupné z: https://www.vecteezy.com/vector-art/2292564-smart-home-logo-in-white-background

[38] SUKOINEN, Mikael. Deploying a Java web app to the Heroku cloud. *Vaadin* [online]. Turku (Finsko): Vaadin, 2020 [cit. 2021-8-20]. Dostupné z: https://vaadin.com/learn/tutorials/cloud-deployment/heroku

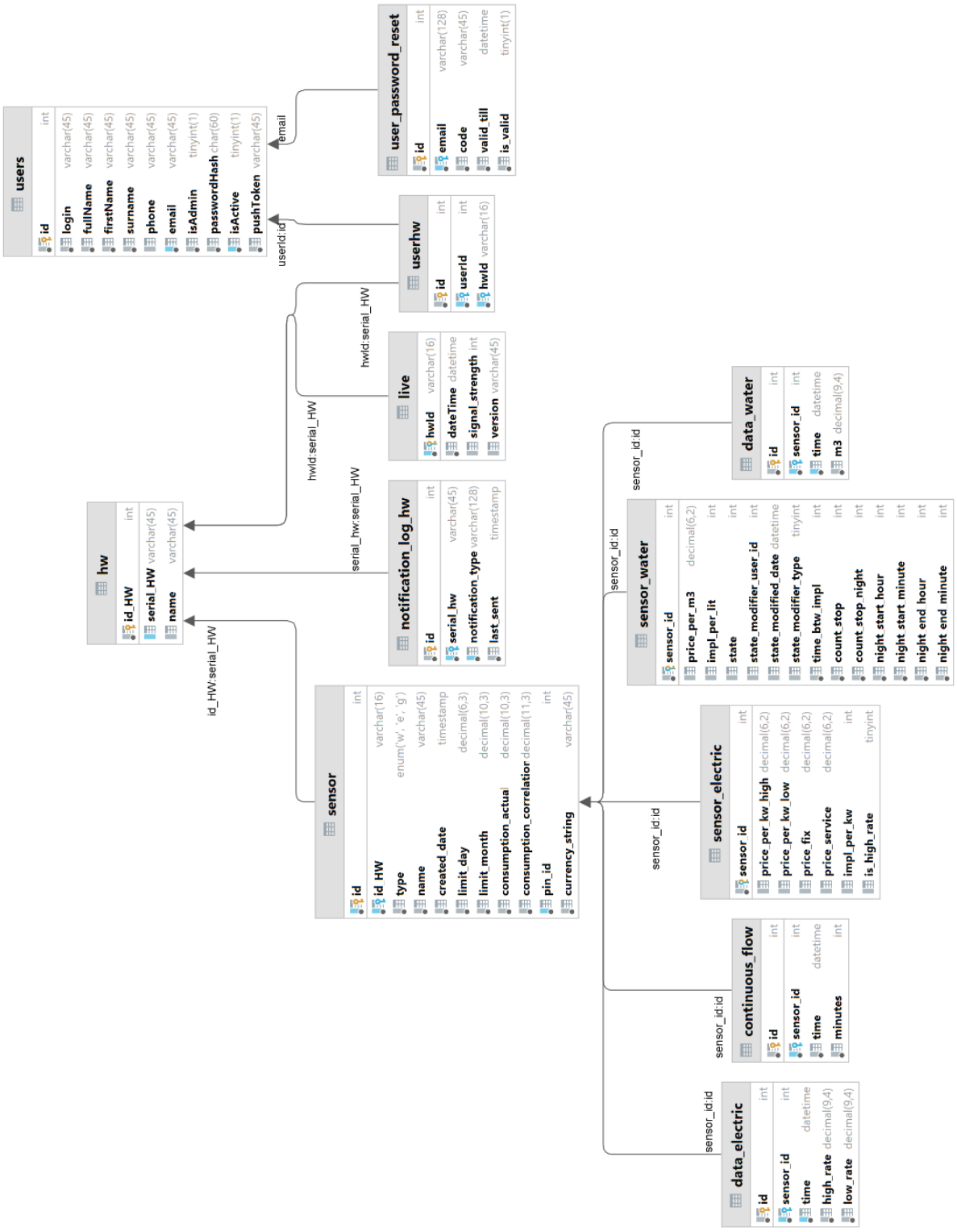
[39] BAELDUNG. *How to Deploy a WAR File to Tomcat* [online]. Bukurešť: Tarnum Java SRL, 12.2.2020 [cit. 2021-8-20]. Dostupné z: https://www.baeldung.com/tomcat-deploy-war

[40] MCKENZIE, Cameron. What are the differences between EAR, JAR and WAR files? *TheServerSide.com* [online]. Newton (Massachusetts): TechTarget, 2.7.2019 [cit. 2021-8-20]. Dostupné z: https://www.theserverside.com/feature/What-are-the-differences-between-EAR-JAR-and-WAR-files

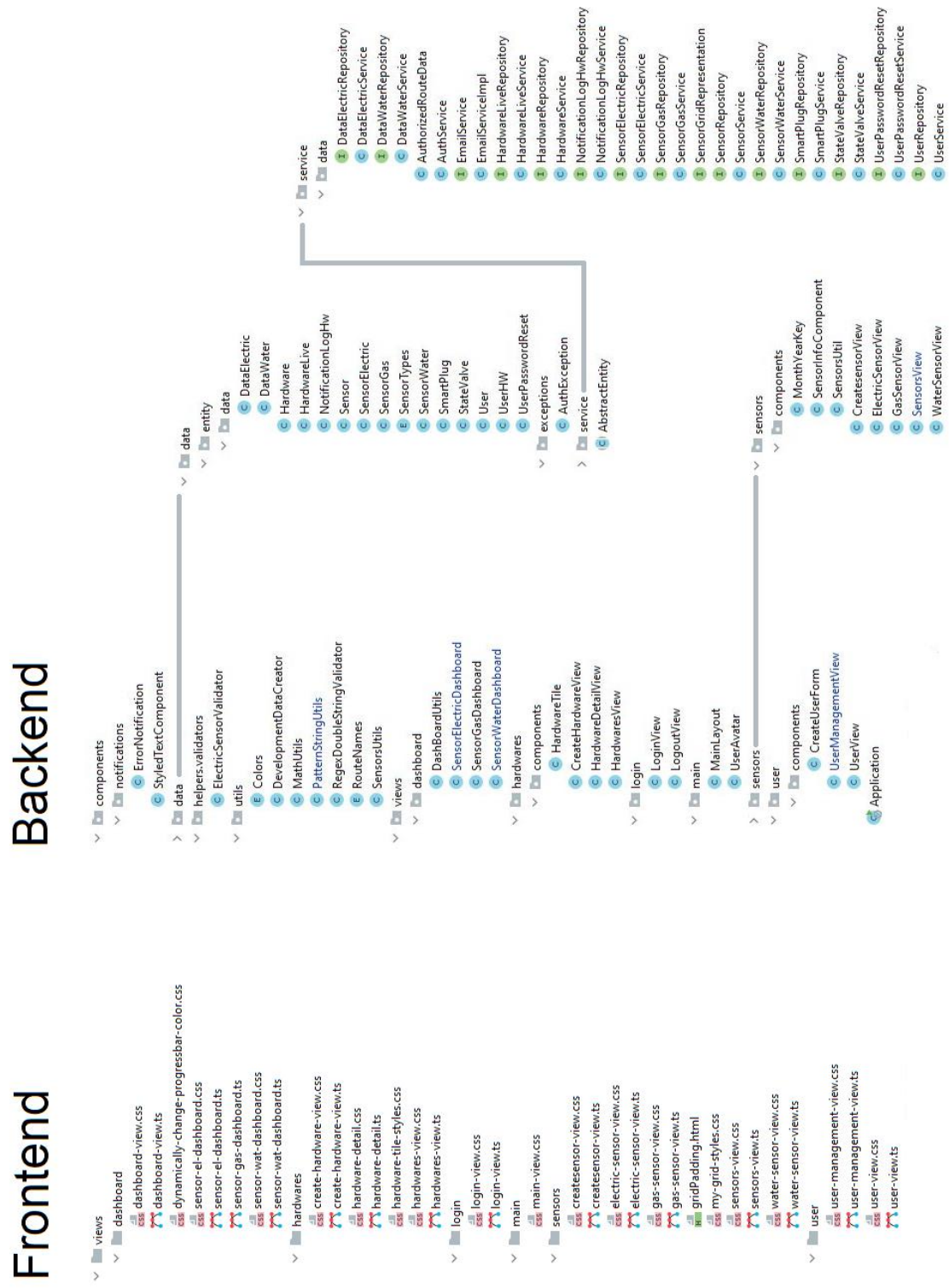
Přílohy

Příloha A – Datový model.....73
Příloha B – Rozsah implementace projektu.....74
Příloha C – Rozsah implementace projektu.....74

PŘÍLOHA A – DATOVÝ MODEL



PŘÍLOHA B – ROZSAH IMPLEMENTACE PROJEKTU



PŘÍLOHA C – ZDROJOVÉ KÓDY

K práci jsou přiloženy veškeré zdrojové kódy aplikace a související dokumenty.