

Univerzita Pardubice  
Fakulta elektrotechniky a informatiky

Návrh a implementace webové aplikace pro správu podnikové  
agendy

Bc. Jakub Pokorný

Diplomová práce

2024

Univerzita Pardubice  
Fakulta elektrotechniky a informatiky  
Akademický rok: 2023/2024

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	<b>Bc. Jakub Pokorný</b>
Osobní číslo:	<b>I22171</b>
Studijní program:	<b>N0613A140007 Informační technologie</b>
Téma práce:	<b>Návrh a implementace webové aplikace pro správu podnikové agendy</b>
Zadávací katedra:	<b>Katedra softwarových technologií</b>

## Zásady pro vypracování

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

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

Aplikace bude testována v reálném prostředí a bude nasazena na webový server pomocí kontejnerového systému Docker

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

Seznam doporučené literatury:

REYNDERS, Fanie. *Modern API design with ASP.net core 2: building cross-platform back-end systems*. New York, NY: Springer Science+Business Media, 2018. ISBN 978-1484235188  
HERMES, Dan. *Xamarin Mobile Application Development: Cross-Platform C# and Xamarin.Forms Fundamentals*. California: Apress, [2015]. ISBN 978-1484202159.

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

Datum zadání diplomové práce: **8. listopadu 2023**  
Termín odevzdání diplomové práce: **17. května 2024**

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 2023

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 1. 8. 2024

Bc. Jakub Pokorný

## Poděkování

Děkuji tímto Ing. Janu Fikejzovi, Ph.D. z Fakulty elektrotechniky a informatiky za odborné vedení s nutnou dávkou trpělivosti. Dále děkuji mému otci, že mi umožnil studovat a děkuji jmenovitě Terez a Vlastíkovi za podporu.

## **ANOTACE**

Diplomová práce je věnována návrhu a implementaci webové aplikace pro malý podnik po vzoru ERP. Cílem je obsloužit obchodní a výrobní procesy na jedné platformě. V teoretické části je popsán význam a moderní přístupy ERP. V praktické části je popsán samotný vývoj aplikace od návrhu po nasazení.

## **KLÍČOVÁ SLOVA**

ERP, podnikový proces, Spring Boot, Vaadin Flow, Docker

## **TITLE**

Design and implementation of a web application for business management

## **ANNOTATION**

The diploma thesis is devoted to the design and implementation of a web application for a small enterprise on the ERP model. The goal is to serve business and production processes on one platform. The theoretical part describes the meaning and modern approaches of ERP. The practical part describes the actual development of the application from design to deployment.

## **KEYWORDS**

ERP, business process, Spring Boot, Vaadin Flow, Docker

# OBSAH

<b>Seznam obrázků</b>	<b>9</b>
<b>Seznam tabulek</b>	<b>10</b>
<b>Seznam zkratk</b>	<b>11</b>
<b>Úvod</b>	<b>13</b>
<b>1 Podnikové procesy</b>	<b>14</b>
<b>2 Enterprise Resource Planning</b>	<b>16</b>
2.1 Výhody a nevýhody . . . . .	18
2.2 Implementační modely ERP . . . . .	20
2.3 Moderní a používané postupy . . . . .	22
2.3.1 Accelerated SAP . . . . .	23
2.3.2 Oracle Unified Method . . . . .	24
2.3.3 Microsoft Dynamics' Sure Step Methodology . . . . .	25
2.3.4 Agilní metodiky . . . . .	27
2.4 Realita střední firmy . . . . .	30
2.4.1 Procesy a používané nástroje . . . . .	31
<b>3 Použité technologie</b>	<b>32</b>
3.1 Spring Boot . . . . .	32
3.2 Vaadin Flow . . . . .	35
3.3 Docker . . . . .	36
3.4 Služby . . . . .	39
3.4.1 ARES API . . . . .	39
3.4.2 Amazon S3 . . . . .	39
3.4.3 Gmail API . . . . .	40
<b>4 Návrh aplikace</b>	<b>41</b>
4.1 Požadavky . . . . .	41
4.2 Use Cases . . . . .	43

4.3	Databázový model . . . . .	44
4.4	Uživatelské rozhraní . . . . .	45
<b>5</b>	<b>Implementace aplikace</b>	<b>49</b>
5.1	Příprava vývojového prostředí . . . . .	49
5.2	Back-end . . . . .	50
5.2.1	Datový model . . . . .	52
5.2.2	Databázová vrstva . . . . .	55
5.2.3	Byznys vrstva . . . . .	56
5.2.4	Výjimky . . . . .	57
5.3	Front-end . . . . .	57
5.3.1	Hlavní menu . . . . .	58
5.3.2	Zobrazení dat . . . . .	59
5.3.3	Správa dat . . . . .	59
<b>6</b>	<b>Testování</b>	<b>62</b>
<b>7</b>	<b>Nasazení</b>	<b>63</b>
	<b>Závěr</b>	<b>64</b>
	<b>Použitá literatura</b>	<b>65</b>
	<b>Seznam příloh</b>	<b>68</b>
	<b>Příloha A – Webová aplikace</b>	<b>69</b>



# SEZNAM OBRÁZKŮ

1	Architektura ERP na začátku milénia [7, 22] . . . . .	17
2	Benefity ERP [19] . . . . .	19
3	Přehled služeb pro Dynamics 365 [3] . . . . .	27
4	Agilní hodnoty [15] . . . . .	28
5	Scrum framework [21] . . . . .	29
6	Spring Boot architektura [24] . . . . .	34
7	Možnosti s Vaadin [25] . . . . .	35
8	Jak Docker usnadnil nasazení [9] . . . . .	37
9	Základní koncept Dockeru [9] . . . . .	38
10	Metodika UP [2] . . . . .	41
11	Use Case diagram . . . . .	43
12	Celkový databázový model . . . . .	44
13	UI návrh . . . . .	46
14	UI návrh pro mobilní zařízení . . . . .	48
15	Ukázka ze souboru <i>application.dist.yml</i> . . . . .	50
16	Struktura projektu webové aplikace . . . . .	51
17	Ukázka použití anotací . . . . .	52
18	Ukázka vazeb v entitách . . . . .	54
19	Ukázka <i>NamedEntityGraph</i> . . . . .	55
20	Ukázka implementace v databázové vrstvě . . . . .	55
21	Ukázka filtrování pomocí Specification . . . . .	56
22	Ukázka implementace v byznys vrstvě . . . . .	57
23	Ukázka výjimky 404 . . . . .	58
24	Ukázka implementace UI . . . . .	59
25	Ukázka data binding ve formulářích . . . . .	60
26	Ukázka formuláře pro CRUD operace . . . . .	61
27	Ukázka testu . . . . .	62

# SEZNAM TABULEK

1	Implementační modely ERP [11]	21
2	Základní příkazy Dockeru [9]	39
3	Funkční požadavky	42
4	Nefunkční požadavky	42

# SEZNAM ZKRATEK

ACLs	Access Control Lists
AI	Artificial Intelligence
AIM	Application Implementation Methodology
AMES	Agile Method for ERP Selection
AOP	Aspect-oriented Programming
API	Application Programming Interface
ASAP	Accelerated SAP
ARES	Administrativní Registr Ekonomických Subjektů
AWS	Amazon Web Services
BPM	Business Process Management
BPMN	Business Process Model and Notation
BPMS	Business Process Management System
CRM	Customer Relationship Management
CRUD	Create Read Update Delete
CSF	Critical Success Factors
CSS	Cascading Style Sheets
CSR	Client-Side Rendering
DAO	Data Access Object
DDL	Data definition language
EPM	Enterprise Performance Management
ERP	Enterprise Resource Planning
HCM	Human Capital Management
HR	Human Resource
HTML	Hyper Text Markup Language
IaaS	Infrastructure as a Service
IAM	Identity and Access Management
IDE	Integrated Development Environment
IČO	Identifikační Číslo Osoby
IoC	Inversion of Control
IT	Information Technology

JDBC	Java Database Connectivity
JPA	Java Persistence API
JPQL	Jakarta Persistence query Language
JS	JavaScript
KMS	Key Management Service
MRP	Material Requirements Planning
MVC	Model, View, and Controller
NIST	United States National Institute of Standards and Technology
NLP	Natural Language Processing
npm	Node package manager
OCI	Oracle Cloud Infrastructure
ORM	Object-Relational Mapping
OUM	Oracle Unified Method
PaaS	Platform as a Service
PDF	Portable Document Format
PLM	Product Lifecycle Management
REST	Representational State Transfer
RPA	Robotic Process Automation
SaaS	Software as a Service
SAP	System Analysis Program Development
SCM	Supply Chain Management
SPA	Single-Page Application
SpEL	The Spring Expression Language
SSL	Secure Sockets Layer
SSO	Single Sign-on
SSR	Server-Side Rendering
S3	Simple Storage Service
TLS	Transport Layer Security
UC	Use Case
UI	User Interface
UML	Unified Modeling Language
UP	Unified Process

# ÚVOD

Zavedené i nové podniky nejrůznějších velikostí a zaměření mají po celém světě společné téma. Musí být řízeny a být ve svých podnikových agendách maximálně efektivní. Pokud nejsou vedeny efektivně, mohou ztratit konkurenční výhodu a následně zaniknout. V našem globálním světě informačních technologií se firmám nabízí nepřeberné množství služeb a aplikací, které mohou jejich triviální nebo komplexní procesy účinně řešit. V rámci současného trendu se tyto nástroje přesunuly převážně do cloudu a výpočetní výkon je již natolik vysoký, že si obvykle nejlépe vedou společnosti, které mohou činit operativní rozhodnutí v reálném čase, které jsou podložené daty a komplexními analýzami.

Úspěšné firmy, tak jednoznačně potřebují informační systémy pro byznys, a již od prapočátku prvních počítačů je jim dodáváno, ačkoliv se tomu vždy neříkalo Enterprise Resource Planning (ERP).

Tato diplomová práce se zabývá právě tímto vzkvétajícím odvětvím IT. Primárním cílem této práce je navrhnout a implementovat komplexní webovou aplikaci pro efektivní správu a sledování životního cyklu zakázek a to včetně skladového hospodářství klíčových komponent, z kterých se konstruuji prodávané produkty. Teoretická část vysvětluje koncept a technologii ERP a blíže jsou zkoumány moderní implementace. Praktická část primárně popisuje tvorbu webové aplikace, která kombinuje podnikové procesy po vzoru ERP. Aplikace je psána v jazyce Java s použitím frameworků Spring Boot a Vaadin Flow, které jsou stručně popsány v kapitole věnované použitým technologiím. Další kapitoly jsou dedikovány návrhu, analýze, implementaci, testování a nasazení vyvinuté webové aplikace.

# 1 PODNIKOVÉ PROCESY

Sousloví podnikový proces je v ang. literatuře hledáno pod heslem business process a již Google vyhledávač bude našeptávat o business process management, modeling, reengineering, modeling notation, outsourcing, automation nebo improvement. Již první setkání s tématem napovídá, že se jedná o komplexní problematiku, které má mnoho odnoží a vlastní spleť životní cyklus. Kde od modelování (modeling) na zelené louce nebo ze stávajícího podnikového procesu (reengineering) skrze standardy modelovacího jazyka či notace (modeling notation) zjistíme, že možná by bylo lepší proces raději automatizovat (automation) nebo rovnou předat jinam, než jej řídit vlastními zdroji (outsourcing).

Pod podnikovým procesem si můžeme představit:

- finanční procesy, kde je potřeba vést účetnictví, spravovat pohledávky nebo rozpočty,
- procesy s lidmi, zaměstnanci, kde nábor či výběr má svůj proces, kolik a jak se budou zvyšovat platy, kdo a kdy bude vyslán na školení nebo kariérní růst,
- výrobní procesy, kde je potřeba plánovat vzhledem k zásobám, termínům a kvalitě,
- prodej a marketing, kde hledáme zákazníka nebo jak a komu co nabízet,
- a další.

Obecně z definice se jedná o firemní činnost transformace vstupů do hodnoty pro samotný podnik [17].

Metodologií pro konkrétní podnikové procesy je Business Process Management (BPM). V rámci BPM, která má mimo jiné za cíl efektivitu a flexibilitu organizace, se procesy modelují skrze Business Process Management System (BPMS), které využívají například Business Process Model and Notation (BPMN) [5]. Obdobou BPMN je Unified Model Language (UML), který je vhodný pro modelování softwarových systémů.

Celá poptávka po BPM stojí na potřebě každé firmy být řízena a vytvářet hodnotu. Z podstaty věci je zřejmé, že firmy s podobným zaměřením budou mít podobné podnikové procesy s rozdíly v detailech kvůli odlišnému působišti, velikosti, jiné firemní kultuře apod. Například:

- výrobní firma potřebuje výrobní a doručovací proces,
- kamenný obchod potřebuje proces pro skladové zásoby, vyřízení objednávek a komunikaci se zákazníkem,

- technologická firma potřebuje řídit softwarový vývoj, finanční reporty a zákaznickou podporu,
- apod.

a zde je prostor pro nejrůznější pomocné nástroje a služby pro odbavení třeba i jen režijní potřeby jako je firemní proces komunikace: email, Slack, Teams, ZOOM a další. Pro složitější či specifické potřeby ku příkladu jako mapování potenciálních zákazníků a komunikace s nimi neboli Customer Relationship Management (CRM): Raynet, Salesforce a další. Konceptně i prakticky si tyto nástroje můžeme představit jako komplexní informační systémy, které integrují více podnikových procesů. Tento koncept a systémy jsou nazývány Enterprise Resource Planning (ERP).

## 2 ENTERPRISE RESOURCE PLANNING

Prapočátky myšlenek jak řešit podnikové procesy a jejich propojení sahá do 60. let minulého století. Jednalo se především o systémy pro řízení nejrůznějších inventářů a skladů, kde byla potřeba identifikovat skladovací podmínky, určit si cíle, provádět inventury, či vylepšovat techniky skladování. Systémy byly velké a pomalé a vyžadovaly technický personál. [7]

V 70. letech mluvíme o prvních Material Requirements Planning systems (MRP). Posun ve výpočetní technologii umožnil se více zaměřit na potřeby trhu, např. na plánovaný výrobní proces. V roce 1972 vzniká i společnost SAP (System Analysis Program Development), která dodnes vyvíjí software pro podnikové procesy. Systémy jsou stále pomalé, drahé a nepropojené s častým následkem, že se cíle procesů neseťkávají s technickými možnostmi. [7]

V 80. letech, v 2. generaci MRP II se vývoj zaměřuje na optimalizaci a synchronizaci výrobních procesů. Vzniká další velká společnost PeopleSoft, kterou 2005 odkoupil Oracle. Mezi výzvy doby patří rozšíření na více platforem, zlepšit sběr přesných informací a chybí funkce pro plánování a automatizaci. [7]

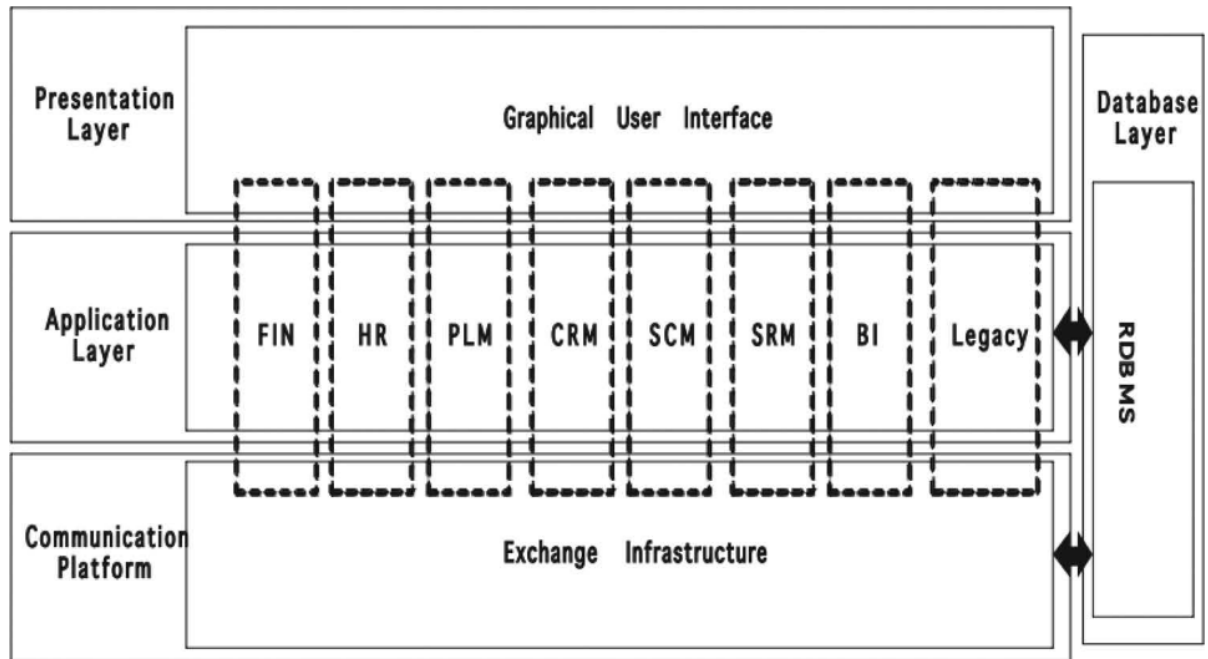
V 90. letech se poprvé objevuje termín ERP od Gartner Group jako popis nové generace softwarů MRP II. Vznik relačních databází akceleruje vývoj. SAP R3 začíná používat klient-server architekturu. Vznikají programy, které integrují více procesů např. pro marketing, finance a human resource (HR) tj. CRM nebo různé varianty výrobních a logistických procesů tj. Supply Chain Management (SCM). Systémy běží na více platformách. Výzvy spočívaly ve složitých a nekonečných implementacích do společností, kde firmy byly nuceny měnit svoje procesy pro chod daného systému a vše se tak časově natahovalo. [7]

V nultých létech s rozvojem cloudu se i ERP mění na novou třívrstvou architekturu: komunikační, aplikační a prezenční viz Obrázek 1. Tato architektura definována United States National Institute of Standards and Technology (NIST) jako standard, umožňovala rychlejší rozvoj, kde identifikovala 3 modely služeb: [7]

1. Software as a Service (SaaS): aplikace je dodána jako uživatelské rozhraní a uživatel nespravuje zbylé spodní vrstvy. (viz Obrázek 1). Např. Microsoft 365, Netflix, Zoom.
2. Platform as a Service (PaaS): poskytovatel zprostředkovává middleware na kterém je schopen developer vytvořit SaaS. Např. Heroku, Microsoft Azure - App Services.



3. Infrastructure as a Service (IaaS): se nabízí jako výpočetní výkon, uložení, síťové spojení na kterém lze nasadit a provozovat systémy. Např. Microsoft Azure App Service, Amazon Web Services (AWS), Oracle Cloud Infrastructure (OCI).



Obrázek 1: Architektura ERP na začátku milénia [7, 22]

Zpětně od poloviny nového tisíciletí bylo rozhodnuto, že internet je budoucnost i pro ERP. Formuje se několik lídrů v IaaS i PaaS a vznikají desítky SaaS společností. Draze na míru vyvíjené ERP postupně migrují nebo rovnou zanikají v průběhu času, protože není nutné dál udržovat vlastní řešení. Další překotný vývoj v ERP je ve znamení rychlosti a dostupnosti takových služeb. Koncový byznys proces má i několik možností, jak být obslužen skrze webové aplikace [7]. Např. SaaS společnost na fakturování, která potřebuje řídit svůj projekt si může vybrat SaaS jako Basecamp, Asana a Trello k tomu určené a naopak. Basecamp, Asana a Trello může k fakturaci využít SaaS pro fakturaci. Současná a budoucí iterace se zaměřuje a je ovlivněna rozvojem umělé inteligence (AI), machine learning, robotic machine automation (RPA), natural language processing (NLP), in-memory databázemi transakčních i nestrukturovaných dat [19].

Po objasnění a představení původu je pod pojmem ERP rozuměno jako:

- ERP je koncept, kde je cílem kontrolovat byznys procesy, používat správně informace a integrovat firemní procesy s maximální efektivitou a tím vytvářet nejlepší možnou hodnotu pro organizaci.

- ERP je technologická infrastruktura, systém, který mění ERP koncept v realitu skrze schopnost propojovat procesy se svými nástroji. V tomto smyslu ERP není onen jeden nástroj propojení ale celý ekosystém integrovaných nástrojů.

Jiné definice se na ERP mohou dívat z perspektivy funkce, technologie či strategickou hodnotu organizace [12].

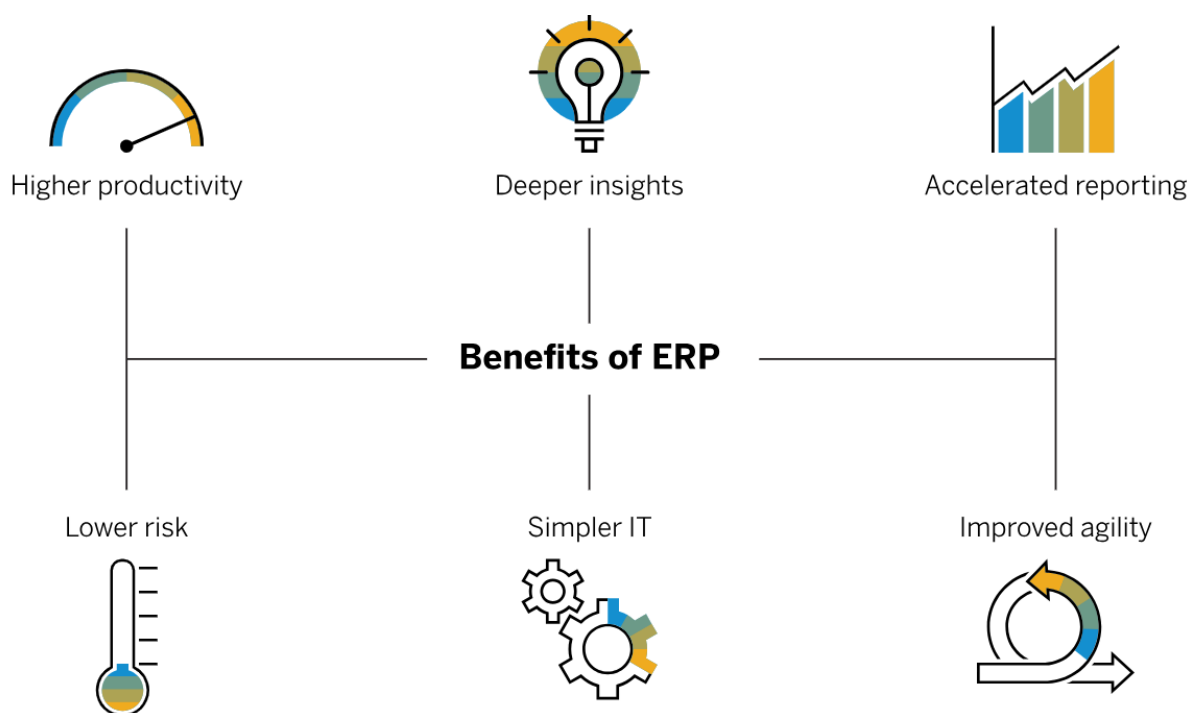
## 2.1 Výhody a nevýhody

SAP, jako jeden z lídrů ERP, uvádí na svých stránkách tyto obecné výhody ERP (viz Obrázek 2) [19]:

1. Vyšší produktivita skrze automatizaci hlavních obchodních procesů s nižšími náklady na provoz.
2. Hlubší poznání skrze eliminaci informačních sil <sup>1</sup>, kde jsme schopni rychleji získat jednotnou odpověď na kritické byznys otázky.
3. Akcelerace reportování, jak ERP má přístup k datům je schopen vykreslovat nejrůznější reporty, které jsou užitečné nejen pro finanční pohled. Díky tomu organizace může reagovat na změny v reálném čase.
4. Snížení rizik skrze vyšší kontrolu a znalosti o svém byznysu. Respektive firma ví co se ve společnosti děje či neděje a tím je vedení umožněno rozpoznávat rizika a učinit příslušné kroky pro eliminaci takových rizik.
5. Jednodušší IT skrze integrace softwarových nástrojů. Snižuje se náročnost včetně databázové vrstvy čímž se usnadňuje práce všem.
6. Zvýšení akceschopnosti v reálném čase díky vyšší efektivitě zpracování jednotlivých kroků v rámci byznys procesů. Obdobně jako u snižování rizik je tak firma schopna adekvátněji reagovat či využít nové možnosti.

---

<sup>1</sup>Systém, který není schopen komunikovat s jinými informačními systémy.



Obrázek 2: Benefit ERP [19]

Informace o nevýhodách ERP se na stránkách SAP nepodařilo dohledat. Úskalí ERP však lze najít v samotné implementaci pro organizace, který chtějí těžit z benefitů a v průběhu implementace zjistí, že cesta k uvedeným benefitům je dlouhá. A když už jsou v pokročilé fázi, kde je možné těžit přínosy, zjišťují, že to nemá nebo nebude mít takový dopad, jak bylo v prapočátku myšleno a smýšleno. Výsledkem takových nezdarů pak jsou funkční, ale nepoužívané aplikace i vinou nefunkčních procesů či špatné vize organizace. Respektive v ERP je potřeba počítat s klíčovými prvky (CSF). Např. klíčová osoba ve vedení může svoji nekompetentností zapříčinit selhání progresivní myšlenky v podobě nenaplnění slibovaných výhod v stanoveném termínu za uvolněné zdroje. Pokud se tak stane, většinou se pak právě na straně nevýhod uvádí [6]:

1. dlouhá implementace,
2. vysoké náklady na protahující se vývoj a údržbu,
3. komplexní a neintuitivní,
4. nedodržená flexibilita a výstup,

Tyto nevýhody se tedy vyskytují spíše u nepovedených implementací. Mezi obecné nevýhody i povedených implementací patří možný odpor ze strany uživatelů, pro které je fungování v rámci ERP složité, protože nové systémy nabízí mnoho nových možností a pro

vyšší technickou náročnost to může být problém. Zároveň z podstaty cíle automatizace mohu zanikat v organizaci pracovní místa [18].

## 2.2 Implementační modely ERP

Pro úspěšné nasazení ERP vznikla řada kuchařek, strategií <sup>2</sup> a metodik <sup>3</sup>.

Standardní metodika pro implementaci ERP [11]:

1. Příprava, kde se vybere dodavatel na základě srovnávacích analýz, dohodnou se podmínky smlouvy a samotný podpis smluv.
2. Příprava projektu včetně připraveného rozvrhu, alokace zdrojů apod.
3. As-Is studie s cílem pochopení, kde se proces nachází a kam se má dostat. Je potřeba brát ohled na současné scénáře, data a toky informací.
4. To-Be návrh skrze provedený reinženýring na stávajících byznys procesech optimalizuje nebo přímo redukuje stávající byznys procesy. Tím se snižují náklady na každý jednotlivý proces. Výsledná mapa optimalizovaných procesů se nazývá To-Be mapa.
5. Srovnávací analýza (Gap analysis) mezi To-Be mapou a ERP softwarem. Cílem je dokumentace požadavků na úpravy.
6. Konfigurace systému, kde systém je konfigurován vstupními daty a proměnnými.
7. Pilotní provoz, kde vedení organizace schvaluje změny.
8. Školení personálu.
9. Testování uživateli, kde se ladí procesy na základě zpětné vazby od uživatelů.
10. Migrování dat ze starého systému na nový.
11. Go-Live neboli konečný přechod z původního systému na nový. Alternativou může být paralelní běh s starším systémem před konečným ukončením.
12. Fáze po implementaci (Post-Implementation), kde je prováděna oprava chyby a dodání aktualizací.

I zde je patrná provázanost významu ERP jako konceptu a technologické infrastruktury, která se metodikou podobá životnímu cyklu vývoje softwaru, mimo As-Is studii, To-Be návrhu a srovnávací analýzy, které jsou pro koncept ERP klíčovými fázemi [11].

Existují i jiné frameworky s různým zaměřením a fázemi (viz Tabulka 1).

---

<sup>2</sup>Strategie mnohdy odkazuje k důležitým dlouhodobým cílům [11].

<sup>3</sup>Metodika je schéma určující postup provádění odborné činnosti vycházející z vědeckého poznání [13].

Tabulka 1: Implementační modely ERP [11]

<b>Autor</b>	<b>implementační model ERP</b>
Bancroft et al. (1998)	(1) Soustředění, (2) Vytváření As-Is mapy, (3) Vytvoření To-Be návrhu, (4) Konstrukce a testování (5) Samotná implementace.
Kuruppuarachchi et al. (2000)	(1) Zahájení, (2) Definice požadavků, (3) Vývoj, (4) implementace, (5) Ukončení.
Makipaa (2003)	(1) Iniciativa, (2) Vyhodnocení, (3) Výběr, (4) Modifikace, reinženýring byznys procesu a konverze dat, (5) Trénování, (6) Go-Live, (7) Ukončení, (8) Provoz a rozvoj.
Parr and Shanks (2000a)	(1) Plánování, (2) Projektování: a. nastavení, b. reinženýring, c. modelování, d. konfigurování and testování, e. instalace (3) Provoz.
Ross (1999)	(1) Návrh, (2) Implementace, (3) Stabilizace, (4) Rozvoj, (5) Transformace.
Verviel and Halington	(1) Plánování, (2) Informační výzkum, (3) Výběr, (4) Vyhodnocení, (5) Jednání.

K strategii učení cílu ERP lze obdobně jako u metodik dohledat návody, které prošly vývojem a stále se vyvíjí především s preferencí agilního vývoje. Klíčovým prvkem se zdá být fakt zdali organizace chce ERP budovat od základu, zaplatit si řešení, nebo potřebuje pouze využít konzultačních služeb na stávající ERP. Pak lze kategorizovat takto [11]:

- Na zakázku: organizace má specifické a unikátní požadavky, že zvolí vývoj na míru, mnohdy vlastními silami. Pak se postupuje v rámci představených modelů i s využitím agilního vývoje.
- Dle dodavatele: dodavatel doporučuje vlastní řešení s vlastními metodikami. Automaticky probíhá mnohem více interakce a kroky jdou mnohdy paralelně. Mezi hlavní metodiky dle dodavatele patří:
  - Accelerated SAP (ASAP) od SAP,
  - Oracle Unified Method (OUM) od Oracle,
  - Microsoft Dynamics' Sure Step od Microsoft.
- Specifické dle konzultanta: konzultant má svoje vlastní postupy, které jsou kombinací jak metod hlavních dodavatelů, tak svých zkušeností z vlastní praxe.

Tak či onak je dobré maximálně využít dostupné nástroje a nevytvářet je znovu. Automaticky se tak šetří především časové a finanční zdroje, které jsou často uváděny na stranu nevýhod ERP [11].

## 2.3 Moderní a používané postupy

Mezi hlavní celosvětové dodavatele komplexních ERP řešení patří SAP, Oracle a Microsoft. Jejich postupy implementace mají své specifikace, ale kontinuum s představenými implementacemi ERP mají. Vždy se jedná o silný důraz na počáteční fázi plánování, budování modelu na základě analýzy, implementace a konfigurace v rámci nabízených nástrojů, průběžné i finální testování a nezapomínají na proškolení organizace, opatrné schválené nasazení s nabízenou podporou a aktualizacemi. Do jejich metodik, tak jak určuje trend, se prolínají i agilní metodiky [11].

Samotné nasazení může probíhat ve třech variantách a to [19]:

- Cloud ERP: tj. nejoblíbenější varianta, kde cloudové ERP je nabízeno jako služba za předplatné. Odpadá tak starost s náklady, aktualizacemi včetně těch bezpečnostních. Výhodami jsou nižší pořizovací náklady, škálovatelnost, jednodušší integrace a další. Z ERP reportu 2020, na který se odkazuje SAP, volilo 63% klientů cloudové řešení.
- On-premise ERP: dodané ERP je instalováno na vlastní datové centrum a vše je v rukou organizace včetně hardwaru. Jedná se o časté řešení, které však ustupuje a firmy přechází do cloudu. Z již jmenovaného ERP reportu 2020 volilo 37% klientů on-premise řešení.
- Hybrid ERP: hybridní řešení kombinuje cloudové a on-premise řešení, kde je tedy možné mít některé moduly u sebe a jiné v cloudu.

Z tohoto pohledu ERP směřuje do cloudu, kde je vysoký důraz na snadné integrace právě v ERP uzlu a vysoká míra konfigurace na konkrétní požadavky organizace a jejich procesů.

ERP je aktuální pro všechny velikosti a zaměření firem. Pro malé firmy může ERP znamenat především nástroj pro rychlejší růst. Střední firmy mohou benefitovat především z dobře nastavené struktury, kde udržují firmu maximálně efektivní a rozhodují se na základě již získaných dat v jednotlivých oblastech: HR, obchod, výroba a podobně. Velké nadnárodní společnosti již vyžadují velké robustní řešení, které zároveň již nemusí být on-

premise díky zrychlení technologie a mohou tak těžit především z propojení, automatizace nebo využití AI apod., kde snižují náklady na provoz, vývoj atd. [19].

### 2.3.1 Accelerated SAP

ASAP aplikuje převrácenou logiku, kde na implementaci procesu navazuje modelování a analýza. ASAP se zaměřuje implementačně na kratší projekty. Hlavní fáze:

1. Příprava projektu: zahrnuje definici, identifikaci rozsahu a specifikací, nastínění implementační strategie, termíny, sekvence implementačních kroků, ustanovení projektové organizace a výborů a v neposlední řadě alokace zdrojů.
2. Byznys plán, specifikace požadavků: závisí na vytvořené databázi během setkání s klientem. Cílem je vytvořit plán, který bere v potaz strukturu organizace a zároveň je plán vstupními daty do dokumentace o firemních procesech.
3. Konfigurace systému na základě byznys plánu: jedná se o dvoufázový postup, který se řídí přístupem shora dolů. Nejprve se provede obecná konfigurace, kde se nastaví struktura organizace s globálními nastaveními jako je např. měna. Poté se provede konfigurace firemních procesů. Vše je důkladně testováno.
4. Závěrečná příprava: zahrnuje testování systému a testování koncovými uživateli. Nedostatky jsou odstraněny a jsou provedeny zátěžové testy.
5. Nasazení do provozu a podpory: zpočátku je třeba vyšší podpory, jejíž využití postupně klesá.

Všechny fáze jsou podpořeny nástroji od SAP včetně projektového řízení. ASAP je tak komplexní a bohatá metodika [11].

Příkladem úspěšné implementace může být firma zabývající se mzdami, pracovní dobou, docházkou, financemi a účetnictvím. Firma potřebovala vylepšit stávající SAP ERP řešení a zavést SAP Human Capital Management <sup>4</sup> (HCM). Použily doporučené postupy od SAP s metodikou ASAP a povedlo se jim v krátkém čase za přijatelnou cenu zpracovat 2,5x více zaměstnanců na HR, snížit chyby v mzdovém procesu o 99% a zkrátit o 40% mzdovou uzávěrku [11].

SAP nabízí i skrze partnery prakticky vše potřebné pro ERP v nejrůznějších variantách: platformy pro technologické společnosti, CRM včetně funkcí pro úspěšnou zákaz-

---

<sup>4</sup>Sada praktik a nástrojů pro nabírání, řízení a rozvoj zaměstnanců od SAP.

nickou podporu, finanční moduly, HCM, SCM, řešení zaměřené na menší a střední firmy apod. [20].

### 2.3.2 Oracle Unified Method

Jedná se o nástupce Oracle Application Implementation Methodology (AIM), kde AIM a OUM mají společné procesy. AIM má 6 fází a 12 procesů oproti OUM, který má 5 fází a 15 procesů. OUM staví na standardu Unified Software Development Process (UP), kde OUM rozšiřuje UP, aby zahrnovala celý rozsah ERP projektů. Fáze OUM [11]:

1. Zahájení: jednání o rozsahu, požadavcích a životnosti cílů pro ERP projekt. Vše se poté schvaluje s pověřenými osobami.
2. Spolupráce: užší spolupráce s cílem vytvořit podrobné požadavky a rozdělení řešení. Mohou vznikat i prototypy za účelem porozumění a pro vytvoření základů pro další vývoj.
3. Vývoj: v této fázi se využívají vytvořené modely pro konfiguraci systému dle poskytovaných funkcí softwaru od Oracle, které doplňují funkce na míru. Modely se testují a integrují. Na konci fáze je systém připraven v beta verzi pro další kontroly a schvalovací testování.
4. Přejít: příprava na možné předání klientovi, aby přijal nový systém a kroky byly naplánovány.
5. Nasazení: systém je nasazen do provozu a je monitorován s vyšší podporou pro požadavky na pomoc. Probíhá optimalizace výkonu a opravy chyb.

Všechny úkony OUM jsou dále rozděleny do procesů a tvoří pomyslnou matici, kde fáze jsou svisle a procesy vodorovně. Procesy [11]:

1. proces byznys požadavků,
2. proces analýzy požadavků,
3. proces mapování a konfigurace.
4. proces analýzy,
5. proces návrhu,
6. implementační proces,
7. testovací proces,
8. proces pro řízení výkonu,
9. proces pro technickou architekturu,



10. proces pro sběr dat a konverzi,
11. dokumentační proces,
12. proces pro změny ve fungování společnosti,
13. školící proces
14. proces pro migraci
15. proces pro běh a podporu

Příkladem úspěšné implementace Oracle EBusiness suite s Oracle byznys aplikacemi OUM může být dopravní a cestovní společnost. Firmě se povedlo zavést byznys proces, který byl srozumitelný a jeho kritické části se podařilo integrovat s ostatními procesy což vedlo k vysoké spokojenosti zákazníků. [11]

Oracle obdobně jako konkurence nabízí řešení pro finance, projektové řízení, veřejné zakázky, řízení rizik a Enterprise Performance Management (EPM), kde jde především o správná rozhodnutí na základě dat [14].

### 2.3.3 Microsoft Dynamics' Sure Step Methodology

Jedná se komplexní metodiku od Microsoft ve spolupráci s Microsoft aplikacemi, které podporují úspěšné nasazení, migraci, konfiguraci a aktualizace v rámci řešení Dynamics, kde je cílové řešení flexibilní a škálovatelné. Sure Step se skládá ze šesti hlavních fází [8, 11]:

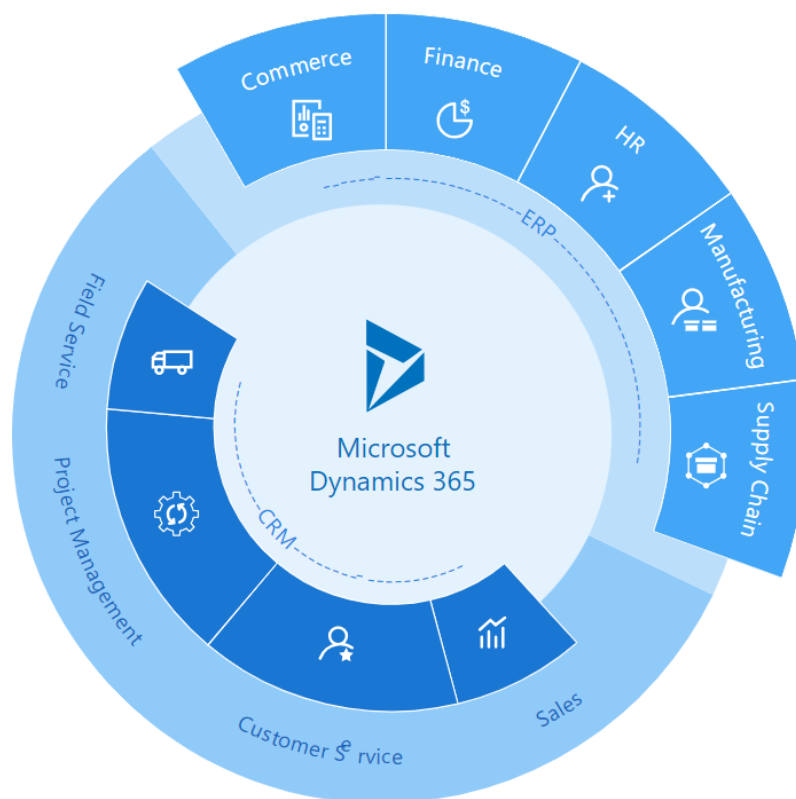
1. Fáze diagnostiky: tato fáze zahrnuje definici byznys požadavků, vyhodnocení současného řešení a identifikace důležitých osob. Cílem je získat jasnou představu o současném stavu a odhalit oblasti, kde jsou možné zlepšení.
2. Analytická fáze: vytváří se návrh řešení na základě analýzy byznys požadavků. Zahrnuje též vytvoření plánu pro projekt, odhad nákladů a definování rolí a odpovědností.
3. Fáze modelování: navrhuje se řešení, dokumentují se funkční požadavky, vytváří se testy, definují se požadavky na školení a identifikují se potřeby, které budou vyžadovat řešení na míru.
4. Fáze vývoje: implementace řešení úprav, testování dle plánu a kontrola plnění byznys požadavků.
5. Fáze nasazení: jedná se o instalaci řešení, migraci dat a závěrečné akceptační testy. Proveďte se plán školení a dokončí se závěrečné nasazení systému do provozu.
6. Fáze provozu: zahrnuje průběžnou podporu, údržbu a aktualizace.

obdobně jako OUM má i Sure Step vnitřní nástroje pro rychlou a efektivní implementaci, prověřené postupy a osm procesů v rámci některých fází [11]:

1. analýza byznys procesu,
2. konfigurace,
3. migrace dat,
4. infrastruktura,
5. instalace,
6. integrace,
7. testování,
8. školení.

Příkladem úspěšné implementace Microsoft Dynamics s metodikou Sure Step může být výrobní firma pro skládací židle a stoly. Firma potřeboval nahradit stávající ERP flexibilnějším a robustnějším systémem, který by integroval jejich CRM a Product Lifecycle Management (PLM) pro celosvětovou expanzi. Povedlo se vylepšit PLM, snížit náklady na produkci, automatizovat procesy což vedlo i k vyšší spokojenosti koncového zákazníka [11].

Microsoft je schopen doručit své řešení skrze autorizované partnery jako například Dynamics Square. Konkrétně tento dodavatel je schopen dodat moduly pro finance, obchod, zákaznickou podporu, projektové řízení, dodavatelský řetězec, marketing a jiné specifické funkce (viz Obrázek 3) [3].

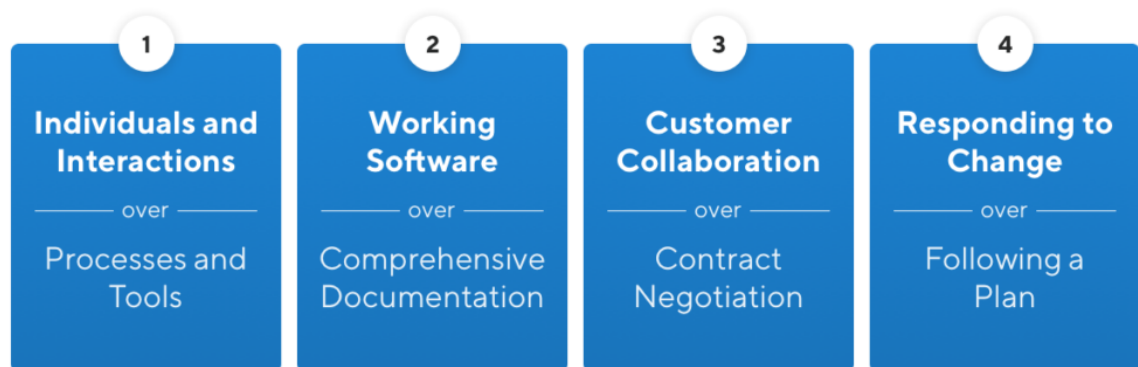


Obrázek 3: Přehled služeb pro Dynamics 365 [3]

### 2.3.4 Agilní metodiky

V IT odvětví se agilní metodiky staly extrémně populární i vzhledem k požadavku na rychlejší doručení a časté změny v požadavcích. Hlavním rozdílem mezi tradičními a agilními metodikami spočívá v myšlence vše naráz proti malým inkrementálním přírůstkům v kratším čase. Jinými slovy robustní vývoj v několika velkých iteracích proti vývoji v početných menších iteracích. Agilní metodika má sepsáno své manifesto, které obsahuje 4 hodnoty (viz Obrázek 4) a 12 principů.

## The 4 Agile Values



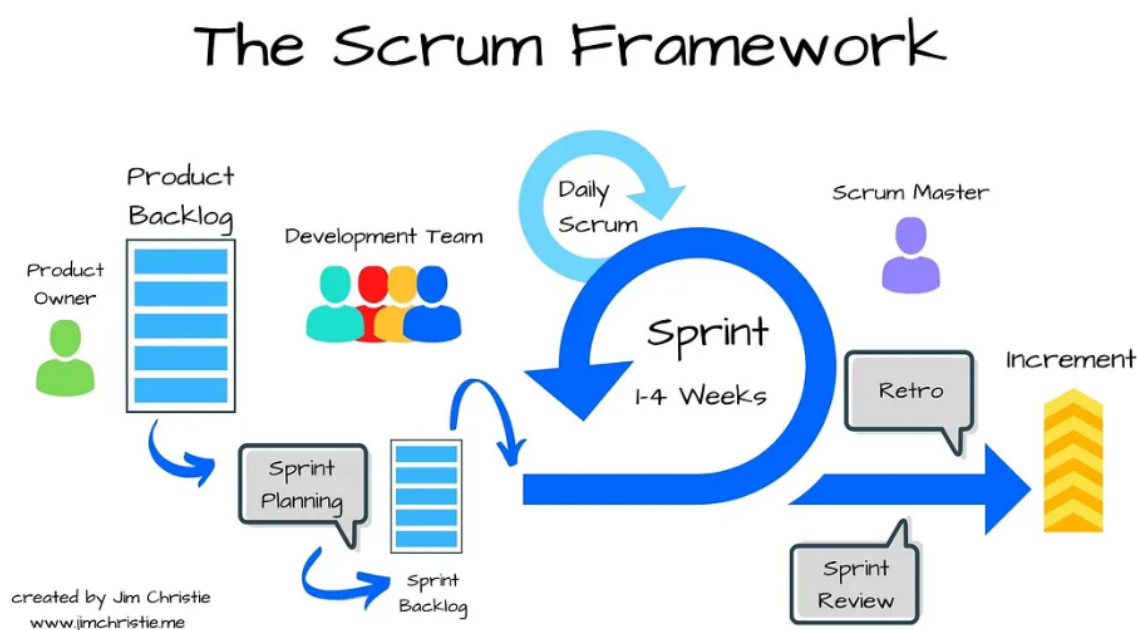
Obrázek 4: Agilní hodnoty [15]

12 agilních principů [16]:

1. Nejvyšší prioritu představuje uspokojení zákazníka skrze brzký a průběžný dodávky hodnotného softwaru.
2. Vítejte změny v požadavcích i v pozdějších fázích vývoje. Agilní procesy využívají změny pro konkurenční výhodu zákazníka.
3. Dodávat funkční software často, od několika týdnů do několika měsíců, přičemž se upřednostňují kratší lhůty.
4. Obchodníci a vývojáři v rámci projektu spolupracují na denní bázi.
5. Vytvářejte projekty kolem motivovaných lidí. Poskytněte jim potřebné prostředí, podporu a důvěru, že svou práci zvládnou.
6. Nejúčinnější a nejefektivnější metodou předání informací vývojovému týmu a v rámci něj je osobní hovor.
7. Hlavním měřítkem pokroku je funkční software.
8. Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři a uživatelé by měli být schopni udržet konstantní tempo po neomezenou dobu.
9. Věnujte trvalou pozornost technické dokonalosti a designu zvyšující obratnost.
10. Jednoduchost, tedy umění maximalizovat množství nevykonané práce, je zásadní.
11. Nejlepší architektury, požadavky a návrhy vznikají v samoorganizujících se týmech.

12. v pravidelných intervalech se tým zamýšlí nad tím, jak se stát efektivnějším a podle toho ladí a upravuje své chování.

V praxi to více či méně znamená vývoj v konceptu Scrums a Sprints (viz Obrázek: 5), kde vývojový tým pracuje v 1 až 4 týdenních Sprintech, kde výsledkem je přírůstek softwaru.



Obrázek 5: Scrum framework [21]

Implementaci ERP tento trend také neminul a již v představených implementačních postupech se s tímto přístupem počítá nebo je využíván i dodavateli. Vznikla i nová metoda pro výběr ERP, jmenuje se Agile Method for ERP Selection (AMES). Obsahuje 3 fáze, které se snaží o soulad s principy agilní metodiky [11]:

1. Představa: vytvořit předběžnou představu o cílech a rozsahu systému. Skládá se z hodnotového prohlášení a projektového modelu.
2. Iterace: identifikovat možné řešení, které nejlépe vyhovuje požadavkům organizace. Skládá se z definice požadavků, analýzy systémových schopností a hodnocení, které je řízené testováním.
3. Rozhodnutí: vybrat nebo nevybrat jedno z řešení na základě kritérií.

## 2.4 Realita střední firmy

V této kapitole popisují realitu středně velké SaaS společnosti na projektové řízení z osobní zkušenosti, která operuje především na českém trhu. V posledních několika letech poměrně vyrostla a vyšší počty zákazníků i zaměstnanců si žádají zvýšenou pozornost právě v oblasti ERP. Sama firma tedy utváří svoje vylepšené byznys procesy, řeší jak je efektivně obsluhovat a jak dál činit správné a především daty podložené rozhodnutí. Firma je pro téma ERP zajímavá i svými klienty, kde se stává, že z nabízené služby, tedy komplexního nástroje pro projektové řízení, chtějí někteří její zakazníci používat jako operativní uzel, který by v mnohém měl plnit funkce i ERP a na tuto poptávku firma musí zároveň reagovat svým vývojem.

Firma, ačkoliv by se stěželo mluvit o plnohodnotném ERP, své řešení ERP má od prvopočátku. Napsala si ho sama a obsluhovala v něm komunikaci se zákazníkem, finanční operace spojené s jejich produktem a jednoduché firemní statistiky. Jinými slovy dokázala maximálně využít výhody, že se jedná o technologickou společnost, která navíc měla k dispozici vlastní datové centrum na kterém běží jak jejich SaaS tak skromné interní ERP. V rámci několika let, kdy prakticky 1 programátor píše ERP a zbylí programátoři SaaS se společnosti daří a roste. Vznikají nové pozice zabývající se podporou, marketingem, obchodem. Z pozic se stávají oddělení a interní ERP již zkrátka nestačí.

Vnitřně se firma dostala na křižovatku, kde noví manažeři nových oddělení pod tíhou požadavků na výkon žádají změny v jejich procesech a technologické podpoře. Oddělení obchodu by bylo zřejmě výkonnější, kdyby pracovalo v plnohodnotném CRM např. od Raynet. Oddělení zákaznické podpory by prospěla vyšší integrace komunikačních kanálů, propracovanější interní systém pro snadnější odbavení zákaznických požadavků apod. Firma může pokračovat ve vlastním řešení, které nestačí, tedy investovat do jeho rozvoje a údržby nebo se nabízí vydat cestou ERP od dodavatele, které by i mohlo stále běžet na jejich hardwaru. Jak se firma rozhodla nechám otevřené, ale je zřejmé, že takové rozhodnutí může ovlivnit růst firmy na mnoho let dopředu.

Klienti této firmy řeší mnohdy identické problémy: vyšší propojení, efektivnější komunikace se zákazníkem, řešení operativních procesů v rámci firmy apod. A nezáleží na tom zda jsou konzultační, účetní, výrobní, dodavatelskou nebo technologickou firmou. Děje se tak, že v nástroji, který by měl být především nástrojem na projektovém řízení se stává

hub pro obsluhu mnoha procesů a supluje mnohdy méně kvalitně moduly pro finance, HR nebo SCM.

Z mého pozorování je realita mnohdy bez ERP. Malé firmy si vystačí bez komplexních řešení, kde jsou mnohdy zaplacené nástroje používány jinak než by měly, aby obsloužily poptávku daných procesů. Střední firmy mají prostředky a potřebu hledat řešení a stává se, že se bojí udělat velké rozhodnutí, kterým např. může být zmiňovaná výměna vlastního řešení či zajetého řešení, které již nestačí. A velké firmy jsou i velké proto, že mají určité funkční řešení, které se snaží především vylepšit.

### **2.4.1 Procesy a používané nástroje**

V této podkapitole se věnuji několika obecným byznys procesům a nástrojům, které je obsluhují. Chci tak demonstrovat, že mnohé procesy prvotně nachází odbavení právě v těchto nástrojích. Následný zájem o vzájemné propojení, sběru dat apod. přichází na řadu až když už jsou nástroje zajeté a migrace z nich může nepříjemně komplikovat fungování samotné firmy.

Firma jako celek může interně komunikovat přes Slack, Teams, Mattermost, emailem, telefonem. Projektové řízení vyřizovat v Basecamp, Asana, Trello. Interně fungovat může v Google Workspace, Microsoft 365.

Zákaznická podpora komunikující na webovém chatu může používat Crisp, Support-Box, LiveChat. Pro emailovou komunikaci: Missive, Front, Hiver, Zendesk. Pro plánování schůzek Google Calendar, Calendly. Pro online školení Demio, Zoom Webinars.

Marketing může a pravděpodobně i bude fungovat v Google ADS, Sklik, Meta Ads. Grafické přípravy bude tvořit ve Figma, Adobe Illustrator apod. Komunikaci přes maily vyřizovat přes EcoMail, Mailchimp.

Je vidno, že procesy mají mnoho alternativ, jak být obslouženy a většina z nich je dostupná jako SaaS a online. Výzvou pak může být jejich propojení a automatizace i bez znalosti programování. I takové nástroje samozřejmě existují: Make, Zapier, Automations.io.

## 3 POUŽITÉ TECHNOLOGIE

Teoretická část je zakončena touto kapitolou, která je věnována představení použité technologie v praktické části, kde navrhují a implementují systém pro malou výrobní firmu. Systém je napsán v jazyce Java ve verzi 22, sestaveným pomocí Maven s použitím Spring Boot frameworku a Vaadin Flow. Jak napovídají použité frameworky, systém bude webová aplikace s relační databází v MySQL, která poběží přes AWS. Aplikace mimo Spring Boot využívá ARES API, Gmail API a Amazon S3. Výsledná aplikace je distribuována skrze technologii Docker.

### 3.1 Spring Boot

Jedná se o framework, tedy sadu nástrojů a knihoven, která usnadňuje vývoj aplikací. Vývojář skrze nabízené moduly, které v sobě mohou obsahovat další frameworky, konfiguruje nastavení těchto modulů a tím se usnadňuje časově vývoj. Spring Boot díky Inversion of Control (IoC) se pak stará o tyto moduly a tím vývojář nemusí řešit hlavní chod programu, ale spíše zaměřuje na komponenty, které Spring Boot volá [23].

Již od začátku je Spring Boot spustitelný bez nutnosti jakékoliv konfigurace a automaticky nastavuje knihovny třetích stran. Přímě vkládá webové servery od Tomcat, Jetty a Undertow. Poskytuje funkce pro produkci jako jsou metriky, kontroly stavu a externí konfigurace. Negeneruje kód a nemá požadavky na konfiguraci XML. Zkratka "just run" [23].

Mezi hlavní funkce Spring frameworku patří [23]:

- Základní technologie.
  - Dependency injection (DI): klíčový návrhový vzor IoC mezi objekty.
  - Aspect-oriented Programming (AOP): programovací paradigma, které doplňuje OOP, kde AOP se zaměřuje na aspekty, tedy na aplikaci se dívá skrze funkčnosti, které se prolínají různými částmi aplikace. Např. logování, zabezpečení, zachytávání výjimek apod.
  - Události a Zdroje.
  - i18n: internalizace jazyka.



- Validování, data binding<sup>5</sup> a typová konverze.
- SpEL: je výkonný výrazový jazyk, který podporuje dotazy a manipulaci s objektovým grafem za běhu. Např. je použit v JPA repozitáři při filtrování dat skrze metody.
- Testování: nástroje pro rychlé a jednotkové testování všech vrstev.
  - Mockování<sup>6</sup> objektů: Jedná se o techniku, kde pouze předstíráme existenci objektu a tím zrychlujeme průběh testování.
  - TestContext framework: zajišťuje infrastrukturu pro testování.
  - Spring MVC Test: framework pro testování MVC aplikací.
- Přístup k datům: nástroje pro práci s relační databází.
  - Transakce: podpora transakčních operací v databázi.
  - DAO podpora: přístup k datům v databázi.
  - JDBC: napojení do databáze.
  - ORM: mapování objektů z databáze.
- Spring MVC a Spring WebFlux: zajišťují provoz serveru.
- Integrace: nástroje pro zasílání emailů, plánování úloh, správa cache paměti a další.
- Jazyky: podpora Kotlinu, Groovy a dynamických jazyků.

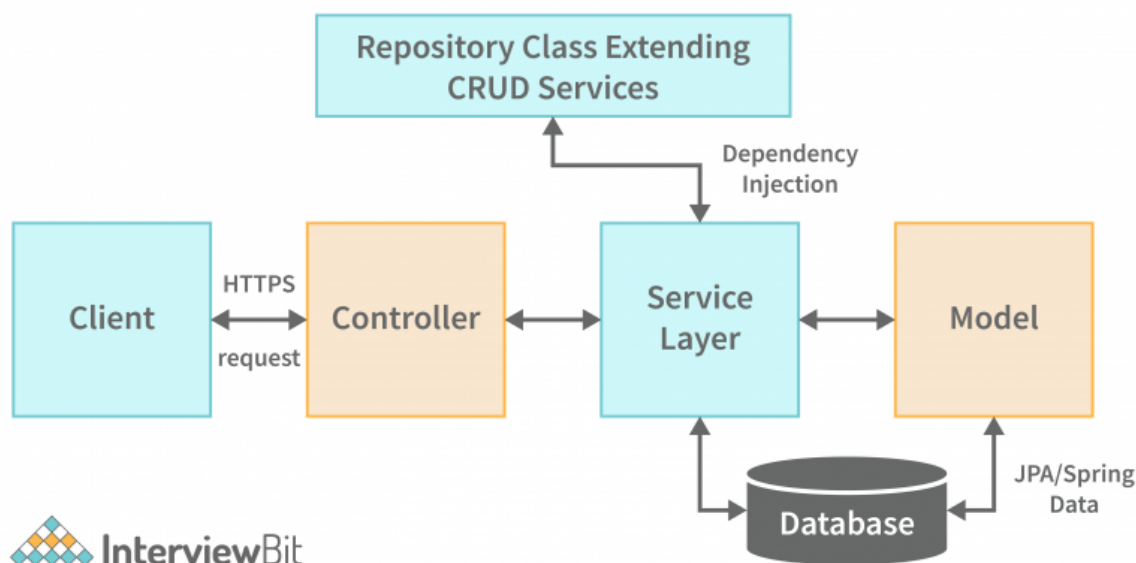
Prakticky se jedná o vše potřebné pro vytvoření REST webové aplikace v rámci představené třívrstvé architektury (viz Obrázek 1). Vývojář může pak jen skrze JPA vytvořit model, který se propíše díky nastavenému JDBC do databáze. Obdobně je JPA schopna vykonávat dotazy v databázi bez nutnosti psaní SQL, vývojář si může psát i vlastní. Výsledky dotazů se zpracovávají v aplikační vrstvě, ze které lze následně obsloužit klienta (viz Obrázek: 6).

---

<sup>5</sup>Propojení datového modelu s objektem.

<sup>6</sup>Z ang. mock.

# Spring Boot Flow Architecture



Obrázek 6: Spring Boot architektura [24]

Relační databáze je spuštěna na open-source MySQL ve verzi 8.0.33. Pro napojení do databáze stačí zapsat připojovací cestu s použitým ovladačem (JDBC) a přihlašovací údaje do konfiguračního souboru. Pro produkční účely mi byla připravena databáze již v AWS. Pro vývojové účely je vhodné založit vlastní databázi v lokálním prostředí. Pro tyto účely lze využít Docker, kde z Docker Hub stačí stáhnout oblíbený systém, nemusí to být MySQL, a zapnout v kontejneru. Pro prvotní připojení do MySQL a vytvoření databáze, do které jsem se později napojil, jsem použil open-source program DBeaver v komunitní verzi. I takto jednoduché může být finální příprava pro webovou aplikaci, nemluvě o tom, že v konfiguračním souboru lze zapsat, že se má použít in-memory databáze. To je oblíbená varianta pro testování. Kompletně je tedy běžné, že pro jeden projekt jsou využívány až tři databáze a Spring Boot je schopen si je řídit sám bez nutnosti velkých SQL skriptů.

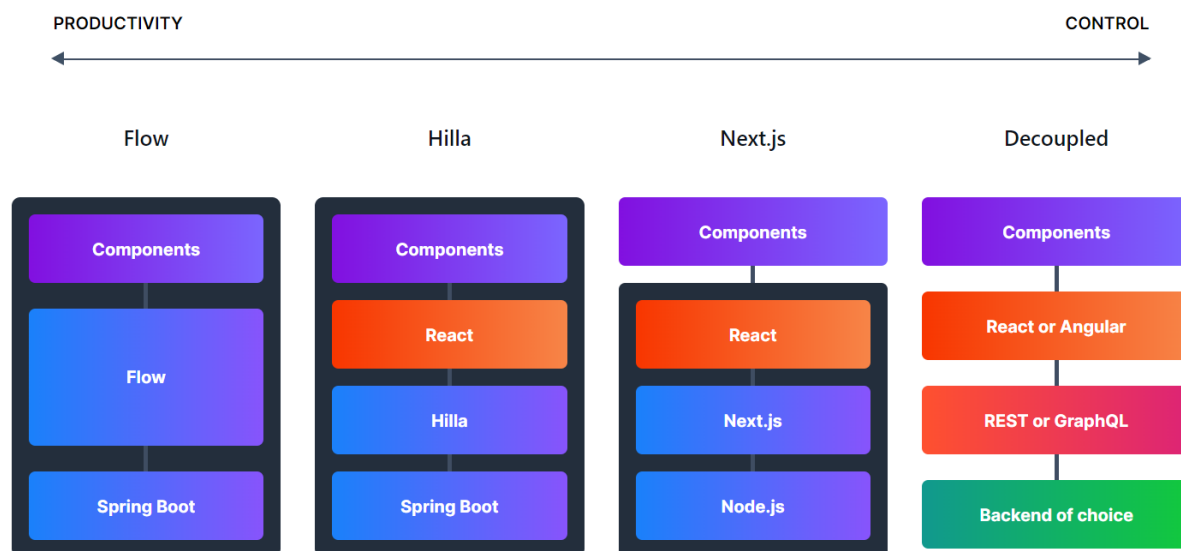
DBeaver nabízí velké množství funkcí, které vývojář ocení především při migraci dat, nebo když se chce přesvědčit o správnosti modelu v grafické podobě, kde DBeaver umí vykreslit kompletní model. Obdobné funkce nabízí i pokročilé IDE jako IntelliJ IDEA Ultimate, která byla pro projekt použita. Mezi užitečné pluginy IDE patří např. JPA Buddy, který dokáže výrazně urychlit vytvoření vlastních SQL dotazů pro JpaRepository třídu.

Spring Boot v projektu používám ve verzi 3.2.4.

## 3.2 Vaadin Flow

Vaadin je plnohodnotná platforma pro webové aplikace psaná v jazyce Java od stejnojmenné finské společnosti Vaadin Ltd. Platforma se stále vyvíjí a je dostupná v několika variantách a to i open-source komunitní verzi. Pro projekt je využita komerční verze Vaadin Flow (viz Obrázek 7) v kombinaci se studentskou licencí poskytovanou v rámci GitHub Student Developer Pack [25].

Vaadin věří, že full-stack přístup je maximálně efektivní cesta produkce kvalitních aplikací. Vaadin Flow nabízí napojení na Spring Boot, který tvoří beck-end a knihovnu komponent z kterých lze vytvořit front-end čistě v Javě. Vaadin se stará o renderování, události, balíčky apod. Společnost nabízí i jiné projení beck-end s front-end aplikace(viz Obrázek: 7). Mezi nejoblíbenější knihovny pro front-end patří knihovna React a i právě proto Vaadin skrze své řešení pojmenované Hilla tuto variantu nabízí a tím se již automaticky nebude jednat o 100% řešení v Javě, ale i JavaScriptu a TypeScriptu. Jak již bylo zmíněno, pro projekt byl použit Vaadin Flow, dle požadavku na technologii od zadavatele [25].



Obrázek 7: Možnosti s Vaadin [25]

Vaadin provádí renderování logiky a UI na straně serveru (SSR). Výstup v podobě HTML, CSS a JS jsou následně posílány klientovi, kde se bez větší nutnosti dalšího zpra-

cování data vykreslí. Tj. rozdíl oproti React, kde většina renderování se provádí v prohlížeči (CSR). V React se postupuje po modulech, které nám kompletuje npm nebo yarn. U Vaadin je přístup monolitický skrze integrované řešení [25].

Mezi hlavní výhody Vaadin patří [25]:

- Vaadin automaticky zabezpečuje komunikaci mezi serverem a prohlížečem. Bezpečnostní opatření jako automatické session klíče, kontroly vstupů a validace na straně serveru jsou výchozím chováním. Vaadin podporuje Spring Boot Security i SSO<sup>7</sup>.
- Jedná se o kompletní řešení, které funguje bez nutnosti konfigurace. Obsahuje frameworky, UI komponenty, má zabudovaný systém pro podporu komponent, které se modelují ve Figma, nástroje pro kompletování a testování i UI komponent.
- Podporuje komunikaci v reálném čase, kde aktualizuje komponenty, dodává data z back-end skrze webové sokety a tzv. lazy loading<sup>8</sup>. Zabudované funkce pro paralelní editování a kolaboraci jako v nástrojích od Google, Notion apod.

Mezi nevýhody patří závislost na serveru, kde Vaadin není tradiční SPA. V rámci Vaadin existují i komponenty které plní SPA vlastnosti, ale stále se jedná o SSR, kde může nastat nutnost optimalizace pro výkon, protože SSR může být náročnější než CSR [25].

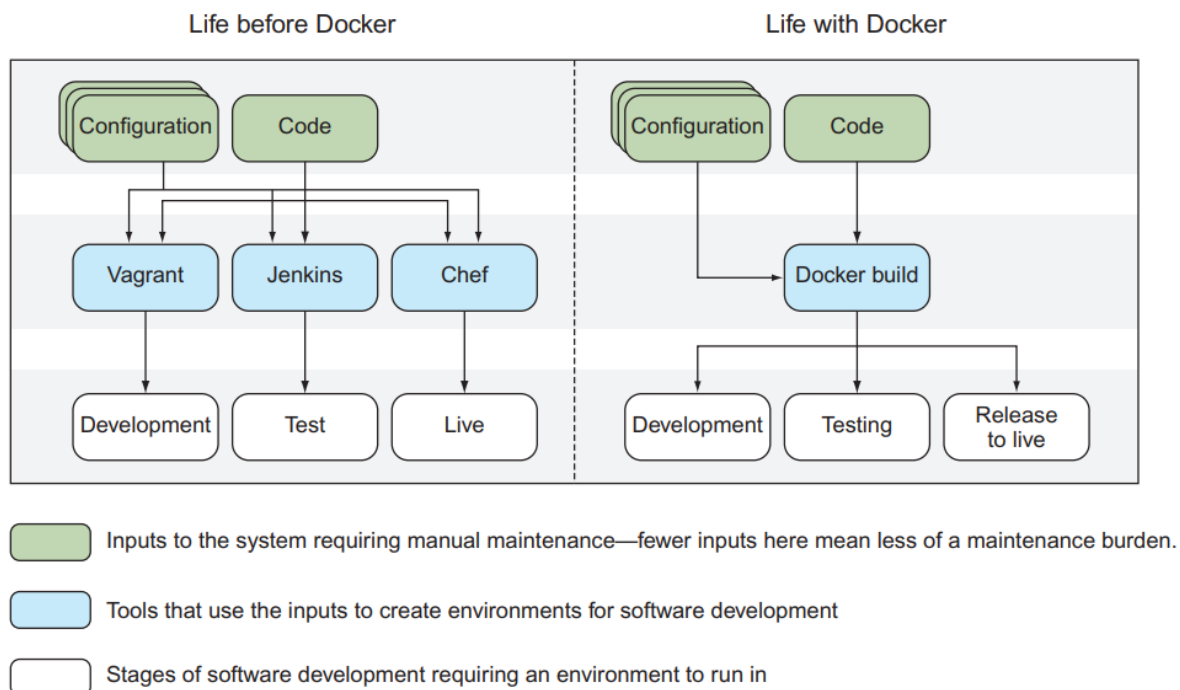
### 3.3 Docker

Docker je platforma, která umožňuje kompletovat, doručit a spustit aplikaci kdekoliv. Odtud také metafora s klasickým lodním dokem na nejrůznější zboží, které putuje v kontejnerech po celém světě. Docker vznikl a žije z poptávky finálních kroků a tj. nasazení a distribuce, kde bez Dockeru bylo nutné psát pro každé prostředí vlastní konfigurační soubory, mít správné verze apod. To vedlo ke zvýšené frustraci, časové i finanční náročnosti apod. (viz Obrázek 8) [9].

---

<sup>7</sup>Koncept pro přihlášení skrze jeden účet, např. od Google, GitHub apod.

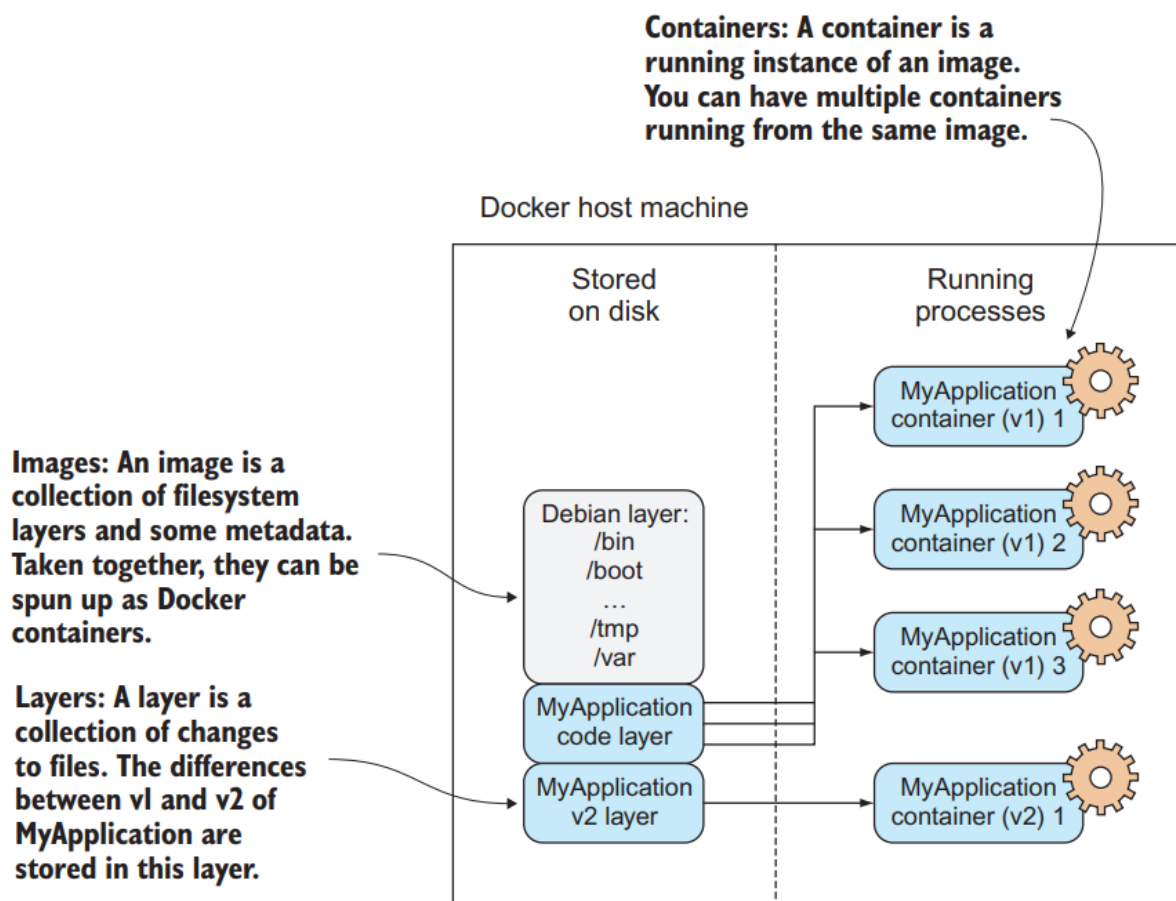
<sup>8</sup>Dotazy do databáze pro další data jsou posílány až když jsou doopravdy potřeba.



Obrázek 8: Jak Docker usnadnil nasazení [9]

Mezi klíčové koncepty patří *images*, *containers* a *layers*, kde *containers* představují běžící systémy definovány *images*, které jsou tvořeny z jedné nebo více *layers* a metadaty pro Docker (viz Obrázek 9) [9].

Docker takto odbourává především nepříjemné překážky, které nemusí ani nutně souviset s posledním finálním krokem vývoje softwaru. Usnadňuje také jeho testování a samotný vývoj. Podobným principem si vývojář na svém lokálním systému může nechat běžet několik nástrojů, které se sebou nutně nemusí kolidovat. Naopak spolu mohou tvořit *networks* a komunikovat. Nepotřebné lze pak snadno smazat bez nepříjemných pozůstatků na hostitelském operačním systému.



Obrázek 9: Základní koncept Dockeru [9]

Základní operace Dockeru jsou popsány v Tabulce 2. Modelovým případem užití může být potřeba spustit lokálně databázi pro účely testování. Můžeme postupovat klasicky přes oficiální stránky např. MySQL a složitě stahovat a instalovat potřebný software přímo na náš počítač a to samé udělat pro PostgreSQL apod. Nebo si můžeme na našem PC zprovoznit Docker. Pomocí *docker pull mysql* si stáhnout image v nejnovější verzi z Docker Hub<sup>9</sup>. Poté pomocí *docker run mysql* ho spustit a je vlastně hotovo. Veškeré další operace se provádějí v rámci kontejneru i skrze software, který fyzicky běží mimo daný kontejner. Smazání takové databáze je tak snadné jako zastavení a odstranění kontejneru.

Dalším modelovým případem užití může být vytvoření vlastního Docker *image*. Pro takový případ existuje více možností. Mezi nejčastější patří Dockerfile, který obsahuje serii příkazů, který se mají provést. Skrze *docker build* se tyto příkazy provedou. Nebo přes *docker commit* na již běžícím kontejneru. Vytvořený *image* pak lze jednoduše označit

<sup>9</sup>Výchozí repozitář z kterého se stahují Docker images.

štítkem a je možné ho nahrát do repozitáře. Kde si obdobně jako databázi MySQL může jiný uživatel vaši image stáhnout a spustit.

Tabulka 2: Základní příkazy Dockeru [9]

příkaz	účel
docker build	Kompletování Docker <i>image</i>
docker run	Spuštění Docker <i>image</i> jako <i>container</i>
docker exec	Spustí nový proces v běžícím <i>container</i>
docker commit	Vytvořit Docker <i>image</i> jako z běžícího <i>container</i>
docker tag	Přidat štítek pro Docker <i>image</i>
docker pull	Stažení Docker <i>image</i> z repozitáře
docker push	Nahrání Docker <i>image</i> do repozitáře

## 3.4 Služby

Mimo již popsané či vestavěné služby v projektu používám několik služeb pro naplnění požadavků.

### 3.4.1 ARES API

Vláda České Republiky provozuje informační systém, který zpřístupňuje veřejné údaje o ekonomických subjektech veřejné správy [10]. V rámci projektu přes poskytované REST API doplňuji dostupné informace na základě vyplněného IČO.

### 3.4.2 Amazon S3

V rámci projektu je požadavek pro ukládání souborů do Amazon S3 (Simple Storage Service). Tj. jeden z prvních produktů nabízených v ekosystému AWS. Jedná se o objektové úložiště, které je škálovatelné, dostupné, zabezpečené a výkonné [1].

Data jsou ukládána do tzv. *buckets* což jsou logické kontejnery s unikátním názvem pro oblast. Přístup lze řídit pomocí politiky IAM (Identity and Access Management), ACLs (Access Control Lists) a Bucket Policies [1].

Samotný přenos lze šifrovat při přenosu (SSL/TLS) i v klidovém stavu na straně serveru skrze AWS KMS, S3-Managed Keys nebo zákaznických klíčů [1].

Mezi klíčové funkce patří verzování, CRUD, archivace, správa práv, automatická replikace a spouštění akcí na základě událostí [1].

### **3.4.3 Gmail API**

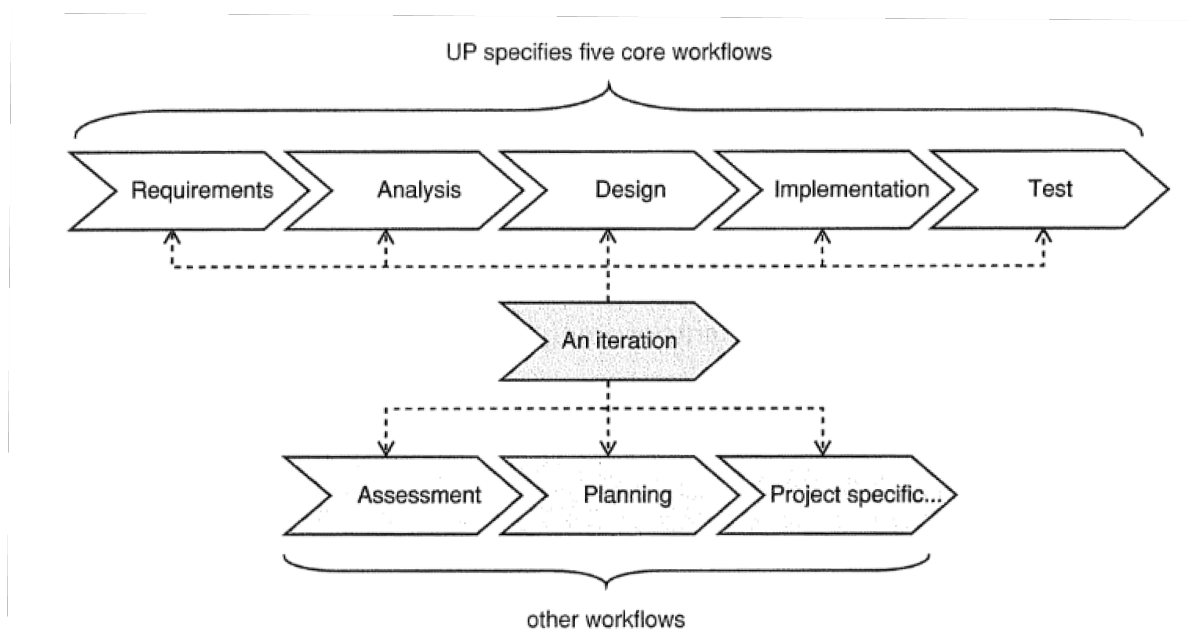
V projektu je požadavek pro zasílání emailů. Tento požadavek lze obsloužit skrze Gmail API. Jedná se o RESTful API, kterým lze provádět operace v připojené schránce [4]:

- zobrazovat emaily v režimu pouze ke čtení, indexovat a provádět zálohy,
- automatické i programové zasílání zpráv,
- migrace emailových účtů,
- organizační funkce včetně filtrování a řazení zpráv,
- podepisování zpráv podpisem organizace.



## 4 NÁVRH APLIKACE

Tvorba aplikace je inspirována metodikou Unified Process [2]. Metodika není striktně plněna, avšak po myšlenkové lince kopíruje průběh mé tvorby. Je tak činěno na základě mého studia, kde právě UP byla představena jako kvalitní must, který v dnešní době jsme schopni doplnit např. agilními technikami vývoje a docílit tak kvalitního a rychlého vývoje.



Obrázek 10: Metodika UP [2]

V této kapitole jsou popsány výstupy sběru požadavků, analýzy a návrhu uživatelského rozhraní.

### 4.1 Požadavky

Ze zadání a osobního hovoru s zadavatelem jsou definovány následující funkční požadavky, tj. co by systém měl nabídnout (viz Tabulka 3) a nefunkční požadavky (viz Tabulka 4), tedy jaké konkrétní vlastnosti či omezení systém má.

Tabulka 3: Funkční požadavky

#	Funkční požadavek
1	Systém musí být zabezpečený skrze přihlášení.
2	Systém musí nabízet mechanismus pro obnovu hesla.
3	Systém by měl pracovat s úrovněmi oprávnění pro více uživatelů.
4	Systém musí spravovat zákaznické údaje vč. adres.
5	Systém by měl využívat ARES API pro hledání údajů o zákazníkovi skrze IČO.
6	Systém musí spravovat zakázky, které mají svého zákazníka, produkty, stav s historií změn, poznámky, více náklady a přílohy.
7	Systém musí spravovat nabízené produkty, respektive jejich komponenty ze skladu.
8	Systém musí spravovat sklad včetně počtu zásob.
9	Systém musí upozornit uživatele pokud počet skladových zásob klesne pod určenou hranici.
10	Systém by měl upozornit uživatele pokud vytváří zakázku navzdory nízkému stavu potřebných skladových zásob.
11	Systém by měl automaticky aktualizovat stav skladových zásob při správě zakázek.

Tabulka 4: Nefunkční požadavky

#	Nefunkční požadavek
1	Systém zasílá token pro obnovení hesla skrze email.
2	Systém získanou odpověď z ARES automaticky plní do příslušných polí ve formuláři.
3	Systém při změně stavu zakázky, ukládá i informaci, kdy ke změně došlo.
4	Systém zobrazuje poznámky s jménem autora, datumem a poznámkou samotnou.
5	Systém používá pro ukládání příloh předpřipravený souborový server Amazon S3 cloud z rodiny Amazon Web Service.
6	Systém používá pro relační databázi předpřipravený MySQL databázi v AWS.
7	Systém dovoluje vytvořit zakázku navzdory nedostatku skladových zásob, ale takovou skutečnost signalizuje a upozorňuje na ni.
8	Systém při klesnutí skladových zásob pod stanovenou hranici může notifikovat vybraného správce přes mail.
9	Systém na výpisu skladových zásob graficky resp. barevně signalizuje množství skladových zásob. Např. červená pro množství skladových zásob $x < 0$ , zelená pro $x >$ stanovená hranice apod.
10	Systém je napsán s využitím frameworku JAVA Spring a technologie Vaadin.
11	Systém je otestován i v reálném prostředí.
12	Systém je nasazen na webový server pomocí kontejnerového systému Docker.
13	Systém je responsivní.

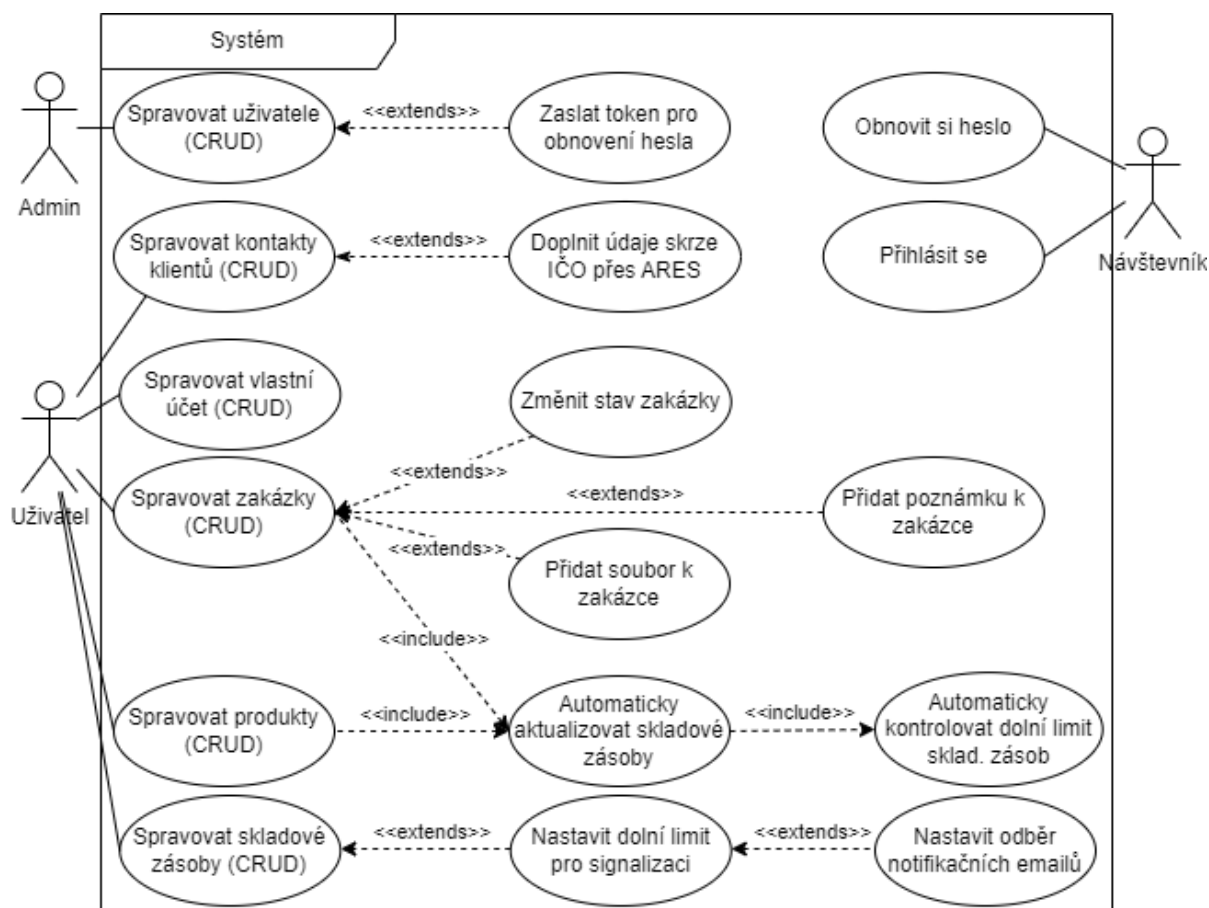
## 4.2 Use Cases

Z definovaných požadavků je namodelován Use Case diagram v UML (viz Obrázek 11).

Jelikož zadavatel reprezentuje malou firmu o jednotkách zaměstnanců, kde zaměstnanec plní více rolí, jsou modelovány pouze 3 základní aktéři:

- admin neboli správce uživatelů,
- uživatel abstrakce přihlášeného zaměstnance (skladník, obchodní zástupce, konstruktér, apod.),
- návštěvník tj. nepřihlášený zaměstnanec.

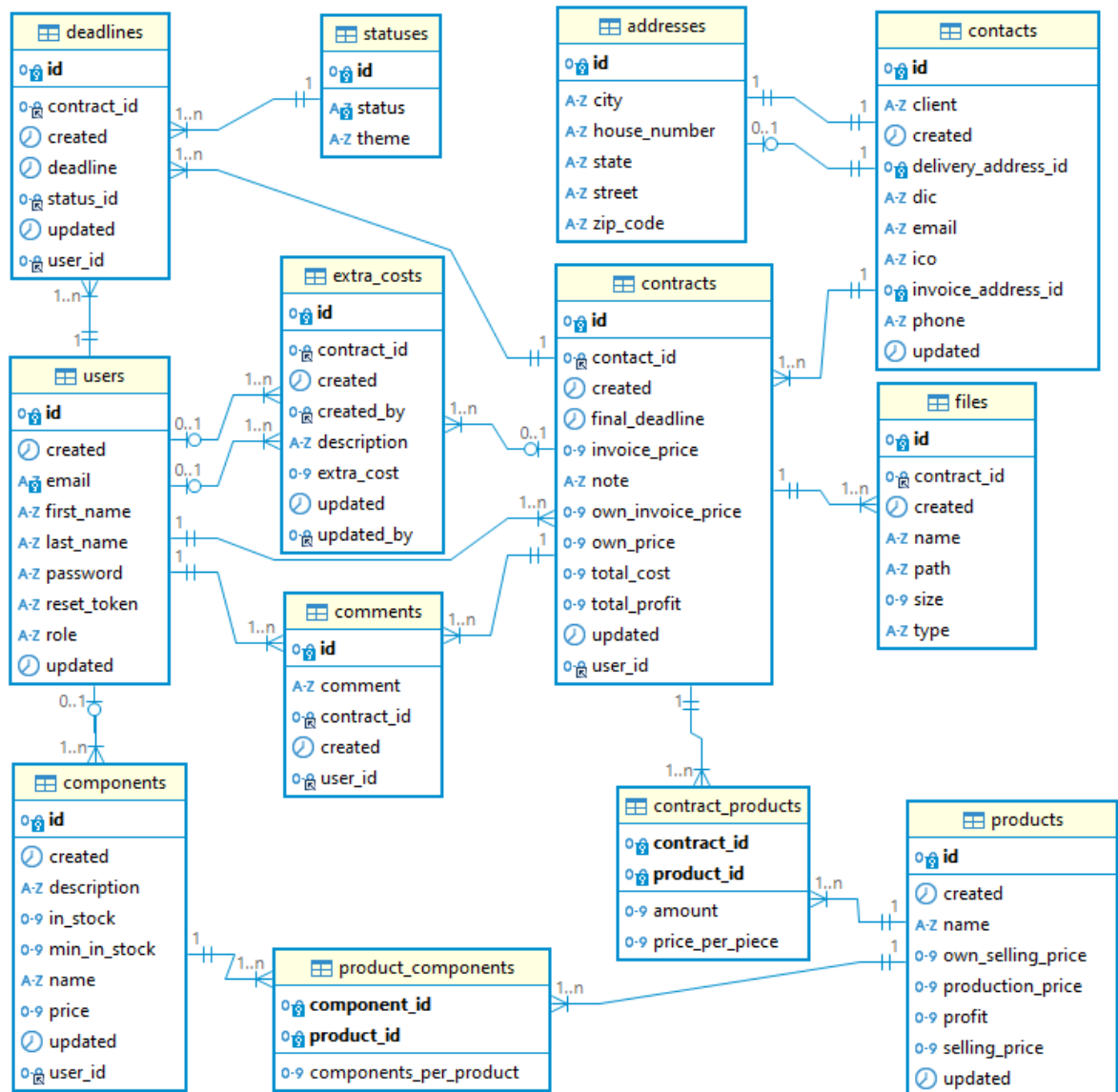
Pro samotné modelování jsou použity pokročilé techniky: extends a include. Extends znamená možné rozšíření. Například pokud uživatel spravuje zakázky, má možnost připojit přílohu, změnit stav nebo přidat poznámku, není však povinný tak činit. Include rozšiřuje UC o povinné rozšíření [2]. Pokud např. uživatel spravuje zakázky, automaticky se vždy aktualizují skladové zásoby a vždy se automaticky kontroluje popř. signalizuje stav zásob pod určenou hranicí.



Obrázek 11: Use Case diagram

## 4.3 Databázový model

Z UC reprezentace požadavků byla navržena databáze (viz Obrázek 12).



Obrázek 12: Celkový databázový model

Systém potřebuje 13 tabulek:

- *users* jsou data o uživateli, přihlášení probíhá skrze email a heslo, oprávnění je enum, obnovení hesla probíhá skrze *reset\_token*,
- *comments* jsou komentáře od uživatelů přiřazené k zakázkám,
- *deadlines* jsou dílčí termíny zakázek, kde každý termín má svůj stav, termíny přiřazují uživatelé,

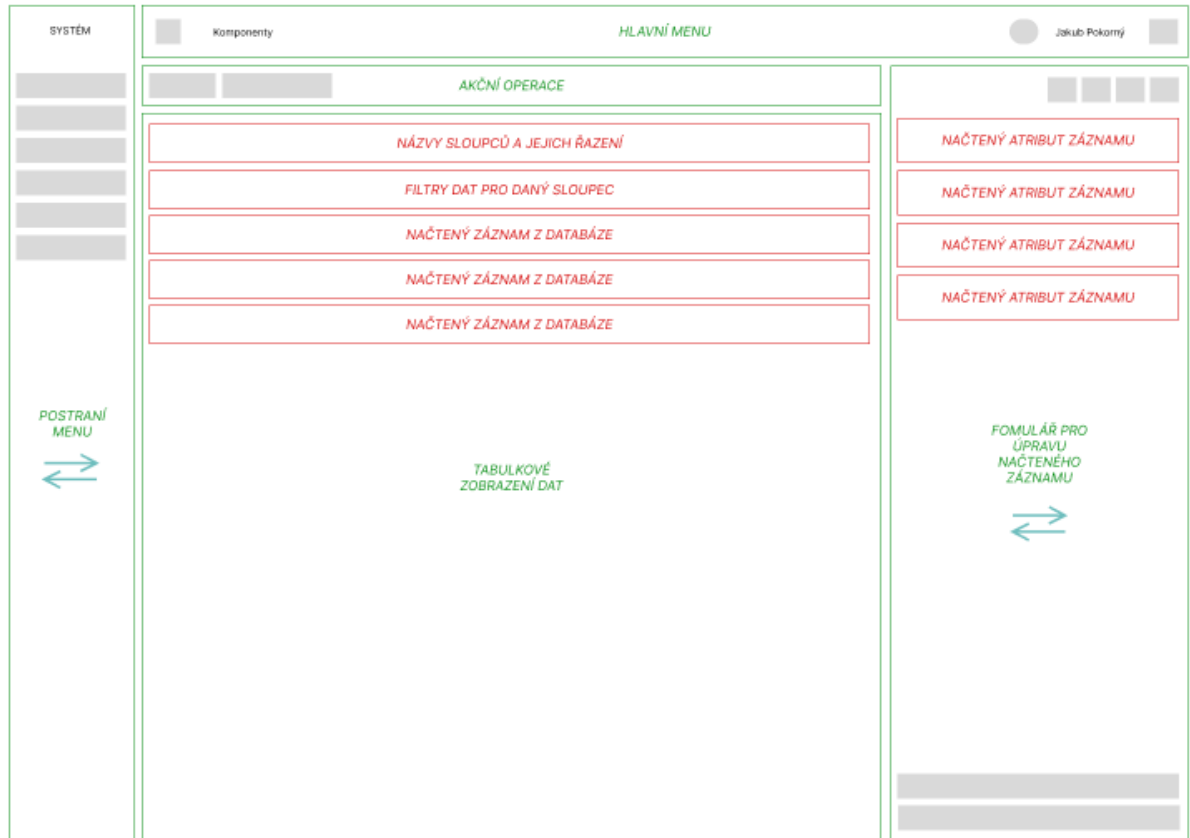
- *statuses* jsou stavy termínu, uživatel je schopen si nadefinovat vlastní stavy s grafickými variacemi dle *theme*,
- *contracts* jsou zakázky, které mají povinně klienta, produkty, termíny a nepovinně poznámky, soubory a více náklady,
- *extra\_costs* jsou více náklady přiřazené k zakázce,
- *contacts* jsou kontakty klientů, má povinně fakturační adresu a nepovinně doručovací adresu,
- *addresses* jsou adresy klientů,
- *files* jsou metadata o souborech v AWS,
- *contract\_products* je spojovací tabulka mezi zakázkami a produkty, zakázka se skládá z produktů,
- *products* jsou data o nabízených produktech,
- *product\_components* je spojovací tabulka mezi produkty a komponenty, produkt se skládá z komponent,
- *components* jsou data skladu komponent, případná signalizace může proběhnout na uživatele.

## 4.4 Uživatelské rozhraní

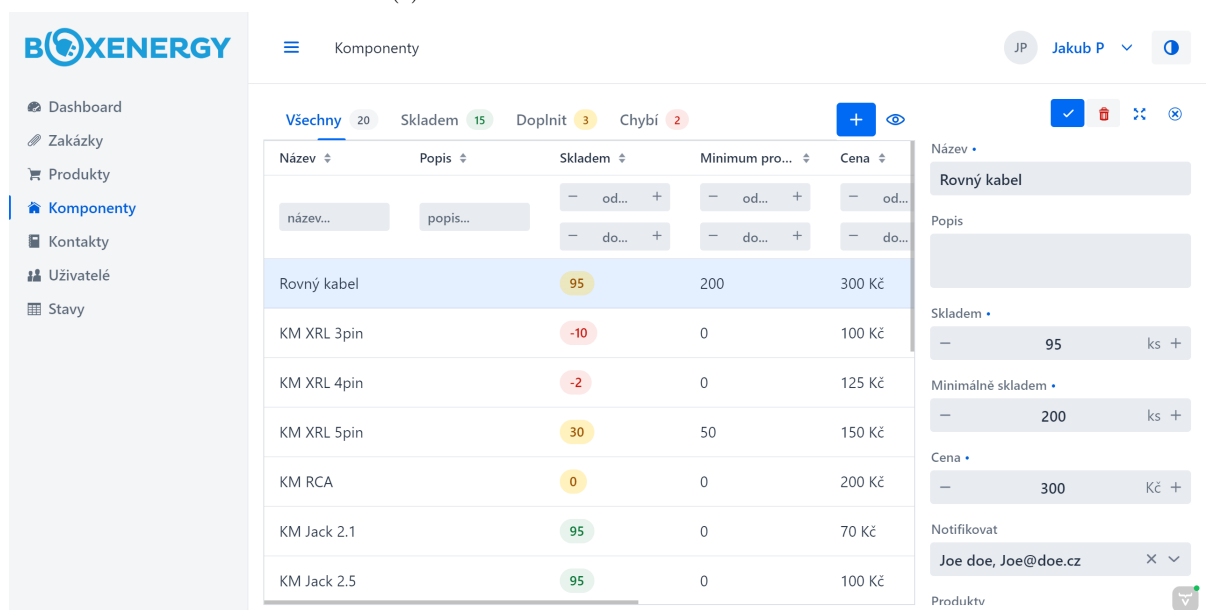
Hlavní gró systému představuje správa, tj. CRUD operace do databáze. Na základě osobního hovoru je předpoklad, že systém bude především využit pro čtení a editaci objednávek a skladových komponent. Proto je kladen důraz právě na přehledné zobrazení s minimem rozptýlení (viz Obrázek 13a). Proto:

- *HLAVNÍ MENU* obsahuje pouze tlačítko pro zobrazení postranního menu, přihlášeného uživatele a přepínač mezi tmavým a světlým režimem,
- *POSTRANNÍ MENU* obsahuje jen název či logo společnosti a položky menu, je zasouvací,
- *FORMULÁŘ PRO ÚPRAVU NAČTENÉHO ZÁZNAMU* obsahuje načtenou n-tici pro úpravy a akční tlačítka pro uložení, smazání, schování formuláře nebo naopak roztažení na úkor tabulkového zobrazení,
- *AKČNÍ OPERACE* obsahují tlačítka, např. pro přidání nového záznamu, export, schování sloupců v tabulce, filtry.

- *TABULKOVÉ ZOBRAZENÍ DAT* obsahuje v hlavičce názvy s řadiči, pod nimi jsou filtry, a pak už jednotlivé jednoduché nebo komponentní vykreslení dat, při kliknutí na záznam se vysune formulář s načteným záznamem na úkor tabulky.



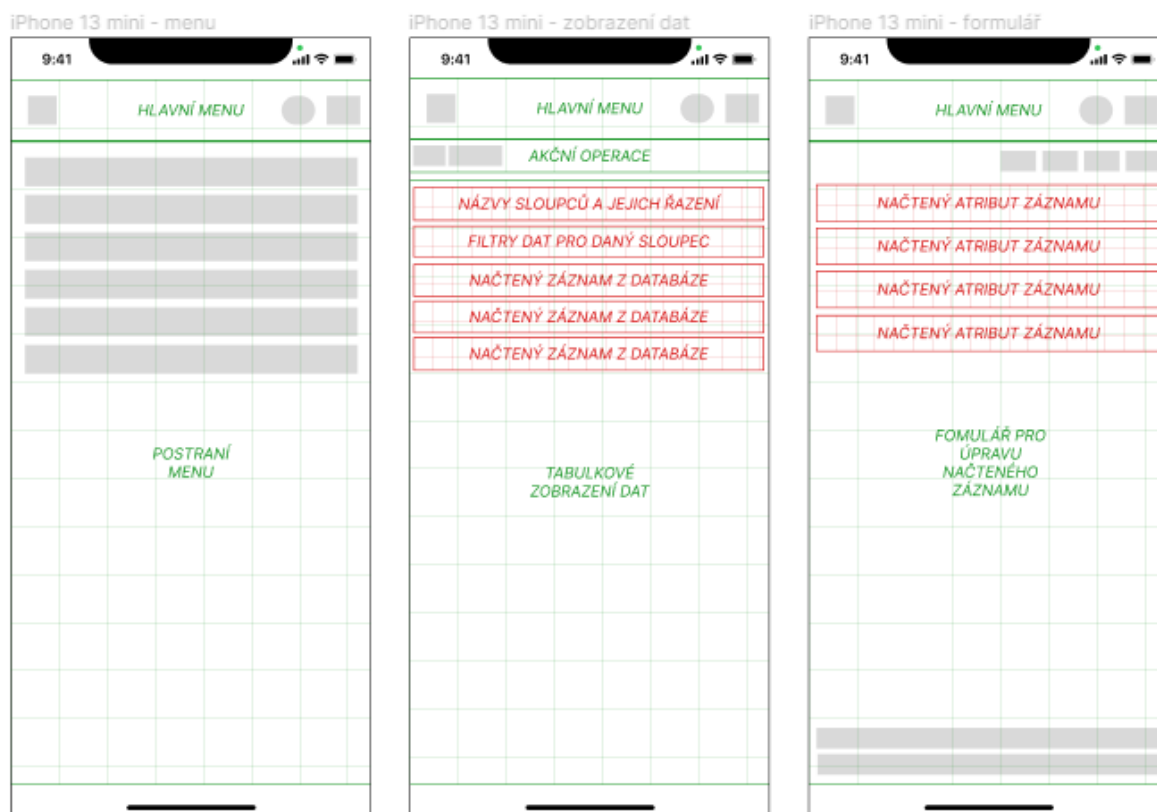
(a) Návrh uživatelského rozhraní 1920x1080



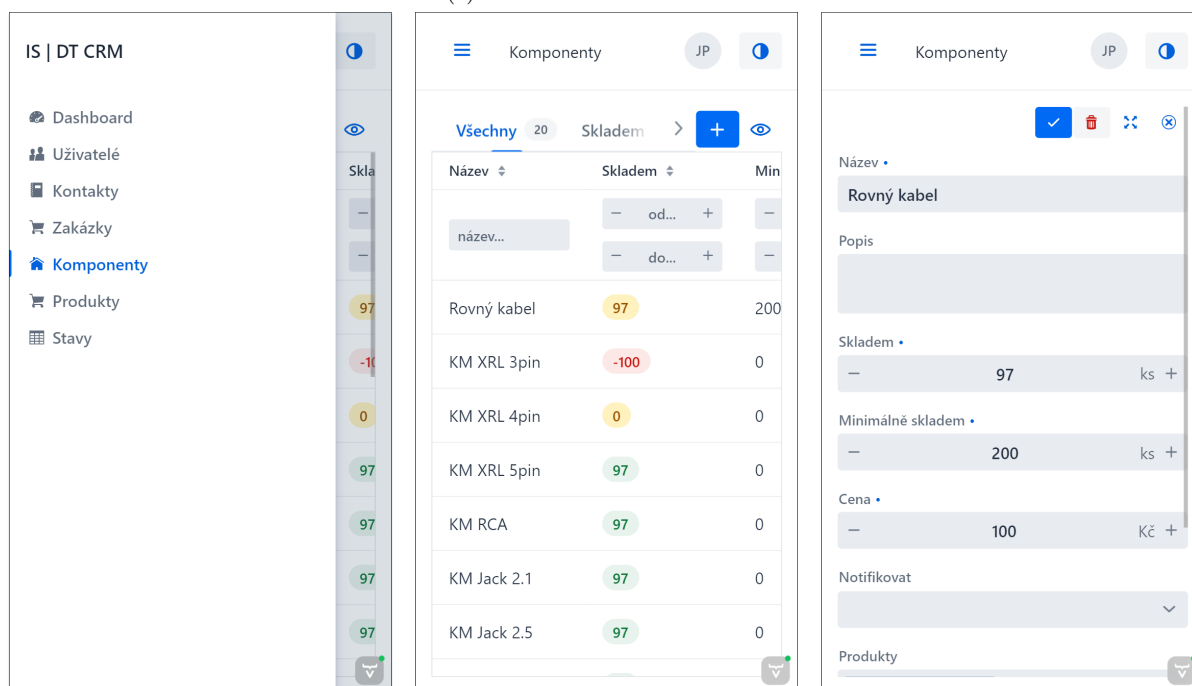
(b) Ukázka implementace uživatelského rozhraní 1280x650

Obrázek 13: UI návrh

Prosté rozložení bez modálních oken usnadňuje responsivní zobrazení na menší obrazovky, kde není prostor zobrazovat více informací najednou. To znamená, že malé mobilní obrazovky plně zaplňuje buď menu, tabulka nebo formulář (viz Obrázek 14a).



(a) Návrh UI na iPhone 13 mini



(b) Ukázka implementace uživatelského rozhraní na iPhone SE

Obrázek 14: UI návrh pro mobilní zařízení



## 5 IMPLEMENTACE APLIKACE

V této kapitole jsou popsány implementační kroky k vytvoření webové aplikace, která po vzoru ERP kombinuje několik byznys procesů a vytváří pro ně technologickou platformu.

### 5.1 Příprava vývojového prostředí

Vaadin na svých stránkách nabízí projektový akcelérátor skrze který lze vytvořit rychle šablonu projektu. Nástroj je poměrně propracovaný a lze si nastavit barevné schéma a připravit si kompletně UI. Pro tento krok i projekt je důležitý hlavně `pom.xml` se závislostmi pro Vaadin Flow, nikoliv Hilla. Generovaný projekt je kompletován pomocí Maven, počítá se rovnou se Spring Boot, v nástroji nebyla možnost MySQL, ale to nevadí, to lze upravit ručně právě v `pom.xml`. Front-end lze kompletovat pomocí npm nebo pnpm. Nástroj dokáže generovat i výchozí `.gitignore` pro Git repozitář a `DockerFile` pro Docker.

Po stažení stačí už jen otevřít v oblíbeném IDE. Pro projekt byl použit IntelliJ IDE v ultimate edition. Je třeba doplnit závislosti:

- pro MySQL jako produkční databázi,
- pro zasílání emailu, v mém případě stačí `spring-boot-starter-mail`,
- pro rychlejší vývoj Lombok, kde skrze anotace se šetří čas při vytváření tříd,
- pro parsování REST odpovědí do tříd například `gson` od Google,
- AWS závislosti: `s3`, `auth` a `regions`,
- nepovinně v rámci generování dat lze použít `javafaker`,
- nepovinně lze také pro generování statického modelu pro použití v kódu `hibernate-jpamodelgen`.

Dále je rozumné své úsilí verzovat např. V GitHub. V souvislosti s GitHub je moudré neposílat konfigurační soubory s reálnou konfigurací v podobě hesel, tokenů apod., protože např. V rámci AWS by tyto přístupové údaje byly automaticky deaktivovány. Proto je dobré tyto soubory dopsat do `.gitignore` a zároveň do cloudu zasílat soubory s přidanou koncovkou `dist`, kde jsou citlivé konfigurace vynechány. Výšeč výsledného souboru `application.dist.yml`, který se dostane do repozitáře, viz Obrázek 15. Vývojáři je pak jasné, že je tuto konfiguraci nutné dopsat.

```

30  jwt:
31    auth:
32      secret: "to-be-replace"
33  aws:
34    region: "to-be-replace"
35    access-key-id: "to-be-replace"
36    secret-access-key: "to-be-replace"
37  s3:
38    buckets:
39      contract: "to-be-replace"
40

```

Obrázek 15: Ukázka ze souboru *application.dist.yml*

V rámci vývoje lze vytvořit profily. Každý profil může mít svůj konfigurační soubor: *application.dev.yml* a *application.prod.yml*. V každém z nich je třeba doplnit konfiguraci pro databázi. Pro tento projekt je produkční databáze již k dispozici, jedná se tedy jen o doplnění. Pro vývoj je vytvořena v Dockeru vlastní MySQL databáze. IntelliJ nabízí plugin pro napojení databází přímo z vývojového prostředí, pro rychlejší kontrolu databázové vrstvy. V potaz přichází i plnohodnotné nástroje jako je DBeaver. Minimálně pro vývoj je dobré vypisovat Hibernate sql dotazy. V obou případech je potřeba nastavit také gmail API, AWS a určit maximální velikosti requestů a souborů pro S3 v rámci servletu. Také nezapomínejme na určení portu celé aplikace.

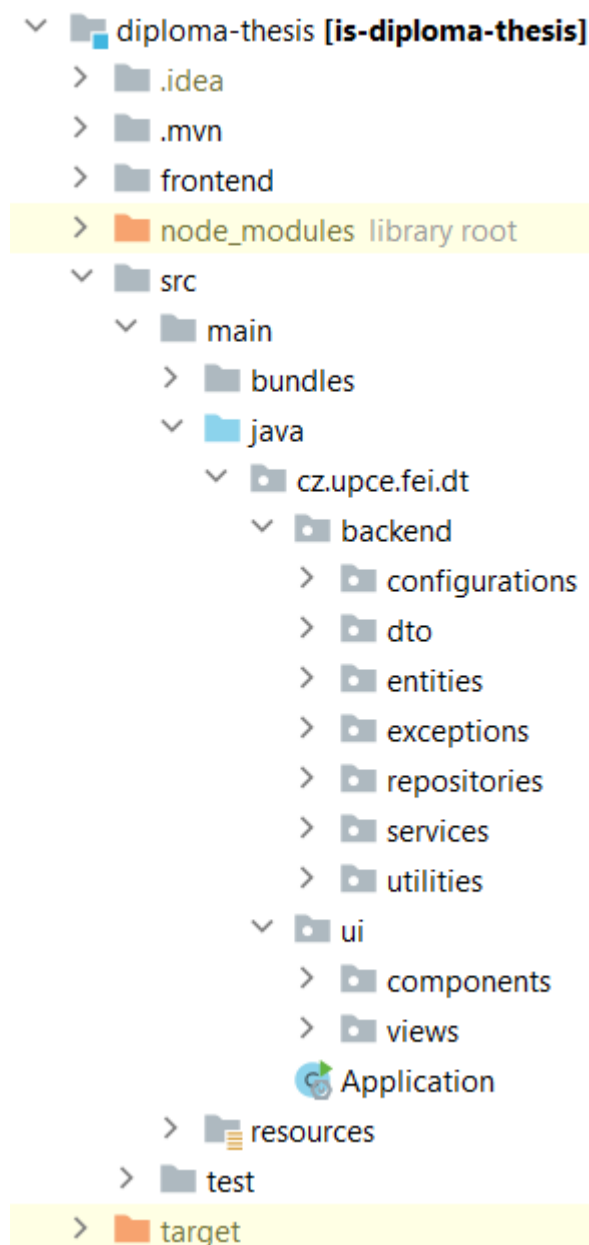
Následuje implementace tzv. Bean v rámci Spring Boot pro bezpečnost, AWS a Email.

## 5.2 Back-end

Implementace back-end části je zjednodušena o *controller layer*, kde by byly definovány koncové body tzv. *endpoints* pro prohlížeč. Následně by prohlížeč doplnil data a jednalo by se tak o CSR. Avšak s Vaadin Flow tato vrstva není nutná, protože jak již bylo zmíněno, Vaadin Flow renderuje na straně serveru (SSR) a přenos dat si obstarává také sám. Veškerou logiku tak lze obsloužit v rámci Spring Boot ve vrstvě pro persistenci dat tzv. *repository layer* a byznys logiku v *service layer*. Ono napojení s front-end bude probíhat na

přímo, kde do stránky (... *View.java*) se skrze DI napojí potřebná služba (... *Service.java*). Obdobně v službách, resp. *service layer* jsou napojeny repozitáře (... *Repository.java*).

I přesto, že back-end a front-end se v projektu silně prolínají ve struktuře projektu je snaha o jejich pomyslné oddělení (Obrázek 16). Ve složce backend tedy konfigurace Spring Boot, model, transformační třídy, obsluha výjimek, databázová a byznys vrstva a obecné nástroje. V ui složce komponenty pro stránky (views).



Obrázek 16: Struktura projektu webové aplikace

### 5.2.1 Datový model

Ve složce *entities* definuji model, který následně skrze JPA je vytvořen v databázi. JPA může pracovat v několika režimech dle nastavené konfigurace *spring.jpa.hibernate.ddl-auto* [23]:

- *create*: vytvoří schéma a smaže předchozí data.
- *create-drop*: vytvoří schéma a zničí jej na konci sezení (session).
- *update*: aktualizuje schéma pokud je to třeba.
- *validate*: Validuje schéma a nečiní změny.
- *none*: deaktivuje DDL <sup>10</sup> funkci Hibernate.

pro produkci je aktivní režim validace, pro vývoj po většinu času režim update a pro testování create.

Základním konceptem Spring Boot jsou anotace (viz Obrázek 17).



```
16
17 @Getter @Setter @ToString @Builder  Pokorný
18 @AllArgsConstructor @NoArgsConstructor
19 @Entity
20 @Table(name = "users")
21 public class User implements UserDetails {
22
23     @Id
24     @GeneratedValue(strategy = GenerationType.IDENTITY)
25     private Long id;
26
27     @Column(length = 100, nullable = false, unique = true)
28     private String email;
29
30     @Column(nullable = false)
31     private String password;
```

Obrázek 17: Ukázka použití anotací

<sup>10</sup>V kontextu SQL se jedná o jazyk pro definici databázových objektů jako jsou tabulky, indexy a uživatelé.

Lombok totožným principem přidává pro atributy getters, setters, toString metodu, koncept builderu, konstruktor se všemi parametry a konstruktor bez parametru čímž snižuje boilerplate code<sup>11</sup>. Anotace @Entity označuje jméno třídy pro Jakarta Persistence query Language (JPQL), kterým je možné se dotazovat do databáze obdobným způsobem jako SQL. S rozdílem, že IDE je schopno JPQL kontrolovat i syntakticky. @Table je opět pro JPA jako označení tabulky v databázi. Obdobně v rámci třídy označujeme konkrétní sloupce (@Column) s omezeními a případnou validací (@NotBlank, @Size apod.). Na obrázku 17 tyto validace nejsou, protože tento nezbytný krok provádím v rámci Vaadin Flow přímo ve formulářích. Každá tabulka musí mít primární klíč (@Id). Zároveň na obrázku 17 je patrná implementace UserDetails v rámci Spring Boot Security. Jinými slovy třída User představuje uloženého uživatele s oprávněním [23].

K relačním databázím neodmyslitelně patří vazby. JPA pro vazby nabízí anotace:

- @OneToOne: propojení 1:1
- @OneToMany: propojení 1:n, kde je nutno mapovat parametrem mappedBy.
- @ManyToOne: propojení n:1, kde se jedná o obousměrné spojení.
- @ManyToMany: propojení m:n, kde si JPA vytvoří propojovací tabulku a rozdělí tak vazbu na dvě. V projektu jsou tyto vazby rozděleny ručně.

U vazeb je zároveň nutné počítat s režimem v kterém se mají plnit data (fetch). Ve výchozím nastavení se jedná o *FetchType.LAZY*, kde jsou data dotazována až když jsou potřeba. Opakem je *FetchType.EAGER*, kde se jedná o strategii, kdy data jsou plněna do struktur předem. Dále je potřeba věnovat pozornost parametru *cascade*, kde nastavujeme, jak se má vazba chovat vzhledem k napojeným entitám. Např. změníme li v rámci entity *contacts* entitu *address* přiřazenou k dané entitě. Má se *address* uložit také? Obdobně lze nastavit chování při vzniku sirotek (*orphanRemoval=true/false*) (viz Obrázek 18) [23].

---

<sup>11</sup>Části kódu, které jsou často opakovány s malou variancí.

```

65     @OneToMany(mappedBy = "component", cascade = CascadeType.MERGE,
66                orphanRemoval = true)
67     @ToString.Exclude
68     @JsonIgnore
69     @Builder.Default
70     private Set<ProductComponent> productComponents = new HashSet<>();
71
72     @ManyToOne
73     @JoinColumn(name = "user_id")
74     private User user;

```

Obrázek 18: Ukázka vazeb v entitách

V souvislostech s vazby se často vyskytují chyby *LazyInitializationException*, která právě souvisí s nenačtenými daty v kolekci, i přestože s nimi chceme pracovat. Problém lze často vyřešit tím, že příslušnou kolekci načteme předem. Tedy změníme *fetch=lazy* na *fetch=eager*. V takovém řešení se může objevit problém se slabším výkonem, kde načítáme větší množství nepotřebných dat a nebo 1+n query problém, kde 1 dotaz na n dat vyvolá n dalších dotazů. Proto je doporučeno si obecně na 1+n dotazy dát pozor a tam kde není potřeba *eager fetch*, tam se mu vyhnout. V projektu jsou tyto situace řešeny pomocí *@NamedEntityGraph* např. viz obrázek 19, kde při načítání Komponent do tabulky je potřeba předem načíst informace o napojených produktech a zároveň je potřeba načíst i konkrétní data o produktech, tz. dotaz přes 3 tabulky. V Takovém případě tyto anotace s parametry JPA značí, jak postupovat bez nutnosti psát JPQL nebo SQL. Obdobně je tento specifický dotaz použit jen v případě pokud je volán. Tedy pokud by jiný případ nepotřeboval takto rozsáhlý dotaz, aplikace je stále schopna pracovat ve výchozím stavu, tedy načítání *productComponent* v režimu *Lazy*. *@JsonIgnore* a *@ToString.Exclude* viz obrázek 18 zajistí, že se JPA nezacyklí při *@NamedEntityGraph*.

```

20 @NamedEntityGraph(
21     name = "Component.eagerlyFetchProduct",
22     attributeNodes = {
23         @NamedAttributeNode("user"),
24         @NamedAttributeNode(value = "productComponents",
25                               subgraph = "productComponentSubgraph")
26     },
27     subgraphs = {
28         @NamedSubgraph(
29             name = "productComponentSubgraph",
30             attributeNodes = {
31                 @NamedAttributeNode("product")
32             }
33         )
34     }
35 )

```

Obrázek 19: Ukázka *NamedEntityGraph*

## 5.2.2 Databázová vrstva

Repositáře jsou rozhraní (*interface*) označené `@Repository`, které rozšiřují *JpaRepository*`<Entity, IdType>`. To přináší mnoho vestavěných metod s kterými si lze i vystačit. *JpaSpecificationExecutor*`<Entity>` slouží pro filtrování dat pomocí *Specification*`<Entity>` (viz Obrázek 20).

```

19 @Repository 4 usages Pokorný *
20 public interface ComponentRepository extends JpaRepository<Component, Long>, JpaSpecificationExecutor<Component> {
21     @EntityGraph(value = "Component.eagerlyFetchProduct") Pokorný
22     @NonNull
23     List<Component> findAll(@Nullable Specification<Component> specification, @NonNull Sort sort);
24
25     @Query(value = "select id, name, price from components where lower(name) like lower(concat('%', :searchTerm, '%'))",
26             nativeQuery = true)
27     @NonNull
28     Page<IComponent> findAllByName(@NonNull Pageable pageable, @Param("searchTerm") String searchTerm);
29
30     @Modifying 3 usages Pokorný
31     @Transactional
32     @Query(value = "update Component c set c.inStock = :inStock where c.id = :id")
33     void updateAmountById(@NonNull @Param("id") Long id, @NonNull @Param("inStock") int inStock);

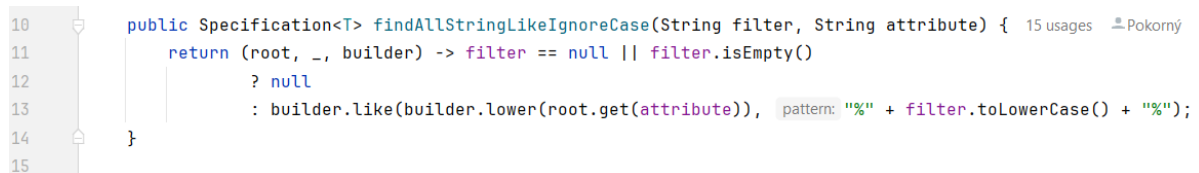
```

Obrázek 20: Ukázka implementace v databázové vrstvě

Na obrázku 20 mimo základní konstrukce lze vidět využití `@NamedEntityGraph` v kombinaci s filtrováním a řazením v metodě `findAll`. Je patrné, že většinu za nás dělá Spring

Boot. V další metodě `findAllByName` se za pomoci nativního SQL vrací stránka která je promítnuta do transformačního rozhraní `IComponent`. Spring Boot nabízí několik možností jak provádět ORM. Můžeme používat DTO v podobě *record*, nebo tříd. Nebo pouhé rozhraní jako v tomto případě. Samotné stránkování předáváme ve rozhraní *Pageable*, které může zapouzdřit i řazení (rozhraní *Sort*) [23]. V další metodě `UpdateAmountById` je vidět použití JPQL v kombinaci s transakční (`@Transactional`) aktualizací (`@Modifying`) dat na základě identifikátoru. Pro výchozí dotazy, které nabízí rozhraní, které rozšiřujeme, se postupuje dle názvu metody. Např. `findAll` bez `@NamedEntityGraph` a `@Query` vrátí do požadovaného typu vše i dle parametrů metody. V tomto ohledu je užitečným nástrojem JPABuddy nebo našeptávač od IDE.

Specifikace, neboli nástroj pro specifické dotazy, např. filtrování dat, funguje na principu Criteria Query, kde je nám poskytnut mechanismus, který obchází složité i repetitivní vytváření SQL dotazů (viz Obrázek 21). Zde i přichází vhod statický model, kde pak v rámci kódu můžeme tento model využívat pro odkazování na sloupce a mít tak i syntaktickou kontrolu (`Component_.NAME`) na místo ručního vypisování potřebných názvů v uvozovkách ("*name*").



```

10 public Specification<T> findAllStringLikeIgnoreCase(String filter, String attribute) {
11     return (root, _, builder) -> filter == null || filter.isEmpty()
12         ? null
13         : builder.like(builder.lower(root.get(attribute)), pattern: "%" + filter.toLowerCase() + "%");
14 }
15

```

Obrázek 21: Ukázka filtrování pomocí Specification

Každá tabulka neboli entita má svůj repozitář různých složitostí a kombinací. Cílem je vždy poskytnout takové metody, které lze využít v byznys vrstvě.

### 5.2.3 Byznys vrstva

Byznys třídy jsou označovány anotací `@Service` a skrze DI mají vloženy potřebné repozitáře a jiné byznys třídy (viz Obrázek 22). DI je vložen skrze Lombok anotaci `@AllArgsConstructor`, tedy konstruktorem. Bez anotací `@Service`, `@Repository`, `@Component`, `@Bean` by DI nefungovalo. Konkrétně v ukázce 22 je vidět implementace metody z rozhraní od Vaadin Flow, které slouží pro naplnění dat do komponenty Grid. Na řádce 38 je připravena specifikace pro filtrování dat s pomocí filtrovací třídy (*ComponentFilter*),



která je k tomu určena a bude se jednat o filtrování na úrovni databáze. Následuje fetch z databáze, kde se bere v potaz i řazení. Pokud bychom v této fázi chtěli použít i stránkování od JPA pomocí rozhraní *Pageable*, bude z neobjasněného důvodu ignorováno. Bylo tedy postupováno dle Vaadin dokumentace, kde stránkování je provedeno stylem, které je vidno na řádku 46 [?]. Na řádku 43 je provedena druhá filtrace získaných dat, protože odbavení v první filtraci by vyžadovala sofistikované Criteria Query.

```
28  @Service  Pokorný *
29  @AllArgsConstructor
30  public class ComponentService extends AbstractBackEndDataProvider<Component, ComponentFilter> {
31      private final ComponentRepository componentRepository;
32      private final ProductComponentService productComponentService;
33      private final ProductService productService;
34      private final EmailService emailService;
35
36      @Override  Pokorný *
37      @ @
38      public Stream<Component> fetchFromBackEnd(Query<Component, ComponentFilter> query) {
39          Specification<Component> spec = ComponentSpec.filterBy(query.getFilter().orElse(new ComponentFilter()));
40          Stream<Component> stream = componentRepository.findAll(spec, VaadinSpringDataHelpers.toSpringDataSort(query))
41              .stream();
42
43          if (query.getFilter().isPresent()) {
44              stream = stream.filter(component -> query.getFilter().get().filter(component));
45          }
46
47          return stream.skip(query.getOffset()).limit(query.getLimit());
48      }
```

Obrázek 22: Ukázka implementace v byznys vrstvě

Tato vrstva slouží pro veškerou logiku která předchází operacím v databázi a operacím na UI. Obdobně jako @Repository je tak pro každou tabulku zpracována i @Service třída.

V této vrstvě je odbavena většina definovaných požadavků.

## 5.2.4 Výjimky

Ke každému projektu patří práce s výjimkami. Vaadin Flow pro tyto účely nabízí rozhraní *ErrorHandler*, které stačí implementovat dle dokumentace [25]. Doporučením od Spring je maximální využití předdefinovaných výjimek v rámci Spring Boot [23].

## 5.3 Front-end

Vaadin Flow nabízí knihovnu svých komponent, komponenty z komunity, které pro projekt nebyly použity a komponenty které si vývojář může přizpůsobit sám rozšířením stávajících.

Projekt si vystačil se základními a rozšiřováním základních komponent. Pro projekt byly použity i některé placené komponenty: (*Charts*, *Board*).

Vaadin nabízí vše potřebné pro vytvoření UI včetně navigování (*@Route*), oprávnění (*@PermitAll*, *@RolesAllowed*, *@AnonymousAllowed*), validace, data binding atd. Vše je relativně přehledně dokumentováno přímo na Vaadin stránkách [25].

### 5.3.1 Hlavní menu

Pro hlavní menu, které představuje hlavní rozvržení (*MainLayout*) pro většinu stránek, tedy minimálně potřebuje aktualizaci dat se postupovalo dle doporučení dokumentace. Kde počet navigačních položek je větší než pět (7: Dashboard, Uživatelé, Kontakty, Zakázky, Komponenty, Produkty, Stavby), proto zasouvací postranní menu (*Drawer*) s názvem stránky v levém horním rohu. Pro hlavní menu pak zbylo dostatek místa pro ovládací prvek postranního menu, název stránky, rozklikávací menu s přihlášeným uživatelem a přepínač světelného režimu.

Za zmínku stojí implementované řešení *VaadinWebSecurity*, kde se v konfigurační metodě *configure* nastavuje základní zabezpečení včetně nastavení stránky pro přihlášení, které nemá hlavní menu. Pokud tedy uživatel chce přistoupit na stránky za přihlášením je automaticky jako nepřihlášený uživatel přesměrován opět na přihlášení.

V rámci *MainLayout* jsou implementovány chyby nepovoleného přístupu (kód 403) a stránka nenalezena (kód 404) (viz Obrázek 23). V jednom i druhém případě se uživateli hlavní menu zobrazí, avšak místo vykreslení komponent s daty se zobrazí chybová zpráva.



Obrázek 23: Ukázka výjimky 404

### 5.3.2 Zobrazení dat

Stránky ...*Views.java* v první radě nastavují skrze anotace oprávnění a routování. Dále třída rozšiřuje *VerticalLayout*, které obsahuje vlastní komponentu *GridFormLayout*, která jak napovídá název obsahuje *Grid* a vlastní *Form* komponentu, kde je obsluhována správa vybraného záznamu. V poslední řadě mohou tyto UI třídy obsahovat implementaci *HasUrlParameter* v rámci navigování (Obrázek 24).

```
34 @Route(value = "components", layout = MainLayout.class)  Pokorný
35 @RouteAlias(value = "komponenty", layout = MainLayout.class)
36 @PageTitle("Komponenty")
37 @PermitAll
38 public class ComponentsView extends VerticalLayout implements HasUrlParameter<String> {
```

Obrázek 24: Ukázka implementace UI

Implementace těchto tříd je víceméně stejná.

- DI a atributy
- implementace *HasUrlParameter*
- konstruktor v kterém se vytváří *GridFormLayout*, nastaví se název stránky v hlavním menu a volají se privátní konfigurační metody.
- konfigurace filtrů, formuláře, gridu a akčních tlačítek.
- nastavení obsluhy pro událost smazání a uložení záznamu, které jsou vyvolány ve formuláři.

*GridFormLayout* je vytvořen čistě za účelem minimalizace boilerplate kódu.

Konfigurace *Grid* komponenty představuje největší výzvu, protože v rámci záznamu jsou sloupce renderovány několika komponentami, které jsou na míru potřebě nebo jen čistě dle typu dat. Mezi další překážky kódové optimalizace je vytvoření filtrovacích polí a jejich filtrovacích tříd v propojení na generické gridy, které existují pro většinu sloupců všech tabulek. Pro tyto účely byla vytvořena třída se statickými metodami, které vrací konfigurované filtrovací pole.

### 5.3.3 Správa dat

CRUD operace lze po selekci v tabulkovém zobrazení obsloužit ve formulářích, které rozšiřují *FormLayout*. Ústřední komponentou těchto sestavených formulářů pro každou entitu

jsou validační mechanismy nabízené Vaadin Flow, tzv. (*BeanValidationBinder*). Jedná se o abstraktní komponentu, která provazuje hodnotu entity s hodnotou komponenty, která tuto hodnotu upravuje. Respektive nabízí funkcionality a pro čtení, načtení a validaci včetně upozornění o nevalidních vstupech viz Obrázek 25.



```
60 private void setupPhone() { 1 usage Pokorný *
61     phoneField.setMinLength(9);
62     phoneField.setPattern("[+][0-9]{2,3}?[0-9]{9}");
63     this.setColspan(phoneField, colspan: 2);
64     customerBinder.forField(phoneField)
65         .asRequired()
66         .bind(Contact::getPhone, Contact::setPhone);
67 }
68
69 private void setupEmail() { 1 usage Pokorný *
70     customerBinder.forField(emailField)
71         .withValidator(new EmailValidator( errorMessage: "Nevalidní email."))
72         .asRequired()
73         .bind(Contact::getEmail, Contact::setEmail);
74 }
```

Obrázek 25: Ukázka data binding ve formulářích

Každý formulář implementuje rozhraní *IEditForm<Entity>*, které slouží pro volání metod formuláře z tříd stránek. Jedná se o načtení dat, čtení dat, validaci dat a událost při roztažení formuláře na úkor zobrazení dat v komponentě *Grid*.

Některé formuláře vyžadují vyšší komplexitu, protože pracují s entitami napříč relační databází. Děje se tak například na stránce s komponentami ve skladu. Při CRUD operacích na komponentě lze komponentu odstranit či přiřadit k produktu a editovat potřebné množství. Jinými slovy uživatel vidí a může pracovat s komponentou ve formuláři pro komponenty v souvislostech, kde je komponenta potřeba a kolik je ji potřeba a s touto informací pracovat při objednávání skladových zásob. Tato a podobné funkcionality jsou plněny vložením formuláře do formuláře. Pro uživatele se opticky (viz Obrázek 26) ani funkčně nic nemění a po technické stránce se jedná vyšší logiku pro validaci a byznys logiku před uložením do databáze.

☰

Komponenty

JP

Jakub P ▾

🌙

✓

🗑️

🔄

⌕

Název •

Rovný kabel

Popis

Skladem •

Minimálně sklad... •

Cena •

Notifikovat

– 97 ks +

– 200 ks +

– 100 Kč +

▾

Produkty

Nabíjecí kabely - sada × Nabíjecí box × Wallbox × × ▾

Počet pro: Nabij... •

Počet pro Nabije... •

Počet pro Wallbox •

– 1 +

– 1 +

– 1 +

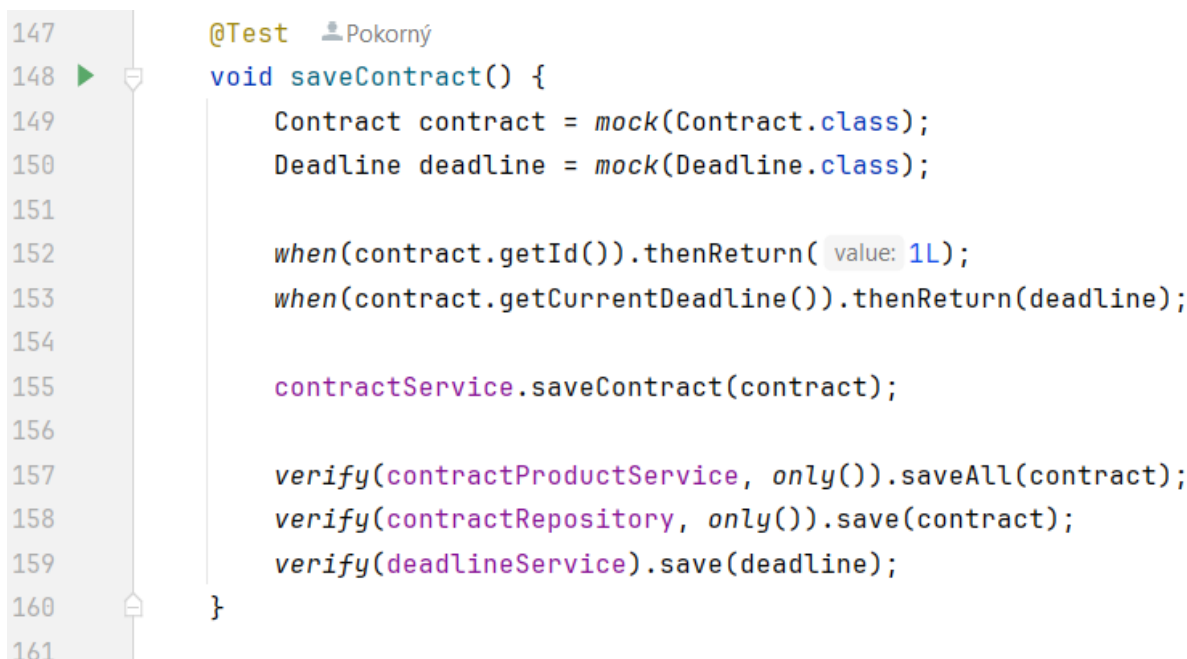
⌵

Obrázek 26: Ukázka formuláře pro CRUD operace

## 6 TESTOVÁNÍ

Testování bylo zaměřeno především na byznys vrstvu, kde je drtivá většina logiky. Pro projekt byl použit JUnit5 framework v kombinaci s Mockito. Bylo napsáno 94 testů (viz Obrázek 27). Každá testovací třída se rozšiřuje o `@ExtendWith(MockitoExtension.class)`. Poté je nutné předstírat (`@Mock`) DI a předstírané DI vložit (`@InjectMocks`) do testované Spring Boot komponenty (`@Service`, `@Repository` apod.). Obdobně lze mock techniku použít i pro ostatní potřebné třídy v rámci testu. Testy mají po většinu strukturu Arrange-Act-Assert (AAA) viz Obrázek 27:

- WHAT (Arrange): nastavení počátečního stavu, inicializace. Naplnění předpokladů.
- WHEN (Act): volání testované události. Také se zde mockují potřebné metody, např. repozitáře.
- THEN (Assert): ověření výsledků s očekáváním.



```
147  @Test  Pokorný
148  void saveContract() {
149      Contract contract = mock(Contract.class);
150      Deadline deadline = mock(Deadline.class);
151
152      when(contract.getId()).thenReturn(value: 1L);
153      when(contract.getCurrentDeadline()).thenReturn(deadline);
154
155      contractService.saveContract(contract);
156
157      verify(contractProductService, only()).saveAll(contract);
158      verify(contractRepository, only()).save(contract);
159      verify(deadlineService).save(deadline);
160  }
161
```

Obrázek 27: Ukázka testu

Cílem je v minimálním čase testovat část kódu při stejných výchozích podmínkách. Proto se pro testování minimálně byznys logiky nepoužívá databáze. Dotazy do databáze jsou pomalé a výsledky se mohou v čase měnit. Mockování je schopno tyto problémy vyřešit, kde nahrazujeme, v ideálním případě, již otestované části kódu.

## 7 NASAZENÍ

Posledním krokem je vytvoření Docker image, který se skrze Docker Hub poskytne klientovi. Klient si následně Docker image je schopen stáhnout a spustit v kontejneru na své serveru.

Před vytvořením image se projekt vyčistí a připraví pro produkční build. Nastavení je připravené od akcelérátoru Vaadin a nebylo třeba provádět změny. Příkaz:

```
mvn clean package -Pproduction
```

Pro build lze použít připravený DockerFile. Pokud se naházíme ve složce, kde se nachází i samotný DockerFile lze použít pouze ".". Přidáním "-t" bude výsledný Docker image mít určený tag, např. "latest":

```
docker build . -t is-diploma-thesis:latest
```

Pokud bychom chtěli vytvořený image otestovat, můžeme je spustit příkazem run. Tomcat server aplikace běží na portu 8888. Parametrem "-p 8080:8888" se určí, že port kontejneru 8888 se mapuje na hostitelský port 8080. Při nasazení na server je dobré doplnit i parameter "--restart=always". Pokud by kontejner spadl automaticky se vždy pokusí restartovat. Alternativně i například "on-failure" nebo "unless-stopped".

```
docker run --restart=always -p 8080:8888 is-diploma-thesis:latest
```

Posledním krokem je nahrání na Docker Hub, kam je potřeba se nejprve přihlásit. V Docker Hub lze pro tyto účely, v rámci účtu, generovat přihlašovací tokeny.

```
docker login
```

Po úspěšném přihlášení samotné nahrání Docker image:

```
docker push st58594/dt:latest
```

Reverzně stažení:

```
docker pull st58594/dt:latest
```

# ZÁVĚR

Hlavním cílem diplomové práce bylo v kontextu moderních možností navrhnout a implementovat systém ERP pro malou firmu zabývající se výrobou a prodejem. Výsledný systém měl spravovat a sledovat životní cyklus zakázek a to včetně skladu s klíčovými komponentami. Na komerčním trhu komplexní řešení poskytují především SAP, Oracle a Microsoft. Alternativou jsou především SaaS nástroje nebo vlastní řešení. Klient požadoval vlastní a specifické řešení, které bylo i finálně implementováno.

Teoretická část je věnována vysvětlení, vývoji a úskalím ERP. Je přidána i osobní zkušenost z firmy, která je nucena se ERP zabývat pro vlastní rozvoj. ERP a obecně SaaS služby jako technologická podpora podnikové agendy, je rostoucí odvětví, které se vyvíjí i vzhledem k novým technologiím. Při implementaci ERP se v mnohém postupuje podobně jako při vývoji softwaru s tím rozdílem, že je kladen vysoký důraz na podnikové procesy, které jsou v rámci implementace zpochybňovány, respektive vylepšovány pro lepší efektivitu.

V praktické části je popsán celý proces vývoje webové aplikace, která plní stanovené funkční a nefunkční požadavky v technologii Spring Boot a Vaadin Flow. Průběh vývoje se držel rámcově metodiky UP. Výsledná aplikace je publikována na Docker Hub jako Docker image.

Teoretické i praktické cíle se podařilo naplnit v plném rozsahu. V aplikaci, i díky nabytým vědomostem v teoretické části, se podařilo vytvořit platformu pro odbavení dodavatelských a obchodních procesů. Vaadin Flow zafungoval jako efektivní framework, který akceleroval především vývoj front-end části aplikace. Na druhou stranu v back-end části nabízené SSR v poněkud black-box podobě představovalo zdržení. Průběh vývoje tak potvrzuje, že Vaadin Flow za cenu nižší kontroly dodává produktivní a použitelné řešení i pro malé webové aplikace.

Rozvoj aplikace je možný, avšak pokud by se mělo jednat o integraci mnoha modulů a funkcí pro různé podnikové procesy, zvažoval bych implementaci ERP řešení od dodavatele.

Tato diplomová práce neměla ambice posouvat poznání v tématu, avšak mě jako autora posunula především v programování, které je těžké zachytit v průvodním textu. Jedná se tak o můj pokus důstojného zakončení studia.



# POUŽITÁ LITERATURA

- [1] AMAZON. *Amazon S3*. Online. 2024. Dostupné z: <https://aws.amazon.com/s3/>. [cit. 2024-08-12].
- [2] ARLOW, Jim a NEUSTADT, Ila. *UML 2 and the unified process: practical object-oriented analysis and design*. Boston: Addison-Wesley, 2005. ISBN 0-321-32127-8.
- [3] DYNAMICS SQUARE. *Dynamics 365 Implementation Services*. Online. Dynamics Square. 2023. Dostupné z: <https://www.dynamicssquare.com/our-services/dynamics-365-implementation-services/>. [cit. 2024-08-08].
- [4] GOOGLE. *Develop Gmail solutions*. Online. 2024. Dostupné z: <https://developers.google.com/gmail>. [cit. 2024-08-12].
- [5] HANIFI, A. *Transition, Turbulence and Combustion Modelling: Lecture Notes from the 2nd ERCOFTAC Summerschool held in Stockholm, 10-16 june, 1998*. Dordrecht: Kluwer Academic Publishers, 1999. ISBN 0-7923-5989-5.
- [6] HIETALA, H. a PÄIVÄRINTA, T. Benefits Realisation in Post-Implementation Development of ERP Systems: A Case Study. Online. *Procedia Computer Science*. 2021, roč. 181, s. 419-426. ISSN 18770509. Dostupné z: <https://doi.org/10.1016/j.procs.2021.01.186>. [cit. 2024-08-07].
- [7] KATUU, Shadrack. Enterprise Resource Planning: Past, Present, and Future. Online. *New Review of Information Networking*. 2020, roč. 25, č. 1, s. 37-46. ISSN 1361-4576. Dostupné z: <https://doi.org/10.1080/13614576.2020.1742770>. [cit. 2024-08-05].
- [8] MICROSOFT. *Microsoft Dynamics Sure Step Methodology: A Complete Guide*. Online. Dynamics 365 Community. 2023. Dostupné z: <https://community.dynamics.com/blogs/post/?postId=1d3c6b24-cc11-49cb-bfcc-8cdb9f83cb0f>. [cit. 2024-08-08].
- [9] MIELL, Ian a Aidan H. SAYERS. *Docker in Practice*. 2nd ed. Shelter Island: Manning Publications Co., 2019. ISBN 978-1-617-29480-8.

- [10] MINISTERSTVO FINANCÍ ČR. *Administrativní registr ekonomických subjektů*. Online. 2023. Dostupné z: <https://ares.gov.cz/>. [cit. 2024-08-12].
- [11] NAGPAL, Shruti; KHATRI, Sunil Kumar a KUMAR, Ashok. Comparative study of ERP implementation strategies. Online. In: 2015 *Long Island Systems, Applications and Technology*. IEEE, 2015, s. 1-9. ISBN 978-1-4799-8643-9. Dostupné z: <https://doi.org/10.1109/LISAT.2015.7160177>. [cit. 2024-08-07].
- [12] NAZEMI, Eslam; TAROKH, Mohammad Jafar a DJAVANSHIR, G. Reza. ERP: a literature survey. Online. *The International Journal of Advanced Manufacturing Technology*. 2012, roč. 61, č. 9-12, s. 999-1018. ISSN 0268-3768. Dostupné z: <https://doi.org/10.1007/s00170-011-3756-x>. [cit. 2024-08-05].
- [13] OCHRANA, František. *Metodologie vědy: úvod do problému*. V Praze: Karolinum, 2009. ISBN 978-80-246-1609-4.
- [14] ORACLE. *Oracle Enterprise Resource Planning (ERP) product tours*. Online. Oracle. 2024. Dostupné z: <https://www.oracle.com/erp/product-tours/>. [cit. 2024-08-08].
- [15] PRODUCTPLAN. *Agile Manifesto*. Online. ProductPlan. 2024. Dostupné z: [urlhttps://www.productplan.com/glossary/agile-manifesto/](https://www.productplan.com/glossary/agile-manifesto/). [cit. 2024-08-08].
- [16] PRODUCTPLAN. *Agile Principles*. Online. ProductPlan. 2024. Dostupné z: <https://www.productplan.com/glossary/agile-principles/>. [cit. 2024-08-08].
- [17] ŘEPA, Václav. *Podnikové procesy: procesní řízení a modelování*. 2., aktualiz. a rozš. vyd. *Management v informační společnosti*. Praha: Grada, 2007. ISBN 978-80-247-2252-8.
- [18] SALUR, Mehmet Nuri a KATTAR, Walaa Khoder. THE IMPACT OF ENTERPRISE RESOURCE PLANNING (ERP) ON THE AUDIT IN THE CONTEXT OF EMERGING TECHNOLOGIES. Online. *Ekonomi Maliye İşletme Dergisi*. 2021, roč. 4, č. 2, s. 115-123. ISSN 2667-4378. Dostupné z: <https://doi.org/10.46737/emid.1032735>. [cit. 2024-08-09].
- [19] SAP. *What is ERP?* Online. SAP. 2024. Dostupné z: <https://www.sap.com/products/erp/what-is-erp.html>. [cit. 2024-08-07].

- [20] SAP. Produkty SAP. Online. SAP. 2024. Dostupné z: <https://www.sap.com/cz/products.html>. [cit. 2024-08-08].
- [21] *Scrum Framework*. Online. In: MEDIUM. 2023. Dostupné z: [https://miro.medium.com/v2/resize:fit:1100/format:webp/1\\*pAjNWHh12kERAU3JzZaN2A.jpeg](https://miro.medium.com/v2/resize:fit:1100/format:webp/1*pAjNWHh12kERAU3JzZaN2A.jpeg). [cit. 2024-08-09].
- [22] SHE, Wei a THURAISINGHAM, Bhavani. Security for Enterprise Resource Planning Systems. Online. *Information Systems Security*. 2007, roč. 16, č. 3, s. 152-163. ISSN 1065-898X. Dostupné z: <https://doi.org/10.1080/10658980701401959>. [cit. 2024-08-05].
- [23] SPRING. *Spring*. Online. 2024. Dostupné z: <https://spring.io/>. [cit. 2024-08-09].
- [24] *Spring Boot Flow Architecture*. Online. In: InterviewBit. 2023. Dostupné z: <https://www.interviewbit.com/blog/wp-content/uploads/2022/06/Spring-Boot-Workflow-Architecture-800x480.png>. [cit. 2024-08-09].
- [25] VAADIN. *Vaadin*. Online. 2024. Dostupné z: <https://vaadin.com/>. [cit. 2024-08-09].

# SEZNAM PŘÍLOH

Příloha A – Webová aplikace .....	69
-----------------------------------	----

# PŘÍLOHA A – WEBOVÁ APLIKACE

Obsahuje Maven projekt se zdrojovými kódy aplikace.