Česká zemědělská univerzita v Praze
Provozně ekonomická fakulta
Katedra informačního inženýrství


# Bakalářská práce
# Otevřený IS malé firmy s použitím webových technologií

Marek Lukáš

13. března 2021

# ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Marek Lukáš

Systémové inženýrství a informatika
Informatika

Název práce

**Otevřený IS malé firmy s použitím webových technologií**

Název anglicky

**Opensource IS for small businesses using web technologies**

---

**Cíle práce**

Cílem práce je navrhnout a implementovat open-source informační systém pro správu probíhajících zakázek malé firmy. Hodlám využít existujících webových technologií a zanalyzovat aktuální stav na poli webových frameworků. Výsledný systém bude implemetován za použití klient – server achitektury.

**Metodika**

V teoretické části bakalářské práce se hodlám zaměřit na analýzu moderních trendů při vytváření informačních systémů. Například vhodnost různých technologií dle rozsahu systému, srovnání výkonu různých řešení a úskalí spojená s moderními stacky založenými okolo Node.js.

V praktické části získané poznatky přímo použiji ke vhodné implementaci vyvíjeného systému za použití Node.js, relační databáze a frameworku Vue. Na závěr připravím řešení pro reprodukovatelné nasazení na produkční instanci.

**Doporučený rozsah práce**

35-40 stran

**Klíčová slova**

web, SQL, node.js, typescript, vue

---

**Doporučené zdroje informací**

BASARAT, Ali Syed. TypeScript Deep Dive [online]. B.m.: Samurai Media Limited, nedatováno. ISBN 978-988-8407-12-5.

BASL, Josef, Roman BLAŽÍČEK a ČESKÁ SPOLEČNOST PRO SYSTÉMOVOU INTEGRACI, 2012. Podnikové informační systémy: podnik v informační společnosti. Praha: Grada. ISBN 978-80-247-4307-3.

GREIF, Sascha a Raphaël BENITTE, 2019. The State of JavaScript 2019 [online] [vid. 2020-12-20]. Dostupné z: https://2019.stateofjs.com/

HAVERBEKE, Marijn. Eloquent JavaScript: a modern introduction to programming [online]. 2019 [vid. 2020-12-19]. ISBN 978-1-59327-951-6.

RICHARDSON, Leonard a Sam RUBY. RESTful web services. Farnham: O'Reilly, 2007. ISBN 978-0-596-52926-0.

---

**Předběžný termín obhajoby**

2020/21 LS – PEF

**Vedoucí práce**

Ing. Jiří Brožek, Ph.D.

**Garantující pracoviště**

Katedra informačního inženýrství

Elektronicky schváleno dne 23. 2. 2021

**Ing. Martin Pelikán, Ph.D.**

Vedoucí katedry

Elektronicky schváleno dne 23. 2. 2021

**Ing. Martin Pelikán, Ph.D.**

Děkan

V Praze dne 01. 03. 2021

**Čestné prohlášení**

Prohlašuji, že svou bakalářskou práci „Otevřený IS malé firmy s použitím webových technologií“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 13. března 2021 _____

**Poděkování**

Rád bych touto cestou poděkoval vedoucímu této práce, Ing. Jiřímu Brožkovi, Ph.D. za ochotu na konzultacích a za vřelé jednání. Též děkuji mé rodině, která mne při psaní bakalářské práce podporovala.

# Otevřený IS malé firmy s použitím webových technologií

## Abstrakt

Tato práce se zabývá tvorbou webové podnikové aplikace pro správu zakázek malé firmy.

Teoretická část popisuje stav trhu s podnikovými systémy a technologie, které se ke tvorbě webových systémů používají.

Praktická část popisuje strukturu jednotlivých částí systému, jeho návrh, implementaci a způsob nasazení.

**Klíčová slova:** web, SQL, node.js, typescript, vue, vuetiy, routing-controllers, typeorm

# Opensource IS for small businesses using web technologies

## Abstract

The main goal of this thesis is to analyse and develop a web-based contract management application for a small business.

The theorical part talks about the current state of existing ERP systems and what kind of web technologies is used for such systems.

The main part of the thesis talks about the developing system structure, its design, implementation and deployment.

**Keywords:** web, SQL, node.js, typescript, vue, vuetiy, routing-controllers, typeorm

# Obsah

**1 Úvod** **14**

**2 Cíl práce a metodika** **15**

**3 Teoretická východiska** **16**
3.1 Svobodný software . . . . . 16
3.2 Licence . . . . . 16
3.2.1 MIT License . . . . . 17
3.2.2 GNU General Public License v3.0 . . . . . 17
3.2.3 Apache License 2.0 . . . . . 17
3.2.4 GNU Affero General Public License v3.0 . . . . . 17
3.3 Informační systémy . . . . . 18
3.4 Podnikové systémy . . . . . 18
3.4.1 ERP . . . . . 19
3.4.2 Existující řešení . . . . . 19
3.4.3 Základní struktura podnikového systému . . . . . 26
3.4.4 Technologie podnikových systémů . . . . . 28
3.5 Webové technologie . . . . . 31
3.5.1 WWW . . . . . 31
3.5.2 Statický web . . . . . 31
3.5.3 Dynamický web . . . . . 32
3.5.4 Server side . . . . . 34
3.5.5 Client side . . . . . 42

**4 Vlastní práce** **50**
4.1 Požadované funkce . . . . . 50
4.1.1 Celkový styl . . . . . 50
4.2 Výběr technologie . . . . . 51
4.2.1 Server . . . . . 51
4.2.2 Klient . . . . . 52
4.3 Struktura projektu . . . . . 52
4.3.1 Databáze . . . . . 53
4.3.2 API server . . . . . 56

4.3.3 Grafické rozhraní . . . . . 64
4.3.4 Nasazení . . . . . 84

**5 Výsledky a diskuse** **89**

**6 Závěr** **91**

**7 Seznam použitých zdrojů** **92**

**8 Přílohy** **99**

# Seznam obrázků


4.1 Ikona aplikace . . . . . 51
4.2 Paleta barev grafického rozhraní . . . . . 51
4.3 Struktura vzájemných závislostí interních balíčků projektu . . . . . 53
4.4 UML relační diagram výsledné databáze . . . . . 56
4.5 Struktura kódu @bokari/api-server . . . . . 58
4.6 Vizualizace dokumentace API z vygenerované OpenAPI specifikace . . 65
4.7 Wireframe přihlašovací obrazovky . . . . . 70
4.8 Výsledná podoba přihlašovací obrazovky . . . . . 70
4.9 Wireframe zobrazení seznamu zakázek . . . . . 71
4.10 Výsledná podoba zobrazení seznamu zakázek . . . . . 71
4.11 Výsledná podoba zobrazení seznamu zakázek na mobilním zařízení . . . 72
4.12 Wireframe zobrazení formuláře pro vytvoření nové zakázky . . . . . 73
4.13 Výsledná podoba zobrazení formuláře pro vytvoření nové zakázky . . . 73
4.14 Wireframe zobrazení existující zakázky . . . . . 74
4.15 Výsledná podoba zobrazení existující zakázky . . . . . 74
4.16 Výsledná podoba zobrazení existující zakázky na mobilním zařízení . . 75
4.17 Struktura kódu @bokari/ui . . . . . 77

# Seznam ukázek kódu


1 Odbavení GET /hello v Spring . . . . . 37
2 Odbavení GET /hello v Laravel . . . . . 37
3 Odbavení GET /hello v Django . . . . . 38
4 Odbavení GET /hello v ASP.NET . . . . . 38
5 Odbavení GET /hello v Express . . . . . 39
6 Odbavení GET /hello pomocí routing-controllers . . . . . 40
7 Odbavení GET /hello v NestJS . . . . . 40
8 Odbavení GET /hello v Feathers . . . . . 41
9 Odbavení GET /hello v Next . . . . . 41
10 Odbavení GET /hello v Nuxt . . . . . 42
11 Jednoduchý reaktivní čítač v React . . . . . 45
12 Jednoduchý reaktivní čítač v Angular (html) . . . . . 46
13 Jednoduchý reaktivní čítač v Angular (logika) . . . . . 46
14 Jednoduchý reaktivní čítač ve Vue 2 . . . . . 48
15 Jednoduchý reaktivní čítač ve Vue 3 . . . . . 49
16 Definice entity Zakázka . . . . . 54
17 Vytvoření účtu správce . . . . . 55
18 Vytvoření trigram indexu pomocí migrace . . . . . 55
19 Třída udržující instanci samotného http serveru . . . . . 57
20 Handler zajišťující vyhledávání v seznamu zakázek . . . . . 59
21 Třídy definující možné parametry vyhledávání . . . . . 61
22 Kontrola a načtení autorizace z tokenu . . . . . 63
23 Obsah handleru GET /openapi-docs . . . . . 64
24 Vue komponenta seznamu zakázek (část šablony) . . . . . 76
25 Vue komponenta seznamu zakázek (část logiky) . . . . . 78
26 Hook zajišťující globální správu upozornění . . . . . 79
27 Komponenta starající se o zobrazení upozornění z globálního stavu . . 80
28 Hook zajišťující validaci pro specifický formulář . . . . . 81
29 Načtení hooku ve skriptové části komponenty . . . . . 81
30 Aplikace validačních pravidel na textové pole . . . . . 81
31 Lokální oprava vygenerovaného kódu (pouze výběr) . . . . . 83
32 Použití vygenerovaného a inicializovaného API klienta . . . . . 83
33 Dockerfile pro sestavení a spuštění Vue části aplikace . . . . . 84

34 Soubor docker-compose.yml . . . . . 85
35 Aktuální konfigurace webové proxy nginx . . . . . 87
36 Výstup testu rychlosti API endpointu GET /users/admin . . . . . 88

# Seznam použitých zkratek

**ACID** Atomicity, Consistency, Isolation, Durability . . . . . 29

**API** Application Programming Interface . . . . . 26, 34, 39, 41, 48, 51, 52, 82, 83, 86

**BI** Business Intelligence . . . . . 19

**CDN** Content Delivery Network . . . . . 47

**CRM** Customer Relationship Management . . . . . 19, 22, 25

**CRUD** Create, Read, Update, Delete . . . . . 40

**CSS** Cascading Style Sheets . . . . . 43, 45, 47

**DBMS** Database Management System . . . . . 29

**DOM** Document Object Model . . . . . 44, 45, 47

**ERP** Enterprise Resource Planning . . . . . 18–20, 24, 25, 27

**ES** ECMAScript . . . . . 52

**GB** Gigabyte . . . . . 86

**GPS** Global Positioning System . . . . . 33

**HR** Human Resources . . . . . 21

**HTML** Hypertext Markup Language . . . . . 42, 43, 45, 47

**HTTP** Hypertext Transfer Protoco . . . . . 31, 33, 34, 39, 41, 43, 82, 83

**IO** Input-Output . . . . . 36

**JS** JavaScript . . . . . 90

**JSON** JavaScript Object Notation . . . . . 43, 63

**JWT** JSON Web token . . . . . 62, 90

**MVT** Model, View, Template . . . . . 37

**NPM** Node.js Package Manager . . . . . 36

**ORM** Object-Relational Mapping . . . . . 40, 52

**PHP** PHP: Hypertext Preprocessor . . . . . 35, 37

**PWA** Progressive Web App . . . . . 33

**RAM** Random Access Memory . . . . . 86

**REST** Representational State Transfer . . . . . 33, 34, 52, 89

**SCM** Supply Chain Management . . . . . 19

**SPA** Single Page App . . . . . 32, 34, 44, 45, 52, 63

**SQL** Structured Query Language . . . . . 29, 34, 52, 57, 89

**SSG** Server Side Generation . . . . . 41

**SSO** Single Sign-On, česky Jednotné přihlášení . . . . . 21

**UI** User Interface . . . . . 21, 44, 52

**URL** Uniform Resource Locator . . . . . 31

**VM** Virtual Machine . . . . . 35

**WWW** World Wide Web . . . . . 31

**XML** Extensible Markup Language . . . . . 42, 43

# 1 Úvod

Správa projektů je pro mnoho, zvláště menších firem, náročnou činností. Většina tento problém řeší buď pracnou organizací v papírové formě, nebo se uchýlí k nějakému finančně náročnému softwarovému řešení.

Pokud si firma vybrala možnost první, není schopna si dokumenty nijak rychle dohledat a aktualizace stavu projektu z terénu je též nemožná.

Malé firmy si zase často nemohou dovolit investovat do komerčního organizačního softwaru. Ať už jde o finanční stránku nebo čistě o komplikovanost jejich nasazení a používání.

Chtěl bych vytvořit svobodně dostupné řešení pro správu zakázek malé firmy, která by se ráda posunula do moderního digitálního světa. Toto řešení by jí mělo zefektivnit vedení dokumentace a nabídnout rychlý přehled o tom, co jsou aktuální priority. Řešení by též mělo být alespoň z části generické, aby jej mohly využívat firmy různých odvětví, od truhlářů po projektanty.

Aplikace bude vyvinuta částečně podle požadavků mého otce, v jehož firmě by měla být po dokončení používána.

Protože řešení bude dostupné pod svobodnou licencí, budu stavět jen na otevřených standardech a technologiích.

# 2 Cíl práce a metodika

Cílem práce je navrhnout a implementovat open-source informační systém pro správu probíhajících zakázek malé firmy. Hodlám využít existujících webových technologií a zanalyzovat aktuální stav na poli webových frameworků. Výsledný systém bude implementován za použití klient - server architektury.

V teoretické části bakalářské práce se hodlám zaměřit na analýzu moderních trendů při vytváření informačních systémů. Například vhodnost různých technologií dle rozsahu systému, srovnání výkonu různých řešení a úskalí spojená s moderními stacky založenými okolo Node.js.

V praktické části získané poznatky přímo použiji ke vhodné implementaci vyvíjeného systému za použití Node.js, relační databáze a frameworku Vue. Na závěr připravím řešení pro reprodukovatelné nasazení na produkční instanci.

# 3 Teoretická východiska

## 3.1 Svobodný software

Dle Free Software Foundation (2001) je svobodný software brán za takový software, který dává uživatelům svobodu spouštět, kopírovat, distribuovat, studovat, měnit a zlepšovat jej. Přesněji řečeno, vztahuje se ke čtyřem svobodám pro všechny uživatele software:

- ***Svoboda spustit program za jakýmkoliv účelem*** *(svoboda 0).*
- ***Svoboda studovat****, jak program pracuje a přizpůsobit ho svým potřebám (svoboda 1). Předpokladem k výše uvedenému je přístup ke zdrojovému kódu.*
- ***Svoboda redistribuovat kopie****, abyste pomohli vašemu kolegovi (svoboda 2).*
- ***Svoboda vylepšovat*** *program a zveřejňovat zlepšení, aby z nich mohla mít prospěch celá komunita. (svoboda 3). Předpokladem k výše uvedenému je přístup ke zdrojovému kódu.*

> *Program je svobodným software, pokud uživatelé mají všechny tyto svobody. Měli byste moct redistribuovat kopie, buď modifikované či nikoliv, zadarmo nebo s poplatkem za distribuci komukoliv kdekoliv. Mít svobodu dělat tyto věci znamená (mimo jiné), že nemusíte nikoho žádat o povolení, nebo za něj platit. (Free Software Foundation 2001)*

## 3.2 Licence

Jak můžeme se softwarem nakládat, se můžeme dočíst v jeho licenční deklaraci. Licence by zpravidla měla být dodávána s každou kopií daného softwaru a měla by jasně obsahovat, jak nám jeho autor dovoluje se softwarem nakládat. U proprietárního komerčního softwaru se tedy může uživatel například dočíst, že nemá oprávnění analyzovat jeho interní fungování či do něj jakkoliv zasahovat. Případně nám může i licence zakázat snahu spouštět program v nepodporovaném prostředí (například pod jiným operačním systémem).

### 3.2.1 MIT License

MIT je brána za jednu z nejjednodušších softwarových licencí. Uživateli umožňuje nakládat se softwarem jak je libo, jen s tím omezením, že musí být při redistribuci zachována stejná licence a ponechán copyright autora. Pokud bychom software ovšem modifikovali a chtěli vydávat jako novou entitu, můžeme tak učinit i se změnou licence. Zachována musí být pouze zmínka o původním copyrightu. (Massachusetts Institute of Technology 1999)

### 3.2.2 GNU General Public License v3.0

GNU General Public Licence v3.0 je oproti MIT rozsáhlejší a zajišťuje i ponechání stejné licence ve všech derivovaných projektech. Protože GNU GPL požaduje zveřejnit celý zdrojový kód našeho „derivátu“ pod GPL licencí a hranice softwarového derivátu je velmi tenká, ve většině případů nelze použít jakýkoliv GNU GPL licencovaný kód v proprietárním softwaru, který distribuujeme. Upravit si ji můžeme, ale musíme dát k dispozici zdrojový kód obsahující veškeré námi provedené změny od originálu nebo licencovaný software používat jen interně. Na této licenci je postaveno například linuxové jádro. (Free Software Foundation 2007b)

### 3.2.3 Apache License 2.0

Apache 2.0 je někde na pomezí mezi GNU GPL a MIT. Podobně jako MIT nevyžaduje zveřejnění zdrojového kódu při použití cizího licencovaného kódu. Co si ale z GNU GPL půjčuje, je zveřejnění veškerých provedených změn od originálu použitého softwaru. Pokud bychom chtěli použít nějakou knihovnu zveřejněnou pod Apache 2.0 v našem proprietárním programu, můžeme tak učinit za podmínky, že uvedeme autora knihovny a do knihovny nebudeme nijak zasahovat. (Apache Software Foundation 2004)

### 3.2.4 GNU Affero General Public License v3.0

GNU Affero GPL (AGPL) vznikla jako reakce na mezeru ve specifikaci GNU GPL. Protože GNU GPL se z většiny omezuje na díla, která jsou dále distribuována, je plně legální použít GPL licencovaný kód v proprietární webové službě. Protože technicky vzato není distribuován žádný software a uživatelé jen komunikují přes vzdálené API, není nutné zveřejnit zdrojový kód webové služby. Tuto „chybu“ AGPL opravuje a autor služby, která by používala nějakou AGPL licencovanou knihovnu, by musel zveřejnit celý kód své služby též pod AGPL licencí. (Free Software Foundation 2007a)

## 3.3 Informační systémy

*„Informační systém je soubor lidí, technických prostředků a metod (programů), zabezpečujících sběr, přenos, zpracování, uchování dat, za účelem prezentace informací pro potřeby uživatelů činných v systémech řízení.“* (Molnár 2009)

Dle této definice je zřejmé, že informační systém nemusí být vůbec provozovaný za pomoci jakéhokoliv počítače. V našem případě bych se ale pro omezení rozsahu práce rád zaměřil pouze na elektronické implementace podnikových systémů zajišťujících Plánování podnikových zdrojů (ERP).

## 3.4 Podnikové systémy

Jedná se o takové informační systémy, které mají pomoci podniku (**E** - enterprise) se správou jeho zdrojů (**R** - resources) a plánováním (**P** - planning). Reálně jde o systémy sjednocující data z různých zdrojů a prezentující je uživatelům jako informace.

Podle Basl et al. (2012, s. 55) lze informační systém implementovat v existujícím podniku třemi variantami řešení:

**Pořízení nového hotového softwarového systému** je první možnost, se kterou mnoho firem počítá jako s nejjednodušším řešením. Umožňuje rychlé zavedení a zaručuje určitou kvalitu a fukčnost, pro kterou si jej firma pořizuje. V případě nákupu komerčního systému může firma počítat i s profesionální podporou. Nevýhodou může být plná závislost na dodavateli softwaru při úpravách nevyhovujících částí systému.

**Rozvoj aktuálně používaného systému** je z krátkodobého hlediska jistě lacinější a rychlejší, ale nemusí odpovídat všem budoucím požadavkům. V závislosti na rozsahu nutných úprav může nastat i vysoké navýšení nákladů a omezení kvality stávajícího systému. Doporučuje se tedy pouze u omezeného rozsahu požadovaných změn.

**Vývoj nového systému na míru** bývá nejdražší a časově nejnáročnějším řešením s negarantovaným výsledkem. Pokud pro firmu ale žádná jiná možnost nepřichází v úvahu, nabízí vývoj na míru nejbližší shodu s potřebami podniku. Od požadovaného hardwaru, spravovatelných funkcích, po styl uživatelského rozhraní.

### 3.4.1 ERP

Kategorie ERP (Enterprise Resource Planning) systémů se obecně zabývá mapováním procesů spojených s podnikovými zdroji. Zaznamenává informace o stavu systému a prezentuje je uživateli. Dále uživatelům umožňuje plánovat a analyzovat průběh sledovaných procesů.

V rozšířené formě, tzv. ERP II, se s těmito systémy lze setkat nejčastěji. V této formě tvoří ERP jakési pomyslné jádro podnikového informačního systému, který ale sám poté obsahuje i aplikace CRM (Customer Relationship Management), SCM (Supply Chain Management) či BI (Business Intelligence). (Basl et al. 2012, s. 67)

### 3.4.2 Existující řešení

#### 3.4.2.1 Proprietární

**SAP Business One**

Podnikový informační systém pro malé firmy od německé softwarové firmy SAP (***S****ysteme,* ***A****nwendungen,* ***P****rodukte in der Datenverarbeitung*).

Firma SAP byla založena již roku 1972 (SAP nedatováno e) a v průběhu let se stala největším dodavatelem podnikových systémů na světě (SAP nedatováno a). Jejich systémy používá například Ministerstvo financí České republiky nebo česká telekomunikační společnost CETIN. (SAP nedatováno d)

SAP se v minulosti zaměřoval spíše na velké společnosti. Pro malé firmy byly jeho systémy příliš komplikované a nákladné. To se částečně změnilo roku 2006 s příchodem systému **SAP Business One**, který SAP vytvořil s cílem zpřístupnit své služby i menším firmám. A to se mu i podařilo. Dle SAP je zisk, pocházející z licencí, tvořen ze 30% firmami pod 2500 zaměstnanců. (SAP nedatováno b)

SAP Business One nabízí vše, co by firma měla ke svému řízení potřebovat. Správu financí, prodeje, skladů a vztahu se zákazníky. Nově nabízí i možnost interakce se systémem z terénu pomocí mobilní aplikace. (SAP nedatováno c)

Pro podniky, které již používali nějaký systém od firmy SAP, lze následně SAP Business One integrovat s platformou SAP HANA, a tedy pracovat s již existujícími daty. (SAP nedatováno c)

**Pro**

- skvělá podpora
- možnost hostování v cloudu nebo na vlastním serveru
- webový i desktopový klient
- velké množství funkcí
- podpora české lokalizace

**Proti**

- komplikovaný ekosystém
- vysoká cena
- velký důraz na externí konzultaci

### Microsoft Dynamics

Microsoft Dynamics je ekosystém produktů pro firemní management od společnosti Microsoft.

Kořeny Dynamics sahají až do roku 1993, kdy společnost Great Plains Software vydala svou první stabilní verzi balíčku podnikového systému Dynamics. O 8 let později, kdy společnost Great Plains Software odkoupil Microsoft, byla započata práce na integraci Dynamics do existujícího ekosystému Microsoft. To se událo o další 4 roky později (prosinec 2005) vydáním Microsoft Dynamics GP 9.0. (Musgrave 2009)

Dynamics procházelo postupným vývojem, až se roku 2016 vyvinulo v plně cloudové řešení nazvané Microsoft Dynamics 365 (Wright 2018). Podobně jako Office 365 se snaží být co možná nejflexibilnější z hlediska smluvených služeb. Zákazníci platí ve formě předplatného jen ty služby, které chtějí používat.

Protože je Microsoft Dynamics 365 balíkem mnoha služeb, nelze ho brát jako takový přímo za ukázku informačního ERP systému. Co ale vyzdvihnout můžeme, je služba **Microsoft Dynamics 365 Business Central**. Vzniklá roku 2018 z produktu Dynamics NAV (Kotelko et al. 2018), nabízí řešení pro malé a střední společnosti, které vyžadují centrální správu účetnictví, výroby, prodeje/dodávek a řízení projektů (Bicek et al. 2020). S Business Central lze pracovat jak čistě cloudově, skrz tenké klienty komunikující s instancí serveru běžící na virtuálních serverech Microsoftu, tak i skrz vlastní hosting. (Kotelko et al. 2018)

**Pro**

- velmi flexibilní
- integrace s Office 365 (emaily z Outlook, přílohy z OneDrive, SSO)
- mnoho funkcí

**Proti**

- finančně náročné
- pro malé firmy příliš komplikované UI a koncepce

**Jira**

Skupina, aktuálně čtyř, produktů pro řízení firemních projektů od australské firmy Atlassian. (Atlassian 2020a)

Původní Jira vznikla roku 2002 jako samostatný nástroj pro správu projektů/úkolů cílící čistě na softwarově zaměřené firmy a týmy. Postupně se ale začala obalovat funkcemi cílícími i na ostatní odvětví trhu. Na podzim 2015 (Brereton 2015) byla Jira rozdělena na dvě oddělené větve produktu (Austin 2015). Jira Software a Jira Core. (Atlassian 2020a)

**Jira Sofware** pochází z původní vize Jira. Umožňuje týmům vývojářů správu celých projektů, hlášených chyb a procesů. Vše lze spravovat i na více úrovních, vývojáři tak mohou být děleni na podtýmy a každý úkol může mít libovolný počet pod-úkolů. (Atlassian nedatováno)

**Jira Core** vychází též z původní Jira, ale je uzpůsobena netechnickým týmovým projektům. Cílí například na skupiny z marketingu, HR a financí. (Forman 2015)

Obě varianta je možné provozovat na jednom serveru a navzájem integrovat. Oddělení vývojářů tedy pracuje s Jira Software a zbytek firmy pracuje s rozhraním Jira Core.

**Pro**

- přehledné rozhraní
- jednoduchá koncepce
- mnoho pluginů třetích stran
- relativně nízká cena

**Proti**

- nižší flexibilita pro úpravy vlastních procesů
- od roku 2021 nemožnost selfhostingu (Atlassian 2020b)

**Jobber**

Cloudová služba navrhnuta ke zjednodušení denní práce malých firem poskytující lokální servisní služby. Umožňuje snadnou komunikaci mezi zaměstnanci v terénu a pokrývá celý proces přijetí zakázky po její průběžné plnění až po její fakturaci. I přes jeho zaměření na CRM (Jobber 2019b) bych jej pro jeho pevnou vizi rád zmínil.

**Jobber** původně vznikl jako systém určený pro správu zakázek a faktur jedné malé britské malířské firmy. Jak se původní řešení rozrůstalo, uvědomil si vývojář tohoto systému, že by tento systém mohl pomoci i dalším desítkám místních firem. Tak vznikl roku 2011 Jobber, řešení s vizí modernizace malých firem, které si nemohou dovolit financovat ani spravovat velké podnikové systémy, jako je například Microsoft Dynamics 365. (Jobber 2019a)

**Pro**

- srozumitelná koncepce
- moderní rozhraní
- mobilní aplikace pro terénní pracovníky
- nejde o balík, ale o jeden samostatný produkt (pro netechnické obory zjednodušení)

**Proti**

- podpora pouze anglického jazyka
- podpora práce s financemi pouze v omezeném množství měn
- nemožnost vlastního hostingu

**monday.com**

Relativní novinkou na trhu se systémy firemní správy je **monday.com**, jehož první stabilní verze byla zveřejněna 2014. (monday.com nedatováno b)

Monday.com se snaží zaujmout moderním uživatelským rozhraním a velkou mírou automatizace. Podobně jako Jira nabízí možnost propojení s velkým množstvím jiných služeb, jako je například Google Drive, Dropbox, Office 365. Oproti Jira má volnější

přístup ke struktuře práce s projekty. Místo pevného dělení projektů, úkolů a procesů, umožňuje uživatelům vytvořit si za pomoci předpřipravených šablon a komponent svůj vlastní způsob práce s vlastními daty. (monday.com nedatováno a)

**Pro**

- velmi flexibilní
- moderní uživatelské rozhraní
- v aktivním vývoji

**Proti**

- není ideální pro firmy s mnoha zaměstnanci
- nutnost vlastní přípravy procesů
- o něco vyšší cena než například Jira
- nemožnost vlastního hostingu
- rozhraní není přeloženo do češtiny

#### 3.4.2.2 Open source

**Odoo**

Vlajková loď open-source podnikových systémů pocházející původně z roku 2005 z Belgie od jediného vývojáře, Fabiena Pinckaerse. Ten si dal za svůj životní cíl stát se vedoucím trhu enterprise podnikových systémů s vlastním open-source řešením. Jako palivo své motivace si zvolil pomyslný souboj s tehdejší špičkou na trhu, německým systémem SAP.(Pinckaers 2013)

**Odoo** představuje skupinu komponent, ze kterých si uživatel může sám sestavit vlastní podnikový systém. Pokud si například uživatelská firma nespravuje vlastní finance, nemusí si aktivovat moduly s nimi spojené. Nabízí veškeré moduly, které by se od podnikového informačního systému daly očekávat. Od správy firemního eshopu po inventuru, podporu zaměstnanců v terénu nebo sledování zásilek.

Do verze Odoo V9 byl systém dostupný plně zdarma. Poté Odoo přešlo na model open core (Ginneken 2015), tedy základní skupinu open-source modulů zdarma (Odoo Community Edition) a přídavné proprietární moduly za poplatek (Odoo Enterprise Edition).

### Pro

- možnost vlastního hostování
- část plně svobodná
- velmi flexibilní díky výběru z 40+ samostatně aktivovatelných komponent
- funkčností konkuruje Microsoft Dynamics 365 a SAP
- podpora českého jazyka
- v některých zemích je jeho znalost v osnovách škol (Pinckaers 2013)

### Proti

- při nákupu Enterprise Edice a většího množství modulů velmi drahý
- nejistá budoucnost vývoje bezplatných modulů Odoo Community Edition

### ERPNext

Plně svobodný webový ERP systém od Frappé Technologies Pvt. Ltd.

Původně vytvořen Revantem Nandgaonkar s Rushabhem Mehta pro jejich vlastní rodinný indický podnik. Později byl zveřejněn jako open source pro všechny podniky s podobnými potřebami. (ERPNext Open Source Software Foundation 2021) **ERPNext** má za cíl pomoci malým a středním firmám ve správě všech jejich podnikových transakcí v jediném sjednoceném systému. Systém zvládá sledovat například stav fakturace, pohyb zboží na skladě, seznam klientely či správu firemních webových stránek. (Frappe 2020)

### Pro

- jednoduchá základní instalace
- mnoho funkcí
- snadná úprava existujících formulářů
- stále v aktivním vývoji
- podpora českého jazyka
- přijatelná cena za oficiálně hostovanou instanci

**Proti**

- pomalý vývoj mobilní aplikace, ne všechny funkce jsou v ní ihned dostupné
- absence centrálního repozitáře doplňků
- komplikovanější nastavování firemních procesů po instalaci

### Apache OFBiz

Skupina podnikových aplikací spravovaná nadací Apache Software Foundation.

Hlavním cílem **Apache OFBiz** není sestavit a distribuovat jeden otevřený podnikový systém, ale spíše vytvořit pevně daný, dobře definovaný ekosystém navzájem propojitelných aplikací. OFBiz totiž není jen skupina aplikací, ale i mocný Java framework, pomocí kterého jsou moduly psány. Díky OFBiz jako frameworku, je možné vytvořit vlastní moduly a zakomponovat je plynule do zbytku systému. (The Apache Software Foundation nedatováno)

Apache OFBiz pokrývá svými funkcemi drtivou většinu potřeb podniků všech velikostí. Obsahuje tedy celkové ERP a CRM řešení s možností správy financí i výroby.

**Pro**

- roky otestované řešení
- mnoho funkcí
- stále ve vývoji

**Proti**

- zastaralé a nepřehledné rozhraní
- pro malé firmy příliš komplexní
- nedostačující uživatelská i technická dokumentace

### ERP5

Systém s primárním zaměřením na ERP od francouzské společnosti Nexedi.

**ERP5** dokáže obstarat i operace spojené se správou zákazníků (CRM), financí a webovou publikací dat. (Nexedi nedatováno)

Interně používá pro popis dat jediný datový model, nazvaný *Unified Business Model* (Nexedi 2016), česky přeložitelný jako Sjednocený Model Podniku. Ten je složen z pěti složek. Zdrojů (Resource), Uzlů (Node), Pohybů (Movement), Položek (Item) a Cest (Path). **Zdroj** představuje abstraktní zdroj v rámci podnikového procesu, například materiál, produkt nebo nějakou dovednost. **Uzel** umí přijmout a odeslat Zdroje, může

se jednat například o bankovní účet nebo osobu. **Pohyb** označuje aktivitu přesunu Zdrojů mezi Uzly, může jít o přepravu materiálu ze skladu do výrobny. **Položka** slouží k popisu fyzické instance nějakého existujícího Zdroje, nejčastěji používána k popisu sériových čísel. **Cesta** definuje způsob, jakým Uzel dokáže přistupovat ke Zdrojům.

Protože je Sjednocený Model Podniku využit napříč všemi datovými komponentami, je snadné ERP5 používat téměř v jakémkoliv odvětví. Správce jen musí popsat procesy reálné firmy pomocí tohoto modelu.

**Pro**

- přehledné rozhraní
- stále ve vývoji

**Proti**

- chybějící podpora mobilních zařízení
- kvůli své obecnosti složitý pro první nastavení
- některé funkce mohou být skryté pod placenými moduly

### 3.4.3 Základní struktura podnikového systému

Reálně může být podnikový systém složen z desítek, navzájem propojených, aplikací, databází, modulů, služeb, API (Application Programming Interface). Jakožto na aplikace ale lze na podnikové systémy nahlížet i jednodušeji. Základní části jsou vždy stejné. Databáze, backend a frontend. (Osetskyi 2018)

**Databáze**: Část systému zodpovědná za uchovávání dat o aktivech, s nimiž uživatel pracuje.

**Backend**: Mozek systému provádějící kritické operace na základě požadavků uživatele.

**Frontend**: Interaktivní rozhraní umožňující uživateli komunikovat s backendem, tvořit požadavky a zobrazit získané výsledky.

#### 3.4.3.1 Tlustý klient

Nejstarší a nejjednodušší architektura podnikového systému. Frontend i backend systému běží přímo na uživatelově počítači. Databáze bývá přesto umístěna na jiném stroji pro snazší synchronizaci dat s více jednotkami.

Výhodou je rychlejší počáteční vývoj systému, protože backend i frontend mohou pracovat přímo se stejnými datovými objekty. Další výhodou mohou být i nižší nároky na kapacitu serveru, protože je většina operací prováděna na straně klienta. (Drábek 2009)

Nevýhodou je naopak vyšší nárok na výkon uživatelského stroje. A protože jsou kritické operace prováděny lokálně, bezpečnost je též snížena.

#### 3.4.3.2 Klient/Server

Přirozený nástupce tlustých klientů je architektura klient/server. Ta oproti tlustému klientu provozuje na straně uživatele pouze frontend. Backend je provozován na vzdáleném serveru.

Slibuje vysokou bezpečnost spojenou s možností autentizace klienta a skrytí aplikační logiky za nějaké předem definované rozhraní.

#### 3.4.3.3 Webové řešení

Podmnožina architektury klient/server. Oproti té ale frontend neběží přímo na uživatelově stroji, ale běží uvnitř prostředí jeho webového prohlížeče. Kvůli omezení na běh ve webovém prohlížeči je též omezen výběr možných protokolů pro komunikaci mezi frontendem a backendem.

Výhodou je maximální multiplatformnost. Frontend může běžet na téměř jakémkoliv zařízení s webovým prohlížečem. Ideální pro tenké klienty.

Trendem na poli ERP systémů je postupný přechod právě na tuto architekturu. A na tuto architekturu se i já zaměřím ve vlastní práci.

#### 3.4.3.4 Hybridní řešení

Kompromisní řešení nacházející se na pomezí tlustého klienta a klient/server. Umožňuje používat jak backend běžící lokálně, tak provozovat backend vzdáleně a lokálně provozovat pouze frontend. (Osetskyi 2018)

Hybridní řešení bylo a je ideální pro podniky přecházející z tlustých klientů na webové klienty. V kritických oblastech, kde není implementována veškerá funkcionalita ze dříve používaných tlustých klientů, mohou být tlusté klienty provozovány bez ztráty dat nebo zaučování na nové rozhraní.

### 3.4.4 Technologie podnikových systémů

#### 3.4.4.1 Správa dat

Protože efektivní podnikové systémy spravují chod většiny podnikových procesů a zdrojů, je nutné zajistit co nejvyšší bezpečnost při práci s podnikovými daty. Problémy spojené se správou dat jsou například (Rainer et al. 2014, s. 145):

*Nekonzistence dat*: Nastává při více instancích stejného údaje, které si protiřečí. Pokud bychom například ukládali údaje o adrese zákazníka ve dvou různých, navzájem nepropojených, datových složkách, a pak zákazník změnil adresu, a my ji opravili jen v jedné ze složek, nešlo by určit, která adresa je aktuální.

*Redundance dat*: Nastává při uložení stejných údajů na více místech zároveň. To může vést k jednoduché k datové *nekonzistenci*, protože při změně údaje musíme zkontrolovat a opravit jeho výskyt napříč všemi umístěními.

*Izolace dat*: Aplikace nejsou schopny přistupovat k datům jiných aplikací.

*Integrita dat*: Ukládaná data musí splňovat určité, předem specifikované, vlastnosti. Například, že telefonní číslo by nemělo obsahovat písmena, každý příspěvek na firemním blogu obsahuje odkaz na svého autora, nebo že data nebyla nijak poškozena při výpadku proudu v areálu datacentra.

### Databáze

Tyto a další problémy dokážeme řešit s pomocí tzv. databáze. Jedná se o centrálně organizovanou kolekci dat, která je většinou spravována DBMS (Database Management System), česky též známým jako *systémem řízení báze dat*. DBMS je softwarový nástroj, který hospodaří s databází a vytváří k ní nějaké vlastní aplikační rozhraní. Aplikace vyžadující přístup k datům z databáze poté nekomunikují přímo s databází, ale pouze s DBMS, který komunikaci s databází zprostředkuje. Oproti přímé komunikaci s daty usnadňuje DBMS zamykání dat při konkurenčním zápisu a zefektivňuje indexaci pro rychlejší vyhledávání ve strukturovaných datech.

Jednotka databázového dotazu je **transakce**. DBMS musí zaručit plnou spolehlivost provedení všech transakcí. S konceptem *ACID* přišli poprvé Haerder a Reuter (1983), kteří jej postavili na rozšířené práci Jima Graye (Gray 1981).

**ACID** je skupina vlastností, které by všechny databázové transakce měly splňovat pro zaručení validity dat i při neočekávané chybě. Slovo ACID (Atomicity, Consistency, Isolation, Durability) je zkratkou čtyř základních vlastností. Atomicita (**A**tomicity), Konzistence (**C**onsistency), Izolace (**I**solation) a Trvalost (**D**urability). (Haerder a Reuter 1983)

### Relační databáze

Databáze fungující na principu relačního datového modelu. Hodnoty uložené v relačních databázích jsou vždy atomické a skalární. Nemohou obsahovat dílčí subhodnoty. Pro modelování vícerozměrných dat se používají tzv. **relace**, abstraktní koncept vztahů vytvořených pomocí cizích klíčů a tabulek. (Vostrovský et al. 2004, s. 16)

Pro tyto databáze je typická též tzv. **normalizace**, *postupný reverzibilní proces nahrazování dané množiny relací souhrnem relací, které mají jednodušší a současně i „regulérnější" strukturu. Tento proces zjednodušování je založen na nestatickém kritériu, přičemž reverzibilita zaručuje, že původní souhrn informací lze kdykoliv obnovit, neboť nedochází ke ztrátě informací.* (Vostrovský et al. 2004, s. 112)

Příkladem DBMS, které pracují s relačními databázemi jsou Oracle, MySQL a PostgreSQL. Nejčastěji podporovaným jazykem pro tvorbu dotazů je jazyk SQL (Structured Query Language).

#### NoSQL databáze

Relativní novinka na poli databází. Díky nepoužívání relací a pevné struktury tabulek, umí přijímat a pracovat i s nestrukturovanými daty. Používají se nejčastěji v případech, kdy je potřeba rychle ukládat velké množství dat, u něhož není nutně pevně známa struktura. (Fowler 2012) Například při skladování dat z IoT (angl. Internet of Things, čes. Internet věcí) nebo ukládání realtime příspěvků na sociálních sítích.

Často zmiňovanou výhodou NoSQL databázových strojů je též podpora **sharding**. Metody horizontálního škálování výkonu stroje rozložením kolekcí dat napříč několika servery, které spolu navzájem komunikují. (MongoDB nedatováno)

Zástupci NoSQL DBMS jsou MongoDB, Redis, Amazon DynamoDB.

### 3.4.4.2 Cloud

Princip výpočetních prostředků provozovaných jako služba a dostupných vzdáleně skrze síť internet. Hlavní výhodou je absolutní minimalizace nákladů na provoz a správu vlastního serverového hardwaru. V určitých případech lze hovořit i o snazším a levnějším provozu podnikových aplikací obecně.

#### Integrace jako služba (angl. Infrastructure as a service, IaaS)

Poskytovatelé IaaS nabízejí počítačové prostředky, jako jsou servery, sítě a úložiště, za poplatek svým zákazníkům. Zákazníci těchto služeb jsou většinou technologicky zaměřené firmy, které potřebují plnou kontrolu nad systémem a prostředky serveru, ale nechtějí investovat do správy vlastního datacentra. Protože si zákazník neplatí nutně za přesně specifikovaný hardware, ale jen za potřebné prostředky (počet jader, frekvence procesoru, kapacita úložiště), je pro něj snazší ušetřit a platit jen za prostředky, které nutně potřebuje.

#### Platforma jako služba (angl. Platform as a service, PaaS)

Vzdálené prostředí, ve kterém je možné provozovat vlastní aplikace. Oproti IaaS není možná plná kontrola nad systémem a přesnými specifikacemi hardwaru. PaaS v posledních letech zaznamenalo velký boom díky příchodu tzv. kontejnerů. Ty umožňují snadné vytvoření aplikačního balíčku používající přesně specifikovatelné izolované softwarové prostředí (knihovny, nástroje, adresářová struktura) bez nutnosti vytváření kompletního virtuálního stroje.

**Software jako služba (angl. Software as a service, SaaS)**

Vzdálený provoz instance nějakého softwaru, plně po technické stránce spravovaný provozovatelem služby. Oproti IaaS a PaaS nabízí nejvyšší škálovatelnost a dostupnost. Příkladem může být Microsoft Office 365 a z podnikových systémů monday.com.

## 3.5 Webové technologie

Jedná se o technologie přímo zodpovědné za funkčnost služby WWW (World Wide Web) a provoz webových stránek.

### 3.5.1 WWW

> The World Wide Web (WWW, or simply Web) is an information space in which the items of interest, referred to as resources, are identified by global identifiers called Uniform Resource Identifiers (URI).
>
> The World Wide Web (též WWW nebo jen Web) je prostor informací, v němž položky zájmů, nazývané zdroje, jsou identifikovány pomocí globálních identifikátorů nazývaných Uniform Resource Identifier (URI)
>
> *(W3C Technical Architecture Group 2004)*

Umožňuje standardizované prohlížení a tvorbu, navzájem propojitelných, dokumentů skrze síť internet. Funguje na principu klient/server. Klientem je webový prohlížeč a serverem je nějaký http server. Prohlížeč se serverem komunikuje skrze protokol HTTP (Hypertext Transfer Protoco). Dokumenty jsou adresované pomocí adresy URL (Uniform Resource Locator). (Masner 2020c, s. 2)

### 3.5.2 Statický web

Nejstarší a nejjednodušší způsob publikace webového obsahu. Autor webové stránky vytvoří sadu html dokumentů, kaskádových stylů, mediálního obsahu, kterou umístí na svůj webový server. Pokud je webový server běžící a přístupný skrze veřejnou adresu, webová stránka je ihned dostupná pro kohokoliv s přístupem k internetu. Po načtení adresy stránky získá koncový uživatel kopii dokumentu přesně ve stejné podobě, v jaké je uložen na serveru. Takovým souborům, které nejsou generovány dynamicky, se říká **statické soubory**. (Rinaldi a Safari 2015)

### 3.5.3 Dynamický web

Plně statické webové stránky byly téměř zcela nahrazeny dynamickými stránkami. Ty můžeme nazvat příslušníky tzv. **dynamického webu**. Dynamické dokumenty nejsou uloženy na serveru ve stejné podobě, jako jsou zobrazeny ve výsledné podobě ve webovém prohlížeči. Místo toho jsou generovány až podle požadavků ze strany prohlížeče. Generovány jsou buď již přímo na serveru a prohlížeč obdrží statický dokument, na straně klienta pomocí javascriptu, nebo případně i kombinací obou předchozích možností. (Masner 2020b, s. 5)

Nevýhodou jsou obecně vyšší nároky na výkon stroje zodpovědného za generování výsledného dokumentu a komplexnější architektura celkové aplikace.

#### 3.5.3.1 SPA (Single Page Application)

Poměrně aktuální novinkou je nástup webových aplikací fungujících na modelu SPA (Single Page App), tedy tzv. modelu Jednostránkových Aplikací. Maximum kódu běží v uživatelově lokálním prohlížeči. Server v roli backendu se stará pouze o obsluhu dat z databáze a zprostředkování statických souborů, jako jsou kaskádové styly, média a v prohlížeči spustitelný javascript. (Mikowski a Benson 2014, s. 1--5)

Výhodou je rozdělení nároků na výkon mezi server a prohlížeč a možnost plného oddělení kódu pro backend a frontend, včetně jejich nasazení. (Mikowski a Benson 2014, s. 20)

Nevýhodou je větší velikost počátečního načtení aplikace a díky lokálnímu generování/vykreslování jsou hardwarové nároky vyšší než u čistě serverově vykreslovaných aplikací.

#### 3.5.3.2 Izomorfní aplikace

Druh aplikací, které jsou spustitelné jak v prohlížeči, tak na straně serveru. Protože je hlavním frontendovým jazykem JavaScript, jsou izomorfní webové aplikace někdy též definovány jako aplikace, které mají klientskou i serverovou část psanou právě v tomto jazyku. (AirbnbEng 2017, s. 17)

Hlavní výhodou je snadnější podmíněné serverové renderování interaktivních JavaScriptových částí aplikace a díky tomu lepší optimalizace pro vyhledávače. (Gordon 2016) A oproti aplikacím, které kombinují různé klientské a serverové jazyky, mizí duplicita kódu, potřebného na obou stranách síťové barikády. Formuláře a datové třídy jsou per-

fektním příkladem kódu, který je běžně nutný ručně synchronizovat mezi serverovým a klientským projektem. S izomorfní aplikací tento problém mizí a obě části aplikace mohou používat tytéž funkce a třídy.

#### 3.5.3.3 Progressive Web Apps (PWA)

> Progressive Web Apps are web applications that have been designed so they are capable, reliable, and installable.
>
> Webové aplikace vytvořené tak, aby byly schopné, spolehlivé a instalovatelné.
>
> *(Richard a LePage 2020)*

**Schopné**: PWA (Progressive Web App) aplikace umí díky moderním webovým API prohlížečů pracovat i s nativními funkcemi zařízení, na němž běží. Pomocí PWA lze například napsat alternativu k systémové aplikaci Kamera, která bude mít přístup k fotoaparátu, poloze pomocí GPS (Global Positioning System) a moci ukládat soubory přímo do zařízení.

**Spolehlivé**: Progresivní webové aplikace by měly být schopné fungovat rychle a spolehlivě bez ohledu na kvalitu internetového připojení. Díky využití webového API *service workers*, mohou po prvotním načtení fungovat progresivní aplikace plně offline bez jakéhokoliv internetového připojení a při přepnutí do režimu online se jen zaktualizují.

**Instalovatelné**: Aplikace, která je *nainstalována*, obdrží své vlastní místo v menu aplikací. Po spuštění je otevřena v samostatném okně nezávisle na jiných záložkách prohlížeče. Jako jakákoliv jiná nativní aplikace může přijímat a odesílat obsah skrze metodu sdílení v daném operačním systému. (Richard a LePage 2020)

#### 3.5.3.4 REST (Representational State Transfer)

Celým názvem *Representational State Transfer* je množina návrhových kritérií pro uniformnější implementaci webových služeb. RESTové služby umožňují přistupovat a manipulovat s textovou reprezentací dat zveřejněných danou službou. Pro určení druhu operace, která má být provedena, jsou použity HTTP metody GET, POST, PUT, PATCH a DELETE.

REST pracuje s konceptem Zdrojů a Metod. Zdroj je identifikován svou HTTP adresou a metoda specifikuje, co s ním má být provedeno. Výsledek operace by měl být navrácen skrze stavový kód HTTP. Nepřikazuje specifický serializační formát ani přesnou strukturu adresy. (Richardson a Ruby 2007, s. 79--105)

#### 3.5.3.5 GraphQL

S nástupem architektury SPA začal být REST pro účely frontendové Ajax komunikace nedostačující.

Jde o dotazovací jazyk, který komunikuje skrze specifický endpoint serveru a své dotazy přenáší pouze skrze GET a POST metody. Podobně jako v SQL může tvůrce dotazu vyvolat specifickou operaci na straně serveru nad specifickými daty a nechat si vrátit jen určité části výsledku. (Eschweiler 2018)

Na straně klienta nabízí velkou flexibilitu a efektivnost. Právě kvůli své flexibilitě může ale být náročné k implementaci na straně serveru a vést teoreticky ke snížení propustnosti API.

### 3.5.4 Server side

#### 3.5.4.1 Technologie

Webové aplikace lze psát téměř v jakémkoliv programovacím jazyku. Přesto se některé k webovému vývoji hodí více, než ostatní.

Protokol **HTTP**, po němž je vedena komunikace Web je protokolem bezstavovým. To znamená, že sám HTTP server si neudržuje stav tvůrců požadavků. Každý HTTP požadavek může fungovat samostatně. Při potřebě udržovat stav sezení je do požadavku přidán nějaký identifikátor a stav je načten až aplikační logikou. (Dwyer 2020) S takovou architekturou je možné každý jednotlivý požadavek odbavit v různém kontextu. To usnadňuje škálování a potenciálně může ochránit před útoky na alokovanou paměť jiných sezení.

Serverová část aplikace může být vývojářem řešena celá, nebo může být použit nějaký **aplikační server**. Jde o softwarový nástroj, který je zodpovědný za zprostředkování platformy pro běh aplikace. Aplikační server poskytuje aplikaci knihovny pro práci s prostředím platformy a navíc přidává vlastní služby jako například administrátorskou konzoli a logování. (Hanel 2009, k. 2.2)

### Java

Jazyk a platforma se zaměřením na crossplatformní běh (funkční na více operačních systémech a architekturách). Dociluje toho pomocí překladu zdrojového kódu do vlastního nízkoúrovňového jazyka, který je poté spuštěn v Java VM (Virtual Machine), zjednodušeném virtuálním stroji interpretujícím přeložený kód. (Gosling a McGilton 1996, k. 1)

Webové aplikace psané v jazyku Java je možné provozovat jak samostatně, tak skrze aplikační server.

### PHP

PHP (PHP: Hypertext Preprocessor) původně vzniklo jako hypertextový preprocesor, ale postupem času se díky jeho jednoduchosti a navázané popularitě proměnilo v plnohodnotný serverový jazyk. Dnes PHP umí komunikovat s databázemi, pracovat se soubory, paralelizovat výpočty pomocí nativních vláken nebo spouštět vlastní podprocesy. (PHP Documentation Group 2021)

### Python

Interpretovaný programovací jazyky s dynamickou sémantikou. Je známý svou přehlednou syntaxí a rozsáhlým ekosystémem existujících knihoven. (Python Software Foundation nedatováno)

### .NET

Framework a vývojářská platforma od firmy Microsoft. Oficiálně podporuje běh aplikací napsaných v jazycích C#, F# a Visual Basic. Za primární jazyk platformy byl ale vždy považován jazyk C#, pro který je .NET brán za doporučenou platformu pro jeho překlad a běh. (Baxter 2016)

### Go

Nejnovější jazyk na tomto seznamu backendových technologií. Go bylo oznámeno v roce 2009 společností Google jako experimentální open source programovací jazyk. Kombinuje rychlost a bezpečnost kompilovaných jazyků jako C++, s rychlostí vývoje dynamický jazyků jako Python. (Kincaid 2009)

#### Node.js

Běhové prostředí pro spuštění JavaScriptového kódu mimo prohlížeč. Obsahuje běhové prostředí V8 a nabízí nad ním ještě vlastní standardní knihovnu funkcí, které načtený kód může používat. Bez této knihovny by nebylo možné pracovat s lokálními soubory nebo naslouchat na síťových portech. Protože v samotném JavaScriptu, vyvinutém pro prohlížeč, žádné takové funkce nemohou být, jen za pomoci sebe sama, implementovány. (Patel 2018)

Stejně jako JavaScript obsahuje podporu pro konkureční odbavování událostí za pomocí smyčky událostí (angl. event loop). Díky tomu je možné při čekání na dokončení nějaké IO (Input-Output) operace, bez blokování hlavního vlákna, přepínat mezi kontexty konkurenčně spuštěných rutin. Nejedná se o klasické paralelní zpracování, ale v kontextu síťových operací, při kterých procesor sám pouze čeká na dokončení operace z jiného zdroje, jde o velmi efektivní řešení. (Patel 2018)

Protože používá V8, které je použito též v jádře webového prohlížeče Chromium/Google Chrome, udržuje Node.js krok s aktuálním standardem ECMAScript.

Kromě smyčky událostí je Node.js skvělá volba pro webové aplikace z důvodu možnosti tvořit frontend i backend ve stejném jazyku, JavaScriptu. Je možné používat stejné knihovny, stejné datové třídy, stejné nástroje.

Podobně jako Python má Node.js též vlastní oficiální balíčkový manažer pro správu externích komunitních modulů, NPM (Node.js Package Manager). NPM používá v základu stejnojmenný repozitář, který je brán, podobně jako PyPi, jako standardní oficiální repozitář, kde je hostována drtivá většina komunitních balíčků. V době psaní podle npm (2021b) obsahuje npm 1493231 balíčků, čímž je největším repozitářem softwaru na světě. (npm 2021a) Podobně tomu je i na službě GitHub, kde je již několik let JavaScript nejpopulárnějším jazykem napříč všemi repozitáři. (Zapponi Q4 2020)

### 3.5.4.2 Frameworky

#### Spring

Skupina projektů pokrývající drtivou většinu odvětví vývoje webových Java, Kotlin a Apache Groovy aplikací. Nejedná se o jeden klasický ucelený framework, ale spíše o rodinu modulů, které spolu v některých případech mohou či nemohou spolupracovat. (VMware, Inc nedatováno b)

Nejjednodušším příkladem pro snadné a rychlé vytvoření webové aplikace je **Spring Boot**. Ten nabízí předkonfigurované řešení s okamžitou možností svou aplikaci sestavit, spustit a publikovat na jeden ze tří možných vestavěných aplikačních serverů

(např. Tomcat). Nevyžaduje žádnou komplikovanou konfiguraci a je přednastavený od začátku pro integraci s několika dalšími Spring moduly. (VMware, Inc nedatováno a) Pro vygenerování počáteční struktury projektu lze použít oficiální webová aplikace start.spring.io, kde si vývojář může interaktivně zvolit požadavky nového projektu a je mu poté vygenerován zip archiv s připravenou kostrou projektu.

```
@SpringBootApplication
@RestController
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    @GetMapping("/hello")
    public String greeting() {
        return String.format("Hello world!");
    }
}
```

Ukázka kódu 1: Odbavení GET /hello v Spring

### Laravel

Webový framework založený na jazyku PHP. Obsahuje moduly pro práci s databází, injektáž závislostí i unit testování. Snaží se přivést moderní vývojové trendy do PHP světa. Protože ne všechny webové aplikace potřebují funkce celého frameworku, umožňuje Laravel používat jen některé své části a to i samostatně. Při vývoji je tedy možné použít i třeba jen Laravel Router a nechat komunikaci s databází na modulu z úplně jiného frameworku. (Laravel LLC nedatováno)

```
Route::get("/hello", function () {
    return "Hello world!";
});
```

Ukázka kódu 2: Odbavení GET /hello v Laravel

### Django

Vysokoúrovňový framework pro jazyk Python, navrhnut podle vzoru MVT (Model, View, Template). Snaží se být tzv. „batteries included“, tedy být ihned od začátku schopen fungovat, a obsahuje všechny funkce už v základním modulu. Dokáže obstarat vše od databázové migrace, dotazování, validace vstupu až po vykreslení frontend šablon.

Latku bezpečnosti se snaží udržovat pomocí vestavěných metod pro validaci vstupů a autentizaci. Vývojář se nemusí starat o práci s cookies ani o hashování a kontrolu uživatelského hesla.

Oproti předchozím frameworkům obsahuje v základu i funkční administrátorské rozhraní, skrz které může správce aplikace spravovat databázi aplikace. (MDN contributors nedatováno a)

```python
# urls.py
urlpatterns = [
    path("hello", views.greeting)
]

# views.py
def greeting(request):
    return HttpResponse("Hello world!")
```

Ukázka kódu 3: Odbavení GET /hello v Django

### ASP.NET

Postaven na .NET, přidává integrovanou podporu pro práci s webovými požadavky, vlastní šablonovací jazyk pro tvorbu dynamických stránek a automatizovaný autentizační systém. Podobně jako Spring je postaven na návrhovém vzoru MVC (Model-View-Controller). (Microsoft nedatováno b)

Do roku 2016 byl dostupný pouze pro systémy Windows. To se změnilo s příchodem **.NET Core** a nyní je **ASP.NET** ve verzi **Core** podporován na všech třech hlavních platformách. Linux, Windows, macOS. (Microsoft nedatováno a)

```csharp
[Route("/hello")]
[ApiController]
public class GreetingController : ControllerBase
{
    [HttpGet]
    public ActionResult<string> Get()
    {
        return "Hello world!";
    }
}
```

Ukázka kódu 4: Odbavení GET /hello v ASP.NET

### Express

Express je aktuálně nejpopulárnějším a nejpoužívanějším frameworkem postaveným na platformě Node.js. Kvůli svému relativnímu minimalismu je sám jádrem dalších frameworků, které jej používají pro abstrakci webové http komunikace s klienty.

Přestože je Express minimalistický a práce s ním se částečně podobá práci s čistým Node.js, má díky své jednoduchosti a podpoře pro tzv. **middlewares** (česky možno přeložit jako „prostředník“) obrovskou komunitní podporu z hlediska možných integrací. (MDN contributors nedatováno b) Příkladem může být modul passport zjednodušující autentizaci webových žádostí. (McKinney et al. 2017)

Oproti ostatním frameworkům nenabízí Express validaci vstupů ani práci s databází. To ale může být pro zkušenější vývojáře výhodou, protože na vývojáře netlačí žádné vlastní paradigma. Dle loňské ankety The State of Javascript je Express nejoblíbenějším backendovým frameworkem. Více než 71% dotazovaných odpovědělo, že s Expressem mají zkušenosti a použili by ho dobrovolně i pro své budoucí projekty. (Greif a Benitte 2019)

```
const app = express();

app.get("/hello", (req, res) => {
  res.send("Hello world!")
});

app.listen(3000);
```

Ukázka kódu 5: Odbavení GET /hello v Express

### routing-controllers

Routing-controllers je nástroj pro vygenerování handlerů cest pro frameworky Express a Koa. Vývojář napíše své controllery za pomoci jazyka TypeScript a odekoruje své controllery pomocí dekorátorů této knihovny. Ta poté dokáže z tohoto kódu, v kombinaci ještě s projekty class-transformer a class-validator, získat informace pro validaci vstupů a vytvořit funkční Express/Koa instanci. (typestack 2021)

Při kombinaci s balíčkem routing-controllers-openapi je možné za běhu vygenerovat z běžící instance specifikaci OpenAPI pro popis vnějšího HTTP API webové aplikace. Oproti čistému Express tak vývojář získává statické typování skrze TypeScript, automatickou validaci vstupů a možnost nechat si na základě vygenerované OpenAPI vygenerovat například klient pro napsané API nebo automatické testy. (APIs You Won't Hate nedatováno)

Svou syntaxí je podobný NestJS a ASP.NET.

```
@JsonController("/hello")
export class GreetingController {
    @Get()
    greeting() {
        return "Hello world!"
    }
}
```

Ukázka kódu 6: Odbavení GET /hello pomocí routing-controllers

### NestJS

Framework celý postavený na možnostech jazyka TypeScript, podobně jako routing--controllers. Oproti routing-controllers ale nabízí mnohem více integrovaných funkcí. Například pokročilejší **injektáž závislostí** (angl. dependency injection), pomocné metody pro integraci externího ORM (Object-Relational Mapping) řešení, vlastní způsob loggování událostí a cachování odpovědí. (Mysliwiec nedatováno a)

Je postaven na podobných vzorech jako Angular. Pro označení hierarchie aplikace používá moduly a injektovatelné služby. (Mysliwiec nedatováno b)

```
@Controller("/hello")
export class GreetingController {
    @Get()
    greeting() {
        return "Hello world!";
    }
}
```

Ukázka kódu 7: Odbavení GET /hello v NestJS

### Feathers

Feathers se snaží odstranit co nejvíce přebytečného kódu, který se opakuje mezi většinou webových aplikací. Zároveň se ale snaží být i flexibilní pro případ, že by vývojář potřeboval mít bližší přístup k fungování komunikace. (Kryski 2017)

Hlavním prvkem, kolem kterého je Feathers vybudováno, jsou **služby** (angl. services). Ty v prostředí Feathers označují skupinu funkcí, které zprostředkovávají CRUD (Create, Read, Update, Delete) operace nad nějakým modelem. Tyto služby poté dokáže Feathers samo zaobalit a publikovat skrze RESTové rozhraní nebo websockets. Oproti ostatním zmiňovaným Node frameworkům nabízí navíc i vlastní ORM pro integraci s více než 15 druhy datových úložišť. (Kryski 2018)

```
class GreetingService {
    async find() {
        return "Hello world!";
    }
}

const app = express(feathers());
app.configure(express.rest());

app.use("/hello", new GreetingService())

app.listen(3000);
```

Ukázka kódu 8: Odbavení GET /hello v Feathers

### Next

Framework pro tvorbu izomorfních React aplikací. Zaměřuje se především na podporu frontendové části aplikace. Umí automaticky stránky vykreslovat dynamicky na serveru (SSR - Server-Side-Rendering), vykreslit staticky dopředu (SSG) nebo i stranu předkreslit pro rychlé prvotní načtení. Klient poté už vykresluje sám a na server dopadají pouze dotazy na dodatečné statické soubory a API. (Vercel nedatováno b)

Routování, tedy směrování uživatelského webového dotazu na korespondující controller/stránku, je řešeno pomocí ukládání souborů endpointů do specifické adresářové struktury. Soubor `pages/hello.js` by byl například použit pro každý HTTP dotaz na cestě `/hello`. (Vercel nedatováno c) Pomocí názvů složek a souborů se dají parsovat i části cesty. Pro pozdrav určitého uživatele by například šlo vytvořit soubor `pages/hello/[name].js` s obsahem:

```
import { useRouter } from "next/router";

export default function() {
    const router = useRouter();

    return <p>Hello {router.query.name}</p>;
}
```

Ukázka kódu 9: Odbavení GET /hello v Next

Next podporuje i správu čistého HTTP API. V takovém případě je nutné vytvořit soubor s handlerem chtěného endpointu v adresáři `pages/api/` v kořeni webové aplikace. Takové soubory by měly obsahovat jedinou exportovanou funkci, označenou heslem *default*, která přijímá argumenty s HTTP žádostí a odpovědí. Podobně jako standardní struktura handlerů v Express. (Vercel nedatováno a)

#### Nuxt

Alternativa Next ve světě Vue. Podobně jako Next umožňuje vytvářet izomorfní aplikace a používá adresářovou strukturu pro automatické vygenerování routeru. (Berning 2018)

Oproti Next nemá možnost automatického načítání handlerů z určité složky. Místo toho umožňuje vytvářet celoaplikační nebo celostránkové middlewary. Lze například přímo v souboru stránky specifikovat funkci, která se spustí s parametry kontextu na straně serveru ještě před načtením na straně klienta. (Chopin nedatováno) Taková funkce může například zamezit načtení stránky pro neautorizované uživatele nebo být použita pro analýzu návštěvnosti.

```
<!-- pages/hello/_name.vue -->
<template>
    <p>Hello {{ this.name }}</p>
<template>

<script>
export default {
    async asyncData({ params }) {
        return {
            name: params.name
        }
    }
}
</script>
```

Ukázka kódu 10: Odbavení GET /hello v Nuxt

### 3.5.5 Client side

#### 3.5.5.1 Technologie

#### HTML (Hypertext Markup Language)

Výchozí značkovací jazyk pro popis struktury webové stránky.

Svou syntaxí je podobné XML (Extensible Markup Language), ale obsahuje jen specifickou podmnožinu jasně definovaných značek. Též oproti XML povoluje liberálnější přístup k uzavírání značek a jejich křížení. (Masner 2020c)

Od verze **HTML5** lze například nativními značkami popisovat vkládání multimédií, validovat formulářový vstup pomocí regulárních výrazů nebo pracovat s virtuálním plátnem `<canvas>`. Mimo interaktnivní prvky přibylo i několik značek sloužících pro snažší semantizaci struktury obsahu. Vývojář tak může například navigační menu označit

značkou `<nav>`, hlavičku/patičku pomocí `<header>/<footer>` a hlavní část obsahu pomocí `<main>`. Díky sémantizaci dokáží webové vyhledávače a nástroje pro usnadnění přístupu lépe porozumět obsahu stránky. (MDN contributors nedatováno c)

### CSS (Cascading Style Sheets)

Stylovací jazyk používaný pro stylování HTML dokumentů. Může být obsažen přímo v HTML dokumentu pomocí značky `<style>`, atributu `style` nebo načten ze samostatného externího souboru. Stejně jako HTML je spravováno skupinou W3C.

Díky jeho existenci je možné oddělit semantickou strukturu dokumentu od jeho vzhledu. (Masner 2020a)

### JavaScript

Výchozí skriptovací jazyk pro tvorbu interaktivních webových dokumentů.

Při standardizaci nebyl zvolen název *JavaScript* ale **ECMAScript** podle mezinárodní neziskové organizace Ecma International, která standardizaci provedla. ECMAScript je programovací jazyk, který formálně popisuje, jak by měl JavaScript fungovat a jaké funkce/možnosti nabízet. Běžně se ale s pojmy JavaScript a ECMAScript lze setkan ve stejném kontextu. JavaScript a ECMAScript tak v běžné řeči označují tentýž jazyk. (Haverbeke 2019, s. 18)

### Ajax

Původně označoval *Asynchronní JavaScript a XML* (angl. Asynchronous JavaScript and XML). Dnes jde hlavně o styl architektury, při níž jsou HTTP požadavky vytvářeny z klientského prostředí prohlížeče. Nemusí se jednat přímo o JavaScript ani o XML. Místo JavaScriptu může být jakýkoliv skriptovací jazyk, který prohlížeč dokáže interpretovat. Stejně tak XML může být nahrazeno jakýmkoliv formátem použitým pro zakódování informace do HTTP komunikace mezi serverovým jazykem a klientským. (Richardson a Ruby 2007, s. 315--316)

XML bylo formátem používaným pro Ajax komunikaci. Pro svou relativní neefektivitu z hlediska velikosti přenášených dat a náročnosti zpracování ve skriptu bylo pro přenos dat při Ajax voláních nahrazeno formátem **JSON (JavaScript Object Notation)** (angl. JavaScript-Object-Notation, čes. JavaScriptový-Objektový-Zápis). Jde o serializační formát vhodný pro popis jakékoliv objektové datové struktury. Je mnohem úspornější na počet potřebných znaků k zakódování informace a protože nepodporuje žádné pokročilé funkce jako komentáře nebo externí odkazy, je též snazší ho parsovat. (Richardson a Ruby 2007, s. 266)

#### 3.5.5.2 Frameworky

Ve spojení s Ajax a SPA je jedním z hlavních úkolů frontendové aplikace zobrazovat strukturovaná data ze serveru a udržovat je aktuální podle uživatelských akcí. Pro vykreslování a správu zobrazovaných informací již klasické imperativní způsoby programování nestačily a moderní frontendové frameworky se postupně uzpůsobily stylu **reaktivního programování**. Ten je podmnožinou stylu deklarativního a umožňuje vývojáři deklarovat datové procesy, které poté částečně fungují již bez potřeby přesné specifikace včech jednotlivých kroků. (Harutyunyan 2019)

U webové stránky reaktivita zjednodušuje práci s DOM (Document Object Model), kdy není nutné při změně stavu dat DOM ručně aktualizovat. Framework to udělá sám.

##### React

Vyvinut společností Facebook pro usnadnění práce na vlastních projektech. Oproti Vue a Angular není považován za klasický framework ale spíše za obecnou knihovnu použitelnou ke tvorbě znovupoužitelných reaktivních UI (User Interface) komponent. React u komponent též částečně zařizuje správu atributů a stavu. Při jejich změně zavolá metodu `render()`, porovná novou podobu komponenty s aktuálním stavem DOM a aktualizuje jen ty části stránky, které se liší od předchozího stavu. (Rascia 2018)

Hodí se i pro malé projekty. Pro svou jednoduchost by ale mohl být paradoxně náročný pro nové vývojáře, protože neobsahuje vše, co by mohl vývojář očekávat a při přehlédnutí hrozí ztráta reaktivity (setState).

```
class ReactCounter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  increment = () => this.setState({count: this.state.count + 1});
  decrement = () => this.setState({count: this.state.count - 1});

  render() {
    return (
        <div className="react-counter">
            <button onClick={this.decrement}> - </button>
            <span>
                Counter has been pressed: {this.state.count} times.
            </span>
            <button onClick={this.increment}> + </button>
        </div>
    );
  }
};
```

Ukázka kódu 11: Jednoduchý reaktivní čítač v React

### Angular

Nástupce AngularJS od společnosti Google. Rozsáhlý framework a platforma pro tvorbu velkých spravovatelných SPA aplikací za pomoci HTML a jazyku TypeScript.

Angular běžně počítá s komponentou složenou ze tří souborů. Klasického souboru HTML popisujícím strukturu HTML obsah komponenty, CSS souboru se styly pro danou komponentu a TypeScript souboru obsahujícím deklaraci referencí na předchozí dva soubory, použité funkce, proměnné a další logiku použité uvnitř komponenty.

Kompilátor při sestavování přemění HTML šablonu komponenty na optimalizovaný JavaScript, generující v prohlížeči potřebné DOM elementy bez potřeby komplexního parsování na straně prohlížeče. (Google nedatováno)

Pro malé projekty se může jevit jako příliš komplikované řešení. Je vhodný na velké projekty spravované vícero programátory.

```
<!-- counter.component.html -->
<div class="angular-counter">
    <button (click)="decrement()"> - </button>
        <span>
            Counter has been pressed: {{ count }} times.
        </span>
    <button (click)="increment()"> + </button>
</div>
```

Ukázka kódu 12: Jednoduchý reaktivní čítač v Angular (html)

```
/** counter.component.ts **/
@Component({
  selector: 'angular-counter',
})
export class AngularCounterComponent {
    count = 0;

    increment() { this.count += 1 }
    decrement() { this.count -= 1 }
}
```

Ukázka kódu 13: Jednoduchý reaktivní čítač v Angular (logika)

### Vue

Za Vue stojí čínský programátor Evan You. Ten Vue vytvořil při své práci v Google jako alternativu pro Angular, který mu tehdy přišel příliš těžkopádný pro jeho vlastní účely. Vue tedy vzniklo jako extrakt dobrých vlastností Angularu smíchaných s Evanovými vlastními nápady. Evan jej později zveřejnil a Vue se docčkalo překvapivého úspěchu. (Cromwell 2018) Od roku 2014, kdy bylo Vue otevřeno veřejnosti se stalo nejpopulárnějším frontendovým frameworkem na síti Github a dnes jde celkově o třetí nejpopulárnější projekt na platformě. (Li 2021)

Dle Evana You je mezi frameworky Vue nejpodobnější Reactu, ale s důrazem na progresivitu. Jádro Vue je jednoduché a skládá se z komponent a vázání dat (angl. data binding). Oproti Reactu nabízí Vue lineárnější křivku poznání, kdy vývojáři ze začátku mohou stačit jen základní principy frameworku a postupem času a narůstající komplexitou projektu může použít pokročilejší možnosti. Vue je částečně rozděleno do několika oficiálních modulů, které vývojář může či nemusí použít ve svém projektu. Nástroj pro správu projektu a jeho transpilaci je spravován zvlášť, stejně tak router a správce globálního stavu. Přesto jsou ale oficiální součástí projektu a mají tak stejnou oficiální podporu, jako jádro Vue. (Cromwell 2018)

Jednou ze specifických vlastností Vue jsou **computed properties**, česky přeložitelné jako *odvozené atributy*. Jde o atributy, které svou hodnotu samy dopočítávají podle předpisu zadaného vývojářem. Oproti klasické proměnné si tyto atributy samy hlídají změny stavu proměnných a atributů použitých v jejich předpisu. Koncept je podobný jako u getterů tříd v C# nebo ES6, ale je navíc optimalizovaný tak, aby si atribut přepočítal svůj stav pouze tehdy, pokud se změnila nějaká z jeho závislostí. Může být použit a volán mnohokrát, ale pokud se žádná z jeho závislostí nezměnila, nebude jeho předpis znovu přepočítán. Při správném použití vedou computed properties ke znatelnému navýšení responzivity aplikace. (vuejs nedatováno a)

Vue lze používat i bez instalace lokálního transpilátoru. Pro své základní fungování si v prohlížeči vystačí s načtením z CDN (Content Delivery Network) a komponenty dokáže připravit i z existujích DOM objektů nebo textového řetezce. V takovém případě je ale uživatel nucen načítat větší množství dat a Vue musí při každém načtení stránky znovu naparsovat definice komponent. (vuejs nedatováno b)

Pro pokročilejší aplikace je doporučeno použít vývojářské nástroje **Vue CLI** nebo nové **Vite**. Ty umí zpracovat zdrojové soubory aplikace, optimalizovat je a přeložit do verze spustitelné v prohlížeči. Výsledkem je potenciálně menší velikost výsledných souborů, rychlejší načtení prohlížečem a možnost používat pokročilé možnosti JavaScriptového ekosystému. (vuejs nedatováno b) Například import externích knihoven, transpilace z jiné verze jazyka (např. TypeScript), jednotkové testování, komprese a obfuskace výsledného souboru.

Jednou z možností, kterou tyto nástroje zpřístupňují, je také možnost psaní Vue komponent pomocí formátu **SFC** (Single-File-Components). Jde o formát jednotlivých komponent psaných v samostatných souborech s příponou `.vue`. Takto psané komponenty obsahují ve stejném souboru všechny tři části komponenty, HTML šablonu, styly a komponentovou logiku. Velkou výhodou oproti jiným způsobům deklarace je zpřehlednění. Není potřeba přepínat mezi kontexty různých souborů. Dále tento způsob zápisu přirozeně umožňuje aplikovat styly CSS pouze na kontext dané komponenty. Vývojář tak může stylovat interní prvky pomocí generických CSS selektorů, které by jinak normálně ovlivnili všechny odpovídající prvky na stránce. (vuejs nedatováno c)

```vue
<!-- Counter.vue -->
<template>
    <div class="vue-counter">
        <button @click="decrement"> - </button>
            <span>
                Counter has been pressed: {{ count }} times.
            </span>
        <button @click="increment"> + </button>
    </div>
</template>

<script>
export default {
    name: 'vue-counter',
    data: () => ({
        count: 0
    }),
    methods: {
        increment() { this.count += 1 },
        decrement() { this.count -= 1 }
    }
}
</script>
```

Ukázka kódu 14: Jednoduchý reaktivní čítač ve Vue 2

S příchodem verze Vue 3, přibyla ve frameworku nová syntaxe **Composition API** pro deklaraci komponentové logiky. Je inspirována konceptem *hooks* používaným v React. Vznikla pro čistší sdílení komponentové logiky napříč různými komponentami. Do komponent přináší metodu `setup`, v níž lze deklarovat reaktivní proměnné, funkce a pracovat s událostmi vyvolanými komponentou. (Findlay 2020) Z důvodu odlišné implementace je při používání Composition API i lépe podporováno statické typování při používání jazyku TypeScript.

```
<!-- Counter.vue -->
<template>
    <div class="vue-counter">
        <button @click="decrement"> - </button>
            <span>
                Counter has been pressed: {{ count }} times.
            </span>
        <button @click="increment"> + </button>
    </div>
</template>

<script>
export default {
    name: 'vue-counter',
    setup() {
        const count = ref(0);
        const increment = () => { count.value += 1 };
        const decrement = () => { count.value -= 1 };

        return {
            count,
            increment,
            decrement
        }
    }
}
</script>
```

Ukázka kódu 15: Jednoduchý reaktivní čítač ve Vue 3

# 4 Vlastní práce

## 4.1 Požadované funkce

Praktickým výstupem této práce by měl být webový podnikový systém, inspirovaný potřebami malé projekční kanceláře mého otce. Měl by umožňovat evidenci zakázek.

U každé zakázky musí být bezpodmínečně evidováno:

- číslo zakázky
- termín zahájení a dokončení
- zákazník
- sjednaná cena
- fáze zákazky
- aktuální stav

Pro zabezpečení dat a dodržení hierarchie interní struktury podniku musí systém nabízet alespoň základní způsob autentizace a autorizace. Nejlépe za pomoci zaměstnaneckých účtů seskupených do skupin s různými druhy oprávnění. Řadový zaměstnanec by například neměl mít přístup k záložce s financemi a nastavením serveru.

Aplikace by měla obsahovat pohodlné vyhledávání zakázek a měla by být dostupná odkudkoliv i skrze mobilní zařízení.

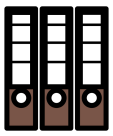
Dodatečné funkce mohou být implementovány volitelně a časem.

### 4.1.1 Celkový styl

Hlavním konceptem je jednoduchost. Jsem milovníkem skandinávské kultury, a tak jsem pro název projektu hledal slova se skandinávským původem.

Název projektu vznikl kombinací islandského slova *bóka*, slovesa s významem *zapisovat*, *registrovat*, *přísahat* (Cleasby et al. 1874, s. 74) a islandské přípony *-ri*, která se užívá k vytváření podstatných jmen ze sloves. (Neijmann 2012, s. 35) Výsledkem je slovo **Bókari** znamenající *zapisovatel*, *účetní* či *správce*. Slovo popisující velmi dobře cíl této práce. Zápis a archivace zakázek malé firmy.

Design grafického rozhraní je tvořen podle návrhů **Material Design** od společnosti Google. Jde o čistý systém designu, jehož cílem je vytvořit ucelený vzhled interaktivních aplikací napříč zařízeními a je inspirován chováním světla a textur v reálném světě. Jednotlivé prvky na stránce je možné si představit jako kusy papíru, které vrhají odpovídající stín a jsou na sobě skládány. (Google 2018)



Obrázek 4.1: Ikona aplikace

Paleta barev je zvolena tak, aby v uživateli podněcovala pozitivivní pocity. Světlé jasné barvy se světle hnědou jako primární barvou korespondující s ideí zápisových knih.


Obrázek 4.2: Paleta barev grafického rozhraní

## 4.2 Výběr technologie

### 4.2.1 Server

Server by měl být od klienta oddělen tak, aby bylo teoreticky možné vytvořit více implementací téže strany. Klient a server musí být na sobě co nejméně závislí a musí mezi nimi existovat nějaké specifikované API, které může být použito například jinou implementací klienta.

Pro specifikaci bude použit standard **OpenAPI**. Uznávaný formát pro popis RESTových API. (SmartBear Software nedatováno) Aby jej nebylo nutné vytvořit a udržovat aktuální ručně, bude použit nějaký z backend frameworků, který dokáže vygenerovat svou API specifikaci.

Na server byla zvolena technologie **Node.js** a framework **Express** s nástrojem **routing-controllers**. Umožní využít některé části kódu i později v klientské aplikaci a vyvarovat se některým komplikovaným konceptům z NestJS.

Server i klient budou psány jazykem **TypeScript**. Bez přílišných komplikací nabízí lepší vývojářský zážitek díky kontrole typování a podpoře nové ES (ECMAScript) syntaxe.

Data budou ukládána do relační databáze na databázovém stroji **PostgreSQL**. Je svobodný a nabízí velké množství funkcí. Alternativou byla SQLite, ale upozadila ji nedostatečná podpora pro modifikaci existujích tabulek a nemožnost správy databáze po síti. (Hipp a SQLite Development Team nedatováno)

K databázi bude přistupováno skrze dodatačnou vrstvu ORM knhovny **TypeORM**. Ta dokáže spravovat stav databázového stroje na základě definice entit pomocí TypeScript tříd a vlastních dekorátorů. TypeORM po navázání spojení s databází aktualizuje relační a entitní schéma dle lokálních entitních tříd. Pro produkční nasazení podporuje i pokročilejší správu migrací pomocí migračních kroků, které též udržuje automaticky v sycnhronizaci s databází vytvořením vlastní tabulky s migračními informacemi. Výhodou, oproti ručnímu psaní SQL dotazů, je získání lepšího typování kódu a nižší vázanost na specifický druh databázového stroje. (Khudoiberdiev nedatováno)

### 4.2.2 Klient

Klient bude realizován jako SPA aplikace napsaná ve **Vue** a **TypeScript**.

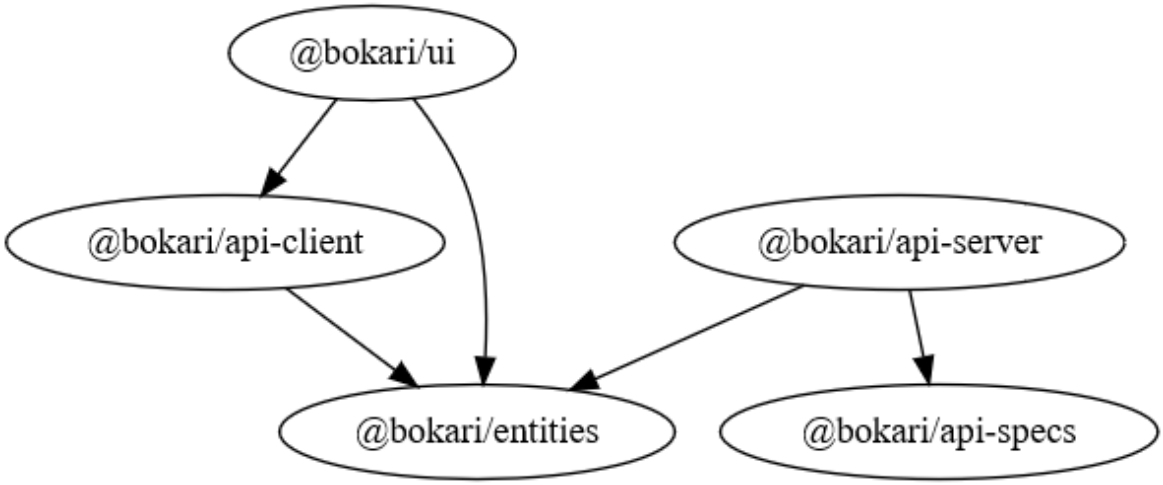
Vue bude zkombinováno s UI frameworkem **Vuetify**. Protože Vuetify zatím nepodporuje Vue verze 3, bude aplikace napsána s použitím Vue 2. Pro snadnější budoucí migraci na verzi 3 budou komponenty psány novou **composition api** syntaxí pomocí oficiální přechodné knihovny.

Pro vygenerování TypeScript služby pro komunikaci s REST API bude použit **OpenAPI Generator**, specificky jeho modul typescript-axios.

## 4.3 Struktura projektu

Protože většina zvolených technologií je závislá na Node.js ekosystému a je žádoucí sdílet kód mezi klientem/serverem, je projekt spravován jako jeden repozitář. Tomuto způsobu organizace se říká **monorepo** a se správnými nástroji umožňuje snadnější provázání jednotlivých modulů repozitáře spolu s jednoduchým celistvým verzováním. Pro jeho správu jsou použity nástroje **yarn** a **lerna**. (Weber 2019)

Byl vytvořen samostatný oddělený neveřejný balíček pro každou jednotlivou část projektu. Každá takto oddělená část má údaje o vlastních závislostech a způsobech jejího sestavení. Projekt je rozdělen celkově na pět, navzájem provázaných, balíčků.




Obrázek 4.3: Struktura vzájemných závislostí interních balíčků projektu

### 4.3.1 Databáze

V balíčku **@bokari/entities** se nachází veškeré entity s jejich TypeScript definicemi. Každý atribut je navíc odekorován dekorátory validace a případnými dekorátory sloužícími pro správnou transformaci JSON dokumentů na použitelnou entitní instanci a zpět.

```
import { Type } from 'class-transformer';
import { IsBoolean, IsDate, IsInt, IsString, Matches, ValidateNested } from 'class-validator';
import { Column, Entity, Index, ManyToOne, OneToMany, PrimaryGeneratedColumn } from 'typeorm';

@Entity()
@Index('contract_name_trgm', { synchronize: false })
export class Contract {
    @PrimaryGeneratedColumn()
    @IsInt()
    id!: number;

    @Column({ unique: true })
    @IsString()
    @Matches(/^\d{5}$/)
    code!: string;

    @Column()
    @IsString()
    name!: string;

    @Column({ nullable: true })
    @IsString()
    description?: string;

    @Column('timestamptz')
    @Type(() => Date)
    @IsDate()
    startAt!: Date;

    @Column('timestamptz')
    @Type(() => Date)
    @IsDate()
    deadlineAt!: Date;

    @Column({ default: false })
    @IsBoolean()
    isDone!: boolean;

    @Column(() => Metadata)
    @Type(() => Metadata)
    @ValidateNested()
    metadata!: Metadata;

    @ManyToOne(() => Customer, customer => customer.contracts, { eager: true, nullable: false })
    @Type(() => Customer)
    @ValidateNested()
    customer!: Customer;

    @Column(() => Monetary)
    @Type(() => Monetary)
    @ValidateNested()
    price!: Monetary;

    @OneToMany(() => ContractAttachment, attachment => attachment.contract, { cascade: true })
    @Type(() => ContractAttachment)
    @ValidateNested({ each: true })
    attachments!: ContractAttachment[];

    @OneToMany(() => WorkLog, workLog => workLog.contract)
    @Type(() => WorkLog)
    @ValidateNested({ each: true })
    workLogs!: WorkLog[];

    @OneToMany(() => ContractPhase, phase => phase.contract, { eager: true, cascade: true })
    @Type(() => ContractPhase)
    @ValidateNested({ each: true })
    contractPhases!: ContractPhase[];
}
```

Ukázka kódu 16: Definice entity Zakázka

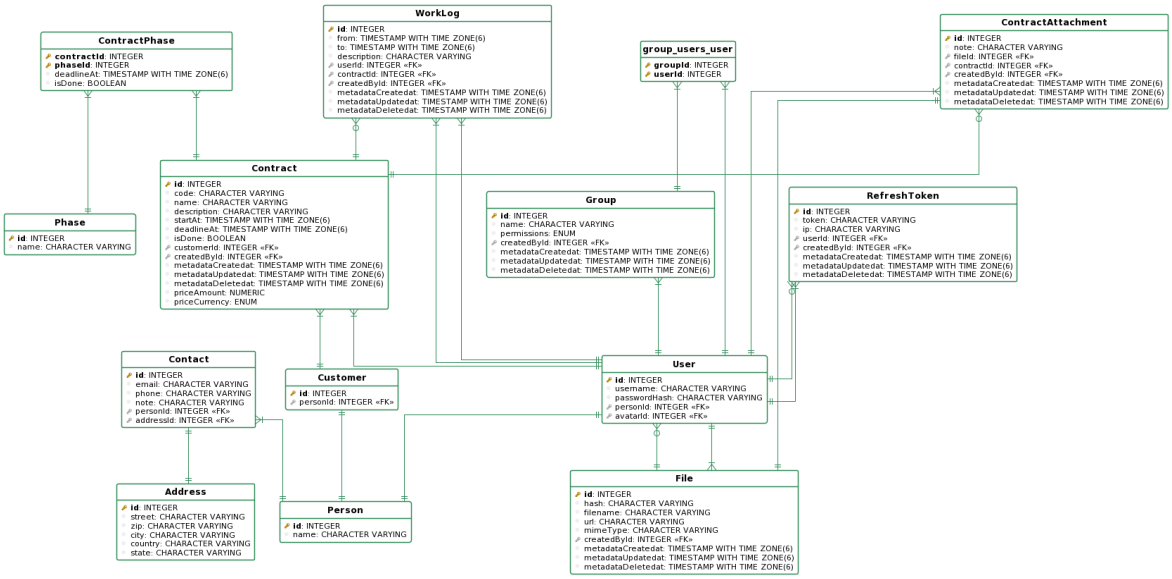
Konstrukce schématu probíhá při spuštění serveru ihned po navázání spojení s databázovým strojem. Pokud je tato funkce povolena (nastavením proměnné prostředí TYPEORM_SYNCHRONIZE=true), jsou s databází synchronizovány i veškeré soubory migrací. I přes aktuální nezávislost na plné integritě schématu jsou migrace použity pro automatické počáteční naplnění databáze základními daty. Pomocí migrace je též řešeno například vytváření indexů trigram pro efektivnější vyhledávání názvů zakázek a jmen osob.

```
import { MigrationInterface, QueryRunner } from 'typeorm';
import { Person, User } from '../../entities';

export const AdminUserSeed = new User({
    username: 'admin',
    passwordHash:
     ↪  '$argon2i$v=19$m=16,t=2,p=1$NWtuT3p3eWF2Q3UxSjRkdQ$OiLMvYNsxzMvXW/UcsAOUQ',
    person: new Person({ name: 'Admin' }),
    workLogs: [],
    refreshTokens: [],
    groups: []
});

export class SeedAdminUser1613643634132 implements MigrationInterface {
    public async up(queryRunner: QueryRunner): Promise<void> {
        await queryRunner.manager.getRepository(User).save(AdminUserSeed);
    }

    public async down(queryRunner: QueryRunner): Promise<void> {
        await queryRunner.manager.getRepository(User).delete({ username:
         ↪  AdminUserSeed.username });
    }
}
```

Ukázka kódu 17: Vytvoření účtu správce

```
import { MigrationInterface, QueryRunner } from 'typeorm';

export class PersonNameTrgm1613643612995 implements MigrationInterface {
    public async up(queryRunner: QueryRunner): Promise<void> {
        await queryRunner.query(
            `CREATE INDEX person_name_trgm ON person USING gin (name
             ↪  gin_trgm_ops)`
        );
    }

    public async down(queryRunner: QueryRunner): Promise<void> {
        await queryRunner.query(`DROP INDEX person_name_trgm`);
    }
}
```

Ukázka kódu 18: Vytvoření trigram indexu pomocí migrace




Obrázek 4.4: UML relační diagram výsledné databáze

### 4.3.2 API server

Server je zapouzdřen v balíčku **@bokari/api-server** a pro persistenci dat používá TypeORM a entity z @bokari/entities. Jeho zdrojový kód je napsán v jazyku TypeScript a jeho transpilace do spustileného jazyka je zajištěna oficiálním kompilátorem **typescript/tsc**.

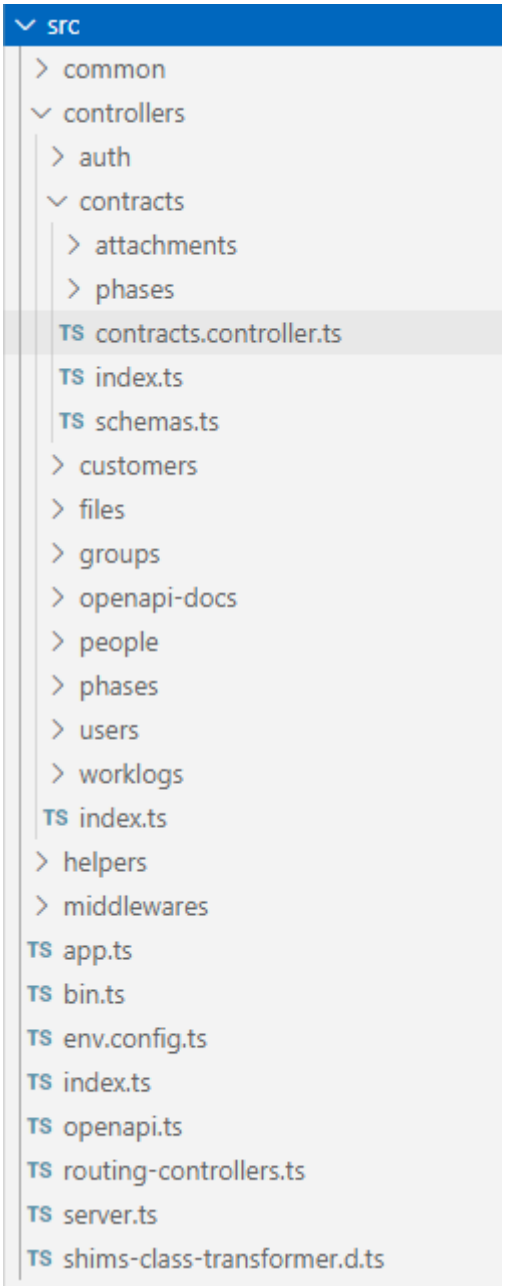
Údaje potřebné ke spojení s databází jsou načítány z proměnných prostředí. Bez pře--transpilace je tedy možné vyměnit jednu instanci databáze za druhou.

```
export class Server {
    private server!: http.Server;

    async listen(port = PORT, hostname = '0.0.0.0'): Promise<http.Server> {
        this.server = http.createServer(app);

        createTerminus(this.server);

        try {
            const connectionOptions = await getConnectionOptions();
            Object.assign(connectionOptions, {
                entities: Object.values(entities),
                migrations: Object.values(migrations)
            });
            await createConnection(connectionOptions);
        } catch (err) {
            console.error(
                'Failed to establish a connection to the database. Check your
                 ↪ environment variables!'
            );
            console.error(err);
            process.exit(1);
        }

        try {
            await getConnection().runMigrations();
        } catch (err) {
            console.error('Failed to run migrations');
            process.exit(1);
        }

        return new Promise(resolve => {
            this.server.listen(port, hostname, () => {
                resolve(this.server);
            });
        });
    }
}
```

Ukázka kódu 19: Třída udržující instanci samotného http serveru

Celkově má výsledný server 12 controllerů. Povětšinou jeden controller koresponduje s jednou databázovou entitou/tabulkou.

Nejrozsáhlejší metodou napříč všemi controllery je metoda **getAllContracts** zajišťující vyhledávání v seznamu zakázek. Z důvodu limitů TypeORM filtrování bylo nutné podmínky vyhledávání vytvořit ručně. Respektive sestavit jejich SQL podobu.




Obrázek 4.5: Struktura kódu @bokari/api-server

```
@Authorized()
@JsonController('/contracts')
export class ContractsController {
    @Get()
    @Authorized([Permission.CONTRACTS_READ])
    @ResponseSchema(Contract, { isArray: true })
    async getAllContracts(@QueryParams() query?: ContractsQueryParams): Promise<Contract[]> {
        if (!query || isEmptyObject(query)) {
            return getRepository(Contract).find();
        }

        const limit = query.limit && query.limit >= 0 ? query.limit : 100;
        const page = query.page || 1;
        const searchLike = `%${query.search}%`;
        const orderBy = query.orderBy || 'code';

        const take = limit;
        const skip = (page - 1) * limit;
        const order: FindOneOptions<Contract>['order'] = {};
        order[orderBy] = query.order || 'DESC';

        const whereQuery = (qb: SelectQueryBuilder<Contract>): void => {
            if (query.search) {
                qb.where('Contract.code = :search', { search: query.search })
                    .orWhere('Contract.name ILIKE :searchLike', { searchLike })
                    .orWhere('Contract_customer_person.name ILIKE :searchLike', {
                        searchLike
                    });
            }

            const filterable: (keyof ContractsQueryFilterable)[] = ['deadlineAt', 'startAt'];
            const filters: { filter?: ContractsQueryFilterable; operator: '<=' | '>=' }[] = [
                { filter: query.filterMax, operator: '<=' },
                { filter: query.filterMin, operator: '>=' }
            ];

            for (const filterProperty of filterable) {
                for (const filterObject of filters) {
                    if (
                        filterObject.filter &&
                        isNotEmptyObject(filterObject.filter) &&
                        filterProperty in filterObject.filter
                    ) {
                        qb.andWhere(`Contract.${filterable} ${filterObject.operator} :value`, {
                            value: filterObject.filter[filterProperty]
                        });
                    }
                }
            }
        };

        const contracts = await getRepository(Contract).find({
            where: (qb: SelectQueryBuilder<Contract>) => whereQuery(qb),
            take,
            skip,
            order
        });

        return contracts;
    }

    ...
}
```

Ukázka kódu 20: Handler zajišťující vyhledávání v seznamu zakázek

Validitu parametrů ověřuje sám balíček route-controllers, který validuje veškeré použité vstupní třídy pomocí balíčku class-validator. Proto je pro správnou validaci nutné popsat všechny vstupy pomocí vlastních tříd a správně odekorovat jejich atributy.

Routing-controllers též prochází jak vstupy, tak výstupy pomocí balíčku class-transformer, který nejen umožňuje například převádět objekty data do textové podoby, ale i umožňuje zahodit všechny nechtěné atributy. Není tak nutné se obávat, že při návratu entit z databáze budou uživateli navráceny například hashe hesel nebo že naopak server přijme k uložení více atributů, než které sám očekává.

```
@Exclude()
export class ContractsQueryFilterable {
    @Expose()
    @Type(() => Date)
    @IsOptional()
    @IsDate()
    deadlineAt?: Date;

    @Expose()
    @Type(() => Date)
    @IsOptional()
    @IsDate()
    startAt?: Date;
}

@Exclude()
export class ContractsQueryParams {
    @Expose()
    @IsOptional()
    @IsInt()
    limit?: number;

    @Expose()
    @IsOptional()
    @IsInt()
    @Min(1)
    page?: number;

    @Expose()
    @IsOptional()
    @IsString()
    search?: string;

    @Expose()
    @IsOptional()
    @IsString()
    @IsIn(Object.keys(new Contract()))
    orderBy?: keyof Contract;

    @Expose()
    @IsOptional()
    @IsString()
    @IsIn(['ASC', 'DESC'])
    order?: 'ASC' | 'DESC';

    @Expose()
    @IsOptional()
    @Type(() => ContractsQueryFilterable)
    @ValidateNested()
    filterMax?: ContractsQueryFilterable;

    @Expose()
    @IsOptional()
    @Type(() => ContractsQueryFilterable)
    @ValidateNested()
    filterMin?: ContractsQueryFilterable;
}
```

Ukázka kódu 21: Třídy definující možné parametry vyhledávání

#### 4.3.2.1 Autentizace

Autentizace je řešena pomocí JWT (JSON Web token). Toto řešení je voleno pro snadnější provoz výsledné aplikace na efemerním úložišti a z čistě akademického důodu, vyzkoušení této technologie.

Předávané tokeny mezi klientem a serverem mají v případě této práce dva druhy. *Access* Token, sloužící k ověření každého jednotlivého požadavku a *Refresh* Token, sloužící k automatickému vydání nového Access tokenu. Protože je Access token předáván při každém požadavku, má pro zajištění vyšší bezpečnosti jen velmi krátkou dobu platnosti. V případě Bokari 15 minut. Po této době server požadavky s tímto tokenem zamítne a očekává se, že klient vyšle nový požadavek na endpoint `POST /api/refresh`, s hodnotou lokálně uloženého Refresh tokenu. Server jej zvaliduje, a pokud je platný, vrátí klientovi nový platný Access token.

Po 7 dnech, což je doba platnosti vystaveného Refresh tokenu, je uživatel přesměrován na stránku přihlášení, kdy musí znovu zadat své přihlašovací údaje, které server ověří a vydá jeho klientu nový pár Access a Refresh tokenů.

JWT je též použit pro přenos informace o aktuálně přihlášeném uživateli a každý token je podepsán pomocí páru RSA klíčů. Klient tak má přístup k obsahu tokenu, ale nemůže jej pozměnit. Server si může být jist, že ho sám vydal a že s ním nebylo nijak manipulováno. Při každém požadavku, který je v jeho příslušném controlleru označen jako vyžadující autorizaci, je token zvalidován a jeho obsah načten do proměnné request, putující skrze všechny následné prostřednické (middleware) vrstvy odbavovaného požadavku.

Každý přístupový token obsahuje informaci o svém typu (Access/Refresh), uživateli, pro kterého byl vytvořen, času vytvoření a času vypršení platnosti. V případě Access tokenu je navíc obohacen o seznam oprávnění, kterými se může daný majitel tokenu prokázat. Při kontrole autorizace jsou poté kontrolována pouze přiložená data z tokenu a nemusí se tak vytvářet nový dotaz na databázi.

```
export async function authorizationChecker(action: Action, roles: Permission[]):
↪  Promise<boolean> {
    const authorizationHeader = action.request.headers.authorization;
    const token = authorizationHeader?.replace(/^Bearer/i, '').trim();

    if (!token) {
        throw new UnauthorizedError('No token provided!');
    }

    const payload = await verifyToken(JwtType.ACCESS, token);

    if (roles.some(scope => !payload.scopes.includes(scope))) {
        throw new UnauthorizedError('Insufficient permissions granted!');
    }

    action.request.jwt = payload;

    return true;
}
```

Ukázka kódu 22: Kontrola a načtení autorizace z tokenu

#### 4.3.2.2 Specifikace

Pro účely dokumentace a možné automatické integrace s dalšími nástroji je z metadat controllerů generována specifikace OpenAPI v3 v JSON podobě. Zpracování metadat do správného formátu řeší balíček **routing-controllers-openapi**, kterému jsou za běhu předána uložená metadata z routing-controllers a získaný objekt specifikace je navrácen klientu.

Pro interaktivní vizualizaci dokumentace jsou implementovány `GET /api/openapi-docs` a `GET /api/openapi3.json`. První endpoint vrátí statický obsah stránky dokumentace s SPA **Redoc**, která si sama z předané zdrojové specifikace vytvaří přímo v prohlížeči uživatele interaktivní dokumentaci.

Druhý endpoint při prvním zavolání sestaví specifikaci, uloží si ji paměti a zapíše na disk do lokálního balíčku **@bokari/api-specs**. Při následujicím volání již navrací specifikaci přímo z paměťové cache. Protože je vracená specifikace generována nově při každém spuštění instance serveru, je vždy plně aktuální. Pokud by provozovatel serveru nechtěl veřejně provozovat tuto dokumentaci, stačí serveru při spuštění předat proměnnou prostředí **BOKARI_GENERATE_API_DOCS** s negativní hodnotou. V takovém případě server vůbec controller, zajišťující generování dokumentace a specifikace, nenačte mezi aktivní controllery.

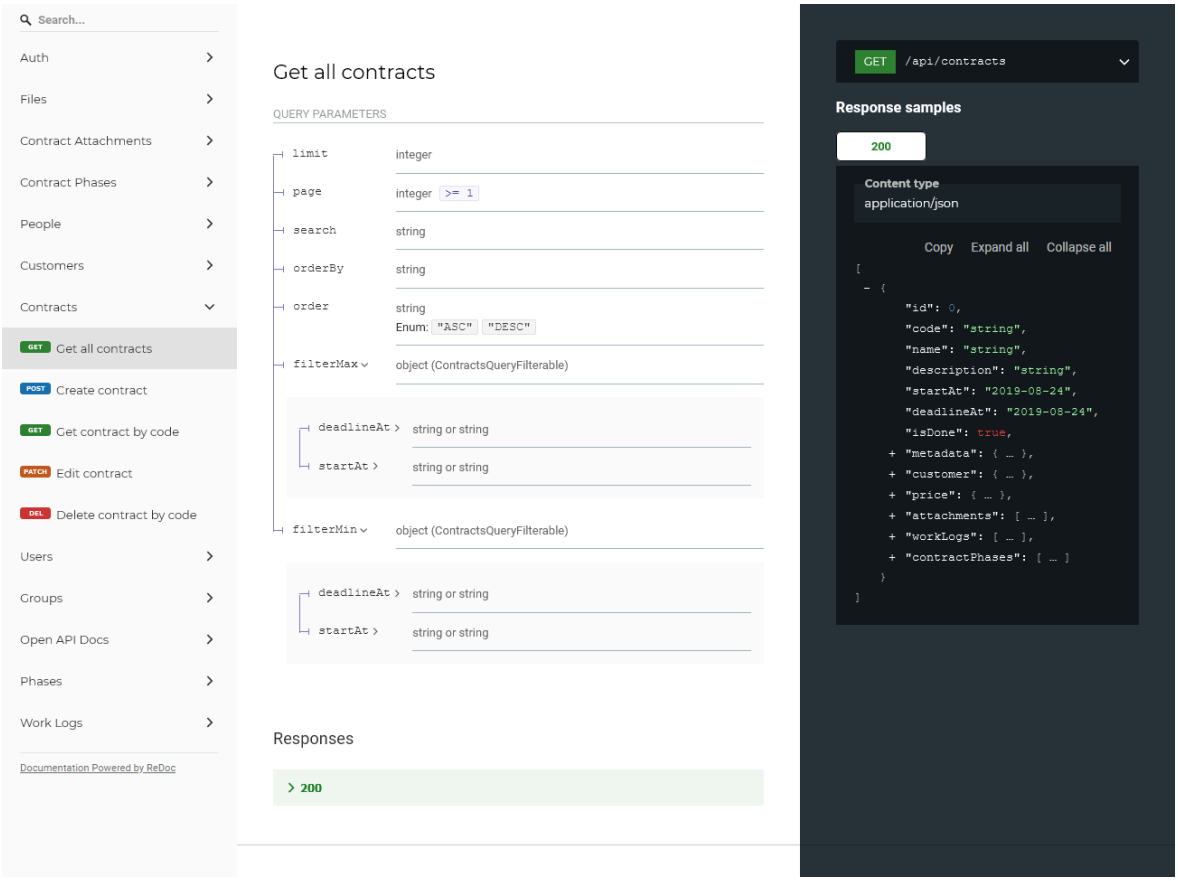
```
@Get()
@ContentType('text/html')
serveDocs(): string {
    return html`
        <!DOCTYPE html>
        <html>
            <head>
                <title>ReDoc</title>
                <meta charset="utf-8" />
                <meta name="viewport" content="width=device-width,
 ↪  initial-scale=1" />
                <link
                    href="https://fonts.googleapis.com/css?family=Montserrat:300,
 ↪  400,700|Roboto:300,400,700"
                    rel="stylesheet"
                />
                <style>
                    body {
                        margin: 0;
                        padding: 0;
                    }
                </style>
            </head>
            <body>
                <redoc spec-url="./openapi-docs/openapi3.json"></redoc>
                <script
 ↪  src="https://cdn.jsdelivr.net/npm/redoc@next/bundles/redoc.standalone.js"/>
            </body>
        </html>
    `;
}
```

Ukázka kódu 23: Obsah handleru GET /openapi-docs

### 4.3.3 Grafické rozhraní

Interaktivní webový klient napsaný za pomoci reaktivního frameworku **Vue** a komponent **Vuetify**. Design je částečně uzpůsoben nutnosti fungovat responzivně i na mobilních obrazovkách.

Pro pevné seznamy, jako jsou měna a oprávnění, používá přímo entity z @bokari/entities.




Obrázek 4.6: Vizualizace dokumentace API z vygenerované OpenAPI specifikace

#### 4.3.3.1 Use Cases

**Autentizace a autorizace**

- Uživatel očekává přihlášení:
  1. zadáním přihlašovacího jména
  2. zadáním přihlašovacího hesla
  3. potvrzením přihlášení
- Uživatel očekává možnost odhlášení na každé obrazovce
- Uživatel očekává, že bude v případě nepřihlášení nasměrován na obrazovku, kde se bude moci opět přihlásit
- Uživatel požaduje zobrazení upozornění o případném nastání chyby

**Tvorba zakázky**

- Uživatel očekává možnost vytvoření více než jedné zakázky ze stejné obrazovky
- Uživatel při vytváření zakázky očekává:
  - možnost ihned označit zakázku za dokončenou
  - zadání názvu zakázky
  - možnost zadání čísla zakázky
  - vybrání zákazníka
  - zadání začátku zakázky
  - zadání uzávěrky
  - zadání smluvené ceny
  - možnost připsat komentář
- Uživatel očekává, že se vytvořená zakázka objeví v seznamu zakázek

**Seznam zakázek**

- Uživatel požaduje možnost vyhledávání v seznamu zakázek podle:
  - názvu zakázky
  - čísla zakázky
  - jména zákazníka
- Uživatel očekává, že budou zakázky řaditelné podle zobrazených sloupců
- Uživatel očekává možnost zobrazení zakázky a její editaci
- Uživatel očekává možnost přidání nové zakázky do seznamu zakázek

**Zobrazení zakázky**

- Uživatel očekává zobrazení všech zadaných informací z dříve vytvořené zakázky
- Uživatel očekává možnost editace zobrazovaných údajů přímo ze stejné obrazovky
- Uživatel očekává zobrazení dodatečných údajů:
  - tvůrce zakázky
  - fáze zakázky
  - přílohy zakázky

#### 4.3.3.2 Scénáře

**Autentizace a autorizace**

- Pokud uživatel není přihlášen, systém zobrazí varování a přesměruje uživatele na obrazovku s přihlašováním
- Na straně s přihlášením:
  - systém zobrazí kartu se dvěma uživatelskými textovými vstupy
  - systém požaduje na uživateli zadat jeho uživatelské jméno do prvního textového pole
  - systém požaduje na uživateli zadat jeho uživatelské heslo do druhého textového pole
  - systém po zvolení potvrzovacího tlačítka pokusí uživatele přihlásit do systému
    * pokud bylo přihlášení úspěšné, systém uživateli odemkne všechny strany aplikace, kam má mít podle jeho oprávnění přístup a přesměruje ho na jeho primární stranu
    * pokud bylo přihlášení neúspěšné, systém uživateli zobrazí hlášku o neplatnosti údajů

**Tvorba zakázky**

- Systém zobrazuje odkaz na vytvoření nové zakázky na obrazovce seznamu zakázek
- Systém po zvolení odkazu přesměruje uživatele na obrazovku tvorby zakázky
- Na obrazovce tvorby zakázky:
  - systém zobrazí tento seznam vstupů:
    * textové pole pro název zakázky (systém jej vyžaduje)
    * textové pole pro zadání čísla zakázky
    * výběr zákazníka (systém jej vyžaduje)
    * datový vstup pro začátek zakázky (systém jej vyžaduje)
    * datový vstup pro uzávěrku zakázky (systém jej vyžaduje)

        * číselné pole pro hodnotu ceny zakázky (systém jej vyžaduje)
        * výběr měny ceny zakázky (systém jej vyžaduje)
        * textové pole pro popis zakázky
        * zaškrtávací pole pro označení zakázky za dokončenou
    - po zvolení potvrzujícího tlačítka systém uloží zakázku

**Seznam zakázek**

- Systém zobrazuje odkaz na seznam zakázek v postranním menu
- Systém po zvolení odkazu přesměruje uživatele na obrazovku seznamu zakázek
- Na obrazovce seznamu zakázek:
    - pokud má uživatel oprávnění vytvářet zakázky, systém zobrazí tlačítko pro přesměrování na obrazovku tvorby zakázek
    - systém zobrazí tabulku s vybranými zakázkami
        * ta obsahuje sloupce
            · číslo zakázky
            · název zakázky
            · jméno klienta
            · uzávěrku zakázky
            · menu s možnostmi
                · tlačítko pro zobrazení detailu dané zakázky
                · tlačítko pro smazání dané zakázky
        * nad tabulkou systém zobrazí textové vyhledávací pole
            · systém průběžně vyhledává zakázky podle zadaného textu ve vyhledávcím poli
            · při změně nalezených zakázek systém aktualizuje obsah tabulky novými daty

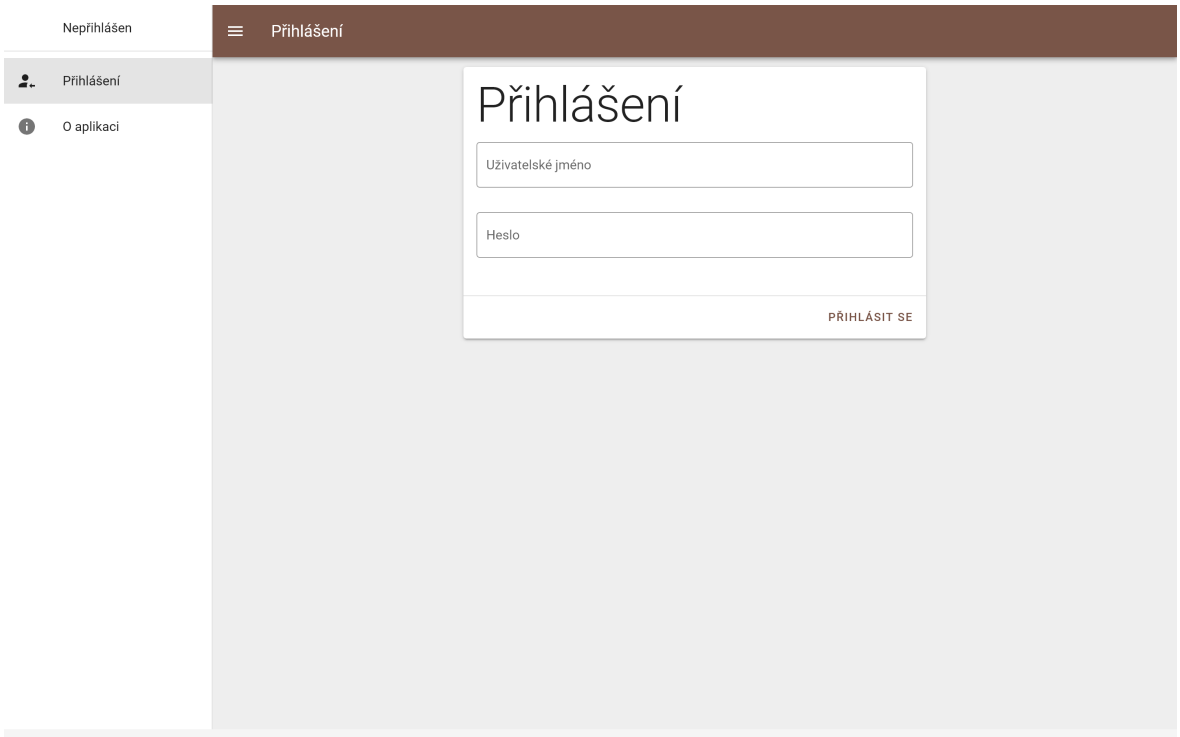
**Zobrazení zakázky**

- Systém zobrazí údaje o zakázce v podobě karet
    - karta se základními údaji o zakázce
        * obsahuje
            · název zakázky
            · číslo zakázky
            · jméno tvůrce zakázky
            · jméno zákazníka
            · cenu zakázky
            · stav dokončení zakázky
            · popis zakázky

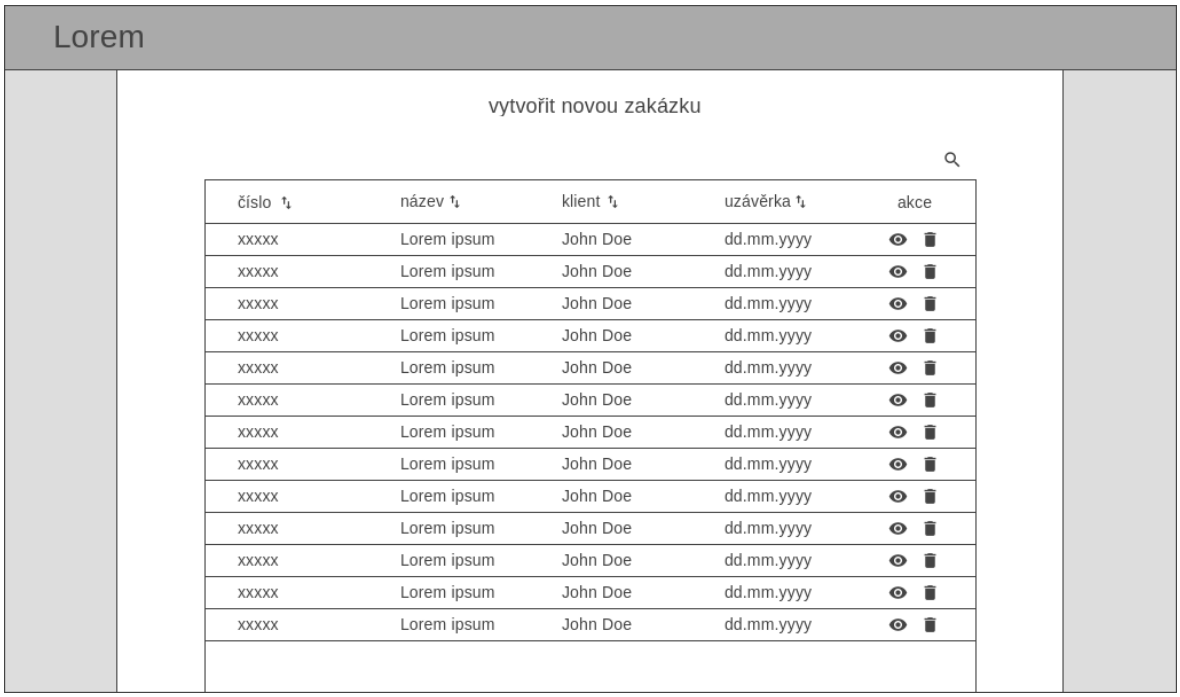
    * v hlavičce je zobrazen název karty a tlačítko pro editaci
        · po zvolení tlačítka editace systém umožní editaci zobrazených údajů zakázky v kartě
        · po druhém zvolení stejného tlačítka systém uloží změněné údaje a zobrazená pole opět uzamkne
- karta s časovými údaji o zakázce
    * ta obsahuje
        · začátek zakázky
        · uzávěrku zakázky
    * v hlavičce je zobrazen název karty a tlačítko pro editaci
        · po zvolení tlačítka editace systém umožní editaci zobrazených údajů zakázky v kartě
        · po druhém zvolení stejného tlačítka systém uloží změněné údaje a zobrazená pole opět uzamkne
- karta s fázemi zakázky
    * obsahuje tabulku fází s těmito sloupci
        · název fáze
        · uzávěrka fáze
        · dokončení fáze
        · menu možností
            · tlačítko pro editaci fáze
            · tlačítko pro odebrání fáze ze zakázky
    * v hlavičce je zobrazen název karty a tlačítko pro přidání nové fáze k zakázce
        · po zvolení tlačítka systém zobrazí plovoucí okno, které obsahuje
            · výběr data uzávěrky fáze (systém jej vyžaduje)
            · zaškrtávací pole pro označení fáze jako dokončené
            · výběr názvu fáze
- karta s přílohami zakázky
    * obsahuje seznam všech příloh zakázky
    * systém na konci seznamu zobrazuje
        · textové pole pro popis přílohy
        · pole pro nahrání souboru
        · tlačítko vytvoření přílohy
    * po výběru tlačítka systém nahraje přílohu spolu se souborem a přidá tuto přílohu do seznamu příloh zakázky

### Design

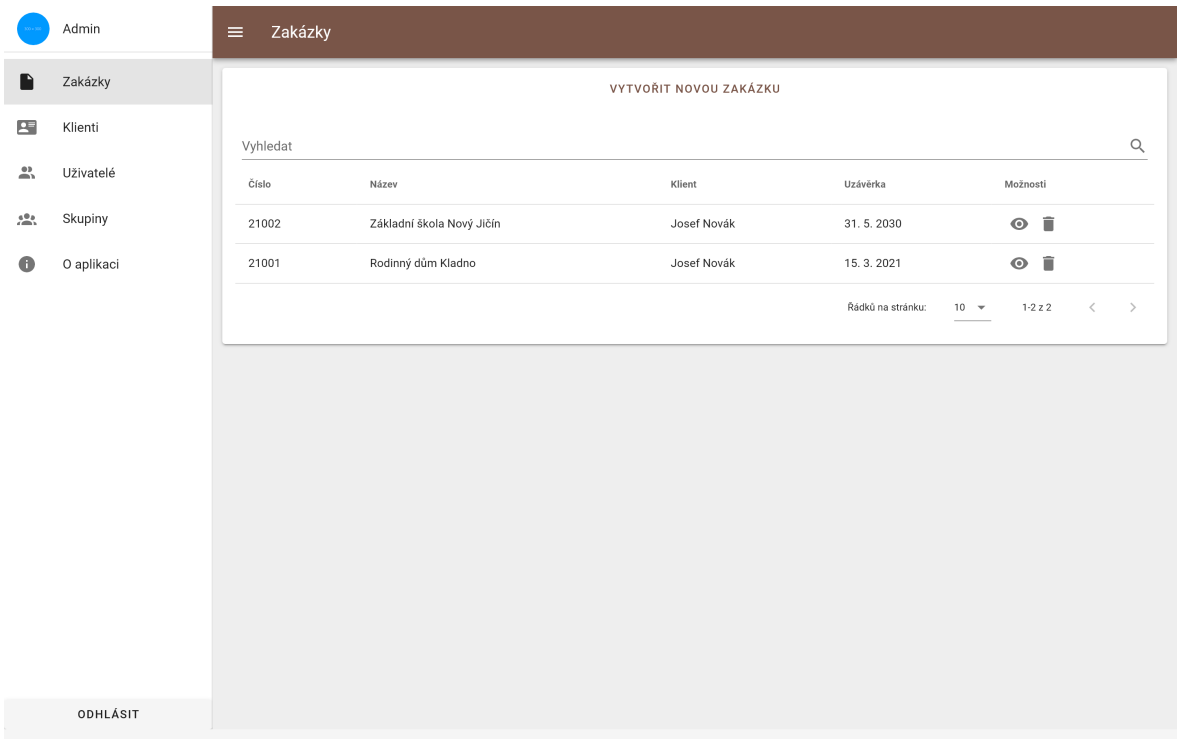

Obrázek 4.7: Wireframe přihlašovací obrazovky



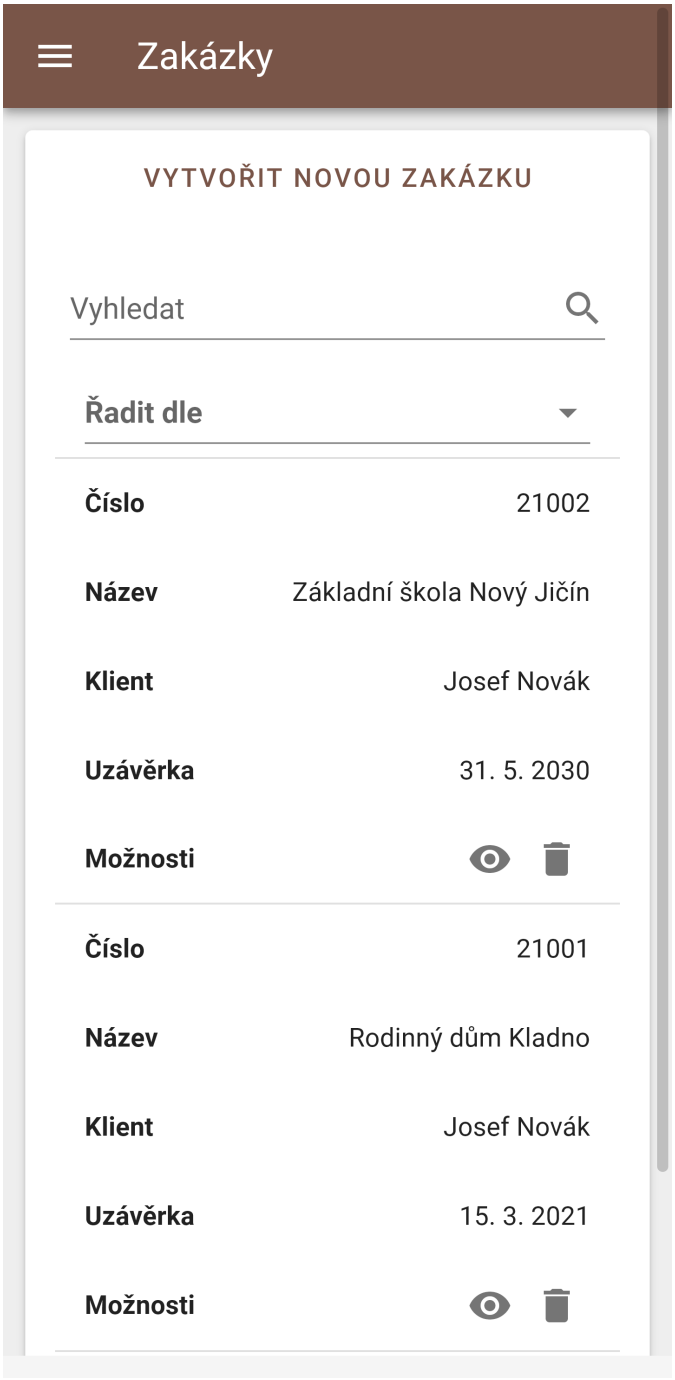

Obrázek 4.8: Výsledná podoba přihlašovací obrazovky



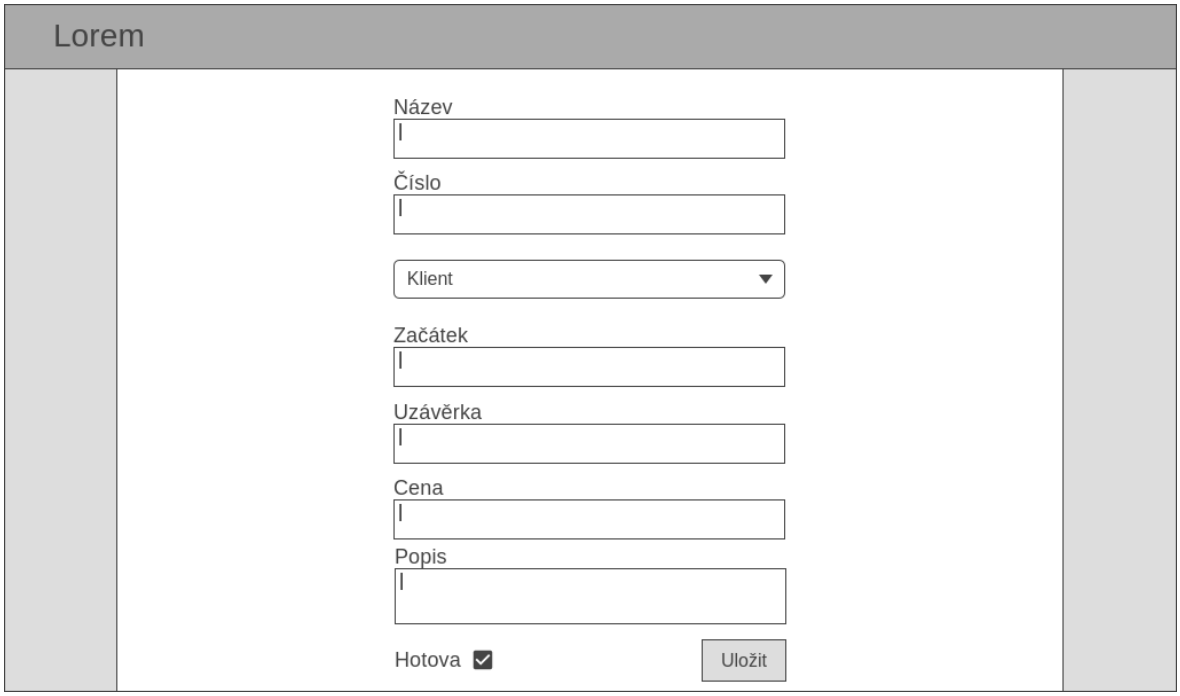

Obrázek 4.9: Wireframe zobrazení seznamu zakázek



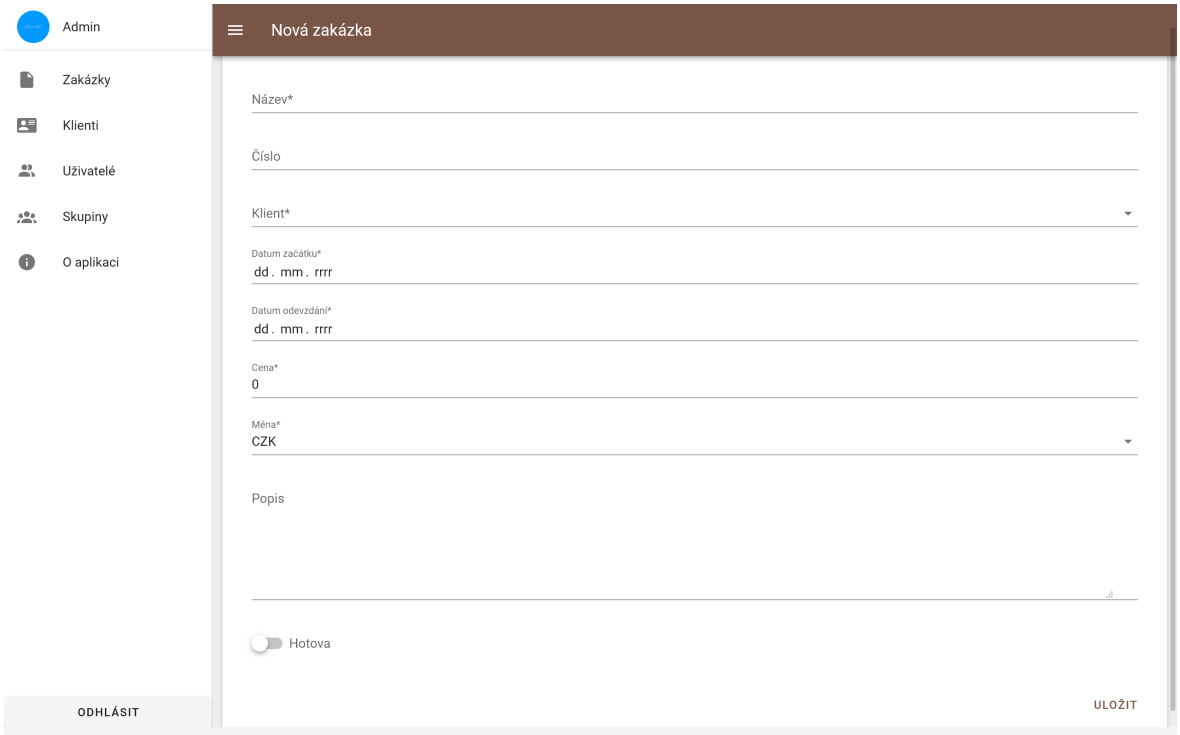

Obrázek 4.10: Výsledná podoba zobrazení seznamu zakázek



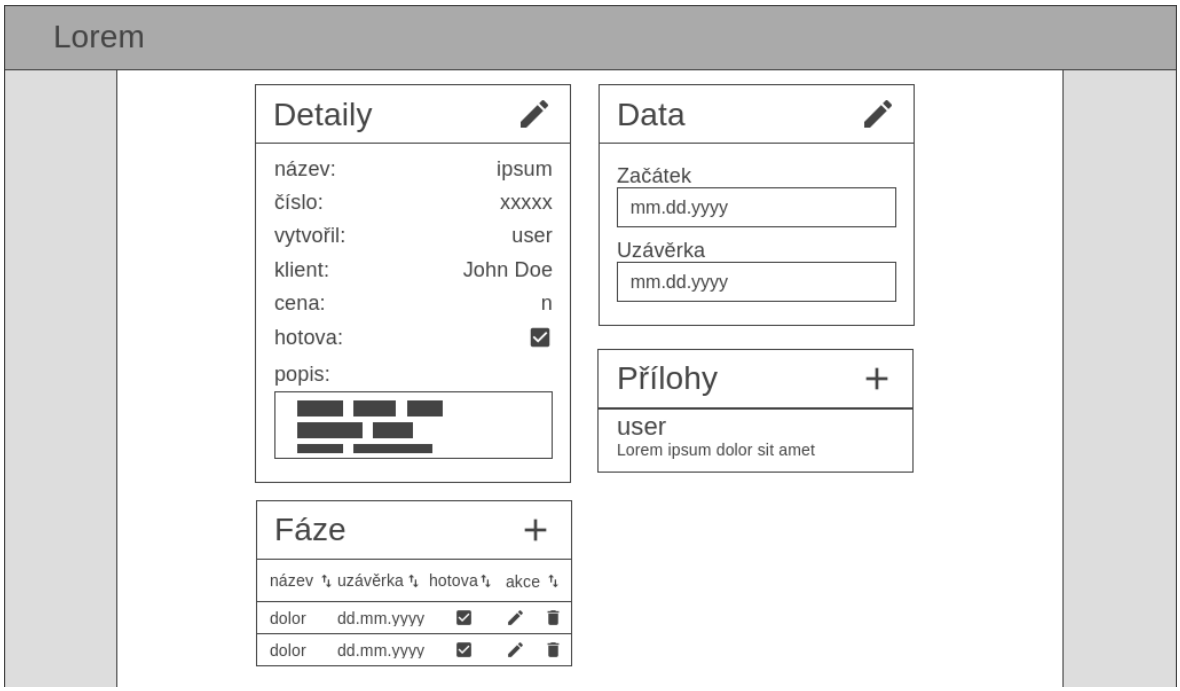

Obrázek 4.11: Výsledná podoba zobrazení seznamu zakázek na mobilním zařízení



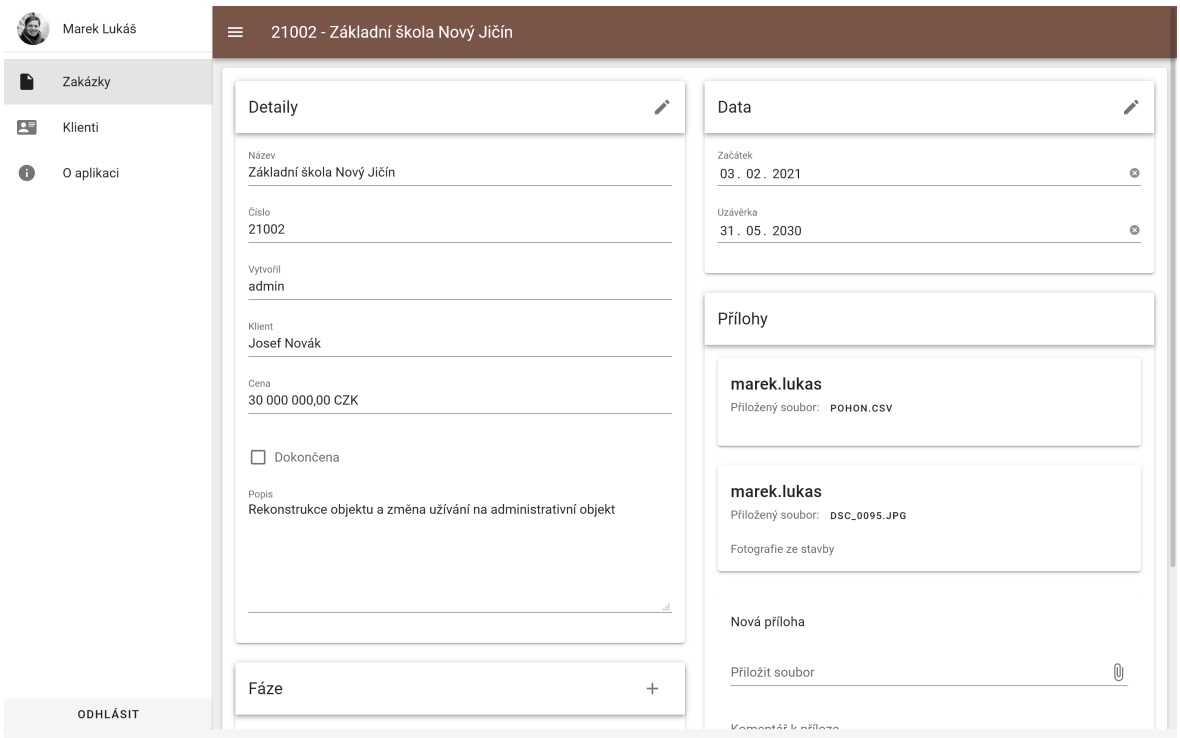

Obrázek 4.12: Wireframe zobrazení formuláře pro vytvoření nové zakázky



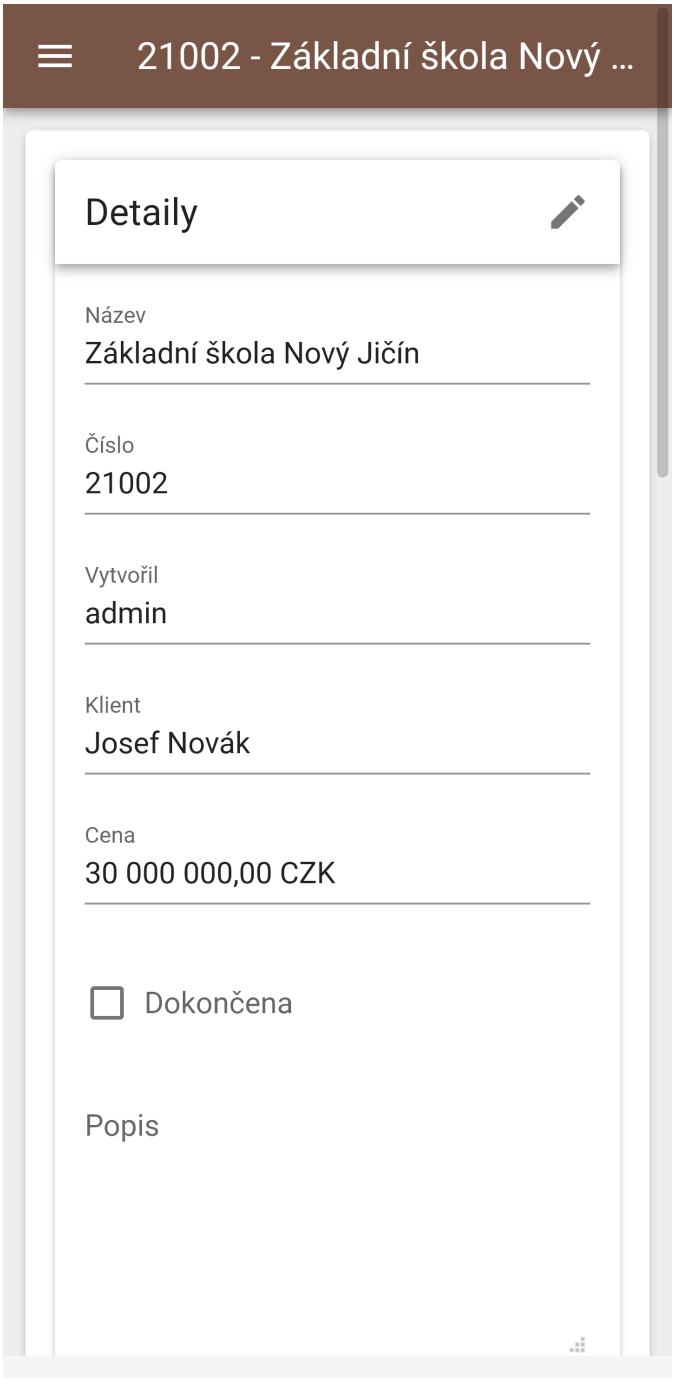

Obrázek 4.13: Výsledná podoba zobrazení formuláře pro vytvoření nové zakázky




Obrázek 4.14: Wireframe zobrazení existující zakázky




Obrázek 4.15: Výsledná podoba zobrazení existující zakázky




Obrázek 4.16: Výsledná podoba zobrazení existující zakázky na mobilním zařízení

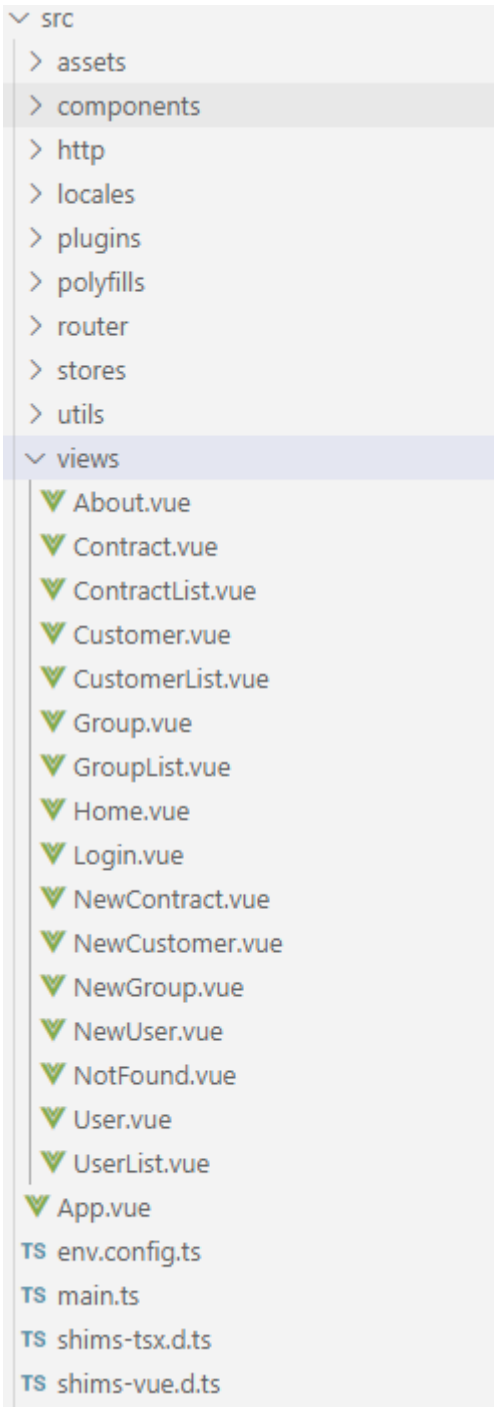
#### 4.3.3.3 Implementace

Prvotní základ projektu klientské aplikace byl vygenerován pomocí nástroje **Vue CLI**.

Celkově aplikace obsahuje 16 obrazovek. Každá samostatně navštívitelná.

```
<template>
    <v-card>
        <v-card-actions v-if="hasPermission(Permission.CONTRACTS_WRITE)">
            <v-spacer />
            <v-btn color="primary" text to="/new-contract">Vytvořit novou
            ↪ zakázku</v-btn>
            <v-spacer />
        </v-card-actions>

        <v-card-text>
            <v-text-field
                v-model="searchInput"
                append-icon="mdi-magnify"
                class="pa-2"
                clearable
                hide-details
                label="Vyhledat"
                singe-line
            />

            <v-data-table
                :headers="headers"
                :items="contracts"
                :items-per-page.sync="itemsPerPage"
                :loading="loading"
                :page.sync="page"
            >
                <template v-slot:item.deadlineAt="{ item }">
                    <span>{{ d(item.deadlineAt) }}</span>
                </template>

                <template v-slot:item.actions="{ item }">
                    <v-btn icon @click="openContractView(item)">
                        <v-icon>mdi-eye</v-icon>
                    </v-btn>
                    <v-btn v-if="hasPermission(Permission.GROUPS_WRITE)" icon
                    ↪ @click="deleteContract(item)">
                        <v-icon>mdi-delete</v-icon>
                    </v-btn>
                </template>
            </v-data-table>
        </v-card-text>
    </v-card>
</template>
```

Ukázka kódu 24: Vue komponenta seznamu zakázek (část šablony)




Obrázek 4.17: Struktura kódu @bokari/ui

```
export default defineComponent({
    name: 'ContractListView',
    setup() {
        useTitle('Zakázky');
        const { hasPermission } = useCurrentUserStore();
        const { showToast } = useToastStore();

        const searchInput = ref('');
        const debouncedSearchInput = useDebounce(searchInput);
        const typesafeSearch = computed(() => debouncedSearchInput.value ?? undefined);
        const loading = ref(true);
        const page = ref(1);
        const itemsPerPage = ref(10);
        const headers = ref<VDataTableHeader[]>([
            {
                text: 'Číslo',
                value: 'code'
            },
            {
                text: 'Název',
                value: 'name'
            },
            {
                text: 'Klient',
                value: 'customer.person.name'
            },
            {
                text: 'Uzávěrka',
                value: 'deadlineAt'
            },
            {
                text: 'Možnosti',
                value: 'actions'
            }
        ]);

        const contracts = asyncComputed(
            () =>
                contractsAPIClient
                    .getAllContracts(
                        itemsPerPage.value < 0 ? undefined : itemsPerPage.value,
                        page.value,
                        typesafeSearch.value
                    )
                    .then(res => res.data),
            [],
            loading
        );

        const router = useRouter();
        const openContractView = (contract: Contract) => {
            router.push({ name: 'Contract', params: { contractCode: contract.code } });
        };

        const deleteContract = async (contract: Contract) => {
            try {
                await contractsAPIClient.deleteContractByCode(contract.code);
                contracts.value = contracts.value.filter(c => c.id !== contract.id);
                showToast({
                    message: `Zakázka ${contract.code} byla úspěšně odebrána`,
                    type: 'success'
                });
            } catch {
                showToast({
                    message: `Nepodařilo se odebrat zakázku ${contract.code}`,
                    type: 'error'
                });
            }
        };

        return {
            loading,
            contracts,
            headers,
            searchInput,
            page,
            itemsPerPage,
            openContractView,
            hasPermission,
            deleteContract,
            Permission,
            ...useTypedI18n()
        };
    }
});
```

Ukázka kódu 25: Vue komponenta seznamu zakázek (část logiky)

Protože Vuetify nenabízí žádnou vlastní službu zajišťující ovládání upozornění, byla vytvořena služba vlastní, umožňující vytvoření nového upozornění z kontextu jakékoliv komponenty. (Vuetify et al. nedatováno) Byl použit koncept tzv *hooků*, koncept z nadcházející verze Vue 3, pomocí kterého je možné vložit do kontextu komponenty externí reaktivní proměnné.

Služba upozornění je implementována jako hook, který si ve vlastním kontextu udržuje stav upozornění k zobrazení. Při každém načtení tohoto hooku si komponenta vytvoří vlastní naslouchač událostí na tento globálně sdílený stav. Pokud je potřeba z nějaké komponenty zobrazit uživateli upozornění, je v komponentně zavolána funkce *useToastStore*, která v návratové hodnotě vrátí reaktivní stav a metody pro zobrazení/schování upozornění.

```
export function useToastStore() {
    if (!state) {
        state = reactive<ToastStoreState>({
            toasts: [],
            idCounter: 0
        });
    }

    const showToast = (options: ToastOptions) => {
        const id = ++state.idCounter;
        const message = options.message;
        const type = options.type;
        const outlined = options.outlined ?? false;
        const timeout = options.timeout ?? 2000;
        const visible = true;

        state.toasts.push({ id, message, type, outlined, timeout, visible });
    };

    const dismissToast = (id: number) => {
        const foundToast = state.toasts.find(toast => toast.id === id);

        if (!foundToast) {
            return;
        }

        foundToast.visible = false;
    };

    return {
        toasts: readonly(state.toasts),
        showToast,
        dismissToast
    };
}
```

Ukázka kódu 26: Hook zajišťující globální správu upozornění

Změnám ve stavu upozornění je naslouchán v komponentě **AppToastContainer**, která je posazena do kořenové komponenty **App** a vykresluje všechna aktuálně zobrazená upozornění.

```
<template>
    <div>
        <v-snackbar v-for="toast of toasts" :key="toast.id"
        ↪ v-model="toast.visible">
            {{ toast.message }}
            <template v-slot:action="{ attrs }">
                <v-btn v-bind="attrs" :color="toast.type" text
                ↪ @click="dismissToast(toast.id)">
                    Schovat
                </v-btn>
            </template>
        </v-snackbar>
    </div>
</template>

<script lang="ts">
import { defineComponent } from '@vue/composition-api';

import { useToastStore } from '../stores/toast.store';

export default defineComponent({
    name: 'AppToastContainer',
    setup() {
        const toastStore = useToastStore();

        return {
            ...toastStore
        };
    }
});
</script>
```

Ukázka kódu 27: Komponenta starající se o zobrazení upozornění z globálního stavu

Validace je též řešena vlastním hook. Hook, s názvem *useValidation*, přijme jako argument reaktivní odkaz na objekt Vuetify formuláře. Ten si uloží a jako svou návratovou hodnotu vrátí metody pro validaci a validační pravidla, která lze poté použít v původní komponentě.

Příkladem validačních pravidel, které byla implementována jsou pravidla *isPattern*, porovnávající uživatelův vstup s regulárním výrazem, pravidlo *isNumber*, kontrolující, že je textový vstup číslem nebo pravidlo *isRequired*, kontrolující neprázdnost vstupu. Mezi další definice validátorů, které *useValidation* navrací, jsou ještě například pravidla *isEmail*, *isPhoneNumber* nebo *isImage*.

```
export function useValidation(formRef: Ref<VFormElement | null>) {
    const isPattern = (pattern: RegExp): InputValidationRule => {
        return (input: string) =>
            !input || pattern.test(input) || 'Vstup neodpovídá požadovanému
             ↪ formátu';
    };

    const isNumber: InputValidationRule = (input: string) =>
        !input || /^[0-9,. ]+$/.test(input) || 'Vstup není ve správném formátu
         ↪ čísla';

    const isRequired: InputValidationRule = (input: unknown) => !!input ||
     ↪ 'Povinný vstup';

    ...

    const validate = () => formRef.value?.validate();
    const reset = () => formRef.value?.reset();
    const resetValidation = () => formRef.value?.resetValidation();

    return {
        isPattern,
        isNumber,
        isRequired,
        ...
        validate,
        resetValidation,
        reset
    };
}
```

Ukázka kódu 28: Hook zajišťující validaci pro specifický formulář

```
const { validate: validatePhaseForm } = useValidation(phaseForm);
```

Ukázka kódu 29: Načtení hooku ve skriptové části komponenty

```
<v-form ref="phaseForm">
    <v-text-field
        v-model="newPhase.name"
        :rules="[isRequired]"
        label="Název fáze"
    />
</v-form>
```

Ukázka kódu 30: Aplikace validačních pravidel na textové pole

#### 4.3.3.4 API klient

Pro funkce komunikující s API serveru byl vyčleněn vlastní lokální balíček **@bokari/api-client**. V něm se nachází klient vygenerovaný pomocí nástroje **OpenAPI Generator**, konkrétně jeho **typescript-axios** generátor. Ten generuje ze vstupní OpenAPI specifikace HTTP API klienta v jazyku TypeScript a využívající externí knihovnu **axios** pro odbavení samotných HTTP požadavků.

Generátory používající pro odbavení požadavků přímo rozhraní **fetch** nebo **XMLHttpRequest** neumožňovaly naslouchat na probíhajících požadavcích a odpovědích. To bylo potřeba k automatickému centrálnímu řešení autentizace požadavků. S výměnou *Access* tokenu poté pomáhá knihovna **axios-auth-refresh**.

Po vygenerování měl ale výstupní kód určité nedostatky. Například funkce, odbavující požadavky na controller spravující nahrávané soubory, neobsahovaly správné návratové typy, což komplikovalo výsledné použití v aplikaci. Dále byly pevné výčtové seznamy oprávnění a měny vygenerovány s různými názvy podle controlleru, v níž byly použity. To opět komplikovalo správnou validaci typů. Protože autor nenašel jinou alternativu, byl vytvořen skript, který vygenerovaný kód po každém vygenerování projde a pomocí regulárních výrazů jej opraví podle nutných požadavků.

```
const EXPECTED_MODEL_FILES = 4;
const MODEL_FILE_USAGE_REGEX = /<any>/g;

const EXPECTED_CURRENCY_ENUMS = 1;
const CURRENCY_ENUM_DECLARATION_REGEX = /\/\*\*\n(?:.+\n)*export enum Currency
↪  {\s*(?:.+\n)*/gm;

const fileToPatch = path.join(__dirname, '..', 'src', 'api.ts');

console.log(`Loading file ${fileToPatch}`);
const originalFileContents = fs.readFileSync(fileToPatch, {
    encoding: 'utf-8'
});
let patchedFileContents = originalFileContents;

console.log('Patching ModelFile...');
if (patchedFileContents.match(MODEL_FILE_USAGE_REGEX).length ===
↪  EXPECTED_MODEL_FILES) {
    patchedFileContents = patchedFileContents.replace(MODEL_FILE_USAGE_REGEX,
    ↪  '<ModelFile>');
} else {
    console.error('Found an unexpected number of ModelFiles to be patched. Check
    ↪  the original file!');
    process.exit(1);
}

console.log('Patching Currency...');
if (patchedFileContents.match(CURRENCY_ENUM_DECLARATION_REGEX).length ===
↪  EXPECTED_CURRENCY_ENUMS) {
    patchedFileContents =
    ↪  patchedFileContents.replace(CURRENCY_ENUM_DECLARATION_REGEX, '');
    patchedFileContents = 'import { Currency } from "@bokari/entities"\n' +
    ↪  patchedFileContents;
} else {
    console.error('Found an unexpected number of Currency to be patched. Check the
    ↪  original file!');
    process.exit(1);
}

console.log('Saving changes to the original file...');
fs.writeFileSync(fileToPatch, patchedFileContents, { encoding: 'utf-8' });

console.log(`File ${fileToPatch} has been successfully patched`);
```

Ukázka kódu 31: Lokální oprava vygenerovaného kódu (pouze výběr)

V kódu uživatelské aplikace bylo poté možné pracovat s HTTP API serveru s plnou kontrolou TypeScript typů a bez nutnosti manuálně specifikovat adresy jednotlivých endpointů a jejich dostupných metod.

```
await contractsAPIClient.getContractByCode('21315')
```

Ukázka kódu 32: Použití vygenerovaného a inicializovaného API klienta

### 4.3.4 Nasazení

Aby byla zajištěna reprodukovatelnost sestavení a běhu aplikace, byl pro každou část systému vytvořen **Dockerfile**, popisující, jak danou část sestavit a spustit v prostředí **Docker**.

```
# Build
FROM node:14 as builder

WORKDIR /build

COPY yarn.lock package.json ./
COPY packages/api-client/package.json packages/api-client/
COPY packages/entities/package.json packages/entities/
COPY packages/ui/package.json packages/ui/

RUN yarn install

COPY . .

ARG VUE_APP_BOKARI_API_URL
RUN yarn lerna run --scope "@bokari/{api-client,entities,ui}" --stream build

# Runtime
FROM nginx:alpine

RUN rm /etc/nginx/conf.d/*

COPY packages/ui/nginx /etc/nginx/conf.d/
COPY --from=builder /build/packages/ui/dist /usr/share/nginx/html

EXPOSE 80
```

Ukázka kódu 33: Dockerfile pro sestavení a spuštění Vue části aplikace

V kořeni repozitáře je připraven orchestrační soubor pro nástroj **docker-compose**. Jsou v něm popsány vzájemné závislosti jednotlivých služeb a je skrze něj možné konfigurovat všechny služby najednou pomocí proměnných prostředí, které docker-compose předá samotným kontejnerům se službami.

```yaml
services:
  api:
    build:
      context: .
      dockerfile: packages/api-server/Dockerfile
    ports:
      - 127.0.0.1:8001:3000
    volumes:
      - uploads_volume:/app/uploads
    depends_on:
      - database
    environment:
      NODE_ENV: production
      BOKARI_UPLOADS_STORAGE_DIR: /app/uploads
      TYPEORM_CONNECTION: postgres
      TYPEORM_HOST: database
      TYPEORM_USERNAME: postgres
      TYPEORM_PASSWORD: changeme2
      TYPEORM_DATABASE: bokari
      TYPEORM_PORT: 5432
      TYPEORM_SYNCHRONIZE: "true"
      TYPEORM_LOGGING: "false"
      JWT_PRIVATE_KEY: changeme1
      JWT_PUBLIC_KEY: changeme1
  ui:
    build:
      context: .
      dockerfile: packages/ui/Dockerfile
    ports:
      - 127.0.0.1:8002:80
  uploads:
    build:
      context: .
      dockerfile: packages/uploads/Dockerfile
    ports:
      - 127.0.0.1:8003:80
    volumes:
      - uploads_volume:/usr/share/nginx/html
  database:
    image: postgres
    volumes:
      - database_volume:/var/lib/postgresql/data
    environment:
      POSTGRES_DB: bokari
      POSTGRES_PASSWORD: changeme2
volumes:
  uploads_volume:
  database_volume:
```

Ukázka kódu 34: Soubor docker-compose.yml

Za pomoci docker-compose je nasazení celkem snadné na jakémkoliv serveru s jakýmkoliv operačním systémem, na němž je platforma Docker dostupná. V budoucnu bude nastaveno automatické sestavení a nasazení pomocí nástroje GitHub Actions. Nyní je aplikace nasazována ručně přes SSH, git repozitář a ruční zavolání `docker-compose build`.

Aplikace byla úspěšně nasazena na virtuálním privátním serveru a je dostupná na adrese bokari.t4t.cz. Bylo žádoucí aplikaci provozovat přes protokol HTTPS a zároveň nezveřejnit API server přímo na vlastním portu. Místo toho byl vytvořen virtuální hostitel v již existujicím webovém serveru **nginx**.

Na virtuálním privátním serveru s jedním jádrem a 1 GB RAM obslouží API server zhruba 50 až 200 požadavků za sekundu. Ve většině případů se množství odbavených požadavků pohybuje na vyšší straně intervalu. Není použito žádné explicitní cachování ani node clusterizace. Pokud by nějaký ze zmíněných vzorů byl použit, mělo by být možné dosáhnout o něco lepších výsledků.

```
upstream api {
    server 127.0.0.1:8001;
}
upstream ui {
    server 127.0.0.1:8002;
}
upstream uploads {
    server 127.0.0.1:8003;
}
server {
    listen 80;
    server_name bokari.t4t.cz;

    gzip on;
    gzip_vary on;
    gzip_proxied any;
    gzip_comp_level 6;
    gzip_types text/plain text/css text/xml application/json
     ↪ application/javascript application/rss+xml application/atom+xml
     ↪ image/svg+xml;

    add_header X-Frame-Options                      "SAMEORIGIN" always;
    add_header X-XSS-Protection                     "1; mode=block" always;
    add_header X-Content-Type-Options               "nosniff" always;
    add_header Referrer-Policy                      "no-referrer-when-downgrade"
     ↪ always;
    add_header Content-Security-Policy              "default-src 'self' http:
     ↪ https: data: blob: 'unsafe-inline' 'unsafe-eval'" always;
    add_header Strict-Transport-Security            "max-age=31536000;
     ↪ includeSubDomains" always;

    location / {
        proxy_pass http://ui;
    }
    location /static/uploads {
        proxy_pass http://uploads/;
    }
    location /api {
        proxy_set_header X-Real-IP         $remote_addr;
        proxy_set_header X-Forwarded-For   $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host  $host;
        proxy_set_header X-Forwarded-Port  $server_port;
        proxy_connect_timeout              60s;
        client_max_body_size               0;
        proxy_pass http://api;
    }

    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

Ukázka kódu 35: Aktuální konfigurace webové proxy nginx

```
Server Software:        nginx
Server Hostname:        bokari.t4t.cz
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES256-GCM-SHA384,2048,256
Server Temp Key:        X25519 253 bits
TLS Server Name:        bokari.t4t.cz

Document Path:          /api/users/admin
Document Length:        37 bytes

Concurrency Level:      100
Time taken for tests:   5.815 seconds
Complete requests:      1000
Failed requests:        0
Non-2xx responses:      1000
Total transferred:      575000 bytes
HTML transferred:       37000 bytes
Requests per second:    171.96 [#/sec] (mean)
Time per request:       581.519 [ms] (mean)
Time per request:       5.815 [ms] (mean, across all concurrent requests)
Transfer rate:          96.56 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        4  495 140.3    554     626
Processing:    10   56 105.3     13     510
Waiting:        0   56 105.3     13     510
Total:         18  551  80.4    574     716

Percentage of the requests served within a certain time (ms)
  50%    574
  66%    583
  75%    588
  80%    596
  90%    633
  95%    649
  98%    682
  99%    692
 100%    716 (longest request)
```

Ukázka kódu 36: Výstup testu rychlosti API endpointu GET /users/admin

# 5 Výsledky a diskuse

Takto rozsáhlá webová aplikace byla zatím největším projektem autora. To z této práce vytvořilo zajímavou výzvu, které by se autor rád věnoval i v budoucnu. Mohl si vyzkoušet práci s velkou řadou technologií a sám si vybrat, které mu a jeho účelu vyhovují.

Podnikových systémů existuje mnoho. Pokud jsou vyloučeny pokročilé či specifické požadavky, je podobných aplikací téměř nekonečná řada.

Ve srovnání s existujícími řešeními zmíněnými v kapitole 3.4.2 má prezentované řešení mnohem jednodušší charakter. Je tomu tak nejen z hlediska použitelnosti, ale též z pohledu dostupných funkcí. Přesto by ale sestavené řešení mělo být použitelné pro velmi malé firmy, které by Bokari používaly jako doplňek k existující papírové dokumentaci.

Výsledná aplikace je funkční a pro účely záznamu údajů o probíhajících zakázkách splňuje zadaná kritéria. Autor by ji rád v budoucnu rozšířil o více funkcí k zuniverzálnění aplikace pro více podnikových postupů. Ne všechny podniky musí například evidovat začátek zakázky nebo jejího přímého zákazníka. Též by rozhraní aplikace mělo být přeloženo do druhé jazykové mutace, angličtiny, a tím by měla být usnadněna budoucí otevřená spolupráce s dalšími potenciálními vývojáři.

Protože Bokari samo nijak neřeší zálohování, je na provozovateli, aby obstaral periodickou zálohu dat z databáze a souborů nahraných uživateli.

Část serveru a databáze byla několikrát přepsána od nuly, protože bylo dosaženo funkčního stropu zvolených technologií. V původní verzi Bokari bylo celé schéma databáze popsáno za pomoci ručně psaného SQL skriptu a přístup k datům byl řešen nástrojem **Prisma**. Bohužel toto řešení neumožňovalo jednoduchou transformaci dat z databázového modelu na model odpovědí publikovatelných uživatelům RESTového API. Protože ale dané řešení bylo flexibilnější než stávající řešení s použitím TypeORM, autor práce sleduje vývoj nástroje Prisma a v budoucnu k němu zváží opětovný přechod.

Podobně bylo nutné přepsat kód controllerů používajících nástroj **tsoa** na stávající knihovnu routing-controllers. Stalo se tak z důvodu nedostatečné podpory externích definic typů a překvapivé citlivosti na názvy typů v různých lokálních balíčcích. Dané řešení vyžadovalo méně dekorátorů a méně ruční specifikace typů, ale z důvodu proble-

mů, se kterými se autor setkal, je zvoleno stávající řešení. Oproti komunikaci s databází nevyžadují nástroje pro tvorbu controllerů příliš velkou změnu kódu a mimo drobné změny pojmenování bylo možné tsoa a routing-controllers vyměnit jeden za druhý.

Volba JWT pro přenos autentizačních a autorizačních údajů není zcela ideální. Pokud by se útočníkovi podařilo získat přístup k uživatelově tokenu, nemohl by s tím administrátor nic udělat a jediné jisté řešení by bylo změnit podpisové klíče tokenů. Lze to řešit ukládáním zavrhnutých tokenů na černou listinu a tu kontrolovat při každé autentizační kontrole. Takové řešení ale nebylo neimplementováno, protože by částečně znevažovalo hlavní výhody JWT.

Pozitivní volbou bylo použití webových technologií a jazyku TypeScript. Nástroje k editaci, dokumentace a ekosystém jsou dnes již dostatečně vyspělé. Protože autor na svých zařízeních používá kombinaci operačních systémů, je možnost vyvíjet aplikaci i ji nativně provozovat na systémech GNU/Linux, Windows i Android (webový prohlížeč + Termux) velmi příjemná. Protože zvolený manažer JS (JavaScript) závislostí automaticky zapisuje přesný seznam verzí jednotlivých závislostí, včetně jejich hashe, je jisté, že při opakovaném nasazení či přesunu mezi systémy je použito přesně stejné prostředí, které autor a jeho kód očekávají.

S příchodem Vue 3 je očekávána velká změna syntaxe v nové verzi frameworku Vuetify a autor počítá s rozsáhlou úpravou webového klienta na novou verzi těchto dvou frameworků. Výsledkem by měla být kratší doba sestavování, menší výsledný balíček a přehlednější struktura kódu balíčku @bokari/ui.

# 6 Závěr

Celkovým cílem práce bylo vytvořit systém pro správu zakázek malé firmy. Výsledný systém měl mít praktické využití v reálné firmě a mít tak i reálný přesah.

Teoretická část měla dvě hlavní části. V první je popsán stav aktuálního trhu s podnikovými systémy. Bylo vybráno několik příkladů z kategorie proprietárních a open-source systémů. Druhá část je zaměřena na stav webových frameworků a technologií, čím se liší a kde se používají.

V praktické části byla navržena a implementována funkční webová aplikace fungující jako primitivní podnikový systém.

Aplikace byla nasazena na reálně používaném produkčním serveru. Svou rychlostí a funkcemi odpovídá požadavkům malé skutečné firmy.

Po dokončení bakalářské práce na založeném projektu bude na projektu dále pracováno a je žádoucí jej dále rozšiřovat.

# 7 Seznam použitých zdrojů


AIRBNBENG, 2017. Isomorphic JavaScript: The Future of Web Apps. *Medium* [online] [vid. 2021-01-22]. Dostupné z: https://medium.com/airbnb-engineering/isomorphic-javascript-the-future-of-web-apps-10882b7a2ebc

APACHE SOFTWARE FOUNDATION, 2004. *Apache License 2.0* [online] [vid. 2021--01-02]. Dostupné z: https://spdx.org/licenses/Apache-2.0.html

APIS YOU WON'T HATE, nedatováno. *OpenAPI.Tools* [online] [vid. 2021-02-26]. Dostupné z: https://openapi.tools/

ATLASSIAN, 2020a. Jira Overview. *Atlassian* [online] [vid. 2021-01-09]. Dostupné z: ht tps://www.atlassian.com/software/jira/guides/getting-started/overview

ATLASSIAN, 2020b. What does Atlassian's cloud-first model mean for you? *Atlassian* [online] [vid. 2021-01-10]. Dostupné z: https://www.atlassian.com/migration/journey-to-cloud

ATLASSIAN, nedatováno. What is Jira used for? *Atlassian* [online] [vid. 2021-01-10]. Dostupné z: https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for

AUSTIN, Jamey, 2015. 6 call-outs from the Summit 2015 keynote address. *Work Life by Atlassian* [online]. [vid. 2021-01-10]. Dostupné z: https://www.atlassian.com/blog/hr-teams/6-call-outs-from-the-summit-2015-keynote-address

BASL, Josef, Roman BLAŽÍČEK a ČESKÁ SPOLEČNOST PRO SYSTÉMOVOU INTEGRACI, 2012. *Podnikové informační systémy: podnik v informační společnosti*. Praha: Grada. ISBN 978-80-247-7594-4.

BAXTER, Lincoln III, 2016. Origins of .NET on Linux: An explanation for Java Developers. *Red Hat Developer* [online]. [vid. 2021-01-23]. Dostupné z: https://developers.redhat.com/blog/2016/03/29/origins-of-net-on-linux-an-explanation-for-java-developers/

BERNING, Dave, 2018. Getting Started with Server-Side Rendering Using Nuxt.js. *DigitalOcean* [online] [vid. 2021-01-26]. Dostupné z: https://www.digitalocean.com/community/tutorials/vuejs-server-side-rendering-with-nuxtjs

BICEK, Zdeněk, Martin KUNEŠ a V-PEJANO, 2020. Seznamte se s Microsoft Dynamics 365 Business Central. *Microsoft Docs* [online] [vid. 2021-01-09]. Dostupné z: https://docs.microsoft.com/cs-cz/dynamics365/business-central/

BRERETON, Jake, 2015. Introducing Jira Software: the #1 software development tool used by agile teams. *Work Life by Atlassian* [online]. [vid. 2021-01-10]. Dostupné z: https://www.atlassian.com/blog/jira-software/introducing-jira-software

CHOPIN, Alex, nedatováno. middleware. *NuxtJS* [online] [vid. 2021-01-26]. Dostupné z: https://nuxtjs.org/docs/2.x/directory-structure/middleware/,%20/docs/2.x/directory-structure/middleware

CLEASBY, Richard, Gudbrand VIGFUSSON, AMERICAN-SCANDIVANIAN FOUNDATION, Sean CRIST, Rebecca KUIPERS, Michael O'KEEFE a Bronwyn WOODS, 1874. *bóka* [online]. Dostupné z: http://lexicon.ff.cuni.cz/html/oi_cleasbyvigfusson/b0074.html

CROMWELL, Vivian, 2018. Between the Wires: An Interview with Vue.js creator Evan You. *Between The Wires* [online]. [vid. 2021-01-28]. Dostupné z: https://dev.to/betweenthewires/evan-you

DRÁBEK, Bc. Jiří, 2009. *Správa informačních systémů velkých firem* [online]. Praha. Diplomová práce. Bankovní institut vysoká škola Praha. Dostupné z: https://is.ambis.cz/th/enp7t/Jiri.Drabek.Diplomova.Prace.SpravaISvelkychFirem.pdf

DWYER, Gareth, 2020. Stateful vs Stateless Architecture: Why Stateless Won. *Virtasant* [online] [vid. 2021-01-31]. Dostupné z: https://virtasant.com/blog/stateful-vs-stateless-architecture-why-stateless-won/

ERPNEXT OPEN SOURCE SOFTWARE FOUNDATION, 2021. *About ERPNext Foundation* [online] [vid. 2021-01-12]. Dostupné z: https://erpnext.org/about

ESCHWEILER, Sebastian, 2018. REST vs. GraphQL. *CodingTheSmartWay.com* [online]. [vid. 2021-01-28]. Dostupné z: https://codingthesmartway.com/rest-vs-graphql/

FINDLAY, Thomas, 2020. Vue Composition API-–What Is it and How Do I Use It? *Telerik Blogs* [online]. [vid. 2021-01-29]. Dostupné z: https://www.telerik.com/blogs/vue-composition-api-what-is-it-how-to-use-it

FORMAN, Aaron, 2015. Say hello to Jira Core. *Work Life by Atlassian* [online]. [vid. 2021-01-10]. Dostupné z: https://www.atlassian.com/blog/jira-core/say-hello-jira-core

FOWLER, Martin, 2012. NosqlDefinition. *martinfowler.com* [online]. [vid. 2021-01-17]. Dostupné z: https://martinfowler.com/bliki/NosqlDefinition.html

FRAPPE, 2020. *Introduction* [online] [vid. 2021-01-12]. Dostupné z: https://docs.erpnext.com/docs/user/manual/en/introduction

FREE SOFTWARE FOUNDATION, 2001. Co je to svobodný software? *The GNU Operating System* [online] [vid. 2021-01-01]. Dostupné z: https://www.gnu.org/philosophy/free-sw.cs.html

FREE SOFTWARE FOUNDATION, 2007a. *GNU Affero General Public License v3.0 or later* [online] [vid. 2021-01-02]. Dostupné z: https://spdx.org/licenses/AGPL-3.0-or-later.html

FREE SOFTWARE FOUNDATION, 2007b. *GNU General Public License v3.0 or later* [online] [vid. 2021-01-02]. Dostupné z: https://spdx.org/licenses/GPL-3.0-or-later.html

GINNEKEN, Yenthe Van, 2015. What will change about the Odoo licensing in Odoo 9? In: *Odoo S.A.* [online]. [vid. 2021-01-11]. Dostupné z: https://www.odoo.com/forum/help-1/what-will-change-about-the-odoo-licensing-in-odoo-9-86439

GOOGLE, 2018. Introduction. *Material Design* [online] [vid. 2021-03-01]. Dostupné z: https://material.io/design/introduction#material-theming

GOOGLE, nedatováno. *Angular - Introduction to Angular concepts* [online] [vid. 2021--01-28]. Dostupné z: https://angular.io/guide/architecture

GORDON, Elyse Kolker, 2016. An Introduction to Isomorphic Web Application Architecture. *Medium* [online] [vid. 2021-01-22]. Dostupné z: https://medium.com/@ElyseKoGo/an-introduction-to-isomorphic-web-application-architecture-a8c81c42f59

GOSLING, James a Henry MCGILTON, 1996. *The Java Language Environment: Contents* [online]. květen 1996. [vid. 2021-01-22]. Dostupné z: https://www.oracle.com/java/technologies/language-environment.html

GRAY, Jim, 1981. The Transaction Concept: Virtues and Limitations. In: *Proceedings of Seventh International Conference on Very Large Databases*. B.m.: Tandem Computers Incorporated.

GREIF, Sascha a Raphaël BENITTE, 2019. *The State of JavaScript 2019* [online] [vid. 2020-12-20]. Dostupné z: https://2019.stateofjs.com/

HAERDER, Theo a Andreas REUTER, 1983. Principles of transaction-oriented database recovery. *ACM Computing Surveys* [online]. **15**(4), 287--317 [vid. 2021-01-15]. ISSN 0360-0300, 1557-7341. Dostupné z: doi:10.1145/289.291

HANEL, David, 2009. *Vývoj webových aplikací pomocí frameworku JavaServer Faces* [online]. B.m. [vid. 2021-02-01]. bakalářská práce. Vysoká škola ekonomická v Praze. Dostupné z: https://java.vse.cz/jsf/chunks/index.html

HARUTYUNYAN, Nairi, 2019. Reactive Programming. *Medium* [online]. [vid. 2021--01-28]. Dostupné z: https://nairihar.medium.com/reactive-programming-e4698b6ee3f

HAVERBEKE, Marijn, 2019. *Eloquent JavaScript: a modern introduction to programming* [online] [vid. 2020-12-19]. ISBN 978-1-59327-951-6. Dostupné z: http://proquest.safaribooksonline.com/?fpi=9781492071198

HIPP, Richard a SQLITE DEVELOPMENT TEAM, nedatováno. ALTER TABLE. *SQLite Documentation* [online] [vid. 2021-02-27]. Dostupné z: https://sqlite.org/lang_altertable.html

JOBBER, 2019a. About Us | Jobber Mobile Service Software. *Jobber* [online] [vid. 2021--01-10]. Dostupné z: https://getjobber.com/about/

JOBBER, 2019b. The #1 Home Service Software. *Jobber* [online] [vid. 2021-01-11]. Dostupné z: https://getjobber.com/features/

KHUDOIBERDIEV, Umed, nedatováno. TypeORM - Amazing ORM for TypeScript and JavaScript (ES7, ES6, ES5). Supports MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, Oracle, WebSQL databases. Works in NodeJS, Browser, Ionic, Cordova and Electron platforms. *TypeORM Documentation* [online] [vid. 2021-02-27]. Dostupné z: https://typeorm.io/#/

KINCAID, Jason, 2009. Google's Go: A New Programming Language That's Python Meets C++. *TechCrunch* [online]. [vid. 2021-01-24]. Dostupné z: https://social.techcrunch.com/2009/11/10/google-go-language/

KOTELKO, Blazej, Jen OLSON, Eva DUPONT a Kumar VIVEK, 2018. *Business Central everywhere - Release Notes* [online]. [vid. 2021-01-09]. Dostupné z: https://docs.microsoft.com/en-us/business-applications-release-notes/october18/dynamics365-business-central/business-central-everywhere

KRYSKI, Eric, 2017. Why we built the best web framework you've probably never heard of (until now). *Medium* [online] [vid. 2021-01-26]. Dostupné z: https://blog.feathersjs.com/why-we-built-the-best-web-framework-you-ve-probably-never-heard-of-until-now-176afc5c6aac

KRYSKI, Eric, 2018. Introducing Feathers 2.0. *Medium* [online] [vid. 2021-01-26]. Dostupné z: https://blog.feathersjs.com/introducing-feathers-2-0-aae8ae8e7920

LARAVEL LLC, nedatováno. Installation. *Laravel - The PHP Framework For Web Artisans* [online] [vid. 2021-01-25]. Dostupné z: https://laravel.com/docs/8.x

LI, Evan, 2021. *Github Top 100 Stars* [online]. 27. leden 2021. [vid. 2021-01-29]. Dostupné z: https://github.com/EvanLi/Github-Ranking

MASNER, Jan, 2020a. Cascading Styles Sheets. In: *Web design*. B.m.

MASNER, Jan, 2020b. Dynamické technologie. In: *Web design*. B.m.

MASNER, Jan, 2020c. HTML. In: *Web design*. B.m.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 1999. *MIT License* [online] [vid. 2021-01-02]. Dostupné z: https://spdx.org/licenses/MIT.html

MCKINNEY, Rand, Hage YAAPA, Dougles WILSON, İsmail ARILIK a Andrey PECHKUROV, 2017. *Express middleware* [online] [vid. 2021-01-26]. Dostupné z: https://expressjs.com/en/resources/middleware.html

MDN CONTRIBUTORS, nedatováno a. Django introduction - Learn web development. *MDN Web Docs* [online] [vid. 2021-01-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction

MDN CONTRIBUTORS, nedatováno b. Express/Node introduction - Learn web development. *MDN Web Docs* [online] [vid. 2021-01-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

MDN CONTRIBUTORS, nedatováno c. HTML5 - Developer guides. *MDN Web Docs* [online] [vid. 2021-01-27]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

MICROSOFT, nedatováno a. What is ASP.NET Core? A cross-platform web-development framework. *Microsoft .NET* [online] [vid. 2021-01-25]. Dostupné z: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet-core

MICROSOFT, nedatováno b. What is ASP.NET? *Microsoft .NET* [online] [vid. 2021--01-25]. Dostupné z: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet

MIKOWSKI, Michael S. a Gregory D. BENSON, 2014. *Single page web applications: Javascript end-to-end*. Shelter Island, NY: Manning. ISBN 978-1-61729-075-6.

MOLNÁR, Zdeněk, 2009. *Podnikové informační systémy*. V Praze: České vysoké učení technické. ISBN 978-80-01-04380-6.

MONDAY.COM, nedatováno a. *monday.com product page* [online] [vid. 2021-01-10]. Dostupné z: https://monday.com/product/

MONDAY.COM, nedatováno b. Our Story - monday.com. *monday.com in the news* [online] [vid. 2021-01-10]. Dostupné z: https://monday.com/p/about/

MONGODB, nedatováno. Sharding. *MongoDB Manual* [online] [vid. 2021-01-17]. Dostupné z: https://docs.mongodb.com/manual/sharding

MUSGRAVE, David, 2009. Great Plains Historical Timeline. *MSDN Blogs* [online]. [vid. 2021-01-09]. Dostupné z: https://web.archive.org/web/20101114115107/http://blogs.msdn.com/b/developingfordynamicsgp/archive/2009/03/13/great-plains-historical-timeline.aspx

MYSLIWIEC, Kamil, nedatováno a. Documentation. *NestJS - A progressive Node.js framework* [online] [vid. 2021-01-26]. Dostupné z: https://docs.nestjs.com

MYSLIWIEC, Kamil, nedatováno b. Modules. *NestJS - A progressive Node.js framework* [online] [vid. 2021-01-26]. Dostupné z: https://docs.nestjs.com

NEIJMANN, Daisy L, 2012. *Colloquial Icelandic The Complete Course for Beginners*. Hoboken: Taylor and Francis. ISBN 978-0-415-20706-5 978-0-203-99545-7.

NEXEDI, 2016. *ERP5 Unified Business Model* [online] [vid. 2021-01-13]. Dostupné z: https://www.erp5.com/P-ERP5.Com.UBM.Technology

NEXEDI, nedatováno. *Executive - Is ERP5 for me?* [online] [vid. 2021-01-13]. Dostupné z: https://www.erp5.com/executive

NPM, 2021a. About npm. *npm Docs* [online] [vid. 2021-01-22]. Dostupné z: https://docs.npmjs.com/about-npm

NPM, 2021b. Welcome page. *npm* [online] [vid. 2021-01-24]. Dostupné z: https://www.npmjs.com/

OSETSKYI, Victor, 2018. *ERP Technologies List: What is Right for Your Project? / Existek Blog* [online]. [vid. 2021-01-19]. Dostupné z: https://existek.com/blog/erp-technologies-list-and-choose-erp-technology/

PATEL, Priyesh, 2018. What exactly is Node.js? *freeCodeCamp.org* [online] [vid. 2021--01-24]. Dostupné z: https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/

PHP DOCUMENTATION GROUP, 2021. PHP: What can PHP do? *PHP Manual* [online] [vid. 2021-01-23]. Dostupné z: https://www.php.net/manual/en/intro-whatcando.php

PINCKAERS, Fabien, 2013. The Odoo story. *Odoo S.A.* [online]. [vid. 2021-01-11]. Dostupné z: https://www.odoo.com/blog/odoo-news-5/the-odoo-story-56

PYTHON SOFTWARE FOUNDATION, nedatováno. What is Python? Executive Summary. *Python.org* [online] [vid. 2021-01-23]. Dostupné z: https://www.python.org/doc/essays/blurb/

RAINER, R. Kelly, Brad PRINCE a Casey G. CEGIELSKI, 2014. *Introduction to information systems: supporting and transforming business*. Fifth edition. Hoboken, NJ: John Wiley and Sons, Inc. ISBN 978-1-118-67436-9.

RASCIA, Tania, 2018. *React Tutorial: An Overview and Walkthrough* [online]. [vid. 2021-01-28]. Dostupné z: https://www.taniarascia.com/getting-started-with-react/

RICHARD, Sam a Pete LEPAGE, 2020. What are Progressive Web Apps? *web.dev* [online] [vid. 2021-01-21]. Dostupné z: https://web.dev/progressive-web-apps/

RICHARDSON, Leonard a Sam RUBY, 2007. *RESTful web services*. Farnham: O'Reilly. ISBN 978-0-596-52926-0.

RINALDI, Brian a an O'Reilly Media Company SAFARI, 2015. *Static Site Generators* [online] [vid. 2021-01-21]. Dostupné z: https://www.safaribooksonline.com/library/view//9781492048558/?ar

SAP, nedatováno a. Company Information | About SAP SE. *SAP* [online] [vid. 2021--01-14]. Dostupné z: https://www.sap.com/corporate/en/company.html

SAP, nedatováno b. Real-time data | SAP History | About SAP SE. *SAP* [online] [vid. 2021-01-14]. Dostupné z: https://www.sap.com/corporate/en/company/history/2001-2010.html

SAP, nedatováno c. SAP Business One | ERP software pro malé firmy. *SAP* [online] [vid. 2021-01-14]. Dostupné z: https://www.sap.com/cz/products/business-one.html

SAP, nedatováno d. SAP Customer Reviews & Stories | Software & Technology Solutions. *SAP* [online] [vid. 2021-01-14]. Dostupné z: https://www.sap.com/about/customer-stories.html

SAP, nedatováno e. The early years | SAP History | About SAP SE. *SAP* [online] [vid. 2021-01-14]. Dostupné z: https://www.sap.com/corporate/en/company/history/1972-1980.html

SMARTBEAR SOFTWARE, nedatováno. OpenAPI Specification - Version 3.0.3. *Swagger* [online] [vid. 2021-01-30]. Dostupné z: https://swagger.io/specification/

THE APACHE SOFTWARE FOUNDATION, nedatováno. *The Apache OFBiz® Project* [online] [vid. 2021-01-12]. Dostupné z: https://ofbiz.apache.org/

TYPESTACK, 2021. typestack/routing-controllers. *Github* [online] [vid. 2021-02-26]. Dostupné z: https://github.com/typestack/routing-controllers

VERCEL, nedatováno a. API Routes: Introduction. *Next.js* [online] [vid. 2021-01-26]. Dostupné z: https://nextjs.org/docs/api-routes/introduction

VERCEL, nedatováno b. Learn. *Next.js* [online] [vid. 2021-01-26]. Dostupné z: https://nextjs.org/learn

VERCEL, nedatováno c. Routing: Dynamic Routes. *Next.js* [online] [vid. 2021-01-26]. Dostupné z: https://nextjs.org/docs/routing/dynamic-routes

VMWARE, INC, nedatováno a. Spring Boot. *Spring* [online] [vid. 2021-01-25]. Dostupné z: https://spring.io/projects/spring-boot

VMWARE, INC, nedatováno b. Spring Projects. *Spring* [online] [vid. 2021-01-25]. Dostupné z: https://spring.io/projects

VOSTROVSKÝ, Václav, ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE a KATEDRA INFORMAČNÍHO INŽENÝRSTVÍ, 2004. *Vytváření databází v ORACLE*. V Praze: Česká zemědělská univerzita, Provozně ekonomická fakulta. ISBN 978-80--213-1191-6.

VUEJS, nedatováno a. Computed Properties and Watchers. *Vue.js* [online] [vid. 2021--01-29]. Dostupné z: https://v3.vuejs.org/guide/computed.html

VUEJS, nedatováno b. Installation. *Vue.js* [online] [vid. 2021-01-29]. Dostupné z: https://v3.vuejs.org/guide/installation.html#with-a-bundler

VUEJS, nedatováno c. Single File Components. *Vue.js* [online] [vid. 2021-01-29]. Dostupné z: https://v3.vuejs.org/guide/single-file-component.html#introduction

VUETIFY, OPEN SOURCE DEVELOPERS a CONTRIBUTORS, nedatováno. Snackbar component. *Vuetify* [online] [vid. 2021-02-28]. Dostupné z: https://vuetifyjs.com/en/components/snackbars/

W3C TECHNICAL ARCHITECTURE GROUP, 2004. *Architecture of the World Wide Web, Volume One* [online]. 15. prosinec 2004. [vid. 2021-01-20]. Dostupné z: http://www.w3.org/TR/2004/REC-webarch-20041215/

WEBER, Sebastian, 2019. Why Lerna and Yarn Workspaces is a Perfect Match for Building Mono-Repos – A Close Look at Features and Performance. *Sebastian Weber* [online]. [vid. 2021-02-27]. Dostupné z: http://doppelmutzi.github.io/monorepo-lerna-yarn-workspaces/

WRIGHT, Nicola, 2018. A brief history of Microsoft Dynamics. *Nigel Frank* [online]. [vid. 2021-01-09]. Dostupné z: https://www.nigelfrank.com/blog/a-brief-history-of-microsoft-dynamics/

ZAPPONI, Carlo, Q4 2020. GitHut - Programming Languages and GitHub. *GitHut* [online] [vid. 2021-01-22]. Dostupné z: https://githut.info/

# 8 Přílohy

**bokari-diagram.png**:
Diagram databázových entit a jejich relací ve formátu PNG

**bokari-diagram.uml**:
Diagram databázových entit a jejich relací ve formátu PUML

**bokari-source.zip**:
Archiv se zdrojovým kódem celé aplikace. Též dostupný na github.com/tajnymag/bokari