VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

APLIKACE PRO UŽIVATELSKOU NAVIGACI V DATABÁZI SE ZADANÝM SCHÉMATEM

INTERACTIVE NAVIGATION THROUGH AN USER DATABASE IN SQLITE3

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR VRTAL**
AUTHOR

VEDOUCÍ PRÁCE **MARTIN HRUBÝ, Ing. Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Vrtal Petr**
Program: Informační technologie
Název: **Aplikace pro uživatelskou navigaci v databázi se zadaným schematem**
Interactive Navigation through an User Database in Sqlite3
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte databázi Sqlite3. Prostudujte programování webových aplikací.
2. Navrhněte aplikaci, která uživateli umožní procházet zadanou databázi se známým schématem. Zaměřte se na přechody mezi tabulkami a vytváření častých pohledů na databázi.
3. Implementujte aplikaci.
4. Testujte aplikaci. Zaměřte se na robustnost aplikace vůči chybným vstupům od uživatele. Vyhodnoťte rychlost práce s aplikací.

Literatura:

- Dle pokynů vedoucího.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hrubý Martin, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Cílem této práce je vytvořit nástroj pro podporu vývojářů, v podobě webové aplikace, schopný interaktivního a intuitivního procházení databáze typu SQLite, představující uživatelská data, jež má předem definované schéma. Toho je docíleno za pomoci vytvoření série častých pohledů na tuto databázi, jenž prohlížení dat pro každou databázovou tabulku určitým způsobem typizují. Mezi pohledy definujeme vzájemný, jednotný mechanismus přechodů, které uživateli zpřehledňují navigaci v datech databáze. Webová aplikace je implementována pomocí PHP frameworku Laravel. Uživatelské rozhraní bylo vytvořené převážně s využitím frameworku Livewire a jeho vzájemně propojených komponent. Důležitou součástí práce bylo sérií testování zjistit míru odolnosti aplikace vůči chybným vstupům od uživatele a také rychlost práce uživatele s aplikací. Výsledná webová aplikace umožňuje velmi konkrétní skupině uživatelů přehlednou orientaci v datech databáze konkrétní výrobní úlohy, jenž na server vloží a aplikace jim nabídne možnost si tyto data prohlížet.

Abstract

The aim of this thesis is to create a web application enhancing developer productivity, capable of interactive and intuitive browsing of a SQLite file database with a specified schema, representing user data. The enhancement is achieved specificaly by using a series of common views of this database and mechanisms for transition between these views. The web application is build using the Laravel PHP framework. User interface functionality is build mainly using Livewire framework by creating its interconnected components. An important part of the work was to determine by seeries of tests the level of resistence of the application against invalid user input as well as the speed analysis of the user's work with the application. The resulting web application allows a very specific group of users a clear orientation in the data of a specific production task, which is uploaded to the server by the users themselfs and the application offers them the option to view this data.

Klíčová slova

Rozvrhování výroby, webová aplikace, nástroj pro podporu vývojářů, uživatelské rozhraní, Sqlite, databázové schéma, typizované databázové pohledy, přechody mezi databázovými tabulkami, Laravel, Livewire, Tailwind CSS

Keywords

Manufacturing planning, web application, developer productivity tool, user interface, Sqlite, database schema, typified database views, transitions between database tables, Laravel, Livewire, Tailwind CSS

Citace

VRTAL, Petr. *Aplikace pro uživatelskou navigaci v databázi se zadaným schématem.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Martin Hrubý, Ing. Ph.D.

Aplikace pro uživatelskou navigaci v databázi se zadaným schématem

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Martina Hrubého, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr Vrtal
11. května 2021

Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce panu Ing. Martinu Hrubému, Ph.D. za odbornou pomoc a cenné rady při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Ve vývojářské praxi vývojáři pracují s databázovými aplikacemi, které mají čím dál tím složitější schéma tabulek. Často je to dáno komplexitou vyvíjeného systému nebo jen zkrátka tím, že dochází ke sběru a uchovávání velkého množství dat. Co by této skupině uživatelů pomohlo je vytvoření vývojářského nástroje, jenž by ulehčil a podpořil jejich práci s konkrétní aplikací. Mluvíme o pomoci zpřehlednění zobrazovaných dat, se kterými jejich aplikace pracuje a zjednodušením navigace v této databázi. Moje bakalářská práce se zabývá vytvořením nejvhodnějšího způsobu řešení čtení a navigace dat v databázi s konkrétním databázovým schématem pro jednu jedinou konkrétní aplikaci, kterou se myslí aplikace pro výpočet optimálních výrobních rozvrhů.

Tato databáze je specifická svou strukturou a vzájemnou provázaností tabulek. K jejímu pochopení bylo nejprve nutné prostudovat základní databázové schéma výrobní úlohy a vzájemnou hierarchii objektů v něm vystupujících. Poté již bylo možné navrhnout řešení, které by představovalo uživatelské rozhraní nad zadanou databází. Mým cílem však bylo, poponést použitelnost této aplikace o kousek dále, vymyšlením způsobu, jak databázi prezentovat formou typizovaných pohledů. Tedy namísto klasické tabulkové reprezentace, kterými by vývojář, potažmo uživatel, mohl na svá data nahlížet a zvýšit svoji produktivitu práce. Nejedná se ovšem o nástroj obecný, vhodný pro každého vývojáře, ale o nástroj, zacílený na jednu konkrétní aplikaci s konkrétním databázovým schématem pro velmi úzkou cílovou skupinu uživatelů.

Jako způsob řešení jsem si zvolil webovou aplikaci, nýbrž je dostupná pro uživatele kdykoliv, na kterékoliv platformě a není ji třeba instalovat na koncovém zařízení. Toto mé rozhodnutí bylo dále ovlivněno tím, že v dnešní době existuje široká škála různých aplikačních webových frameworků, jenž poskytují rychlý a spolehlivý vývoj webových aplikací a jsou ve svém výchozím stavu technologicky bohaté natolik, že dokáží naplnit veškeré požadavky na tuto aplikaci. V mém konkrétním případě jsem zvolil PHP aplikační framework Laravel, který se na tuto úlohu hodí především kvůli tomu, že v základu podporuje komunikaci s databází typu SQLite, mechanismus autentizace uživatelů, zabezpečení vůči nejčastějším útokům na webovou aplikaci a celkově se jedná o velmi dobře zdokumentovanou, výchozí platformu, na jejichž robustním a spolehlivém základě mohu svoji aplikaci stavět. Výsledné řešení webové aplikace má charakter vysoce specializovaného podpůrného nástroje, do které uživatel vkládá svá data a aplikace mu tato data umožní prohlížet typizovanými pohledy.

Ve finální části mé bakalářské práce jsem se zaměřil na důkladné prozkoumání celkové efektivity práce s tímto nástrojem. Zaměřil jsem se na měření doby odezvy u jednotlivých databázových pohledů, přičemž bylo mým cílem najít nedostatky v rychlosti jejich načítání. Současně jsem na různých databázových souborech a jiných vstupech ověřil, že nástroj je

odolný vůči chybným vstupům a funguje korektně. Celý nástroj byl nahrán na webový server a je k dispozici na adrese: https://opt3-browser.cz.

Kapitola 2

Podnikové informační systémy

Podnikové informační systémy (zkr. PIS) [22] jsou z pohledu podniku nebo organizace velmi užitečným nástrojem pro sjednocení všech funkčních oblastí podniku. Vzájemně kombinují a propojují oblasti **plánování**, **výroby**, **logistiky**, **personalistiky**, **prodeje** a **marketingu**.

Konceptem podnikového informačního systému je integrace všech těchto dílčích sektorů podniku a jejich aplikačních systémů do jediného integrovaného systému za účelem vylepšení stávajících podnikových procesů, jako např. efektivní synchronizace poptávky, zásobování a výroby, snížení celkových nákladů na výrobu, poskytování přesnějších termínů dodání zboží, zlepšení zákaznického servisu a další.

Na trhu podnikových informačních systémů existuje mnoho různých typů. Mezi často se vyskytující patří systém ERP. Systém **ERP** (angl. Enterprise Resource Planning) neboli systém plánování podnikových zdrojů [2] představuje jádro podnikového informačního systému. Organizacím umožňuje propojovat agendy jednotlivých oddělení organizace do jediného softwarového řešení, které zároveň integruje a uchovává všechna podniková data (transakce) těchto oddělení ve společné databázi. Data ze společné databáze jsou v reálném čase analyzována, monitorována a vzájemně sdílena v rámci společného podniku.

Výrobní informační systémy neboli **MES** (angl. Manufacturing Execution Systems) [21] jsou skupina informačních systémů, zodpovědných za **řízení produkce**. MES tvoří vrstvu mezi podnikovými systémy ERP a systémy pro řízení a automatizací výrobních procesů (systémy SCADA).

Mezi hlavní funkční činnosti systému MES patří mimo jiné:

1. *Detailní rozvrhování výroby* – optimální plánování výrobních sekvencí s ohledem na konkrétní podmínky (priority, technologické postupy, doba zpracování apod.) a disponibilitu zdrojů – tzn. sestavení zadání výrobní úlohy.

2. *Správa a přidělování výrobních zdrojů* – skupina funkcionalit zodpovědná za sledování a správu výrobních zdrojů. Těmi jsou považovány např. obsluha, stroje, nástroje atd.

3. *Dispečerské řízení* – řízení toku výroby, přidělováním dostupných zdrojů. Umožňuje okamžitou reakci na události vzniklé při výrobě a případné pozměnění výrobního plánu.

4. *Správa dokumentace* – veškeré nezbytné technické dokumentace, výrobní postupy a informační zdroje, týkající se výrobních procesů, jsou v případě potřeby v reálném čase dostupné pro personál z jediného centralizovaného místa.

5. *Sběr dat a jejich archivace.* – manuální nebo automatické snímání všech relevantních dat, souvisejících s výrobou.

6. *Výkonnostní analýza* – všechny důležité údaje spjaté s výrobou jsou uživateli v podobě statistik zobrazovány v reálném čase. To uživateli (operátorovi) umožňuje rychlou detekci problémů a vyhodnocení efektivity výroby.

2.1 Plánování a rozvrhování výroby

Plánování výroby je pro výrobu naprosto klíčové, avšak sestavení výrobního rozvrhu představuje výpočetní problém. Problémem je vypočíst výrobní rozvrh, který by splňoval všechna technologická kritéria výroby a zároveň by výsledný plán představoval co nejefektivnější řešení (např. z časového hlediska).

Jedna z dílčích činností systému MES je seřadit zakázky ze systému ERP do optimální výrobní sekvence za využití různě komplexních plánovacích algoritmů, simulací nebo genetických algoritmů. Výsledný produkt plánování výroby je tzv. fronta práce, která udává pořadí zpracování jednotlivých výrobních příkazů dané výrobní úlohy.

Existuje celá řada strategií plánování výroby. Mezi nejčastěji používané patří např. **zpětné plánování** (Reverse Planning) nebo **dopředné plánování** (Forward Planning) [21].

2.2 Vznik výchozích dat výrobní úlohy

Řešením netriviálního problému sestavení optimalizovaného výrobního rozvrhu se zabývá vedoucí mé bakalářské práce, pan Ph.D. Ing. Martin Hrubý, jehož výpočetní program na základě simulačně - kombinatoricko - optimalizačních výpočtů sestavuje optimalizovaný výrobní rozvrh. Výpočetní program funguje pro potřeby konkrétního spotřebitele, využívající systém MES, jehož je simulační výpočetní program pana Ing. Martina Hrubého, Ph.D. součástí.

Vstupní databázový soubor výrobní úlohy, figurující jako předpoklad zadání této bakalářské práce (resp. schéma tohoto databázového souboru), vzniká výstupem několika účastníků. Výchozím zdrojem dat databázového souboru výrobní úlohy je *IS MES*, nainstalovaný u konkrétního zákazníka, který po komunikaci s podnikovým informačním systémem *ERP* daného výrobního podniku z různých agend vypočte zadání výrobní úlohy. Pro vypočtené zadání výrobní úlohy je simulačním programem, popsaným výše, vypočten optimalizovaný rozvrh, který reprezentuje onen zmíněný vstupní databázový soubor.

2.3 Motivace k vytvoření aplikace

Reprezentace dat v databázových tabulkách může být nepřehledná. Obzvlášť je tomu tak v podnikových informačních systémech (typu MES a ERP) kde dochází ke sběru velkého množství dat. Přidejme k tomu fakt, že záznamy v tabulce využívají jako svůj identifikátor 128bitové UUID [9] a rázem se z databázové tabulky a celé databáze stává nepřehledný seznam strohých údajů, který je pro čtenáře špatně čitelný.

Motivací práce tedy je navrhnout a vytvořit uživateli, studující vygenerovaný rozvrh výrobní úlohy nástroj, jenž by **zpřehlednil** interpretaci suchých dat v databázových tabulkách výrobní úlohy, včetně provedení vhodné **substituce UUID** objektů, vystupujících

v konkrétní výrobní úloze. Toho docílíme tvorbou **typizovaných pohledů**, které představují jednotný způsob nahlížení na data výrobní úlohy.

2.4 Základní objekty výrobní úlohy

Celková struktura výchozí výrobní úlohy je hierarchická a skládá se z několika základních komponent. Počáteční výrobní úloha je zadaná jako množina výrobních příkazů, respektive jejich sled.

Výrobní příkazy

Výrobní příkaz představuje příkaz (požadavek) na výrobu konkrétně specifikovaného množství požadovaného produktu, jenž je nutno vyrobit k určitému termínu [21]. Výrobní příkazy se dále rozpadají na dvě podmnožiny, množinu **výrobních operací** (tzv. plných výrobních operací) a množinu **kooperací**, jenž představují výrobní operace vykonávané mimo rámec podniku.

Výrobní operace

Výrobní operace popisuje výrobu konkrétního produktu. Obsahuje nezbytné informace spojené s výrobou, jako je časová náročnost výroby, požadované množství do výroby atp. Výrobní operace se rozpadá na několik dílčích fázových operací, z nichž nejdůležitější je fázová operace představující samotnou výrobu produktu. Každá výrobní operace je vztažena k výrobnímu příkazu. K vykonání své činnosti si operace jak fázové, tak plné, zabírají zdroje [21].

Zdroje v úloze

Za zdroje v úloze považujeme stroje, nástroje a obsluhující personál. Zdroje jsou vždy spjaty s výrobními operacemi. Pro zdroje v úloze typu personál a stroj je typicky udávaná i jejich maximální kapacita [21].

Materiály v úloze

Materiál představuje ve výrobě jediný zdroj, který podléhá neustálému použití. Musí být tedy vedena evidence, která zaznamenává veškeré událostí v čase, v souvztažnosti s konzumací, produkcí, dostupností a dodávkou materiálů. Každý materiál se nachází v jednom a více skladech v různých počtech. Materiál je podobně, jak tomu bylo u zdrojů, vždy vztažen ke konkrétní výrobní operaci [21].

2.5 Výchozí schéma databázového souboru SQLite

Databázové soubory výrobních úloh sloužící jako vstup pro tuto aplikaci, představují kombinaci dat podnikových informačních systémů MES a ERP a z nich následně vypočtený výrobní rozvrh. Vstupní databáze výrobní úlohy vstupuje do aplikace s pevně zadaným schématem. Popis vzniku tohoto souboru byl podrobně rozebrán v sekci 2.2

Databázová tabulka cProp

Databázová tabulka *cProp*, zobrazená v tabulce 2.1, obsahuje dodatečné informace o kontextových kanálech vyplněné plnými operacemi a zdroji.

Význam jednotlivých atributů:

- *Type* – typ objektu kontextového kanálu (viz číselník níže)
  - **0** – Zdroj typu stroj
  - **1** – Plná operace
  - **2** – Zdroj typu nástroj
- *refID* – identifikátor objektu zaplňující kontextový kanál (plná operace, zdroj)
- *cChannel* – číslo kontextového kanálu
- *reqValue* – požadovaná hodnota
- *initValue* – počáteční hodnota
- *cTableID* – identifikátor tabulky s kontextovými kanály

Název atributu	Typ atributu	Povinnost výskytu atributu
Type	INTEGER	NOT NULL
refID	TEXT	NOT NULL
resourceRefID	TEXT	NULL
cChannel	INTEGER	NULL
Name	TEXT	NULL
reqValue	TEXT	NULL
initValue	TEXT	NULL
opcode	INTEGER	NULL
schedConst	INTEGER	NULL
validityConst	INTEGER	NULL
cTableID	TEXT	NULL
groupId	TEXT	NULL
linkedId	TEXT	NULL

Tabulka 2.1: Výčet atributů databázové tabulky *cProp*

Databázová tabulka JobInOperation

Databázová tabulka *JobInOperation*, zobrazená v tabulce 2.2, obsahuje seznam rozpracovaných operací a míru jejich rozpracovanosti. Dělí operace na aktuálně běžící a operace, které byly v minulosti již rozpracované.

Název atributu	Typ atributu	Povinnost výskytu atributu
type	INTEGER	NOT NULL
jobsId	TEXT	NULL
ressId	TEXT	NULL
amount	DOUBLE	NULL
tmConst	INTEGER	NULL

Tabulka 2.2: Výčet atributů databázové tabulky *JobInOperation*

Význam jednotlivých atributů:

- *type* – typ prováděné operace (viz číselník níže)
  - **0** – Výrobní operace je do jisté míry rozpracovaná (zahájeno v minulosti)
  - **1** – Výrobní operace je aktuálně běžící
- *jobsId* – identifikátor výrobní operace
- *ressId* – identifikátor zdroje využívaného operace
- *amount* – množství použitých zdrojů

Databázová tabulka JobResource

Databázová tabulka *JobResource*, zobrazená v tabulce 2.3, obsahuje požadavky operací na zdroje (angl. Job – Resource Request). JRR je vztažen buď k plné operaci – jakožto její primární zdroj nebo k fázové operaci (sekundární zdroj). Každá plná operace vyžaduje ke své definici alespoň jeden stroj. Nástroje, personál a materiál jsou přiděleny volitelně.

Požadavky plné operace na zdroje je strukturovaně:

- množina módů určených strojem (operace může být provedena na stroji **A** nebo **B** nebo **C** . . .)
- každém módu pak platí ostatní požadavky (*mode* = -1) na nástroje a personál
- zdroje jsou zabrány po celou dobu rozvrhu plné operace, tj. přes všechny její fáze

Požadavky fázové operace na zdroje:

- vyžadování zdroje pouze na provedení fázové operace (typicky personál na **INST** / **ROZJ** / **ITER**)

Význam jednotlivých atributů:

- *ressId* – identifikátor zdroje (stroj, nástroj, personál)
- *jobsId* – identifikátor aktivity (plná operace, fázová operace)
- *mode* – číslo módu stroje pro plnou operaci. Pro zdroje typu nástroj a personál nabývá toto číslo hodnoty -1
- *iCap* – požadovaná kapacita pro zdroj typu stroj
- *dCap* – požadovaná kapacita pro zdroj typu personál

Název atributu	Typ atributu	Povinnost výskytu atributu
ressid	TEXT	NULL
jobsId	TEXT	NOT NULL
mode	INTEGER	NOT NULL
priority	INTEGER	NULL
contextCode	TEXT	NULL
iCap	INTEGER	NULL
dCap	DOUBLE	NULL

Tabulka 2.3: Výčet atributů databázové tabulky *JobResource*

Databázová tabulka Jobs

Databázová tabulka *Jobs*, zobrazená v tabulce 2.4, představuje tabulku výrobních operací. Výrobní operace se dělí na tři typy: **plné operace**, **fázové operace** a **kooperace**. Kooperace je prováděna mimo rámec podniku, tudíž jsou pro účely rozvrhování její konkrétní fáze irelevantní, i přesto se tento typ výrobní operace rozvrhuje.

Plná operace ($type = 0$) se skládá z posloupnosti fázových operací, a to konkrétně typů: instalace (**INST**), rozjezd (**ROZJ**), výroba (**ITER**) a dokončení (**FIN**). Výskyt uvedených typů fázových operací je v této posloupnosti volitelný vyjma fázové operace typu výroba (**ITER**), která je typicky povinná.

Název atributu	Typ atributu	Povinnost výskytu atributu
sId	TEXT	NULL
type	INTEGER	NOT NULL
alternativeNumber	INTEGER	NULL
jobColor	INTEGER	NULL
duration	INTEGER	NULL
amount	DOUBLE	NULL
amountPackage	INTEGER	NULL
amountPerCycle	INTEGER	NULL
transportTime	INTEGER	NULL
productsId	TEXT	NULL
standingTime	INTEGER	NULL
rmOptions	TEXT	NULL

Tabulka 2.4: Výčet atributů databázové tabulky *Jobs*

Význam jednotlivých atributů:

- *sId* – identifikátor aktivity (fázové operace, plné operace, kooperace)
- *type* – typ aktivity (viz číselník níže)
  - **0** – Plná operace
  - **2** – Fázová operace instalace (INST)
  - **3** – Fázová operace rozjezd (ROZJ)
  - **4** – Fázová operace výroba (ITER)

5 – Fázová operace dokončení (FIN)

6 – Údržba

7 – Kooperace

- *jobsId* – identifikátor výrobní operace
- *ressId* – identifikátor zdroje využívaného operace
- *amount* – množství použitých zdrojů

Databázová tabulka LogOpts

Databázová tabulka *LogOpts*, zobrazená v tabulce 2.5, obsahuje veškerá nastavení software použitého pro optimalizaci výrobního rozvrhu. Nastavení reprezentují pár **klíč – hodnota**, přičemž hodnota atributu type je pro všechny záznamy této databázové tabulky implicitně nastavená na hodnotu **3**.

Název atributu	Typ atributu	Povinnost výskytu atributu
type	INTEGER	NULL
optChar	TEXT	NULL
optValue	TEXT	NULL

Tabulka 2.5: Výčet atributů databázové tabulky *LogOpts*

Význam jednotlivých atributů:

- *type* – typ nastavení
- *optChar* – unikátní název nastavení
- *OptValue* – hodnota nastavení

Databázová tabulka Materials

Databázová tabulka *Materials*, zobrazená v tabulce 2.6, obsahuje všechny materiály, vyskytující se ve výrobní úloze a jejich uložení v konkrétních skladech. Jeden typ materiálu se může vyskytovat ve více skladech, v různém množství.

Název atributu	Typ atributu	Povinnost výskytu atributu
sId	TEXT	NULL
whouseID	TEXT	NULL
type	INTEGER	NULL
amount	DOUBLE	NULL
matCapacity	DOUBLE	NULL

Tabulka 2.6: Výčet atributů databázové tabulky *Materials*

Význam jednotlivých atributů:

- *sId* – identifikátor materiálu
- *whouseID* – UUID skladu, ve kterém je materiál uložen
- *type* – typ materiálu
- *amount* – množství materiálu uloženého ve skladu
- *matCapacity* – kapacita skladu

Databázová tabulka MatEvents

Databázová tabulka *MatEvents*, zobrazená v tabulce 2.7, obsahuje záznamy o událostech, vyvolaných nad konkrétním materiálem. Události, které nad materiály nastávají jsou např. dodávka materiálu do skladu, produkce materiálu výrobní operací nebo jeho konzumace.

Název atributu	Typ atributu	Povinnost výskytu atributu
sId	TEXT	NULL
type	INTEGER	NOT NULL
whouseID	TEXT	NULL
jobId	TEXT	NULL
Amount	DOUBLE	NULL
tmConst	TEXT	NULL
matSpeedOption	INTEGER	NULL

Tabulka 2.7: Výčet atributů databázové tabulky *MatEvents*

Význam jednotlivých atributů:

- *sId* – identifikátor materiálu spjatý s událostí
- *type* – typ materiálu (viz číselník níže)
  - **0** – Dodávka do skladu (jednorázová)
  - **1** – Produkce
  - **2** – Konzumace materiálu výrobní operací
  - **3** – Neomezená dostupnost materiálu
- *whouseID* – identifikátor skladu, nad kterým je událost v souvislosti s materiálem prováděna
- *jobId* – identifikátor výrobní operace
- *Amount* – množství materiálu, význam tohoto atributu záleží na typu události
- *tmConst* – datum (časová konstanta) dané události, záleží na typu události

Databázová tabulka Ownership

Záznamy databázové tabulky *Ownership*, zobrazené v tabulce 2.8, definují vzájemné vztahy mezi objekty výrobní úlohy. Pro zadané schéma databázového souboru definujeme dva vztahy vlastnictví:

1. Vlastnický vztah mezi plnou a fázovou výrobní operací, přičemž kardinalita tohoto vztahu je 1:N.
2. Vlastnický vztah mezi výrobním příkazem a výrobní operací (plnou operací). Kardinalita tohoto vztahu je 1:1, kdy jeden výrobní příkaz odpovídá jedné výrobní operaci.

Název atributu	Typ atributu	Povinnost výskytu atributu
type	INTEGER	NOT NULL
fobjsId	TEXT	NULL
tobjsId	TEXT	NULL

Tabulka 2.8: Výčet atributů databázové tabulky *Ownership*

Význam jednotlivých atributů:

- *type* – typ vztahu (viz číselník níže)
  - **0** – Plná operace vlastní operaci fázovou
  - **1** – Výrobní příkaz (VP) vlastní plnou operaci
- *fobjsId* – identifikátor objektu vytvářejícího vztah
- *tobjsId* – identifikátor objektu ve vztahu

Databázová tabulka ResCalendar

Databázová tabulka *ResCalendar*, zobrazená v tabulce 2.9, obsahuje záznamy o **kalendáři dostupnosti** jednotlivých zdrojů ve výrobní úloze.

Název atributu	Typ atributu	Povinnost výskytu atributu
ressId	TEXT	NULL
tstart	TEXT	NOT NULL
tend	TEXT	NOT NULL
relavail	INTEGER	NULL
iCap	INTEGER	NULL
dCap	DOUBLE	NULL

Tabulka 2.9: Výčet atributů databázové tabulky *ResCalendar*

Význam jednotlivých atributů:

- *ressId* – identifikátor zdroje
- *tstart* – termín udávající začátek bloku dostupnosti zdroje

- *tend* – termín udávající konec bloku dostupnosti zdroje
- *relavail* – identifikátor objektu ve vztahu
- *iCap* – kapacita dostupnosti relevantní pro zdroj typu stroj
- *dCap* – kapacita dostupnosti relevantní pro zdroj typu personál

Databázová tabulka Resources

Databázová tabulka *Resources*, zobrazená v tabulce 2.10, obsahuje záznamy všech zdrojů, vyskytujících se ve výrobní úloze a jejich kapacity. Zdroje se dělí do tří kategorií na: **stroje**, **nástroje** a **personál**.

Název atributu	Typ atributu	Povinnost výskytu atributu
sId	TEXT	NULL
type	INTEGER	NOT NULL
parallelType	INTEGER	NULL
maxICap	INTEGER	NULL
parallelCap	INTEGER	NULL
maxDCap	DOUBLE	NULL
matCapacity	DOUBLE	NULL

Tabulka 2.10: Výčet atributů databázové tabulky *Resources*

Význam jednotlivých atributů:

- *sId* – identifikátor zdroje v úloze
- *type* – typ zdroje (viz číselník níže)
  - **0** – Stroj
  - **2** – Personál
  - **3** – Nástroj
- *maxICap* – maximální kapacita udávaná pro zdroj typu stroj
- *maxDCap* – maximální kapacita udávaná pro zdroj typu personál

Databázová tabulka VP

Databázová tabulka *VP*, zobrazená v tabulce 2.11, obsahuje záznamy všech výrobních příkazů (VP) a termíny jejich dokončení.

Název atributu	Typ atributu	Povinnost výskytu atributu
sId	TEXT	NULL
deadline	TEXT	NOT NULL
lrjs	INTEGER	NULL
matStrat	INTEGER	NULL
priority	INTEGER	NULL

Tabulka 2.11: Výčet atributů databázové tabulky *VP*

Význam jednotlivých atributů:

- *sId* – identifikátor výrobního příkazu
- *deadline* – termín dokončení (**deadline**) výrobního příkazu
- *matStrat* – maximální kapacita udávaná pro zdroj typu personál
- *priority* – uživatelsky zadaná priorita

Kapitola 3

Nástroje pro tvorbu této webové aplikace

Základními technologiemi pro tvorbu webových aplikací a webů jsou HTML, Javascript a CSS. Tyto technologie umožňují tvorbu front-endu webových stránek. Webové aplikace, ovšem vyžadují, aby se jejich obsah dynamicky měnil na základě vstupů a akcí, vyvolaných uživatelem. Musíme tedy aplikaci přidat logiku ve formě back-endu, která bude obstarávat žádosti na straně serveru. K tomu můžeme využít některého z programovacích jazyků, jenž umí komunikovat s webovým serverem, jako např. jazyk Ruby, PHP nebo Python [5]. Nicméně tvorba funkční webové aplikace je velmi komplexní úkol, skládající se z mnoha dílčích částí, které je třeba implementovat, než jsme vůbec schopni vykreslit jednoduchou dynamickou stránku (směrování, struktura zdrojových souborů aplikace atp.). Proto dnes čím dál tím víc programátorů dává přednost tuto základní, již naimplementovanou funkcionalitu v podobě webového aplikačního frameworku převzít a svým konkrétním řešením webové aplikace na tyto robustní základy navázat [3].

Existuje nepřeberné množství webových frameworků, ze kterých by si mohl programátor webové aplikace vybírat. Liší se ve svém návrhu, programovacím jazykem, architekturou, ale to důležité je, že se snaží programátorům urychlit vývoj webových aplikací pomocí předem připravených funkcionalit. Mezi nejpopulárnější frameworky PHP patří CakePHP, Codeigniter, Symphony a jeden z nejnovějších přírůstků do této skupiny – framework Laravel [3].

3.1 Laravel

Laravel [29] je open-source aplikační framework napsaný v programovacím jazyce *PHP*, který má za cíl zjednodušit, a především urychlit vývoj webových aplikací. Laravel vznikl v roce 2010, kdy jeho tvůrce, programátor Tailor Otwell, nebyl dlouhodobě spokojený s frameworkem *CodeIgniter* a jeho dlouhotrvající integrací nových funkcionalit jazyka *PHP* a dalších moderních technologií.

Tento framework se netají tím, že kombinuje osvědčené postupy ostatních aplikačních frameworků, jakým jsou např. *Ruby on Rails* a *Sinatra*. Největší vliv na vývoj měl framework *Symfony*, jehož moduly jsou přímo zabudovány jako stavební součásti samotného frameworku Laravel. Laravel se snaží nabídnout programátorovi co nejpříjemnější prostředí pro vývoj webových aplikací, především tím, že disponuje sadou běžných funkcionalit (modulů) již zaintegrovaných při samotné instalaci tohoto frameworku. Není po nás tedy nutné

opět tyto základní funkcionality, které během vývoje webových aplikací často vyžadujeme opakovaně implementovat, jako např. systém autentizace uživatelů, směrovač, mechanismus vytváření a posílání e-mailových zpráv apod.

Svou koncepcí nabídnout ty nejlepší, osvědčené praktiky ostatních frameworků a využitím populárních podpůrných systémů, se během několika málo let stal Laravel velmi využívaným PHP aplikačním frameworkem.

Důvodem, proč si autor práce zvolil právě tento webový framework pro sovu práci je ten, že se jedná o **robustní**, **rozšiřitelný** framework založený na jazyce PHP, který je velmi dobře zdokumentovaný.

3.1.1 Předpřipravená funkcionalita

Směrování (routing)

Jedná se o výchozí vstupní bod webové aplikace, kdy je od uživatele přijata žádost (angl. Request) pomocí některé z HTTP dotazovacích metod (kupř. GET nebo POST). Žádost je následně vyhodnocena a je jí na základě URL přiřazena definovaná korespondující routa nebo metoda řadiče (kontroleru), ze kterých následně uživatel dostává odpověď v podobě HTML dokumentu [19].

Routy lze dále sdružovat do skupin, aplikovat prefixy pro danou skupinu nebo jim přidružit middleware. Pro vstupující HTTP žádosti poskytuje middleware příhodný mechanismus pro jejich filtrování. Middleware může sloužit také ke kontrole a případné modifikaci odpovědi řadiče zasílané zpět uživateli. Používá se např. při kontrole, zda je uživatel autentizován (přihlášen do aplikace) či nikoliv [18].

Šablonovací systém Blade

Pohledy využívají ke své prezentaci šablonovací systém Blade. Ten umožňuje do obsahu dokumentu pohledu krom HTML kódu umisťovat speciální direktivy, které umožňují výpis hodnoty proměnné, tvorbu if – else bloků, iteraci nad daty nebo kompozici obsahu dvou pohledů dohromady (zahrnutím subpohledu). Na rozdíl od ostatních PHP šablonovacích systémů nezakazuje systém Blade výskyt samotného PHP kódu v šablonách, jelikož se ony samy do čistého PHP kompilují. Další výhodou je, že se takto zkompilované šablony ukládají do paměti, ze které se získávají pokaždé, dokud nedojde k jejich změně. To nám urychlí zpracování žádosti a navrácení dat uživateli [13].

Autentizace

Autentizace je proces, ve kterém uživatel ověřuje svoji identitu vůči webové aplikaci [1]. U některých webových aplikací panuje potřeba, povolit vstup do aplikace pouze konkrétní skupině uživatelů, popřípadě vytvořit zvlášť oddělenou sekci aplikace, přístupnou pouze těmto autentizovaným uživatelům. Přístup do těchto sekcí většinou rozšiřuje základní funkcionalitu webové aplikace.

Tvorba vlastního autentizačního systému může být komplikovaná, zdlouhavá, ba dokonce z hlediska bezpečnosti riskantní a nedostatečná. Proto Laravel tuto funkcionalitu implementuje ve svém základu a nabízí uživatelům hned několik bezpečnostních mechanismů (včetně autentizace) pro použití ve svých webových aplikacích [14].

Komunikace s databází

V dnešní době v podstatě téměř každá webová aplikace využívá nějakou formu databáze k perzistentnímu uchování dat. Laravel tedy podporuje rozhraní pro následující, nejrozšířenější typy databází [16]:

- MySQL 5.7+
- PostgreSQL 9.6+
- SQLite 3.8.8+
- SQL Server 2017+

V této aplikaci je pro komunikaci s databázemi využito rozhraní (driver) *SQLite*. Využívá se pro komunikaci jak s nahranými databázovými soubory od uživatele, tak s oddělenou databází uchovávající aplikační data.

Cache

Některé výpočetní operace nebo databázové dotazy na databázi, jenž obsahuje velký počet záznamů, mohou mít negativní vliv na rychlost odezvy webové aplikace. S využitím cache (kešování) lze výsledky náročných výpočtů nebo databázových dotazů uchovat po určitou dobu v paměti a odtud tato stejná data při následném dotazování získávat do doby jejich aktualizace. Nejčastěji se ke kešování používají datová úložiště jako jsou např. Redis, Memcached nebo DynamoDB [15].

3.1.2 Významné komponenty

Laravel je tvořen knihovnami třetích stran a také vlastními komponenty, přičemž hlavním úkolem frameworku je tyto komponenty vzájemně funkčně propojit a umožnit jednotné rozhraní pro jejich používání a konfiguraci, pomocí konfiguračních souborů [29].

Composer

Composer je nástroj[1], využívající se pro správu závislostí (knihoven) pro jazyk PHP. Každý projekt založený v PHP, používající tento nástroj obsahuje svou vlastní sadu závislostí, ty se posléze dají uceleně a hromadně spravovat (instalace, aktualizace atp.). Umožňuje nám explicitně deklarovat knihovny na kterých projekt funkčně závisí. Funguje na stejném principu jako nástroj npm[2] pro Node.js [4].

Artisan CLI

Během vývoje webových aplikací vytváříme mnoho nových souborů. Ty většinou obsahují identický strukturální základ, respektive stejnou syntaxi. Pro vytváření těchto tzv. skeletonových souborů a také provádění často se opakujících rutinních úkonů, souvisejících s aplikačním vývojovým prostředím, existuje v Laravel frameworku speciální příkazový řádek (CLI) jménem *Artisan*. Artisan [29] je ve svém návrhu jen nadstavbou nad komponentou *Symfony console*. Namísto ručního vykonání daného úkolu programátorem nám

[1] Oficiální stránka nástroje Composer - https://getcomposer.org/doc/00-intro.md
[2] npm – Node Package Manager – https://www.npmjs.com/about

tento nástroj umožňuje využít některý z předdefinovaných příkazů, jehož spuštěním danou činnost uskutečníme. Například vytvoření konkrétního typu třídy a jejího souboru (model, řadič, komponentu) nebo spuštění migrace databáze. Jedním z nejdůležitějších je příkaz `php artisan serve`, který zajistí vytvoření a spuštění lokálního vývojového prostředí webové aplikace nastartováním vestavěného PHP serveru na adrese: `http://localhost:8000/`, portu 8000 (výchozí).

Dále máme možnost, v případě potřeby, nadefinovat vlastní příkazy, např. pro vytvoření administrátorského účtu aplikace (popsáno v sekci 4.4). Seznam definovaných příkazů lze zobrazit pomocí příkazu `php artisan list`.

Eloquent ORM

ORM neboli objektově relační mapování (angl. Object - Relational Mapping) je způsob získávání databázových údajů (dat), za pomocí objektově orientovaného paradigmatu, místo nutnosti použití jazyka SQL [3]. Jedná se o mapování dat databázové tabulky na objekty objektově orientovaného programovacího jazyka, v našem případě PHP [29].

V objektově orientovaných jazycích představuje entita (objekt nebo proces reálného světa) instanci konkrétní třídy. Jinak tomu je u relačních databází, kdy řádek tabulky odpovídá jedné entitě objektu. Cílem ORM je tyto dvě rozdílně reprezentované entity vzájemně synchronizovat a zajistit persistenci uchovávaných dat a jednotné rozhraní pro práci s entitou (objektem) v objektově orientovaném jazyce [6].

Eloquent ORM je objektově relační mapovač (angl. Object-relation-mapper), který ulehčuje práci s databází. Pro každou tabulku v databázi vytváříme korespondující model (třídu), pomocí něhož je nám umožněna interakce s danou databázovou tabulkou. Nad instancí této třídy (modelu) můžeme provádět operace typu zápis, čtení, aktualizace a mazání, známé také pod pojmem CRUD (create, read, update, delete) [17].

3.1.3 Architektura MVC

Model - View - Controller (zkráceně MVC) je architektonický vzor, dnes hojně užíván především k vývoji webových aplikací, neboť jeho koncepce umožňuje rychlý a inkrementální vývoj. Silným rysem architektury je separace aplikační a prezentační vrstvy aplikace [7]. Framework Laravel jej využívá jako svůj výchozí architektonický vzor [29].

Model (model)

Doménový model (zkráceně jen model) je hlavní komponenta architektury MVC. Model reprezentuje více než jen data a logiku, které na něm operují. Slouží ke správě informací v aplikaci a také jako abstrakce nějakého procesu nebo systému z reálného světa. Zachycuje nejen stav procesu nebo systému, ale také podstatu toho, jak systém funguje. Díky tomu je při definování modelů velmi snadné používat techniky modelování. Pro každou skupinu se sebou nesouvisejících dat vytváříme samostatný model (např. model databázového souboru, uživatele, výrobní úlohy apod.) [23].

Modely jsou v aplikací reprezentovanými třídami dědícími od bázové třídy `Model`, standardně uloženy ve složce `app/Models` v adresářové struktuře aplikace. Model většinou koresponduje s jednou tabulkou databáze a pomocí jeho instance jsme schopni korespondující záznam v tabulce modifikovat (viz CRUD operace v sekci 3.1.2).

View (pohled)

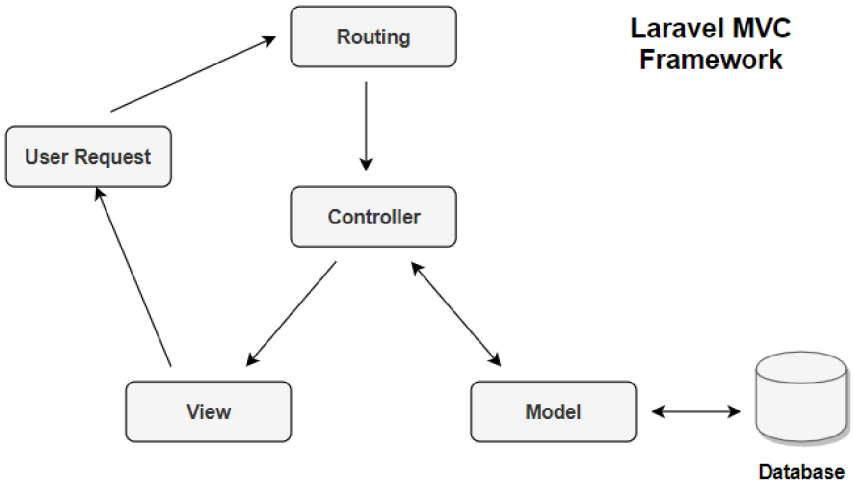
Jedná se o vrstvu architektury zobrazující uživateli data konkrétního modelu v podobě interaktivní prezentace. V případě, kdy nastane změna modelových dat, dojde k aktualizaci pohledu [23].

Pohledy v Laravel frameworku umožňují rozdělit HTML kód stránky do vícero souborů. Pohledy využívají ke své prezentaci šablonovací systém Blade. Ten umožňuje do obsahu dokumentu pohledu vyjma HTML kódu umisťovat speciální direktivy, které umožňují např. výpis hodnoty proměnné, tvorbu if – else bloků, iteraci nad daty anebo kompozici obsahu dvou pohledů dohromady (zahrnutím jiného pohledu). Na rozdíl od ostatních PHP šablonovacích systémů nezakazuje systém Blade výskyt samotného PHP kódu v šablonách, jelikož se samotné šablony kompilují do čistého PHP kódu. Další výhodou je, že se takto zkompilované šablony ukládají do paměti, ze které se získávají pokaždé, dokud nedojde k jejich změně. To nám urychlí zpracování žádosti a navrácení dat uživateli [13].

Controller (řadič)

Řadič představuje komunikační rozhraní mezi aplikací a uživatelem. Uživatel předává řadiči vstupní data a na základě těchto vstupů vyvolává řadič akce u příslušných modelů a pohledů [23]. Samotné předání vstupních dat je uskutečněno pomocí procesu směrování (detailněji popsáno v kapitole 3.1.1 Směrování), který předchází volání dotyčného řadiče, respektive jeho konkrétní akce. Řadič nejprve komunikuje s modelem, jenž mu navrací relevantní data. Ty jsou následně zakomponovány do pohledu, jehož výsledná interpretace je zaslána zpět uživateli [19].

Řadiče jsou v Laravel implementovány třídami dědícími od bázové třídy `Controller`.



Obrázek 3.1: Schéma architektury MVC ve frameworku Laravel (převzato z: https://interview-guru.club/laravel)

K vytvoření jednotlivých komponent architektury MVC jsme schopni využít příkazové řádky *Artisan CLI* (popsané v sekci 3.1.2), který nám pomocí příkazu vytvoří potřebné zdrojové soubory na definovaných místech v adresářové struktuře:

- `php artisan make:model NázevModelu` – vytvoří novou modelovou třídu `Eloquet`

- `php artisan make:view NázevPohledu` – vytvoří nový soubor pohledu
- `php artisan make:controller NázevŘadiče` – vytvoří novou třídu řadiče odvozenou od bázové třídy `Controller`

3.1.4 Modularita

Ne veškerá funkcionalita frameworku využívaná jedním uživatelem, musí být také potřebná pro účely uživatele druhého. Proto Laravel zvolil přístup veškerou, jinou něž základní funkcionalitu rozdělit do zvlášť udržovaných balíčků, které programátor může či nemusí ve svém projektu použít. Tím dojde k markantnímu úbytku nepotřebného kódu (funkcionality) a také k urychlení aplikace [29].

Laravel Jetstream

Laravel Jetstream je balíček pro framework Laravel, implementující základ pro administrační část aplikace a funkcí, jako jsou např. implementace autentizace, dvou-faktorové ověření, správa sessions a další. Tento balíček byl využit pro implementaci aplikace, jejíž tvorbu tato bakalářská práce popisuje. Konkrétně byla při instalaci zvolena konfigurace kombinující technologie Livewire a Blade [10].

3.2 Dynamické UI pomocí frameworku Livewire

Tvorba dynamických uživatelských rozhraní je u dnešních moderních webových aplikací téměř nezbytností. Tento fakt přidává na komplexitě implementace webové aplikace. Programátorovi přidává nutnost rozšíření znalostí o některý z front-endovým frameworků pro tvorbu dynamických uživatelských rozhraní, jakými jsou např. Vue.js nebo React [5]. V případě použití frameworku Livewire tato nutnost odpadá.

Livewire [20] je plnohodnotný framework, specificky vytvořený pro Laravel, který umožňuje snadnou tvorbu dynamických webových uživatelských rozhraní bez nutnosti znalosti ostatních frameworků. Pro tvorbu stačí pouze předešlá znalost aplikačního frameworku Laravel a programovacího jazyka PHP.

Livewire umožňuje vytvářet komponenty, které v případě vyvolání události (např. stisk tlačítka nebo změna obsahu vstupního pole formuláře) klientem v podobě uživatele, asynchronně komunikují se serverem pomocí AJAX[3].

Komponenty spolu komunikují za pomoci událostí, vyvolaných funkcí:

```
emit('nazev-udalosti')
```

a listenerů, které naslouchají těmto událostem a volají příslušné metody naslouchající komponenty.

Komponent typu Livewire se při svém vytváření rozděluje na dva soubory:

(a) Soubor **Komponent.php** – uložený přímo nebo v podsložce adresáře `app/Http/Livewire`, definující veškerou logiku komponenty.

(b) Soubor **komponent.blade.php** – uložený přímo nebo v podsložce adresáře `resources/views/livewire`, jenž definuje vzhled komponenty, pomocí jazyku HTML nebo eventuálně s využitím šablonovacího systému Blade.

[3] AJAX (Asynchronous JavaScript and XML) – Asynchronní JavaScript a XML

Jak Livewire funguje:

1. Livewire vytiskne na stránce počáteční obsah komponenty (HTML) a vyrenderuje jej společně se stránkou, tím zajistí existenci komponenty v dokumentu, a tudíž optimalizaci pro webové prohlížeče (SEO – search engine optimization [4]).

2. V případě interakce uživatele s uživatelským rozhraním (např. modifikace vstupního pole formuláře) je serveru zaslána AJAX[5] žádost, obsahující pozměněná data.

3. Server překreslí příslušnou komponentu a odpoví novým HTML.

4. Ve finále dojde k pozměnění objektového modelu dokumentu DOM[6], na základě rozpoznaných změn.

Komponenty umisťujeme do pohledů mezi HTML kód, pomocí speciální direktivy `<livewire nazev-komponenty />` nebo `@livewire('nazev-komponenty')`. Můžeme pomocí nich rovnou definovat celé stránky (pohledy), nalinkováním komponentu příslušné routě v souboru `routes/web.php`.

Komponenty lze dále vnořovat (includovat) do sebe nebo pomocí nich rovnou definovat celé stránky, kdy Livewire komponenta odpovídá příslušné routě (cestě) ve směrovači.

Komponenty (jejich zdrojové soubory) lze snadno vytvářet s využitím příkazového řádku *Artisan* (popsaného v sekci 3.1.2) příkazem:

```
php artisan make:livewire nazev-komponenty
```

3.3 Databáze SQLite

SQLite je relační databázový systém. Relační databázové systémy se využívají k ukládání uživatelsky definovaných záznamů do tabulek. Vyjma ukládání slouží také ke zpracování komplexních dotazů (angl. query commands) nad těmito záznamy, které kombinují data z více tabulek pro tvorbu souhrnných dat a manipulují s daty. Základními objekty relační databáze jsou tabulky. Jednotlivé řádky v nich uložené chápeme jako záznamy [11]. SQLite je distribuován pod licencí public domain, tzn., že je volně šiřitelný, a tudíž i nejpopulárnější databázový systém [8].

3.3.1 Vlastnosti SQLite

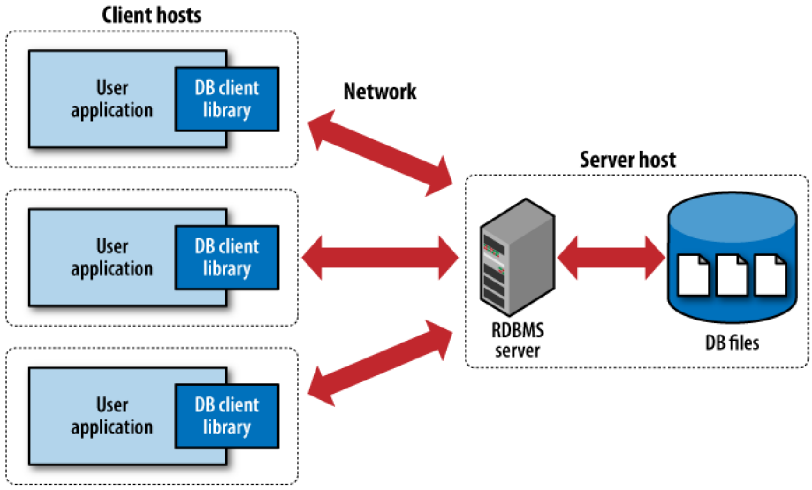
Absence serveru

Většina systémů řízení báze dat – RDBMS (angl. Relational Database Management Systems) je implementována architekturou klient – server [11]. Programy, které chtějí přistoupit k databázi komunikují se serverem pomocí některého druhu mezi procesové komunikace (typicky protokol TCP/IP) [27]. Databázový server se obvykle skládá z několika procesů, které za současného běhu spravují spojení s klientem, zápis a čtení ze souboru, kešování, optimalizaci dotazu a zpracování dotazu. Typická databáze se skládá z velkého množství souborů, organizovaných do jedné nebo více adresářových struktur, nacházející se na serverovém uložišti. Všechny tyto dílčí části vyžadují podporu u uživatelského zařízení.

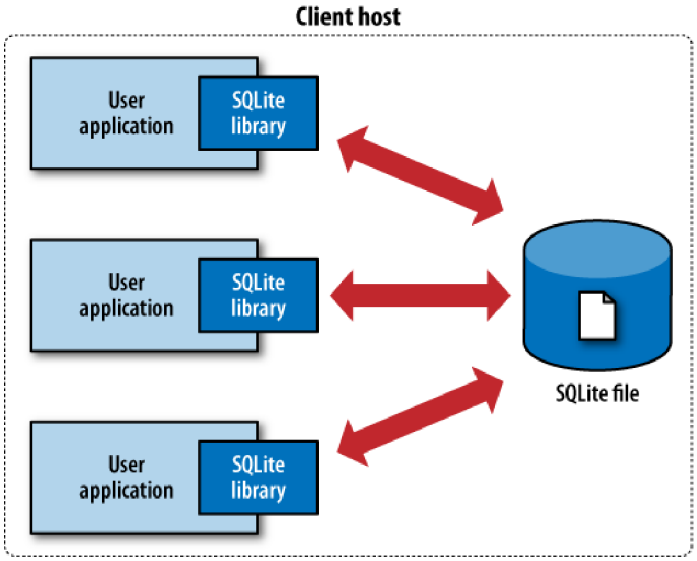
[4] SEO – optimalizace pro prohlížeče
[5] AJAX (Asynchronous JavaScript and XML) – Asynchronní JavaScript a XML
[6] DOM – Document Object Model

Pro přístup k databázi jsou od databázového vendora k dispozici uživatelské softwarové knihovny, které musí být integrovány do každé uživatelské aplikace, která požaduje přístup k databázovému serveru. Tyto knihovny poskytují API pro připojení k databázovému serveru, spolu se spouštěním databázových dotazů a příkazů [30]. Tuto architekturu popisuje obrázek 3.2.

SQLite neimplementuje tutéž architekturu. Celý databázový systém je integrován do uživatelské aplikace, která si přeje komunikovat s databází. Jediným sdíleným prostředkem mezi aplikacemi, tzv. SQLite klienty, je jediný databázový soubor, uložený na disku [11]. Tuto infrastrukturu popisuje Obrázek 3.3.



Obrázek 3.2: Tradiční RDBMS klient/server architektura. (Převzato z: https://www.oreilly.com/library/view/using-sqlite/9781449394592/ch01s01.html)



Obrázek 3.3: Architektura SQLite s absencí serveru. (Převzato z: https://www.oreilly.com/library/view/using-sqlite/9781449394592/ch01s01.html)

Absence počáteční konfigurace

Díky absenci serveru a přímému nalinkování databázového systému v aplikaci je použití SQLite databáze velice přímočaré, jelikož kompletně odpadá nutnost počáteční instalace či konfigurace [11].

Multiplatformní

Celý databázový systém se skládá z jediného souboru, jehož formát je multiplatformní. Ten se může nacházet kdekoliv v adresářové struktuře uložiště. Databázový soubor vytvořený na jednom zařízení může být pro použití na druhém zařízení zkopírován, a to i za skutečnosti, že druhé zařízení je odlišné architektury [27].

Soběstačnost

Databáze SQLite je zapouzdřena do jediného souboru, ten obsahuje jak schéma databáze, indexy, tak data uložená ve všech tabulkách databáze. Tato knihovna se přímo integruje do hostitelské aplikace pomocí jednoduchého rozhraní [11].

Kompaktnost

SQLite knihovna je vytvořena s nízkými paměťovými nároky v potaz. Celá knihovna se všemi vylepšeními zabírá méně než 700KiB, což je méně v porovnání s většinou ostatních databázových systémů. Je tedy velmi vhodná pro použití u vestavěných systémů s malými operačními nároky [11], např. mobilní telefony, televizory, herní konzole apod [26].

3.3.2 Použití

Databáze SQLite je díky svým vlastnostem hojně využívaná napříč různě zaměřenými softwarovými produkty. Používá se např. u desktopových aplikací, k ukládání aplikačních souborů (např. pro Adobe Photoshop Lightroom).

Dále se používá u mobilních zařízení (smartphonů, tabletů atd.) a jejich operačních systémech. Společnosti Apple i Google pro své operační systémy iOS a Android využívají databázi SQLite k perzistentnímu ukládání systémových a aplikačních dat.

SQLite se nasazuje i do komplexních výpočetních systémů, jako je např. letový software letounů společnosti Airbus a mnoho dalších [28].

3.3.3 Kdy zvolit alternativní databázový systém

I přes velkou flexibilitu SQLite se najdou scénáře, ve kterých je použití tohoto databázového systému nepřiměřené svému návrhu a původnímu záměru. I když by použití v těchto oblastech bylo z hlediska implementace možné, nemusí být pro dané řešení nejoptimálnější. Při výskytu následujících situací je vhodnější, zvolit tradičnější relační systém řízení báze dat, implementující architekturu klient-server [11].

Současný zápis mnoha procesy

SQLite podporuje pro každý databázový soubor zápis pouze jedním procesem v konkrétní okamžik. Ve většině případů trvá zapisovací transakce velmi krátkou dobu, obvykle v řádech milisekund, a tak je pro vícero zapisujících procesů možné přistupovat k souboru střídavě.

Pokud nastává situace, kdy několik vláken nebo procesů potřebuje provést zápis do databáze ve stejný okamžik a neřadí se do fronty, pak je opět lepší zvolit alternativní databázový systém, který tuto možnost implicitně podporuje. Tím je opět již zmíněný databázový systém typu klient-server, který je schopen obsloužit více zapisovacích procesů, než by byl SQLite kdy schopen obsloužit [26].

Velké data sety

Maximální velikost SQLite databáze je omezena na 281 terabytů. Dalším důvodem k využití alternativního klient – server databázového systému je fakt, že veškerý obsah databáze je u SQLite uložen v jediném souboru, což by mohlo v případě velké databáze způsobovat problémy pro souborový systém operačního systému. U klient – server databázových systému je obsah rozdělen mezi několik disků, a tak velikost nehraje roli [26].

Omezení přístupu

Databáze SQLite neobsahuje žádné autentizační ani autorizační mechanismy. V případě, že chceme některým uživatelům omezit zápis do databázového souboru, popřípadě znemožnit modifikaci databáze jako takové, musíme omezit přístupová práva k tomuto souboru operačním systémem. Tím přístup omezíme pouze na tři varianty: právo pro čtení a zápis, právo pouze číst a žádné oprávnění. Právo zápisu umožňuje uživateli vyjma modifikace dat také modifikaci struktury databáze. Použití databáze SQLite je nevhodné pro uchování citlivých dat, jelikož lze nedostatečné oprávnění triviálně obejít, máme-li přístup k databázovému souboru [26].

3.3.4 Napojení k SQLite

Každý programovací jazyk implementuje vlastní rozhraní pro komunikaci s SQLite databází a databázemi obecně. U některých jazyků, jako je například Python nebo PHP, je toto databázové rozhraní s SQLite zahrnuto v oficiální distribuci [11]. Tudíž není potřeba instalace dalších rozšíření, abychom docílili komunikace s databází SQLite. Zaměříme se konkrétně na jazyk PHP a jeho databázové rozhraní PDO.

PDO

PHP Data Objects (PDO) [24] je rozšíření pro jazyk PHP, které poskytuje konsistentní rozhraní pro komunikaci s databází. PDO poskytuje abstraktní vrstvu pro přístup k datům a zajišťuje jednotné rozhraní pro různé typy databází. Pro přístup k databázovému serveru je nejdříve nutné specifikovat konkrétní databázový ovladač (driver), specifický pro daný typ databáze. Pro databázi SQLite volíme řadič `PDO_SQLITE`, který podporuje SQLite verze 2 a 3.

Připojení k databázi představuje instance třídy PDO, jež je navrácena skriptu po úspěšném navázání spojení s databází a je aktivní po dobu existence tohoto objektu.

Kapitola 4

Návrh aplikace

Aplikace je koncipovaná jako webová aplikace umístěná na serveru. Jejím úkolem je umožnit úzce definované skupině uživatelů interaktivní prohlížení databázových souborů SQLite s daty konkrétní výrobní úlohy. Databázové soubory typu SQLite jsou nahrávány uživateli webové aplikace a ukládány na uložišti webového serveru, čímž je umožněno jejich sdílení mezi ostatními uživateli a vzájemná kolaborace při práci s těmito soubory.

Od ostatních aplikací podobného druhu se návrh aplikace liší převážně tím, že veškerá činnost je prováděna na webovém serveru, na kterém dochází ke zpracování databázových souborů a uživatel vytváří dynamické spojení s cílovou databází výrobní úlohy.

Nezbytnou funkcionalitou aplikace je procházení dat databázových tabulek za pomocí typizovaných pohledů, které poskytují přehlednou prezentaci dat, založených na tabulkách suchých údajů.

V této kapitole si vytyčíme, co by měla aplikace poskytovat za funkcionalitu, definujeme si podobu uživatelského rozhraní aplikace. Dále si popíšeme rozvržení jednotlivých pohledů, deklarujeme mechanismus přechodů mezi těmito typizovanými pohledy a navrhneme proces autentizace a registrace uživatelů.

4.1 Požadavky na aplikaci

Základní požadavky byly sestaveny na základě konzultací s cílovou skupinou, pro kterou je tato aplikace realizována. Mezi základní požadavky pro práci s aplikací patří:

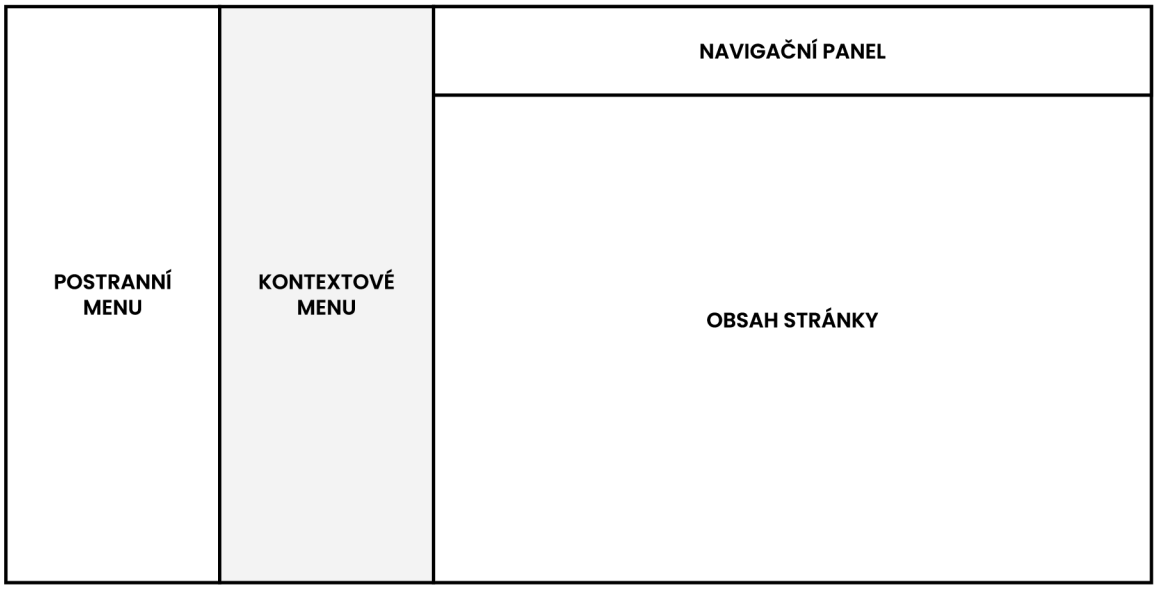
- Možnost nahrávání databázových souborů výrobní úlohy typu SQLite a jejich ukládání na lokálním uložišti webové aplikace.
- Sdílení těchto nahraných souborů mezi uživatele aplikace a možnost jejich stahování.
- Připojení a čtení databázového souboru poskytnuté pro více než jednoho uživatele současně.
- Zobrazení dat aktuálně připojeného databázového souboru pomocí databázových pohledů.
- Průchod mezi pohledy, při výskytu jiného objektu výrobní úlohy v aktuálním pohledu (tzn. možnost přechodu na konkrétní pohled vytvořený pro tento objekt).
- Substituce dlouhých unikátních identifikátorů UUID za kratší a přehlednější reprezentaci a možnost dvojího zobrazení (přepínání mezi těmito tvary).

- Vyhledávání objektů výrobní úlohy na základě specifikace alespoň části jejich identifikátorů a filtrování těchto výsledků na základě typu výrobního objektu.
- Filtrování a seskupování záznamů objektu výrobní úlohy na základě hodnot atributů.

4.2 Návrh uživatelského rozhraní

Uživatelské rozhraní je implementováno pro vybranou skupinu vysoce kvalifikovaných uživatelů, jenž ovládají nezbytnou znalost problematiky pro používání aplikace. Není tudíž za cíl vytvořit uživatelské rozhraní, jemuž by jednoduše rozuměl každý uživatel. Tomuto rozhodnutí odpovídá rozložení uživatelského rozhraní, koncepce jednotlivých pohledů i použití odborné terminologie bez jakéhokoliv dovysvětlení uživateli. Zároveň se snažíme vytvořit rozhraní, které je z hlediska použitelnosti v dané problematice funkční, přehledné, intuitivní a co nejvíce dynamické.

Základní rozložení uživatelského rozhraní aplikace je následující (viz obrázek 4.1). Rozhraní se dělí na čtyři hlavní části: **postranní menu**, **kontextové menu**, **navigační panel** a **obsah stránky**. Tato podoba rozložení byla zvolena proto, neboť nám umožní některé části fixně (nehybně) umístit na webovou stránku a zároveň maximalizaci plochy pro zobrazení **obsahu stránky** (pohledu).



Obrázek 4.1: Základní rozložení uživatelského rozhraní aplikace rozdělené do 4 hlavních částí. Proporce jednotlivých částí nejsou korektně znázorněny, z důvodu zajištění vyšší čitelnosti čtenářem.

Popišme si detailněji funkci a obsah jednotlivých dílčích částí uživatelského rozhraní:

4.2.1 Postranní menu

Postranní menu představuje trvale zobrazenou a neměnnou sekci uživatelského rozhraní, jenž obsahuje hlavní navigaci webové aplikace a sekci pro správu profilu uživatele, uži-

vatelských účtů a další. Stisknutím vybraného tlačítka hlavní navigace dochází ke změně kontextu (obsahu) kontextového menu.

Tlačítka hlavní navigace:

- **Pohledy** – tlačítko změní obsah kontextového menu, které zobrazí definovaný seznam všech dostupných pohledů k prohlížení připojeného databázového souboru výrobní úlohy.
- **Databázové soubory** – tlačítko změní obsah kontextového menu, které zobrazí seznam nahraných databázových souborů a vstupní formulář pro jejich nahrávání.

Další podsekcí umístěnou v postranním menu je podsekce pro správu uživatelského účtu. Ta vyjma zobrazení údajů aktuálně přihlášeného uživatele (celé jméno a emailová adresa) obsahuje tři tlačítka:

- **Profil** – tlačítko směřuje na stránku s konfigurací nastavení spjatými s uživatelským účtem a přihlašováním. Na této stránce je uživateli umožněna změna profilových informací (jméno, email), změna stávajícího hesla, možnost zapnutí / vypnutí dvoufaktorového ověření, zobrazení seznamu zařízení, pomocí kterých byl uživatel k aplikaci přihlášen a možnost odstranění účtu.
- **Uživatelské účty** – tlačítko směřuje na stránku sloužící k zaslání nové a zobrazení seznamu odeslaných pozvánek k registraci do aplikace.
- **Odhlásit se** – tlačítko odhlásí aktuálně přihlášeného uživatele a přesměruje ho na stránku přihlášení.

4.2.2 Kontextové menu

Kontextové menu opět představuje fixní sekci rozhraní, avšak mění svůj obsah na základě vyvolání akce v hlavní navigaci postranního menu. Jeho obsah je tvořen buď (obě varianty znázorněny na obrázku 4.2):

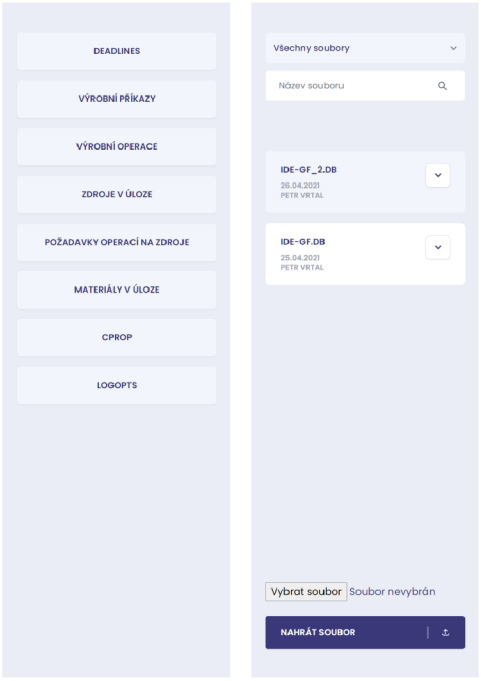
(a) Seznamem všech pohledů, které jsou reprezentovány tlačítky s odkazy na příslušné pohledy (stránky),

(b) Seznamem všech nahraných databázových souborů uživateli aplikace s polem pro vyhledávání souboru a formulářem pro nahrání nového souboru.

4.2.3 Navigační panel

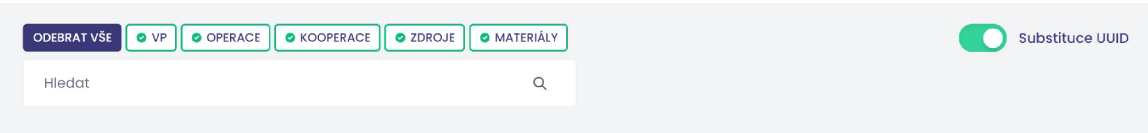
Hlavní dílčí komponentu navigačního panelu představuje globální vyhledávací pole (vlevo na obrázku 4.3), jež vyhledává objekty výrobní úlohy pomocí předloženého identifikátoru objektu (nebo alespoň jeho části) ve vstupním poli. Výsledky vyhledávání jsou ve formě hypertextových odkazů na nalezené objekty výrobní úlohy, vedoucí na příslušný pohled. Vyhledávání objektů probíhá výlučně v doméně aktuálně připojeného databázového souboru.

Uživatel má možnost filtrovat požadované kategorie výsledků pomocí přepínačů nad vyhledávacím polem (tlačítka: ODEBRAT VŠE, VP, OPERACE, KOOPERACE, ZDROJE, MATERIÁLY).



Obrázek 4.2: Obě varianty obsahu kontextového menu. Vlevo seznam všech pohledů zobrazení po stisknutí tlačítka „Pohledy“ v navigaci postranního menu. Vpravo seznam všech databázových souborů nahraných do aplikace spolu s možností vyhledávání a nahrávání databázových souborů.

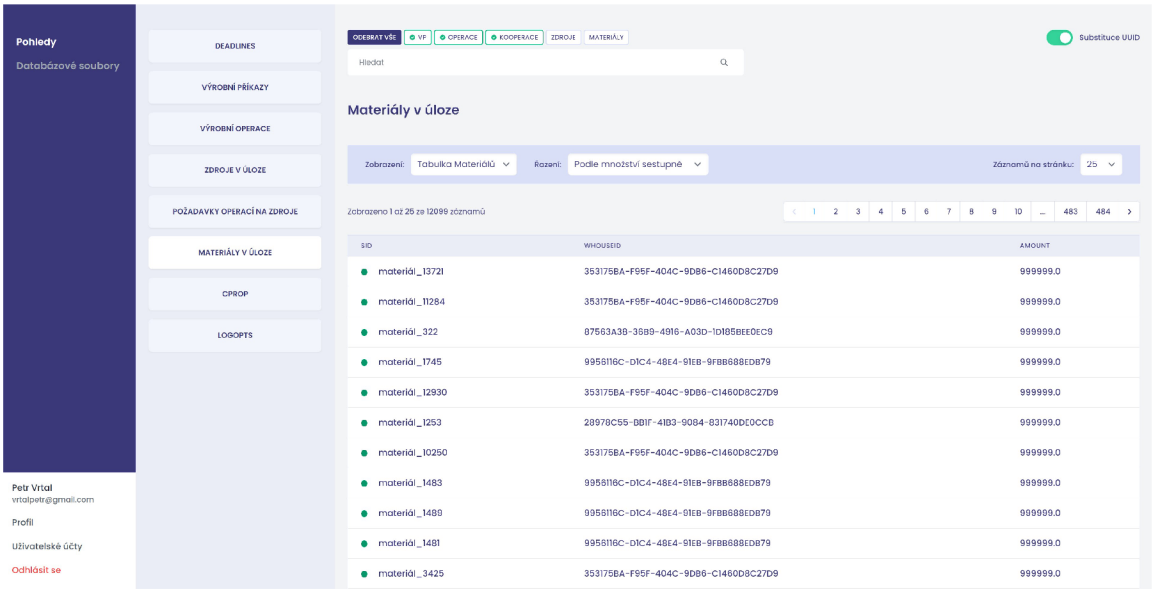
Další dílčí komponentou této sekce je přepínač substituce UUID (vpravo na obrázku 4.3). Pokud se vyskytuje v poloze „zapnuto“, pak dochází na všech místech pohledu k substituci původních UUID identifikátorů objektů výrobní úlohy za jejich kratší a přehlednější reprezentaci (proces substituce detailně popsán v nadcházející sekci 5.3.



Obrázek 4.3: Finální podoba navigačního panelu. Vlevo vyhledávací pole spolu s přepínači filtrů. Vpravo přepínač substituce.

4.2.4 Obsah stránky

Sekce obsah stránky, jak již její název napovídá, zobrazuje obsah aktuálně navštívené stránky (pohledu). Poskytuje tzv. slot pro vykreslení obsahu pohledu v aplikační šabloně.



Obrázek 4.4: Finální podoba uživatelského rozhraní. Zde konkrétně zobrazen pohled materiálů v úloze, substituce je zapnuta. Můžeme vidět jasné rozdělení na čtyři výše popsané sekce: postranní menu s hlavní navigací a menu pro správu profilu, dále kontextové menu, aktuálně zobrazující seznam pohledů, v pravé části stránky navigační panel s vyhledávacím polem a přepínačem substituce a posledně pod ním obsah stránky – v tomto případě pohledu pro zobrazení všech materiálů v úloze.

4.3 Databázové pohledy a přechody mezi nimi

Databázové pohledy představují typizovaný způsob reprezentace dat jednotlivých databázových tabulek výrobní úlohy. Každý pohled je dostupný na své vlastní stránce na konkrétní adrese webové aplikace.

4.3.1 Skupiny databázových pohledů

Pro každý model výrobní úlohy je zapotřebí vytvořit pohled, který by na základě klíčových vlastností tohoto modelu a vztahů s ostatními objekty výrobní úlohy vhodně prezentoval uložená data. Obecně lze pohledy klasifikovat do dvou skupin podle toho, jaký je jejich zobrazovací účel.

Pohledy tvořící seznam základních objektů výrobní úlohy

Podobně, jak je tomu u tabulkové reprezentace, zobrazují tyto pohledy celkové obsahy příslušných databázových tabulek. Mohou to být také pohledy, které tyto seznamy třídí na základě klíčové vlastnosti objektu (např. deadliny, které jsou klíčové pro výrobní příkazy a optimalizace výrobních úloh celkově). Základními objekty úlohy jsou výrobní příkazy, operace (plné operace, fázové operace, kooperace), zdroje v úloze a materiály v úloze. Dále jsou to pohledy, které zobrazují kontingenční tabulku, tedy vztah dvou základních objektů výrobní úlohy, jako např. pohled **Požadavky operací na zdroje**. Skupina těchto pohledů slouží uživateli jako výchozí bod pro čtení databázového souboru konkrétní výrobní úlohy. Seznam všech pohledů, patřící do této skupiny reprezentuje 4.1.

Název	Absolutní URL	Zdrojový soubor pohledu
Deadliny výrobních příkazů	`/view/deadlines`	`vp-deadlines-insight.blade.php`
Výrobní příkazy	`/view/vps`	`vps-insight.blade.php`
Výrobní operace	`/view/jobs`	`jobs.blade.php`
Zdroje v úloze	`/view/resources`	`resources-insight.blade.php`
Požadavky operací na zdroje	`/view/job-resources`	`job-resources-insight.blade.php`
Materiály v úloze	`/view/materials`	`materials-insight.blade.php`
cProp	`/view/cprops`	`cprops-insight.blade.php`
LogOpts	`/view/log-opts`	`log-opts.blade.php`

Tabulka 4.1: Seznam pohledů sloužící jako výchozí bod pro prohlížení databázového souboru. První sloupec udává název pohledu, druhým je jeho relativní URL a třetím je název zdrojového souboru pohledu, uložený ve složce `/resources/views/livewire/pages/insights`.

Pohledy zobrazující data konkrétního objektu výrobní úlohy

Do druhé skupiny pohledů patří pohledy, zobrazující konkrétní data záznamu objektu výrobní úlohy, uloženého v korespondující databázové tabulce. Spolu s atributy záznamu zobrazují také příslušná data objektů, jenž mají s původním objektem nějakou formu relačního vztahu (např. plná operace vlastní množinu fázových operací nebo požadavky operace na konkrétní zdroje). Všechny pohledy této skupiny jsou vyčteny v tabulce 4.2.

Název	Absolutní URL	Zdrojový soubor pohledu
Konkrétním výrobní příkaz	`/view/vp/{UUID}`	`vp-insight.blade.php`
Konkrétní operace	`/view/job/{UUID}`	`job-insight.blade.php`
Konkrétní zdroj v úloze	`/view/resource/{UUID}`	`resource-insight.blade.php`
Konkrétní materiál v úloze	`/view/material/{UUID}`	`material-insight.blade.php`

Tabulka 4.2: Seznam pohledů sloužící pro zobrazení dat konkrétního objektu výrobní úlohy se zadaným identifikátorem UUID. První sloupec tabulky popisuje název pohledu. Druhým je absolutní URL, vedoucí na příslušný pohled. Stojí za zmínku popsat funkci posledního segment URI **{UUID}**, jenž se využívá jako parametr předávaný komponentě pohledu, přenášející konkrétní identifikátor objektu výrobní úlohy. Ten se využívá pro účely dotazování dat k vyhledání příslušného databázového záznamu modelu.

4.3.2 Přechody mezi pohledy

Přechody mezi pohledy je funkcionalita, která řešení této aplikace odlišuje od ostatních konvenčních aplikací na procházení databází SQLite.

Přechod pomocí historie prohlížeče

Jelikož je každý pohled navržen jako celostránková komponenta typu Livewire (třída dědící od bázové třídy `Livewire\Component`), dostupná z prohlížeče na konkrétní adrese, je tedy

navigace mezi pohledy jednoduchá. Můžeme využít navigace a historie webového prohlížeče (tlačítko zpět a vpřed) a mezi pohledy přecházet pomocí ní.

Každý pohled ukládá své nastavení filtrů a jiných parametrů (např. aktuální číslo stránky stránkování, počet záznamů na stránku apod.) v podobě query stringu, jenž vkládá data přímo do URL stránky. Po znovunačtení stránky je tedy pohled zobrazen ve stejné podobě pro konkrétní hodnotu parametrů query stringu. Uvedený příklad popisuje implementaci query stringu pro uložení parametrů pohledu výrobní operace v úloze.

`opt3-browser.cz/view/jobs?filterByType=full-ops&paginationCount=50&page=2`

V pohledu zobrazující všechny výrobní úlohy se můžeme setkat s následujícími parametry, uložené v query stringu URL pohledu:

- `filterByType` – parametr definující typ zobrazení pohledu
- `paginationCount` – parametr určující maximální počet záznamů pro jednu zobrazenou stránku stránkování
- `page` – parametr udávající aktuální zobrazenou stránku stránkování

Přechod pomocí hypertextových odkazů v pohledech

Navigace jen za použití historie prohlížeče by byla značně omezující a nedostatečná. Proto jako další možnost umožňuje aplikace přechody mezi pohledy v podobě hypertextových odkazů na předem určených místech. Jsou to místa, kdy se v aktuálním pohledu vyskytuje identifikátor patřící jinému objektu výrobní úlohy. Namísto zobrazení identifikátoru objektu v podobě čistého textu jej v pohledu vytiskneme jako hypertextové odkazy, vedoucí na příslušný pohled, jehož cestu jsme schopni odvodit na základě typu objektu výrobní úlohy (korespondující URL viz druhý sloupec v tabulce 4.1 a 4.2).

Přechod pomocí nálezu objektu v globálním vyhledávacím poli

Za předpokladu, kdy uživatel potřebuje zjistit informace (data) o konkrétním objektu výrobní úlohy a má k dispozici jeho unikátní identifikátor, může uživatel využít globálního vyhledávání k nalezení tohoto objektu. Vyhledávací pole v uživatelském rozhraní aplikace navrací výsledky v podobě hypertextového odkazu, směřujícího na pohled hledaného objektu.

4.3.3 Návrh jednotlivých pohledů

Jak je již z kapitoly 2.4, popisující hierarchii mezi jednotlivými základními objekty výrobní úlohy patrné, potřebujeme každý pohled navrhnout tak, aby byl schopen adekvátně zobrazit tyto vzájemné hierarchické vztahy a objekty v nich figurující. Každý jednotlivý pohled má tedy odlišný zobrazovací účel. Některé pohledy podporují více typů zobrazení. V následujícím textu této podkapitoly si popíšeme ty nejzajímavější a nejdůležitější pohledy, které se v aplikaci nachází.

Pohled na termíny výrobních příkazů v úloze

Úkolem tohoto typizovaného pohledu je zobrazit všechny termíny (deadliny) výrobních příkazů, vyskytující se ve výrobní úloze. Dodržení těchto termínu je pro výrobní podnik

esenciální. Nastane-li situace, kdy je termín výrobního příkazu naplánován do minulosti, dochází ke zpoždění ve výrobě a vstupní výrobní rozvrh musí být pozměněn. Proto je pro uživatele nutné aplikací rozlišovat termíny vzhledem k současnému datu na:

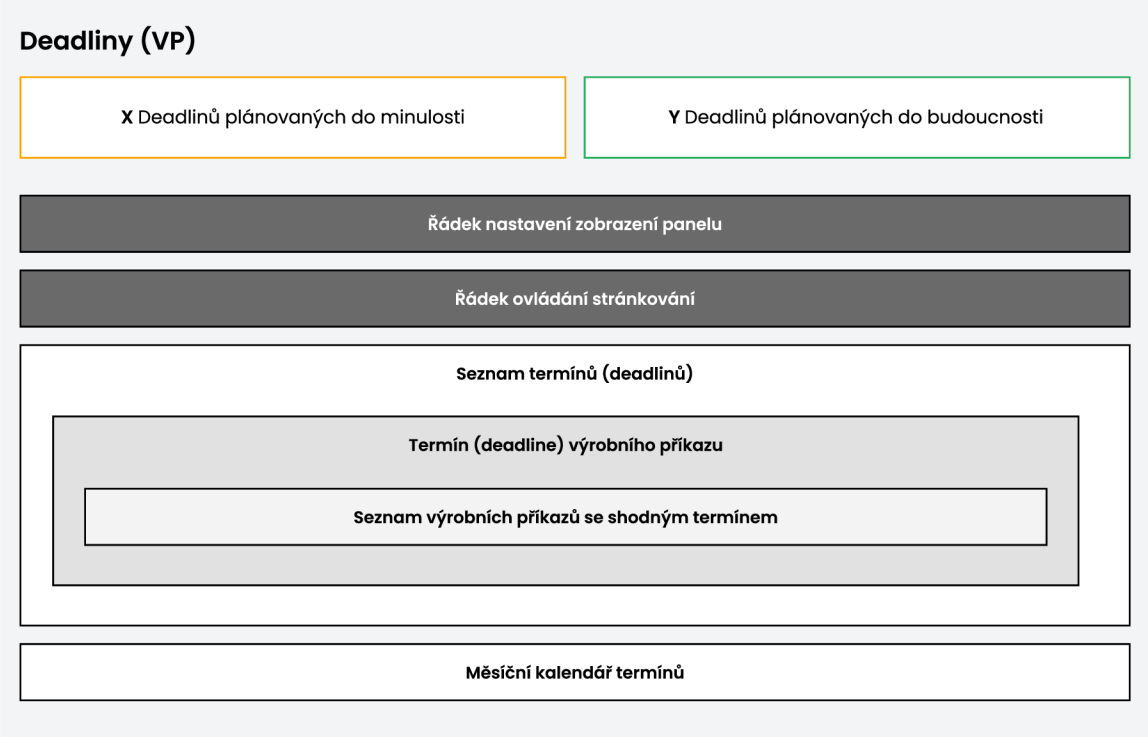
1. Termíny výrobních příkazů, naplánovaných **do budoucnosti**
2. Termíny výrobních operací, jejichž termín je **již v minulosti**

Celkové počty těchto dvou skupin jsou přehledně zobrazeny **v číselnících** v horní části pohledu. Na ně následně navazuje seznam všech výrobních příkazů ve výrobní úloze, tvořen skupinami na základě konkrétního termínu. Termíny, respektive jejich skupiny, reprezentují časovou osu výrobní úlohy.

Pohled skýtá **možnosti filtrování termínů výrobních příkazů**. Především je to možnost přepnutí zobrazení termínů na základě rozdělení popsaného výše čili na termíny v minulosti, naplánované do budoucnosti a jako výchozí, zobrazení obou těchto kategorií současně. Dále je pro výsledné termíny vytvořena možnost přepínání **řazení podle data** vzestupně nebo sestupně.

V neposlední řadě je uživateli s využitím knihovny Fullcalendar.js[1] zobrazen **měsíční kalendář** s počty výrobních příkazů na jednotlivé termíny.

Výsledný návrh pohledu můžeme vidět na obrázku 4.7.



Obrázek 4.5: Návrh pohledu zobrazující termíny výrobních příkazů ve výrobní úloze.

Pohled na výrobní příkazy v úloze

Posloupnost výrobních příkazů tvoří celek výrobní úlohy. Namísto potřeby zaměřit se na jejich termíny, jak tomu bylo u pohledu zobrazující termíny výrobních příkazů, je účelem

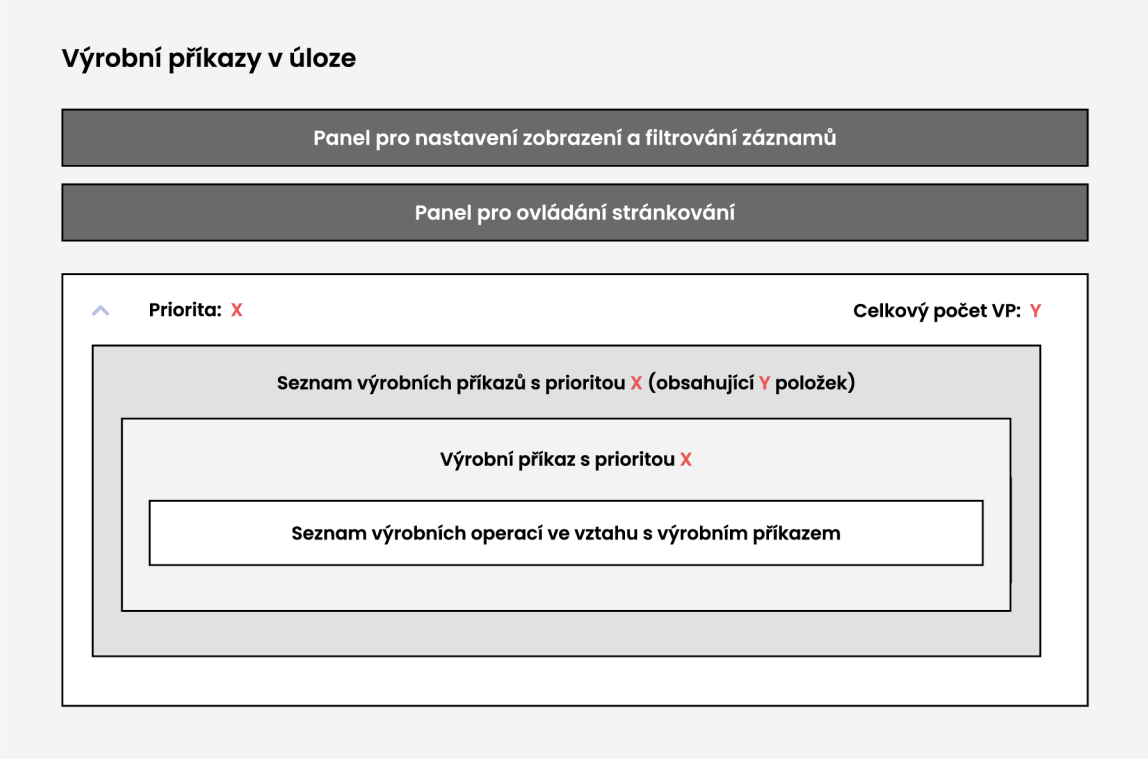
[1] Knihovna Fullcalendar.js pro tvorbu kalendářů - https://fullcalendar.io/

tohoto pohledu zobrazit především výrobní operace, které ve výrobních příkazech figurují. Dále je to poskytnutí možnosti uživateli procházet seznam všech výrobních příkazů na základě jejich atributu **priority**.

Pohled tedy nabízí dvě možnosti zobrazení:

- **Zobrazení Tabulky VP** – zobrazuje seznam záznamů výrobních příkazů příslušné databázové tabulky, bez jejich řazení dle priority. Pro každou položku výrobního příkazu v seznamu načítá pomocí databázové tabulky *Ownership* příslušné výrobní operace a její dílčí fázové operace, včetně jejich dat. Dále uživateli umožníme položky seznamu výrobních operací filtrovat na výrobní příkazy obsahující kooperace, výrobní operace nebo obojí současně.
- **VP seřazeny do skupin dle jejich priority** – uživateli je zobrazen seznam všech možných priorit (řazené vzestupně nebo sestupně na základě preference uživatele), jež mohou výrobní příkazy nabývat. Do jednotlivých položek tohoto seznamu priorit se řadí výrobní příkazy se shodnou prioritou. Uživatel může skupiny priorit řadit dle potřeby **vzestupně** či **sestupně** na základě priority nebo skrýt výrobní příkazy, jenž jsou neprioritní (priorita = 0).

Výsledný návrh pohledu *výrobních operací*, se zobrazením skupin *dle priority*, můžeme vidět na obrázku 4.6.



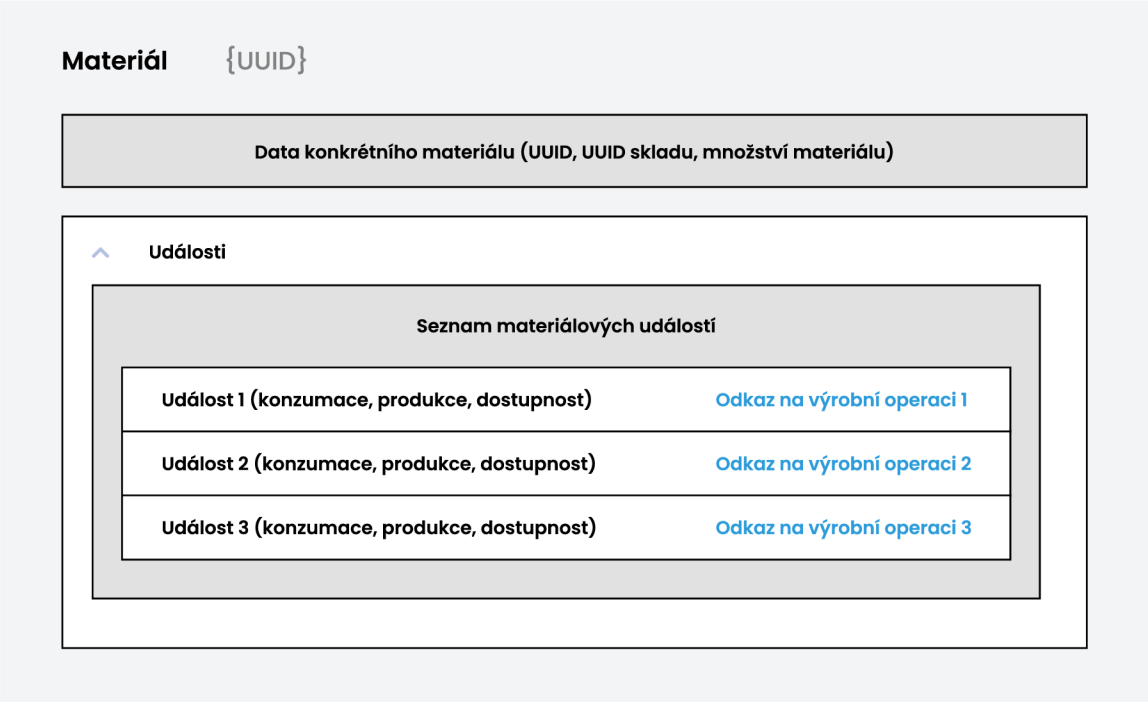
Obrázek 4.6: Návrh pohledu zobrazující seznam výrobních příkazů, seřazený do karet podle své priority.

Pohled na konkrétní materiál v úloze

Úkolem tohoto pohledu je přehledně zobrazit data pro konkrétní materiál. Na rozdíl od prvních dvou příkladů návrhu pohledů spadá tento pohled do kategorie pohledů, zobrazující data jednoho konkrétního objektu výrobní úlohy. V tomto případě jde o objekt typu *materiál*.

Nejdůležitější informací, jenž si materiál uchovává (konkrétně jde o data z databázové tabulky *MatEvents*), jsou **události, které v čase modifikují stav tohoto materiálu**, např. aktuální stav jeho skladových zásob při **produkci** / **konzumaci**. Proto tomuto pohledu přidružíme tabulku, která všechny události, spojené s tímto konkrétním materiálem zobrazí. Na všechny výrobní úlohy, vystupující v této tabulce jako konzument / producent, umožníme uživateli přechod.

Výsledný návrh pohledu, zobrazující konkrétní materiál, můžeme vidět na obrázku 4.7.



Obrázek 4.7: Návrh pohledu zobrazující data konkrétního materiálu a spolu s nimi i tabulku materiálových událostí.

4.4 Autentizace uživatelů

Jelikož je aplikace koncipovaná pro konkrétní skupinu uživatelů, je zapotřebí vytvořit mechanismus, kterým budeme schopni tuto uživatelskou bázi regulovat. Registrace nových uživatelů by neměla být volně přístupná kterémukoliv návštěvníkovi naší webové aplikace. Zároveň je zapotřebí uživatelům, kteří dostali povolení od administrátora tuto registraci umožnit. Možností řešení tohoto problému existuje spousta, avšak hlavním cílem všech způsobů řešení je umožnit uživateli tvorbu účtu bez nutnosti přímého zásahu do aplikační

databáze. Pro tuto konkrétní aplikaci byly navrženy dvě metody založení nového uživatelského účtu do systému.

Prvním způsobem je vytvořit uživatelský účet pomocí speciálního příkazu příkazové řádky aplikace z prostředí webového serveru. Tento postup je vhodný pro vytvoření prvního (administrátorského) účtu po nasazení aplikace na server, jelikož k tomu nevyžaduje samotné webové aplikační rozhraní. Pozvánky vytvoříme příkazem:

```
php artisan ivitation:send [adresa1 adresa2 ...]
```

např. v tomto konkrétním tvaru:

```
php artisan ivitation:send xvrtal01@fit.vutbr.cz xnovak00@fit.vutbr.cz
```

Druhou metodou je využití stránky se správou uživatelských účtů ve webovém aplikačním rozhraní. Tato metoda ovšem předpokládá, že je uživatel navštěvující tuto stránku do aplikace přihlášen (autentizován). Z tohoto důvodu nelze tuto metodu využít pro vytvoření prvotního uživatelského účtu.

K vytvoření uživatelského účtu nedochází napřímo, pomocí zadaných údajů (e-mail, heslo, jméno atd.) - tzn. nezadáváme veškeré údaje najednou. Administrátorem aplikace nebo uživatelem je pomocí jedné z výše specifikovaných metod vytvořena **pozvánka k registraci do webové aplikace**, která je následně **zaslána pomocí služby SMTP ve formě mailu** na zadané e-mailové adresy. V pozvánce se nachází odkaz, po jehož kliknutí je adresát přesměrován na stránku s registračním formulářem, pomocí kterého je po zadání nezbytných osobních údajů zaregistrován.

Tento odkaz obsahuje speciální **token**, unikátně vygenerovaný pro každou pozvánku, jehož platnost je časově omezená. Stránka s registračním formulářem je chráněna před vstupem uživatele, který nedisponuje tímto speciálním tokenem a neumožní mu tak na tuto stránku přistoupit.

Kapitola 5

Implementace aplikace a nasazení

Kapitola implementace popisuje zejména některé zajímavé a důležité části implementace webové aplikace a jejich funkci, na základě specifikace návrhu v kapitole 4. K implementaci webové aplikace byly použity technologie popsané v kapitole 3.

Základem implementace aplikace se stal PHP webový framework Laravel, jenž poskytuje robustní základ pro tvorbu webových aplikací. K němu bylo přidáno několik frameworků, které rozšiřují funkčnost nebo citelně ulehčují vývoj aplikace, náhradou (nadstavbou) základních webových technologií jako jsou Javascript, CSS nebo AJAX. Jedná se o frameworky Alpine.js[1], Tailwind CSS[2] a Livewire v tomto pořadí.

Dále bylo potřeba navrhnout a implementovat databázi, uchovávající data samotné webové aplikace. Pro tu byl zvolen databázový systém SQLite, jenž poskytuje rychlou a kompaktní alternativu k tradičním RDBMS relačním databázím, jako např. MySQL. Schéma této aplikační databáze je podrobně popsáno v kapitole 5.2.

5.1 Struktura řešení aplikace

Jak již bylo zmíněno v úvodu této kapitoly, k vývoji webové aplikace byl použit PHP aplikační framework Laravel. Ten implementuje **architektonický vzor MVC** (popsaný v kapitole 3.1.3), jenž se do jisté míry v implementované aplikaci využívá. Převažuje však koncept **rozdělení aplikace do několika dílčích komponent** typu Livewire.

5.1.1 Směrovač

Výchozím bodem aplikace je **proces směrování** (angl. routing). Směrovač vyhodnotí příchozí HTML žádost a na jejím základě je žádost předána korespondujícímu řadiči nebo komponentě, jež jsou definovány v souboru `routes/web.php`. V aplikaci je každý typizovaný pohled na databázi, jenž představuje způsob procházení dat výrobní úlohy, implementován jako samostatná celostránková Livewire komponenta, na kterou směrovač odkazuje.

5.1.2 Komponenty

Aplikace sestává z několika jednotlivých komponent typu Livewire (postranní menu, navigační panel, celostránkový typizovaný pohled apod.). Funkce komponent se různí. Některé slouží pouze k zestručnění a zpřehlednění kódu, jiné mohou plnit důležitou funkcionalitu

[1] Alpine.js - javascriptový framework - https://github.com/alpinejs/alpine
[2] Tailwind CSS - CSS framework s tzv. utility - first principem - https://tailwindcss.com/

aplikace (např. funkcionalita vyhledávání objektů výrobní úlohy). Skrz komponenty Livewire jsme schopni se serverem komunikovat **asynchronně** pomocí AJAX žádostí, a tudíž vynechat potřebu vytvářet řadiče pro příslušné celostránkové pohledy a další dílčí komponenty.

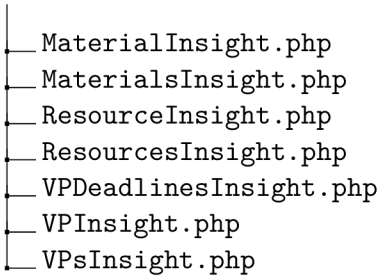
Celkem je vytvořeno několik komponent. Část z nich se využívají jako dílčí komponenty, které se vnořují ke komponentám celostránkovým. Komponenty spolu vzájemně komunikují a předávají si nezbytná data. Celostránkové komponenty, jak již jejich název napovídá, představují obsah celé stránky (např. pohledu), který je navrácen uživateli.

U všech dílčích komponent je vytvořena následující hierarchie na základě atomického principu vizuálního designu (viz článek [25]):

- **Atomy** – představují nejzákladnější jednotky (stavební bloky) uživatelského rozhraní. Do této kategorie řadíme například tlačítka, nadpisy aj. Vytváření komponent, představující atomy nebylo během vývoje aplikace s využitím frameworku *Tailwind CSS* zapotřebí, nýbrž s jeho pomocí snadno definujeme vzhled a formu atomů přímo na molekulární úrovni.
- **Molekuly** – vzájemně kombinují atomární komponenty do většího celku, kdy tato unikátní kombinace dává celku zcela nový význam. (např. přepínač substituce jako kombinace atomů přepínače a jeho popisku, nebo vyhledávací pole, kombinující tlačítko samotný vstup)
- **Organismy** – představují již komplexnější skupinu vzájemně zkombinovaných organizmů, avšak nejedná se ještě o kompletní strukturu stránky, spíše o její část.
- **Stránky** – jakožto finální kompozice jednotek organizmů, reprezentují celkovou podobu stránky zobrazovanou uživateli. Stránka odpovídá celostránkové Livewire komponentě a je složena z několika dílčích komponent.

Komponenty Livewire tvoří v adresářové struktuře projektu následující strukturu:

```
/
└── /app
    └── /Http
        └── /Livewire
            ├── /Molecules
            │   ├── ReturnNavigationButton.php
            │   └── SubstitutionToggle.php
            ├── /Organisms
            │   ├── CreateInvitationForm.php
            │   ├── GlobalSearchBar.php
            │   ├── InvitationsForm.php
            │   └── SideMenu.php
            └── /Pages
                └── /Insights
                    ├── cPropsInsight.php
                    ├── JobInsight.php
                    ├── JobResourcesInsight.php
                    ├── Jobs.php
                    └── LogOpts.php
```



5.1.3 Šablony

Některé prvky stránky si zachovávají stálou podobu a pozici pro každou stránku webové aplikace. Je tedy z hlediska pozdějších úprav výhodnější, moci toto výchozí rozložení spravovat z jediného souboru a šablony Blade umožňující právě to. Například pro všechny stránky webové aplikace definujeme (dle návrhu v sekci 4.2) fixní části uživatelského rozhraní, jako je postranní nebo kontextové menu, které jsou k dispozici pro každou stránku.

Komponenty celostránkového typu využívají předem vytvořenou šablonu, uloženou v souboru `resources/views/layouts/app.blade.php`. Pomocí ní na předem určená místa, do tzv. *slotů*, umisťují svůj vlastní obsah. Vzhled je tím pádem konzistentní pro všechny pohledy, využívající danou šablonu jako své východisko.

5.1.4 Další struktury

Pro některé případy je ovšem využito výchozí architektury MVC a pro ty vytvořeny konkrétní řadiče (kontrolery). Můžeme například zmínit řadič *CustomRegisteredUserController*, který dědí od bázové třídy *RegisteredUserController*. Tento řadič byl vytvořen z důvodu potřeby, rozšířit funkcionalitu původního řadiče o kontrolu tokenů, jenž jsou uděleny pouze uživatelům, jimž byla pomocí SMTP zaslána pozvánka k registraci formou e-mailu.

5.2 Struktura aplikační databáze

K persistentnímu ukládání interních aplikačních dat využívá aplikace opět databáze typu SQLite. Aplikační data jsou tedy oddělena od dat výrobní úlohy, uložených v jednotlivých databázových souborech výrobní úlohy.

Databázová tabulka database_files

Databázová tabulka *database_files* uchovává veškeré potřebné záznamy o uložených databázových souborech, které byli do aplikace nahrány autentizovanými uživateli. Slouží primárně k uchování informace o autorovi daného souboru a místa jeho uložení (cesty k souboru).

Popis sloupců databázové tabulky:

- `id` – primární klíč položky databázového souboru typu unsigned big integer
- `user_id` – cizí klíč do tabulky users, představuje identifikátor uživatele, zodpovědného za nahrání dotyčného souboru, atribut typu unsigned big integer
- `url` – absolutní cesta k databázovému souboru, název souboru je unikátní, typu varchar
- `original_name` – původní název nahrávaného souboru, typu varchar

- `created_at` – časová značka vytvoření databázového souboru (záznamu v tabulce)
- `updated_at` – časová značka poslední úpravy atributů databázového souboru (záznamu v tabulce)

Databázová tabulka registration_invites

Databázová tabulka *registration_invites* jak její název napovídá, obsahuje záznamy o veškerých vytvořených a zaslaných pozvánkách k registraci do aplikace, vystavené z aplikace již zaregistrovaným uživatelem.

Popis sloupců databázové tabulky:

- `id` – primární klíč pozvánky typu unsigned big integer
- `email` – emailová adresa příjemce pozvánky, unikátní v rámci celé tabulky, typu varchar
- `token` – přístupový token sloužící pro umožnění přístupu k registrační routě zabezpečené podpisem, typu varchar
- `created_at` – časová značka vytvoření záznamu v tabulce
- `updated_at` – časová značka poslední úpravy záznamu v tabulce

Databázová tabulka short_ids

Databázová tabulka *short_ids* obsahuje seznam vytvořených substitucí za původní identifikátory objektů výrobní úlohy (operace, výrobní příkazy, zdroje v úloze apod.).

Popis sloupců databázové tabulky:

- `id` – primární klíč položky substituce typu unsigned big integer
- `file_id` – název databázového souboru, z jehož objektu se substituce vytvořila
- `model_id` – původní nesubstituovaný identifikátor objektu typu varchar
- `group` – název skupiny (prefixu) substituce typu varchar
- `short_id` – výsledný tvar substituce UUID
- `created_at` – časová značka vytvoření záznamu v tabulce
- `updated_at` – časová značka poslední úpravy záznamu v tabulce

Databázová tabulka users

Databázová tabulka *users* ukládá informace registrovaných uživatelů aplikace. Obsahuje také přihlašovací údaje uživatelů.

Popis sloupců tabulky:

- `id` – primární klíč uživatele typu unsigned big integer
- `name` – celé jméno uživatele typu varchar

- `email` – emailová adresa uživatele, slouží jako přihlašovací jméno, unikátní v rámci celé tabulky, typu varchar
- `email_verified_at` – časová značka verifikace vytvořeného uživatelského účtu
- `password` – uživatelské heslo uložené v podobě hashe
- `two_factor_secret` – tajný kód sloužící ke dvou-faktorovému ověření
- `two_factor_recovery_codes` – kódy pro zotavení sloužící dvou-faktorovému ověření
- `remember_token` – token pro uložení přihlašovacích údajů uživatele pro příští přihlášení
- `current_database_file_id` – cizí klíč databázového souboru, k němuž je uživatel momentálně připojen
- `created_at` – časová značka vytvoření záznamu v tabulce
- `updated_at` – časová značka poslední úpravy záznamu v tabulce

5.3 Proces substituce UUID

Substituce unikátních identifikátorů výrobních objektů představuje jeden z výchozích požadavků návrhu aplikace. Substituce nahrazuje původní 128bitovou číselnou UUID reprezentaci identifikátorů za kratší, a hlavně přehlednější tvar, jenž uživateli pomůže napovědět, o který konkrétní typ objektu výrobní úlohy se jedná. Funkcionalita substituce jako celku je rozdělena do několika souborů.

Prvním je pomocná třída *ShortIdTranslator* (nacházející se ve složce `app/Helpers`), která obsahuje kolekci metod, odpovědných za vytváření a navracení substitučního tvaru identifikátoru objektu – instanci třídy *ShortId* a jejich ukládání do databázové tabulky *short_ids*. Obsahuje metodu *trans()*, která v případě neexistence substituce konkrétního UUID tuto substituci vytvoří nebo v opačném případě navrací z databáze záznam s již vytvořenou substitucí. Deklarací pomocné funkce *shortId()* v souboru `app/Helpers/helpers.php`, je nám umožněno kdekoliv ve zdrojovém kódu dle potřeby vytvořit a přistupovat k instanci třídy *ShortIdTranslator*.

Druhou část funkcionality substituce reprezentuje PHP trait *Namable* (uložený ve složce `app/Traits`), jenž pomocí klausule `use` dědíme v jednotlivých modelech objektu výrobní úlohy, u nichž chceme proces substituce UUID využít. Ten pomocí výše uvedené pomocné funkce *shortId()* přistupuje k metodám *ShortIdTranslator*. Trait *Namable* ve své inicializační funkci apenduje instanci dané mateřské třídy atribut *short_id*, který představuje textovou hodnotu substituce, navrácenou funkcí *getShortIdAttribute()* . Nový atribut je apendován záznamům navráceným z tabulek výrobních objektů. Tímto způsobem dochází k vytváření substituce jen pro ty záznamy, které uživatel po databázi vyžaduje, tzv. ad hoc.

5.3.1 Tvar substituce

Tvar substituce se dá klasifikovat jako `{nazev_skupiny}_{index}`, přičemž:

- `{nazev_skupiny}` je slovní popis objektu výrobní úlohy nebo skupiny, na které se konkrétní typ objektu výrobní úlohy rozpadá (např. výrobní operace se dále dělí na

plné operace, fázové operace a kooperace). Výchozí tvar názvu skupiny je vypočten v metodě *setSubstitutionGroup()* traitu *Namable* a odpovídá názvu bázové třídy v malých písmenech. Název skupiny je model využívající trait *Namable* schopen pozměnit překrytím metody *setSubstitutionGroup()* ve své třídě. Překrytím zmíněné funkce se využívá např. v modelu `Job.php` nebo `Resource.php`, kdy je jako název skupiny potřeba uvést konkrétní typ instance daného modelu.

  - Pro model **Job** (výrobní operace): „plná_operace“, „fázová_operace“, „kooperace“.
  - Pro model **Resource** (zdroj v úloze): „stroj“, „nástroj“, „personál“.

- `{index}` je celočíselná hodnota, která odpovídá indexu prvku v poli všech záznamů výrobního objektu z korespondující databázové tabulky, nad jehož identifikátorem substituci provádíme. Hodnota a význam indexu může být vyložena podobně, jako čítač, který oproti běžné inkrementace hodnoty o jedna hodnotu vyčítá z pořadí záznamů uložených v tabulkách. Opět, jak tomu bylo i u názvu skupiny substituce, je možné, překrytím metody `setSubstitutionPosition()` traitu *Namable*, definovat vlastní výpočet indexu tvaru substituce. Této možnosti se využívá např. u modelu *Job*, kdy se index substituce definuje zvlášť pro plné a fázové operace (figurují zde dvě pole, jedno s plnými, druhé s fázovými operacemi).

5.3.2 Přepínání zobrazení UUID ve tvaru substituce

Uživatel může volit mezi režimem zobrazení původních UUID anebo jejich substituovanou notací. Zobrazení identifikátorů ovládá pomocí přepínače v navigačním panelu uživatelského rozhraní popsaného v kapitole 4.2.3. Stav tohoto přepínače je uchováván v session (sezení) `substitutionToggle`, tak aby byl stav přepínače při načtení stránky vždy v původním stavu.

V databázových pohledech pak jen přepínáme pomocí direktiv Blade (např. direktivy `@if` a `@else`) mezi dvěma atributy modelu: `$model->sId` a `$model->short_id`.

5.4 Napojení na databázové soubory výrobní úlohy

Databázové soubory výrobní úlohy typu SQLite jsou po nahrání uživatelem uloženy v lokálním souborovém uložišti aplikace, ve složce `storage/app/databases`. Napojení na databázové soubory je prováděno dynamicky pomocí rozhraní `tenant`, definované v konfiguračním souboru `config/database.php`, v poli connections.

Informace o aktuálně připojeném souboru je uložena v databázové tabulce users, ve sloupci `current_file`, jenž představuje cizí klíč na záznam databázového souboru v tabulce `database_files`.

Pro každý modelu objektu výrobní úlohy stanovíme, kterou výchozí konfiguraci databázového spojení si přejeme pro dotazování využívat. To nastavíme v proměnné `connection`, která specifikuje databázové připojení vytvořené, v již uvedeném souboru `database.php`.

Napojení na databázové soubory není při přechodu mezi stránkami (pohledy) zachováváno. Pro každé načtení stránky je třeba vytvořit připojení na aktuálně zvolený databázový soubor. To se provede specifikací hodnoty s klíčem „database“ v asociativním poli nastavení připojení `tenant`, vložením absolutní cesty k připojovanému databázovému souboru výrobní úlohy, uložené v záznamu databázové tabulce `database_files`. Tuto asociaci pro-

vede metoda `changeDatabaseFile()`, jež se nachází v traitu `HasDatabaseFiles`. Po specifikaci cesty a nastavení konfigurace připojení je pro každý dotaz prováděný nad modelem automaticky využíváno specifikované připojení.

5.5 Nasazení na webový server

Aplikace byla nejprve vyvíjena na lokálním prostředí počítače. Pomocí příkazu `php artisan serve`, poskytovaným frameworkem Laravel bylo možné, vytvořit lokální vývojové prostředí s využitím PHP serveru na adrese `http://localhost:8000`, jenž je pro potřeby počátečního vývoje aplikace dostačující.

Posléze byla aplikace pro účely testování a prezentace nasazena na ostrý produkční server, který je veřejně dostupný na adrese https://opt3-browser.cz. Pro nasazení aplikace byl zvolený virtuální server od poskytovatele Váš-hosting s.r.o.[3], jenž nabízí služby v oblasti web hostingu. Konkrétní typ virtuálního serveru byl zvolen jako Mini VPS, s těmito konkrétními parametry, které jsou pro tuto aplikaci dostačující:

- Operační systém: Debian + VPS Centrum (administrační GUI nad serverem)
- Operační paměť: 2 GB RAM
- Výkon procesoru: 3 GHz
- Kapacita SSD disku: 10 GB

Mezi nezbytné prvky, potřebné k nasazení této webové aplikace na produkční server se řadí především možnost přihlásit se na server pomocí protokolu SSH, kvůli nezbytným nastavením a instalaci balíčků.

Pro potřeby snazšího přemístění aplikačních souborů na cílový server byl s využitím služby GitHub vytvořen veřejný repositář *ibt-opt3-browser.git*[4] , obsahující zdrojové soubory aplikace.

Výsledná aplikace, implementovaná ve frameworku Laravel 8 požaduje pro svůj běh splnění minimálně těchto systémových požadavků:

- PHP verze 7.3 a vyšší
- Rozšíření `BCMath` pro PHP
- Rozšíření `Ctype` pro PHP
- Rozšíření `Fileinfo` pro PHP
- Rozšíření `JSON` pro PHP
- Rozšíření `Mbstring` pro PHP
- Rozšíření `OpenSSL` pro PHP
- Rozšíření `PDO` pro PHP

[3] Váš - hosting s.r.o. – poskytovatel hostingových služeb – https://www.vas-hosting.cz/virtualni-servery

[4] Veřejný repositář, obsahující zdrojové soubory aplikace pro instalaci na webovém serveru – https://github.com/Zyrmox/ibt-opt3-browser.git

- Rozšíření `Tokenizer` pro PHP
- Rozšíření `XML` pro PHP

S využitím internetového článku[5] od vybraného poskytovatele webhostingu, podrobně popisující jednotlivé kroky instalace projektu Laravel na server, jsem byl schopný aplikaci nasadit pomocí následujícího sledu příkazů příkazové řádky (po spojení SSH) / kroků, popsaných ve výpisu 5.1.

```
 1. $ apt-get install curl
 2. $ curl -sS https://getcomposer.org/installer | php
 3. $ mv composer.phar /usr/local/bin/composer
 4. $ chmod +x /usr/local/bin/composer
 5. $ cd /www/hosting/opt3-browser.cz/www
 6. $ apt-get install git
 7. $ git clone https://github.com/Zyrmox/ibt-opt3-browser.git
 8. $ cd ibt-opt3-browser
 9. $ composer install
11. $ touch database/database.sqlite
12. $ cp .env.example .env
13. ZMENA PARAMETRU V SOUBORU .env
14. $ chmod -R 755 /www/hosting/opt3-browser.cz/www/ibt-opt3-browser
15. $ php artisan key:generate
16. VYTVORENI SOUBORU .htaccess
17. $ php artisan storage:link
18. $ php artisan migrate
19. $ chown -R www-data.www-data
      /www/hosting/opt3-browser.cz/www/ibt-opt3-browser
20. ZMENA VYCHOZICH NASTAVENI v php.ini souboru
```

Výpis 5.1: Sled příkazů / kroků pro instalaci webové aplikace na virtuálním serveru (VPS) s OS Debian.

V kroku **13.** instalačního postupu (viz výpis 5.1) bylo potřeba změnit obsah konfiguračního souboru `.etc`. Ten může obsahovat citlivá data v podobě hesel a uživatelských tokenů, tudíž se pro repozitář tento soubor ignoruje a je zapotřebí jej vytvořit kopií souboru `.env.example` v předešlém kroku 12. Parametry, které bylo potřeba do souboru `.env` vyplnit, pro nasazení na konkrétní server jsou:

- `APP_URL` – URL aplikace (její doména)
- `MAIL_HOST` – SMTP server (poskytovatel)
- `MAIL_PORT` – port protokolu SMTP (s využitím SSL/TLS je tento port 465 nebo 587)
- `MAIL_USERNAME` – uživatelské jméno e-mailového účtu schránky
- `MAIL_PASSWORD` – heslo e-mailového učtu schránky
- `MAIL_ENCRYPTION` – metoda šifrování „ssl“ nebo „tls“

[5] Nasazení webové aplikace Laravel na server váš-hosting – https://www.vas-hosting.cz/blog-tutorial-jak-nainstalovat-laravel

- `MAIL_FROM_ADDRESS` – e-mailová adresa odesílatele

Z důvodu velikosti nahrávaných souborů (v řádech desítek MB), vyšší, než je výchozí nastavení PHP, jsem musel v kroku **19.** navýšit kapacitu (hodnoty) proměnných `post_max_size` a `upload_max_filesize` v serverové konfiguraci PHP na hodnotu 128MB:

```
php_admin_value[post_max_size] = 128M
php_admin_value[upload_max_filesize] = 128M
```

Po provedení výše uvedeného postupu byla aplikace úspěšně nahrána na webový server a je dostupná k použití na doméně: https://opt3-browser.cz.

Kapitola 6

Testování

Nepostradatelnou součástí této bakalářské práce bylo podrobit výslednou aplikaci testování. To spočívalo především v tom, zhodnotit rychlost práce s aplikací, tzn. jak rychle dokáže aplikace navracet data. Dále bylo testováním zkontrolovat a zajistit robustnost aplikace v podobě ošetření všech možných vstupů od uživatele. Jelikož je aplikace koncipovaná jako velmi specializovaná, odborná aplikace pro velmi úzký výčet uživatelů, nebyla předmětem testování přívětivost uživatelského rozhraní aplikace ani provádění hromadné testování na několika uživatelích.

6.1 Robustnost aplikace vůči chybným vstupům

Nejzranitelnějšími místy každé aplikace jsou právě ta, která jsou veřejně přístupná a ke kterým mají přístup uživatelé aplikace. Útoky na webovou aplikaci využívají nedostatků v implementaci aplikace a pomocí původně neškodně navržených mechanismů, jako např. prvky webového formuláře, skriptů, a jiných tyto nezabezpečené stránky napadají nebo zneužívají citlivých osobních dat uložených v těchto aplikacích.

Přestože existují různé mechanismy ochrany před těmito útoky v aplikačních frameworcích, je potřeba dodržovat jisté programátorské konvence, které nám tuto ochranu zaručí. Aplikační framework Laravel (viz článek [12]) skýtá ochranu před několika typy útoků, tj. ochrana proti *SQL injection* útoku, *Cross-Site Request Forgery* (CSRF) a *XSS* (Cross site scripting).

Co však framework Laravel nepokrývá jsou útoky na samotný server, na kterém se aplikace nachází. Jak již bylo řečeno, Laravel je aplikační webový framework, který nám pomáhá zabezpečit samotnou webovou aplikaci, nikoliv webový server. Pokrytí ochranou serveru před těmito útoky už zcela závisí na infrastruktuře a konfiguraci webového serveru a na způsobu nasazení webové aplikace.

I pro případ, že by uživatel žádné špatné záměry s aplikací neměl, musíme vstupy od těchto uživatelů (i autentizovaných) patřičně zabezpečit pro běžnou práci s aplikací. Validace vstupujících dat představuje vrstvu, s jejíž pomocí můžeme kýžené kontroly docílit. S využitím vestavěné validace kontrolujeme obsah a typ vstupních dat, na základě specifických pravidel a omezení pro daný vstup. Tímto způsobem jsme schopni uživateli např. kontrolovat, zda zadal správný formát vstupního souboru, nebo že vstupní pole pro uživatelské jméno opravdu obsahuje vstup v podobě formátu emailové adresy.

Nahraný databázový soubor SQLite obsahující výrobní úlohu představuje **nejdůležitější datový vstup** uživatele v aplikaci. Je proto důležité tento vstup patřičně zabezpečit

před případně chybným souborem. Testování různých forem vstupů a výsledky těchto testů můžeme vidět v tabulce 6.1.

Testovaný formát vstupu	Odpověď aplikace / výsledek zpracování aplikací	Výsledek testu
Nesprávný formát souboru	Nahrávaný soubor musí být typu: application/vnd.sqlite3, application/x-sqlite3.	Chyba odchycena
Databázový soubor neobsahující požadované schéma	Tabulka „cProp“ nebyla nalezena ve schématu nahrávaného databázového souboru.	Chyba odchycena
Nevybrán žádný soubor	Pole „soubor“ je vyžadováno.	Chyba odchycena

Tabulka 6.1: Testování vstupního místa pro nahrání souboru výrobní úlohy

Ostatní vstupy webové aplikace jsou ve formě textových polí. Všechny pokusy, dosadit jim chybný vstup byli aplikací odchyceny. Navíc, díky použití Laravel query builderu při dotazování databáze nám útoky v podobě *SQL Injection*, vyvolané z těchto vstupů nehrozí. To je především díky tomu, jelikož je ke kompozici databázového dotazu využíván balíček *Eloquent ORM*.

6.2 Rychlost práce s aplikací

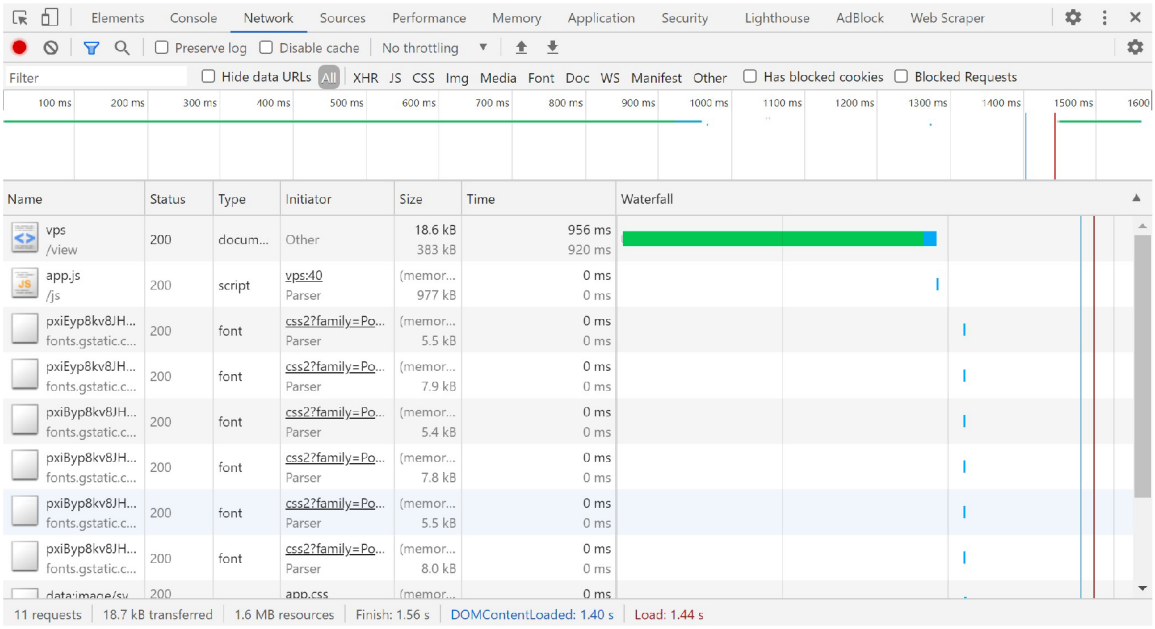
Principem implementované aplikace je neustále dávat databázové dotazy na připojenou souborovou databázi výrobní úlohy, procházením typizovaných pohledů. Rychlost odezvy u tohoto typu aplikace se dá považovat za klíčovou, tím pádem jsem se zaměřil na profilling rychlosti odezvy aplikace abych našel nedostatky v samotné implementaci a vyhodnotil její úzká místa.

K měření doby trvání požadavku jsem zvolil nástroj *Chrome DevTools*[1], zabudovaný v prohlížeči Google Chrome, který v záložce *Network* umí měřit celkovou dobu trvání dotazu načtení webové stránky i trvání jednotlivých fází / žádostí (ukázka tohoto nástroje na obrázku 6.1). Námi naměřenou a statisticky zpracovanou hodnotu představuje počáteční žádost na zaslání HTML dokumentu pohledu. Ostatní žádosti představující převážně žádosti na stažení skriptů stylů a jiných zdrojů zanedbáme.

Používáním implementované aplikace bylo vypozorováno, že proces substituce, využívající databázi k ukládání substitucí identifikátorů objektů výrobní úlohy, značně zhoršuje odezvu aplikace a její celkovou rychlost. Zhoršení rychlosti se projevovalo především při prvním navštívení pohledu nově nahraného a připojeného souboru výrobní úlohy, pro který ještě neproběhla substituce identifikátorů objektů výrobní úlohy. Tento problém byl následně důkladněji zkoumán a proběhlo měření rychlosti doby odezvy stránky, v závislosti na stavu substituce identifikátorů.

Naměřené hodnoty doby načítání stránky, tedy doby počítané od zasláním požadavku na HTML dokument po jeho navrácení serverem, byli následně zprůměrovány a sepsány do tabulky 6.2.

[1] Chrome DevTools - vývojářský nástroj - https://developer.chrome.com/docs/devtools/



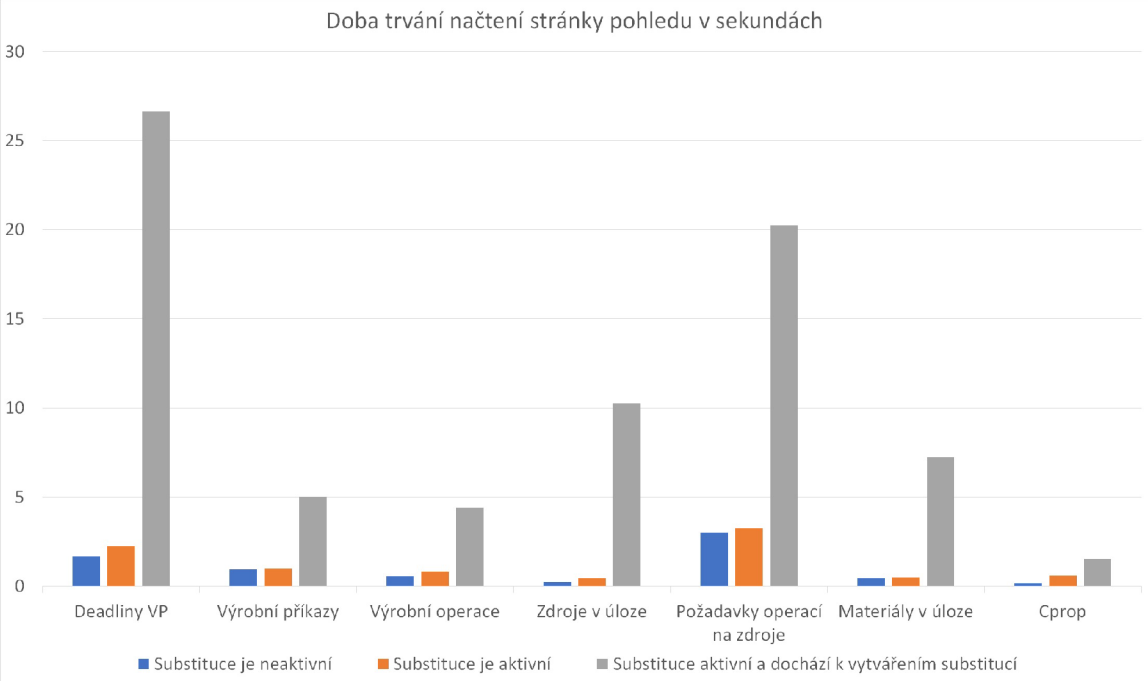
Obrázek 6.1: Nástroj Chrome DevTools prohlížeče Google Chrome. Na tomto obrázku konkrétně panel Network, analyzující dobu trvání stažení HTML dokumentu pohledu *Výrobní příkazy v úloze*.

Název pohledu	Substituce není požadována	Substituce je požadována	Substituce je požadována a dochází k substituci UUID
Deadliny výrobních	1,67 s	2,26 s	26,64 s
Výrobní příkazy	0,94 s	1,01 s	5,03 s
Výrobní operace	0,57 s	0,81 s	4,40 s
Zdroje v úloze	0,23 s	0,45 s	10,24 s
Požadavky operací na zdroje	3,01 s	3,24 s	20,23 s
Materiály v úloze	0,45 s	0,50 s	7,25 s
cProp	0,18 s	0,58 s	1,53 s

Tabulka 6.2: Naměřené hodnoty rychlosti načtení pohledů v závislosti na stavu substituce identifikátorů.

Měřením rychlosti aplikace bylo zjištěno, že použití substituce ovlivňuje rychlost načítání stránky (pohledu) především při jejím prvním načtení. Při neexistenci substituce zobrazovaného objektu výrobní úlohy a přepnutím stavu přepínače substituce do polohy vyžadující překlad UUID, dochází k okamžitému vytváření substituce a jejímu ukládání do tabulky aplikační databáze. To značně zpomaluje celý proces načítání webové stránky. Po vytvoření substituce UUID zobrazovaných objektů výrobní úlohy je rychlost načtení stránky v porovnání s případem, kdy není substituce požadována, téměř srovnatelná (naměřeno zhoršení doby odezvy v průměru o 253ms). Toto tvrzení ilustruje graf na obrázku 6.2.

Dále bylo analýzou měření a kódu zjištěno, že doba načtení stránky se odvíjí převážně od počtu objektů výrobní úlohy, vyskytujících se v aktuálním pohledu, dožadující se překladu svého identifikátoru. Problém tedy nepramení až tak z neefektivnosti samotného procesu



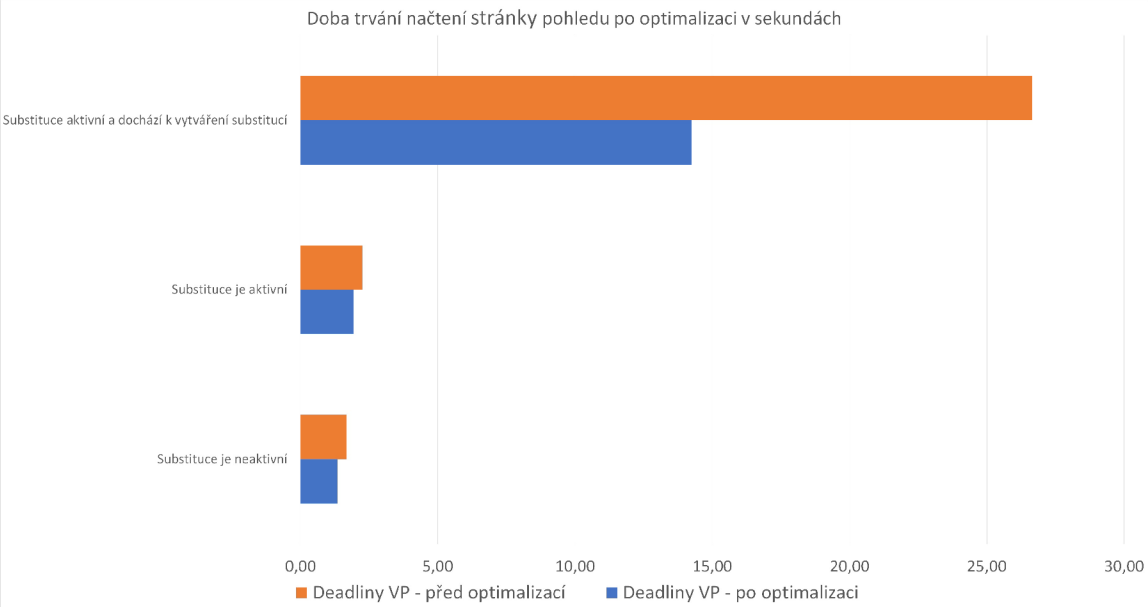
Obrázek 6.2: Sloupcový graf popisující dobu trvání načtení stránky daného pohledu, na tomto grafu konkrétně skupina pohledů tvořící seznam základních objektů výrobní úlohy (popsané v sekci 4.2). Z grafu lze vyčíst, že proces prvotní substituce identifikátoru výrobní úlohy trvá podstatně dlouhou dobu – u pohledu *Deadliny výrobních příkazů* bylo naměřeno čekání přes 26 sekund na odpověď serveru a navrácení HTML dokumentu pohledu (stránky). Avšak při navštívení stejného pohledu, aktuálně zobrazující záznamy, jejímž identifikátorům již byla vytvořena substituce, je toto trvání v porovnáním s vypnutou substitucí jen nepatrně delší, až skoro stejné. Dá se tedy konstatovat, že proces substituce identifikátorů výrobních objektů v aktuálním pohledu představuje časově náročnou operaci pouze při samotném vytváření substituce. Po jejím vytvoření se uživateli již z časového hlediska oplatí substituci při prohlížení databázových souborů výrobní úlohy využívat.

překladu identifikátoru a jeho ukládání do databáze, ale spíše v neuvážlivém návrhu pohledů a dopuštění načítání dat až několika stovek objektů výrobní úlohy bez jakéhokoliv mechanismu stránkování apod.

Z tohoto důvodu byl pro pohledy nejvíce zatíženými zhoršením rychlosti kvůli překladu k současnému mechanismu stránkování vytvořen a implementován další mechanismus postupného načítání objektů výrobní úlohy, tzv. on demand, v podobě tlačítka „Načíst další záznamy“. Pohled je načten s fixním počtem záznamů objektů výrobní úlohy, který může být v případě potřeby uživatelem navýšen a další obsah asynchronně načten a zobrazen ze serveru k původnímu obsahu. To v praxi znamená, že překlad proběhne jen pro fixně stanovené množství objektů výrobní úlohy a probíhá postupně pro další, později načtené (vyžadované) záznamy. Je tedy rozložen do delšího časového úseku, jenž vylepší průvodní rychlost načtení stránky.

U pohledu s termíny výrobních příkazů došlo využitím novému mechanismu k optimalizaci průvodního načtení stránky a substituce identifikátorů téměř o polovinu na hodnotu 14,24s oproti původním 26,64s. Pozdější načítaní pohledu, ať už s použitím substituce identifikátorů či bez ní si taktéž oproti původnímu měření v tabulce 6.2 drobně polepšilo. Srov-

nání hodnot obou měření pro pohled s termíny výrobních příkazů je dostupné na obrázku 6.3.



Obrázek 6.3: Sloupcový graf prezentující naměřenou dobu trvání načtení stránky pohledu termínů výrobních operací, srovnávající stav před a po optimalizaci mechanismem postupného načítání záznamů.

Kapitola 7

Závěr

Cílem práce bylo prostudovat zadané schéma databáze výrobní úlohy, s jehož porozuměním jsem navrhl podpůrný vývojářský nástroj v podobě webové aplikace, ulehčující úzce zacílené skupině uživatelů zobrazování a procházení dat komplexního schématu databáze. Navrhl jsem koncepci několika typizovaných pohledů, které může uživatel pro procházení svých dat použít. Za použití frameworku Laravel a dalších podpůrných nástrojů jsem navrženou aplikaci implementoval. K základní funkcionalitě procházení dat databázových tabulek jsem aplikaci obohatil o možnost vytváření substituce za původní UUID entit výrobní úlohy. Aplikaci jsem podrobil sérii testování, k zjištění míry odolnosti aplikace vůči nekorektním vstupům od uživatele a změření rychlosti trvání navrácení stránky pohledu, při překladu UUID na substituovaný tvar. Cílem bylo především odladit aplikaci na její funkčnost, jenž jsem testováním aplikace docílil, odhalil jsem nedostatky v její implementaci, které jsem posléze opravil. Následně jsem implementovanou, funkční aplikaci nasadil na produkční webový server, kde je připravena k použití na adrese: https://opt3-browser.cz.

Literatura

[1] Alenius, F. *Authentication and Authorization: Achieving Single Sign-on in an Erlang Environment* [online]. 2010 [cit. 2021-04-12]. Dostupné z: http://uu.diva-portal.org/smash/get/diva2:344199/FULLTEXT01.

[2] Basl, J. a Blažíček, R. *Podnikové informační systémy: podnik v informační společnosti.* 2., výrazně přeprac. a rozš. vyd. Praha: Grada, 2008. Management v informační společnosti. ISBN 978-80-247-2279-5.

[3] Brotherton, C. *The Most Popular PHP Frameworks to Use in 2021* [online]. Březen 2021 [cit. 2021-04-23]. Dostupné z: https://kinsta.com/blog/php-frameworks/.

[4] Composer. *Introduction* [online]. Duben 2021 [cit. 2021-04-21]. Dostupné z: https://getcomposer.org/doc/00-intro.md.

[5] Ferguson, N. *What's The Difference Between Frontend And Backend Web Development?* [online]. Leden 2021 [cit. 2021-04-22]. Dostupné z: https://careerfoundry.com/en/blog/web-development/whats-the-difference-between-frontend-and-backend/.

[6] Fowler, M. *Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch.* 1. vyd. Addison Wesley, 2012. ISBN 0-321-12742-0.

[7] Fowler, M. *Separated Presentation* [online]. Laravel 8. 2020 [cit. 2021-04-21]. Dostupné z: https://martinfowler.com/eaaDev/SeparatedPresentation.html.

[8] Hipp, R. D. *SQLite.* 2020 [cit. 2021-04-19]. Dostupné z: https://www.sqlite.org/index.html.

[9] ITU. *Universally Unique Identifiers (UUIDs)* [online]. 2009 [cit. 2021-04-25]. Dostupné z: https://www.itu.int/en/ITU-T/asn1/Pages/UUID/uuids.aspx.

[10] Jetstream, L. *Introduction* [online]. Jetstream 2. 2021 [cit. 2021-05-02]. Dostupné z: https://jetstream.laravel.com/2.x/introduction.html.

[11] Kreibich, J. *Using SQLite.* 1. vyd. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., srpen 2010. ISBN 978-0-596-52118-9.

[12] Kumar, P. *An Overview of the Best Laravel Security Practices* [online]. Červenec 2018 [cit. 2021-05-08]. Dostupné z: https://www.cloudways.com/blog/laravel-security/#security-features.

[13] Laravel. *Blade Templates* [online]. [cit. 2021-05-07]. Dostupné z: https://laravel.com/docs/8.x/blade.

[14] Laravel. *Authentication* [online]. Laravel 8. 2020 [cit. 2021-05-02]. Dostupné z: https://laravel.com/docs/8.x/authentication.

[15] Laravel. *Cache* [online]. Laravel 8. 2020 [cit. 2021-05-02]. Dostupné z: https://laravel.com/docs/8.x/cache.

[16] Laravel. *Database* [online]. Laravel 8. 2020 [cit. 2021-05-02]. Dostupné z: https://laravel.com/docs/8.x/database.

[17] Laravel. *Eloquent* [online]. Laravel 8. 2020 [cit. 2021-05-01]. Dostupné z: https://laravel.com/docs/8.x/eloquent.

[18] Laravel. *Middleware* [online]. Laravel 8. 2020 [cit. 2021-05-08]. Dostupné z: https://laravel.com/docs/8.x/middleware.

[19] Laravel. *Routing* [online]. Laravel 8. 2020 [cit. 2021-05-02]. Dostupné z: https://laravel.com/docs/8.x/routing.

[20] Livewire. *Introduction* [online]. Laravel 8. 2020 [cit. 2021-05-02]. Dostupné z: https://laravel-livewire.com/.

[21] Meyer, H. *Manufacturing execution systems: optimal design, planning, and deployment*. 1. vyd. McGraw-Hill Education, 2009. ISBN 978-0-07-162602-6.

[22] Olson, D. a Kesharwani, S. *Enterprise Information Systems: Contemporary Trends and Issues*. 1. vyd. 5 Toh Tuck Link, Singapore 596224: World Scientific Publishing Co. Pte. Ltd., říjen 2009. ISBN 978-981-4273-15-2.

[23] OOTips. *Separated Presentation* [online]. Květen 1998 [cit. 2021-05-01]. Dostupné z: http://ootips.org/mvc-pattern.html.

[24] PHP Documentation Group. *PHP Manual* [online]. 7.3. 2021 [cit. 2021-04-22]. Dostupné z: https://www.php.net/manual/en/index.php.

[25] Rae, M. *Atomic Design Principles & Methodology 101* [online]. Červen 2020 [cit. 2021-05-08]. Dostupné z: https://xd.adobe.com/ideas/process/ui-design/atomic-design-principles-methodology-101/.

[26] SQLite. *Appropriate Uses For SQLite* [online]. 3.31.1. 2020 [cit. 2021-04-20]. Dostupné z: https://www.sqlite.org/whentouse.html.

[27] SQLite. *Distinctive Features Of SQLite* [online]. 3.31.1. 2020 [cit. 2021-04-20]. Dostupné z: https://www.sqlite.org/different.html.

[28] SQLite. *Well-Known Users of SQLite* [online]. 3.31.1. 2020 [cit. 2021-04-20]. Dostupné z: https://www.sqlite.org/famous.html.

[29] Stauffer, M. *Laravel: Up & Running: A Framework for Building Modern PHP Apps*. 1. vyd. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, prosinec 2016. ISBN 978-1-491-93608-5.

[30] TechTerms. *RDBMS* [online]. Červenec 2017 [cit. 2021-05-03]. Dostupné z: https://techterms.com/definition/rdbms.

Příloha A

Obsah přiloženého paměťového média

Přiložené paměťové médium obsahuje následující soubory. Níže je vykreslena adresářová struktura popisující umístění nejdůležitějších souborů a archivů na paměťovém médiu.

- README.txt – Soubor, popisující spuštění webové aplikace
- application.zip – Archiv obsahující zdrojové soubory webové aplikace
- thesis.pdf – Soubor samotné technické zprávy
- thesis.zip – Archiv se zdrojovými soubory pro tuto technickou zprávu