
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**
**DEPARTMENT OF INTELLIGENT SYSTEMS**


# VIZUALIZACE VÝROBNÍCH ROZVRHŮ

**VISUALIZATION OF MANUFACTURE SCHEDULES**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** — **ARTSIOM LUHIN**
**AUTHOR**

**VEDOUCÍ PRÁCE** — **Ing. MARTIN HRUBÝ, 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: **Luhin Artsiom**
Program: Informační technologie
Název: **Vizualizace výrobních rozvrhů**
**Visualisation of Manufacture Schedules**
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte programování webových aplikací. Prostudujte základní schémata výrobních rozvrhů.
2. Navrhněte webovou aplikaci, která zobrazí uživatelsky přívětivě výrobní rozvrh ze zadaného souboru ve formátu SQLITE3. Navrhněte ovládací prvky pro navigaci rozvrhem (zobrazení detailu, posuv v čase, změna měřítka časové osy, selekce zdrojů, apod.).
3. Aplikaci implementujte.
4. Testujte aplikaci s použitím několika různých rozvrhů.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body.

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

Práce je zaměřena na vytvoření webove aplikace, která uživatelsky přívětivě vizualizuje výrobní rozvrh. Většina průmyslových aktivit v současné době se snaží maximalizovat využití výrobních zdrojů, což v důsledku vede ke zvýšení efektivity samotné výroby. Základním cílem aplikace je totiž dosažení zmíněné efektivity. Aplikace zobrazuje masivní výrobní rozvrh v podobě Ganttova diagramu, který lze jednoduše zkoumat a analyzovat. Diagram dovoluje různé manipulace s jeho prvky: selekce jednotlivých zdrojů, změna měřítka časové osy, zobrazení dodatečných informací. Výsledkem vizualizace je pak souhrnná představa o naplánované průmyslové výrobě.



# Abstract

Thesis is focused on creating a web application that visualizes the manufacture schedule in a user friendly way. Most industrial activities are currently trying to maximize the use of production resources, which in turn leads to increased efficiency of production itself. The basic goal of the application is to achieve the mentioned efficiency. The application displays a massive manufacture schedule in the form of a Gantt chart, which can be easily examined and analyzed. The diagram allows various manipulations with its elements: selection of individual sources, scaling of the timeline, display of additional information. The result of the visualization is a comprehensive picture of the planned industrial manufacture.



# Klíčová slova

Vizualizace, web, aplikace, Gantt, diagram, rozvrh, výroba, plán.

# Keywords

Visualization, web, application, Gantt, diagram, schedule, manufacture, plan.


# Citace

LUHIN, Artsiom. *Vizualizace výrobních rozvrhů*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Hrubý, Ph.D.

# Vizualizace výrobních rozvrhů

## 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.

.....
Artsiom Luhin
19. května 2021


## Poděkování

Toutu cestou bych chtěl poděkovat vedoucímu práce Ing. Martinovi Hrubému za vedení a užitečné rady, které mi pomohly při psaní teto práce. Kromě toho bych rád poděkoval svému spolužákovi Nikolajovi Vorobievu za různorodou pomoc během psaní závěrečné práce a celkově v průběhu studia na univerzitě. Naposled bych poděkoval své přítelkyni Viktorii Shtanko za důležitou morální podporu.

# Obsah

# Kapitola 1

# Úvod

V dnešní době informační technologie již našly své uplatnění v nejrůznějších oblastech působení člověka. Průmysl je jednou z nich. Současné konkurenční prostředí nutí podniky neustále zlepšovat své procesy. Proto klíčovým principem jakéhokoliv průmyslu se stává zvýšení jeho efektivity. Toto se obvykle dosahuje zavedením automatizace ve výrobě. Sama o sobě automatizace ale není finálním řešením. Je zcela jasné, že pro dosažení co nejvyšších výsledků práce není dostačující pouze vyměnit manuální za automatické, je nutné také vše správně uspořádat a optimalizovat.

Zmíněnými věcmi se zabývají plánovací a rozvrhovací systémy. Tyto systémy zpracovávají velká množství výrobních dat, mezi která spadá dostupnost materiálu, kapacita strojů a personálu, termíny vyhotovení produktů a různá další. Výstupem je pak rozvrh, ve kterém výrobní zdroje jsou přiřazené určitému času, který ukazuje, kdy přesně ten konkrétní zdroj bude využít. Získané rozvrhy jsou však většinou obrovských rozměrů a je velice těžké je analyzovat a zkoumat. Právě tento problém řeší následující bakalářská práce.

Hlavní motivací práce je vytvořit nástroj, který jednoduše a přehledně vizualizuje velká plánovací data a poskytne možnosti manipulace s těmito daty. Aplikace nabídne způsob snadno a efektivně prohledávat rozvrh, zaměřovat se na podstatná místa, zkoumat potřebné detaily. Výsledek vizuální analýzy bude pak možné použít na vylepšení plánování výroby.

V následujícím textu je podrobně rozebrán vývoj cílové aplikace a jsou vysvětleny použité k tomu technologie. Celá práce je rozdělena do pěti kapitol, ve kterých se postupně popusují všechna stadia vývoje konečného systému. Na začátku je čtenář uveden do problematiky výrobních rozvrhu. V této kapitole se dozví o základních pojmech a definicích, vyskytujících se v dané oblasti. Dále následuje stručný popis technologií, které budou využity při implementaci aplikace. Ústřední místo zaujímají kapitoly, zabývající se návrhem a implementací systému. Na základě struktury vstupních dat a požadavků na cílový systém jsou zde podrobně navrženy a implementovány jednotlivé jeho části. Během návrhu se vyčleňují jednotlivé komponenty systému a definují se aplikační procesy. Následně v části implementace je uvedena konkrétní struktura projektu a samotného systému, na ukázkách kódu jsou popsány zajímavé části implementace. Poslední kapitolou práce je testování a vyhodnocení výsledků. V ní je představena část systému, zodpovědná za automatické generování testovacích dat. Na těchto datech se vysvětlují postupy prováděného testování a ukazují se dosažené výsledky.

# Kapitola 2

# Problematika výrobních rozvrhů

Procesy plánování a rozvrhování jsou neoddělitelnou součástí dnešní výroby. Bezpochyby zaujímají jedno z její ústředních míst. Příčina je jednoduchá – správné plánování značně zvyšuje efektivitu jakékoliv výroby. Zvýšení efektivity potom vede k dosažení lepších výsledků. Na laické úrovni s pojmem plánování a rozvrhování jsou pravděpodobně známi všichni lidé. Cílem této kapitoly je trochu detailněji popsat uvedené procesy se zaměřením na průmyslovou výrobu.

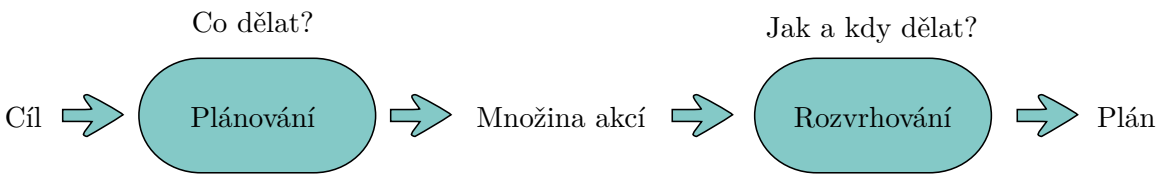
První část kapitoly uvede čtenáře do obecné teorie plánování a rozvrhování. Budou zde vysvětleny dotyčné pojmy a příčiny existence samotné teorie. Druhá část pak ukáže obecné schéma výrobního rozvrhu, jeho základní části a princip fungování.

## 2.1 Plánování a rozvrhování ve výrobě

### Co je plánování a rozvrhování

Plánování je analitycky proces, který se používá ve výrobním podniku. Jeho účelem je optimální přidělování výrobních kapacit a materiálu tak, aby byla zajištěna poptávka. Rozvrhování je pak sestavení samotného rozvrhu, kde posloupnost prováděných akci se umisťuje na časové ose s ohledem na kapacitní omezení. Tyto analytické procesy jsou většinou prováděny počítačovými programy, které k tomu používají matematické algoritmy a simulační metody. Programy respektují zadána podniková omezení a pravidla, provádějí plánování a rozvrhování a jako výsledek generují rozvrh s informacemi o možném průběhu výroby [4].

Pojmy plánování a rozvrhování se často zaměňují. Rozdíl mezi nimi vystihuje obrázek 2.1.




Obrázek 2.1: Rozdíl mezi plánováním a rozvrhováním

### Základní definice

Pro pochopení následujícího obsahu práce je nutné se orientovat v základních definicích, které se používají v dané problematice [7].

**Definice 2.1** (Operace)**.** Objekt plánování. Operace se skládá z jedné nebo více dílčích operací, které mohou být prováděny na jednom nebo více zdrojích.

**Definice 2.2** (Zdroj)**.** Objekt, který zpracovává nebo provádí jednotlivé operace.

**Definice 2.3** (Plánování)**.** Proces zajištění dostatku potřebných zdrojů za účelem dosažení stanovených cílů.

**Definice 2.4** (Rozvrhování)**.** Proces alokace zdrojů v časoprostoru za daných podmínek s cílem splnit stanovena kritéria, například minimalize nákladů.

### Současné rozvrhovací systémy

Již od šedesátých let dvacátého století se v podnicích začali objevovat informační systémy, účelem kterých bylo sledování stavu zásob a skladů. Postupem času se systémy rozvíjeli a dostávali se do fáze, kde nejenom sledovali aktuální zásoby, ale již mohli provádět materiálové plánování a kontrolovat výrobní činnosti. V devadesátých letech se objevují první ERP[1] systémy, které v transformované podobě se zachovali do dnes. Tyto systémy postupně zahrnují většinu firemních procesu, automatizují je a tímto přinášejí podnikům zefektivnění jejich provozu. V současnosti ERP systémy jsou velice komplexní softwarové nástroje. Skládají se z mnoha samostatných programu a modulů. Sledují většinu podnikových procesů z různých oblasti, jako například přijímání a skladování materiálů, plánování a řízení výroby, expedice zboží, účetnictví, řízení lidských zdrojů [12].

Tato bakalářská práce se zaměřuje pouze na jednu část takového systému – plánování výrobních zdrojů. V podobě Ganttova diagramu se provede vizualizace již naplánovaného rozvrhu. Struktura diagramu je rozebrána v následující podkapitole.

## 2.2 Schéma výrobního rozvrhu

Výrobní rozvrh je možné vizualizovat různými způsoby, nejvíce používaným ale je Ganttův diagram. Následně je uvedena obecná struktura zdrojových dat, nutných pro vytvoření diagramu takového typu. Dále následuje popis jeho struktury a vyjmenovávají se výhody a nedostatky.

### Zdrojová data

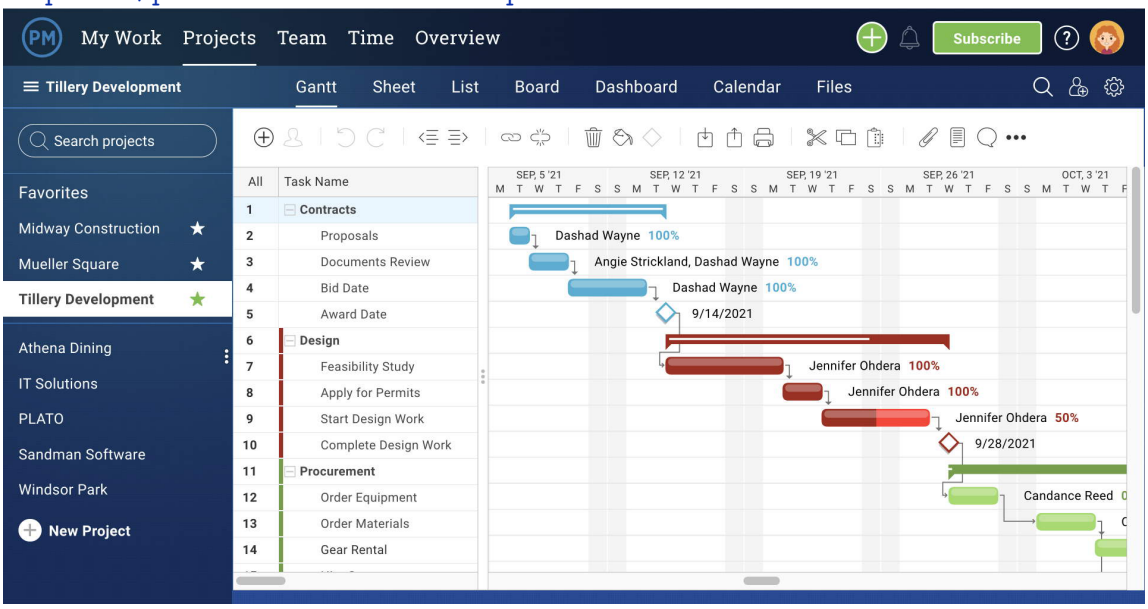
Základními daty pro výrobní rozvrh jsou informace o zdrojích a operacích. Zdrojem se rozumí objekt, který provádí určitou operaci. Nejedná se pouze o konkrétní zařízení nebo stroj. Zdrojem může být v podstatě cokoliv, co se používá při výrobě, například místnost, zaměstnanec atd. Tato skutečnost právě ukazuje na to, že zdroj zpravidla spadá do určitého typu/kategorie, což se projevuje ve výsledném rozvrhu. Operace je pak určitá akce neboli činnost, která je prováděna zdrojem. Pro operaci jsou důležitá dvě fakta. Za prvé operace má vždycky definovány časový úsek, po který je vykonávána. Za druhé tato aktivita se většinou skládá z menších části – dílčích operaci. Obojí zdroj a operace obyčejně mají zadány další atributy, které doplňují užitečné informace a podrobněji popisují naplánovány rozvrh.

[1] https://en.wikipedia.org/wiki/Enterprise_resource_planning

## Ganttův diagram

I když v současnosti se Ganttův diagram široce využívá při projektovém plánování a snad nikoho nedokáže překvapit, v době svého objevení byl považován za revoluční. Ganttův diagram je druhém pruhového diagramu, který byl vypracován H. L. Ganttem na začátku dvacátého století. V základní podobě diagram graficky znázorňuje naplánované posloupnosti činnosti v čase a vypadá následujícím způsobem 2.2.

Obrázek 2.2: Ganttův diagram. Obrázek je převzat z: https://www.projectmanager.com/templates/production-schedule-template




Horizontální osa diagramu představuje časové období trvání celé výroby. Osa je rozdělena do stejně dlouhých časových úseků (hodiny, dny, týdny). Na vertikální ose jsou pak zobrazeny jednotlivé zdroje, roztříděné do různých kategorií. Každému zdroji odpovídá jeden řádek diagramu.

Hlavní plocha schématu obsahuje operace, které jsou znázorněny pomocí obdélníků. Každá operace má definované začátek a konec, které odpovídají levé a právé straně obdélníku. Jeho délka pak reprezentuje délku trvání celé operace. Obdélníky jsou obvykle obarveny různými barvami podle toho, jaký typ aktivity představují. Vedle každé operace často bývají vypsány dodatečné informace.

Rozšířená verze diagramu může zobrazovat posloupnost a závislost mezi operacemi, kde poslední jsou propojeny pomocí šipek nebo čar. Také se mohou vyskytovat označení klíčových milníků výroby, míry dokončení jednotlivých aktivit, aktuální datum a různá další.

## Výhody a nedostatky Ganttova diagramu

Největší výhodou Ganttova diagramu je jeho přehlednost a srozumitelnost. Již na první pohled je možné odhalit, v jakém stavu se nachází aktuální výroba. Zjistit, jak silně jsou využity zdroje, v jakém stadiu jsou prováděné operace a jak hodně času zbývá do jejich završení. U pokročilejších systému je velkou výhodou možnost řízení naplánovaných procesů,

která dovoluje je různě měnit, přemisťovat, seskupovat a provádět jiné transformace. Toto všechno dělá Ganntův diagram výborným nástrojem pro analýzu, zkoumání a prezentaci.

Bohužel pro větší projekty Ganntův diagram může již být nepraktický. Hlavně proto, že celá výroba se nedokáže rozumně vejít na obrazovku, což pak zmenšuje celkovou přehlednost. S tím úzce souvisí nevýhoda, že diagram sděluje relativně málo informací na jednotku plochy. Ještě jedním důležitým úskalím diagramu tohoto typu je, že se primárně zaměřuje na časový plán, přitom ale slabě udává náročnost celkové výroby a jednotlivých operací.

# Kapitola 3

# Rozbor použitých pro aplikaci technologií

Svět webových aplikaci se v poslední době vyvíjí velice rychle. S každým rokem se na trh dostávají nové technologie a značně se vylepšují staré. Programátorovi významně ulehčují práci, a tvorba aplikací se tím pádem stává mnohem efektivnější. Ignorovat tyto technologie je v současnosti velkým nesmyslem.

V této kapitole budou představeny použité při implementaci technologie. Bude vysvětlen jejich účel a důležité vlastnosti.

## 3.1 Databázový systém SQLite

Dle zadání bakalářské práce datovým zdrojem pro vytvoření diagramu v aplikaci jsou databázové soubory ve formátu SQLite. SQLite je open-source knihovna, která implementuje samostatný, bezserverový transakční SQL databázový stroj (angl. database engine). Databázovým strojem se rozumí modul, pomocí kterého SŘBD – systém řízení báze dat – manipuluje s daty v databázi. SQLite je jedním z nejrozšířenějších databázových strojů SQL na světě.

Oproti většině SQL databází, SQLite neobsahuje samostatný serverový proces. Čtení a zápis se provádí přímo do soubouru. Celá database se tím pádem ukládá v jednom souboru souborového systému. Formát databázového soubouru je multiplatformní. SQLite je velmi kompaktní knihovnou a zároveň ji nechybí rychlost a výkonnost. Databázové transakce splňují ACID vlastnosti i v případě selhání systému, a celkově SQLite se považuje za hodně stabilní a spolehlivou knihovnu. Toto všechno dělá SQLite velmi populární volbou mezi programátory.

Protože výsledná webová aplikace bude implementována v jazyce **JavaScript** za pomocí běhového prostředí **Node.js**[1] (spolu s NPM je popsáno v následující podkapitole), k napojení aplikace na databázi bude využít **NPM**[2] modul `sqlite3`. Podrobněji proces navázání spojení bude vysvětlen v kapitole, zabývající se implementací konečné aplikace.

[1] Node.js: https://nodejs.org/
[2] Node Package Manager: https://www.npmjs.com/

## 3.2 JavaScript. Verze ES5 a ES6

### Vlastnosti jazyku JavaScript

JavaScript je multiplatformnní, dynamický, interpretovaný programovací jazyk, nejrozšířenější implementace třídy jazyků **ECMAScript**[3]. Je pravděpodobně nejvhodnějším prostředkem pro vývoj webových aplikací. I když je nejvíce znám, jako skriptovací jazyk pro webové stránky, jeho působení se stále rozrůstá a aktuální oblast použití zahrnuje mnohem víc. Dnes se často vyskytuje v jiných než prohlížečových aplikacích, a dokonce i ve vestavěných systémech.

JavaScript je nejvíce orientován na klientskou část webové aplikace. Zpravidla je zahrnut do HTML dokumentu v podobě skriptu a po spuštění jeho kód interaguje s DOM[4] elementy. Typickými příklady používání JavaScriptu mohou být:

- Načítání dodatečného obsahu stránky bez opětovného načtení celé stránky prostřednictvím technologie Ajax[5]
- Animace webové stránky, jako například zobrazení/schování objektů, jejich přemísťování, změna velikosti atd.
- Validace hodnot ve vstupních polích

S ohledem na výše zmíněné vlastnosti a protože hlavním cílem bakalářské práce je vytvoření interaktivní aplikace umístěné na straně klienta, jako implementační nástroj jsem bezpochyby zvolil JavaScript.

### Jednotlivé verze

JavaScript jako programovací jazyk prošel několika etapami standardizace během svého vývoje. Proces standardizace provádí nezisková organizace Ecma International. Dnes je již oficiálně publikováno 11 verzí programovacího jazyku ECMAScript. ECMAScript je totiž formální název univerzálního skriptovacího jazyku popsaného standardem ECMA-262. JavaScript je pak nejznámější implementaci tohoto jazyku [1].

Nejvíce významnými verzemi jsou pravděpodobně 5. a 6. revize jazyku ECMAScript – **ECMAScript 5** a **ECMAScript 6**. Však po nějaké době obě verze byly přejmenovány a tak získaly název podle roku vydání. Následně je uveden seznam několika významných vlastností, zavedených v těchto verzích.

- Nové metody pro práci s objekty typu `Array`
- Podpora formátu `JSON`
- Třídy
- Anonymní funkce
- Objekty typu `Promise` pro asynchronní programování

Všechny zmíněné vlastnosti značně podporují a rozšiřují možnosti jazyku JavaScript a proto budou využity při implementaci výsledné aplikace.

[3] https://www.ecma-international.org/publications-and-standards/standards/ecma-262/
[4] https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
[5] https://en.wikipedia.org/wiki/Ajax_(programming)

## 3.3 Technologie HTML, SVG a CSS

### HTML

Hypertext Markup Language, neboli HTML, je nezbytnou technologií pro jakoukoliv webovou aplikaci. Je to značkovací jazyk používaný pro tvorbu webových stránek.

HTML sémanticky popisuje strukturu webového dokumentu. K popisu využívá množinu značek (angl. tags) a jejich vlastnosti (anlg. attributes). Použitím těchto prostředků se vytváří strukturovaný obsah, kde části dokumentu uzavřené mezi odpovídající značky se vyčleňují jako určité sémantické jednotky. Názvy značek a jejich vlastnosti jsou uzavřeny mezi uhlové závorky ⟨ a ⟩. Otevírací a zavírací značky dohromady s vnitřním obsahem pak tvoří elementy, ze kterých se skládá celý dokument.

### SVG

Pro vykreslení některých části diagramu konečné aplikace bude také využita technologie SVG. Zkrátka SVG z anglického jazyku označuje Scalable Vector Graphics. SVG je také jako HTML značkovacím jazykem. Používá se pro vykreslení elementů dvojrozměrné vektorové grafiky.

Základním principem SVG je to, že výsledná grafika je popsána ne jednotlivými pixely uspořádanými do mřížky, ale seznamem základních přesně definovaných útvaru. Proces vytváření jednotlivých elementů je velmi podobný HTML a nepotřebuje další vysvětlení. SVG je ideálním nástrojem pro tvorbu jednoduché grafiky. Mezi jeho hlavní výhody patří:

- Nezávislost na rozlišení
- Jednoduchá spravovatelnost a čitelnost
- Malá výsledná velikost
- Nezávislost na platformě
- Dynamická upravovatelnost z CSS nebo JavaScript

SVG gravika je dnes podporována drtivou většinou prohlížečů.

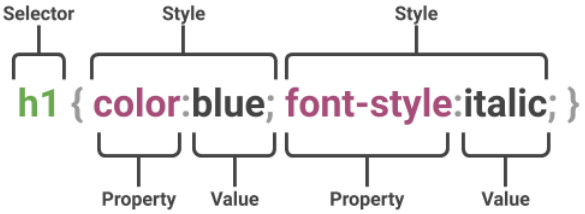
### CSS

Cascading Style Sheets je programovací jazyk, který se používá pro popis zobrazení dokumentu vytvořeného pomocí značkovacího jazyků typu HTML. Principem je pomocí určité syntaxe vydefinovat pravidla, které provedou selekci jednotlivých elementů obsahu stránky a určí jejich vzhled.

Hlavním důvodem pro vytvoření CSS a jeho vyčlenění jako samostatného programovacího jazyku bylo rozdělení prezentace a obsahu. Toto rozdělení totiž přináší programátorovi velkou flexibilitu a kontrolu nad stylováním. Dovoluje mu například vydefinovat určité styly v jednom souboru, které pak budou sdíleny mezi mnoha stránkami. Výsledkem je odstranění zbytečné duplicitity a zjednodušení procesu programování.

Syntaxi CSS krásně popisuje následující obrázek 3.1.

Obrázek 3.1: Syntaxe CSS. Obrázek je převzat z: https://lucidar.me/en/web-dev-class/lesson-2-05-css-selectors/




## 3.4 Běhové prostředí Node.js

Node.js je multiplatformní, asynchronní běhove prostředí pro JavaScript, které ke svému běhu využívá V8 stroj[6] od společnosti Google a vykonává kód jazyku JavaScript mimo webový prohlížeč. Hlavním účelem této technologie je programování serverové části dynamických webových aplikací.

Node.js přináší důležitou koncepci událostmi řízeného programování[7] do vývoje webových serverů. V architektuře tohoto typu jednotlivé funkce programu nejsou blokující. To znamená, že program nečeká na dokončení určité externí operace a pokračuje ve vykonávání. Až se tato operace dokončí, vytvoří se událost a na její vznik program zareaguje provedením předem určené reakce – „callback“ funkce. Popsaná architektura umožňuje programátorům vytvářet rychlé a škálovatelné systémy bez použití vláknového programování [10].

Při programování Node.js aplikací je dobrým zvykem používat nástroj NPM – Node Package Manager. Je to nástroj pro spravování balíčku z prostředí Node.js. Nástroj však nenabízí pouhou práci s balíčky, ale dovoluje kompletně spravovat celý projekt – překládat zdrojové kódy, sledovat instalované závislosti, definovat vlastní skripty atd. Také umožňuje vytvářet své vlastní balíčky a publikovat je.

Práce s NPM je velice jednoduchá. Pro inicializaci projektu je dostačující v příkazovém řádku provést příkaz `npm init`. Pokud se vytváří nový projekt, bude nutné uvést některé základní informace. V případě již existujícího projektu se všechno nainstaluje automaticky. Informace týkající se správy projektu jsou umístěny v souborech `package.json` a `package-lock.json`. Jsou v nich obsaženy veškeré závislosti projektu a různé metainformace. Další potřebné pro vývoj balíčky se jednoduše doinstaluji pomocí příkazu `npm install <package-name>`. Podrobnější informace lze nalézt na oficiálních stránkách[8].

## 3.5 Webový rámec Express

Express.js je běžně považován za standardní serverový rámec používaný spolu s Node.js. Je navržen pro vývoj webových a mobilních aplikací. Je oblíben mezi programátory za svou jednoduchost a minimalistický návrh. Po instalaci nabízí základní funkcionalitu, která se lehce rozšiřuje skrz doinstalování potřebných balíčků. V navrhované aplikaci bude využít pro implementaci serverové aplikační vrstvy [8].

[6] https://en.wikipedia.org/wiki/V8_(JavaScript_engine)
[7] https://en.wikipedia.org/wiki/Event-driven_architecture
[8] https://www.npmjs.com/

# Kapitola 4

# Návrh aplikace pro vizualizaci rozvrhů

Daná kapitola se zabývá návrhem výsledného vizualizačního systému. Na její začátku se rozebírají původní požadavky na aplikaci a popisují se klíčové vlastnosti, které musí být splněny. Po analýze vstupních požadavků se provede dekompozice řešení. Výsledkem dekompozice je seznam samostatných částí aplikace. Každé z těchto částí budou věnovány samostatné pasáže textu. Čtenář se nejdřív seznámí s formátem dat nutných k zobrazení a jejich významu. Následně se dozví o návrhu klientské části aplikace, která je ústředním místem celého systému. Bude zde představen návrh jednotlivých komponent systému a jeho požadované funkcionality. Na závěr bude uvedeno schéma budoucího uživatelského rozhraní se zobrazením základních grafických elementů, jejich rozložení a významu.

## 4.1 Požadavky na aplikaci

Požadavky nikdy nevznikají jen tak. Vždycky mají za sebou nějakou příčinu, která musí vést k určitému cíli. Toto samozřejmě platí i pro dotyčnou aplikaci. Primárním účelem mnou navrhovaného systému je **přívětivé zobrazení naplánovaného výrobního rozvrhu**. Za tímto účelem se skrývá seznam konkrétních požadavků, při splnění kterých bude dosažen hlavní cíl.

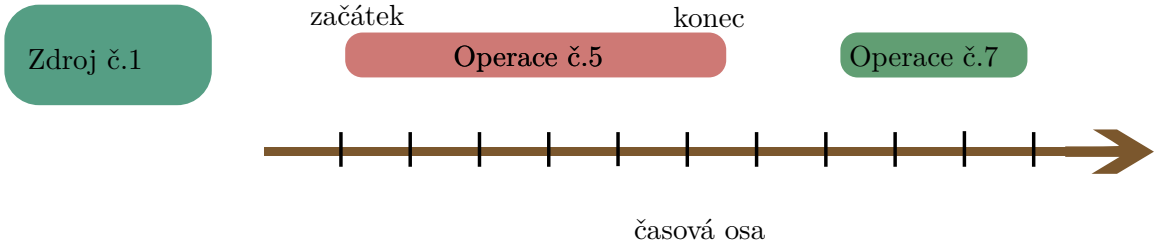
### Vstupní data a jejich vykreslení

Prvním požadavkem je samo o sobě **vykreslení obrovského množství** dat ve vhodné čitelné podobě. Základní jednotkou těchto dat jsou informace o jedné naplánované operací. Operace, neboli aktivita, představuje určitou činnost prováděnou na konkrétním zdroji. Operace je zpravidla popsána seznamem atributů, které ji dodatečně charakterizují. Avšak hlavním a určujícím rysem každé operace je její časové umístění. Toto totiž znamená, že každá aktivita má přesně definovaný začátek a konec její vykonávání.

Zdroj je druhým zásadním objektem vstupních dat. Reprezentuje objekt, který zpracovává přidělené mu operace. Každý zdroj má naplánované velké množství takových operací, jejich počet se pohybuje od desítek do několika stovek. Všechny ativity vztažené k určitému zdroji musí být zobrazeny na jednom řádku výsledného diagramu. Takové zobrazení znamená, že se operace provádějí postupně jedna za druhou v čase. Pro zdroje určitého typu však tato skutečnost nebude platit – jejich operace se budou moct překrývat, tudíž v jed-

nom časovém okamžiku zdroj bude zpracovávat dvě aktivity zároveň. Toto bude podrobněji popsáno v následujících podkapitolách.

Zjednodušené základní schéma ukazující vztah mezi zdrojem a operacemi je uvedeno na obrázku 4.1. Toto schéma je vlastně východiskem pro finální vizualizaci výrobního rozvrhu.




Obrázek 4.1: Shema zdroje a jeho operací (aktivit) umístěných na časové ose

### Časová osa, filtr operací a okno s informacemi

Horizontální umístění velkého počtu operací na časovou osu označuje obrovskou výslednou šířku diagramu. Tento fakt ukazuje na potřebu snadné orientace a pohybování mezi operace v horizontální rovině. Což vede ke druhému požadavku na aplikaci – možnost **změny měřítka časové osy**, její **škálovatelnost**. Po provedené změně se diagram musí překreslit v souladu s vybraným měřítkem.

Každá výrobní operace má definovaný její typ, který jistým způsobem ji charakterizuje. Přítomnost tohoto faktu vede ke dvěma dalším požadavkům. Za prvé operace musí být **vizuálně rozlišeny podle typu**, do kterého spadají. Za druhé diagram je povinen dovolit **možnost filtrování** jednotlivých aktivit s ohledem na jejich typ. Tyto vlastnosti aplikace značně zpřehlední vizualizovaný rozvrh a zvýší flexibilitu při jeho budoucím zkoumání.

Posledním důležitým požadavkem týkajícím se výrobních aktivit je **okno s dodatečnými informacemi**, které musí být zobrazeno po kliknutí na operaci. Okno může obsahovat podrobnější popis vybrané operace, její přesný začátek a konec, vztah k závislému zdroji a příbuzným operacím anebo další užitečné informace.

### Panel zdrojů a jejich selekce

Dalším požadavkem na výsledný systém je umístění výrobních zdrojů na **samostatném bočním panelu**. Jelikož výrobní plán je obsahuje ve velkém počtu, výsledná výška diagramu bude také mnohem větší než obrazovka zařízení, na kterém se bude zobrazovat. Navíc podobně jak je tomu u operací, každý zdroj spadá do určité kategorie. Z dvou daných skutečnosti přímo plynou upřesňující požadavky na panel zdrojů. Za prvé je nutné, aby v ní obsažená zařízení byly **roztříděné podle kategorií**. Za druhé pro pohodlnější manipulaci se zdrojí aplikace musí umožňovat **shovavat a zpětně zobrazovat jednotlivé kategorie**.

Při zkoumání diagramu je velmi důležité mít možnost si vybírat, které zdroje budou následně vykresleny. Tato funkcionalita hodně zvyšuje flexibilitu a ulehčuje proces analyzování rozvrhu. Poskytuje způsob se zaměřovat na určité zdroje a přitom nezatěžovat diagram zbytečnými informacemi. Dalším požadavkem na aplikaci je proto **selekce jednotlivých zdrojů** určených k zobrazení.

### Vyhledávač a nahrávání souborů

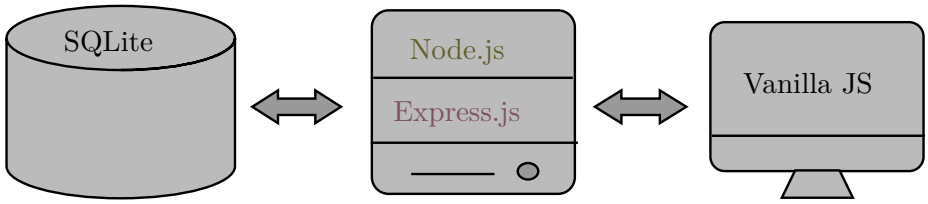
Jak již bylo řečeno, výrobní rozvrhy skutečně používané v praxi většinou zaujímají velké rozměry. Vyhledávání určitého místa pak v takovém diagramu se může stát příliš dlouhým. Proto zcela nezbytnou funkcionalitou je **přítomnost vyhledávače**. Jedna se o prvek, který se použije pro vyhledávání jak konkrétního zdroje/operace, tak i určitého data. Při úspěšném nalezení se provede skok a diagram se automaticky zaměří na zvolené místo.

V neposlední řadě je pro aplikaci důležitý jednoduchý proces **nahrávání souboru** se vstupními daty. K tomu naštěstí hodně přispívá jednosouborový formát použité při implementaci SQLite databáze. Nahrávání tím pádem bude možné provést pouhým vybráním souboru z lokálního úložiště.

## 4.2 Dekompozice

Absolutní většina webových aplikaci v dnešní době má stejnou základní strukturu. Skládají se ze tři významných částí propojených mezi sebou – databáze, server a klient. Odlišnosti pak vznikají v typu zvolené databáze, jestli se používají webové služby pro generování dynamického obsahu, jak velkou část aplikace zaujímá klientský kód atd.

Navrhovaná aplikace je též složená z těchto částí. Databázovou vrstvu tvoří systém **SQLite**. Pro serverovou část aplikace byl zvolen rámec **Express**, který poběží v běhovém prostředí **Node.js** [5]. Samotný diagram se pak na klientské straně vykreslí s použitím **HTML/CSS** a **Vanilla JS** (je to název pro čistý JavaScript, bez dalších knihoven jako jQuery). Pro některé komponenty, jako vyhledávač a element pro výběr data, však budou využity dodatečné knihovny. Architektura aplikace je uvedena na obrázku 4.2.




Obrázek 4.2: Architektura aplikace

Databázová a serverová vrstvy budou v aplikaci představeny jen v základní podobě. Pro program je totiž dostačující pouhé načtení dat z několika databázových tabulek a jejich minimální transformace do vhodného tvaru. Pak jsou data hned předána klientské části aplikace, kde se bude odehrávat veškerá důležitá logika a samotné vykreslování. Nehledě na to, že v daném okamžiku pro aplikaci není potřebná velká serverová vrstva, technologie použité pro vytvoření její základu jsou velice škálovatelné. A v případě potenciálního budoucího vývoje nebudou omezovat rozšíření programu.

Největší část aplikace tvoří klientská strana. Zde se vlastně nachází jádro celého systému. Protože při vykreslování obrovského diagramu je kladen důraz na rychlost, pro klientskou část aplikace byl zvolen Vanilla JS. Tato část systému bude vytvořena v podobě objektové JavaScript aplikace rozbité do jednotlivých modulů, které se pak jako jeden celek naimportují do cílové HTML stránky.

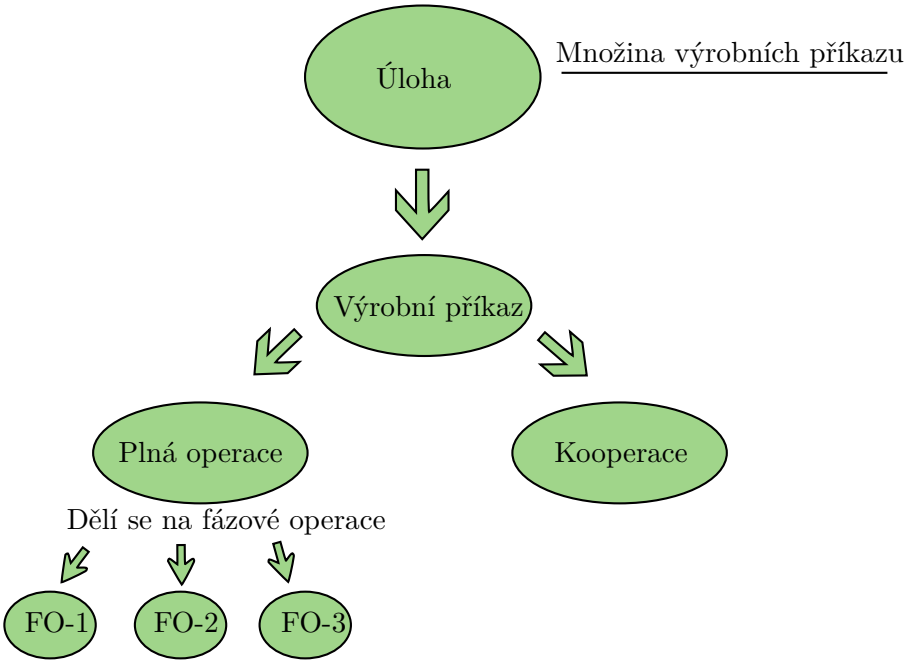
## 4.3 Datová vrstva

Pro správnou vizualizaci dat je důležité porozumět formátu jejich ukládání v databázi, pochopit význam objektů databázového modelu a jejich atributů. Tato podkapitola se zaměřuje na detailní analýzu struktury vstupních dat nutných pro vykreslení diagramu. Na závěr je navržen proces načítání těchto dat.

### Základní databázové tabulky

Veškerá potřebná data pro výsledný diagram se načítají v podstatě pouze z jedné databázové tabulky – `PlannedTimeWindow`. Tato tabulka obsahuje výsledek již provedeného rozvrhování výroby. Tímto výsledkem je množina naplánovaných v čase aktivit (operací).

Před samotným popisem databázových tabulek je třeba dodat, že všechny aktivity jsou uspořádány do hierarchické struktury 4.3:




Obrázek 4.3: Hierarchická struktura aktivit

Tabulka `PlannedTimeWindow` zahrnuje následující atributy:

- `whoseId` – ID skladu (aktuálně se v aplikaci nepoužívá)
- `OperationId` – ID aktivity
- `EquipmentId` – ID zdroje nebo materiálu
- `TimeWindowType` – Typ aktivity
- `TimeWindowStart` – Datum začátku aktivity
- `TimeWindowEnd` – Datum konce aktivity
- `Layer` – Vrstva zdroje s vícenásobnou kapacitou
- `Iterations` – Počet provedených cyklů výroby (nepoužívá se)

Z výše uvedených atributů pravděpodobně potřebuje dodatečné vysvětlení pouze atribut `Layer`. Některé stroje (stroj je typem zdrojů, viz. tabulku `Resources` totiž mohou mít násobnou kapacitu. To znamená, že v jednom časovém okamiku jsou shopné provádět několik aktivit zároveň. Pro rozlišení se potom používá atribut `Layer`.

Číselník `TimeWindowType` obsahuje možné typy aktivit:

- 0 – fázová operace „výroba“ (ITER)
- 1 – fázová operace „instalace“ (INST)
- 2 – fázová operace „rozjezd“ (START)
- 3 – plná operace
- 5 – údržba
- 7 – produkce materiálů
- 8 – fázová operace „ukončení“ (FIN)
- 9 – kooperace

Pomocná tabulka `Resources` (použije se pro zjištění typu zdroje):

- `sId` – ID zdroje
- `type` – Typ zdroje (0 – stroj, 2 – personál, 3 – nástroj)
- `maxICap` – (aktuálně se v aplikaci nepoužívá)
- `maxDCap` – (aktuálně se v aplikaci nepoužívá)

### Ukázka rozvrhu plné operace

Popsané výše databázové tabulky obsahují veškeré nezbytné informace pro vykreslení výrobního rozvrhu. Pro lepší pochopení rozvrhování plné operace a tudíž i jednodušší orientaci ve vstupních datech je vhodné uvést schematický příklad.

Představme si, že máme jednu výrobní plnou operaci OP, která se skládá ze dvou fází: OP-INST a OP-ITER. Pro svůj běh tato operace bude potřebovat zdroje: stroj M (machine), nástroj T (tool) a personál P.

Operace OP zažádá o zdroje následujícím způsobem:

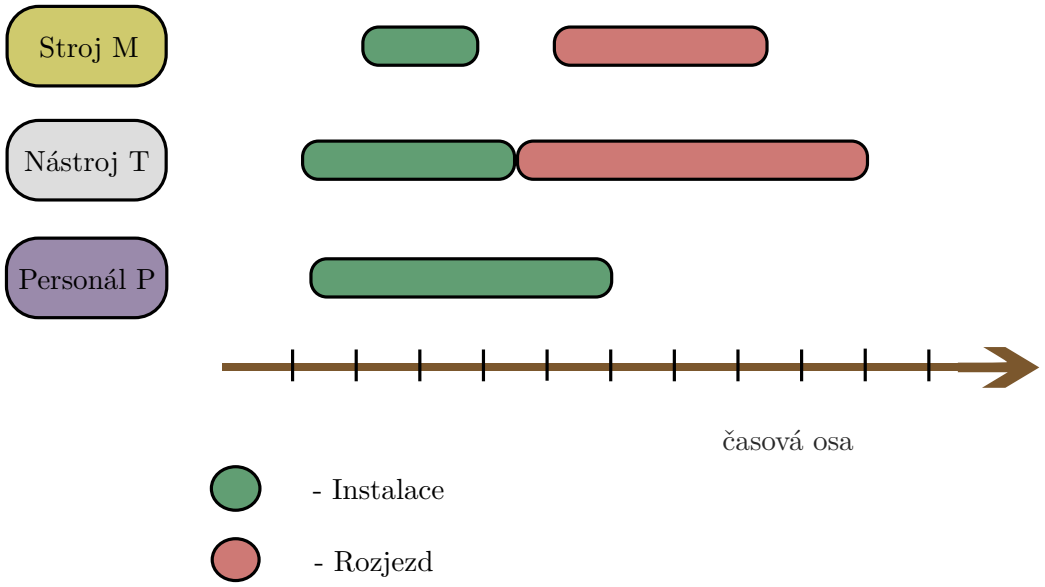
- OP žádá stroj M (obě dvě fáze potřebují stroj M)
- OP žádá nástroj T (obě dvě fáze potřebují nástroj T)
- OP-INST žádá personál P (pouze fáze instalace potřebuje personál P)

Rozvrh v takovém případě bude obsahovat šest aktivit. Všechny záznamy o aktivitách se budou nacházet v databázové tabulce `PlannedTimeWindow`. Aktivity jsou:

- OP-INST - stroj M, typ aktivity 1
- OP-INST - nástroj T, typ aktivity 1
- OP-INST - personál P, typ aktivity 1
- OP-ITER - stroj M, typ aktivity 0

- OP-ITER - stroj T, typ aktivity 0
- OP - typ aktivity 3 (plná operace)

Výsledný diagram pro danou operaci by pak mohl vypadat následovně 4.4:



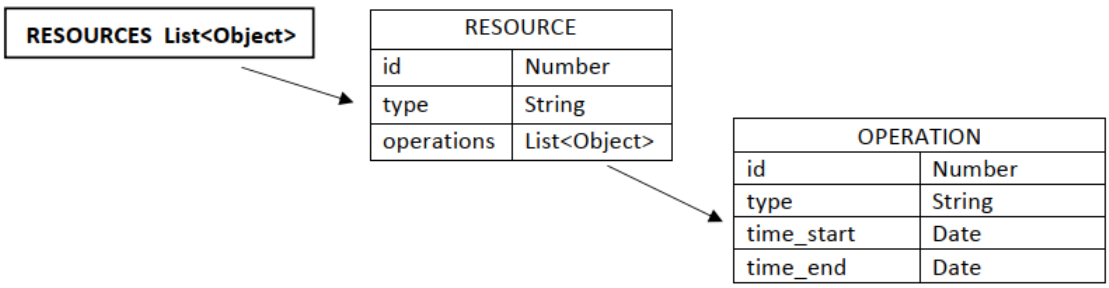

Obrázek 4.4: Ukázka schematického rozvrhu plné operace

### Načítání aktivit

Návrh přístupové vrstvy k databázi aplikace je jednoduchý. Je nutné pouze načíst aktivity z tabulky `PlannedTimeWindow` a pak je správně uspořádat. K tomu postačí jeden objekt, který po nahrání databázového souboru se k němu připojí (pomocí knihovny `sqlite3`), provede databázový dotaz a poté uspořádá získána data do vhodné formy.

Data po uspořádání budou mít následující podobu 4.5. **`Resources`** představuje seznam objektů výrobních zdrojů (`RESOURCE`), kde každý z těchto zdrojů obsahuje seznam vztažených k němu objektů operací (`OPERATION`).

Obrázek 4.5: Typ uspořádání dat načtených z databáze



### Přeposlání vyčtených dat

Aplikační vrstva (z pohledu návrhu MCV[1]) ve výsledném systému bude mít spíše symbolický charakter. Jejím hlavním účelem bude funkcionalita nahrávání souboru a následné vyčtení jeho dat. Hned po vyčtení data budou přeposlána do klientské části aplikace, kde proběhne jejich další zpracování a následné zobrazení v podobě Ganttova diagramu.

## 4.4 Klientská část aplikace

Protože cílem je navrhnout systém, který dynamický vykresluje výrobní rozvrh, veškerá logika – příprava diagramu, jeho eventuální transformace a následné překreslování – je posunuta do klientské části aplikace. Tato skutečnost dělá klientskou část jádrem systému, kde probíhají všechny zásadní činnosti.

Informace o zdrojích a aktivitách se načítají z databáze pouze jednou, po nahrání souboru. Pak aplikace pracuje s těmito načtenými daty a již nesahá do databáze. Znovunačtení dat se uskuteční pouze po restartu aplikace anebo po opětovném nahrání souboru (více o tom v kapitole, týkající se implementace). Tento fakt znamená, že všechny úpravy provedené uživatelem behem práce s diagramem neovlivní samotná zdrojová data, ale pouze jejich vizualizaci.

### Aplikační procesy

Zásadním rysem vykreslování diagramu je to, že všechny jeho části se generují dynamicky. Po jeho úpravě (například odstranění některých zdrojů, změně časové osy, vystavení určitého filtru) se musí znovu přepočítat potřebné údaje získané z původní sady databázových operací. Pak na základě nových informací se diagram opět celkově překreslí.

Celý proces vykreslení je tím pádem možné rozbít na tři logicky oddělené části – **příprava dat**, **samotné vykreslování**, **navázání funkčních událostí**. Dále se podrobně rozepisují detaily jednotlivých částí. Pro každý dílčí proces bude vypsán seznam věcí, které musí být ovlivněné (upravením, přepočítáním, překreslením). Za pomlčkou pak bude následovat podrobnější popis akce.

**Příprava dat**:

- Obecná nastavení – Provede se základní konfigurace diagramu: implicitní rozměry řádků a sloupců, režim zobrazení časové osy, odsazení atd.
- Zdroje a operace – Na základě parametrů zvolených uživatelem se provedou úpravy zdrojových dat pro následné správné zobrazení.
- Časová osa – S ohledem na zdrojová data a zvolený režim časové osy se přepočtou její dílčí jednotky a výsledné rozmezí.
- Vyhledávač – Obnoví se data pro vyhledávač zdrojů a operací.

**Vykreslování**:

- Primární vrstvy – Vykreslí se základní vrstvy-kontejnery (HTML nebo SVG) pro časovou osu, panel zdrojů a operace.

[1] https://en.wikipedia.org/wiki/Model-view-controller

- Časová osa – Překreslí se obsah časové osy.
- Panel zdrojů – Vykreslí se panel se zdroji, roztříděnými podle kategorií.
- Operace – Vykreslí se jednotlivé operace a jejich fáze.

**Navázání funkčních událostí**:

- Časová osa – Navážou se události změny měřítka časové osy.
- Panel zdrojů – Události selekce jednotlivých zdrojů, selekce všech zdrojů, zobrazení/schování kategorie.
- Operace – Události zobrazení okna s informacemi.
- Filtr operací – Události selekce typu operací, určených k zobrazení.

Výše popsané procesy znázorňují logicky běh programu – posloupnost akcí, které se provedou během jednoho vykreslení diagramu. Tyto procesy jsou na různých etapách vykreslovacího cyklu prováděny různými komponentami systému, které dohromady tvoří klientskou část aplikace.

### Základní komponenty

Systém na klientské straně je rozdělen do několika komponent, které spolupracují během procesu vytváření diagramu. Dále následuje popis těchto komponent.

**Diagram** (Chart)
Hlavní komponenta. Odpovídá za počáteční konfiguraci, přípravu zdrojů a operací, přepočet časové osy. Vykresluje vrstvy-kontejnery pro vnitřní elementy diagramu (časovou osu, panel zdrojů, pole operací). Volá komponenty **Časová osa**, **Kategorie**, **Zdroj**, **Operace**. Inicializuje pomocné elementy diagramu Ovladač časové osy, Vyhledávač zdrojů a operací, Vybírač data, Filtr operací.

**Časová osa** (Timeline)
Komponenta připravuje data pro časovou osu v závislosti na vybraném režimu. Následně provádí její vykreslení.

**Kategorie** (Category)
**Zdroj** (Resource)
**Operace** (Operation)
Tyto tři komponenty provádějí stejnou posloupnost akci. Na začátku připravují potřebná pro jejich fungování data (například rozměry elementu operace, pořadí elementu zdroje atd.). Dále vykonávají vykreslování. Na závěr inicializují funkce poslouchání odpovídajících událostí.

**Informační okno** (Information popup)
Komponenta poslouchá událost kliknutí na objekt operace. Po uskutečnění události vykreslí okno s podrobnými informacemi o dané operaci.

Ovladač časové osy (Timeline control)

Element pro výběr režimu časové osy diagramu.

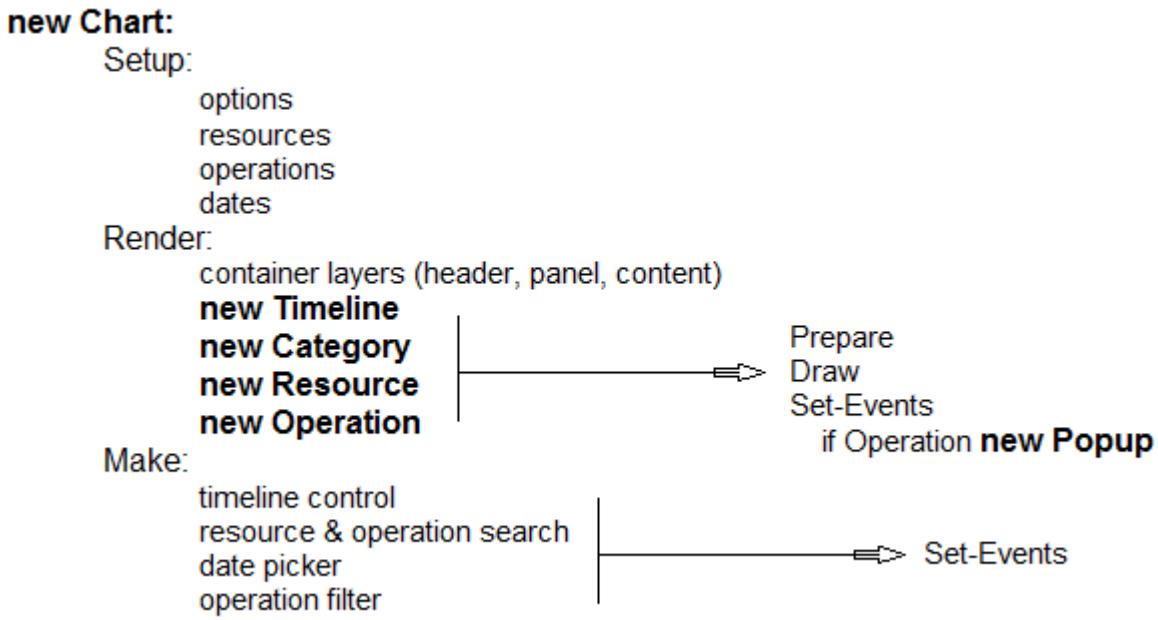
Vyhledávač zdrojů a operací (Resouce & operation search)
Element pro vyhledávání konkrétního zdroje nebo operace. Po úspěšném nalezení provede vycentrování diagramu na odpovídající zdroj/operaci.

Vybírač data (Date picker)
Element pro vybírání určitého data. Po vybrání se provede skok na zvolené datum.

Filtr operací (Operation filter)
Element pro vybírání typu operací určených k zobrazení. Poskytuje možnost vícenásobného výběru. Po nastavení se vykreslí operace pouze vybraného typu.

Průběh vykreslování diagramu ve schematické podobě je znázorněn na obrázku 4.6.

Obrázek 4.6: Průběh vykreslování diagramu ve schematické podobě

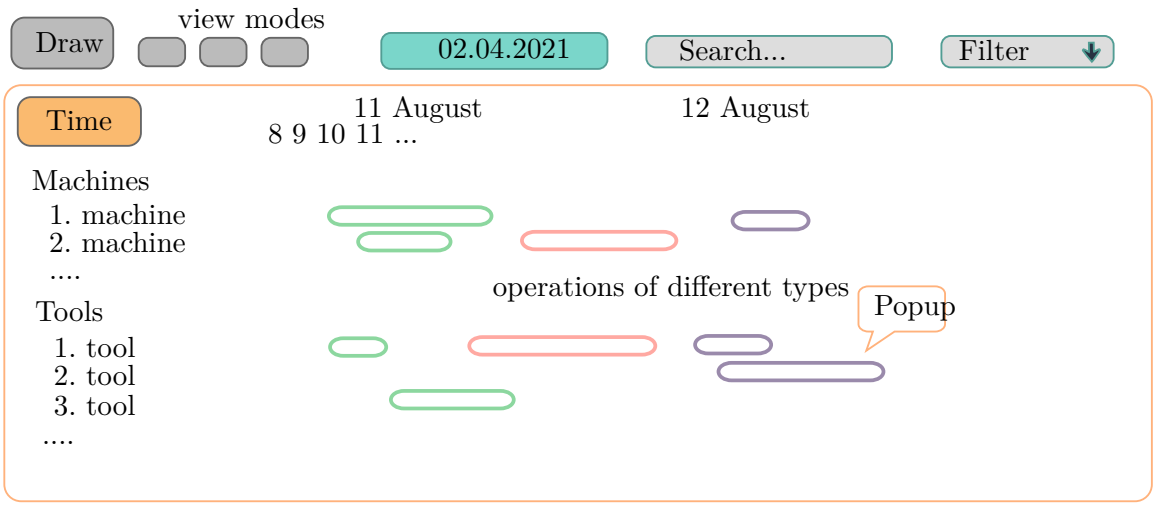



## 4.5 Uživatelské rozhraní a funkcionalita

Uživatelské rozhraní je bezpochyby velmi důležitým prvkem celé aplikace. Rychle určuje, jestli vůbec vyvinutá aplikace bude někým používaná. Špatně navržené uživatelské rozhraní nedokáže zachránit aplikaci i když její funkcionalita je na dobré úrovni. Proto je vždycky vhodné inspirovat se existujícími systémy a hodně přemýšlet nad výsledným řešením.

Návrh uživatelského rozhraní dotyčné aplikace hodně ulehčuje skutečnost, že zhruba 80% výsledné aplikace bude tvořit schema rozvrhu v podobě Ganttova diagramu. Ganttův diagram již existuje dlouhou dobu a jeho vzhled je dobře vydefinován. Návrh pomocných elementů aplikace byl inspirován existujícími komerčními řešeními.

Výsledné uživatelské rozhraní bude mít následující podobu 4.7. V horní části aplikace se umístí ovládací elementy: vykreslovací tlačítko, tlačítka pro změnu režimu časové osy, element pro výběr data, vyhledávač zdrojů a operací, filtr typu operací. V dolní části se bude nacházet samotný diagram naplánovaného rozvrhu. Diagram bude rozdělen do tří částí: hlavička s časovou osou, panel zdrojů a pole operací. Všechny zdroje na bočním panelu budou rozděleny podle kategorie. Vedle každé kategorie se umístí tři tlačítka, která dovolí zobrazit předtím schované zdroje, vybrat všechny zdroje a zrušit tento výběr, zobrazit a shovat celou kategorii. Každý zdroj navíc bude obsahovat tlačítko pro jeho selekci. V hlavním poli diagramu budou vykresleny jednotlivé operace obarvené podle jejich typu. Při kliknutí na operaci se zobrazí okno s dodatečnými informacemi.




Obrázek 4.7: Schéma uživatelského rozhraní aplikace

# Kapitola 5

# Implementace navrženého systému

Již ve stadiu návrhu budoucí aplikace bylo zřejmé, že výsledný program bude implementován ve webovém rozhraní a jeho hlavní a největší částí bude klientská. Systémy takového typu běží na straně klienta a proto nosí název„client-side“ aplikace. Ve většině případů pro implementaci takových aplikací se volí programovací jazyk JavaScript. V mém případě navíc bylo důležité vyvinout rychlý systém vykreslování rozvrhů. Proto pro implementaci ústřední části systému byl vybrán čistý JavaScript standardu ES6 [6]. Pro pomocné komponenty, jako vyhledávač a vybírač data, jsem také využil dodatečné knihovny.

V této kapitole bude postupně vysvětlen celý proces implementace výsledného systému. Na začátku bude detailně popsána struktura projektu. Pak se krátce vysvětlí implementace databázové vrstvy a kontroléru aplikace. Nejpodrobněji bude rozebrána klientská část, která zaujímá většinu systému. Budou zde vysvětleny aspekty implementace jednotlivých komponent a jejich účel. Na ukázkách zdrojových kódu se pak předvedou zajímavé části týkající se samotného generování rozvrhu. Na závěr bude popsán proces nasazení aplikace a její výsledná podoba.

## 5.1 Struktura projektu

Vyvíjená aplikace je webová a proto je uspořádána do klasické struktury aplikací podobného typu. Odlišením je to, že serverová a databázová vrstvy mají zde spíše symbolický charakter. Jejich účelem je pouze načítání dat z databázového souboru a přeposlání těchto dat do klientské části, kde proběhne samotné vykreslení. Absolutní většina aplikace je tím pádem soustředěna ve složce `/public/js`. A přesto celý projekt je navržen tak, aby se eventuálně dal rozšiřovat. Struktura projektu je následující:

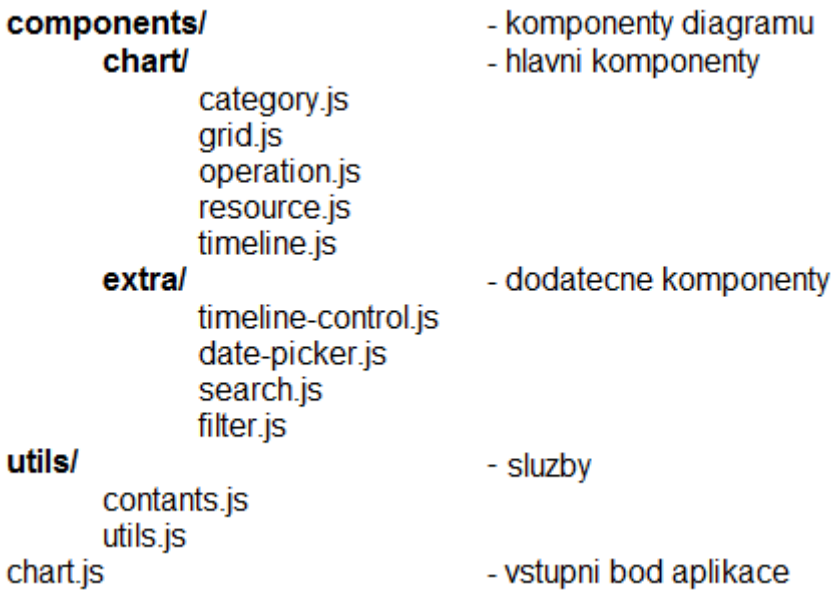
- `database` – Adresář, který obsahuje zdrojové kódy nutné pro načtení dat z databáze.
- `public` – Složka, která je veřejně přístupná klientovi. Zpravidla obsahuje vstupní stránku `Index.html` do webové aplikace. Pak jsou v ní obsaženy různé statické soubory, jako obrázky, CSS styly a zdrojové kódy v jazyce JavaScript, které jsou pak vykonávány na straně klienta.
  - `css` – Adresář s definovanými styly pro vnější vzhled aplikace.
  - `img` – Složka s obrázky.
  - `js` – Složka se zdrojovými kódy v jazyce JavaScript, které slouží pro dynamické generování obsahu stránek. V mém projektu je to centrální adresář, který ob-

sahuje všechny potřebné moduly pro generování rozvrhu. Jeho obsah je popsán dále.

- `views` – Tento adresář obsahuje stránky v jazyce HTML, prip. dalších odvozených formátech, které tvoří uživatelské rozhraní webové aplikace.
- `app.js` – Tento soubor je vstupním bodem aplikace. Importuje potřebné knihovny, nastavuje cesty ke složkám, definuje vstupní kontroléry pro běh aplikace a spouští ji.
- `package.json` – Soubor slouží pro správu balíčku a jejich závislosti systémem NPM. Obsahuje také skripty pro sestavení a běh aplikace a jiná metadata.
- `package-lock.json` – Soubor, který definuje strom závislosti verzí jednotlivých balíčků. Je generován automaticky.

Ústředním místem projektu je složka `/public/js`. Obsahuje skripty sloužící pro vykreslení diagramu výrobního rozvrhu. Skripty jsou definovány jako JavaScript moduly, které dohromady tvoří funkční celek. Jeho vstupním bodem je soubor `chart.js`. Struktura této samostatné aplikace je znázorněna na obrázku 5.1. Jednotlivé moduly budou vysvětleny v následující podkapitole.

Obrázek 5.1: Struktura modularizované aplikace pro vykreslení rozvrhů v jazyce JavaScript




## 5.2 Aplikační vrstva a interakce s databází

Jak již bylo řečeno, datová a aplikační vrstvy zaujímají malou část celé webové aplikace – slouží pouze pro načtení dat, jejich lehkou transformaci a následující přeposlání techto dat skriptu `chart.js`, který pak vykoná tu nejdůležitější práci.

### Databázový konektor

Pro načítání dat z databázového souboru je vytvořena přístupová vrstva v podobě objektu `DBConnector`. Tento objekt se připojí k databázi pomoci NPM modulu `sqlite3` [3]. `DBConnector` definuje dvě nezbytné metody: `query()` a `formát(data)`:

- metoda `query()` provede asynchronní načtení vstupních dat. Všechna potřebná data je možné získat pomocé jednoho SQL dotazu.
- metoda `format(data)` upraví načtená data do následující podoby 4.5. Pro každou operaci v tabulce `PlannedTimeWindow` je vyhrazen jeden řádek. Proto se musí provést formátování, které seskupí operace provedené na jednom zdoji a přiřadí je tomuto zdroji.

### Kontrolér aplikační vrstvy

Všechna serverová aplikační logika je řízená z jednoho souboru `app.js`, který je také vstupním bodem celého webového systému. Serverová vrstva pro svůj běh využívá rámec `Express`. V `app.js` je proto importován odpovídající modul, který zprovozní aplikaci. Dále jsou zde nastaveny cesty k potřebným adresářům, definovány základní kontroléry a také importován objekt `DBConnector`.

Po kliknutí na tlačítko „Draw“ (vykreslení diagramu) umístěné na hlavní strance se metodou `Ajax` zavolá odpovídající kontrolér, který pomocí metody `query()` vyčte data z databázi. Z těchto dat se následně inicializuje objekt `Chart` odpovídající za vykreslení rozvrhu.

Protože klientská část aplikace se skládá z modulů v jazyce JavaScript, cílový soubor `chart.js` musí být importován do skriptového „tagu“ cílové HTML stránky s označeným příslušným typem: `⟨script type="module"⟩`.

## 5.3 Komponenty klientské strany

Klientská strana aplikace představuje logicky oddělený program se strukturou znázorněnou na obrázku 5.1. Celý program pro generování rozvrhů je rozdělen do jednotlivých komponent, tzv. modulů, které dohromady tvoří modularizovanou objektově orientovanou aplikaci [9]. V následujícím textu pojem aplikace bude označovat výše popsaný program, tudíž ne celou webovou aplikaci, ale pouze její klientskou část.

### Životní cyklus komponenty

Každá z komponent, jak hlavní tak i dodatečné, během vykreslování diagramu prochází obdobným životním cyklem. Cyklus zahrnuje následující etapy (metody komponenty):

- **`set_defaults()`** – V této metodě jsou komponentou nastaveny implicitní hodnoty předané její konstruktoru a inicializovány potřebné proměnné.
- **`prepare()`** – Zde jsou provedeny potřebné přípravy předcházející samotnému vykreslování komponenty. Pro operaci to bude znamenat výpočet hodnot její umístění v diagramu, pro ovladač časové osy – přípravu seznamu zobrazených dat.

- `draw()` – Tato medota provádí samotné vykreslování. Vygeneruje se potřebný HTML/SVG element, nastaví se jeho atributy a element se umístí do vnitřní struktury odpovídajícího rodičovského elementu.
- `bind()` – Tady komponenta provede navázání určitých posluchačů na odpovídající události – zobrazení/shovani kategorie, selekce zdroje, zobrazení informačního okna atd.

### Základní třídy

Každá komponenta je tvořena příslušnou třídou umístěnou ve samostatném souboru s odpovídajícím názvem. Hlavním objektem, který inicializuje všechny komponenty, je objekt třídy `Chart`. Při inicializaci komponenty tento objekt je vždycky předán jako hlavní parametr. Další parametry nejsou povinné.

Dále jsou podrobně popsány jednotlivé třídy a moduly aplikace, jejich účel a důležité rysy implementace. **Tučným** písmem jsou zvýrazněny základní komponenty tvořící obsah diagramu. Podtržené jsou pak dodatečné elementy diagramu sloužící pro výběr data, vyhledávání, filtrování. Klasickým písmem jsou nakonec označeny pomocné třídy.

- **Chart** – Hlavní třída aplikace. Inicializuje cílový kontejner pro diagram a nastavuje počáteční konfigurace (výšku a šířku jednotlivých polí, odstupy, režim časové osy). Připravuje data pro časovou osu diagramu. Vykresluje vrstvy-kontejnery pro dílčí elementy diagramu.

  Ze vstupních zdrojových dat inicializuje všechny základní komponenty systému. Také provádí inicializaci dodatečných elementů diagramu (ovladač časově osy, vyhledávač atd.). Nakonec volá metody vykreslování všech komponent.
- **Category** – Třída slouží pro rozlišení typů jednotlivých zdrojů. Vykresluje základní element kategorie a příslušná tlačítka pro manipulaci se zdrojí této kategorie (jejich zobrazení, schování, selekce).
- **Resource** – Třída označuje výrobní zdroj (stroj, nástroj, personál atd.). Uchovává vstupní data o zdroji. Na jejich základě vykresluje odpovídající element v diagramu a spravuje reakce na události tohoto elementu (selekce).
- **Operation** – Touto třídou se označuje výrobní aktivita neboli operace prováděna na konkrétním zdroji. Na základě vstupních dat vypočítává koordináty umístění operace v diagramu. Vykresluje element operace a reaguje na události s ním spojené (zobrazení okna s výpisem informací).
- **Grid** – Třída se zabývá vykreslením všech vrstev SVG kontejnerů, ve kterých se pak nacházejí operace. Vykresluje základní pozadí, jednotlivé řádky a sloupce tvořící výslednou síť pole pro operace, sekce pro operace různých typů a mezery mezi těmito sekcemi.
- **Timeline** – Tato třída odpovídá za hlavičku diagramu s časovou osou. Připravuje příslušnou textovou podobu času a data v zavisloti na zvoleném režimu osy. Následně vykresluje elementy s připraveným obsahem do hlavičky diagramu.
- TimelineControl – Třída má pod kontrolou ovládací tlačítka časové osy. Vykresluje tyto tlačítka a definuje příslušné posluchače událostí.

- Search – Tato třída má na starosti proces vyhledávání určitého zdroje nebo operace. Inicializuje cílový element pomocí externí knihovny. Funguje jako našeptavač. Po úspěšném nalezení provede vycentrování diagramu na příslušné místo.
- DatePicker – Tato třída je zodpovědná za inicializaci elementu pro výběr data. Třída spolupracuje s externí knihovnou.
- Filter – Třída slouží pro inicializaci elementu pro výběr zobrazovaných typů operací. Umožňuje vicenasoby výběr. Také využívá externí knihovny.
- Utils – Pomocná třída, která obsahuje statické metody využívané při vykreslování diagramu. Metody dovolují různě upravovat a transformovat čas a datum, dynamicky vykreslovat elementy, prip. další možnosti.
- Constants – Modul, který definuje základní konstanty.

### Definování cílových elementů

Pro správnou inicializaci komponent ještě před zavoláním skriptu pro vykreslení diagramu je potřeba ve výchozím HTML souboru vydefinovat cílové elementy, tzv. „targets“. Tyto elementy definují jednotlivá umístění komponent systému (kde se rozmístí diagram, kde bude vyhledávač zdrojů atd.). Elementy se pošlou jako parametry do kostruktoru objektu `Chart`. Při inicializaci dílčích komponent se cílové elementy přepošlou dále.

Takový způsob inicializace hodně zvyšuje flexibilitu konečného systému. Dovoluje totiž různě přemísťovat cílové elementy a upravovat jejich vzhled nastavením parametrů elementů-kontejnerů.

## 5.4 Důležité části implementace

Tato podkapitola předvádí ukázky některých podstatných částí zdrojových kódů aplikace. Jsou zde popsány způsob vykreslení elementů, příprava dat pro časovou osu, obnovení stavu zdrojů a operací, inicializace komponenty a navázání posluchačů událostí.

### Dynamické generování

Základním principem dynamického generování rozvrhu je postupné vykreslování jeho jednotlivých elementů. Před vykreslováním je potřeba vygenerovat odpovídající element. K tomu slouží statické metody `create_html()` a `create_svg()` třídy `Utils`. Pro generování je potřeba definovat značku elementu, jeho atributy a rodičovský element. Implementace metody 5.1 je inspirována projektem **Frappe Gantt**[1] [2]

```
static create_html(tag, attrs, prepend) {
    const elem = document.createElement(tag);
    for (let attr in attrs) {
        if (attr === 'parent') {
            const parent = attrs[attr];
            prepend ? parent.prepend(elem) : parent.appendChild(elem);
        } else if (attr === 'innerHTML') {
            elem.innerHTML = attrs[attr];
```

[1] https://frappe.io/gantt

```
        } else {
            elem.setAttribute(attr, attrs[attr]);
        }
    }
    return elem;
}
```

Výpis 5.1: Dynamické generování HTML elementu

### Vytváření časové osy

Před každým vykreslováním diagramu je potřeba znovu přepočítat celou časovou osu, připravit texty, které budou tvořit její obsah, a následně osu vykreslit. Příčinou zcela nového vytváření časové osy při každém překreslování je samotná změna její režimu a také změna hraničních hodnot. Tyto dvě skutečnosti vždycky ovlivňují výsledný vnější vzhled časové osy.

Změna režimu vede k tomu, že na časové ose budou umístěny jiné hodnoty. Například v režimu „Hodina“ v horní části časové osy se bude nacházet den a měsíc, v dolní pak jednotlivé hodiny. Pro režim „Den“ je to následující – v horní části bude zobrazen měsíc, v dolní se vypisou dny tohoto měsíce. Druhým faktorem, který ovlivňuje vzhled časové osy, je změna její hraničních hodnot. Toto nastává při zobrazení/shovani jednotlivých zdrojů anebo při vybírání typů operací určených k zobrazení. Obě dvě věci vedou ke změně výsledné šířky časové osy.

Na začátku je připraven seznam dat pro časovou osu. Tento seznam je ovlivněn hraničními hodnotami a zvoleným režimem. Další dvě metody 5.2 a 5.3 jsou také inspirovány projektem **Frappe Gantt**.

```
const resources = this.resources.filter_hidden_resources();
for (let resource of resources) {
    for (let operation of resource.operations) {
        if (!this.chart_start || operation.time_start < this.chart_start) {
            this.chart_start = operation.time_start;
        }
        if (!this.chart_end || operation.time_end > this.chart_end) {
            this.chart_end = operation.time_end;
        }
  }
}
```

Výpis 5.2: Nalezení hraničních hodnot časové osy

```
while (current_date < this.chart_end) {
    if (this.options.view_mode == VIEW_MODE.HOUR) {
        current_date = Utils.inc_date(current_date, 1, 'hour');
    } else if (this.options.view_mode == VIEW_MODE.QUARTER_DAY) {
        current_date = Utils.inc_date(current_date, 6, 'hour');
    } ...
}
this.chart_dates.push(current_date);
```

Výpis 5.3: Nalezení seznamu dat časové osy v závislosti na zvoleném režimu

Z nalezeného seznamu dat se následně připraví jednotlivé texty pro časovou osu. Časová osa, jak již bylo řečeno, je rozdělena na horní a dolní část. Horní vždycky představuje větší časovou jednotku, dolní část je pak složená z menších jednotek (například horní – den, dolní – hodina). S ohledem na danou skutečnost a zvolený režim časové osy během etapy přípravy objektu `Timeline` se naleznou odpovídající texty, které se následně vypíší do časové osy diagramu.

### Obnovení stavu zdrojů

Důležitým faktorem každého zdroje, který ovlivňuje budoucí vykreslování diagramu, je stav, ve kterém se tento zdroj právě nachází. Zdroje jsou roztříděny do různých kategorií podle jejich typu. Zároveň každý zdroj je určen dvěma stavy, ve kterých se může nacházet. První stav určuje, jestli zdroj je vybrán, druhý – jestli je schován. Zdroje jsou také označeny indexem, který určuje jejich pořadí v kategorii. Pokud stroj je schován, má nastavený záporný index. Tento index se používá při výpočtu umístění odpovídajících operací.

Stav „je vybrán“ – `is_selected` – znamená, že při následném vykreslování diagramu zdroj a jeho operace budou zobrazeny. Stav „je schován“ – extttis_hidden – také znamená, jestli zdroj a jeho operace budou zobrazeny, neoznačuje ale totéž. Mezi těmito stavy existuje rozdíl.

Zdroj, který se nachází ve stavu `is_selected` označuje pouze skutečnost, že v daném okamžiku je zaškrtnut v panelu zdrojů. Hned po zavolání metody překreslování se každý zdroj ve stavu `is_selected == true` také poznačí stavem `is_hidden == false`. Analogicky se označí zbytek zdrojů nacházející se v opačném stavu. Dále proběhne vykreslování.

Na druhou stranu zdroj ve stavu `is_hidden == false` ne vždycky musí být vybrán. Toto nastává v případě, kdy je požadované zobrazit předtím schované zdroje. Po zmáčknutí odpovídajícího tlačítka se schované zdroje zobrazí a přitom nebudou vybrány. Pokud v tomto místě nedojde k jejich vybrání, během následujícího překreslování budou zase schovány. Algoritmus upravující stav jednotlivých zdrojů je znázorněn níže 5.4.

```
update_resources() {
    for (let category of this.categories) {
        let index = 0;
        for (let resource of category.resources) {
            if (category.show_hidden) {
                resource.is_hidden = false;
                resource.index = index++;
            } else {
                resource.is_hidden = !resource.is_selected;
                if (!resource.is_hidden) {
                    resource.index = index++;
                } else {
                    resource.index = -1;
                }
            }
        }
    }
}
```

Výpis 5.4: Algoritmus obnovení stavu zdrojů

### Vykreslení operace

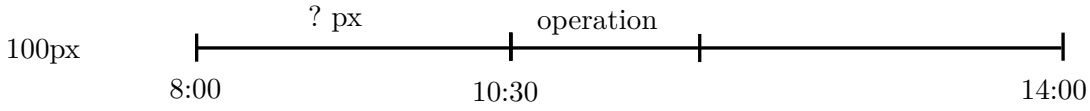
Operace se vykreslují jako SVG elementy do kontejnerů, vztaženého k určitě kategorii podle její typu. Operace je představena `<rect >` elementem. Pro správné vykreslení potřebuje mít nastavené základní atributy: `x`, `y`, `width` a `height`.

Výpočet atributů se provádí následujícím algoritmem 5.5:

```
prepare() {
    ...
    this.x = (this.time_start - this.chart.chart_start) / this.chart.full_time
        * options.grid_width + options.grid_offset * 2;
    this.y = options.row_height * this.resource.index + options.bar_padding / 2;
    this.width = (this.time_end - this.time_start) / this.chart.full_time
        * options.grid_width;
    this.height = options.bar_height;
  ...
}
```

Výpis 5.5: Algoritmus výpočtu atributů `<rect>` elementu odpovídajícího jedné operaci

Atribut `„x"` se vypočítává jako odstup od začátku SVG kontejneru v horizontální rovině. Odstup je úměrný času začátku a konce operace a celkové šířce SVG kontejneru. Atribut `„width"` se vypočte podobným způsobem. Nejlépe to vysvětluje následující obrázek 5.2:




? = (10:30 - 8:00) / (14:00 - 8:00) * 100 = 41.67 px

Obrázek 5.2: Výpočet odstupu operace v pixelech od začátku SVG kontejneru

Atribut `„y"` je analogicky odstupem ve vertikální rovině. Je závislý pouze na šířce jednoho řádku diagramu a na indexu zdroje, ke kterému se váže odpovídající operace. Atribut `„height"` odpovídá implicitně nastavené výšce elementu operace.

### Navázání událostí a komunikace mezi komponentami

Komponenty diagramu reagují na vyskytující se v něm události. Události jsou vždycky vázány na určité HTML nebo SVG elementy. Objekty komponent však představují abstraktní nadstavbu nad těmito elementy. Jako jeden ze svých atributů tyto objekty vždycky mají ukazatel na odpovídající element.

Komunikace tím pádem probíhá na dvou úrovních – jednak mezi HTML/SVG elementy, jednak mezi komponenty. Při obdržení události komponenta vždycky zareaguje nalezením odpovídajícího elementu ve svém vnitřním HTML a provede příslušnou akci na tomto elementu. Příklad navázání události a komunikace mezi komponentami je znázorněn níže 5.6.

```
CATEGORY
bind_select() {
    const category = this;
    const button = this.html.querySelector('.select');
```

```
    button.addEventListener('click', function() {
        ...
        category.resources.forEach(resource => {
            resource.toggle_select();
        });
        ...
    });
}

RESOURCE
toggle_select() {
    this.is_selected = !this.is_selected;
    const elem = this.html.querySelector('.res-selector');
    elem.checked = !elem.checked;
}
```

Výpis 5.6: Navázání události a komunikace mezi komponentami

### Pomocné elementy

Pomocnými elementy aplikace jsou ovladač časové osy, vyhledávač zdrojů a operací, vybírač data a filtr operací. Tyto elementy nejsou součástí dynamický generovaného diagramu. Jsou inicializovány a vykresleny zvlášť, po generování samotného rozvrhu.

První element – ovladač časové osy – představuje sadu dynamický vykreslených tlačítek. Je generován stejným způsobem jako diagram a je oddělen od něho pouze logicky. Na každém z tlačítek ovladače časové osy je navázána událost zodpovědná za výběr příslužného režimu zobrazení. Výběr režimu zavolá přepočet zdrojových dat a pak samotné překreslení diagramu.

Zbylé tři elementy jsou v aplikaci vytvořeny pomocí externích veřejně dostupných knihoven. Všechny tyto knihovny lze nainstalovat jako NPM balíčky. Bohužel, pro vývoj webových aplikací určených k použití v prohlížeči, NPM ve svém výchozím stavu neumožňuje jednoduché a pohodlné importovaní knihoven. Je nezbytné instalovat další nástroje jako Webpack[2], ktere provedou správné importovaní, vytvoří nutné závislosti a ubalí celý sytém do funkčního celku. Další alternativou je vložit do HTML stránky skript element s odkazem na knihovnu do CDN[3] repozitáře, který pak automaticky dodá veškeré soubory nutné pro běh programu. Díky své jednoduchosti jsem zvolil tuto alternativu.

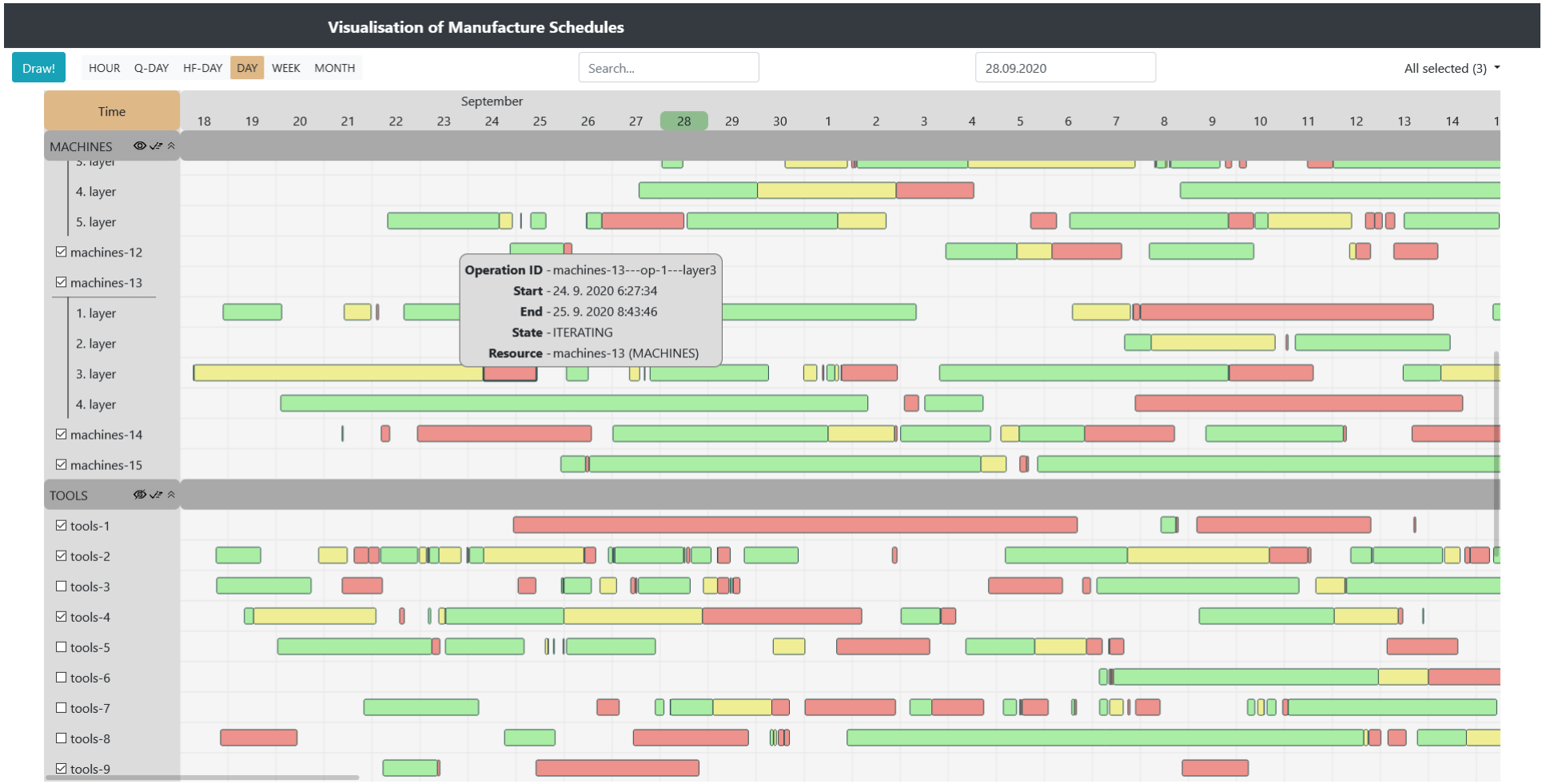
Pro vyhledávač zdrojů a operací je využita knihovna **@tarekraafat/autocomplete.js**. Element pro výběr data je implementován pomoci knihovny **vanillajs-datepicker**. Pro filtr operací je zvolena knihovna **bootstrap-multiselect**. Pro svůj běh pomocné elementy aplikace vnitřně využívají tyto knihovny. Zvenku jsou pak obaleny do samostatných objektů pro jednodušší komunikaci se zbytkem systému. Kařdý ze zmíněných elementů aplikace během své inicializace provede přípravu vstupních dat, těmito daty inicializuje cílový HTML element a případně naváže potřebné posluchače událostí.

[2] https://webpack.js.org/
[3] https://en.wikipedia.org/wiki/Content_delivery_network

## 5.5 Výsledná podoba aplikace

Na obrázku 5.3 je znázorněna výsledná podoba aplikace. V její horní části na levé straně je umístěno tlačítko „Draw“ (č.1), zodpovědné za vykreslení diagramu. Na právé straně od něho se nacházejí tlačítka pro ovládání časové osy (č.2). Tyto tlačítka také zavolají překreslení rozvrhu a navíc k tomu změní režim zobrazení. Dále je umístěn element sloužící pro výběr data (č.3). Po jeho zvolení se digram vycentruje na příslušné místo. Hned za vybíračem data je umístěn vyhledávač zdrojů a operací (č.4). Podobně jako při výběru data, po nalezení zdroje nebo operace se diagram vycentruje na odpovídající místo. Jako poslední element v horní části aplikace je umístěn filtr typů operací (č.5). Dovoluje možnost vícenásobného výběru. Při následném překreslení diagramu se zobrazí pouze operace vybraného typu.

Pod ovládacími elementy aplikace je zobrazen samotný výrobní rozvrh v podobě Ganttova diagramu. Skládá se ze tří hlavních částí: hlavička s časovou osou (č.6), panel zdrojů (č.7), pole s naplánovanými operacemi (č.8). Časová osa a panel zdrojů jsou upevněny tak, aby se při skrolovani diagramu neztrácel kontext [11]. To znamená, že při pohybu v horizontální rovině časová osa je pohyblivá a panel zdrojů upevněn. Ve vertikální rovině to pak funguje opačně. Na panelu zdrojů jsou zobrazeny kategorie (č.9). Každá z nich obsahuje tři tlačítka. První tlačítko zobrazí schované zdroje a znovu překreslí diagram. Druhé provede výběr zároveň všech zdrojů v kategorii anebo zruší tento výběr. Poslední třetí tlačítko buď schová anebo zobrazí celou kategorii. U každého zdroje v kategorii je také umístěno tlačítko pro jeho selekci. V hlavním poli diagramu jsou vykresleny jednotlivé operace obarvené podle jejich typu. Po kliknutí na operaci se zobrazí okno s dodatečnými informacemi (č.10).




Obrázek 5.3: Výsledná podoba aplikace

# Kapitola 6

# Testování a vyhodnocení výsledků

Neoddělitelnou součástí vývoje jakékoliv aplikace je její testování. Protože tato aplikace patří mezi vizualizační nástroje, cílem kterých je uživatelsky vhodná prezentace zdrojových dat, testování především bude zaměřeno na vzhledovou stranu aplikace. S testováním vzhledové části aplikace je úzce spojeno testování jejího správného chování. Obě tyto činnosti zahrnují provádění různorodých akcí dovolených v aplikaci s cílem dosažení co největšího počtu možných stavů programu, přitom aby každý z těchto stavů odpovídal očekáváním uživatele a byl v souladu s reprezentovanými daty.

Tato kapitola pojednává o procesu testování výsledné aplikace. Pro úspěšně otestování je potřeba vygenerovat množinu výrobních rozvrhu s různými vstupními parametry. Tyto rozvrhy je zatím nutné vykreslit pomocí aplikace a manuálně je „osahat“. Testování se bude zaměřovat na jednotlivé funkcionální části aplikace. Především se budou vyhodnocovat výsledky zadávání hraničních hodnot. V kapitole se podrobně vysvětlí postup generování výrobních rozvrhů, popíše se formát jejich vstupních konfiguračních souboru a následné zpracování. Potom se provede vizualizace vygenerovaného příkladového rozvrhu a na ní se ukáže, jakým způsobem probíhá testování vzhledu a chování aplikace.

## 6.1 Generování testovacích rozvrhu

Jak již bylo popsáno v kapitolách zabývajících se návrhem a implementací systému, veškerá potřebná pro vizualizaci data jsou obsažena ve dvou databázových tabulkách. Základem testovacího procesu je tudíž vytvoření databázového souboru obsahujícího tyto dvě tabulky a jeho naplnění správně vygenerovanými daty. Automatizace výše popsaných aktivit umožní rychle a pohodlně měnit vstupní data, znovu generovat rozvrh rozvrh a zkoumat chování diagramu za určitých podmínek.

Pro účely generování testovacích rozvrhů je proto vytvořena sada skriptu nacházející se ve složce `/tests`. Je tvořena třemi logicky oddělenými částmi. Nejdůležitější a největší z nich má na starosti samotné generování databázového souboru a jeho naplnění testovacími daty (soubor `generate.js`). Druhá v podobě konfiguračních souboru ve formátu JSON (složka `/configs`) slouží pro nastavení vstupních podmínek a omezení, které musí splňovat vygenerovaná data. Třetí část je zodpovědná za mazání vygenerovaných dat a souborů (soubory `drop.js` a `clean.js`).

### Formát konfiguračního souboru

Konfigurační soubor je uložen ve formátu JSON a je tvořen seznamem elementů, každý ze kterých popisuje jeden databázový soubor. Základními polozkami elementů je název databáze a data určující omezení, která musí být splněna budoucím vygenerovaným rozvrhem.

Mezi omezení, která je možné nastavit před generováním, patří čas začátku a konce výrobního rozvrhu, typ jednotlivých kategorií zdrojů, počet zdrojů v každé kategorii, rozmezí počtu operací pro jeden zdroj (nedefinuje se přesným číslem, aby byla zajištěna náhodnost dat při generování), a nakonec parametr, který určí, jestli zdroje v kategorii budou mít násobnou kapacitu (jejich počet je také náhodný). Příklad konfiguračního souboru je znázorněn níže 6.1.

```
{
  "name": "db-1.db",
  "data": {
    "date_start": "2020-11-30",
    "date_end": "2021-02-15",
    "categories": [
      {
        "type": "machines",
        "details": {
          "res_number": 10,
          "ops_range": [10, 30],
          "layers": true
        }
      },
      {
        "type": "tools",
        "details": {
          "res_number": 10,
          "ops_range": [10, 50],
          "layers": false
        }
      }
    ]
  }
}
```

Výpis 6.1: Format konfiguracniho souboru pro generovani testovacich rovrhu

### Příprava a generování testovacích dat

Veškerá logika generování testovacích rozvrhů je obsažena ve skriptu `generate.js`. Proces generování se skládá ze dvou části. Za prvé se vytvoří databázový soubor, ve kterém se vydefinuji databázové tabulky `PlannedTimeWindow` a `Resources`. Jednotlivé atributy a jejich typy jsou popsány v kapitole zabývající se návrhem aplikace. Dále s ohledem na omezení nastavená v konfiguračním souboru se vygeneruje množina fázových operací (je vhodné připomenout, že jeden řádek tabulky `PlannedTimeWindow` obsahuje informace o jedné fázové operací).

Proces přípravy a generování dat pro fázové operace využívá náhodných jevů s cílem zajistit přirozenější rozložení vygenerovaných dat. U zdrojů jednoho typu platí, že pokud v konfiguračním souboru příznak „layers“ je nastaven na „true“, tak při generování dat každý zdroj z této kategorie s předem nastavenou pravděpodobností bude mít vícenásobnou kapacitu. Počet vrstev (konkrétně 1 až 5) u zdrojů takového typu se také určí náhodným jevem.

Náhodnost u operací je zajištěna jejich počtem (náhodně se vybere z rozmezí definovaného v konfiguračním souboru) a rozložením na časové ose. Pro zvolený počet operací se vygeneruje seznam náhodných časů. Ty budou odpovídat začátku každé plné operace. Pak pro každou operaci se náhodně vybere počet jejich dílčích fází (opět v rozmezí 1 až 5). Jednotlivé časy dílčích fázových operací se zase vygeneruji náhodně. Přitom se dodržuje omezení, že pro jeden zdroj (v případě vícenásobného zdroje pro jeho jednu vrstvu) jednotlivé fáze operace se nesmí překrývat a musí být ukončeny před začátkem další plné operace.

ID jednotlivých zdrojů a operací je generováno v závislosti na jejich umístění v diagramu. Toto ulehčí následnou kontrolu spavnosti vizualizovaných dat. Například jedenáctý zdroj typu „tools“ bude mít ID „tools-11“. Jeho šestá operace prováděna na třetí vrstvě bude mít ID „tools-11—op-3—layer-3“.

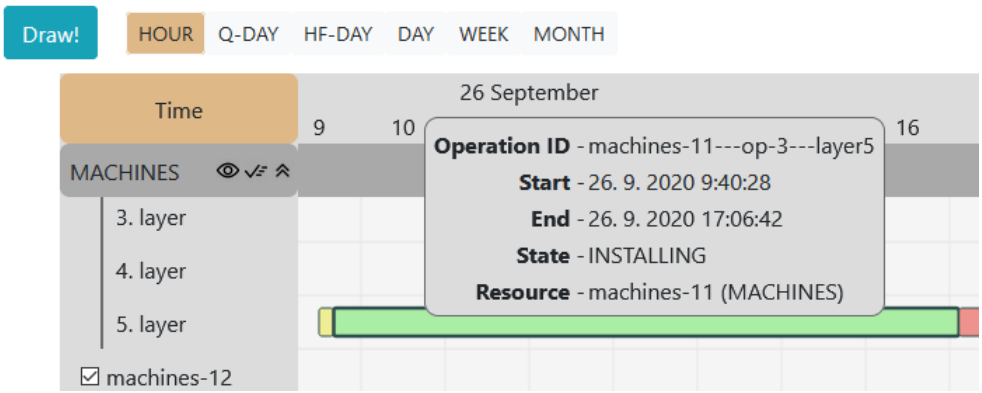
## 6.2 Testování vzhledu a chování aplikace

Pro správné otestování vzhledu a chování aplikace je nutné vygenerovat množinu různorodých testovacích rozvrhů a pro každý z nich zkontrolovat co nejvíce možných stavů. V této podkapitole na jednom příkladu testovacího rozvrhu bude popsáno, na která místa byl kladen důraz během testování.

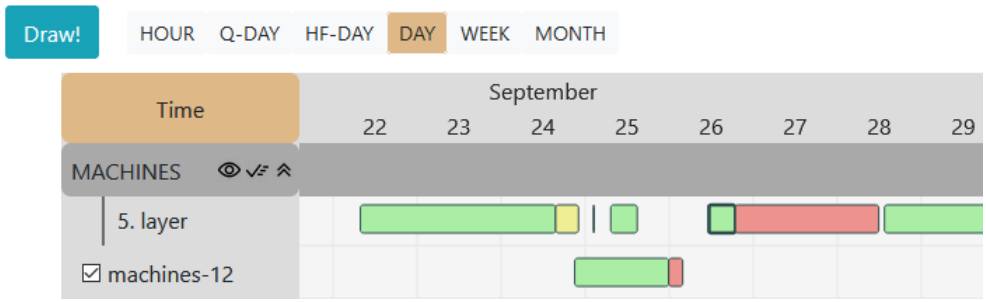
### Režimy časově osy a vykreslení operaci

Za prvé je nutné otestovat časovou osu. Toto místo je pro Ganntův diagram nejdůležitější. V každém režimu časové osy je otestováno, jestli čas začátku a konce určitých operací se shoduje s tím, jak jsou tyto operace vykresleny vzhledem k časové ose. Na obrázcích 6.1 a 6.2 je zobrazena stejná operace ve dvou časových režimech – hodina a den. Její začátek a konec odpovídají dělením časové osy diagramu.

Obrázek 6.1: Vykreslení operací v režimu časové osy „Hodina“



Obrázek 6.2: Vykreslení operací v režimu časové osy „Den“

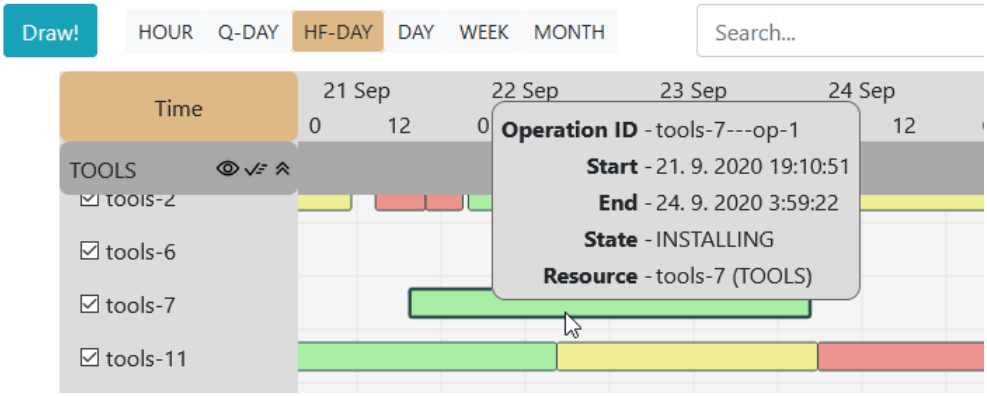



### Zdroje, jejich vrstvy a operace

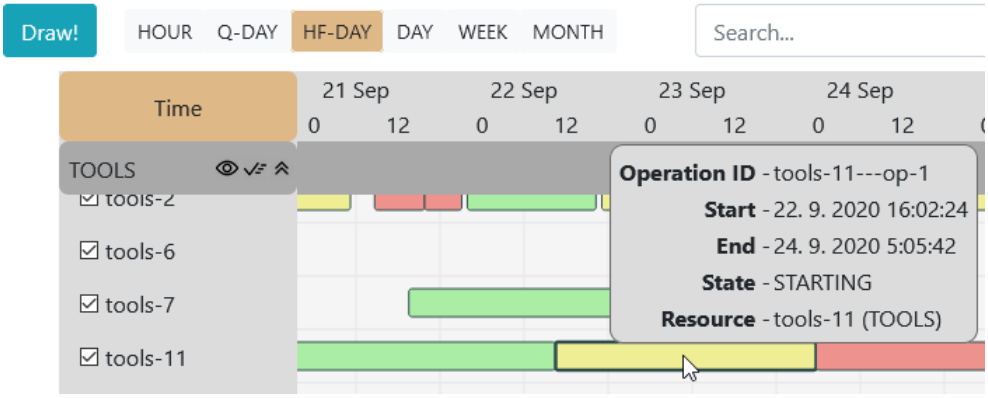
V této části testování je nutné se zaměřit na vykreslení jednotlivých zdrojů, jejich vrstev a odpovídajících operací. Je kontrolováno, jestli počet vykreslených zdrojů odpovídá číslu zdrojů definovanému pro tuto kategorii v konfiguračním souboru. Jestli při schovaní několika zdrojů se správné zobrazí zbytek kategorie. Jestli se správně vykreslují jednotlivé vrstvy. Jestli se korektně provádí selekce zdrojů. Jestli okno s informacemi (id zdroje, id operace, fáze operace atd.) skutečně odpovídá tomu, co je vykresleno.

Na obrázcích 6.3, 6.4 a 6.5 je vidět, že zdroje `tools-8`, `tools-9` a `tools-10` jsou schovány. Přitom indexování zbylých zdrojů je provedeno správně – informace ve vyskakovacím okně, **Operation ID** a **Resource**, odpovídají skutečnosti. Také je vidět, že je správně označen typ fázové operace (zelená – INSTALACE, žlutá – ROZJEZD).

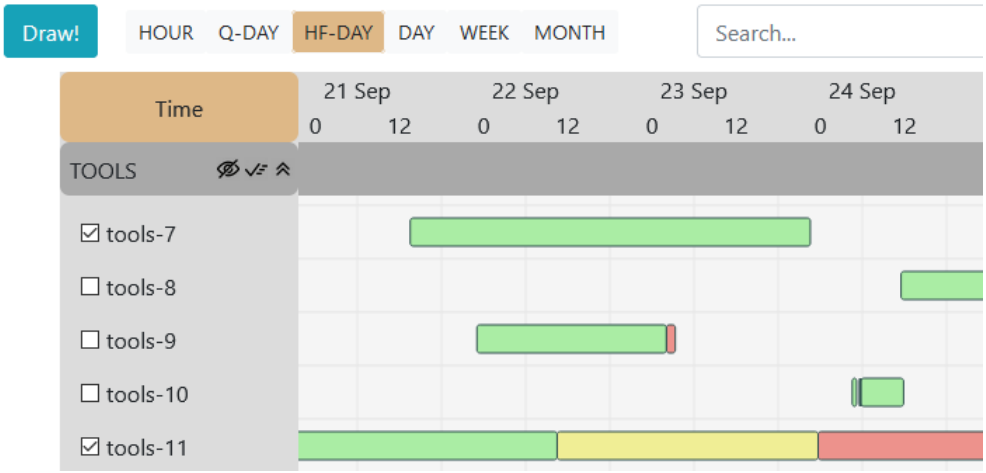
Obrázek 6.3: Informační okno pro operaci typu „Instalace“ zdroje „tools-7“



Obrázek 6.4: Informační okno pro operaci typu „Rozjezd“ zdroje „tools-11“



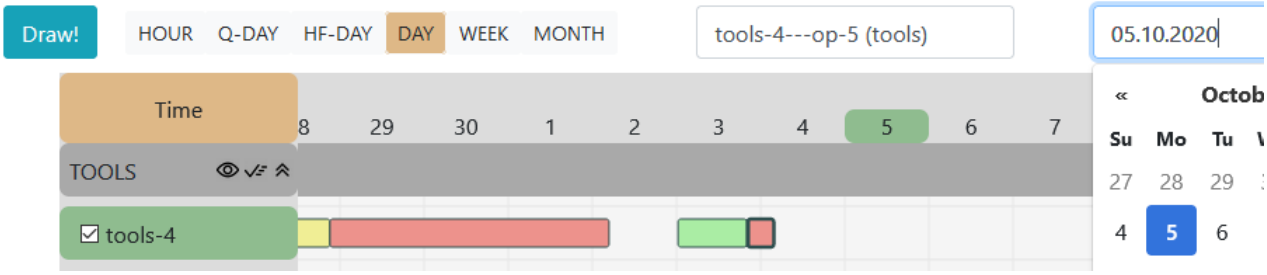

Obrázek 6.5: Zobrazení minule schovaných zdrojů „tools-8“, „tools-9“ a „tools-10“




### Vyhledávač, výběr data, filtr operaci

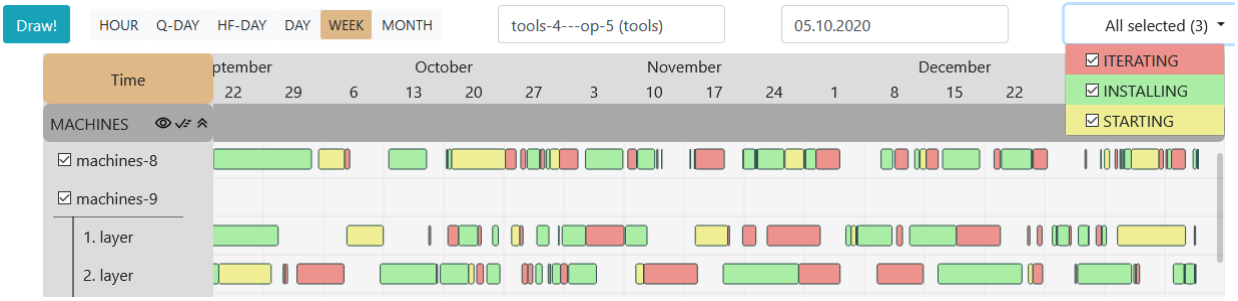
Testování vyhledávače a nástroje pro výběr data se provádí jednoduše. Vyhledávač dovoluje vyhledávat jak zdroje, tak i operace. Po vyhledání je nutné pouze ověřit informace ve vyskakovacím okně. Výběr data zabarví příslušné pole na časové ose. Ukázka je na obrázku 6.6. Nalezené operace a datum jsou zvýrazněny.

Obrázek 6.6: Ukázka nalezených operace a data. Na diagramu jsou zvýrazněny

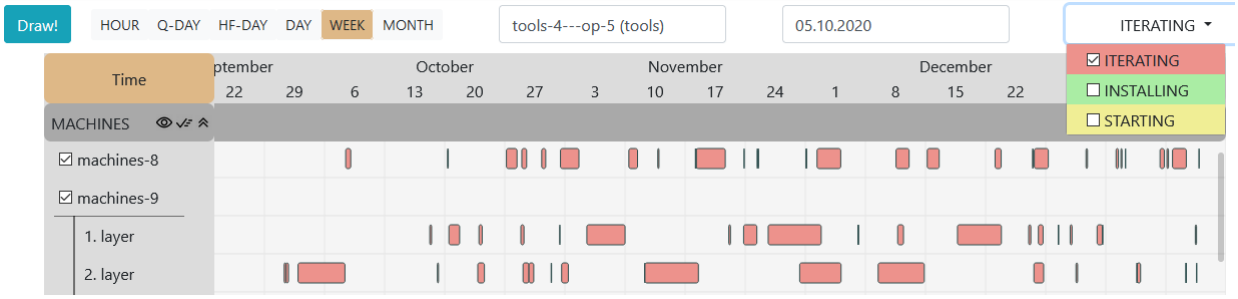



Naposledy je nutné otestovat, jestli správně funguje filtrování typů fázových operací určených k zobrazení. Výsledek filtrování je znázorněn na obrázcích 6.7 a 6.8.

Obrázek 6.7: Jsou vybrány všechny tři typy operací




Obrázek 6.8: Je vybrán pouze jeden typ operace




## 6.3 Výsledky testování

Během testování bylo odhaleno některé množství chyb. Nejvíce složitými pro opravu byly chyby vztažené ke přepočtu umístění elementů v diagramu. Pro zdroje se muselo kontrolovat správné indexování v rámci kategorie. Pro operace byl důležitý výpočet správných kooordinat. Nejvíce zajímavou chybou, která mi zabrala chvilku času, bylo špatné umístění operací na přelomu 24. a 25. října. Na levé straně všechno bylo v pořádku, na právé pak byl posun o jedničku. Ne hned jsem si uvědomil, že toto bylo způsobeno změnou letního času na zimní a potřebovalo dodatečnou kontrolu. Tuto chybu jsem si zatím rozhodl ponechat.

# Kapitola 7

# Závěr

Po uplynutí více než půlroku od začátku studia zadání a následného vývoje aplikace mohu s jistotou říct, že práce je završena. Aplikace pro vizualizaci výrobních rozvrhů je plně funkční a splňuje všechny počáteční požadavky. Má přívětivé uživatelské rozhraní, je jednoduchá k porozumění, dokáže rychle vykreslit obrovské množství dát, je lehce dostupná na webu, nabízí vhodné prostředky pro pohodlné zkoumání a analýzu výrobního rozvrhu a navíc je škálovatelná.

Tato práce prošla všemi etapami klasického vývoje webových aplikací. Na začátku jsem se seznámil s problematikou výrobních rozvrhů a jejich současnou vizualizací. Dále následovala nejsložitější a zároveň nejzajímavější část – návrh aplikace. Na této etapě jsem hodně studoval existující řešení a přemýšlel jsem o tom, jak navrhnout aplikaci, která by splňovala vstupní požadavky a přitom byla elegantní a lehce se rozšiřovala. Zde jsem narazil na veřejné dostupnou aplikaci Frappe Gantt, ze které jsem následně převzal základní myšlenky pro návrh své vlastní aplikace. Potom následovalo stadium implementace, kde kromě samotného vývoje bylo také výzvou vytvořit modularizovanou webovou aplikaci v jazyce JavaScript. S několika technologiemi jsem se setkal poprvé v životě a získal jsem tak cenné zkušenosti. Etapa testování pak potvrdila správný přístup k návrhu systému. Odhalené chyby byly bez větších potíží opraveny.

Osobně považují tuto práci za velice přínosnou. I když webové programování pro mě není úplně cizí, stále jsem se setkal se spoustou zajímavých věcí během vývoje aplikace. Získané znalosti určitě uplatním v budoucích projektech.

# Literatura


[1] *JavaScript* [online]. Mozilla Developer Network Web Docs, 2005-2021 [cit. 2021-04-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[2] *Frappe Gantt* [online]. Frappe, 2021 [cit. 2021-04-10]. Dostupné z: https://github.com/frappe/gantt.

[3] *SQLite Node.js* [online]. SQLite Tutorial, 2021 [cit. 2021-04-10]. Dostupné z: https://www.sqlitetutorial.net/.

[4] Framinan, J. M., Leisten, R. a García, R. R. Manufacturing scheduling systems. *An integrated view on Models, Methods and Tools*. Springer. 2014, s. 51–63.

[5] Grider, S. *Node JS: Advanced Concepts* [online]. Udemy, 2021 [cit. 2021-04-15]. Dostupné z: https://www.udemy.com/course/advanced-node-for-developers/.

[6] Haverbeke, M. *Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming*. 3. edition. Wiley, prosinec 2018. ISBN 978-1593279509.

[7] KLUSÁČEK, D. *Plánování úloh v paralelním a distribuovaném prostředí [online]*. Brno, CZ, 2006. [cit. 2021-04-05]. Diplomová práce. Masaryk University, Faculty of Informatics. Vedoucí práce Mgr. Hana Rudová, P. doc. Dostupné z: https://theses.cz/id/4nfm0h/.

[8] Lim, G. *Beginning Node.js, Express & MongoDB Development*. 1. edition. Independently published, 2019. ISBN 978-1078379557.

[9] Orendorff, J. *ES6 In Depth: Modules* [online]. Srpen 2015 [cit. 2021-04-18]. Dostupné z: https://hacks.mozilla.org/2015/08/es6-in-depth-modules/.

[10] Rauch, G. *Smashing Node.js: JavaScript Everywhere*. 2. edition. Wiley, září 2012. ISBN 978-1119962595.

[11] Vinther, D. *Position: stuck; — and a way to fix it* [online]. Únor 2019 [cit. 2021-04-28]. Dostupné z: https://uxdesign.cc/position-stuck-96c9f55d9526.

[12] VÁGNER, L. *Rozvrhování v diskrétní výrobě [online]*. Brno, CZ, 2015. [cit. 2021-04-05]. Diplomová práce. Vysoké učení technické v Brně. Fakulta strojního inženýrství. Ústav výrobních strojů, systémů a robotiky. Vedoucí práce Ing. Simeon Simeonov CSc. doc. Dostupné z: http://hdl.handle.net/11012/40060.