

Webové knihovny pro vykreslování specifických dashboardů pro technologie v dopravě

Bc. Kamil Stokláška

Diplomová práce
2016



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

akademický rok: 2015/2016

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Kamil Stokláška**

Osobní číslo: **A14449**

Studijní program: **N3902 Inženýrská informatika**

Studijní obor: **Informační technologie**

Forma studia: **prezenční**

Téma práce: **Webové knihovny pro vykreslování specifických dashboardů pro technologie v dopravě**

Téma anglicky: **Web Libraries for Drawing Specific Dashboards in Transport Technologies**

Zásady pro vypracování:

1. Prostudujte obecně používané webové knihovny zaměřené na vizualizaci většího množství dat.
2. Porovnejte vlastnosti těchto knihoven a vyhodnoťte jejich použitelnost.
3. Navrhněte univerzální webové rozhraní, které zapouzdří vybranou technologii a bude vykreslovat specifické dashboardy na základě vložených dat.
4. Realizujte navržené rozhraní tak, aby se dalo použít v reálném prostředí s reálnými daty.
5. K projektu vytvořte podrobnou programátorskou dokumentaci ve formě HTML prezentace.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. LUBBERS, Peter, Brian ALBERS a Frank SALIM. HTML5: programujeme moderní webové aplikace. Vyd. 1. Brno: Computer Press, 2011, 304 s. ISBN 978-80-251-3539-6.
2. ŽÁRA, Ondřej. JavaScript: programátorské techniky a webové technologie. 1. vydání. Brno: Computer Press, 2015, 180 stran. ISBN 978-80-251-4573-9.
3. PILGRIM, Mark. HTML5: up and running. 1st ed. Sebastopol, CA: O'Reilly, 2010, xii, 205 p. ISBN 978-0-596-80602-6.
4. FULTON, Steve a Jeff FULTON. HTML5 canvas. Second edition. Farnham: O'Reilly, 2013, xx, 726 pages. ISBN 14-493-3498-9.
5. STEFANOV, Stoyan a Kumar Chetan SHARMA. Object-oriented JavaScript. 2nd ed. Birmingham, UK: Packt Publishing, 2013, viii, 363 p. ISBN 9781849693127.
6. W3C[online]. The World Wide Web Consortium [cit. 2016-01-29]. Dostupné z: <https://www.w3.org/>

Vedoucí diplomové práce:

Ing. Pavel Pokorný, Ph.D.

Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce:

5. února 2016

Termín odevzdání diplomové práce:

20. května 2016

Ve Zlíně dne 5. února 2016



doc. Mgr. Milan Adámek, Ph.D.
děkan



doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu


Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 18. 5. 2016


.....
podpis diplomanta

ABSTRAKT

Cílem této práce je navrhnout JavaScriptové komponenty, které zapouzdří vybranou technologii pro práci s grafy ve webovém prostředí. V teoretické části práce jsou podrobně popsány existující knihovny určené pro vizualizaci dat a porovnány jejich vlastnosti. Součástí teoretické práce je také srovnání dostupných technologií pro tvorbu vizualizací a detailní popis možností a zavedených postupů v této oblasti. Obsahem praktické části je návrh a vlastní implementace komplexní knihovny pro vizualizaci dat, která zapouzdří vybrané technologie a umožní jejich snadnou integraci do webových informačních systémů. Návrh a veškerá realizace je provedena s ohledem na použití výstupu pro reálné technologie a data z oblasti monitorování dopravy.

Klíčová slova: vizualizace dat, JavaScript, HTML5, canvas, dashboard

ABSTRACT

The aim of this thesis is to propose JavaScript components which cover the selected technology of using charts in web environment. In theoretical part are described already existing libraries, which are designated for data visualization. There is also comparison of their properties. The theoretical part also includes comparison of available technologies for creating visualizations and detailed description of the possibilities and established practices in this domain. In practical part is draft and own implementation of complex library for data visualization, which encapsulate selected technologies and provides their integration to web information systems. All drafts and realizations are performed for output usage on real technologies and data from domain of traffic monitoring.

Keywords: data visualization, JavaScript, HTML5, canvas, dashboard

Tímto bych chtěl poděkovat vedoucímu diplomové práce Ing. Pavlu Pokornému, Ph.D. za odborný dohled, ochotu, spolupráci a čas, který mi po dobu realizace této práce věnoval.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

| | |
|---|-----------|
| ÚVOD..... | 9 |
| I TEORETICKÁ ČÁST | 10 |
| 1 VIZUALIZACE DAT | 11 |
| 1.1 GRAFICKÁ REPREZENTACE DAT | 11 |
| 1.1.1 Spojnicový graf | 11 |
| 1.1.2 Sloupcový graf | 12 |
| 1.1.3 Segmentový graf | 13 |
| 1.1.4 Ganttův graf | 14 |
| 1.1.5 Ostatní grafy | 15 |
| 2 VYKRESLOVÁNÍ VE WEBOVÉM PROSTŘEDÍ..... | 16 |
| 2.1 CANVAS | 16 |
| 2.1.1 2D Context | 16 |
| 2.1.2 Transformace..... | 17 |
| 2.1.3 Nastavení výřezu | 17 |
| 2.1.4 Parametry vykreslování..... | 17 |
| 2.1.5 Animace | 18 |
| 2.1.6 Vykreslování | 19 |
| 2.1.7 Výplň..... | 19 |
| 2.1.8 Obrysy a čáry | 20 |
| 2.1.9 Text | 22 |
| 2.1.10 Vykreslování bitmapy | 22 |
| 2.1.11 Nastavení kompozice | 24 |
| 2.1.12 Optimalizace vykreslování..... | 25 |
| 2.2 SVG..... | 26 |
| 3 ZÍSKÁVÁNÍ DAT | 28 |
| 3.1 WEBOVÉ SLUŽBY | 28 |
| 3.2 REST API..... | 29 |
| 4 FORMÁT A STRUKTURA DAT..... | 30 |
| 4.1 JSON | 30 |
| 5 KNIHOVNY PRO VYKRESLOVÁNÍ D DAT | 32 |
| 5.1 GOOGLE CHARTS | 32 |
| 5.2 D3.JS..... | 33 |
| 5.3 FLOT CHARTS..... | 34 |
| 5.4 CHARTJS..... | 35 |
| 5.5 AMCHARTS | 36 |
| 5.6 HIGHCHARTS..... | 37 |
| 5.7 SROVNÁNÍ..... | 38 |
| 6 WEBOVÉ TECHNOLOGIE..... | 40 |
| 6.1 HTML5..... | 40 |
| 6.2 CSS3..... | 40 |
| 6.3 JAVASCRIPT | 41 |
| 6.3.1 Použití na straně klienta | 41 |

| | | |
|-----------|--|-----------|
| 6.3.2 | Použití na straně serveru | 42 |
| 6.3.3 | ECMAScript..... | 42 |
| II | PRAKTICKÁ ČÁST..... | 43 |
| 7 | VYTVOŘENÁ KNIHOVNA..... | 44 |
| 7.1 | STRUKTURA A POPIS TŘÍD..... | 44 |
| 7.2 | TECHNOLOGIE A KOMPATIBILITA | 47 |
| 7.3 | VYTVÁŘENÍ KOMPONENT | 48 |
| 7.4 | PODPOROVANÉ TYPY GRAFŮ | 48 |
| 7.4.1 | Segmentový graf | 49 |
| 7.4.2 | Sloupcový graf | 50 |
| 7.4.3 | Spojnicový graf | 51 |
| 7.5 | ZOBRAZENÍ LEGENDY GRAFU | 53 |
| 7.6 | KONFIGURACE..... | 53 |
| 7.7 | ZPRACOVÁNÍ DAT..... | 53 |
| 7.8 | INTERAKTIVITA | 55 |
| 7.9 | ANIMACE | 57 |
| 7.10 | PŘEKRESLOVÁNÍ | 58 |
| 7.11 | ADAPTIVITA | 59 |
| 7.12 | KONSTRUKCE DASHBOARDU | 60 |
| 8 | HTML DOKUMENTACE | 61 |
| | ZÁVĚR | 62 |
| | SEZNAM POUŽITÉ LITERATURY..... | 63 |
| | SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK..... | 65 |
| | SEZNAM OBRÁZKŮ | 66 |
| | SEZNAM TABULEK..... | 68 |
| | SEZNAM PŘÍLOH..... | 69 |

ÚVOD

Vizualizace dat je nejrychlejší cesta předání informační hodnoty a prostředky pro tyto vizualizace se tak stávají velmi žádaným nástrojem v oblasti zpracování dat. Proces vizualizace umožňuje nahlížet na data zcela novým způsobem a ulehčit přístup k informační hodnotě, která je v datové struktuře ukryta.

Se stoupající hodnotou a množstvím zpracovávaných informací vznikají také stále nové nástroje a technologie, které práci s daty usnadňují. Moderní informační systémy shromažďují velké množství dat, která jsou uložena v různých formátech a jejich zpracování je proto často komplikované. Sběr dat probíhá obvykle formou online komunikace, při které se data ukládají do serverových databází. K těmto záznamům je následně možné snadno přistupovat pomocí serverových aplikací a filtrovat je pro potřeby uživatele. O zobrazení dat samotnému uživateli se pak stará klientská aplikace, která může běžet přímo ve webovém prostředí v podobě interaktivního dashboardu.

Cílem této práce je podrobně prostudovat a porovnat nástroje, které jsou běžně dostupné pro vizualizaci dat ve webovém prostředí. Na základě těchto údajů bude navržena komplexní webová knihovna psaná v jazyce JavaScript, která zapouzdří vybrané technologie a vlastní implementaci metod pro interaktivní datovou vizualizaci. Výsledná knihovna umožní vykreslit data do víceúrovňových grafů a adaptivně měnit jejich vlastnosti na základě vložených parametrů. Návrh a veškerá realizace bude provedena s ohledem na použití výsledné aplikace pro reálné technologie a data z oblasti monitorování dopravy.

I. TEORETICKÁ ČÁST

1 VIZUALIZACE DAT

Vizualizace je proces zkoumání dat a informací po jejich převedení do grafické podoby. Jejím cílem je pochopení zkoumaných jevů a vniknutí do problému. Proto o vizualizaci mluvíme také jako o vizuální analýze dat. Díky grafické interpretaci dat je možné mnohem efektivněji pracovat s velkým množstvím nepřehledných údajů v tabulkách. Základním grafickým výstupem dat jsou grafy, které se mohou vyskytovat v různých podobách v závislosti na informaci, kterou data nesou.

Výhodou vizualizace je lepší porozumění datům, které jsou zpracovávány. Vizualizaci dat je možné velmi dobře využít také proto, že lidská psychika dokáže vytěžit mnohem více informací z vnímaných tvarů a barev než při čtení textu, nebo při prohlížení tabulek. V případě, že je grafická vizualizace interaktivní, může navíc uživatel lépe vystopovat nové souvislosti. [1]

1.1 Grafická reprezentace dat

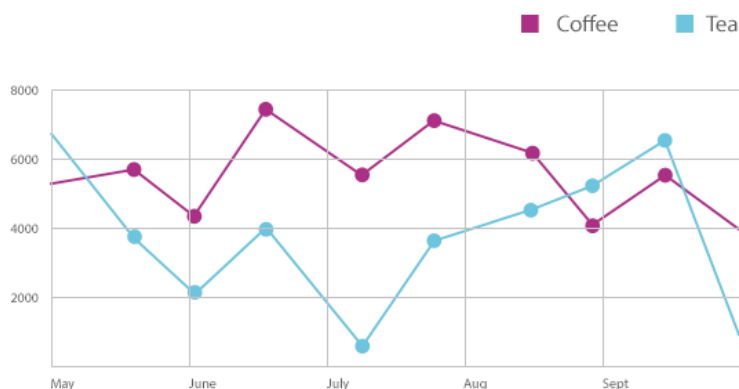
Pro grafickou reprezentaci dat se nejčastěji využívají grafy, které jsou schopny zobrazit specifické souvislosti a rysy vykreslovaných údajů. Výběr vhodného grafu závisí na povaze dat a účelu, za jakým se tato data prezentují. V praxi se nejběžněji využívají tři základní typy grafů – sloupcový, spojnicový a segmentový (koláčový).

1.1.1 Spojnicový graf

Spojnicový graf je nejvíce využívaná grafická reprezentace dat. Vykreslení dat do spojnicového grafu je velmi jednoduché a pro uživatele je snadné se v takto vykreslených datech zorientovat. Organizace a prezentace dat je v tomto grafu vždy jednoznačná, stejně jako vztahy mezi jednotlivými hodnotami. Spojnicové grafy se velmi často využívají ve statistice, kde mohou naznačit vedle historického vývoje sledované veličiny i predikci vývoje budoucího.

Obdobně jako u sloupcového grafu jsou jednotlivé hodnoty do grafu zaneseny jako změny pozice vykreslovaných značek. Na rozdíl od sloupcového grafu jsou zde však kromě změny hodnoty zanášeny i změny směru, díky kterým je možné sledovat vývoj veličiny v čase. [3]

Spojnicové grafy jsou často využívány pro porovnávání dvou a více veličin. Díky změnám ve dvou osách lze z grafu velmi snadno vypozorovat vývoj všech zobrazovaných veličin a vzájemné vztahy mezi nimi. Osa Y představuje zpravidla hodnotu veličiny, zatímco osa X se obvykle využívá pro znázornění času.



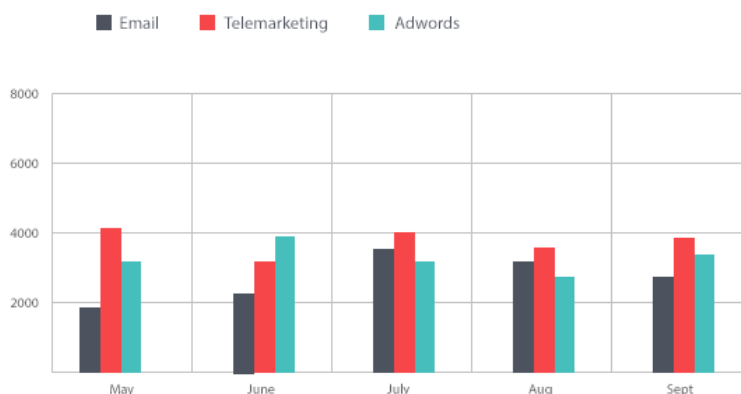
Obrázek 1 Příklad spojnicového grafu [3]

1.1.2 Sloupcový graf

Sloupcové grafy jsou velmi jednoduché na pochopení, a proto se hojně využívají pro prezentaci a porovnávání statistických údajů a dat. Informace je do grafu zanesena v podobě obdélníkových pruhů o různé výšce, nebo šířce, která je přímo úměrná hodnotě zobrazované veličiny. Sloupcové grafy se používají v podobných případech jako grafy spojnicové, ale na rozdíl od nich zobrazují rozdíly hodnot pouze v jednom směru. Existují dva základní typy sloupcového grafu - horizontální a vertikální.

Horizontální sloupcové grafy zobrazují hodnoty v ose X. Data jsou rozdělena do kategorií, které jsou rovnoměrně rozloženy do osy Y. Šířka každého pruhu pak představuje zobrazovanou hodnotu v daném měřítku. Horizontální rozložení sloupcového grafu má výhodu v případě delších názvů kategorií, nebo při jejich malém počtu. Hodnoty jsou v horizontální podobě lépe čitelné a graf působí přehledněji. Vertikální zobrazení se používá častěji v případě většího množství kategorií, nebo pokud jsou mezi sebou jednotlivé kategorie v časové závislosti. [4]

Kategorie mohou obsahovat libovolné množství hodnot. Díky tomu je možné velmi jednoduše porovnávat hodnoty v kategoriích, i celé kategorie najednou. V případě více hodnot v jedné kategorii existuje několik způsobů jak s těmito daty pracovat. Nejčastěji jsou sloupce každé kategorie shlukovány vedle sebe, nebo se hodnoty kategorií vrství na sebe a skládají do jednoho sloupce. Metoda skládání sloupců má výhodu při větším počtu hodnot v kategorii, kdy nezbývá dostatek prostoru pro zobrazení všech hodnot jednotlivě.



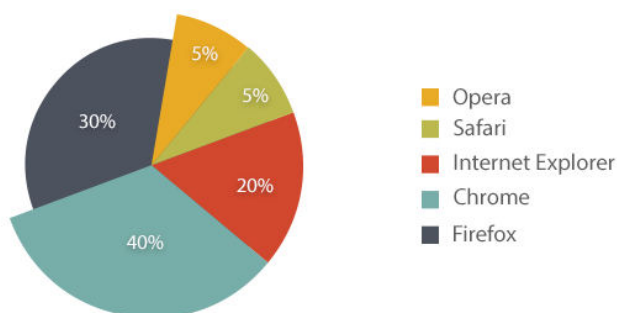
Obrázek 2 Příklad sloupcového grafu [3]

1.1.3 Segmentový graf

Segmentové grafy se využívají pro reprezentaci dat, jejichž hodnoty jsou sdružovány do kategorií. Tyto grafy jsou velmi dobře čitelné a jednoduché na konstrukci. Data jsou uspořádána do kruhu, který je rozdělen na segmenty podle jednotlivých kategorií. Velikost segmentu pak odpovídá poměru hodnoty kategorie k celku. Díky rozložení dat do celé plochy mají velké využití především pro zobrazení procentuálního poměru.

Segmentové grafy jsou velmi oblíbené, ale mají řadu omezení, které limitují jejich použití v praxi. Data jsou obvykle špatně čitelná v případě, že datová sada obsahuje větší množství kategorií. Pokud jsou si navíc hodnoty jednotlivých kategorií blízké, segmentový graf velmi rychle ztrácí na přehlednosti a informace obsažená v datech nemusí být na první pohled zřejmá. [5]

Segmentové grafy existují v různých variantách – při konstrukci grafu nemusí být například využita celá plocha kruhu, ale jen její poměrová část.



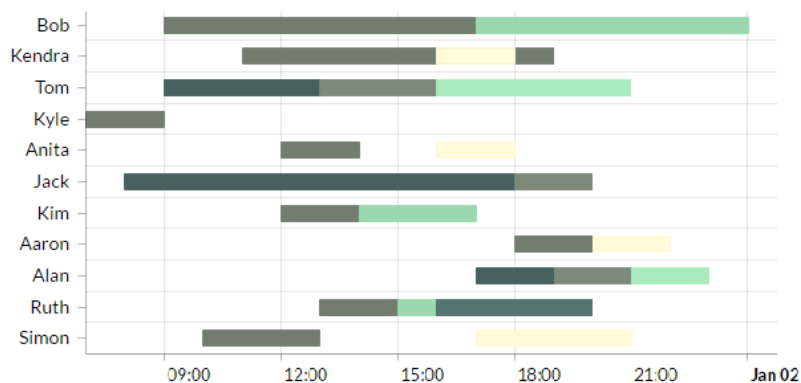
Obrázek 3 Příklad segmentového grafu [3]

1.1.4 Ganttův graf

Ganttův graf je jedním z nejpoužívanějších řešení pro vizualizaci aktivit a událostí. Často se využívá v oblasti řízení projektů a plánování. Ganttův graf umí zobrazit události v časovém kontextu tak, aby bylo vždy na první pohled patrné, kde událost začíná, kde končí, a jaké jsou mezi nimi vztahy.

Na horizontální ose je vždy zkoumané období, které je rozděleno na stejně dlouhé časové úseky. Na vertikální osu se zanáší zkoumané aktivity, projekty, nebo obecné objekty, u kterých jsou aktivity zkoumány. Aktivity pak mohou být odděleny řádky, nebo pouze vizuálním stylem dat.

Data se do grafu zanáší v podobě horizontálních pruhů, jejichž délka vždy značí dobu trvání aktivity. Z časové osy lze pak snadno odvodit začátek a konec každé události. V upravené podobě mohou diagramy ukazovat také návaznosti událostí pomocí lomených čar, kterými jsou vzájemně propojeny.



Obrázek Příklad Ganttova grafu

1.1.5 Ostatní grafy

Každý typ grafu má své specifické vlastnosti a omezení. Pro komplexní data se často využívají různé modifikace základních typů grafů, nebo speciální grafické struktury, které jsou uzpůsobeny pro reprezentaci konkrétních souvislostí.

Podle účelu vizualizace lze grafy obecně rozdělit do čtyř základních kategorií:

- *Srovnávací* – obecné porovnávání a třídění dat (sloupcový graf, bodový graf...)
- *Kompoziční* – porovnávání části dat vůči celku (segmentový graf...)
- *Distribuční* – zobrazují rozložení dat (krabicové grafy, histogramy...)
- *Vztahové* – ukazují vztah mezi dvěma a více veličinami

2 VYKRESLOVÁNÍ VE WEBOVÉM PROSTŘEDÍ

Při vykreslování obsahu se musejí webové aplikace zcela spoléhat na webové prohlížeče uživatelů a jejich schopnosti. Protože není vždy zaručena možnost přístupu ke všem prostředkům uživatele, je nutné vykreslovat obsah do mezivrstvy v podobě HTML dokumentu. Před nástupem standardu HTML5 se pro tento účel využívaly aplikace třetích stran, které bylo pro zobrazení interaktivního obsahu nutné do prohlížeče nainstalovat. S nástupem standardu HTML5 se objevily hned dvě možnosti, jak dynamicky vykreslovat interaktivní obsah přímo do obsahu webové stránky.

2.1 Canvas

Canvas je HTML5 element, který představuje oblast bitmapy umístěnou do obsahu HTML dokumentu a nabízí tak snadnou možnost vykreslování grafiky pomocí JavaScriptu. Základní Canvas API obsahuje 2D kontext, který umožňuje pomocí JavaScriptu provádět nad canvasem jednoduché grafické operace jako je vykreslování tvarů, textu, obrázků, filtrů apod. Před každou grafickou operací je možné kontextu nastavit parametry, s jakými se následující operace provede. [6]

Pomocí elementu canvas je možné dynamicky vykreslovat do obsahu stránky libovolnou grafiku a tento element tak představuje zcela nové odvětví pro programování webových aplikací. Vykreslováním rastrové grafiky přímo do HTML obsahu se uživatel oprostí od závislosti na aplikacích třetích stran, které se v minulosti pro tyto operace používaly. Při práci s elementem canvas lze navíc jednoduše veškerou uživatelskou interakci převést na jediný element a odstínit tak obsah aplikace od ostatních elementů na stránce. Elementy canvas je tedy možné používat i jako samostatné kontejnery pro celé aplikace.

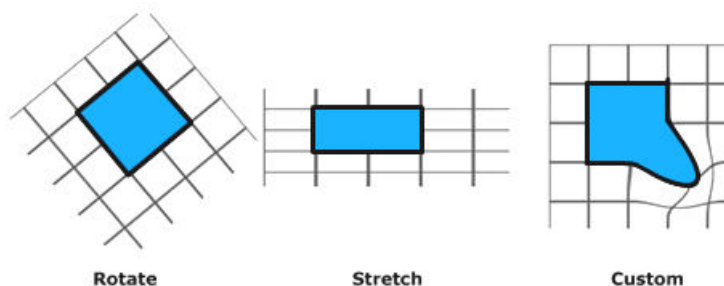
2.1.1 2D Context

Element canvas je navržen tak, aby mohl vykreslovat data do různých kontextů. Pro vykreslování jednoduché 2D grafiky se využívá objekt typu `CanvasRenderingContext2D` (2D Context), který je dostupný po zavolání metody `getContext()` s parametrem „2D“. 2D Context obsahuje všechny metody a parametry, které jsou potřebné pro samotné vykreslování. Obdobně jako většina bitmapových objektů pracuje 2D Context s Kartézskou soustavou souřadnic se středem v levém horním rohu bitmapové plochy.

Před vykreslením každého prvku je možné přes 2D Context nastavit parametry, se kterými se následující operace provede. Tyto parametry se ukládají jako globální stav, který je platný pro celé plátno a je možné jej kdykoliv změnit. Pomocí globálního stavu lze nastavit tři základní vlastnosti plátna – transformaci, výřez a parametry pro vykreslování. [7]

2.1.2 Transformace

Transformace plátna se provádí za pomoci transformační matice, jejíž parametry lze nastavit hromadně zavoláním metody *transform()*, nebo jednotlivě v závislosti na požadované operaci. Transformační maticí lze změnit posun, rotaci, měřítko, nebo deformaci zvolené oblasti. [15]



Obrázek 4 Příklady transformace obrazu

2.1.3 Nastavení výřezu

Nastavením výřezu je možné vymezit oblast plátna, která bude pro následující grafické operace relevantní. Výřez může mít podobu libovolného tvaru a jeho nastavení probíhá totožně jako při vykreslování. Namísto vykreslení je však po definování tvaru zavolána metoda *crop()*, která jej použije jako masku pro vymezení aktivní oblasti. [8]

2.1.4 Parametry vykreslování

Součástí globálního stavu jsou vykreslovací parametry, s kterými se následující operace provede. Mezi tyto parametry patří například zvolená barva výplně, styl a barva obrysu, nastavení písma, nebo nastavení kompozice.

Globální stav plátna je možné kdykoliv uložit a znovu obnovit. Pro uložení stavu se využívá metoda *save()*, která zkopíruje jednotlivé parametry stavu a uloží je na zásobník v dočasné paměti. Na tento zásobník je možné uložit libovolný počet stavů a následně je postupně uvolňovat zavoláním metody *restore()*. [7]



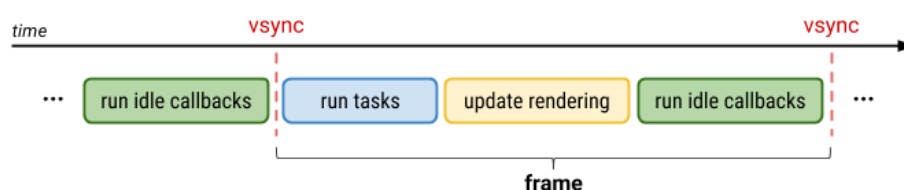
Obrázek 5 Uložení a obnova stavu plátna

2.1.5 Animace

Pro rozpohybování grafických objektů v elementu canvas se využívá nekonečné animační smyčky. Animační smyčka je funkce, která v pravidelném intervalu překresluje obsah plátna s aktualizovanými parametry.

Protože JavaScript běží v hlavním vlákne prohlížeče, není možné vytvořit nekonečnou smyčku pomocí cyklu, který při průběhu celé vlákno uzamkne. Při vlastní konstrukci animační smyčky je tedy nutné využít časovače, který je nastaven na zvolený interval a při každém přetečení se v návratové funkci spolu se zavoláním animační smyčky resetuje. Při použití časovače však dochází vlivem nadbytečné režie ke zpoždění a nepřesnostem, které se mohou snadno promítnout až do samotné animace.

Standard HTML5 zavádí pro konstrukci animační smyčky specifickou metodu, která se provádí na úrovni prohlížeče a eliminuje tak nepřesnosti vznikající při použití časovače. Zavoláním metody *requestAnimationFrame(loop)* se zaregistruje požadavek na prohlížeč, který vyčká na událost překreslení okna a poté zavolá návratovou funkci. [7]



Obrázek 6 Diagram překreslovací smyčky

2.1.6 Vykreslování

Před samotným vykreslením jakéhokoliv grafického objektu do plátna dojde vždy na pozadí prohlížeče k sérii operací. Grafický objekt se nejprve vykreslí do průhledné, nekonečné bitmapy s ohledem na aktuální nastavení barev a stylů. Následně dojde k vytvoření další bitmapy, do které jsou vykresleny stíny a u obou bitmap jsou přepočítány hodnoty pixelů podle aktuálního nastavení globální průhlednosti. Teprve poté jsou bitmapy překresleny do reálného plátna podle určené kompozice. [6]

Aplikační rozhraní pro práci s elementem canvas rozlišuje 3 základní grafické operace – obrys, výplň a jejich kombinaci. Před každou operací je třeba definovat plochu, na kterou se má daná operace aplikovat. Vymezení plochy je možné vytvořením cesty, která se definuje pomocí čar, křivek, nebo předdefinovaných geometrických tvarů.

Vytváření cesty pro vymezení plochy by mělo vždy začínat metodou *beginPath()*, která vymaže reference na všechny dílčí plochy vymezené předchozími cestami. Následuje samotné vymezení plochy, které je možné realizovat jako sérii navazujících bodů, nebo vytvořením předdefinovaného tvaru. S vymezenou plochou je dále možné pracovat až do dalšího zavolání metody *beginPath()*. [7]

Základním tvarem pro vytvoření cesty je obdélník. Obdélníková plocha má vždy stejné argumenty, a to pozici, výšku a šířku. Plochu obdélníků je možné vytvořit metodou *rect*, případně rovnou vykreslit zkrácenými metodami *fillRect* a *strokeRect*.

Dalšími základními tvary jsou kruh a kruhová výseč. Ty se vytvářejí metodou *arc*, u které je vždy možné definovat absolutní pozici, počáteční a koncový úhel v radiánech, radius, a směr úhlu.

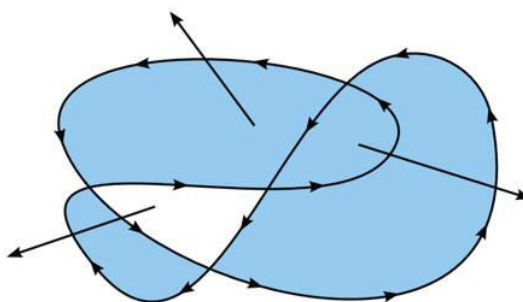
2.1.7 Výplň

Uzavřenou oblast definovanou vymezovací cestou je vždy možné vyplnit jednolitou barvou, bitmapovým vzorem, nebo barevným přechodem. Styl výplně se definuje do parametru *fillStyle*, který je součástí globálního stavu. Hodnota parametru může být vložena ve tvaru hexadecimálního kódu barvy, případně jako speciální objekt se specifikací barevného přechodu, nebo bitmapového vzoru. [6]



Obrázek 7 Varianty výplně

Pokud se při vyplňování oblasti navzájem kříží vymezení cesty a dochází ke vzniku vnitřních ploch, probíhá vyplňování podle tzv. „non-zero winding“ pravidla. Pro každou uzavřenou plochu je spočítán směr cesty, která plochu vymezuje a ten je srovnán se směrem nadřazených ploch. Opačné směry se navzájem vyruší, a pokud na konci výpočtu dojde k úplnému vynulování, prohlížeč danou plochu nevyplní. [7]

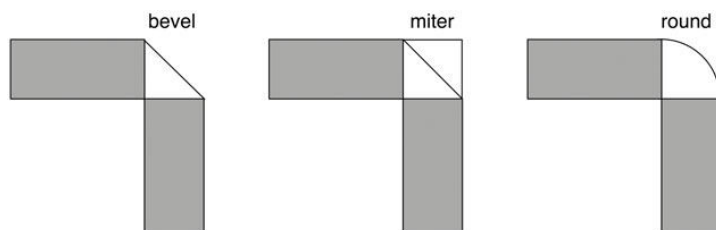


Obrázek 8 Vyplňování vnitřních oblastí

2.1.8 Obrysy a čáry

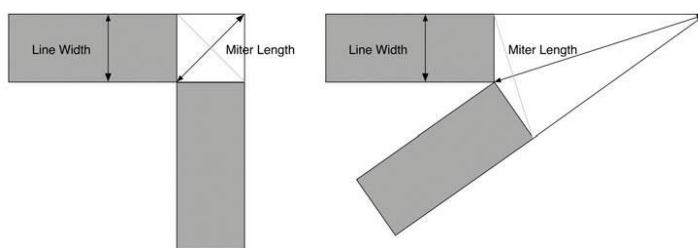
2D kontext obsahuje dvě metody, pomocí kterých je možné vytvářet a spojovat čáry – *moveTo* a *lineTo*. Pomocí metody *moveTo* se posune virtuální kurzor na zvolenou pozici a tím je určen začátek cesty. Zavoláním metody *lineTo* dojde opět k posunutí na zvolenou pozici, nicméně mezi oběma body je vytvořena reference cesty pro následné vykreslení.

Jako parametry globálního stavu lze u čar a obrysů vždy definovat jejich barvu, styl, tloušťku, zalamování a zakončení. Aplikační rozhraní nabízí celkem tři možnosti pro styl zalomení – „bevel“, „miter“, a „round“. [7]



Obrázek 9 Styl zalomení čáry

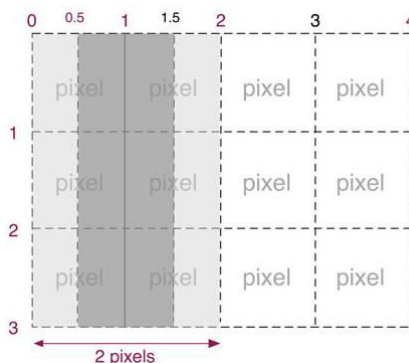
U ostrého zalomení (miter) je dále možné specifikovat parametr „miterLimit“, který představuje maximální velikost přesahu spojení v poměru k zadané tloušťce čáry.



Obrázek 10 Varianty ostrého zalomení

Při vykreslování do rastru je možné pracovat pouze s celými pixely. Pokud čára při vykreslování částečně zasahuje do okolních pixelů, je hodnota barvy těchto pixelů přepočítána podle poměru, kterým čára do daného pixelu zasahuje. Pozice čáry se přitom vždy bere relativně ke středu tloušťky. [7]

Pro potlačení efektu nežádoucího rozostření je tedy nutné přepočítávat pozice čar pro vykreslení vždy na poloviny pixelů.

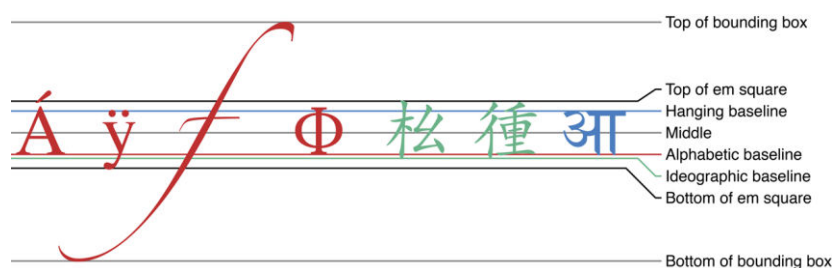


Obrázek 11 Pozice pixelů v elementu Canvas

2.1.9 Text

Nastavení parametrů pro vykreslování textu se provádí pomocí standardního CSS zápisu stylu písma, který se uloží jako hodnota parametru *font* do globálního stavu plátna. Vykreslování a výplň textu probíhá obdobně jako u ostatních grafických objektů.

Pomocí metody *measureText()* je možné zjistit velikost textu ještě před samotným vykreslením. Metoda vrátí objekt textové metriky s atributem *width*, ve kterém je obsažena šířka textu po vykreslení do plátna. Objekt textové metriky však neobsahuje údaje o výšce a pro její výpočet je proto nutné text nejdříve umístit do elementu stránky. [7]



Obrázek 12 Rozložení a rozměr textu

2.1.10 Vykreslování bitmapy

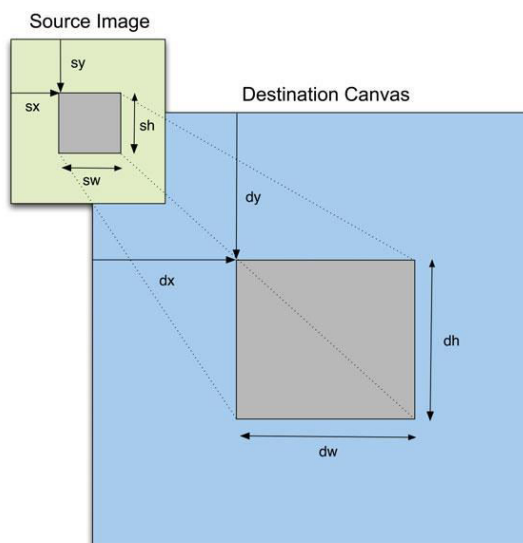
Element canvas obsahuje rozsáhlou podporu pro práci s obrazovými daty. Obrazová data lze vykreslovat po částech, měnit jejich měřítko, nebo upravovat jednotlivé pixely. Pro vykreslení obrazových dat je v kontextu zavedena funkce „drawImage“, která umožňuje data nejen vykreslit, ale také přímo změnit jejich měřítko, nebo specifikovat konkrétní část bitmapy pro vykreslení. [8]

Vykreslovat bitmapu je možné se třemi různými typy argumentů:

`drawImage(image, dx, dy)` – slouží pro jednoduché vykreslení obrazových dat na požadovanou pozici bez úprav a v plném měřítku.

`drawImage(image, dx, dy, dw, dh)` - umožňuje při vykreslování volit nejen pozici, ale také výslednou velikost, do které se obrazová data roztáhnou.

`drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)` – vykreslí specifickou oblast zdrojového obrázku do definované oblasti plátna.



Obrázek 13 Vykreslování bitmapy

Vykreslovat je možné jakýkoliv DOM element, který obsahuje obrazová data (img, video, canvas...). Podmínkou pro vykreslení je však vždy požadavek na načtení obrazových dat do dočasné paměti. V praxi se proto velmi často využívají obrazové mapy, kdy se do jednoho souboru uloží více bitmapových zdrojů, které se takto načtou v jediném kroku. Při vykreslování se pak mění pouze výřez a odsazení, které odpovídá pozici vykreslovaných dat ve zdrojovém souboru. [7]

Standard HTML5 zavádí pro práci s obrazovými daty objekt typu „ImageData“, který obsahuje údaje o rozměrech obrazových dat a samotná data v podobě jednorozměrného dynamického pole. Obrazová data jsou uložena jako 8 bitové hodnoty barevných složek RGBA. [8]

| | | |
|-------------------|-----|-------|
| imagedata.data[0] | 55 | red |
| imagedata.data[1] | 255 | green |
| imagedata.data[2] | 38 | blue |
| imagedata.data[3] | 255 | alpha |

Obrázek 14 Struktura obrazových dat

Obrazová data lze získat přímo z elementu Canvas zavoláním metody „getImageData()“. Extrahovaná data je možné procházet, analyzovat, nebo libovolně modifikovat a poté znovu nahrát do elementu canvas. Přepočítáním hodnot pixelů lze docílit nejrůznějších úprav a efektů bez nutnosti zásahu aplikací třetích stran.

2.1.11 Nastavení kompozice

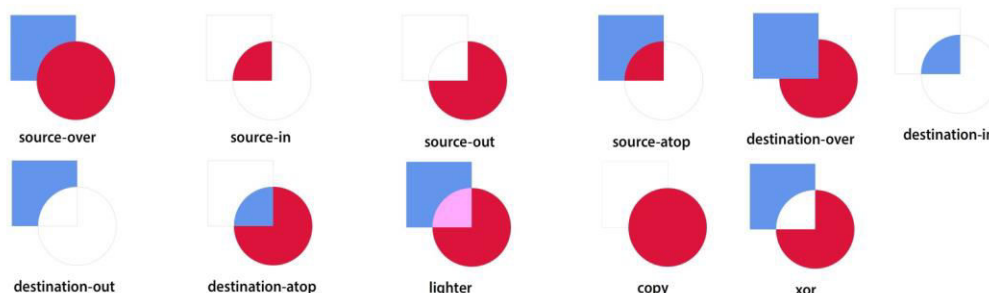
Veškeré vykreslování do 2D kontextu plátna je ovlivněno dvojicí kompozičních atributů, které jsou součástí globálního stavu plátna a ovlivňují způsob, jakým jsou objekty překreslovány a vrstveny přes sebe. [8]

globalAlpha [0-1] – atribut vrací aktuální nastavenou hodnotu průhlednosti. Přiřazením hodnoty je možné průhlednost změnit. Hodnota se musí nacházet v rozsahu od 0.0 (plně průhledný) do 1.0 (neprůhledný). Výchozí hodnotou při vytvoření kontextu je 1.0, tedy neprůhlednost objektů.

globalCompositeOperation – určuje, jakým způsobem jsou grafické objekty překreslovány a vrstveny přes sebe. K aplikaci této vlastnosti dochází až po nastavení průhlednosti a transformace. V současné době je ve většině prohlížečů dostupná podpora pro celkem 11 kompozičních stavů: [8]

- Copy – pokud se grafické objekty překrývají, je vykreslen vždy pouze objekt v popředí.
- Destination-atop – objekt v nižší vrstvě je vykreslen do popředí. Část původního objektu, která přesahuje přes aktuálně vykreslovaný objekt je v tomto případě ignorována.
- Destination-in – vykreslí se pouze část objektu v popředí, která zasahuje do původního objektu. Původní objekt je při vykreslování ignorován.
- Destination-out - vykreslení se pouze část původního objektu, která nezasahuje do objektu v popředí. Objekt v popředí je při vykreslování ignorován.
- Destination-over – původní objekt je přenesen do popředí a překreslen přes nový objekt.

- **Lighter** – oblasti objektů, které se navzájem překrývají, jsou vykresleny jako součet barevných složek obou objektů.
- **Source-atop** – vykreslovaný objekt je vymezen původním objektem v nižší vrstvě.
- **Source-in** – vykreslí se pouze část nového objektu, která zasahuje do objektu v nižší vrstvě. Původní objekt je při vykreslování ignorován.
- **Source-out** - vykreslí se pouze část nového objektu, která nezasahuje do objektu v nižší vrstvě. Původní objekt je při vykreslování ignorován.
- **Source-over** – základní stav. Objekt ve vyšší vrstvě je překreslen přes původní objekt v pozadí.
- **Xor** – oblasti objektů, které se navzájem překresleny jsou při vykreslování ignorovány.



Obrázek 15 Ukázka obrazových vrstev

2.1.12 Optimalizace vykreslování

Při opakovaném vykreslování většího množství objektů je vhodné celý proces optimalizovat tím, že se obsah scény rozdělí do vrstev, které se uloží do dočasné paměti v podobě virtuálního elementu. Každá vrstva by měla obsahovat objekty, které je nutné překreslovat s podobnou frekvencí. Obsah vrstvy se pak překreslí pouze při své vlastní změně, a při obnovení plátna je celá vrstva vykreslena najednou.

Vykreslovací plochu plátna je také vhodné rozdělit na rovnoměrné části a překreslovat jen ty plochy, kde dojde ke skutečné změně obsahu.

Pokud jsou při obnovování scény prováděny složitější výpočty, je možné tyto výpočty přesunout do vlastního vlákna a zabránit tak nežádoucímu vytížení hlavního procesu.

Standard HTML5 umožňuje vytvořit instanci třídy „Worker“, která zpracuje přidružený JavaScriptový kód v samostatném vlákně. Této optimalizace je vhodné využít především při procházení a analýze obrazových dat s vysokým rozlišením, kde často dochází ke komplexní zátěži hlavního vlákna.

2.2 SVG

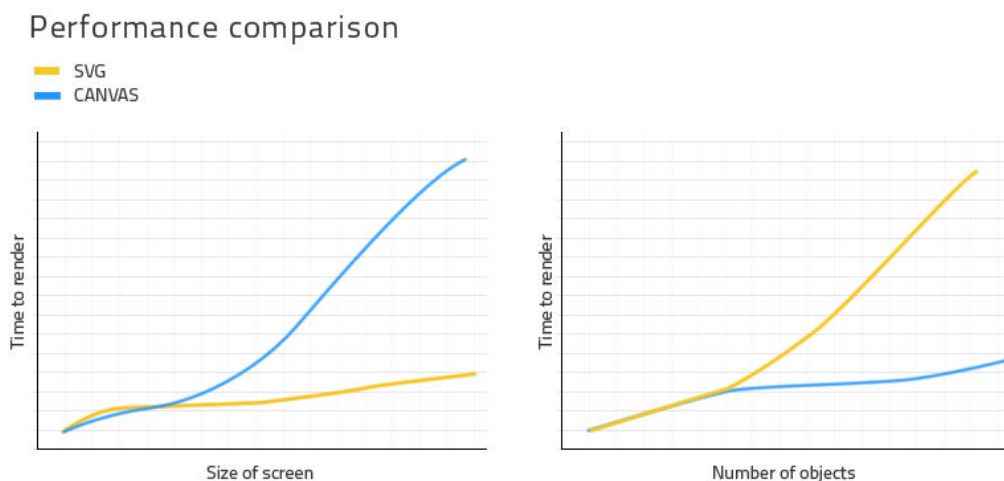
SVG (Scalable Vector Graphics) je flexibilní vektorový formát založený na struktuře XML (Extensible Markup Language) dokumentu. V dnešní době má ve webových prezentacích široké uplatnění díky masivnímu rozšíření podpory ze strany internetových prohlížečů. Standard HTML5 zavádí specifikaci pro vložení SVG grafiky přímo do HTML5 kódu za pomoci značky <SVG>. To umožňuje přistupovat k této grafice jako ke kterémukoliv jinému DOM element a manipulovat s ním pomocí CSS a JavaScriptového kódu. Díky tomu je možné vytvářet, upravovat, nebo mazat jednotlivé prvky za běhu a použít tak obsah elementu pro dynamické vykreslování libovolné grafiky. [9]

Vektorová grafika má velkou výhodu ve velikosti dat, protože na rozdíl od rastrové reprezentace obrazu jsou ve formátu uloženy pouze reference na jednotlivé objekty v podobě souřadnic a doplňujících parametrů. Na základě matematických výrazů a operací se pak z těchto objektů vytvoří celkový obraz. Obrazce ve formátu SVG lze navzájem snadno slučovat, transformovat, nebo animovat. Pro zavedení interaktivity je možné na jednotlivých objektech odchyťovat události a obsluhovat je přímo s patřičnou referencí. Velkou výhodou vektorového formátu dat je také možnost měnit velikost obrazu bez závislosti na výsledné kvalitě. [10]

Práce s formátem SVG je v mnoha případech jednodušší. Vykreslování vektorové grafiky je však náročnější na výpočetní výkon a při volbě technologie je proto potřeba zohlednit nejen výhody vektorového formátu, ale také výkonovou optimalizaci při samotném vykreslování.

Vykreslení obrazových bodů do rastru je v tomto ohledu obvykle výhodnější a optimalizační výhoda se zvyšuje spolu se složitostí grafiky a počtem vykreslovaných prvků. Rychlost vykreslování do rastru je nepřímo úměrná především velikosti plátna. Využití formátu SVG bude tedy výhodnější pro menší množství objektů, nebo v případě

velké vykreslovací plochy. Pro vizualizaci většího množství dat a složitějších prvků, nebo animací, je však vhodnější využít vykreslování do elementu canvas.



Obrázek 16 Výkonové srovnání vykreslovacích technologií

3 ZÍSKÁVÁNÍ DAT

Nejdůležitějším předpokladem pro vizualizace je přístup ke zdroji dat. Protože je JavaScript zpracováván na straně klienta, nenabízí tato technologie žádné nástroje, které by dovolily vytvářet přímé vazby mezi klientem a databází. Z tohoto důvodu je nutné pro přístup ke zdroji dat využít mezivrstvu v podobě serverové aplikace.

Požadavek na získání dat je odeslán na vzdálený script, který je umístěný na serveru a má přímou vazbu na databázi, nebo alternativní zdroj dat. Po odeslání požadavku z klientské aplikace se na serveru zavolá funkce, která přebere parametry požadavku a vytvoří dotaz na databázi. Databáze vrátí odpověď v podobě čitelné struktury dat. Ta je následně převedena na formát vhodný pro JavaScript a je odeslána zpět klientské aplikaci jako odpověď na odechozí požadavek. V rámci optimalizace síťové komunikace a zachování stabilní odezvy je vhodné omezit odesílání požadavků na minimum. [11]

Další možností pro optimalizaci serverové komunikace je využití konceptu paměťového uložení pomocí standardu HTML5/Web Storage. I v tomto případě je potřeba serverový script pro komunikaci s databází, nicméně data jsou po získání uložena do dočasné paměti webového prohlížeče a při každém dalším požadavku na server je ještě před odesláním dat zkontrolováno, jestli od posledního požadavku nastala nějaká změna. [11]

3.1 Webové služby

Webové služby jsou kolekcí otevřených protokolů a standardů pro výměnu dat mezi klientskou aplikací a serverem. Na rozdíl od tradičního klient-server modelu nemají webové služby grafické rozhraní, ale sdílí logiku, data a procesy přes síťový kanál. Nejběžněji používané standardy jsou XML (Extensible Markup Language) pro formát dat, SOAP (Simple Object Access Protocol) pro jejich odesílání a standardy UDDI (Universal Description/Discovery and Integration) a WSDL (Web Services Description Language) pro popis webových služeb a prostředků. V technologii JavaScript je možné k webovým službám přistupovat pomocí standardu AJAX (Asynchronous JavaScript and XML), kde je výměna dat realizována formou požadavků založených na protokolu HTTP (Hypertext Transfer Protocol). [11]

3.2 REST API

Representational State Transfer (REST) je architektura rozhraní navržená pro distribuované prostředí. Rozhraní navržené architekturou REST je možné využít pro jednotný a snadný přístup ke zdrojům. REST je orientovaná datově, nikoliv procedurálně, a vytváření dotazů není závislé na skládání XML struktury. Stav a chování aplikace je vyjádřeno takzvaným resourcem (klíčová abstrakce), který vždy musí mít unikátní identifikátor (URI). Přístup k datům se pak realizuje pomocí čtyř základních metod, které jsou někdy souhrnně známé pod označením CRUD (create, read, update, delete). [12]

- **GET (Retrieve)** - metoda pro získání zdrojů. Vráťí kolekci dat, nebo reprezentaci adresovaného členu v kolekci.
- **POST (Create)** – metoda pro vytvoření nového datového záznamu. U metody POST není ve chvíli volání známý přesný identifikátor zdroje. Ten je přidělen automaticky po zpracování požadavku a obvykle odeslán zpět ve formě odpovědi.
- **PUT (Update)** – metoda pro úpravu dat. Operace je velmi podobná vytváření záznamu, ale pracuje se s konkrétním identifikátorem. Umožňuje upravit jednotlivé prvky v kolekci, nebo vyměnit celé datové sady.
- **DELETE** – metoda pro smazání záznamu. V praxi se pro úpravu a mazání dat využívá spíše metoda POST se speciálními parametry, nebo specifickou URI adresou. [13]

Při výměně dat je možné v architektuře REST využít hned několik standardizovaných formátů. Nejčastěji se však v souvislosti s webovými aplikacemi využívá formát JSON, který spolu s architekturou REST vytváří jednotný standard komunikace pro aplikační rozhraní webových služeb. [12]

4 FORMÁT A STRUKTURA DAT

Při načítání dat je důležitá nejen samotná komunikace se serverem, ale také způsob, jakým jsou data uložena a jejich struktura. Na serveru bývají data uložena nejčastěji formou relačních databází. Databáze by měla být navržena s ohledem na uchovávaná data a požadavky webové služby, která by měla být vždy schopná strukturovat data do vhodného formátu pro klientskou aplikaci. [11]

Struktura dat získaných ze serveru by měla být co nejvíce přizpůsobena specifickým požadavkům na vizualizaci, aby režie spojená se zpracováním dat byla na straně klienta co nejmenší. Všechny relevantní informace pro samotnou vizualizaci by měly vycházet z podmnožiny relačního modelu.

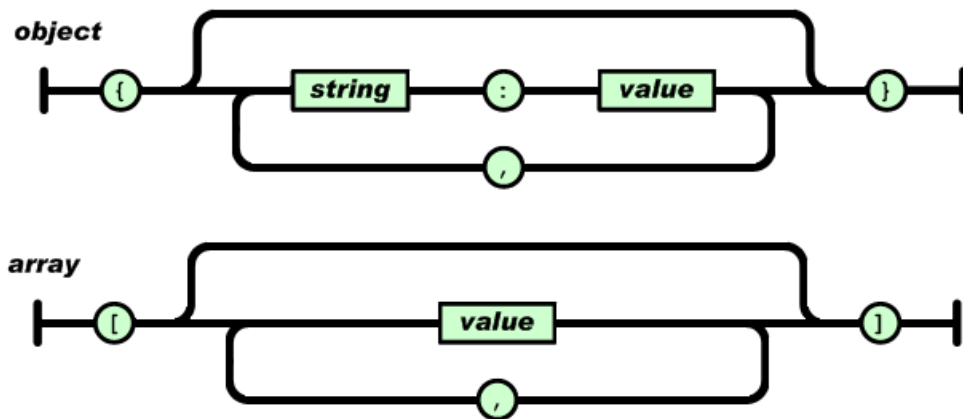
Pro výběr formátu dat existuje celá řada standardizovaných možností a správná volba proto vždy závisí na konkrétních požadavcích aplikace a jejím účelu. Webové aplikace založené na technologii AJAX využívají pro přenos dat nejčastěji formát XML, nebo JSON.

Formát XML je vhodný především pro složitější datové struktury, protože kromě samotných dat obsahuje i popisové značky a data jsou tak vždy přenášena spolu s kontextem. Právě popisové značky a jejich atributy však mohou být velkou nevýhodou v případě, že aplikace klade důraz na rychlost a minimalizaci přenosů. U moderních webových aplikací se proto využívá více formát JSON, který se někdy považuje za odlehčenou verzi XML. [14]

4.1 JSON

JavaScript Object Notation (JSON) je otevřený formát pro výměnu dat založený na struktuře JavaScriptových objektů. Stejně jako XML je formát JSON nezávislý na platformě a má širokou dostupnost implementací. Nespornou výhodou formátu JSON je jednoduchost zápisu a minimální datová náročnost. Rozhraní pro práci s tímto formátem je navíc přímou součástí moderních prohlížečů a zápis ve formátu JSON je zároveň nativním zápisem v jazyce JavaScript. Sestavení datové struktury se tedy provádí na úrovni webového prohlížeče a není proto nutné implementovat vlastní řešení. V současné době je formát JSON neoptimálnější řešení pro realizaci asynchronní komunikace. [14]

JSON je založen na dvou strukturách – neuspořádaná množina párů název/hodnota (objekt) a seřazená kolekce hodnot (pole). Jedná se o univerzální datové struktury a v podstatě všechny moderní programovací jazyky je v nějaké formě podporují.



Obrázek 17 struktury formátu JSON [15]

Do JSON je možné uložit následující typy dat:

- JSONString – textový řetězec.
- JSONNumber – číslo (celočíslné nebo reálné, včetně zápisu s exponentem)
- JSONBoolean – logická hodnota
- JSONNull – hodnota null
- JSONArray – pole
- JSONObject- objekt

5 KNIHOVNY PRO VYKRESLOVÁNÍ DDAT

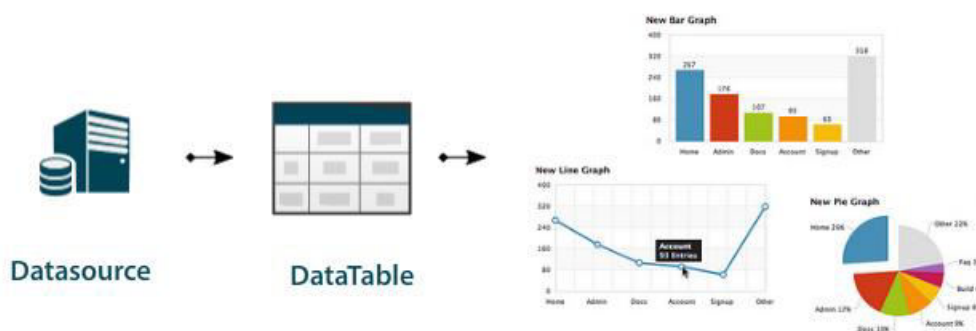
5.1 Google Charts

Aplikační rozhraní Google Charts umožňuje snadné vykreslování dat do grafických komponent. Rozhraní obsahuje celou řadu předpřipravených grafů od jednoduchých čar, až po komplexní stromové struktury.

Komponenty grafů jsou zapouzdřeny do oddělených tříd, které se načítají synchronně v závislosti na požadavcích aplikace. Veškeré grafické komponenty jsou interaktivní a obsluhují celou řadu událostí, které usnadní implementaci do komplexních dashboardů a pokročilých webových aplikací.

Pro vykreslování využívá tato knihovna technologii HTML5/SVG, která umožňuje vykreslovat grafické prvky v podobě vektorového formátu přímo do struktury HTML dokumentu bez závislosti na zvolené platformě. Díky implementované technologii VML (Vector Markup Language) je navíc zajištěna zpětná kompatibilita se staršími prohlížeči bez podpory standardu HTML5.

Do všech grafických komponent jsou data načítána pomocí speciální třídy *DataTable*, která funguje jako mezivrstva pro standardizaci formátu a základní manipulaci s daty. Třída *DataTable* reprezentuje dvourozměrnou tabulku dat a obsahuje mimo jiné metody pro filtraci, třídění a modifikaci dat, které mohou být načteny přímo z webové adresy, databáze, nebo jakéhokoli dalšího rozhraní s podporou protokolu *Chart Tools DataSource*. [16]



Obrázek 18 Struktura Google Charts vizualizace

Pro práci s knihovnou Google Charts stačí načíst soubor se statickou třídou *loader*, pomocí které lze načíst jednotlivé komponenty ve zvolené verzi. Licenční podmínky knihovny

Google Charts nedovolují přímé načítání komponent z lokálního úložiště. Aktuální verze knihovny je vždy dostupná pod identifikačním názvem „current“.

```
// načtení aktuální verze knihovny
google.charts.load('current', { packages: ['corechart'] });

// vytvoření instance datové struktury
var data = new google.visualization.DataTable();
data.addColumn('string', 'label');
data.addColumn('number', 'value');
data.addRows([ { label: 'data_1', value: 10 }, { label: 'data_2', value: 20 },,, ]);

// vytvoření a umístění koláčového grafu
var chart = new google.visualization.PieChart(document.getElementById('chart_div'));

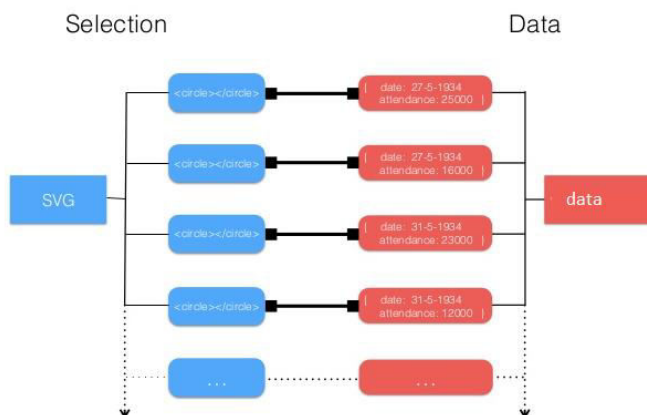
// vykreslení dat
chart.draw(data, options);
```

Obrázek 19 Vytvoření grafu pomocí knihovny Google Charts

5.2 D3.js

D3 (Data-Driven Documents) je JavaScriptová knihovna, která slouží k manipulaci s dokumenty na základě dat. Místo použití vlastních komponent a abstraktního rozhraní umožňuje D3 přímou kontrolu a manipulaci s nativní webovou prezentací a jejími elementy. Knihovna D3 umožňuje selektivně navázat data na konkrétní prvky a jejich vlastnosti, které se dynamicky mění na základě vložených dat. [17]

D3 na rozdíl od ostatních knihoven nepracuje s předdefinovanými grafickými komponenty, které si zde uživatel vytváří za pomoci definic a atributů dynamicky sám.



Obrázek 20 Struktura datově řízeného dokumentu

Pro složitější vizualizace se využívá vektorový formát SVG, který je možné ve standardu HTML5 vkládat do webové prezentace formou HTML elementů. D3 tak s těmito elementy pracuje stejně jako se zbytkem HTML struktury a pomocí specifických selektorů umožňuje navázat data přímo na jednotlivé grafické objekty. Díky minimální režii je D3 velice rychlý a podporuje práci s velkými soubory dat. Při použití HTML elementů lze navíc pro úpravu vzhledu vizualizace využít CSS styly.

Knihovna D3 je vhodná především pro vytváření specifických grafických struktur a komplexních aplikací. Nevýhodou je však složitost implementace a omezená podpora pro starší prohlížeče.

```
// vykreslení dat
chart.draw(data, options);

// data
var dataset = [{ label: 'data_1', value: 10 }, { label: 'data_2', value: 20 }, , , ];

// vložení svg elementu
var svg = d3.select('#chart').append('svg').attr('width', width).attr('height', height).append('g');
var arc = d3.svg.arc().outerRadius(radius);

// vytvoření struktury
var pie = d3.layout.pie().value(function (d) { return d.value; })

// propojení atributů s daty
var path = svg.selectAll('path')
    .data(pie(dataset))
    .enter()
    .append('path')
    .attr('d', arc)
    .attr('fill', function (d, i) {
        return color(d.data.label);
    });
```

Obrázek 21 Vytvoření grafu pomocí knihovny D3

5.3 Flot Charts

Flot Charts je rozšíření pro knihovnu jQuery, která usnadňuje manipulaci s DOM elementy a odděluje funkční logiku od struktury HTML. Flot má předpřipravené komponenty pro 4 základní typy grafů - sloupcový, spojnicový, bodový a segmentový. Tyto komponenty je však možné neomezeně rozšiřovat a měnit širokou škálu konfiguračních parametrů. Knihovna podporuje více časových os a umožňuje tak rozdílné datové sady do jednoho zobrazení. Díky interakci s technologií jQuery je navíc možné přidávat externí

komponenty, které si mohou vytvářet sami uživatelé a nemusí být pevnou součástí implementace.

Komponenty se do HTML struktury vkládají pomocí jQuery selektorů a vstupních parametrů v podobě vícerozměrného pole dat a konfiguračního objektu. [18]

Pro vykreslování využívá knihovna Flot Charts technologii HTML5/Canvas s nástavbou pro zpětnou kompatibilitu se staršími prohlížeči v podobě externí knihovny excanvas, která pro vykreslování využívá technologii VML. [18]

Díky jQuery je vytváření komponent a manipulace s nimi velmi intuitivní a nenáročná. Knihovna Flot Charts je proto velmi vhodná pro začínající uživatele, nebo webové aplikace, které již ve své struktuře mají implementovanou technologii jQuery.

```
// data
var dataset = [
    { label: 'data_1', data: 10 },
    { label: 'data_2', data: 20 },,,
];

// vytvoření komponenty koláčového grafu
$.plot('#chart_div', data, {
    series: {
        pie: {
            show: true
        }
    }
});
```

Obrázek 22 Vytvoření grafu pomocí knihovny Flot Charts

5.4 ChartJS

ChartJS je otevřená modulární knihovna, která se zaměřuje na jednoduchost a interaktivitu řešení. Knihovna obsahuje 8 základních komponent grafů, které je možné libovolně upravovat a rozvíjet. Každá komponenta má podobu vlastní Třídy a je umístěna v samostatném souboru. Uživatel tak má možnost načíst jen ty komponenty, se kterými chce ve své aplikaci pracovat. Předpokladem pro použití je však vždy načtení souboru se statickou třídou jádra, která obsahuje základní logiku společnou pro všechno komponenty a globální metody pro pomocné výpočty. Knihovna nabízí velké množství rozšíření a specifických funkcí, na jejichž vývoji se částečně podílejí sami uživatelé. [19]

Pro vykreslování využívá ChartJS standard HTML5/Canvas bez zpětné kompatibility se staršími prohlížeči, který poskytuje značnou optimalizační výhodu pro rozsáhlou nabídku

animačních přechodů. Komponenty grafů jsou plně responzivní a je tak možné umístit je do kterékoliv webové prezentace bez závislosti na koncové platformě. [19]

Grafy se zde vytváří jako instance obecné třídy *Chart*, přičemž data i typ komponenty jsou součástí konfiguračního objektu, který se předává v konstruktoru třídy.

Knihovnu ChartJS je vhodné implementovat především pro menší webové aplikace, nebo při jednorázovém použití.

```
// data
var dataset = {
  labels: ["data_1", "data_2", ...],
  datasets: [
    { data: [10, 20, , , ] }
  ]
};

// vytvoření komponenty koláčového grafu
var chart = new Chart(context, {
  type: 'pie',
  data: dataset,
  options: {}
});
```

Obrázek 23 Vytvoření grafu pomocí knihovny ChartJS

5.5 AMCharts

AMChart je zástupcem komerčního řešení a zároveň je jednou z nejkompaktnějších dostupných knihoven pro grafické vykreslování dat. Kromě standardních grafických komponent obsahuje celou řadu specifických prvků pro tvorbu interaktivních vizualizací a komplexních dashboardů. [20]



Obrázek 24 Ukázka komplexního výstupu knihovny AMCharts

Tato knihovna cílí na pokročilé systémy se specifickým zaměřením a obsahuje velké množství předpřipravených šablon, od vlastního řešení mapových podkladů, až po simulace dopravní infrastruktury. Součástí knihovny je také rozhraní pro interakci s moderními frameworky, které zajišťuje snadnou implementaci do libovolného webového prostředí.

Vykreslování dat je stejně jako u předešlé knihovny realizováno za pomoci standardu HTML5/SVG se zpětnou podporou pro starší prohlížeče v podobě technologie VML. [20]

```
// vytvoření komponenty koláčového grafu
var chart = AmCharts.makeChart("chart_div", {
    "type": "pie",
    "theme": "light",
    "dataProvider": [
        { "name": "data_1", "value": 10 },
        { "name": "data_2", "value": 20 },,,
    ],
    "valueField": "value",
    "titleField": "name",
    "balloon": {
        "fixedPosition": true
    },
    "export": {
        "enabled": true
    }
});
```

Obrázek 25 Vytvoření grafu pomocí knihovny AMCharts

5.6 Highcharts

HighCharts je jednoduchá knihovna napsaná v čistém JavaScriptu. Nabízí celou řadu předpřipravených grafických komponent, které je možné mezi sebou libovolně kombinovat. Knihovna podporuje vícenásobné časové osy, interaktivní rozhraní a komplexní práci s časovými jednotkami. Součástí řešení pro vytváření spojnicových grafů je inteligentní časová osa, která se automaticky přizpůsobuje vloženým hodnotám. Mimo jiné umožňuje knihovna HighCharts také export aktuálního zobrazení do PDF a přímé uložení na disk. [21]

Přestože je knihovna zdarma pouze pro nekomerční použití, licenční podmínky umožňují uživateli libovolně zasahovat do zdrojového kódu a provádět jakékoliv změny.

Konfigurace a vložení dat probíhá prostřednictvím konfiguračního objektu, který se přes selektivní rozhraní naváže na konkrétní HTML element.

Pro vykreslení se zde využívá vektorový standard HTML5/SVG se zpětnou podporou starších prohlížečů v podobě technologie VML. [21]

```
// vytvoření komponenty koláčového grafu
$('#container').highcharts({
  chart: {
    type: 'pie'
  },
  plotOptions: {
    pie: {}
  },
  series: [{
    name: '#',
    colorByPoint: true,
    data: [
      { name: 'data_1', y: 10 },
      { name: 'data_2', y: 20 },,,
    ]
  }]
});
```

Obrázek 26 Vytvoření grafu pomocí knihovny HighCharts

5.7 Srovnání

Každá z uvedených knihoven představuje zástupce specifické skupiny, nebo řešení, které se v současné době běžně používají pro vizualizaci dat a konstrukci webových dashboardů. V praxi se dostupné knihovny liší především technologií pro vykreslování, formátem vstupních dat a složitostí implementace. Technologie vykreslování se s příchodem standardu HTML5 sjednotila na dvě používané řešení - Canvas a SVG. Obě technologie mají stejnou podporu ve webových prohlížečích a výběr by tedy měl záviset pouze na konkrétním účelu použití. Komerční knihovny dávají často přednost formátu SVG, který je vektorový a umožňuje vykreslovat data ve velkém měřítku bez zásadního vlivu na výkon. Pro zpětnou kompatibilitu se staršími prohlížeči se z pravidla využívá standard VML, s jehož udržováním je však spojeno mnoho kompromisů a nároků na samotnou knihovnu. Moderní knihovny, které cílí především na praktičnost a vizuální podobu komponent, proto se zpětnou kompatibilitou často nepočítají.

Tabulka 1 Srovnání vybraných knihoven

| | Google Charts | D3 | Flot Charts | ChartJS | AMCharts | HighCharts |
|------------------------------------|---------------------|-----------------|--------------------|--------------------|------------------|--------------------------|
| Rok vydání | 2007 | 2011 | 2007 | 2013 | 2006 | 2009 |
| Technologie vykreslování | SVG/VML | SVG | Canvas/VML | Canvas | SVG/VML | Canvas/SVG/VML |
| Grafické komponenty | 13 základních grafů | neobsahuje | 8 základních grafů | 6 základních grafů | 10 a více typů | 13 základních grafů + 3D |
| Vstupní formát dat | JavaScript API | JSON/XML | JSON | JavaScript API | JSON/XML API | JSON |
| Minimální velikost knihovny | 24 KB | 116 KB | 95 KB | 161 KB | 190 KB | 45 KB |
| Licenční omezení | Volné užití | BSD Licence | MIT Licence | MIT Licence | Komerční licence | Komerční licence |
| Otevřený kód | NE | pod licencí BSD | pod licencí MIT | pod licencí MIT | NE | pod licencí CC by-nc 3.0 |

Nejvíce možností a doplňkových funkcionalit nabízí zástupci komerčních knihoven, které kladou důraz na univerzálnost použití a cílí především na komplexní informační systémy. Pro potřeby menších systémů je však většina funkcionalit nevyužitelná a způsobuje zbytečné zvýšení režie spojené se zpracováním knihovny. Velkou nevýhodou je také uzavřený kód, který nedovoluje přizpůsobení knihovny pro vlastní potřeby a účelné použití. Menší knihovny s otevřeným kódem naopak umožňují snadnou rozšiřitelnost a jednoduchou implementaci do téměř libovolného systému.

Jako vstupní data využívají všechny uvedené knihovny strukturu ve formátu JSON, nebo specifické JavaScriptové rozhraní, které unifikuje data pro jednotné použití ve všech grafech. Knihovny Google Charts a komerční řešení AMCharts obsahuje navíc rozhraní pro přímé načítání dat ze serveru bez nutnosti externí mezivrstvy. U menších knihoven se ve většině případů musí data přeformátovat pro každý typ grafu zvlášť.

6 WEBOVÉ TECHNOLOGIE

6.1 HTML5

HTML je značkovací jazyk používaný pro tvorbu webových stránek a aplikací. Obsah se vytváří pomocí standardizovaných značek dle specifikace HTML, které slouží k označení jednotlivých sekcí a prvků na webu. Internetové prohlížeče po načtení HTML souboru analyzují jeho obsah a na základě nalezených značek vykreslí obsah stránky uživateli. HTML umožňuje na stránku vkládat obrázky, videa, formuláře a další interaktivní prvky, které poskytují prostředky pro vytváření komplexních strukturovaných dokumentů. Díky HTML značkám je možné oddělovat jednotlivé části webového dokumentu nebo je vzájemně provazovat a kontrolovat tak strukturu pro zavádění stylů.

6.2 CSS3

CSS (Cascading Style Sheets) je jazyk pro popis vizuálních atributů u elementů vytvořených pomocí značkovacího jazyka (HTML, XML...). CSS je navržen pro oddělení vizuální stránky dokumentu od jeho obsahu a stává se tak zásadní technologií pro tvorbu uživatelského rozhraní pro webové stránky a aplikace. První specifikace CSS byla publikována v roce 1996 a od té doby se stále vyvíjí. Nejnovější specifikace pro CSS3 byla vydána v roce 2015. [10]

V nejnovější specifikaci se kromě nových vizuálních stylů objevují také dynamické funkcionality, které umožňují implementovat řadu užitečných vlastností. Mezi nové funkce patří například možnost přizpůsobovat obsah dokumentu velikosti okna, nebo vytvořit přechodové animace pro změnu vizuálních stylů.

Definovat jednotlivé styly lze třemi běžně používanými způsoby:

- Přímým zápisem pomocí atributu „style“ formátovaného elementu.
- Stylopisem (stylesheet) uváděným v hlavičce stránky pomocí značky <style>.
- Externím souborem obsahujícím stylopis, na který se stránka odkazuje pomocí značky <link>. Hlavní výhodou použití „*.css“ souboru je možnost odkazovat se na tento jeden soubor na mnoha stránkách, které jsou poté formátovány ve stejném stylu.

6.3 JavaScript

JavaScript je vysokoúrovňový, multiplatformní programovací jazyk, který se nejčastěji využívá v prostředí webového prohlížeče. Je to jeden z nejjednodušších objektově orientovaných jazyků, který však zároveň poskytuje nepřeberné možnosti využití.

JavaScript byl původně vyvinut během pouhých několika dní a sloužit měl pouze jako jednoduchá pomůcka pro ovládání interaktivních prvků na webu. Vydán byl v roce 1995 společností Netscape Communications Corporation pod názvem LiveScript a svůj nynější název dostal až spolu s novou verzí prohlížeče Netscape, ve kterém získal podporu pro interakci s technologií Java. [23]

Spolu s vývojem nových prohlížečů a zaváděním vlastních standardů ze strany vývojářských společností vznikaly v počátcích JavaScriptu poměrně velké komplikace s multiplatformní implementací. Řada vývojářů nedokázala při použití JavaScriptu zaručit kompatibilitu stránek napříč všemi rozšířenými prohlížeči a jeho používání proto nebylo příliš oblíbené. Popularita JavaScriptu se prudce zvýšila s nástupem technologie AJAX a jednostránkových aplikací. V této době začalo vznikat velké množství knihoven a frameworků, které měli za úkol ulehčit práci začínajícím programátorům a přilákat pozornost z jiných prostředí. Díky své všestrannosti a jednoduchému zápisu kódu se JavaScript velmi rychle rozšířil i mimo webové prohlížeče a dnes už není problém využít JavaScript ani na straně serveru. [23]

Základní implementace JavaScriptu obsahuje standardní knihovny objektů, operátory, řídicí struktury a pevné konstanty. Tato implementace může být pro doplnění funkcionality v hostitelském prostředí rozšířena o nejrůznější objekty.

6.3.1 Použití na straně klienta

Základní implementace JavaScriptu je pro použití ve webovém prohlížeči rozšířena především o objekty pro manipulaci a řízení DOM (Document Object Model) prvků. Tato rozšíření umožňují vytvářet a upravovat obsah HTML stránky, přiřazovat jednotlivým elementům styly a obsluhovat události, které jsou vyvolány na straně prohlížeče. Mezi tyto události patří například manipulace s oknem prohlížeče, vstupní signály z periferií, nebo požadavek na obnovení obsahu okna. Program napsaný v jazyce JavaScript se v tomto případě spouští na straně klienta až po stažení ze serveru, z čehož například plynou také omezení pro práci se soubory. [23]

6.3.2 Použití na straně serveru

JavaScript se v určitých případech dá použít i na straně serveru, kdy se veškeré instrukce provedou bez nutnosti stažení obsahu stránky. Pro použití na serveru je implementace JavaScriptu rozšířena o další objekty, které aplikaci umožňují komunikovat s databází, poskytovat informace dalším aplikacím, nebo neomezeně manipulovat se soubory. Tyto interakce jsou vykonávány bez přímého zásahu uživatele a nejsou proto nutná žádná omezení.

6.3.3 ECMAScript

V roce 1990 byl JavaScript standardizován asociací ECMA a jeho standardizovaná verze byla vydána pod názvem ECMAScript. Během následujících let byla specifikace ECMAScriptu postupně rozšiřována až do nynější podoby. Nejnovější specifikace tohoto standardu je v současné době ECMAScript 6. Ze Standardu ECMAScript byly následně také odvozeny další implementace a vznikla řada zcela nových jazyků, jako například ActionScript od společnosti Adobe. [24]

II. PRAKTICKÁ ČÁST

7 VYTVOŘENÁ KNIHOVNA

Praktická část této práce se zabývá vytvořením univerzální knihovny v jazyce JavaScript, která implementuje vybrané technologie a umožní vytvářet grafické komponenty pro vizualizaci dat s ohledem na využití v informačním systému pro monitoring dopravy.

V rámci snahy o zachování co největší optimalizace byl vytvořen specifický návrh knihovny bez použití dílčích řešení a knihoven třetích stran. Celá knihovna je vytvořena pouze za pomoci technologie JavaScript a standardu HTML5.

Vytvořená knihovna má pracovní název *inCharts*, který vychází ze složeniny zkrácené podoby slov „integration, information, interactive“ a „charts“.

7.1 Struktura a popis tříd

Vytvořená knihovna se skládá z jednotlivých komponentů a globálních statických tříd jádra, které jsou společné pro všechny grafy a obsahují metody pro obecné výpočty a vytváření objektů. Jednotlivé komponenty jsou implementovány vždy jako samostatná třída, jejíž instance je přidružena ke konkrétnímu elementu canvas. Vykreslovací element canvas je vytvořen automaticky při každé konstrukci grafu.

Základní objekt *incharts* obsahuje statickou metodu *create(element, type, options)*, která na základě vložených parametrů vytvoří požadovanou komponentu a umístí ji přímo do webové prezentace.

Třída Tweener

Třída Tweener slouží pro obsluhu pohybových animací. Obsahuje statické metody pro vytváření instancí třídy Tween a hlavní animační smyčku, která se automaticky volá spolu s překreslením obsahu okna. Každá pohybová animace je po vytvoření vložena do dynamického pole *tweens[]*, které udržuje reference na aktuální stav všech probíhajících animací.

- `addTween(target, params, handlers, loop)`

pomocí předaných parametrů vytvoří instanci třídy tween a vloží ji do globálního pole animací

- `update(t)`
hlavní animační smyčka. Po zavolání se postupně projde globální pole animací *tweens* a na každé instanci animace se zavolá funkce pro její aktualizaci.
- `finish(target)`
ukončí probíhající animaci na daném objektu a aktualizuje jeho parametry do konečného stavu
- `extract(target)`
ukončí probíhající animaci na daném objektu a zachová aktuální stav jeho parametrů

Součástí statické třídy *Tweener* je kolekce přechodových funkcí, které slouží pro vytváření animovaných efektů a nelineárního pohybu.

Třída Chart

Univerzální komponenta grafu. Obsahuje obecné metody pro nastavení konfigurace, vykreslení a správu dat.

- `render()` – překreslí celý layout grafu
- `calculateFrame()` – přepočítá velikost aktivní oblasti dat
- `recalculateAxisScale()` – přepočítá rozložení dat do měřítka grafu
- `setData(data)` - vloží do komponenty nový soubor dat a automaticky přepočítá a vykreslí celý layout
- `mouseMoveHandler` - obslužná rutina pohybu kurzoru myši nad grafem

Třída BarChart

Komponenta sloupcového grafu. Dědí od obecné třídy *Charts* a obsahuje specifickou logiku vykreslování.

Třída PieChart

Komponenta segmentového grafu. Dědí od obecné třídy *Charts* a obsahuje specifickou logiku vykreslování.

Třída LineChart

Komponenta spojnicového grafu. Dědí od obecné třídy *Charts* a obsahuje specifickou logiku vykreslování.

Třída Core

Globální třída, která obsahuje všechny univerzální objekty a statické funkce pro pomocné výpočty.

Součástí třídy jsou následující pomocné metody:

- `Total()` – metoda pro součet všech číselných prvků v poli
- `Max()` – metoda pro získání největší hodnoty z pole
- `Min()` – metoda pro získání nejmenší hodnoty z pole
- `FontString(size, style, family)` – metoda pro převod parametrů vykreslení textu do css formátu v podobě textového řetězce
- `GetTextHeight(font, style, size)` – pomocná metoda pro získání výšky vykresleného textu pomocí CSS a DOM elementů
- `GetColorShade(color, factor)` – pomocná metoda pro získání světlejšího/tmavšího odstínu vložené barvy
- `ToRadians(d)` – metoda pro převod stupňů na radiány
- `ToDegrees(r)` – metoda pro převod radiánů na stupně
- `GetDecimalPlaces(num)` – vrátí celkový počet míst vloženého čísla
- `drawRoundRect` – pomocná metoda pro vykreslení obdélníků se zaoblenými rohy

Třída Tooltip

Třída pro vytvoření popisku hodnot. Objekt při konstrukci přebírá referenci na kontext, do kterého se má popisek vykreslovat. Obsahuje metodu *render()*, která se automaticky volá při každé změně atributu *text* a *angle*. Na základě těchto atributů se vždy vypočítá rozměr a popisek se překreslí.

Třída Legend

Třída pro vytvoření legendy grafu. Objekt při konstrukci přebírá referenci na kontext, do kterého se má legenda vykreslovat. Obsahuje metody *addItem(color, label)*, *removeItem(color)* a *removeAll()*, které slouží pro správu datových řad pro zobrazení v legendě.

Třída DisplayObject

Základní třída pro reprezentaci grafického objektu.

Třída Network

Pomocná třída pro komunikaci se serverem. Obsahuje statické metody pro odesílání požadavků a zpracování odpovědi.

7.2 Technologie a kompatibilita

Pro vykreslování dat byla zvolena technologie Canvas, která je blíže popsána v teoretické části této práce. Výběr technologie pro vykreslování byl proveden na základě detailního rozboru dostupných možností a srovnání jejich vlastností a výhod vzhledem k účelu použití této knihovny.

Veškerý obsah knihovny je zapouzdřen v objektu „*incharts*“. Zdrojové kódy komponent jsou dostupné v oddělených souborech, které se při distribuci sloučí a do jediného výstupního souboru. Pro práci s knihovnou stačí do hlavičky HTML dokumentu importovat distribuční soubor *incharts.js*.

Vzhledem k použité technologii pro vykreslování je knihovna optimalizována pro všechny moderní prohlížeče a Internet Explorer od verze 9.

7.3 Vytváření komponent

Vytváření komponent a struktura zadávání vstupních parametrů je navržena tak, aby práce s komponenty byla co nejvíce intuitivní a uživatelsky přívětivá. Knihovna inChart obsahuje základní statickou funkci „*create()*“, která je jednotná pro všechny grafické komponenty. Typ grafu se zadává jako vstupní parametr spolu s referencí na rodičovský element a konfigurační objekt.

Příklad:

```
incharts.create(element, type, options);
```

Po zavolání metody „*create()*“ se vytvoří nový element canvas o rozměrech rodičovského elementu, do kterého se zároveň vloží. Reference na vytvořený canvas je následně předána do konstruktoru komponenty grafu, který je volán v závislosti na zadaném typu grafu. Další parametr konstrukturu je konfigurační objekt, který se projde při inicializaci a každý prvek objektu se porovnává s dostupnými atributy grafu. Pokud graf disponuje stejným atributem, hodnota z konfiguračního objektu se uloží. V opačném případě je prvek ignorován. Tímto způsobem je možné zachovat jednotný formát konfiguračního objektu pro všechny typy grafu.

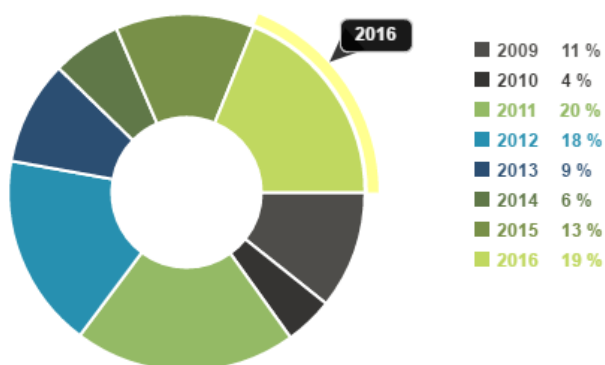
7.4 Podporované typy grafů

Knihovna inCharts podporuje celkem 4 základní typy grafů, které jsou uzpůsobeny pro zobrazování konkrétních dat z informačního systému pro monitorování dopravy. Každou z těchto komponent lze však velmi snadno konfigurovat a uzpůsobit pro téměř libovolná data a použití.

7.4.1 Segmentový graf

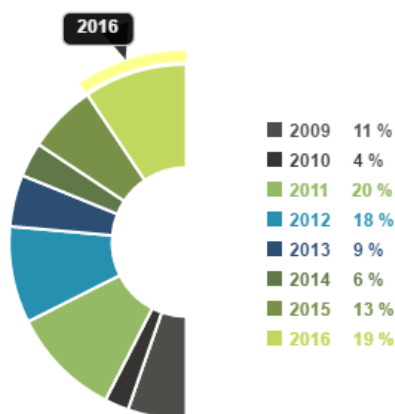
Komponenta segmentového grafu zobrazuje data v podobě kategorií o různé velikosti kruhové výseče - segmentu. V případě více datových řad se data sdružují do kategorií podle hodnoty doménového jména. U každého segmentu je možné zobrazit jeho hodnotu přímo, nebo převedenou do procentuálního zastoupení

Pomocí parametrů `options.radius` a `options.innerRadius` lze nastavit fixní velikost poloměru kruhu a jeho vnitřní části, která se při vykreslování nezobrazí. Oba údaje se zadávají v poměru k maximální velikosti, která je dána rozměry rodičovského elementu.



Obrázek 27 Varianta koláčového grafu

Při vykreslování dat není nutné využít celý rozsah 360° kruhu. Omezení rozsahu lze volit parametry `options.startAngle` a `options.endAngle`, které jsou v počátečním stavu inicializovány na hodnoty 0 a 360.

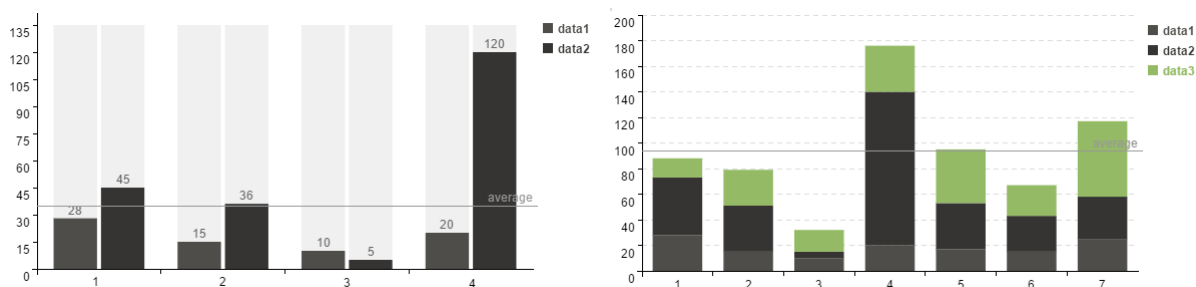


Obrázek 28 Koláčový graf s omezeným rozsahem

7.4.2 Sloupcový graf

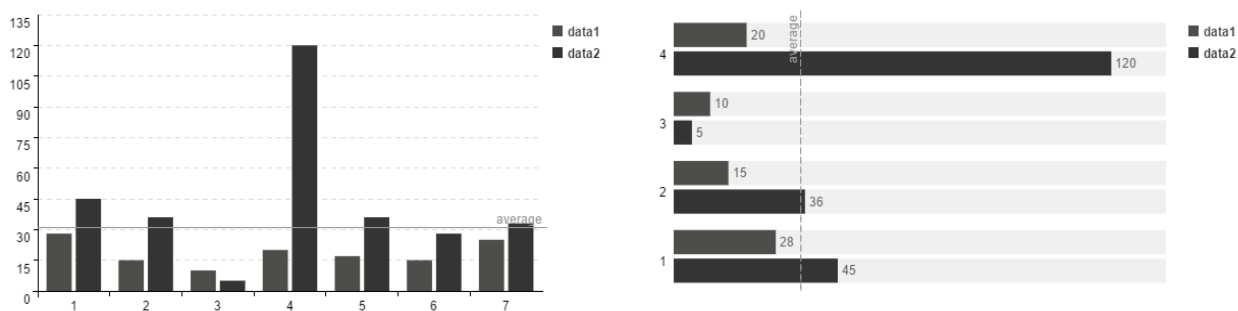
U komponenty sloupcového grafu se data vykreslují v podobě obdélníkových pruhů o různé šířce, nebo výšce, která je přímo úměrná hodnotě zobrazované veličiny. Hodnoty dat jsou sdružovány do kategorií podle doménového jména, které je vždy součástí struktury dat. Kategorie jsou rozmístěny rovnoměrně po celé délce osy v podobě statického textového řetězce.

Jednotlivé kategorie mohou obsahovat libovolné množství hodnot, které je možné shlukovat vedle sebe, nebo vrstvit do jediného sloupce. Skládání sloupců má výhodu při větším počtu hodnot v kategorii, kdy nezbývá dostatek prostoru pro zobrazení všech hodnot jednotlivě. Skládání hodnot v kategorii lze povolit parametrem `options.stacked – true/false`.



Obrázek 29 Varianty seskupování kategorií

Sloupcové grafy lze generovat ve dvou variantách – horizontální a vertikální. U horizontální varianty platí, že se názvy kategorií rozloží na ose Y a na ose X se vygeneruje rozsah hodnot. Aktuální hodnotu pak znázorňuje šířka pruhu. Vertikální vykreslování má osy prohozené a aktuální hodnota se znázorňuje pomocí výšky. Směr vykreslování lze volit parametrem `options.direction`, který lze mimo zmíněné varianty nastavit také na hodnotu „*auto*“. Při této volbě se varianta grafu zvolí automaticky podle proporcí rodičovského kontejneru.

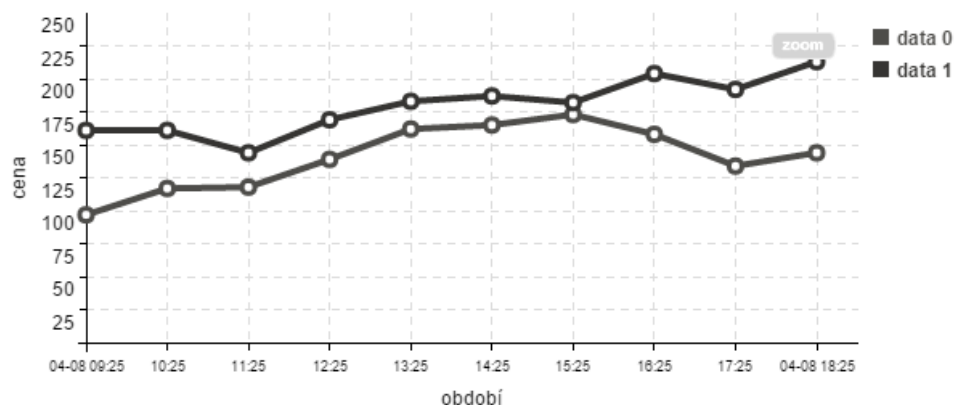


Obrázek 30 Modifikace sloupcového grafu

Rozsah na ose hodnot se generuje vždy automaticky podle vstupních dat a celkového rozměru grafu. Dalšími parametry lze zvolit fixní počet dílků, na které se má rozsah osy rozdělit, nebo například jednotku, která se má u hodnot na ose zobrazovat.

7.4.3 Spojnicový graf

Komponenta spojnicového grafu – používá se pro porovnání dvourozměrných dat. Hodnoty se vykreslují jako interaktivní body s pozicí X a Y, které jsou propojeny spojnici a reprezentují tak průběh veličiny. Do struktury zdrojových dat lze vložit libovolné množství průběhů, které se v této grafové komponentě vykreslí najednou.



Obrázek 31 Zobrazení průběhů ve spojnicovém grafu

Součástí grafu je inteligentní časová osa, která se vždy přizpůsobí hodnotám a formátu vložených dat.

Časová osa podporuje 3 základní datové typy:

- *Unix timestamp* - Formát unix timestamp obsahuje celkový počet sekund od počáteční epochy. Pomocí tohoto formátu je možné vzájemně porovnávat časové údaje a počítat jejich rozdíly.

Při sestavování časové osy se vypočítá rozdíl mezi nejnižší a nejvyšší zobrazovanou hodnotou, a porovnáním s definovanými časovými úseky se určí formát a jednotky, ve kterých se rozsah osy vygeneruje.

- *Number* - V případě obecných číselných hodnot je vypočítán rozdíl mezi nejnižší a nejvyšší hodnotou, který je rozdělen podle požadovaného počtu dílků na ose. Rozdělený rozsah je pak vydělen číselným řádem, v kterém se rozsah nachází a zaokrouhlen podle předdefinované tabulky.
- *String* - V případě hodnot ve formátu textového řetězce je časová osa vygenerována pouze z těchto řetězců, které se rozmístí rovnoměrně po celé délce osy.

7.5 Zobrazení legendy grafu

Nastavení pro zobrazení legendy je u všech komponent grafů jednotné. Umístění legendy je možné definovat pomocí parametru `options.legendState` – NONE/BOTTOM/RIGHT.

Podoba legendy je pak dána šablonou ve formě textového řetězce, který obsahuje klíčová slova ohraničená ve složených závorkách. Klíčová slova se při vytváření legendy nahradí hodnotami, které odpovídají skutečným vlastnostem grafu.

Příklad:

```
options.legendTemplate = "{name} \t {value} \t {percent}"
```

7.6 Konfigurace

Konfigurační strukturu je možné předat konstruktoru při vytváření instance grafu, nebo ji dynamicky upravovat za běhu přímo na pomocném objektu *options*, který je součástí každé komponenty grafu. Tento objekt obsahuje parametry pro vykreslování grafických prvků a celkového rozložení.

Všechny parametry na objektu *options* jsou dynamicky konfigurovatelné pomocí nadefinovaných setterů, které po zadání hodnoty automaticky zavolají funkci pro překreslení grafu. Každý graf obsahuje také veřejnou metodu *updateOptions()* pomocí které je možné dynamicky upravit více konfiguračních parametrů najednou.

Většina konfigurovatelných parametrů je pro všechny komponenty totožná. Každý typ grafu ale obsahuje také několik speciálních parametrů, které jsou určeny pouze pro dané rozložení.

7.7 Zpracování dat

Každá instance grafu obsahuje pomocný objekt *data*, který udržuje referenci na aktuální soubor dat.

Data je možné vložit do komponenty grafu staticky, nebo dynamicky pomocí odkazu na aplikační rozhraní serveru, odkud si komponenta stáhne data sama. Pro dynamické načítání

dat ze serveru má každý komponenta dostupný objektu *loader*, který udržuje referenci na serverové API a parametry dat.

Příklad statického načtení dat:

```
graf.data.set(data);
```

V případě dynamického načítání dat ze serveru lze v konfiguraci komponenty nastavit parametry dat a časový interval, s jakým se mají data aktualizovat.

Pro načítání dat ze serveru se využívá napojení na REST API, která data zpracuje dle požadavku komponenty a vrátí zpět v předepsaném formátu. Každý požadavek na server se skládá z URI adresy a doplňujících parametrů pro filtrování dat. Dotaz se odešle přes pomocnou třídu *Network*, která vytvoří XHR požadavek, odešle jej na server a následně zpracuje odpověď.

Vytvořená knihovna předepisuje formát dotazů a jejich parametry, pomocí kterých lze získávat data ze serveru.

Každý dotaz musí povinně obsahovat časový interval, doménu hodnot a požadovanou metriku. Mimo povinné parametry lze do dotazu přidat nepovinné parametry, jako jsou filtry, nebo omezení počtu hodnot.

Jednotlivé parametry a formát dotazů je navržen tak, aby co nejvíce odpovídal struktuře SQL dotazů na databázi, do kterých se parametry po odeslání na server přegenerují. Parametry lze do instance grafu zadávat přes pomocný objekt *loader* jako jeho atributy, nebo pomocí konfiguračního souboru.

Příklad dynamického načtení dat:

```
graf.loader.load(uri, from, to, domain, metrics, count, scale, filters);
```

from – začátek intervalu, ve kterém se nacházejí požadovaná data (*unix timestamp*)

to – konec intervalu, ve kterém se nacházejí požadovaná data (*unix timestamp*)

domain – obor hodnot, podle kterého se data seskupují. U spojnicového a sloupcového grafu představuje obor hodnot časovou osu, nebo kategorie dat.

(years, type, city atd.)

metrics – název hodnot (count, budget, speed atd.)

count – maximální počet vrácených hodnot

scale – měřítko, ve kterém jsou data vzorkována (days, hours, months atd.)

filters – požadované filtry pro obor hodnot, nebo zvolenou metriku

Data jsou přijímána ve formátu JSON (*JavaScript Object Notation*), který obsahuje univerzální strukturu pro všechny podporované typy grafů. Struktura dat je vždy dvouúrovňová, přičemž první úroveň obsahuje jednotlivé datové řady, a druhá již samotné parametry dat.

```
{
  "řada_a": [
    {
      "datetime": 1460100347,
      "count": 28,
      "label": "a0"
    },...
  ],
  "řada_b": [
    {
      "datetime": 1460100347,
      "count": 42,
      "label": "b0"
    },...
  ],...
}
```

Obrázek 32 Struktura vstupních dat

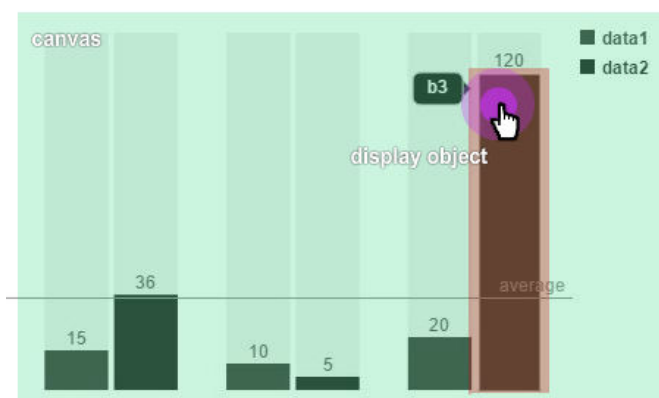
7.8 Interaktivita

Vytváření interaktivního obsahu na webu se provádí pomocí odchyťování událostí na jednotlivých elementech. V případě vykreslování pomocí technologie canvas se však používá pouze jediný element a pro každý interaktivní prvek je proto nutné parametry události přepočítat. Pro tento účel byl vytvořen systém interaktivních objektů a manažer událostí, který veškerou interaktivitu spravuje.

Základním prvkem je třída `DisplayObject`, od které všechny zobrazované objekty s interakcí dědí. Tato třída nese informace o poloze, rozměrech a stavech daného objektu.

Každý `DisplayObject` na sobě může mít zaregistrováno neomezené množství událostí, které jsou vždy spojeny s konkrétní návratovou funkcí.

Při odchycení události kurzoru na elementu canvas jsou postupně procházeny všechny dostupné objekty v okolí s povolenou interakcí, a na každém z nich je zavolána metoda pro kontrolu, zda se kurzor nachází právě nad tímto objektem. V případě kladné návratové hodnoty je do objektu odeslána reference odchycené události, na základě které se z objektu zavolá přidružená funkce.



Obrázek 33 Systém display objektů

Knihovna má v základní podobě implementované následující statické události:

- kliknutí na objekt představující položku dat (`incharts.ITEM_CLICK`)
- kliknutí na celý graf (`incharts.GRAPH_CLICK`)
- změna konfigurace (`incharts.UPDATE`)
- změna dat (`incharts.DATA_UPDATE`)

Kromě předdefinovaných událostí lze každému grafu přiřadit návratovou funkci k jakékoliv další změně. Přiřazení akce události se provádí pomocí metody `on`, která je dostupná na každé instanci grafu.

Příklad:

```
graf.on(incharts.ITEM_CLICK, callback);
```


7.9 Animace

Knihovna inCharts obsahuje dva nezávislé managery pro podporu animací. Prvním z nich je statická třída *Animator*, která slouží pro snímkové animace. Každá snímková animace musí být před použitím v této třídě zaregistrována jednoznačným identifikátorem, kterému se přiřadí zdrojový soubor a parametry snímků. Pro registraci slouží metoda *Animator.register(id, source, frames)*. Animaci lze po zaregistrování přehrát na kterémkoliv objektu typu *Sprite*, který slouží jako účelový kontejner pro vykreslení bitmapy.

Při snímkové animaci dochází k posouvání offsetu na zdrojové bitmapě v periodických intervalech, které jsou určeny požadovanou délkou animace. Zaregistrovanou animaci lze na objektu přehrát zavoláním funkce *Animator.play(sprite, id, time)*.

Druhý manager slouží pro podporu pohybových animací a reprezentuje jej statická třída *Tweener*. Ta umožňuje animovat libovolné parametry na zadaném objektu se zvolenou rychlostí a obsluhovat události začátku, změny, nebo dokončení animace. Zavoláním metody *addTween* se přidá objekt do fronty, která se aktualizuje při každém obnovení okna. Aktuální změny parametrů objektu jsou dpočítávány jako rozdíl počáteční a požadované hodnoty vynásobený hodnotou přechodové funkce v daném čase.

Příklad:

```
Tweener.addTween(object, {param:value, ...}, {time: t, transition: id, callbacks..})
```

Třída *Tweener* obsahuje celkem 32 přechodových funkcí, které je možné pro animace využít. Přechodovou funkci pro vykreslování grafů lze měnit v konfiguračním souboru.

Příklad přechodových funkcí:

easeInQuad => $c * (t/d) * t + b$ – plynulý náběh

$\text{easeOutQuad} \Rightarrow (-c) * (t/d) * (t-2) + b$ – plynulé dokončení

$\text{easeInExpo} \Rightarrow c * \text{Math.pow}(2, 10 * (t/d - 1)) + b$ – exponenciální průběh

$\text{easeInSine} \Rightarrow -c * \text{Math.cos}(t/d * (\text{Math.PI}/2)) + c + b$ – sinusový průběh

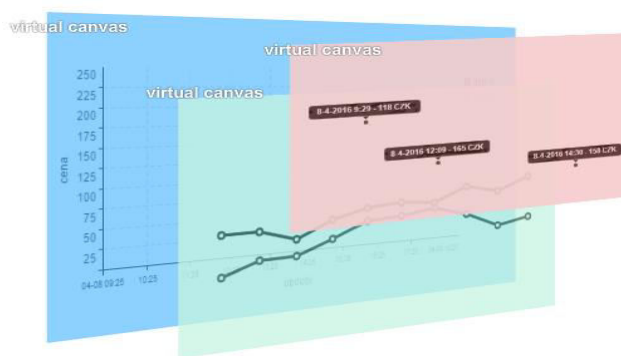
t: aktuální čas, b: počáteční hodnota, c: koncová hodnota, d: délka animace

Obě animační třídy jsou řízené metodou *requestAnimationFrame()*, která v pravidelných intervalech (60 /s) odešle prohlížeči požadavek na překreslení obsahu stránky a prostřednictvím jednotlivých managerů zavolá metodu *update* na všech probíhajících animacích.

7.10 Překreslování

Některé prvky se v grafech nepřekreslují tak často jako ostatní. Například při animování dat je potřeba překreslovat jen objekty reprezentující samotná data, ale časová osa a dispoziční řešení zůstává statické.

Vykreslování obsahu se proto provádí do několika bitmapových úrovní, které jsou rozděleny podle frekvence vykreslování. Každá bitmapová úroveň představuje samostatný virtuální element canvas, který není reálně umístěn do obsahu stránky. Po překreslení dané úrovně je do globální proměnné uložena reference, na základě které se při aktualizaci grafu vykreslí bitmapový obsah jednotlivých úrovní do hlavního plátna.



Obrázek 34 Vrstvení bitmapových úrovní

7.11 Adaptivita

Prizpůsobovat velikost elementu canvas je v HTML5 možné hned dvěma způsoby. Prvním způsobem je možnost nastavit dynamickou výšku a šířku elementu pomocí css stylu. Definováním dynamických poměrů se bude element vždy přizpůsobovat velikosti okolí, nicméně obsah elementu zůstane vykreslený v původní velikosti. V takovém případě může dojít k deformaci obrazových dat a ztrátě kvality.

Pro přizpůsobování velikosti grafů je výhodnější zvolit metodu druhou, která funguje na principu kompletního překreslení dat při každé změně velikosti rodičovského kontejneru. Po odchycení události, která indikuje změnu velikosti kontejneru, jsou upraveny atributy *width* a *height* samotného elementu, čímž se změní nativní velikost plátna. Zavoláním funkce *recalculate()* na instanci grafu poté dojde ke kompletnímu přepočtu všech bitmapových vrstev a jejich následnému překreslení do nové velikosti.

Při vykreslování grafu do dashboardu je aktuální velikost elementu canvas dána velikostí jeho rodičovského elementu, který se již přizpůsobuje velikosti okna standardně podle přiřazeného css stylu.

7.12 Konstrukce dashboardu

Základní uplatnění má knihovna při vytváření webových dashboardů. Pro ukázkou tvorby dashboardů byla vytvořena pomocná komponenta „dashboard“ a série css stylů, které je možné libovolně upravovat. Dashboard se vytváří jako dynamický, nebo statický tabulkový layout, který má předdefinované rozložení buněk. Každá buňka je rozdělena na

tři základní, oddělené kategorie – ovládací prvky, hlavička, obsah. Kategorie jsou označeny názvem třídy „*ic-controls*“, „*ic-header*“ a „*ic-content*“.

Po načtení stránky se postupně projde celá struktura označená třídou „*ic-dashboad*“ a do každého nalezeného elementu, který je vyčleněn pro obsah, je na základě identifikátoru automaticky vygenerována komponenta grafu, která se spáruje s ovládacími prvky obsaženými uvnitř kategorie „*ic-controls*“



Obrázek 35 Ukázka rozložení dashboardu

Buňky dashboardu i jejich jednotlivých částí lze neomezeně stylovat přidáváním, nebo úpravou stávajících stylů.

8 HTML DOKUMENTACE

Kromě samotné knihovny pro generování a správu grafů byla vytvořena také jednoduchá HTML prezentace, která slouží jako dokumentace a návod pro správnou implementaci knihovny.

V prezentaci je uveden kompletní postup od přidání knihovny do struktury HTML, až po samotné vytváření grafů a jejich parametry. U každého typu grafu je popsána struktura třídy pro jeho vytváření, konfigurační parametry a soubor dat, se kterými daná komponenta pracuje. Kromě základního popisu a poznámek existuje ke každému typu grafu v rámci dokumentace také ukázka praktického použití.

Webová prezentace je rozčleněna do úvodu a jednotlivých kategorií pro každý typ grafu, který je knihovnou podporovaný. Pro každou kapitolu je vytvořena samostatná podstránka, mezi kterou je možné plynule přecházet prostřednictvím navigačního menu. Pro přechod mezi stránkami se využívá technologie AJAX a pro správné zobrazení celé prezentace je proto nutné mít aktivovaný JavaScript.

Na vytváření webové prezentace byl použit responzivní framework Bootstrap, který obsahuje sadu nástrojů pro usnadnění vytváření uživatelského prostředí a ošetřuje správné zobrazení obsahu napříč prohlížeči.

ZÁVĚR

Tato práce se zabývá možnostmi vizualizace dat ve webovém prostředí. Cílem práce bylo podrobně prostudovat a porovnat nástroje, které jsou v současné době běžně využívány pro práci s grafy, a vytvořit návrh vlastní implementace v podobě JavaScriptových komponent pro specifické použití u informačního systému pro monitorování dopravy.

V teoretické části této práce jsou popsány dostupné možnosti, technologie a postupy běžně využívané v oblasti vizualizace dat. Součástí teoretické části práce je také podrobný popis existujících knihoven a porovnání jejich vlastností.

Na základě srovnání dostupných technologií byl v rámci praktické části vytvořen návrh vlastního řešení, které je postavené na technologii HTML5 a JavaScript bez využití dílčích komponent a knihoven třetích stran. Díky nezávislé implementaci je možné knihovnu lépe přizpůsobit pro specifická data a docílit maximální optimalizace při vykreslování dat.

V praktické části práce je dále popsána celková vnitřní struktura vytvořené knihovny, implementované funkce a dostupné možnosti, které lze při práci s knihovnou využít. Pro zjednodušení práce s knihovnou byla vytvořena jednoduchá programátorská dokumentace, která je dostupná ve formě HTML prezentace a kromě strukturovaného popisu tříd obsahuje i praktické příklady použití.

Vytvořená knihovna je navržena tak, aby splňovala všechny požadavky na použití v moderních informačních systémech a prostředích s výstupem pro zpracování dat. Vzhledem k použité technologii a implementované funkci automatické adaptace je možné komponenty grafů využít i samostatně na libovolné platformě s podporou technologie HTML5. V současné době je knihovna stále ve stádiu vývoje a některé funkce proto nejsou zcela podporované.

Díky použité technologii a přehledné struktuře kódu je možné knihovnu snadno rozšířit o další komponenty, nebo libovolnou funkcionalitu pro přizpůsobení konkrétnímu systému.

SEZNAM POUŽITÉ LITERATURY

- [1] Pojem vizualizace. In: ČVUT - geologie [online]. 2006 [cit. 2016-04-12]. Dostupné z: http://geo3.fsv.cvut.cz/~soukup/dip/krejny/2_pojem_vizualizace.htm
- [2] ČERNÝ, Michal. Vizualizace dat: Jak odhalit utajené souvislosti. In: VTM E15 [online]. 2012 [cit. 2016-04-12]. Dostupné z: <http://vtm.e15.cz/vizualizace-dat-jak-odhalit-utajene-souvislosti>
- [3] Types of Graphs: Graph types - definitions and examples [online]. Types of Graphs., 2016 [cit. 2016-04-13]. Dostupné z: <http://www.typesofgraphs.com/>
- [4] MURRAY, Scott. Interactive data visualization for the web. Sebastopol, CA: O'Reilly Media, 2013. ISBN 14-493-6108-0.
- [5] SPENCE, Ian. No Humble Pie: The Origins and Usage of a Statistical Chart [online]. University of Toronto, 2005 [cit. 2016-04-15]. Dostupné z: <http://www.psych.utoronto.ca/users/spence/Spence%202005.pdf>
- [6] PILGRIM, Mark. HTML5: up and running. 1st ed. Sebastopol, CA: O'Reilly, 2010, xii, 205 p. ISBN 978-0-596-80602-6.
- [7] FULTON, Steve a Jeff FULTON. HTML5 canvas. Second edition. Farnham: O'Reilly, 2013, xx, 726 pages. ISBN 14-493-3498-9.
- [8] W3C[online]. The World Wide Web Consortium [cit. 2016-05-1]. Dostupné z: <https://www.w3.org/>
- [9] ŽÁRA, Ondřej. JavaScript: programátorské techniky a webové technologie. 1. vydání. Brno: Computer Press, 2015, 180 stran. ISBN 978-80-251-4573-9.
- [10] LUBBERS, Peter, Brian ALBERS a Frank SALIM. HTML5: programujeme moderní webové aplikace. Vyd. 1. Brno: Computer Press, 2011, 304 s. ISBN 978-80-251-3539-6.
- [11] GLENN HOSTETLER, Sandor Hasznos a ARTWORK BY CHRISTINE HERON. Web service and SOA technologies: protect your project and career by understanding the common mistakes. 2nd ed. Denver, Colo.: Practicing Safe Techs, 2009. ISBN 09-823-0370-X.
- [12] RICHARDSON, Leonard a Sam. RUBY. RESTful web services: protect your project and career by understanding the common mistakes. 2nd ed. Farnham: O'Reilly, c2007. ISBN 05-965-2926-0.

- [13] MALÝ, Martin. REST: architektura pro webové API. Zdrojak.cz [online]. 2009 [cit. 2016-04-17]. Dostupné z: <https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/>
- [14] HASSMAN, Martin. JSON : jednotný formát pro výměnu dat. Zdrojak.cz [online]. Devel.cz Lab s.r.o., 2008 [cit. 2016-05-13]. ISSN 1803-5620. Dostupné z: <https://www.zdrojak.cz/serialy/json-pro-vymenu-dat-na-webu/>
- [15] Úvod do JSON [online]. 1999 [cit. 2016-04-20]. Dostupné z: <http://www.json.org/json-cz.html>
- [16] Charts: Interactive charts for browsers and mobile devices. [online]. Google, 2016 [cit. 2016-05-2]. Dostupné z: <https://developers.google.com/chart/>
- [17] D3: Data-Driven Documents [online]. Mike Bostock, 2015 [cit. 2016-05-05]. Dostupné z: <https://d3js.org/>
- [18] Flot: Attractive JavaScript plotting for jQuery [online]. IOLA and Ole Laursen, ©2007-2014 [cit. 2016-05-05]. Dostupné z: <http://www.flotcharts.org/>
- [19] Chart.js [online]. 2016 [cit. 2016-05-05]. Dostupné z: <http://www.chartjs.org/>
- [20] AMCHARTS [online]. amcharts.com, 2015 [cit. 2016-05-05]. Dostupné z: <https://www.amcharts.com/>
- [21] HIGHCHARTS [online]. Highcharts, 2016 [cit. 2016-05-05]. Dostupné z: <http://www.highcharts.com/>
- [22] FusionCharts: JavaScript charts for web & mobile [online]. InfoSoft Global Private Limited, ©2002-2016 [cit. 2016-05-06]. Dostupné z: <http://www.fusioncharts.com/>
- [23] STEFANOV, Stoyan a Kumar Chetan SHARMA. Object-oriented JavaScript. 2nd ed. Birmingham, UK: Packt Publishing, 2013, viii, 363 p. ISBN 9781849693127.
- [24] ECMAScript 6: compatibility table. Compat ES [online]. 2016 [cit. 2016-05-05]. Dostupné z: <https://kangax.github.io/compat-table/es6/>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|------|--|
| HTML | HyperText Markup Language. |
| API | Application Programming Interface. |
| 2D | Two-dimensional space. |
| CSS | Cascading Style Sheets. |
| DOM | Document Object Model. |
| XML | Extensible Markup Language. |
| SVG | Scalable Vector Graphics. |
| SOAP | Simple Object Access Protocol. |
| UDDI | Universal Description Discovery and Integration. |
| WSDL | Web Services Description Language. |
| AJAX | Asynchronous JavaScript and XML. |
| HTTP | Hypertext Transfer Protocol. |
| REST | Representational State Transfer. |
| URI | Uniform Resource Identifier. |
| CRUD | Create, Read, Update, Delete |
| JSON | JavaScript Object Notation |
| VML | Vector Markup Language |
| PDF | Portable Document Format |

SEZNAM OBRÁZKŮ

| | |
|--|----|
| Obrázek 1 Příklad spojnicového grafu [3] | 12 |
| Obrázek 2 Příklad sloupcového grafu [3] | 13 |
| Obrázek 3 Příklad segmentového grafu [3] | 14 |
| Obrázek 4 Příklady transformace obrazu..... | 17 |
| Obrázek 5 Uložení a obnova stavu plátna | 18 |
| Obrázek 6 Diagram překreslovací smyčky | 18 |
| Obrázek 7 Varianty výplně | 20 |
| Obrázek 8 Vyplňování vnitřních oblastí | 20 |
| Obrázek 9 Styl zalomení čáry | 21 |
| Obrázek 10 Varianty ostrého zalomení | 21 |
| Obrázek 11 Pozice pixelů v elementu Canvas | 21 |
| Obrázek 12 Rozložení a rozměr textu..... | 22 |
| Obrázek 13 Vykreslování bitmapy | 23 |
| Obrázek 14 Struktura obrazových dat | 23 |
| Obrázek 15 Ukázka obrazových vrstev | 25 |
| Obrázek 16 Výkonové srovnání vykreslovacích technologií | 27 |
| Obrázek 17 struktury formátu JSON [15]..... | 31 |
| Obrázek 18 Struktura Google Charts vizualizace | 32 |
| Obrázek 19 Vytvoření grafu pomocí knihovny Google Charts | 33 |
| Obrázek 20 Struktura datově řízeného dokumentu | 33 |
| Obrázek 21 Vytvoření grafu pomocí knihovny D3 | 34 |
| Obrázek 22 Vytvoření grafu pomocí knihovny Flot Charts | 35 |
| Obrázek 23 Vytvoření grafu pomocí knihovny ChartJS | 36 |
| Obrázek 24 Ukázka komplexního výstupu knihovny AMCharts | 36 |
| Obrázek 25 Vytvoření grafu pomocí knihovny AMCharts | 37 |
| Obrázek 26 Vytvoření grafu pomocí knihovny HighCharts..... | 38 |
| Obrázek 27 Varianta koláčového grafu | 49 |
| Obrázek 28 Koláčový graf s omezeným rozsahem | 50 |
| Obrázek 29 Varianty seskupování kategorií | 50 |
| Obrázek 30 Modifikace sloupcového grafu..... | 51 |
| Obrázek 31 Zobrazení průběhů ve spojnicovém grafu | 52 |
| Obrázek 32 Struktura vstupních dat..... | 55 |

| | |
|--|----|
| Obrázek 33 Systém display objektů..... | 56 |
| Obrázek 34 Vrstvení bitmapových úrovní..... | 59 |
| Obrázek 35 Ukázka rozložení dashboardu | 60 |

SEZNAM TABULEK

| | |
|---|----|
| Tabulka 1 Srovnání vybraných knihoven | 39 |
|---|----|

SEZNAM PŘÍLOH

PŘÍLOHA P I: Obsah vloženého CD-ROM disku

PŘÍLOHA P I: OBSAH VLOŽENÉHO CD-ROM DISKU

Obsah disku:

- Elektronická verze diplomové práce
- Zdrojové soubory knihovny
- Distribuce knihovny
- HTML Dokumentace