Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

# Bakalářská práce

# Implementace testovacího nástroje pro časovou osu

Plzeň 2019 Michal Fiala

**Místo této strany bude zadání práce.**

# Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 1. května 2019

Michal Fiala

# Abstract

In this work is tool for timeline visualization enhanced with a new format of data, to which the renderers are created and described. In addition, this tool is extended to include band unification and user activity tracking. The effectiveness of the new version of the tool is tested and compared to the previsou version.



# Abstrakt

V této práci je nástroj pro časovou osu rozšířen o nový formát dat, ke kterému jsou vytvořeny a popsány příslušné renderery. Tento nástroj je navíc rozšířen o fuknčnost sjednocování pásů a modul pro sledování aktivity uživatele. Je zde otestována efektivita nové verze nástroje a porovnána s předchozí verzí.

# Poděkování

Rád bych poděkoval vedoucímu bakalářské práce Ing. Richardu Lipkovi, Ph.D., za vstřícnost a cenné rady. Dále děkuji všem testerům Zuzaně Kubincové, Ing. Ivanu Gruberovi, Bc. Jakubu Vaštovi, Jakubu Jirkovskému, Jiřímu Šimánovi, Karolíně Podstatné, Kristýně Gruberové, Michaele Fialové, Bc. Věře Fialové, Zdeňku Vildovi.

# Obsah

# 1 Úvod

V dnešní době se můžeme setkat s vizualizací dat téměř každodenně a to například v médiích, ve vzdělání nebo na internetu, kde hledáme nějakou informaci, či vytváříme rozvrh na další týden. I vytváření rozvrhu lze považovat za znázorňování informace v čase. Z těchto vizualizací získáváme data, se kterými dále pracujeme.

Obvykle však pracujeme se základní vizualizací, jakou jsou například tabulky nebo jednoduché grafy. Problém nastává tehdy, kdy chceme zobrazit objemná data a jejich vztahy. Dokázat znázornit rozsáhlá data a jejich spojitosti tak, abychom mohli efektivně s těmito daty pracovat, je nelehký úkol. Je to z toho důvodu, že tak velké množství dat nedokáže náš mozek zpracovat najednou. Avšak analýza takových dat je velmi užitečná, například při tvorbě politických průzkumů, v marketingu a dalších odvětvích.

Tato bakalářská práce bude navazovat na vytvořený nástroj pro zobrazování dat v časové ose viz [6] a dále se zaměří na to, jak efektivně v tomto nástroji vizualizovat historická data, aby z nich bylo patrné, jaký je mezi nimi vztah a v jakém časovém sledu se události nacházely. Tato práce se zaměří i na rozšíření uživatelského prostředí tohoto nástroje, které bude pomáhat uživateli práci s daty.

Výslednou práci jsem nakonec otestoval oproti předchozí verzi, která je nyní již funkční a kterou jsem vylepšil o efektivnost zobrazení. Testování proběhlo na vybrané množině lidí a porovnávala se rychlost, snadnost a průběh nalezení odpovědi na dané otázky týkající se zobrazené datové množiny.

Výsledkem této práce je nová verze nástroje, která je vylepšena o efektivní zobrazení a která umí lépe zobrazovat stávající data.

# 2 Úvod do vizualizace

Tato bakalářská práce se zaobírá především vizualizací dat v časové ose, aby bylo dále z textu zřejmé, co jaký pojem znamená, je třeba popsat pár základních pojmů.

## 2.1 Vizualizace

Slovo vizualizace je široce používaný pojem, který vyjadřuje postup, při kterém popisujeme nějaké informace, data nebo hodnoty pomocí obrazu. Vizualizace má dlouhou a úctyhodnou historii, kořeny této oblasti sahají až do rané tvorby map a vizuálního zobrazení [4].

Cílem této oblasti výzkumu je integrace vynikajících schopností lidského vizuálního vnímání a enormní výpočetní síly počítačů, které uživatelům pomáhají analyzovat, porozumět a pracovat s těmito daty. Aby bylo dosaženo tohoto cíle, je třeba splnit následující tři kritéria:

- Vyjádřitelnost
- Efektivita
- Vhodnost

Vyjádřitelnost odkazuje na požadavek přesně zobrazovat informace obsažené v datech. Efektivita především zvažuje, do jaké míry se vizualizace zaměřuje na kognitivní schopnosti lidského mozku. Cílem je získat intuitivně rozpoznatelná a reprezentovatelná data. Vhodnost zahrnuje vyvážení ceny zobrazení vůči rychlosti dosažení úkolu [11]. Pokud chceme například data v reálném čase vizuálně obohatit, musíme zvolit kompromis tak, aby rychlost dosažení úkolu byla pro uživatele přijatelná.

## 2.2 Časová osa

Časová osa je množina událostí uspořádaných chronologicky podle jejich časového zasazení. Je to typicky grafický návrh, který zobrazuje dlouhý pruh označený daty vedle sebe a zobrazení samotných událostí nad tímto pruhem. Časové osy lze dělit do několika základních typů [10].

**Typy časových os:**

- Textové
- Číselné
- Interaktivní, s možností přiblížení

Jako řešení časové osy v přecházející práci [6] byl využit třetí typ.

## 2.3 Řešení zobrazení časové osy

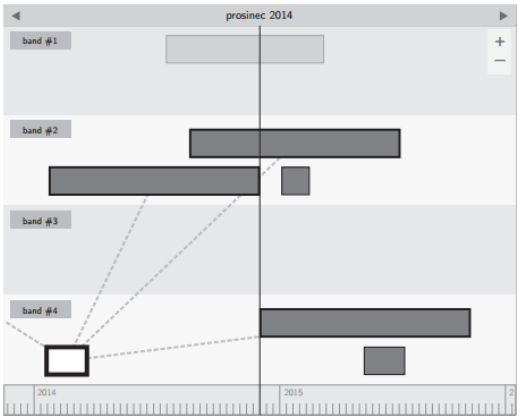
V roce 2015 byl [6] publikován nástroj pro zobrazení časové osy, na který budu touto prací navazovat. V této sekci tento nástroj stručně popíši.

### 2.3.1 Implementace

Tato časová osa byla implementována pro webové stránky a její využití je tedy nezávislé na platformách. Pro zobrazování slouží elementy HTML a pro práci s těmito elementy jsou použity javascriptové soubory. Celý popis implementace nástroje můžete najít zde [6].

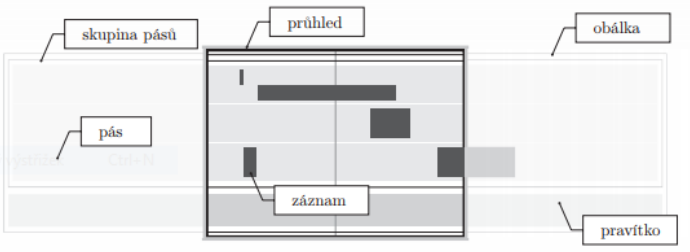
### 2.3.2 Uživatelské rozhraní

Uživatelské prostředí je složeno z časové osy, která je umístěna dole. Zbytek okna tvoří zobrazení historických událostí viz Obrázek 2.1.




Obrázek 2.1: Uživatelské rozhraní [6].

Obrázek 2.1 zobrazuje pouze část časové osy. Celý kontext můžeme vidět na Obrázku 2.2, kde je tučně zvýrazněno aktuálně zobrazené okno v kontextu s časovou osou.




Obrázek 2.2: Celý kontext okna [6].

### 2.3.3 Základní vlastnosti

Měřítko osy je závislé na úrovni přiblížení osy, v nástroji jsou úrovně definované v tabulce 2.1.

| Úroveň | Hodnota | Úroveň | Hodnota |
|---|---|---|---|
| 1 | 10 let | 14 | 3 hodiny |
| 2 | 5 let | 15 | 1 hodina |
| 3 | 3 roky | 16 | 30 minut |
| 4 | 1 rok | 17 | 15 minut |
| 5 | 6 měsíců | 18 | 10 minut |
| 6 | 3 měsíce | 19 | 5 minut |
| 7 | 1 měsíc | 20 | 1 minuta |
| 8 | 14 dní | 21 | 30 sekund |
| 9 | 1 týden | 22 | 15 sekund |
| 10 | 3 dny | 23 | 10 sekund |
| 11 | 1 den | 24 | 5 sekund |
| 12 | 12 hodin | 25 | 1 sekunda |
| 13 | 6 hodin | 26 | 0,5 sekundy |

Tabulka 2.1: Úrovně přiblížení

Čas je reprezentován ve formátu ISO 8601 a jeho úskalí jsou podrobně rozebrána v práci [6].

Veškeré reakce na události v nástroji lze sledovat ve dvou třídách. Základní reakce na události s nástrojem jako je přiblížení, posun v ose a další jsou definovány v *App.js*. Události týkající se dat jsou obsluhovány a definovány v *Timeline.js*.

### 2.3.4 Pás (band)

Pás je komponenta, do které se zařazují záznamy podle určitého typu. Každý pás má jeden jediný možný typ dat. Tato komponenta zajišťuje například schování záznamů pokud nejsou v zorném poli časové osy, dále zajišťuje metodu pro zabránění kolize záznamů.

### 2.3.5 Položka pásu (band item)

Položka pásu slouží pro abstrakci reálného záznamu. Tato komponenta uchovává skutečná data a funguje jako jejich reprezentace. O vykreslení této položky se stará objekt zvaný *renderer* [6].

### 2.3.6 Renderer

Tato komponenta určuje jak má položka pásu nebo hrana grafu vypadat. Zajišťuje vytvoření HTML obálky, která je uložena v položce pásu. Obálka pro položku pásu je tvořena pomocí metody *renderContinuous* nebo *renderMoment*. Pro vytvoření HTML obálky položky pásu, která má dobu trvání, slouží metoda *renderContinuous* a vypadá následovně:

```
1  renderContinuous : function(item) {
2          var element = new $("<div>")
3          .addClass(this.INTERVAL_CLASS)
4          .css({
5           "background-color" : this._color.getRgba(),
6           "border-color" : this._color.darken(20).getRgba()
7          });
8
9          return element;
10 }
```

Tato metoda vytvoří a vrací div element, kterému přiřadí css třídu *INTERVAL_CLASS*. Dále tomuto elementu nastaví barvu pozadí a ohraničení.

Pro vytvoření HTML obálky položky pásu, která vyjadřuje moment, slouží metoda *renderMoment* a vypadá následovně:

```
1 renderMoment : function(item) {
2         var element = new $("<div>")
3         .addClass(this.MOMENT_CLASS)
4         .css({
5          "background-color" : this._color.getRgba(),
6          "border-color" : this._color.darken(20).getRgb()
7         });
8         return element;
9 }
```

Tato metoda vytvoří a vrací div element, kterému přiřadí css třídu *MOMENT_CLASS*. Dále tomuto elementu nastaví barvu pozadí a ohraničení.

Renderer dále obsahuje metody pro úpravy rozměrů a pozice této HTML obálky. Tyto algoritmy jsou blíže popsány v *BandItemRenderer.js*.

### 2.3.7 Základní interakce

V této části je popis základních interakcí implementované časové osy viz [6].

#### Posun osy

Způsob jakým se uživatel může posouvat po ose je drag&drop. Jde o uchopení stisknutím levého tlačítka myši a následné tažení doleva nebo doprava. Po upuštění časová osa zobrazí data ve zvoleném období.

#### Změna úrovně přiblížení

Uživatel může časovou osu přibližovat a oddalovat použitím tlačítka + a -. Při této manipulaci s osou nedochází k žádnému posunu, středový čas zůstává zachován. Pokud použije uživatel k přiblížení kolečko myši, dojde k jinému chování. Tehdy je totiž brána v potaz pozice kurzoru a osa se přiblíží k myší určenému místu.

#### Zaostření

Tato funkce kombinuje výše popsané interakce. Průhled se posune tak, aby zaostřený objekt byl v jeho středu, a dále se přiblíží na zaostřený objekt tak, aby byl zobrazen celý a v co největších rozměrech.

# 3 Analýza možných řešení

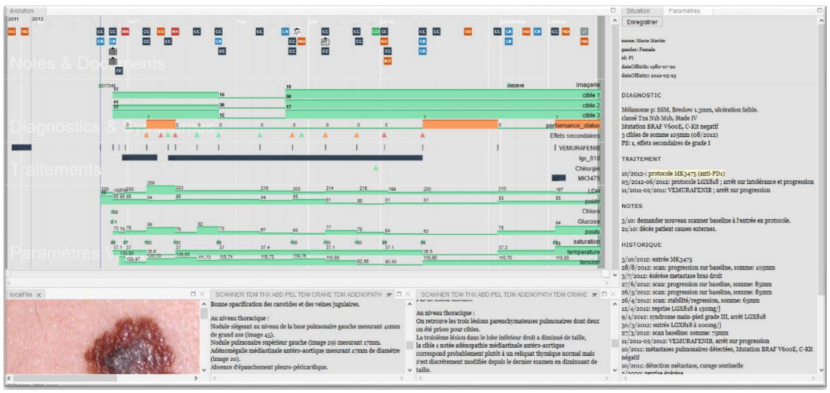
V této části jsou popsány postupy pro změnu zobrazení časové osy tak, aby se zajistila co největší efektivita práce s daty. Z analytické části poté vyberu postupy, které budu implementovat do stávajícího projektu časové osy. Implementované řešení později otestuji na testovací skupině lidí tak, abych zjistil, zda-li mělo řešení nějaké přínosy pro orientaci a vyhledávání v datech.

## 3.1 Lifeline

První ze způsobů zefektivnění zobrazených dat je Lifeline. Využívá se především ve zdravotnictví, kde ho využívají pro zobrazení zdravotní historie pacienta.

Postup pro zobrazení dat vypadá následovně. Obrazovka je horizontálně rozdělena do regionů, kde je každý region obarven jinou barvou a reprezentuje určitý typ dat. Data jsou poté zařazena do regionu odpovídajícího typu dat. Určité vizuální podněty jako jsou barvy, tloušťky čar nebo zvýraznění označují význam určitých informací a vztahy mezi událostmi.

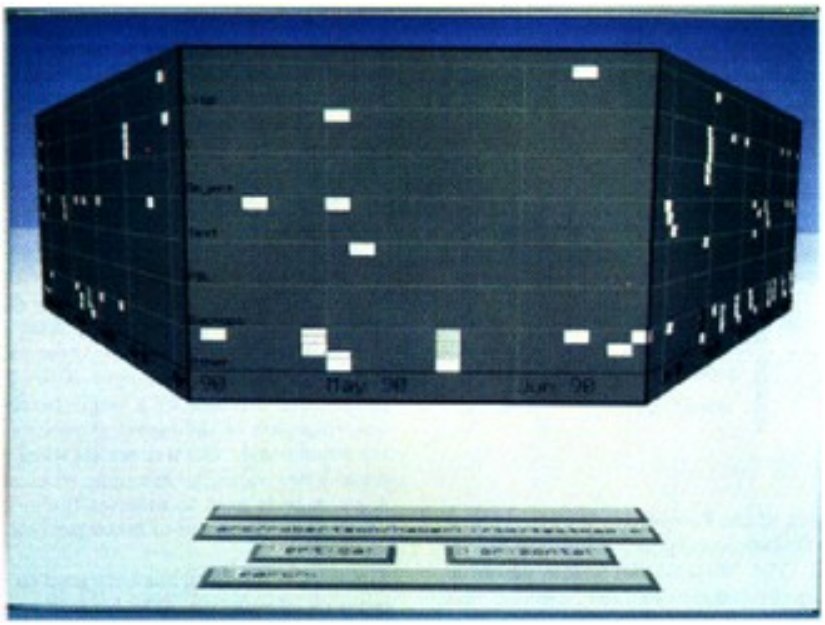
Tento způsob přináší lepší přehlednost v datech zvýrazněním důležitých dat a lepším filtrováním dat podle regionů. Nevýhodou je, že dokáže zobrazovat pouze malé množství dat [9].



Obrázek 3.1: Ukázka lifeline [1].

## 3.2 Perspektivní zeď

Tento způsob využívá zkreslovacích technik pro zobrazení velkého množství dat na obrazovce, zajišťující větší efektivitu prohledávání. Tato technika lépe využívá dostupný prostor na obrazovce integrací detailního a kontextového okna. Řešením je zkreslit zobrazení tak, že data mimo hlavní zorné pole jsou zmenšena a zkreslena. Tato technika se jinak nazývá "Rybí oko". Nevýhodou je, že perspektivní zeď dokáže zobrazovat pouze jeden atribut entity [9].



Obrázek 3.2: Ukázka perspektivní zdi [7].

## 3.3 Komponenta pro práci se slovy

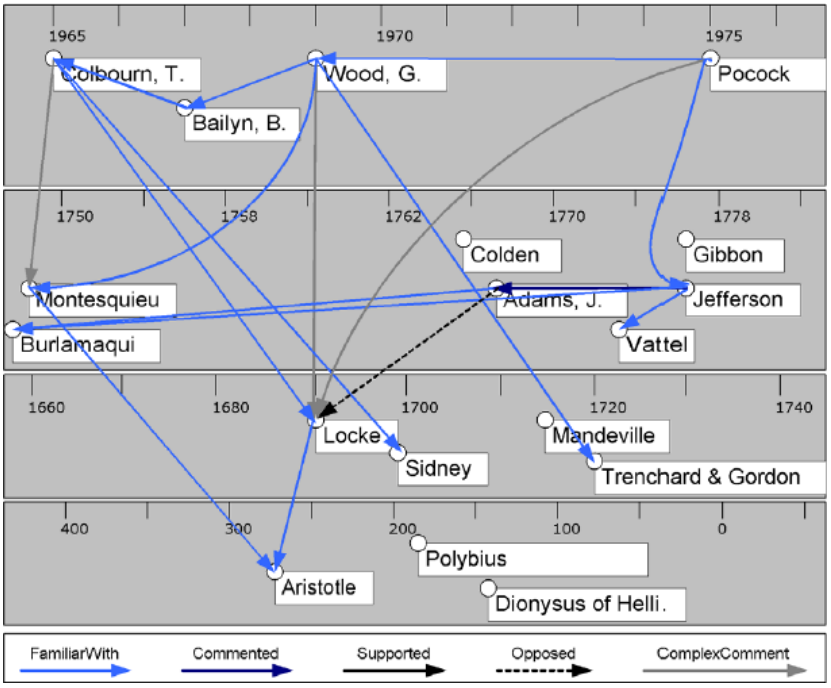
Je mnoho způsobů jak interagovat s časovou osou. V této sekci jsou popsány způsoby jak pracovat s vyhledáváním podle slov.

Toto vyhledávání je zaměřeno na nalezení slova nebo četnosti nejblíže se podobajících slov v textu zobrazených událostí. To umožní uživateli hledat v textu podle slova tak, že pokud je slovo nalezeno, označí se všechny události obsahující toto slovo a okno se zaměří na místo, kde je první nalezená událost. Toto zaměření zahrnuje posun osy na nalezenou instanci tak, aby byla instance uprostřed. Přesun by měl být inicializován animací, aby se uživatel orientoval v přesunu. Pokud není zadáno slovo, vypíšeme seznam nejbližších slov nalezených v textu seřazených sestupně podle četnosti slov a po kliknutí

na jedno z těchto slov opakujeme akci jako u nalezení slova. Není-li zadané žádné slovo, vypíše se seznam nejfrekventovanějších slov v textu [2].

## 3.4 Časově nezávislé skládání

Tento způsob umožňuje rozdělit obrazovku na více nezávislých časových os. Čas na těchto osách může být nastaven nezávisle na ostatních. Tento způsob pomáhá zobrazovat vztah mezi dlouho vzdálenými instancemi viz Obrázek 3.3.




Obrázek 3.3: Časově nezávislé skládání [5].

Každý prvek může být považován za funkční časovou osu a takováto osa může být zabalena nebo rozbalena. Pokud je osa zabalená, ostatní prvky odkazující na prvky shluknuté v pod-ose odkazují na tuto zabalenou osu, aby byla zřejmá spojitost mezi prvky.

Tím, že shlukneme prvky do pod-os, dosáhneme toho, že zajistíme zobrazení velkého množství dat a schopnost rozeznat spojitosti mezi těmito daty [5].

## 3.5 Filtrovací strom

Tento způsob zahrnuje přidání bloku obsahující různé aspekty, které mohou být přidány jako filtry. Dále filtrační panel, který zobrazuje aktuální strom filtrování viz levý panel Obrázek 3.4.


Obrázek 3.4: Filtrační strom [12].

Filtrační strom reprezentuje skupinu dotazů, které jsou využity pro zobrazení událostí nad množinou. Každá cesta od kořene k listu je jeden dotaz. To znamená, že je formován cestou do listu, zatímco se provádí filtrování podle aktuálního vrcholu.

Rozhraní začíná s prázdným filtračním stromem a zobrazením celé množiny časové řady. Uživatel může přidat aspekt pomocí výběru z boxu aspektů a poté je časová osa rozdělena do více časových os s položkami, které odpovídají atributu aspektu. Dále má uživatel možnost přidat podtyp aspektu, následně je osa znovu rozdělena podle aspektů a filtrována.[12]

## 3.6 Metody redukce vizuálního šumu (Visual clutter reduction)

S vizuálním šumem (visual clutter) se setkáme na každém kroku našeho života. Vizuální šum je třeba uvážit při vyvíjení uživatelských rozhraních a vizualizaci informací. Abychom mohli uvést metody pro redukci vizuálního šumu, je třeba si říci co tento pojem znamená. Můžeme si ho definovat následovně:

**Definice:** Šum je stav, ve kterém přebytečné položky, nebo jejich reprezentace a nebo organizace, vede k degradaci výkonu u některých úkolů [8].

Spojení mezi vizuálním šumem a reprezentací nebo organizací informací v zobrazení bylo zaznamenáno mnoha designéry a výzkumníky. Pro návrhy webových stránek je doporučováno zaměřit se na organizování textové a grafické informace, abychom zredukovali vizuální šum. Pro zobecněné mapy při změně měřítka zobrazení je třeba upravit reprezentaci informací tak, aby se snížil šum a zvýšila použitelnost [8].

Především u velkých grafů se můžeme setkat s pojmem vizuální šum, abychom tento problém vyřešili, uvedu zde metody, které slouží k redukci vizuálního šumu.

### 3.6.1 Nahrazení hran

Touto metodou se snažíme nahradit hrany grafu spline křivkami (spojitá množina souřadnic bodů) a tím zredukovat počet protínání hran, jelikož je považováno za hlavní příčinu vizuálního šumu. Tento způsob lze především použít u grafů, které mají předdefinované souřadnice uzlů, například geografické pozice. Avšak minimalizování počtu protínání hran není dobrý přístup a to z toho důvodu, že minimalizování počtu protnutí je NP-úplný problém, a zároveň tento způsob potřebuje hodně času na nalezení řešení, proto je velmi nevhodné použít tento způsob pro interaktivní grafy.

Další způsob jak zmírnit protínání hran je vykreslovat protnutí konfluentně (stékajíc se). Tento způsob přímo neřeší zredukování počtu protnutí hran, ale vykresluje hrany jako křivky, které přecházejí hladce do místa protnutí, a udělá je více přirozené pro pozorovatele viz Obrázek 3.5 [3].

Obrázek 3.5: Ukázka konfluentního vykreslení [3]

### 3.6.2 Shlukování uzlů

Shlukování uzlů (reprezentace dat) je známo pod mnoha jmény jako například seskupování, klasifikace a rozpoznávání vzorů. Tento pojem v souvislosti s vizuálním šumem znamená, rozdělení celé struktury do několika pod-grafů.

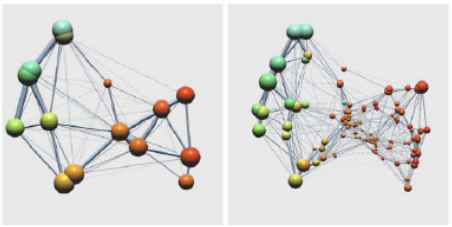
Tyto pod-grafy poté vykreslovat jako jednotlivé uzly grafu. Tímto shlukováním podobných a nebo odlišných uzlů uvolníme velké množství prostoru a tím snížíme vizuální šum [3].

Pro shlukování uzlů existují různé způsoby. Hlavní rozdíl mezi nimi je definice vzdálenosti a nebo podobnosti mezi dvěma položkami v datech. O podobnosti dat existují dva základní přístupy [3].

První z přístupů je založen na obsahu a využívá sémantiku (význam) dat pro shlukování uzlů. Ačkoli tento přístup může produkovat velmi smysluplné shlukování, není tolik prostudován, protože je velmi závislý na aplikaci a nepřináší obecné řešení [3].

Druhý z přístupů je založen na struktuře, tento přístup je více obecný a používá pouze informaci o struktuře (relační informace například spojení a nebo hierarchické). Ve shlucích založených na struktuře se dobře definované shluky obvykle vztahují ke komponentám grafu, které mají více spojení uvnitř než-li k vnějším uzlům. Různé techniky můžou docílit shlukování podle struktury a mohou být rozděleny do tří metodik [3]:

- **Grafově teoretické:** Grafově teoretické algoritmy se opírají o podobnostní matici reprezentující podobnost mezi jednotlivými uzly. Shluky jsou tvořeny úzce souvisejícími uzly.
- **Jedno průchodové:** Jedno průchodové algoritmy dokáží vytvářet shluky jejich růstem od jednotlivých datových bodů zvané shluková semínka. Nejjednodušší způsob funguje tak, že se vybere startovní uzel a pak ostatní uzly jeden po druhém přidává do shluku, dokud není shluk dosti velký.
- **Iterativní algoritmy:** Iterativní algoritmy mohou používat generované shluky jinými shlukovými algoritmy. Při realizaci této metody existují tři hlavní kroky a to zhrubnutí, dělení a promítání grafu.



Obrázek 3.6: Dvě různé metody shlukování pro stejný graf [3].

# 4 Analýza požadavků

V této části je popsán seznam požadavků pouze na základě konzultací s vedoucím práce a týmu spolupracujícím na tomto projektu. Každý uvedený požadavek níže byl zrealizován v rámci této práce.

## 4.1 Technické požadavky

### 4.1.1 Formát dat

Základním požadavkem je změna aktuálního formátu dat tak, aby každý prvek dat mohl obsahovat podmnožinu prvků, každý z těchto prvků má mít podobné chování jako původní entita, tzn. zobrazení momentu nebo nějakého trvání. S touto podmnožinou bude umět aplikace pracovat a náležitě jí zobrazit. Ukázka pseudo dat:

```
{"id":1,"stereotype":"firm","name":"Vanillas"},
{"id":2,"stereotype":"event","name":"Zalozeni firmy"},
{"id":3,"stereotype":"event","name":"Prodej firmy"},
```

Zde vidíme, že události existují jako samostatné entity, ale jsou přitom závislé k firmě a musíme k nim vést hrany, abychom toto spojení zobrazili. Pokud tyto události vložíme přímo do entity dostaneme:

```
{"id":1,"stereotype":"firm","name":"Vanillas", subItems:
   [
        {"id":2,"name":"Zalozeni firmy"},
        {"id":3,"name":"Prodej firmy"},
]},
```

Tímto se ušetří spojení mezi entitami a je přímo jasné k jaké entitě tyto události patří.

Dále, musí existovat možnost ke každému prvku přidat třídy kaskádových stylů, které se projeví při zobrazení v aplikaci. Tyto třídy bude možné definovat v souboru speciálně pro tyto třídy.

### 4.1.2 Změna určování rozdělení

V původní aplikaci jsou pruhy definovány přímo v kódu a ne každý uživatel by byl schopen je měnit. Proto dalším požadavkem je přesunutí definice pruhů do samostatného souboru, kde uživatel bude moci určit jak se jaký

typ dat bude zobrazovat. Tyto pruhy pak mohou obsahovat více různých datových typů. To znamená, že pruhy půjdou sloučit dohromady.

## 4.2 Funkční požadavky

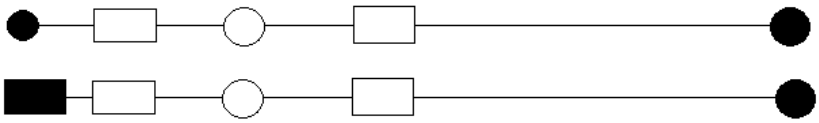
### 4.2.1 Renderery pro zobrazení pod entit

Nejdůležitějším požadavkem na práci je vytvoření nových rendererů - objekty, které rozhodnou, jak mají být data vyobrazena uživateli.

První je renderer, který bude umět zobrazovat podmnožinu prvků entity a to tak, aby uměl zobrazit jak moment, tak dobu nějakého trvání. Další vlastností rendereru bude, že při určitém zobrazení bude umět podmnožinu schovat a nebo naopak zviditelnit. Využití je především pro sledování určitých aktivit událostí během trvání nad prvku například sledování aktivity uživatele. Návrh výsledku rendereru viz 4.1.

Obrázek 4.1: Návrh rendereru

Druhý je renderer, který bude umět taktéž zobrazovat podmnožinu prvků entity, ale v jiném formátu. Prvek typu moment se zobrazí jako kruh a prvek typu doba trvání se bude zobrazovat jako obdélník. Dále bude vykreslovat spojovací čáru mezi vykreslenými prvky. Využití je především pro entitu, která obsahuje méně důležitá data, která chceme v rámci entity zobrazit například projekt git. Návrh výsledku rendereru viz 4.2.



Obrázek 4.2: Návrh rendereru

## 4.3 Testovací nástroj

Poslední z požadavků je požadavek na testovací nástroj, který bude sloužit pro otestování, zda-li došlo ke zlepšení výsledků oproti předchozí verzi aplikace. Bude obsahovat testovací otázky a bude sledovat aktivitu uživatele a časy odpovědí. Po skončení testu tyto zaznamenané aktivity ve vhodném

formátu dá k dispozici testujícímu. Časy poté budou porovnány s výsledky z předchozí verze aplikace.

# 5 Realizace

## 5.1 Formát dat

Požadavek je na změnu formátu dat, tak aby každý prvek dat mohl obsahovat podmnožinu datových prvků, které budou vykazovat podobné chování jako původní entita (reprezentace dat v grafu).

### 5.1.1 Změna datového formátu

Aby každý prvek dat mohl obsahovat podmnožinu dalších datových prvků, je třeba definovat pole, které bude obsahovat tuto podmnožinu.

Můžeme si zde prohlédnout jak jeden datový prvek ve formátu JSON vypadá a co jednotlivé vlastnosti znamenají:

```
{
        "id":<Number>,
        "stereotype":<String>,
        "name":<String>,
        "begin":<String ISO8601>,
        "end":<String ISO8601>,
        "description":<String>,
        "properties": { ... }
}
```

- **id:** Vyjadřuje identifikaci prvku v grafu, díky tomuto můžeme prvek v grafu najít a nebo ho spojit s jiným prvkem pomocí hrany.
- **stereotype:** Vyjadřuje typ prvku, podle toho typu dokážeme poznat jaký použít *renderer* pro jeho vykreslení.
- **name:** Obsahuje hlavní název prvku, který se zobrazí uživateli.
- **begin:** Vyjadřuje datum začátku prvku v grafu.
- **end:** Vyjadřuje datum konce prvku v grafu. Pokud tento údaj neuvedeme, považujeme prvek za moment.
- **description:** Nepovinný údaj, který obsahuje popis prvku.
- **properties:** Nepovinný údaj, který obsahuje doplňující informace o prvku.

Proto, aby prvek obsahoval podmnožinu dalších prvků, jsem přidal do stávajícího formátu jednu vlastnost ("subItems"), která bude obsahovat pole všech pod-prvků viz:

```
{
        ...
        "properties": { ... },
        "subItems": { ... }
}
```

Pro možnost zobrazení pod-prvků je třeba definovat jejich formát. Formát by se měl podobat původnímu prvku a vypadá následovně:

```
{
        "id":<Number>,
        "name":<String>,
        "begin":<String ISO8601>,
        "end":<String ISO8601>,
        "css":<String>
}
```

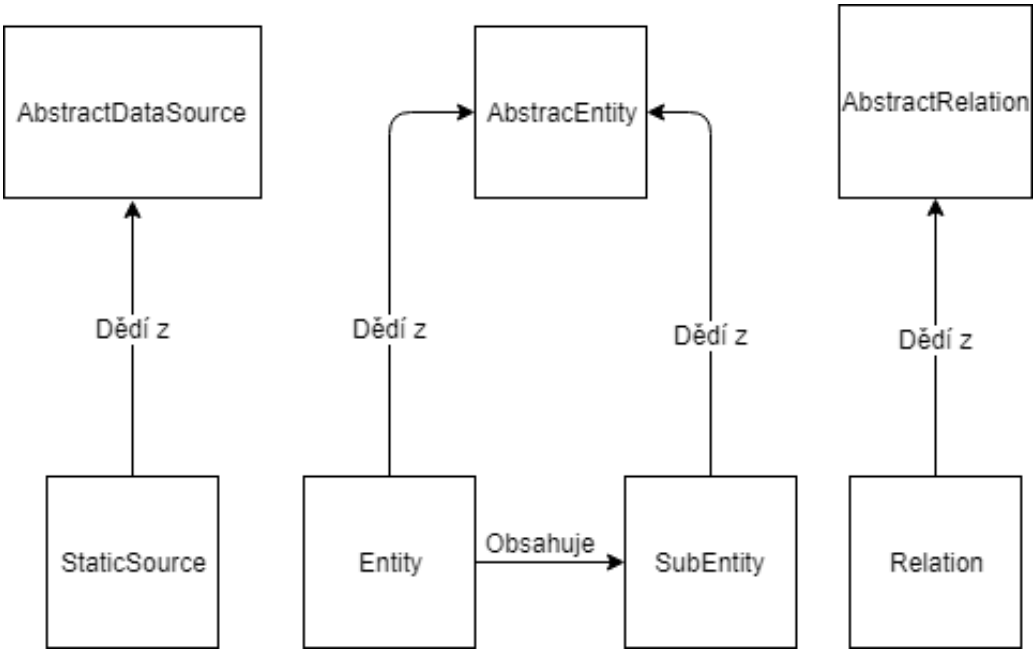
- **id:** Vyjadřuje identifikaci pod-prvku v grafu, díky tomuto můžeme pod-prvek v grafu najít a nebo ho spojit s jiným prvkem pomocí hrany. Toto identifikační číslo musí být unikátní v celé množině dat (nemůže ho obsahovat ani žádný nad-prvek).
- **name:** Obsahuje hlavní název pod-prvku, který se zobrazí uživateli.
- **begin:** Vyjadřuje datum začátku pod-prvku v grafu.
- **end:** Vyjadřuje datum konce pod-prvku v grafu. Pokud tento údaj neuvedeme, považujeme pod-prvek za moment.
- **css:** Nepovinný údaj, který obsahuje doplňující kaskádové styly, které se přiřadí pod-prvku při vykreslování. Pro definování tříd jsem vytvořil soubor (*../src/css/customStyles.css*), ve kterém by se měly styly objevovat, je to doporučeno pro lepší přehlednost a údržbu stylů aplikace.

### 5.1.2 Zpracování nového datového formátu

Stávající aplikace obsahovala definici třídy *Entity*. Tato třída slouží jako reprezentace jednoho datového prvku a když je volán její konstruktor, jsou jí předána data, která se mají v této třídě uchovat. Z těchto dat lze jednoduše vyčíst atributy, které obsahují, a pak uložit do struktur třídy. Proto, aby

tato třída dokázala uchovávat pod-prvky, bylo přidáno pole *subEntities*, do kterého se pod-prvky vkládají.

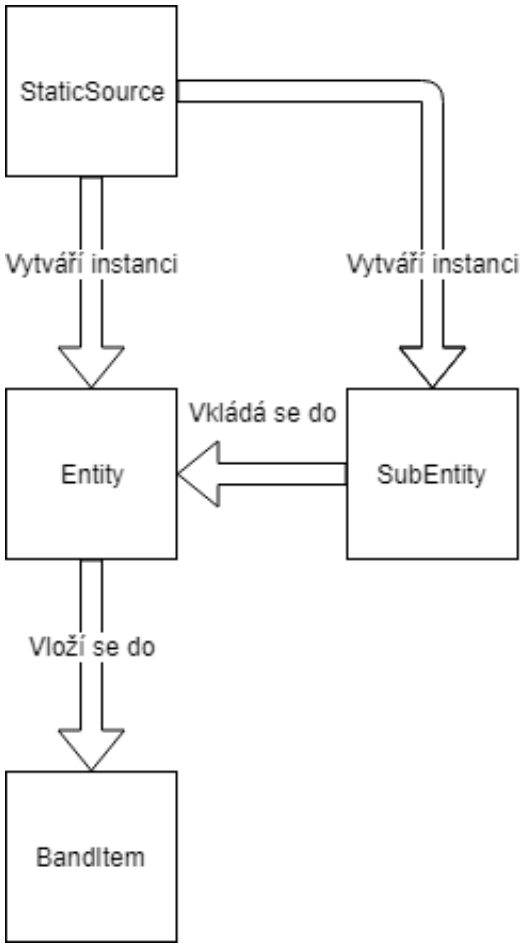
Pro reprezentaci těchto pod-prvků byla vytvořena třída *SubEntity*, která má podobné vlastnosti jako *Entity*, avšak nemůžeme tuto třídu přímo dědit kvůli chování v kontextu, kde nechceme interagovat přímo s každým pod-prvkem, ale nadřazeným prvkem, který je typu *Entity*. Proto třída *SubEntity*, dědí pouze od třídy *AbstractEntity*, abych získal vlastnosti jako je pozice v ose a přístup k základním informacím jako je identifikační číslo, priorita, typ a další. Na Obrázku 5.1 si můžeme prohlédnout jak zapadá třída do výchozí implementace.




Obrázek 5.1: SubEntity ve výchozí implementaci

Zpracování těchto pod-prvků zprvu probíhalo přímo ve třídě *Entity*, ale po další implementaci jsem zjistil, že je třeba tyto nové pod-prvky nějakým způsobem registrovat u zdroje dat *StaticSource*, proto veškeré zpracovávání probíhá právě v tomto zdroji dat viz Obrázek 5.2. Do stávající doby tento zdroj obsahoval slovník *entities* pro uchovávání instancí *Entity*, proto jsem přidal nový slovník *allMappedEntities* pro mapování všech všech prvků, které se v grafu nachází.

Tímto je zajištěno veškeré zpracování nového formátu dat, dále se budeme zabývat jak tyto data zobrazit.




Obrázek 5.2: Zpracování dat

### 5.1.3 Základy pro vizualizaci

V předchozí části jsem si připravil data pro vizualizaci, teď je potřeba vytvořit obálku, která by uchovávala definovaná data a informace o zobrazení v grafu. Ve stávající aplikaci tuto obálku zastupovala třída *BandItem*, která dědí od abstraktní třídy *AbstractItem*, tato třída uchovává mnoho informací, ale nejdůležitější z nich jsou informace o datech, pozici, popisku a zásadní je informace o elementu, který reprezentuje tyto data. S tímto elementem se dále pracuje při zobrazování prvku v grafu nebo taktéž při zobrazení spojení mezi prvky v grafu.

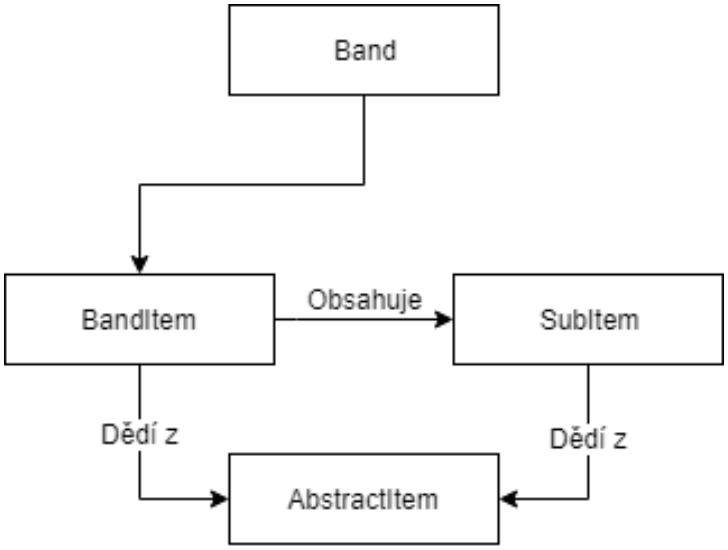
Pro vytvoření obálky je třeba definovat si podobnou, ne-li stejnou třídu jako je *BandItem*. Mám zde několik možností jak tohoto docílit.

První ze způsobů je novou třídu *SubItem* navrhnout tak, aby dědila všechny vlastnosti od *BandItem*, tímto bych vyřešil veškeré problémy s reprezentací, avšak nastal by problém, že pro každý druh pod-prvku by musel

existovat vlastní renderer. Další a závažnější problém je, že by aplikace pracovala s touto instancí jako stejně rovnou s *BandItem* instancí, to by vedlo k nechtěnému chování pod-prvků a k velkému množství ošetřování kódu ve stávající aplikaci.

Druhý ze způsobů je novou třídu navrhnout tak, aby dědila pouze od *AbstractItem*, tím získáme obálku pro základní práci jako je přístup k datům, elementu, pozici a velikosti. Tímto způsobem renderery zůstanou pouze pro *BandItemy* a do těchto rendererů bude třeba přidat pouze jednu metodu pro vyrenderování a překreslení *SubItemů*. Jednoduše poté přidám těmto instancím požadované chování.

Nakonec jsem použil druhý ze způsobů a přidal jsem každé instanci *BandItem* vlastní pole *subItems* pro uchovávání instancí třídy *SubItem*. Tyto instance třídy *SubItem* jsou přidávané při přidávání *BandItem* do instance *Band*, která zastupuje jednotlivý pás zobrazení. Na Obrázku 5.3 vidíme jak třída *SubItem* zapadá do původní implementace.




Obrázek 5.3: SubItem ve výchozí implementaci

Tímto máme definovanou obálku pro data a další zpracovávání této obálky uvedu v části s renderery.

## 5.2 Změna určování rozdělení

Z Obrázek 2.1 vidíme, že je aplikace rozdělená na více pásů. Stávající řešení bylo takové, že každý z těchto pásů je vytvořen pouze pro jeden typ prvku dat. Původní definice pásů v *main.js* vypadala následovně:

```
bands : [
{
        id: "place",
        label: "Mista",
        itemRenderer: new BandItemRenderer("#7DD968"),
        color: "#f5f5f5"
},
{
        id: "person",
        label: "Lide",
        itemRenderer: new BandItemRenderer("#FFB182"),
        color: "#fafafa"
}
]
```

- **id:** Vyjadřuje jaký *stereotype*(typ) dat do pásu patří viz Formát dat.
- **label:** Obsahuje popisek pásu.
- **itemRenderer:** Určuje jakým rendererem se mají prvky v pásu zpracovávat.
- **color:** Jakou barvou budou prvky v pásu obarveny.

Tímto způsobem jsem nemohl zajistit to, aby jeden pás dokázal obsahovat více typů dat. Proto jsem implementoval nové řešení rozdělení a definice pásů.

Definice pásů se nyní odehrává ve třídě *BandsDistribution* a vypadá následovně:

```
bands : [

{
        id: "person",
        label: "Lide",
        itemRenderer: new BandItemRenderer("#FFB182"),
        color: "#fafafa"
},
{
        id: "Sjednoceny pruh",
        label: "Sjednoceni",
        types: [
          {
```

```
14              id: "event",
15              itemRenderer: new BandItemRenderer("#F2BC53"),
16              color: "#f5f5f5"
17              },
18              {
19              id: "item",
20              itemRenderer: new BandItemRenderer("#78B4FF"),
21              color: "#fafafa"
22              }
23          ]
24  }
25  ]
```

Pásy nyní lze definovat dvěma způsoby. Prvním je původní způsob viz pás definovaný jako "Lide". Druhý je nový způsob, ve kterém je přidáno nové pole *types*, které definuje jaké typy dat lze v tomto pásu zobrazovat. Tyto typy lze definovat jako původní pás, ale bez potřeby definice popisu *label*.

Abych zajistil zpracování této nové definice pásu bylo potřeba zajistit několik úprav ve stávající aplikaci. Ve třídě *BandGroup*, bylo zapotřebí definice nového slovníku *bandTypes*, který obsahuje pro každý typ dat přiřazený pás instance *Band*. Pak následné hledání pásů a entit probíhá nejdříve v nově definovaném slovníku *bandTypes* a až v původním slovníku *bands*, tímto jsem dosáhl kompatibility s původní definicí pásů. Dále bylo zapotřebí definovat ve třídě *Band* nový slovník *types*, který obsahuje ke každému typu dat renderer, kterým se mají data vyobrazit, při vytváření instance *BandItem* nebo *SubItem* lze zjistit jaký renderer jim přiřadit podle jejich typu.

## 5.3 Nové renderery

Nyní po implementaci výše zmíněných věcí, může pod-prvky zobrazovat jakýkoliv renderer ve stávající aplikaci, jedinou podmínkou pro toto zobrazení je implementace dvou hlavních metod, pro práci s těmito pod-prvky. Tyto metody jsou *renderSubItem* a *correctProtrusionSubItem*, funkčnosti těchto metod budou popsány níže.

První z metod *renderSubItem* určuje jakým způsobem bude generována HTML obálka instance *SubItem*. Této obálce nastaví základní atributy jako je identifikační číslo v kontextu, třídy kaskádových stylů definované u dat. Tato metoda je volána pouze jednou a to při generování grafu.

Druhá z metod *correctProtrusionSubItem* určuje jakým způsobem se počítá pozice a šířka výše generované obálky. Tato metoda je volána při ja-

kémkoliv překreslení grafu. Dále je potřeba implementovat způsob jakým se bude určovat pozice *SubItem*.

Tím že dokáži zobrazovat pod-prvky entity, jsem docílil jakéhosi shlukování uzlů, které se vytváří přímo v datovém souboru, kde určíme jaké pod-prvky entita obsahuje. Shluky pak lze jednoduše skrývat a zanechávat pouze hlavní entitu, toto chování by mohlo zajistit lepší přehlednost dat a snížení vizuálního šumu. Zda-li došlo ke zlepšení oproti starému formátu budu uvádět ve výsledku testování, kde budu porovnávat nový formát oproti starému formátu.

### 5.3.1 Počítání pozice *SubItem*

Jelikož jsou všechny prvky v grafu určeny absolutní pozicí, je třeba definovat tuto pozici vztahující se k nadřazenému prvku *BandItem*. Proto jsem přidal do třídy *SubItem* atribut *leftPositionToParent*, která určuje vzdálenost zleva k nadřazenému prvku. Tuto vzdálenost lze počítat 3 různými způsoby.

První ze způsobů jak získat vzdálenost zleva je určen vztahem:

$$l = w \times \frac{d}{d_e}$$

, kde $l$ je počítaná vzdálenost, $w$ je šířka nadřazeného prvku, $d$ je rozdíl trvání nadřazeného prvku a podřazeného prvku, $d_e$ je doba trvání nadřazeného prvku.

Druhý ze způsobů jak získat vzdálenost zleva je určen vztahem:

$$l = px(s_s - s_e)$$

, kde $l$ je počítaná vzdálenost, $px$ je funkce pro převod na pixely, $s_s$ je začátek podřazeného prvku, $s_e$ je začátek nadřazeného prvku.

Třetí ze způsobů jak získat vzdálenost zleva je určen vztahem:

$$l = px(s_s) - l_e$$

, kde $l$ je počítaná vzdálenost, $px$ je funkce pro převod na pixely, $s_s$ je začátek podřazeného prvku, $l_e$ je vzdálenost zleva nadřazeného prvku.

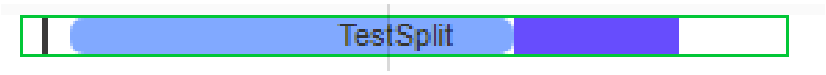
Pro řešení byl zvolen třetí způsob počítání vzdálenosti a to proto, že dokázal správně počítat vzdálenost, i když prvek nebyl přímo zobrazený v zorném poli grafu.

### 5.3.2 Implementace rendererů

#### SplitBandItemRenderer

Požadavek na první renderer je takový, aby uměl zobrazovat podmnožinu prvků entity. Tato podmnožina prvků může obsahovat jak typ moment, tak dobu nějakého trvání. Další vlastností tohoto rendereru bude, aby uměl tuto podmnožinu schovat a zobrazit za určitých podmínek.

Abych tento požadavek splnil, vytvořil jsem třídu *SplitBandItemRenderer*. Tato třída dědí od původního *BandItemRenderer*, tím získává metody pro práci s *BandItem*. Tento nový renderer musí obsahovat metody pro práci s instancemi *SubItem* popsány viz výše. Dále je třeba přetížit metodu pro překreslení *redraw*, tato metoda slouží k volání metod pro přepočet pozice a šířky podprvků. Tuto metodu jsem zachoval podobnou jako původní definovanou v *BandItemRenderer*, ale přidal jsem do ní volání metody *correctProtrusionSubItem*, která pro každou instanci *SubItem* opraví pozici a šířku. Dále v této metodě *redraw* byl přidán kód, který zajišťuje schování a zobrazení podmnožiny prvků za určité velikosti instance *BandItem*. Těmito změnami jsem dosáhl požadovaného chování viz požadavek. Výsledek tohoto rendereru si můžete prohlédnout na Obrázku 5.4.




Obrázek 5.4: Výsledek rendereru

#### DumbbellItemRenderer

Požadavek na druhý renderer je takový, aby uměl taktéž podmnožinu prvků, ale v jiném formátu. Prvek typu moment se zobrazí jako kruh a prvek typu doba se vykreslí jako obdélník. Tyto prvky budou spojeny spojovací čárou, aby bylo zřejmé, že k sobě patří.

Abych tento požadavek splnil, vytvořil jsem třídu *DumbbellItemRenderer*. Tato třída opět dědí od původního *BandItemRenderer*, avšak z důvodu jiného způsobu vykreslování bylo třeba přetížit více základních metod. Abych vyřešil spojení podmnožiny prvků čárou, přetížil jsem metodu *renderContinuous*, v této metodě jsem přidal vložení čáry do obálky celého *BandItem*. Avšak nastal problém s tím, že původní *BandItem* měl text podle pozice v grafu buď uvnitř nebo vedle této obálky. Proto bylo třeba přetížit metodu *redrawLabel*, která upravuje pozici textu popisku vůči obálce. Tuto metodu jsem upravil tak, aby text popisku byl vždy vedle obálky *BandItem*,

tímto bylo vyřešeno "přeškrtnutí textu". Dále bylo třeba definovat nový atribut *type* dat proto, aby byl tato instance měla vždy nějaký konec a začátek. Tento atribut dat vypadá následovně:

```
{"id":5,stereotype":"dumbbell-entity",
"name":"TestDumbbell",
"begin":"2010-09-27T00:00:00",
"end":"2150-06-21T23:59:59",
"properties":{"startPrecision":"day"},
"subItems":
[
  {"id":6,"name":"King","type":"start"},
  {"id":7,"name":"Dumbell","type":"end"},
  {"id":8,"name":"Queen","begin":"2020-09-27T00:00:00"},
]
},
```

Tímto typem řekneme jaký prvek je považován za start a jaký za konec, není třeba dodávat atribut *begin*, protože se tomto prvku přiřadí buď to začátek nebo konec nad-prvku. Metoda *redraw* je taktéž přetížena a doplněna o volání přepočítání všech pod-prvků a taktéž za určité podmínky schová všechny pod-prvky a nebo je zobrazí. Tímto způsobem jsem dosáhl požadovaného chování viz požadavek. Výsledek tohoto rendereru si můžete prohlédnout na Obrázku 5.5.


Obrázek 5.5: Příklad výsledku rendereru

## 5.4 Testovací nástroj

Požadavkem bylo vytvořit testovací nástroj, který provede uživatele posloupností otázek a bude sledovat jeho aktivitu.

Vytvořil jsem třídu *QuestionTool*, která slouží jako testovací nástroj. Tato třída zajišťuje správu otázek, uživatelského rozhraní a aktivity uživatele. Instance této třídy se vytvoří, jakmile je stisknuto tlačítko pro spuštění testu. Po vytvoření instance je volána funkce *setup*:

```
_setup : function(){
 // Pripravi objekt pro uchovavani vysledku testu
 //a aktivity uzivatele
 this._setupTestObject();
 // Namapuje otazky a pripravi prvni otazku
 this._setupQuestions();
 // Nastavi udalosti, na ktere nastroj reaguje
 this._setupEvents();
 // Spusti test
 this._startTest();
},
```

### 5.4.1 Načítání a formát otázek

Za prvé jsem vytvořil jednoduchý soubor *questions.js*, ve kterém jsou obsaženy posloupnost otázek a odpovědí. Obsahuje jednoduchou funkci, která vrací pole otázek v JSON formátu:

```
questions : [
{
        "text" : <String>,
        "answer" : <String>
},
{ ... }
]
```

Pak reálný příklad může vypadat takto:

```
questions : [
{
        "text" : "V kolikatem stoleti zil Karel IV.?",
        "answer" : "14"
},
{ ... }
]
```

Tímto mám zajištěno načítání otázek a správných odpovědích, které jsou díky tomuto formátu lehce použitelné.

### 5.4.2 Sledování aktivity uživatele

Abych zajistil sledování aktivity uživatele, bylo třeba přidat nové *listenery*. Ty se přiřadí jakmile je vytvořena instance třídy v metodě *setupEvents*:

```
1  _setupEvents : function(){
2   //Kliknuti na prvek
3   this.addListener(
4    "itemLogClick", new Closure(this, this._itemLogClicked)
5   );
6   //Priblizeni osy
7   this.addListener(
8    "timelineLogZoom", new Closure(this, this._timelineLogZoomed)
9   );
10  //Kliknuti na hranu
11  this.addListener(
12   "relationLogClick", new Closure(this, this._relationLogClicked)
13  );
14  //Najeti mysi na prvek
15  this.addListener(
16   "itemLogEnter", new Closure(this, this._itemLogEnter)
17  );
18  //Najeti mysi na hranu
19  this.addListener(
20   "relationLogEnter", new Closure(this, this._relationLogEnter)
21  );
22  //Pravy klik na prvek
23  this.addListener(
24   "itemLogRightClick", new Closure(this, this._itemLogRightClicked)
25  );
26  //Posun osy
27  this.addListener(
28   "timelineLogShifted", new Closure(this, this._timelineLogShifted)
29  );
30  //Potvrzeni odpovedi
31  $("#confirm_btn").on(
32   "click", new Closure(this, this._confirmAnswer)
33  );
34  //Stiskt klavesy
35  this._answerElement.on(
36   "keyup", new Closure(this, this._keyUpEvent)
37  );
38  //Kliknuti mysi
39  document.body.addEventListener(
40   "click",new Closure(this, this._mouseClickEvent), true
41  );
42 },
```

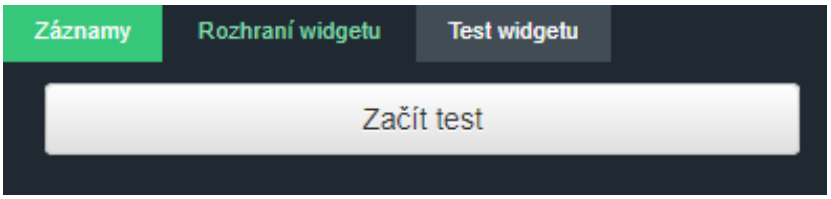
Vysvětlení konstrukce *Closure* naleznete v práci [6]. Díky definovaným *listenerům*, dokáži sledovat tyto aktivity uživatele:

1. Kliknutí na jakýkoliv prvek grafu.
2. Najetí myší na jakýkoliv prvek grafu.
3. Pravý klik na jakýkoliv prvek grafu.
4. Kliknutí na jakoukoliv hranu grafu.
5. Najetí myší na jakoukoliv hranu grafu.
6. Přiblížení a oddálení osy.
7. Posun osy.
8. Kliknutí myši kdekoliv v grafu.

Aktivity pak lze jednoduše ukládat do struktury testu s příslušnými informacemi a popisky. Formát struktury testu je popsán viz Výstup nástroje.

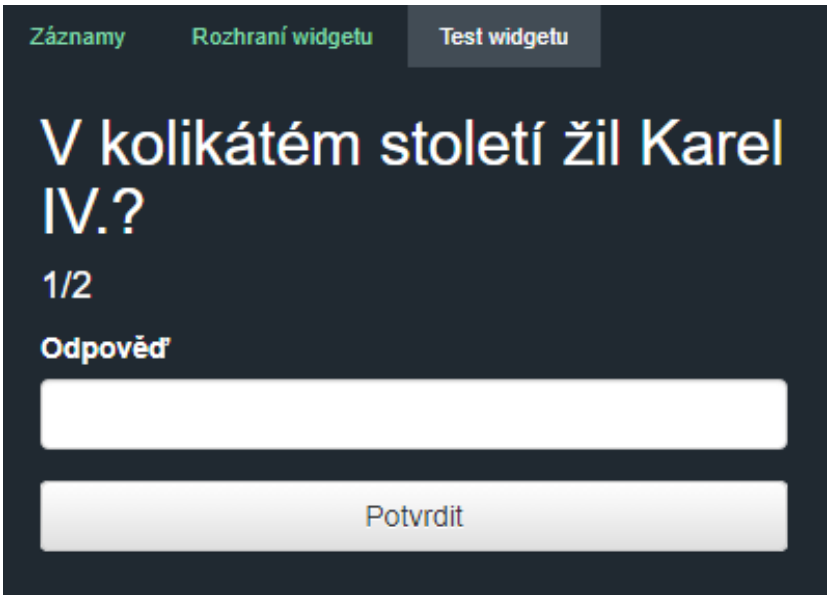
### 5.4.3 Práce s uživatelským rozhraním

Vytvořil jsem záložku "Test widgetu", ve které probíhá celý test.

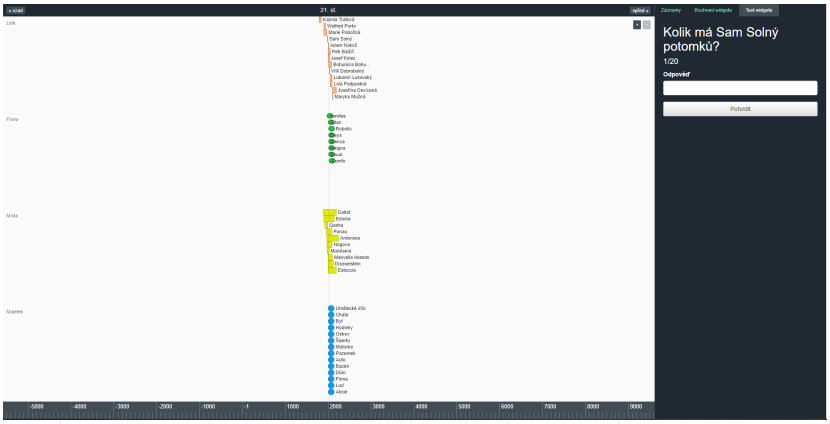



Obrázek 5.6: Záložka testu

Tento test začne po kliknutí na tlačítko "Začít test". A nástroj zajistí schování tlačítka a zobrazení komponent otázky. Je zobrazen text otázky, postup v testu a pole pro odpověď s potvrzovacím tlačítkem viz 5.7.

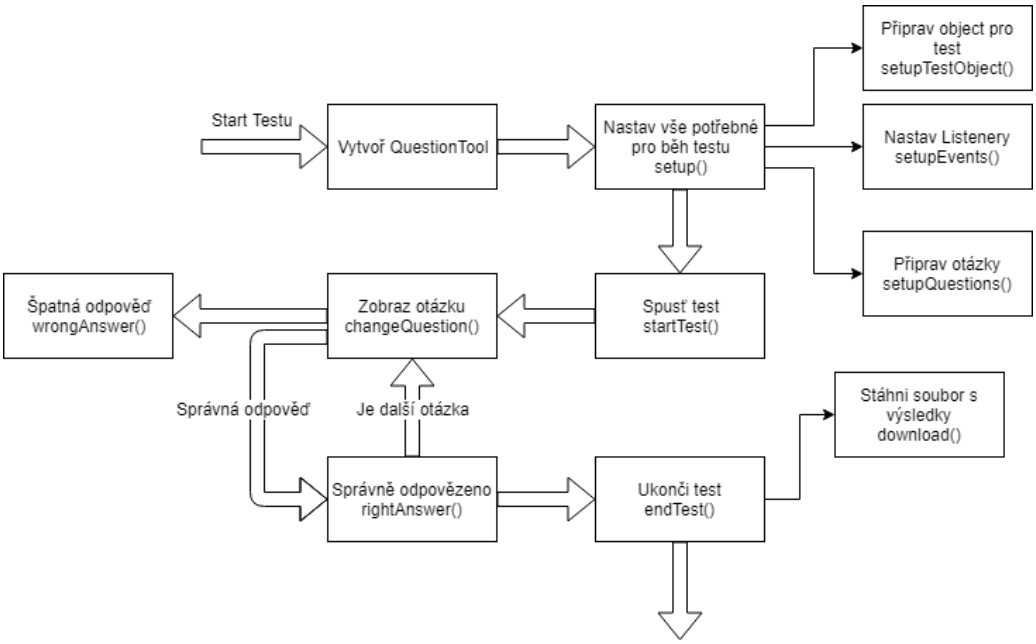



Obrázek 5.7: Otázka testu




Obrázek 5.8: Záložka testu v celém kontextu

Pokud uživatel odpoví správně, změní se otázka a postup v testu pomocí metody *rightAnswer*. Pokud uživatel odpoví špatně, je zobrazen červený popisek, že byla zadána nesprávná odpověď pomocí metody *wrongAnswer*. Po ukončení testu jsou komponenty otázky schovány a je vygenerován soubor s výsledky testu pomocí metody *endTest*. Výsledný soubor je automaticky stažen a je označen datem testu pomocí metody *download*. Vývojový diagram celého testu vypadá následovně:




Obrázek 5.9: Vývojový diagram testu

### 5.4.4 Výstup nástroje

Abych dokázal výsledky testu jednoduchým způsobem uchovávat, vytvořil jsem objekt *test*. Formát objektu vypadá v JSON reprezentaci následovně:

```
1        "startTime": <String>,
2        "endTime": <String>
3        "duration": <Number>,
4        "eventsCount": <Number>
5        "mouseClickedCount": <Number>,
6        "questions": [],
```

- **startTime:** Obsahuje čas startu testu.
- **endTime:** Obsahuje čas konce testu.
- **duration:** Obsahuje dobu trvání testu v sekundách.
- **eventsCount:** Vyjadřuje počet aktivit uživatele během testu.
- **mouseClickedCount:** Vyjadřuje počet kliknutí uživatele během testu.
- **questions:** Slouží jako pole objektů pro otázky.

Instance tohoto objektu se vytvoří při počátku testu a jsou mu nastaveny základní atributy jako je například čas spuštění testu. V průběhu jsou do pole *questions* ukládány jednotlivé instance objektů otázek, které mají následující formát:

```
1        "questionStartTime": <String>,
2        "questionEndTime": <String>
3        "questionDuration": <Number>,
4        "questionText": <String>
5        "index": <Number>,
6        "events": [],
```

- **questionStartTime:** Obsahuje čas startu otázky.
- **questionEndTime:** Obsahuje čas konce otázky.
- **questionDuration:** Obsahuje dobu trvání otázky v sekundách.
- **questionText:** Obsahuje znění otázky.
- **index:** Vyjadřuje pořadí otázky.
- **events:** Slouží jako pole objektů pro aktivity.

Otázka se chová podobně jako instance testu, ale v průběhu otázky jsou vytvořeny jednotlivé aktivity, které mají jednoduchý formát:

```
1        "eventType": <String>,
2        "eventTime": <String>
3        "info": <Object>,
```

- **eventType:** Obsahuje typ aktivity.
- **eventTime:** Obsahuje čas odehrání aktivity.
- **info:** Obsahuje objekt specifický pro aktivitu.

Jednotlivá aktivita viz Sledování aktivity uživatele může mít přiřazeny tyto objekty:

1. Kliknutí na jakýkoliv prvek grafu - Entita.
2. Najetí myší na jakýkoliv prvek grafu - Entita.
3. Pravý klik na jakýkoliv prvek grafu - Entita.

4. Kliknutí na jakoukoliv hranu grafu - Hrana.
5. Najetí myší na jakoukoliv hranu grafu - Hrana.
6. Přiblížení a oddálení osy - Směr.
7. Posun osy - Pozice.

Tímto jsem docílil toho, že dokáži sledovat téměř vše, co uživatel udělá. Čas celého testu, jednotlivých otázek a počet událostí mohu použít v porovnávání zlepšení oproti starému formátu dat. Dále díky této struktuře dokáži zpětně uživatelovo chování napodobit, což by mohlo míti v budoucnosti hodně využití. Po dokončení testu se objekt převede z formátu JSON na řetězec a tento řetězec je vložen do souboru, který se hned po dokončení testu stáhne a je označen datem testu.

# 6 Testování

## 6.1 Cíl testování

Cílem experimentálního testování bylo zjistit, zda-li došlo ke zlepšení práce s nástrojem časové osy. Toto bylo zkoumáno na množině otázek vztahujících se k datům. Porovnáním jednotlivých časů testů jsem zjistil v jaké verzi si uživatel vedl rychleji. Dále jsem tímto otestoval funkčnost nové verze aplikace a zda-li nedošlo k chybám za běhu.

## 6.2 Scénář

Pro každého uživatele, který se zúčastnil testu vypadal scénář následovně:

1. Jsou vyobrazena data ve formátu předchozí verze nástroje.
2. Uživatel je seznámen s daty a jejich rozložením.
3. Dále je seznámen s ovládáním nástroje.
4. Následovně jsou mu ukázané jednotlivé záložky nástroje a vysvětleny, co jaká znamená.
5. Uživatel spustí test a vyplní jednotlivé odpovědi na otázky.
6. Po dokončení testu je vygenerován soubor s aktivitou uživatele a je uložen.
7. Jsou vyobrazena data v novém formátu a pokračuje se dále stejně jako od 2. bodu.

Pořadí dat je vybíráno náhodně. U každého testu byl přítomen dohlížející, který kdykoliv na některé nesrovnalosti ohledně nástroje odpověděl.

## 6.3 Data

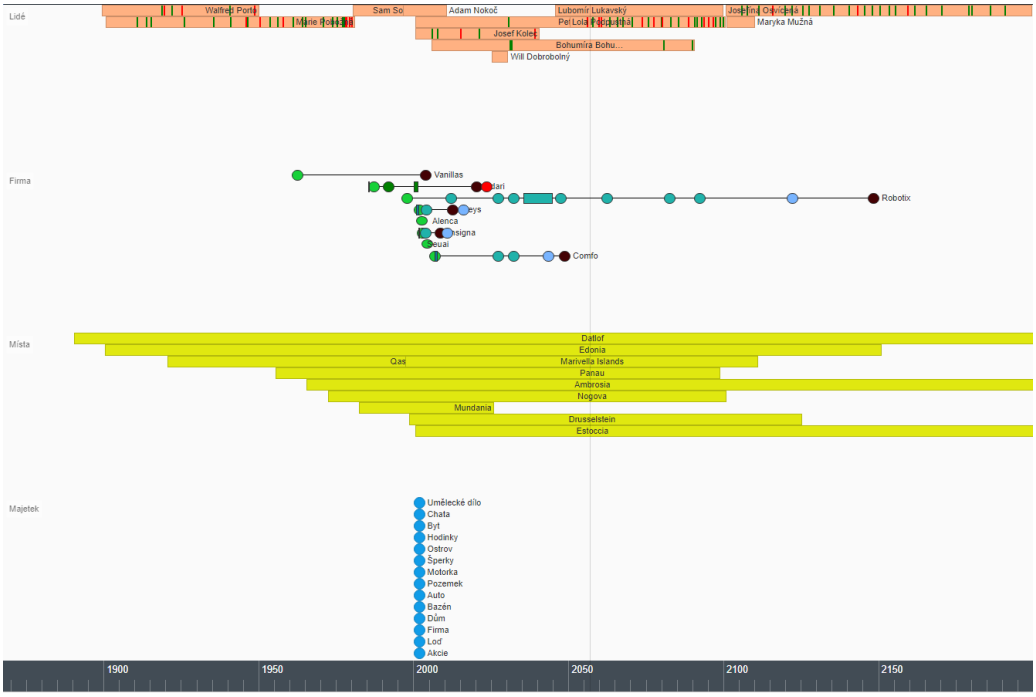
Byla vytvořena fiktivní data, která sloužila pouze pro otestování nástroje.

Data se týkala lidí, firem a majetku. Jednotlivým lidem se za život přihodily různé události. To platí i u jednotlivých firem, kde se události odehrály během existence firmy.

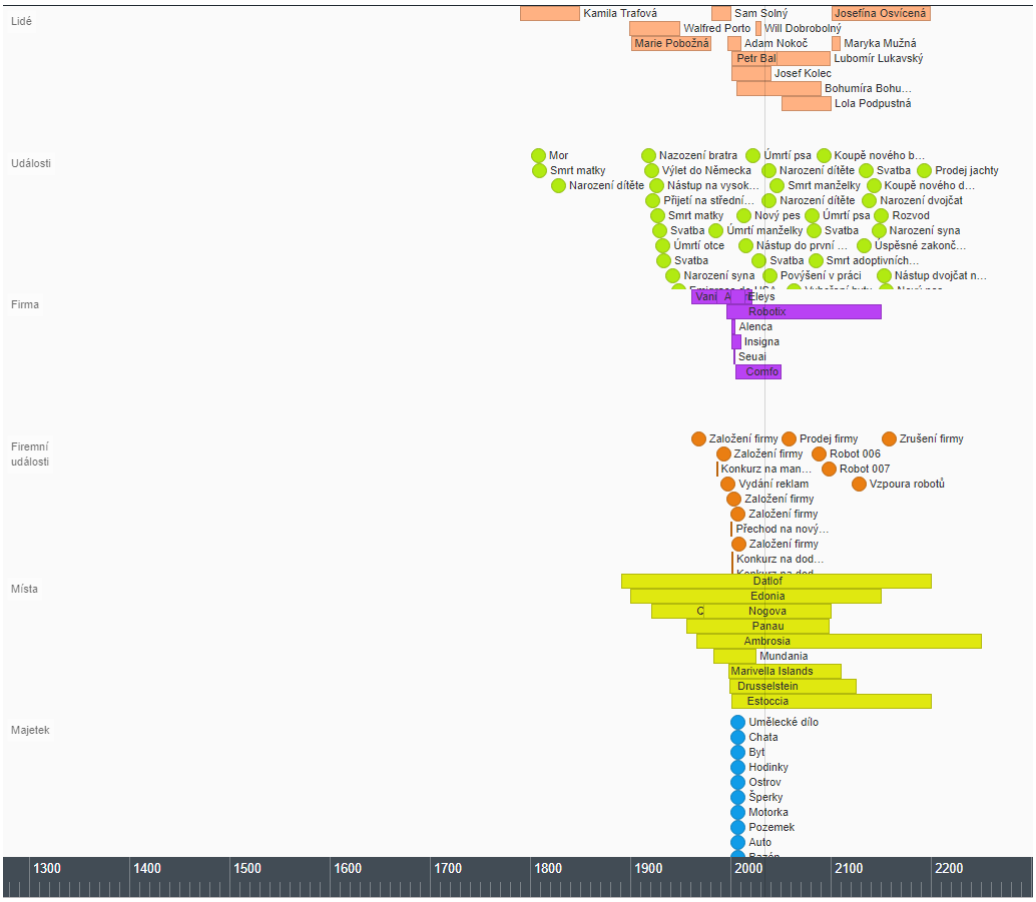
Ve starém formátu nástroje jsou pro jednotlivé události vytvořeny pruhy. Pro události lidí je vytvořen pruh s názvem "Události". Pro události firem je vytvořen pruh "Firemní události". Počet entit je 180 a počet hran je 206.

V novém formátu nástroje jsou události vloženy do jednotlivých entit pomocí nových rendererů. Lidé a jejich události jsou vyobrazeny pomocí **SplitBandItemRenderer** a firmy a jejich události jsou vyobrazeny pomocí **DumbbellItemRenderer**. Počet entit je 180 a počet hran je 68.

Data a jejich rozložení jsou přiložena k nástroji ve složkách *New-data* a *Old-data*. Jejich vizualizaci si můžeme prohlédnout na Obrázku 6.1 a 6.2.




Obrázek 6.1: Nový formát




Obrázek 6.2: Starý formát

## 6.4 Otázky

Jednotlivé otázky byly vypracovány vzhledem k zobrazeným datům a jelikož jsou pro obě verze nástroje identická data, jsou tedy i identické otázky. Z tohoto důvodu jsem vypracoval otázky tak, aby nebyly snadno zapamatovatelné a uživatel nevyužil zapamatovaných odpovědí v druhém průchodu testu.

Otázky a jejich odpovědi jsou uloženy ve formátu JSON v souboru *question.js*, jednotlivé otázky v testu vypadají následovně:

1. Kolik má Sam Solný potomků?
2. V jakém roce zemřel Walfred Porto?
3. Kolik událostí se přihodilo během života Josefíně Osvícené?
4. Kdo je sourozenec Adama Nokoče?(celé jméno)

5. Kolika let se dožil Petr Baclíř?
6. Jakou firmu založila Marie Pobožná?
7. Kde sídlí firma Robotix?
8. Kolik firem sídlí v Edonii?
9. Kolik lidí bydlí v Datlofu?
10. Kolik událostí se odehrálo ve firmě Robotix?
11. Kolik lidí se podílelo na Výzkumu nového druhu?
12. Jakou firmu založil Sam Solný?
13. Kolik událostí se přihodilo během života Marii Pobožné?
14. Kdo vlastní ostrov?(celé jméno)
15. Kolik lidí vlastní umělecké dílo?
16. Kolik lidí se podílelo na celé firmě Comfo?
17. Co vlastní Lola Podpustná, ale Sam Solný ne?
18. V jakém místě se nic neodehrálo?
19. V jakém místě žije nejvíce lidí?
20. Kolik událostí se přihodilo během života Samovi Solnému?

## 6.5 Výsledky

Testování proběhlo na 10 uživatelích a skládalo se pouze z testovací skupiny. Odpovědi na jednotlivé otázky byly ručně zadávány a u vypracování byl přítomen dozor, aby se zabránilo nesmyslnému tipování odpovědi. U výsledků se zaměřuji na dobu průběhu celého testu a na dobu trvání jednotlivých otázek. V tabulkách viz 6.1, 6.2, 8.1, 8.2 lze vidět všechny výsledky uživatelů. Během testů nedošlo k žádné chybě aplikace a k žádné výjimce. V případě nové verze bylo zřejmé, že díky novému formátu běží nástroj plynuleji oproti staré verzi. V tabulce 6.3 jsou porovnány jednotlivé výsledky otázek a zlepšení oproti starší verzi nástroje.

| Nástroje | Subjekt 1 | Subjekt 2 | Subjekt 3 | Subjekt 4 | Subjekt 5 |
|---|---|---|---|---|---|
| Původní[s] | 692 | 657 | 808 | 1069 | 681 |
| Aktuální[s] | 681 | 507 | 435 | 660 | 482 |

Tabulka 6.1: Výsledky celého testu prvních 5 uživatelů

| Nástroje | Subjekt 6 | Subjekt 7 | Subjekt 8 | Subjekt 9 | Subjekt 10 |
|---|---|---|---|---|---|
| Původní[s] | 716 | 468 | 209 | 938 | 864 |
| Aktuální[s] | 596 | 357 | 202 | 595 | 791 |

Tabulka 6.2: Výsledky celého testu posledních 5 uživatelů

$$\sigma_{puvodni} \doteq 229.73$$

$$\sigma_{aktualni} \doteq 163.02$$

$$\phi_{puvodni} \doteq 710.2$$

$$\phi_{aktualni} \doteq 530.6$$

,kde $\sigma$ představuje směrodatnou odchylku a $\phi$ představuje průměr.

$$zefektivnění = (1 - \frac{\phi_{aktualni}}{\phi_{puvodni}}) \times 100 \doteq 25.29\%$$

| Otázka | $\phi_{otázky}$[s] (s) | $\sigma_{otázky}$[s] (s) | Otázka | $\phi_{otázky}$ (s) | $\sigma_{otázky}$ (s) | zlepšení (%) |
|---|---|---|---|---|---|---|
| 1. - Původní | 32.2 | 18.05 | 1. - Aktuální | 17.3 | 20.10 | 46.27 |
| 2. - Původní | 27.4 | 11.93 | 2. - Aktuální | 14 | 2.05 | 48.91 |
| 3. - Původní | 65.9 | 36.7 | 3. - Aktuální | 35.9 | 21.13 | 45.52 |
| 4. - Původní | 35.5 | 17.92 | 4. - Aktuální | 30.1 | 15.26 | 15.21 |
| 5. - Původní | 11.8 | 3.25 | 5. - Aktuální | 12.4 | 7.23 | -5.08 |
| 6. - Původní | 87.8 | 106.03 | 6. - Aktuální | 35.6 | 30.36 | 59.45 |
| 7. - Původní | 23.1 | 10.04 | 7. - Aktuální | 17.3 | 7.72 | 25.11 |
| 8. - Původní | 20.4 | 10.36 | 8. - Aktuální | 16.4 | 12.50 | 19.61 |
| 9. - Původní | 15.7 | 5.62 | 9. - Aktuální | 11.8 | 7.76 | 24.84 |
| 10. - Původní | 44.1 | 29.13 | 10. - Aktuální | 33.5 | 16.32 | 24.04 |
| 11. - Původní | 39.5 | 34.97 | 11. - Aktuální | 80.4 | 81.09 | -103.54 |
| 12. - Původní | 35.5 | 15.14 | 12. - Aktuální | 37.1 | 18.48 | -4.51 |
| 13. - Původní | 38.5 | 16.99 | 13. - Aktuální | 40.2 | 28.91 | -4.42 |
| 14. - Původní | 30.6 | 10.47 | 14. - Aktuální | 48.8 | 65.4 | -59.48 |
| 15. - Původní | 10.9 | 3.45 | 15. - Aktuální | 9.5 | 2.17 | 12.84 |
| 16. - Původní | 16 | 9.79 | 16. - Aktuální | 13.2 | 2.6 | 17.50 |
| 17. - Původní | 74.2 | 82.96 | 17. - Aktuální | 26.1 | 13.10 | 64.82 |
| 18. - Původní | 127.3 | 184.66 | 18. - Aktuální | 39.3 | 53.26 | 69.13 |
| 19. - Původní | 36.7 | 19.28 | 19. - Aktuální | 14.8 | 8.93 | 59.67 |
| 20. - Původní | 33 | 23.87 | 20. - Aktuální | 18.8 | 15.26 | 43.03 |

Tabulka 6.3: Porovnání výsledků otázek

,kde $\sigma$ představuje směrodatnou odchylku a $\phi$ představuje průměr.

## 6.6 Funkční testování

Nebyly vytvořeny testy pro funkční testování a z tohoto ohledu aplikace není otestována. Avšak během experimentálního testování nedošlo k žádné chybě nástroje. Veškerá data byla načtena a zobrazena v pořádku.

# 7 Diskuze

Testování jednoznačně ukázalo, že nástroj je v nové verzi plně funkční a neprojevil žádné chování, které by vedlo k chybě. Byla zachována veškerá původní funkčnost nástroje. Co se týká funkční stránky nevidím důvod v budoucnu nějakým způsobem implementaci měnit.

Pokud porovnáme jednotlivé časy testů, vidíme že nová verze ve všech případech vede ke zlepšení časů. Uživatelům byla data zobrazena v náhodném pořadí, proto si myslím, že jsou výsledky relevantní. Domnívám se, že shluknutí informací vede k přehlednějšímu řešení a to z několika důvodů.

První z důvodů je, že shluknutím můžeme z časové osy vynechat jeden pás dat a tím dát větší prostor důležitějším datům. Toto je velmi důležité, protože nástroj rozděluje okno rovnoměrně mezi pásy. Tím, že dokážeme ubrat některé pásy a neztratit přitom informace o datech, dostaneme více místa pro prvky nesoucí důležitější informace. V mém případě jsem shlukl události do jednotlivých osob a tím jsem docílil toho, že jsem mohl zrušit pás "Události".

Další důvod je takový, že zmizí veškeré hrany z hlavní entity ke všem vedlejším. Tím jsem v mém případě ušetřil polovinu hran v grafu, toto si myslím, že mělo největší vliv na zlepšení výkonu nástroje. Dále díky tomuto se zobrazuje méně hran z entity a jsou viditelné hrany vedoucí k důležitějším datům. Výsledek je vidět na příkladu Obrázku 7.1 a Obrázku 7.2.

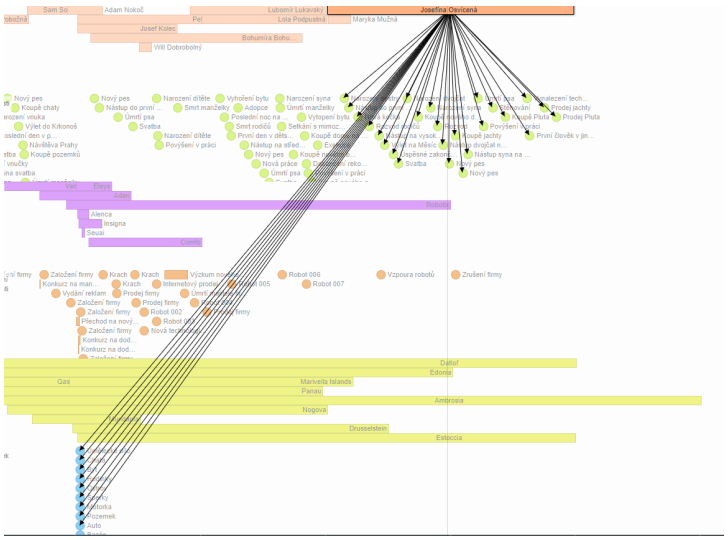
Nevýhodou tohoto přístupu je to, že byla nová verze navržena tak, aby se nedalo kliknout na pod-prvky entity. Tím se ztratí přímá informace o tom, jaké hrany z této entity vedou. Většina uživatelů testujících aplikaci byla zmatena pokud měli najít všechny entity vedoucí z určeného pod-prvku. Další nevýhodou je nutnost uživateli používající verzi s tímto shlukováním vysvětlit, co jaké pod-prvky znamenají a vytvořit jakousi legendu. V mém případě bylo třeba dovysvětlit všem účastněným, že pod-prvky skrývající se pod osobou jsou jejich události a pod-prvky skrývající se pod firmou jsou události náležící firmě. Pokud by taková informace nebyla uživateli řečena, uživatel by nevěděl, co jednotlivé prvky znamenají.

Další nevýhodou je to, že zmizí vizuální textová informace o pod-prvcích. Jakákoliv textová informace o pod-prvku je viditelná pouze pokud se na pod-prvek proklikneme skrze hranu.

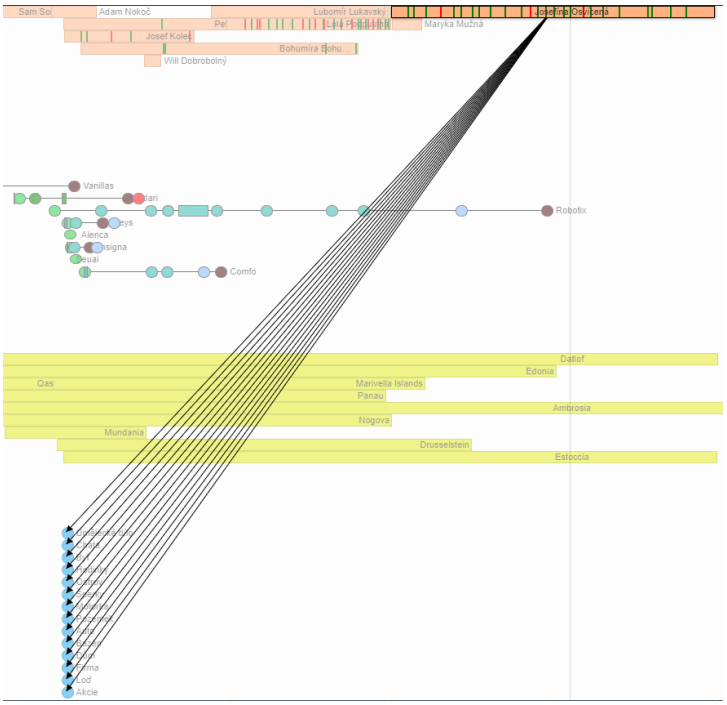
Pokud porovnáme jednotlivé časy otázek vidíme, že u většiny došlo ke zlepšení výsledků. Nejspíše to bylo zapříčiněno větším zobrazovacím prostorem pásů a zobrazením pouze důležitějších hran. Avšak u otázky číslo 11

došlo k ohromnému poklesu efektivity, z důvodu nevýhody kliknout na pod-prvky entity. Uživatel byl pak nucen hledat spojení k pod-entitě náhodným způsobem. Další pokles efektivity můžeme vidět u otázky číslo 14. V tomto případě není zřejmý důvod poklesu, jelikož je v obou verzích zobrazena identicky.

Myslím si, že nová verze nástroje má smysl a význam u dat, která se dělí na důležitá a méně důležitá. V tomto případě lze taková data shluknout. Pak si myslím, že lze dosáhnout efektivnější práce s daty a lepších výkonů nástroje. Nevýhody této verze v tomto případě nebudou žádné a nedojde k žádné ztrátě informace.



Obrázek 7.1: Původní verze dat




Obrázek 7.2: Aktualní verze dat

# 8 Závěr

Tato bakalářská práce popisuje novou verzi nástroje, která úspěšně prošla všemi testy bez jakékoliv chyby. Testy poukazují na to, že nová verze zvyšuje efektivitu práce s nástrojem. Tento nástroj nyní umí zpracovávat nový formát dat, který je detailně popsán a vysvětlen. Pro tento nový formát jsem implementoval dva nové renderery, které splňují veškeré požadavky a lze je použít na reálná data. Pokud budou třeba v budoucnu další renderery zpracovávající tento nový formát, lze je jednoduše implementovat. Stačí se držet pouze popisu nových rendererů a implementovat jejich hlavní metody. Tyto metody jsou popsány v kapitolách o renderech a jejich implementace by měla být nyní prostá díky tomu, že jsem vyřešil problém s počítáním pozice a šířky pod-prvků. V nové verzi je nyní možnost sjednocovat pruhy a tím vylepšovat zobrazovací schopnosti nástroje. Byla vytvořena komponenta, která umí sledovat aktivitu uživatele a lze díky ní rekonstruovat provedené testy. Tato komponenta nemusí sloužit jen pro testování aplikace, ale v budoucnu může být například stále aktivní a shromažďovat data o aktivitě. Díky těmto nashromážděným datům pak lze zlepšovat tento zobrazovací nástroj.

Hlavním cílem práce bylo zefektivnit zobrazovací schopnosti nástroje a tyto schopnosti pak otestovat. Tento cíl byl dosažen a výsledky ukazují, že se mi podařilo vylepšit celkovou efektivitu, celkový průměr a rozptyl časů odpovědí. Tento nástroj může být nyní využit pro jakákoliv data, která lze zobrazit do časové osy. Dále by se dalo na práci navázat z hlediska pod-prvků a to tím, že by se jim přidala další funkčnost a jiné zobrazovací atributy.

# Literatura


[1] Baudel, T. A Canonical Representation of Data-Linear Visualization Algorithms. *arXiv preprint arXiv:1412.4246*. 2014, s. 1–2.

[2] Craig, P. – Roa-Seïler, N. A vertical timeline visualization for the exploratory analysis of dialogue data. *Proceedings of the International Conference on Information Visualisation*. 2012, s. 68–73. ISSN 10939547.

[3] Cui, W. – Qu, H. A Survey on Graph Visualization. *The Journal of infectious diseases*. 2013, 208, s. NP. ISSN 1537-6613.

[4] Friendly, M. A brief history of data visualization BT - Handbook of Computational Statistics Data Visualization. *Handbook of Computational Statistics Data Visualization*. 2008.

[5] Jensen, M. Visualizing complex semantic timelines. *Derived from the World Wide Web: http://newsblip. com/tr*. 2003.

[6] Kacerovský, M. Vizuální reprezentace precedenčního grafu. 2015.

[7] Mackinlay, J. D. – Robertson, G. G. – Card, S. K. The perspective wall. 2003, s. 173–176.

[8] Rosenholtz, R. et al. Feature congestion: A measure of visual clutter. *Journal of Vision*. 2010, 6, 6, s. 827–827.

[9] Silva, S. F. – Catarci, T. Visualization of linear time-oriented data: A survey. *Proceedings of the 1st International Conference on Web Information Systems Engineering, WISE 2000*. 2000, 1, March, s. 310–319.

[10] Wikipedia. Timeline — Wikipedia, The Free Encyclopedia, 2019. Dostupné z: `https://en.wikipedia.org/wiki/Timeline`.

[11] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, C. T. *Visualisation of Time-Oriented Data*. Springer Science & Business Media, 2011. ISBN 0857290797.

[12] Zhao, J. et al. TimeSlice: interactive faceted browsing of timeline data. *Proceedings of the International Working Conference on Advanced Visual Interfaces SE - AVI '12*. 2012, s. 433–436.

# Přílohy

## Podrobné výsledky otázek

| Otázky | Subjekt 1 | Subjekt 2 | Subjekt 3 | Subjekt 4 | Subjekt 5 |
|---|---|---|---|---|---|
| 1. - Původní[s] | 55 | 35 | 16 | 19 | 39 |
| 1. - Aktuální[s] | 3 | 13 | 10 | 14 | 4 |
| 2. - Původní[s] | 24 | 53 | 18 | 22 | 20 |
| 2. - Aktuální[s] | 12 | 16 | 13 | 14 | 14 |
| 3. - Původní[s] | 47 | 78 | 58 | 32 | 95 |
| 3. - Aktuální[s] | 61 | 49 | 24 | 3 | 51 |
| 4. - Původní[s] | 27 | 26 | 23 | 58 | 39 |
| 4. - Aktuální[s] | 26 | 71 | 33 | 31 | 24 |
| 5. - Původní[s] | 11 | 8 | 19 | 9 | 15 |
| 5. - Aktuální[s] | 8 | 16 | 7 | 7 | 30 |
| 6. - Původní[s] | 134 | 31 | 56 | 61 | 391 |
| 6. - Aktuální[s] | 117 | 18 | 22 | 14 | 31 |
| 7. - Původní[s] | 20 | 21 | 31 | 14 | 28 |
| 7. - Aktuální[s] | 8 | 22 | 30 | 11 | 24 |
| 8. - Původní[s] | 11 | 19 | 12 | 33 | 13 |
| 8. - Aktuální[s] | 19 | 8 | 14 | 7 | 7 |
| 9. - Původní[s] | 9 | 13 | 13 | 16 | 24 |
| 9. - Aktuální[s] | 11 | 9 | 9 | 6 | 7 |
| 10. - Původní[s] | 52 | 30 | 36 | 125 | 29 |
| 10. - Aktuální[s] | 28 | 65 | 40 | 24 | 19 |
| 11. - Původní[s] | 21 | 36 | 132 | 13 | 13 |
| 11. - Aktuální[s] | 30 | 71 | 37 | 293 | 59 |
| 12. - Původní[s] | 70 | 36 | 24 | 30 | 43 |
| 12. - Aktuální[s] | 40 | 52 | 67 | 29 | 36 |
| 13. - Původní[s] | 39 | 31 | 40 | 66 | 48 |
| 13. - Aktuální[s] | 31 | 32 | 17 | 41 | 23 |
| 14. - Původní[s] | 29 | 29 | 34 | 37 | 24 |
| 14. - Aktuální[s] | 26 | 15 | 33 | 16 | 242 |
| 15. - Původní[s] | 8 | 9 | 13 | 7 | 8 |

| | | | | | |
|---|---|---|---|---|---|
| 15. - Aktuální[s] | 7 | 8 | 9 | 13 | 8 |
| 16. - Původní[s] | 10 | 13 | 15 | 20 | 13 |
| 16. - Aktuální[s] | 8 | 13 | 15 | 13 | 13 |
| 17. - Původní[s] | 40 | 56 | 94 | 316 | 53 |
| 17. - Aktuální[s] | 25 | 6 | 14 | 48 | 40 |
| 18. - Původní[s] | 39 | 65 | 61 | 137 | 671 |
| 18. - Aktuální[s] | 197 | 8 | 23 | 21 | 22 |
| 19. - Původní[s] | 29 | 30 | 40 | 35 | 40 |
| 19. - Aktuální[s] | 8 | 14 | 11 | 6 | 32 |
| 20. - Původní[s] | 14 | 40 | 75 | 18 | 30 |
| 20. - Aktuální[s] | 15 | 2 | 8 | 49 | 12 |

Tabulka 8.1: Podrobné výsledky otázek prvních 5 uživatelů

| Otázky | Subjekt 6 | Subjekt 7 | Subjekt 8 | Subjekt 9 | Subjekt 10 |
|---|---|---|---|---|---|
| 1. - Původní[s] | 45 | 10 | 5 | 37 | 61 |
| 1. - Aktuální[s] | 14 | 17 | 9 | 73 | 16 |
| 2. - Původní[s] | 27 | 36 | 8 | 39 | 27 |
| 2. - Aktuální[s] | 12 | 16 | 11 | 14 | 18 |
| 3. - Původní[s] | 90 | 35 | 13 | 64 | 147 |
| 3. - Aktuální[s] | 36 | 4 | 21 | 66 | 44 |
| 4. - Původní[s] | 32 | 34 | 9 | 76 | 31 |
| 4. - Aktuální[s] | 19 | 28 | 11 | 36 | 22 |
| 5. - Původní[s] | 10 | 10 | 9 | 14 | 13 |
| 5. - Aktuální[s] | 18 | 8 | 7 | 7 | 16 |
| 6. - Původní[s] | 24 | 65 | 12 | 68 | 36 |
| 6. - Aktuální[s] | 28 | 24 | 8 | 33 | 61 |
| 7. - Původní[s] | 31 | 16 | 7 | 44 | 19 |
| 7. - Aktuální[s] | 13 | 25 | 5 | 15 | 20 |
| 8. - Původní[s] | 25 | 20 | 10 | 44 | 17 |
| 8. - Aktuální[s] | 52 | 11 | 17 | 14 | 15 |
| 9. - Původní[s] | 22 | 13 | 6 | 20 | 21 |
| 9. - Aktuální[s] | 14 | 9 | 7 | 12 | 34 |
| 10. - Původní[s] | 30 | 20 | 26 | 35 | 58 |
| 10. - Aktuální[s] | 34 | 21 | 8 | 42 | 54 |
| 11. - Původní[s] | 25 | 66 | 9 | 46 | 34 |

| | | | | | |
|---|---|---|---|---|---|
| 11. - Aktuální[s] | 74 | 22 | 13 | 45 | 160 |
| 12. - Původní[s] | 43 | 26 | 10 | 43 | 30 |
| 12. - Aktuální[s] | 34 | 10 | 7 | 35 | 61 |
| 13. - Původní[s] | 43 | 7 | 12 | 46 | 53 |
| 13. - Aktuální[s] | 67 | 20 | 18 | 37 | 116 |
| 14. - Původní[s] | 35 | 32 | 16 | 54 | 16 |
| 14. - Aktuální[s] | 43 | 41 | 11 | 42 | 19 |
| 15. - Původní[s] | 8 | 10 | 14 | 14 | 18 |
| 15. - Aktuální[s] | 10 | 8 | 9 | 9 | 14 |
| 16. - Původní[s] | 18 | 8 | 7 | 13 | 43 |
| 16. - Aktuální[s] | 12 | 13 | 11 | 18 | 16 |
| 17. - Původní[s] | 41 | 21 | 21 | 56 | 44 |
| 17. - Aktuální[s] | 31 | 31 | 8 | 36 | 22 |
| 18. - Původní[s] | 90 | 23 | 9 | 86 | 92 |
| 18. - Aktuální[s] | 20 | 21 | 10 | 32 | 39 |
| 19. - Původní[s] | 45 | 10 | 5 | 63 | 70 |
| 19. - Aktuální[s] | 29 | 19 | 5 | 16 | 8 |
| 20. - Původní[s] | 34 | 8 | 3 | 75 | 33 |
| 20. - Aktuální[s] | 37 | 9 | 6 | 13 | 37 |

Tabulka 8.2: Podrobné výsledky otázek posledních 5 uživatelů

# Uživatelská dokumentace

## Zprovoznění

Pro základní prohlížení nového formátu dát stačí otevřít soubor /src/index-sub-items-source.html.

## Úprava dat

Pro úpravu dat je třeba upravit soubor /src/data/data.js. Datový soubor může obsahovat definice původních entit viz [6] a nebo novou definici viz **Ukázky nového formátu entit**.

### Definice pásů

Pokud chceme konfigurovat rozložení a typy pásů je třeba upravit soubor /src/data/BandsDistribution.js.

Příklad rozložení pásů:

```
bands : [
 {
 id: "person",
 label: "Lide",
 itemRenderer: new BandItemRenderer("#FFB182"),
 color: "#fafafa"
 },
 {
 id: "Sjednoceny pruh",
 label: "Sjednoceni",
 types: [
  {
   id: "event",
   itemRenderer: new BandItemRenderer("#F2BC53"),
   color: "#f5f5f5"
  },
  {
   id: "item",
   itemRenderer: new BandItemRenderer("#78B4FF"),
   color: "#fafafa"
  }
 ]
 }
]
```

Toto vytvoří jeden pás "Lide"pro typ *person* a druhý pás "Sjednoceni"pro typy *event* a *item*.

## Testovací data

Testovací data jsou obsažena ve složce /testing/test-data. Ve složce /testing/test-data/New-data jsou obsažena data nového formátu spolu s potřebným rozložením pásů BandsDistribution.js. Ve složce /testing/test-data/Old-data jsou obsažena data předcházejícího formátu spolu s potřebným rozložením pásů BandsDistribution.js. Pokud chceme data použít, stačí nakopírovat obsah jedné ze složek do složky /src/data.

## Detailní výsledky testů

Výsledky jednotlivých testů jsou obsaženy ve složce /testing/test-results. Každý uživatel má svoji vlastní složku, ve které se nacházejí složky New a Old. Ve složce New se nachází výsledek testu pro novou verzi nástroje a ve složce Old se nachází výsledek testu pro starou verzi nástroje.

## Definice otázek

Otázky jsou definované v souboru /src/data/questions.js. Soubor obsahuje pole těchto otázek v JSON formátu. Pro přidání nebo změnu otázek jednoduše použijeme tento formát:

```
{
"text" : <String>,
"answer" : <String>
},
```

Příklad jedné otázky vypadá následovně:

```
{
 "text" : "V jakem roce zemrel Walfred Porto?",
 "answer" : "1950"
},
```

## Ukázky nového formátu entit

Nový formát entit je popsán v kapitole 5.1.1. Zde jsou uvedeny příklady a výsledky nového formátu.

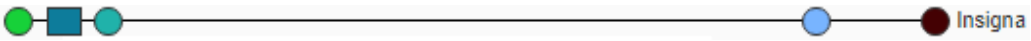
```
{"id":228,"stereotype":"firm","name":"Insigna",
 "description":"Firma zamerena na prodej her",
 "begin":"2001-03-10T00:00:00",
 "end":"2010-06-25T00:00:00",
 "properties":{"startPrecision":"day",
               "endPrecision":"day"},
 subItems:[
        {"id":229,"name": "Zalozeni firmy",
        "type":"start",
        "css":"zaloz-firmy"},
        {"id":230,"name": "Prodej firmy",
        "type":"end",
```

```
        "css":"prodej-firmy"},
        {"id":231,"name": "Konkurz na dodavatele",
        "begin":"2001-08-06T00:00:00",
        "end":"2001-12-12T00:00:00",
        "css":"konkurz"},
        {"id":232,"name": "Internetovy prodej",
        "begin":"2002-01-21T00:00:00",
        "css":"nova-tech"},
        {"id":233,"name": "Krach",
        "begin":"2009-01-08T00:00:00"},
]},
```

A pokud je takto definován pás pro entity typu *firm:*

```
{
 id: "firm",
 label: "Firma",
 itemRenderer: new DumbbellItemRenderer("#B942F4"),
 color: "#fafafa"
},
```

Tato entita se pak zobrazí takto:




Obrázek 8.1: Výsledek entity

Pozor, barev bylo dosaženo nadefinováním příslušných css tříd.
Další nový typ formátu entity vypadá následovně:

```
{"id":20,"stereotype":"person","name":"Bohumira
   Bohumilna",
 "description":"","begin":"2005-11-12T00:00:00",
 "end":"2090-10-17T00:00:00",
 "properties":{"startPrecision":"day",
               "endPrecision":"day"},
 subItems:[
  {"id":21,"name": "Lecba",
   "begin":"2080-08-14T00:00:00",
   "end":"2090-10-17T00:00:00",
   "css":"dobra-udalost"},
  {"id":22,"name": "Koupe domu na Marsu",
   "begin":"2089-10-14T00:00:00",
   "css":"spatna-udalost"},
  {"id":23,"name": "Narozeni ditete",
```

```
    "begin":"2030-12-06T00:00:00",
    "css":"dobra-udalost"},
   {"id":24,"name": "Povyseni v praci",
    "begin":"2031-07-12T00:00:00",
    "css":"dobra-udalost"},
]},
```

A pokud je takto definován pás pro entity typu *person:*

```
{
 id: "person",
 label: "Lide",
 itemRenderer: new SplitBandItemRenderer("#FFFFFF"),
 color: "#fafafa"
},
```

Tato entita se pak zobrazí takto:


Obrázek 8.2: Výsledek entity