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

Bakalářská práce

Vizualizace časové osy

Plzeň, 2017 Daniel Holubář

Obsah

1 Úvod

Na internetu se dnes dá najít obrovské množství informací. Stačí chvíli hledat a nalezneme téměř cokoliv. Také můžeme diskutovat s lidmi z celého světa o různorodých problémech a tématech. Všichni lidé s přístupem na internet tak mají přístup k nepřebernému množství dat.

Při takovém množství dat je velmi důležité, jakým způsobem jsou data prezentována. Dobře strukturovaný obsah webové stránky pomůže člověku v hledání odpovědí a v čerpání znalostí, proto jsou kladeny požadavky na přehlednost a srozumitelnost. Přehledně prezentované a dobře zpracované informace jsou proto na internetu téměř nutností.

Tato bakalářská práce staví na již hotových pracích studentů Západočeské Univerzity. Slečna Gabriela Hessová zpracovala problematiku automatického získávání historických údajů z webových zdrojů [5]. Pan David Hrbáček se zabýval zpracováním těchto údajů a ohodnocením jejich důležitosti při zobrazení uživateli [6]. Pan Michal Kacerovský [7] potom vytvořil nástroj pro přehledné zobrazení všech získaných dat a souvislostí mezi nimi pomocí časové osy. Pan David Merunko potom zpracoval problematiku generování udržování a zpracování časové osy pomocí samostatné aplikace v programovacím jazyku Java [8].

Cílem práce je vytvoření aplikace, která zastřeší všechny předchozí práce. Tato aplikace bude zobrazovat časovou osu pro vyobrazení historických dat, dále bude obsahovat nástroj pro vytváření testů, pomocí kterých se bude zkoumat, zda je vizualizace dat prospěšná a pomůže člověku k rychlejšímu pochopení a naučení se historických událostí. Součástí bude i testování aplikace.

2 Technologie pro tvorbu webových aplikací

Díky rychlému rozvoji existuje i mnoho technologií, které umožňují tvorbu webových aplikací. Mezi těmito je vhodné vybrat takové technologie, díky kterým bude aplikace kvalitní, rychlá a robustní. Pro bakalářskou práci jsem kromě základních pro tvorbu webů vybral následující technologie:

- **MySQL** - SQL - Structured Query Language
- **PHP** - Hypertext Preprocessor (původně Personal Home Page)

2.1 MySQL

Databázový systém postavený na jazyku SQL, který uplatňuje relační databázový model. Mezi přednosti této technologie patří zejména rychlost a rozšiřitelnost. Jedná se o databázový systém, který zná naprostá většina programátorů. Komunikace s touto databází probíhá pomocí SQL dotazů. Pro účely práce je MySQL naprosto dostačujícím databázovým nástrojem. Při porovnáním s PostgreSQL se klady druhého zmiňovaného nástroje projevují až při velmi obsáhlých databázích a složitějších dotazových konstrukcích.

2.2 PHP

Jedná se o velmi rozšířený interpretovaný skriptovací jazyk, který se používá k tvorbě dynamických webových aplikací. Dynamické aplikace jsou nejprve zpracovány serverem. To znamená, že PHP skript je spuštěn na straně serveru a jeho výsledek je poté poslán uživateli. Je to velmi rozšířený jazyk, jehož syntax je inspirována několik staršími programovacími jazyky, například C a Perl. Obsahuje velké množství knihoven pro práci s různými databázovými systémy, dále knihovny pro zpracování grafiky, přenos a ukládání souborů a zpracování textů. Mezi hlavní přednosti patří přenositelnost a jednoduchost, díky čemuž je dodnes jedním z nejvybíranějších jazyků pro tvorbu webových aplikací [4], kdy pokud vezmeme v potaz programovací jazyky, které se běžně pro tvorbu webových aplikací vybírají, se před PHP nachází pouze Java, C# a Python. Tento žebříček je z hlediska webových aplikací pouze informativní, neboť se jedná o celkové využití programovacího jazyka.

2.2.1 PHP Framework - Laravel

Tento framework patří mezi nejoblíbenější nástroje vybírané pro tvorbu PHP aplikací. V posledních letech zažívá rozmach proti ostatním známým frameworkům [3]. Mezi jeho výhody patří zejména rychlé vytvoření základní aplikace, která v sobě již obsahuje základní login a registraci uživatelů. Pomocí širokého spektra příkazů pro příkazovou řádku [2] lze rychle vytvořit potřebné modely, controllery, migrace pro databázi a také vložení základních dat do databáze. Programátor se tak nemusí

zabývat ruční tvorbou tříd nebo tabulek v databázi a vše potřebné se tvoří automaticky.

3 Stávající nástroje pro vizualizaci časové osy

Existující řešení řešení zkoumal ve své práci již Michal Kacerovský. Bude tedy vhodné rozdělit seznam stávajících řešení na ta zkoumaná již dříve proti těm, které existují v dnešní době.

3.1 Řešení zkoumaná Michalem Kacerovským

3.1.1 TimelineJS

Jak bylo psáno již dříve, jedná se o službu, která podle vstupních dat vygeneruje časovou osu. Během vytváření časové osy mu byla inspirací. Tato služba je stále aktivní.

3.1.2 Dipity

Tato služba byla obdobná jako služba TimelineJS s tím rozdílem, že uživatelé museli být registrováni. tato služba již není aktivní.

3.1.3 Timglider

Oproti předchozím produktům nabízí Timeglider možnost využití jQuery widgetu se základními nástroji pro práci s osou, jako je vkládání nových záznamů, editace nebo přiblížení časové osy pomocí kolečka myši. Timeglider je stále aktivní.

3.1.4 vis.js

Jedná se o knihovnu s mnoha různými službami, mezi nimiž je i služba vizualizace časovou osou. Vis.js nabízí i možnost forku, díky čemuž si ji může uživatel přizpůsobit pro své účely nebo rozšířit podle svých představ. Tato knihovna je stále aktivní.

3.1.5 Timeline (SimileWidgets)

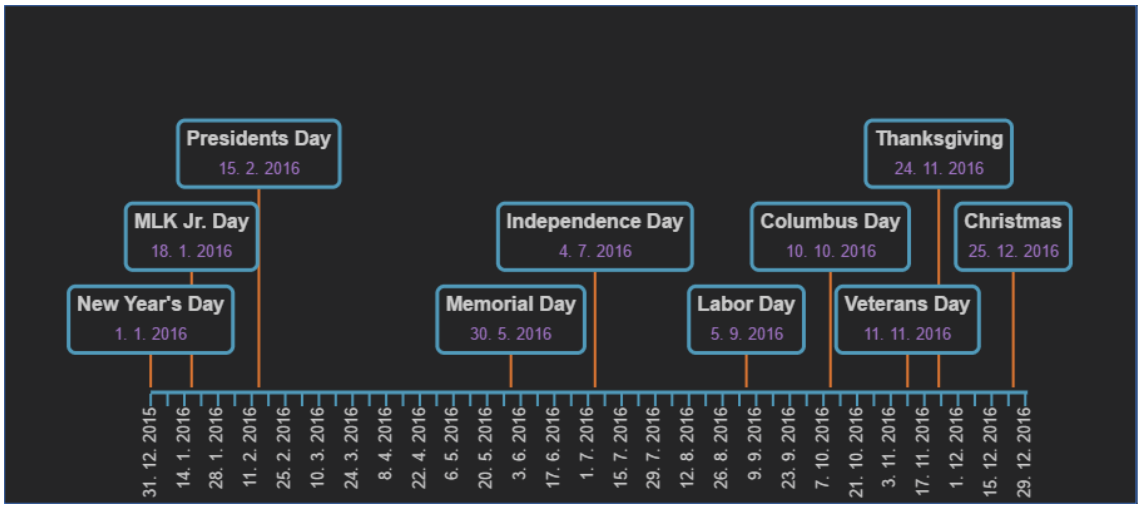
Vývoj této knihovny byl ukončen již v roce 2009. Tento stav stále trvá.

3.2 Řešení dostupná v dnešní době

Díky rozvoji moderních internetových technologií jako jsou HTML5 a CSS3 je množství služeb a knihoven pro zobrazení časové osy obrovské. Z toho důvodu uvedu jen několik pro mne nejzajímavějších.

3.2.1 GoJS

Knihovna GoJS nabízí velké množství nástrojů a možností vizualizace. Jednou z nich je i časová osa. Obsahuje mnoho návodů a ukázek, díky čemuž je její použití snadné. Pro osobní účely je tato knihovna zadarmo, pro komerční účely se musí zaplatit.



Obr. 1: Příklad použití knihovny GoJS

Knihovna disponuje přehlednou dokumentací a jednoduchou implementací. Data se vkládají pomocí JSON formátu a osa samotná se dá velmi jednoduše roztahovat nebo naopak smršťovat. Záznamy na ní včetně časových údajů se dynamicky přizpůsobují těmto změnám. Po ose se dá pohybovat myší nebo klávesami.

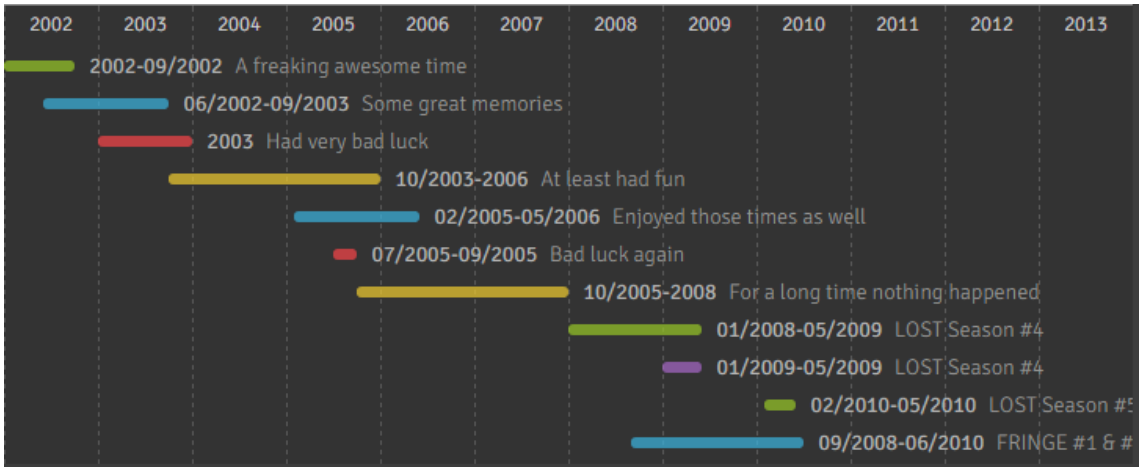
Do časové osy není možné vkládat obrázky a osa nelze přibližovat ani oddalovat. Jednotlivé záznamy není možné žádným způsobem spojovat.

Knihovna GoJS je užitečným nástrojem pro jednoduché a přehledné časové osy, avšak díky absenci některých nástrojů a možností se stává nevhodnou pro aplikaci na časové ose zpracovávané panem Kacerovským.

Vydavatel	Northwoods Software Corporation
URL	http://gojs.net
Licence	https://gojs.net/latest/doc/license.html
Aktuální verze	1.7

3.2.2 Timesheet.js

Tato knihovna nabízí grafickou vizualizaci časové osy pomocí HTML5 a CSS3. Optimalizace na mobilní zařízení je pro tuto knihovnu samozřejmostí.



Obr. 2: Příklad použití knihovny Timesheet.js

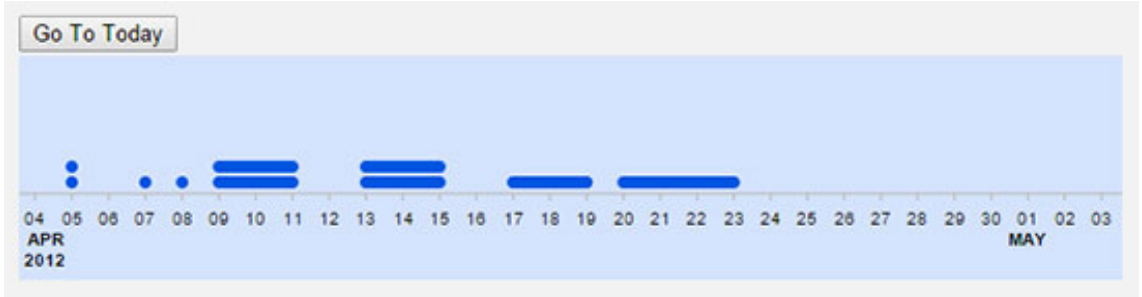
Při bližším zkoumání knihovna umí pouze to, co je na obrázku, znázorňuje časové úseky, a i když přehledně a se zajištěním nepřekrývání jednotlivých událostí, neumí například vložit obrázek, nedokáže pospojovat události ani neumožňuje manipulace jako jsou zoom a ovládání pomocí kláves.

I přes moderní zpracování a přehledný design je tato knihovna pro naše účely nepoužitelná.

Vydavatel	Sebastian Müller
URL	https://sbstjn.com/timesheet.js
Licence	MIT
Aktuální verze	—

3.2.3 Chronoline.js

Časová osa stejně jako předchozí umožňuje zobrazení pouze v jednoduché formě, přestož oproti předchozí nabízí více možností a nástrojů.



Obr. 3: Příklad použití knihovny Chronoline

Knihovna nabízí ovládání pomocí myši, dále umí i zoomovat, k čemuž však potřebuje externí formulář, do kterého se vloží číselná hodnota zoomu. tato knihovna také umí

vkládat dva druhy událostí. Jedny se zobrazují jako barevné pásy o výšce celé časové osy, jedná se o dlouhodobé události. Kratší události jsou zobrazovány pomocí čar se zaoblenými konci. Podle informací na stránce se jedná primárně nástroj vytvořený na míru pro stránky https://zanbato.com/.

Této knihovně opět scházejí nástroje, které bychom potřebovali pro naši časovou osu, tedy spojování událostí, ovládání pomocí klávesnice nebo zobrazení dodatečných informací při najetí na událost.

Vydavatel	StoicLoofah
URL	http://stoicloofah.github.io/chronoline.js/
Licence	MIT
Aktuální verze	—

I přes přehršel knihoven a služeb pro tvorbu časových os se mezi nimi nevyskytuje žádná, která by splňovala dané požadavky, a která by zvládala komplexně a přehledně zobrazit informace spolu se závislostmi. Výše zmíněné byly vybrány s cílem poukázat pravě na tyto nedostatky i přes použití moderních technologií a nejnovějších postupů.

4 Souhrn ostatních prací

V této sekci se budu zabývat tím, co bylo doposud vytvořeno, tedy pracemi slečny Gabriely Hessové a pánů Michala Kacerovského, Davida Hrbáčka a Davida Merunka.

4.1 Automatické získání historických údajů z webových zdrojů [5]

Cílem práce Gabriely Hessové bylo získání dat z internetových zdrojů a jejich zpracování pro aplikace, které s těmito daty pracují. Práce se tedy zabývá metodami získávání dat z textu z různých webových zdrojů. Jako hlavní zdroj pro data byla vybrána Wikipedie.

V druhé části práce se autorka zabývá praktickou implementací nástroje pro získání dat z webových zdrojů. Velikosti souboru s daty z WIkipedie má řádově desítky GiB(46,7 GiB v době tvroby práce slečny Hessové [5, str. 11]). To má za následek prodlevy, které autorka řeší pomocí indexu. Jedná se o pomocnou datovou strukturu díky které je vyhledávání rychlejší [5, str. 21].

Pro získání dat Není využit žádný již realizovaný nástroj jako je SAX nebo DOM, autorka problém řeší sekvenčním čtením dat a jejich zpracováním. Při zpracování se data ukládají do relační databáze, v dalším kroku se data převedou do grafové databáze vytvořené panem Merunkem.

4.2 Generování a vizualizace časové osy [8]

Cílem práce Davida Merunka bylo vytvoření grafové databáze, která by byla uložena na serveru. Dále bylo třeba vytvořit rozhraní pro přístup k této databázi. Poslední částí bylo vytvoření aplikace pro zobrazení časové osy pomocí dat uložených na serveru.

Při práci s grafy je nutné správně volit algoritmy, zejména pro určení metrik jako jsou přehled o náročnosti vyhledávání a informace o velikosti grafu. Autor v práci metriku využívá pro určení relevantnosti vrcholů vůči sobě [8, str. 23]. Pro určení důležitosti vrcholu potom využívá stupeň vrcholu, tedy počet hran, které vedou do/z vrcholu [8, str. 23]. Autor dále uvádí principy a mechaniky grafové databáze, její rozdíly proti relačním databázím a porovnává je z hlediska výkonosti [8, str. 26].

Jako databáze byla vybrána Neo4j. Jedná se o grafovou databázi, která má všechny potřebné vlastnosti. Při procházení databáze se dá vybrat ze dvou přístupů: do hloubky a do šířky. Pro databázi je nutné také správně určit typy vrcholů, čemuž

se autor věnuje v kapitole 7.2 [8, str. 34]. Vrcholy jsou vybrány čtyři.

- **Person** - typ vrcholu pro důležitou historickou osobnost, který udržuje vlastnosti spjaté s životem osobnosti [8, str. 35].
- **Event** - typ vrcholu pro důležité historické události a data s nimi spjatá. [8, str. 35]
- **Place** - typ vrcholu pro důležitá historická místa a politické útvary. Tento typ vrcholu není zobrazen na časové ose, nicméně slouží jako obalovací vrstva pro ostatní vrcholy [8, str. 35].
- **Item** - typ vrcholu pro historické vynálezy, dokumenty a technologie [8, str. 34].

K těmto vrcholům jsou dále navrženy vztahy, aby co nejlépe definovaly souvislosti mezi jednotlivými vrcholy.

- **Relationship** - vztah pro vrchol typu Person nebo Place, tedy například pro definování vztahu mezi matkou a potomkem, nebo přerod státu do jiného státu [8, str. 37].
- **Interaction** - tento typ vztahu byl vytvořen pro definování souvislostí mezi dvěma vědci, kteří spolu spolupracovali nebo státníky, kteří jednali o důležitých dohodách mezi svými státy [8, str. 37].
- **Participation** - tento vztah představuje účast osoby na události nebo na určitém místě [8, str. 37].

- **Creation** - definuje vznik místa, vynálezu nebo narození důležité osoby nebo počátek události [8, str. 37].
- **Cause** - vztah definuje příčinu vzniku vrcholu ve vztahu k jinému vrcholu [8, str. 37].
- **Part_of** - tento typ vztahu definuje vrcholy, které jsou součástí většího celku [8, str. 37].
- **Takes_place** - souží pro určení vztahu mezi událostí a místem, kde se konala [8, str. 37].

Databázové API je zodpovědné za ukládání načítání a vyhledávání dat. Autor bere v úvahu i možnosti vyhledávání. Za tímto účelem byly vytvořeny odpovídající metody popsány v 8.1.1 [8, str. 40]. Tyto metody zajišťují vytvoření databáze, načtení databáze a odpojení databáze. Dále obsahují logiku pro prohledávání grafů podle zadaných parametrů, vkládání vrcholu, editaci vrcholu, odstranění vrcholu. Metody pro vkládání, editaci a odstranění jsou vytvořené i pro hrany. API obsahuje i další pomocné metody pro manipulaci s grafem nebo hranou.

4.3 Zpracovnání časových údajů pro jejich vizualizaci [6]

Cílem práce Davida Hrbáčka bylo vytvoření algoritmu pro ohodnocování částí grafu podle hodnot zadaných uživatelem. Dále měl autor vytvořit REST server, který by komunikoval s grafovou databází a poté jak s klientem webovým, tak javovskou aplikací.

Jako jedna z možných metrik se může používat tzv. centralita uzlu. Tato metrika se rozděluje na následující druhy:

- **Degree centrality** - u této metriky je důležitost uzlu určena jeho stupněm, tedy čím více sousedů vrchol má, tím důležitější je [6, str. 5].
- **Closeness centrality** - tato centralita je založena na vzdálenosti vrcholu od ostatních uzlů, kdy s větší vzdáleností klesá jeho důležitost [6, str. 5].
- **Betweenness centrality** - tato metrika je založena na počtu nejkratších cest mezi dvěma vrcholy, na kterých leží pozorovaný vrchol [6, str. 5].

Úpravou v používání výše zmíněné metriky získáme ohodnocení grafu pomocí sady kritérií [6, str. 5]. Tato metoda určuje důležitost a celkovou úlohu uzlu v rámci grafu podle jeho vlastností.

Poslední možností ohodnocení uzlu v práci je PageRank [6, str. 5]. Právě tento si autor vybral k implementaci pro účely své práce. Algoritmus patří mezi známější zejména díky společnosti Google, která ho využívala pro ohodnocování důležitosti

webových stránek. Výhodou tohoto algoritmu oproti předchozím metrikám je, že nebere vrchol grafu samostatně, ale v souvislosti s okolními vrcholy.

Knihovna pro práci s grafem je inspirována již existující knihovnou GraphStream, konkrétně je tedy rozdělena do dvou samostatných celků - graph a graph-ranking. První slouží k implementaci grafu a druhá poté obsluhuje ohodnocování [6, str. 23].

Knihovna graph je koncipována tak, že hrana i uzel je reprezentován rozhraním, což zamezuje možnost zdvojení hran nebo vrcholů. V knihovně je též ošetřena potřeba otestování, zda hrana propojuje existující vrcholy. Samozřejmostí jsou potom metody pro vkládání a mazání vrcholů a hran. Graf je tedy reprezentován pomocí rozhraní, což reflektují třídy a rozhraní implementující knihovnu [6, str. 26].

Pro samotný algoritmus PageRank byla vybrána možnost výpočtu pomocí spojových seznamů. Na rozdíl od maticového výpočtu se při tomto výpočtu počítá důležitost každého uzlu zvlášť, což je výhodné zejména kvůli řídkosti matice přechodů používané ve výpočtu pomocí matic. Algoritmus taktéž zahrnuje i uživatelem zadané priority.

Knihovna pro vnější přístup obstarává komunikaci s databází a s klientem. Uživatel má možnost zadat parametry pro zobrazení grafu historických událostí, knihovna data od uživatele vezme a zavolá metodu, jejíž parametry jsou právě data od uživatele. Metoda potom vrací graf. Mezi další možnosti patří získání podgrafu, vložení hrany či vrcholu a získání dat uložených ve vrcholu.

Vytvořený REST server poskytuje data ve formátu JSON, což má kladný vliv na rychlost komunikace. Metody serveru kopírují metody knihovny pro vnější přístup. Server tedy získá data z knihovny, převede do formátu JSON a následně odešle.

Samotná implementace serveru byla realizována pomocí frameworku Spring MVC 4. důvodem výběru bylo:

- Snadná implementace REST rozhraní pomocí anotací.
- Hotové řešení pro JSON formát.
- Získávání objektů pomocí návrhového vzoru IoC.
- Velká základna uživatelů.

4.4 Vizuální reprezentace precedenčního grafu [7]

Cílem práce Michala Kacerovského je vizuální reprezentace časové osy pomocí webového widgetu. Práce navazuje na práce zmíněné výše. Jedná se o koncový produkt celého zpracování historických dat.

Nejprve je nutné definovat časovou osu a upřesnit její vlastnosti spolu s požadavky, které jsou na ní kladené. [7, str. 3]. Protože práce úzce navazuje na práci pana Merunka, Využívá pan Kacerovský stejný typ vrcholů a vztahů definovaných v analýze práce pana Merunka.

Jak autor popisuje v kapitole 4.1 [7, str. 8], vývoj webového widgetu grafického charakteru může být chápán jako analogie k grafickým aplikacím realizovaným pomocí jiných programovacích jazyků, například Java a C++. Tím je myšleno, že tak jako grafické aplikace využívají rozdělení na vrstvy, kdy jedna vrstva mapuje data na objekty, další vrstva obstarává grafickou podobu a poslední vrstva rozhoduje o vykreslování, tedy co má uživatel vidět. Stejným způsobem můžeme realizovat i webový widget grafického charakteru. Pomocí canvasu, může programátor prezentovat jakoukoliv grafiku, ovšem za cenu ztráty možnosti interakce, protože vyobrazení převádí data na pixely. Na proti tomu je zde možnost použití technologie SVG, která vytvoří mimo jiné vrstvu mezi programátorem a vykreslováním pixelů. Protože tato technologie pracuje s DOM, je zde pro programátora stále možnost zachycení akcí nad vykreslovanými objekty [7, str. 8].

Při práci s časovou osou je nutné dát si pozor na určité věci spojené s časem [7, str. 9]. Například ISO 8601, což je norma pro časový formát, nebere rok 0 v potaz, to znamená, že tento rok neexistuje v pojetí zobrazení času. Je nutné na to tedy dát pozor a při práci vyřešit problémy s tím spjaté. Mezi další problémy patří například letní a zimní čas a další podobné anomálie, o kterých se autor podrobně zmiňuje ve své práci.

Pro tvorbu své práce autor vybírá HTML5 a CSS3. Jedná se o nejnovější verze těchto technologií, které disponují mnohými nástroji, jenž předčí první dvě zmiňované technologie. Podstatnou podmínkou je v neposlední řadě také to, aby byl widget responzivní, tedy aby dokázal reflektovat hustotu pixelů a rozlišení zobrazovacího zařízení a dokázal tak data zobrazovat stále přehledně. Protože widget komunikuje se serverovou částí, musí být stanoven způsob přenosu dat. V dřívější práci byl vybrán formát JSON, čímž je formát fakticky stanoven a autor to musí vzít v potaz.

Aby bylo grafické rozhraní přehledné a dobře se v něm orientovalo, je jako způsob grafického zobrazení vybrán systém se souběžnými pásy, kdy každý pás reprezentuje jiný typ události. Události jsou dále chronologicky seřazené, což je předpoklad samotné časové osy. Data jsou tedy řazena od nejstarších po nejnovější ve směru zleva doprava. Mezi událostmi, které jsou vůči sobě nějakým způsobem závislé, je jako reprezentace tohoto vztahu zobrazena čára. Protože kompletní zobrazení vztahů by bylo nepřehledné, má uživatel možnost procházet podmnožinu těchto vztahů [7, str. 14]. Jako ovládací prvky jsou vybrány logicky klávesnice a myš. Uživatel má možnost osu ovládat pomocí myši jednak kolečkem pro přiblížení a metodou drag and drop může osu posouvat podle své potřeby. Na klávesnici slouží

pro přiblížení klávesy + a -, zatímco pro posun osy se využívají tlačítka kurzorových šipek.

Nedílnou součástí widgetu je možnost filtrování informací. Filtry nejsou součástí vykreslovacího widgetu, a musejí být realizovány jako prostředek, který přistupuje přímo k datům, která widget zobrazuje. Autor dále uvádí některé možnosti filtrování [7, str. 16].

V další části autor navazuje na práce kolegů a stejně jako oni i on provádí výčet existujících služeb pro vykreslování časové osy a diskutuje jejich přednosti a nedostatky. Stejně jako oni i on dochází k závěru, že pro svou práci navrhne knihovnu novou, tak, aby vyhovovala všem stanoveným požadavkům. Dále widget zapojuje mezi práce ostatních, protože David Hrbáček [6] a David Merunko [8] vytvořili další části z celého projektu, konkrétně určování priorit a důležitosti vrcholů a hran a databázi pro ukládání grafu, který reprezentuje tuto časovou osu.

V kapitole uživatelského rozhraní(7.3) [7, str. 25] se autor zabývá rozložením a vzhledem samotného widgetu. Jak již bylo zmíněno, aplikace bude zobrazovat více souběžných pásů pro různé druhy událostí. Je nutné, aby pásy reflektovaly i souběžné události a přizpůsobovaly se svému obsahu. Samotná časová osa pro přehled časového charakter je umístěna ve spodní části widgetu. Je nutné, aby začátek a konec zobrazovaných dat správně začínal a končil i podle časové osy. Nástroj je tedy vytvořen tak, aby bral v ohled různá rozlišení a přibližování a oddalování. Autor detailně popisuje způsob výpočtu [7, str. 25] a řešení tohoto problému. Plocha celého widgetu je zhruba trojnásobná ploše viditelné na zobrazovacím zařízení. Autor tedy řeší i to, že některé věci vidět nejsou a není tedy třeba je zobrazovat uživateli [7, str. 32]. Případné souběžné události jsou řešeny pomocí Lane algoritmu. ten obstarává to, aby se souběžné události nepřekrývaly a aby události byly správně řazeny [7, str. 33]. Aby se pásy správně vykreslovaly, jsou obaleny do skupiny pásů. Při posouvání a přibližování osy se výpočty pásů provádějí přes tuto skupinu. Jejím dalším úkelm je i distribuce volného místa. To znamená, že se zjistí, kolik prostoru pro vykreslení je na šířku a výšku a tento prostor je rozdělen mezi všechny pásy. Jako abstrakce události je uvedena položka pásu. tato je svázána s objektem, který nese data pro tuto položku na časové ose. Každá událost je poté rozdílný způsobem vykreslena podle svých vlastností a svého typu, aby měl uživatel na první pohled přehled [7, str. 35]. Součástí vykreslení je jsou i vztahy. Ty, jak již bylo řečeno dříve, nejsou vykreslovány najednou, ale vykreslí se vždy podmnožina podle výběru události uživatelem. Součástí zpracování je i různé pojetí zoomu podle použití klávesnice nebo myši. Při použití klávesnice se mění měřítko celého zobrazení. Při použití myši se potom bere v úvahu i její pozice, kdy pozice kurzoru určuje střed celé transformace. Předpokládá se, že když je myš na nějaké události má při přiblížení nebo oddálení být tato pozice brána jako střed. Další funkcí widgetu je zaostření. to funguje tak, že pokud je nějaká událost vybrána pro zaostření, widget postupně zoomuje do té doby, než využije maximální

prostor pro zobrazení dané události.

Důležitou součástí widgetu je i objekt datového zdroje, který má za úkol komunikaci mezi widgetem a zdrojem dat. Musí umět přijímat data ze vzdáleného zdroje, nebo ze statického JSON souboru. Dále musí získaná data převést do formátu vhodného pro časovou osu, která je následně zobrazí.

Pro realizaci widgetu autor využívá knihovny již existující. Konkrétně jQuery a jQueryUI. Obě knihovny ulehčují práci programátora již hotovými řešeními pro jazyk JavaScript. Knihovna jQueryUI je rozšířením první knihovny, která obsahuje navíc nástroje pro práci s grafickým rozhraním. Další knihovnou je RequirejS, jejímž úkolem je načítání skriptů až ve chvíli, kdy jsou potřeba,čímž urychluje načítání stránek. Moment.js slouží pro práci s časem, řeší mnoho problémů spjatých s formátem data a jeho manipulací. Nástroj pro vykreslení vztahů mezi událostmi se jmenuje Snap.svg. Ten umožňuje práci se SVG na vysoké úrovni. Omezení je pouze to, že funguje na moderních prohlížečích s podporou nejnovějších funkcí. Pro další práci autor vytvořil několik svých podpůrných modulů:

- **oop.js** - tento podpůrný modul maskuje odlišný přístup JavaScriptu k vytváření tříd. Zavádí typ Closure, který uchovává kód funkce společně s daty nebo prostředím [7, str. 49]. Dále zavádí typ Class, díky kterému se dá v JavaScriptu vytvořit třída stejně jako například v Javě. Další vlastností je i možnost dědění [7, str. 50]. Díky tomuto modulu lze definovat i rozhraní. V podstatě jsou to jednoduché objekty s prázdnými třídami [7, str. 52].
- **moment.extension.js** - totorozšíření již existující výše zmíněné knihovny doplňuje dodatečné funkce z oblasti formátování času pro potřeby osy [7, str. 52].

Aplikace je rozdělena do logických celků v rámci adresářové struktury. Ve složce css jsou uloženy styly pro časovou osu. Ve složce data je uložen skript pracující s daty. Ve složce js jsou veškeré obslužné skripty aplikace ve složce auxiliary, ve složce lib jsou knihovny třetích stran a ve složce cz jsou hlavní soubory pro práci s daty, obecné třídy a rozhraní, pásy a položky pásů a rendery a pomocné třídy. Soubor index.html je potom v nejvyšší vrstvě. [7, obr. 8.2, str. 53]. Na straně [7, str. 54] začíná stručný popis jednotlivých tříd a rozhraní v rámci widgetu.

5 Návrh aplikace

Základní částí aplikace je widget pro časovou osu, kterou vytvořil pan Kacerovský. Aplikace samotná se bude sestávat z více částí, které obstarávají funkcionalitu podle zadání:

- Správa uživatelů

- Vytváření vlastní časové osy
- Vytváření testů
- Realizace testů

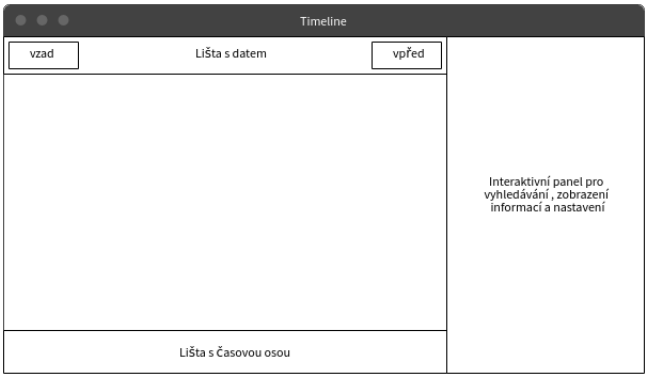
Z výše uvedených dat vyplívá, že aplikace bude muset být rozdělena na dvě části. První část, uživatelská, bude fungovat jako jednoduchá prezentace časových os a testů, případně bude obsahovat základní informace. Nejdůležitější součástí bude správné začlenení widgetu.

5.1 Uživatelská část

Tato část by měla fungovat jako jednoduché rozhraní, které nabídne uživateli seznam časových os, které může procházet a seznam všech přístupných testů, které může vyplnit. Součástí této části bude i jednoduchý formulář, pomocí kterého budou moci uživatelé vyjádřit svoje názoru a připomínky. Protože některé osy mohou být rozpracované a neúplné, budou mít administrátoři možnost vybrat osy, které bude moci uživatel procházet. Zároveň by nemělo být možné nad neaktivní časovou osou spustit test. Seznam časových os bude reprezentován tabulkou, která bude obsahovat vždy jméno časové osy a tlačíko pro její procházení. Seznam dostupných testů bude mít stejnou podobu, navíc však bude nabízet vygenerování URL odkazu pro jednoduché sdílení.

5.1.1 Widget

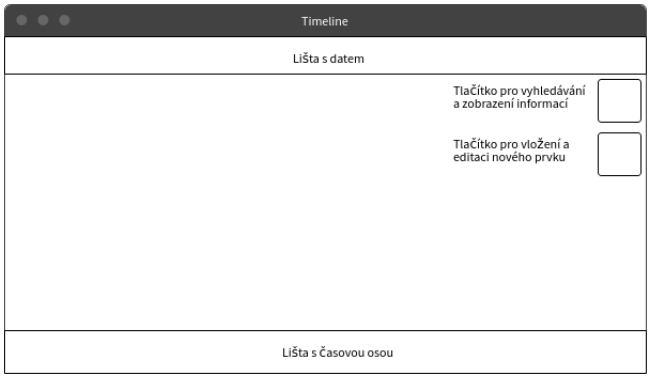
Ve svém základním provedení widget demonstruje veškerou funkcionalitu. Může proto na první pohled působit nepřehledně. Pro účel této práce je widget stěžejní a proto bude jeho prostředí odlehčit. Widget také musí reflektovat jednotlivé módy. Ty aplikace rozezná pomocí URL parametrů.



Obr. 4: Původní podoba widgetu

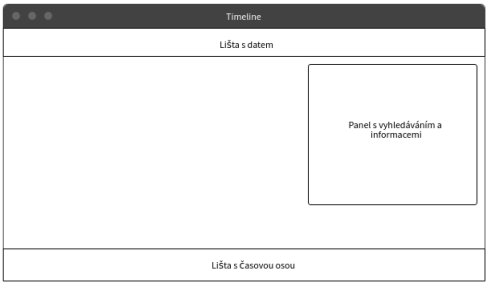
Aby byla vidět co největší část časové osy a minimalizovala se tak možnost překrytí dat bu nejlepší využít zvětšování a zmenšování jednotlivých prvků podle toho, zda je

uživatel využívá nebo ne. Z původní podoby widgetu tedy zmizí Interaktivní panel v pravé části. Protože se ukázalo, že nejlepší metoda manipulace s časovou osou je pomocí myši, odeberu také tlačítka pro přesouvání se po ose vpřed a vzad. Místo Interaktivního panelu přidám dvě tlačítka reprezentující vyhledávání a přidávání/editaci prvku časové osy.

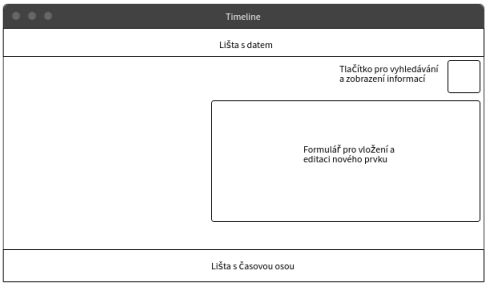


Obr. 5: Nová podoba widgetu

Tato tlačítka jsou interaktivní, a při kliknutí myší se zvětší a ve vzniklé oblasti se objeví příslušné ovládací prvky. Uživatel si tak může sám rozhodnout, kdy potřebuje spíše vidět na osu a kdy potřebuje něco vyhledat nebo přidat. Tlačítko pro přidávání a editaci je také přístupné pouze lidem přihlášeným.



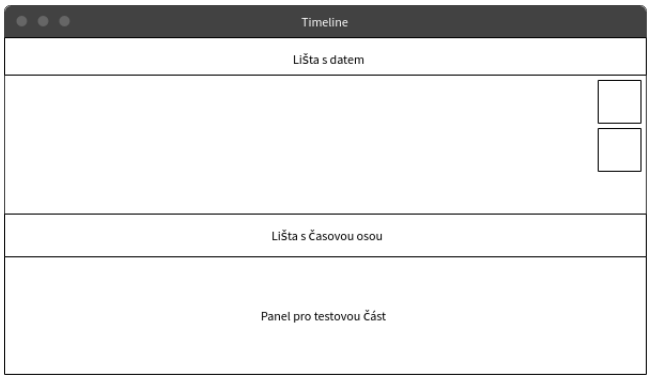
(a) Po kliknutí na vyhledávací prvek



(b) Po kliknutí na vkládací/editční prvek

Obr. 6: Widget s novými interaktivními prvky

Další funkcionalita widgetu spočívá v tvorbě a vyplňování testů. Uživatel musí mít přístup k panelu, který bude obsahovat formulář pro vytváření testových otázek nebo formulář pro odpovídání na tyto otázky a zároveň by měl mít přehled, aby mohl hledat údaje pro vytváření nebo vyplňování otázek. V případě tvorbz test bude také dobré poskztnout uživateli seznam všech dposoud vytvořených otázek, aby je mohl v případě potřeby upravit. Aby se minimalizoval dopad na přehled v rámci dat, bude nejlepší zúžit osu ve vertikálním směru. Díky tomu zůstane osa v hlavním horizontálním směru nezmenšená, což umožní vidět co největší část dat spolu se souvislostmi.



Obr. 7: Návrh pro tvobu a vyplňování testu

5.2 Administrativní část

Administrační část musí být oddělena a chráněna před možným přístupem uživatelů, kteří by neměli mít možnost manipulovat s daty v databázi. Za tímto účelem bude vstup do administrace povolen pouze přihlášeným uživatelům. Ti budou moci získat přístup jen pokud jim administrátor zašle pozvánku na email, která obsahuje registrační odkaz. Díky tomu bude zajištěn přehled o všech uživatelích, kteří mají možnost manipulovat s daty. Administrátor bude také moci využít nástroje uživatelských rolí, díky němuž může ostatním registrovaným uživatelům povolit pouze určité akce. V rámci administrativní části bude možné vytvářet vlastní časové osy. Uživatel tak naplní časovou osu daty podle potřeby a uloží do databáze. Samozřejmostí je možnost pozdějšího editování nebo smazání takto vytvořené časové osy. Vytváření testů zajistí další sekce, ve které bude uživatel vytvářet testy k existujícím časovým osám. Aby se zamezilo nehodám, test půjde smazat pouze autorovi nebo administrátorovi s vyšší rolí. Další část administrace bude sekce pro vyhodnocování testů. Jedná se o stěžejní část celé aplikace, kde je možno procházet statistiky testů, které podstoupili uživatelé.

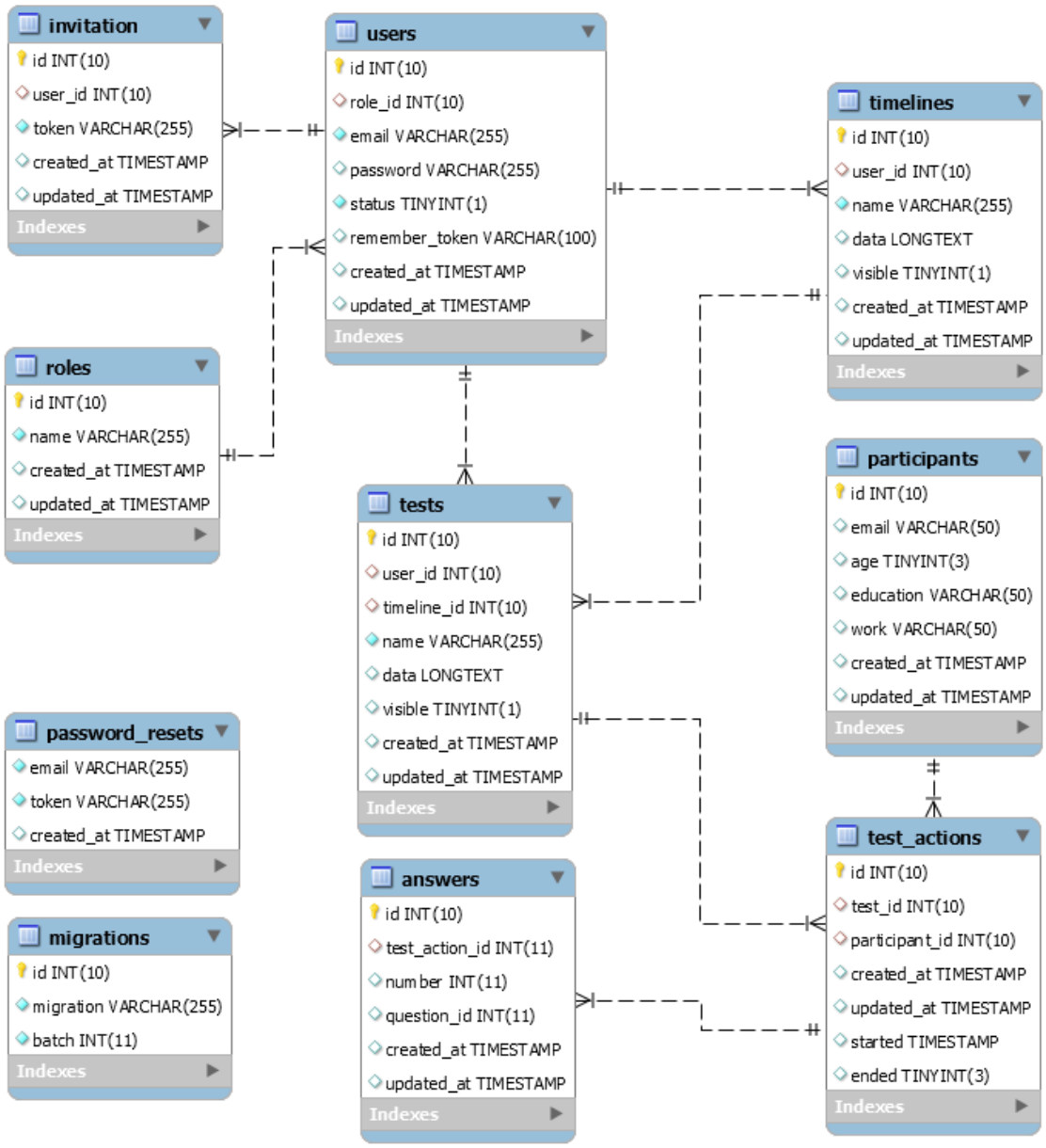
6 Implementace

Jádro aplikace tvoří PHP Framewok Laravel spolu s MySQL databází. Aby mohl widget poskytovat popsanou funkcionalitu, bylo nutné ho v základu rozšířit o třídu `OptionsPanel`. Obsluha testů a časové osy byla implementována pomocí scriptů v jazyce JavaScript. Jedná se o soubory `ui-script.js`, `options-bar.js` a také `test-panel.js`.

6.1 Databáze

Databáze musi být koncipována tak, aby udržela data v přehledné a konzistentní podobě. Databáze je strukturně jednoduchá a zároveň se nepředpokládají složité dotazy, proto bude pro aplikaci stačit databáze MySQL.

6.1.1 Databázový model



Obr. 8: ERA model databáze

6.1.2 Tabulka users

Tato tabulka obsahuje informace o všech uživatelích - aktivních i pozvaných. Díky bezpečnostním prvkům také zamezuje možnostem zneužití pro proniknutí do systému. Protože aplikace pracuje při přihlášení s emailem a nikoliv s uživatelským jménem,

je této skutečnosti využito, a vždy při zadání emailu, na který je odeslána pozvánka, je vytvořen nový uživatel, který je zároveň označen jako neaktivní.

- **id** - unikátní identifikátor uživatele
- **role_id** - identifikátor role uživatele z tabulky ***roles***
- **email** - email uživatele používaný k přihlášení
- **password** - zahashované heslo uživatele
- **status** - indikátor toho, zda je uživatel aktivován a odpověděl na pozvánku.
- **remember_token** - ochrana proti zneužití "remember me" cookie
- **created_at** - datum vytvoření uživatele
- **updated_at** - datum aktualizace uživatele

6.1.3 Tabulka tests

Všechny vytvořené testy jsou uchovávané v této tabulce spolu s odkazy na uživatele a časovou osu. Data jsou ve formátu JSON. Důvodem této volby navázání na práci pana Kacerovského, protože widget pro zobrazení časové osy pracuje s daty právě v tomto formátu. Stěžejním bodem práce jsou testy samotné a ukládání dat do databáze na úrovni uzlů a relací mezi nimi bylo předmětem jiné práce.

- **id** - unikátní identifikátor testu
- **user_id** - identifikátor autora testu
- **timeline_id** - identifikátor časové osy, k níž se test vztahuje
- **name** - jméno testu
- **data** - data testu ve formátu JSON
- **visible** - indikuje, zda je test viditelný při výběru k testování
- **created_at** - datum vytvoření testu
- **updated_at** - datum aktualizace testu

6.1.4 Tabulka timelines

Data časových os jsou ze stejného důvodu jako u dat testových ve formátu JSON. Pomocí spojení s tabulkou uživatelů se dá určit autor a díky tomu také zamezit ostatním uživatelům, kteromě výše postavených administrátorů, aby časovou osu neměnili nebo dokonce nesmazali.

- **id** - unikátní identifikátor časové osy
- **user_id** - identifikátor autora časové osy
- **name** - jméno časové osy
- **data** - data časové osy ve formátu JSON
- **visible** - indikuje, zda je osa viditelá k procházení
- **created_at** - datum vytvoření časové osy
- **updated_at** - datum aktualizace časové osy

6.1.5 Tabulka invitation

Všechny pozvánky se z bezpečnostních důvodů uchovávají, aby v případě využití pozvánky mohlo být ověřeno, že taková pozvánka byla opravdu vytvořena a uživatel tak může dokončit svo uregistraci vyplněním hesla. Všechny pozvánky jsou pouze jednorázové, což indikuje pole *used*.

- **id** - unikátní identifikátor pozvánky
- **user_id** - identifikátor pozvaného uživatele
- **token** - token pro kontrolu při uplatnění pozvánky uživatelem
- **used** - indikátor použité pozvánky
- **created_at** - datum vytvoření pozvánky
- **updated_at** - datum aktualizace pozvánky

6.1.6 Tabulka roles

Tato tabulka obsahuje seznam všech rolí. Odkazy na role mají uživatelé v tabulce *users*. Ke každé roli se váží v aplikaci určitá oprávnění, která zamezují nebo povolují uživatelům přistupovat do různých sekcí nebo provádět rozličné akce.

- **id** - unikátní identifikátor role
- **name** - jméno role
- **created_at** - datum vytvoření role
- **updated_at** - datum aktualizace role

6.1.7 Tabulka test_actions

Účelem této tabulky je uchovávat data jednotlivých testů, které uživatelé spustí a začnou vyplňovat. Tabulka také rozlišuje testy dokončené a nedokončené, čímž se zajistí konzistence statistik.

- **id** - unikátní identifikátor testové akce
- **test_id** - identifikátor testu, který uživatel vyplňuje
- **participant_id** - identifikátor osoby, která test vyplňuje
- **started** - čas začátku testu
- **ended** - indikátor, zda byl test kompletně vyplňen
- **creatd_at** - čas vytvoření položky v databázi
- **updated_at** - čas aktualizace položky v databázi

6.1.8 Tabulka participants

Tabulka uchovává data jednotlivých osob, které začaly vyplňovat nějaký test. Data slouží hlavně ke statistickým účelům, protože obsahují určité informace o osobě.

- **id** - unikátní identifikátor účasníka testu
- **email** - email účastníka testu
- **age** - věk účastníka testu
- **education** - vzdělání účastníka testu
- **work** - údaj o zaměstnání/studiu účastníka testu
- **created_at** - čas vytvoření položky účastníka testu v databázi
- **updated_at** - čas úpravy položky účastníka testu v databázi

6.1.9 Tabulka answers

Tato tabulka uchovává jednotlivé odpovědi účastníků testů. Díky odkazu na testovou akci je tak možné lehce získat odpovědi k jednotlivým testům a na jejich základě vytvářet požadované statistiky.

- **id** - unikátní identifikátor odpovědi
- **test_action_id** - identifikátor testové akce, k níž se odpověď vztahuje
- **number** - číslo odpovědi na danou otázku

- **question_id** - indentifikátor otázky, na kterou účastník testu odpovídání
- **created_at** - čas vytvoření položky odpovědi v databázi
- **updated_at** - čas úpravy položky odpovědi v databázi

Tabulky password_resets a migrations jsou automaticky vytvářené Framewokem Laravel kvůli vnitřní funkcionalitě aplikace a nemají z hlediska této bakalářské práce větší význam.

6.2 Klient

Klientská část aplikace představuje co možná nejpřehlednější reprezentaci všech nástrojů a možností přístupých uživatelům. Obsahuje seznam přístupných časových os, které mohou uživatelé procházet. Dále obsahuje seznam testů, které si může kdokoliv zkusit vyplnit. Hlavní částí klienta představuje widget pro časovou osu, který je doplněný o funkce přístupné obyčejným uživatelům. Mezi tyto funkce patří možnost procházení všech časových os včetně prohledávání dat, a možnost vyplnění testu na základě určité časové osy. Uživatelé mohou také napsat a odeslat zpětnou vazbu a reakce na webovou aplikaci.

V případě této práce se klientská část bere jako ta část aplikace, kam mají přístup nepřihlášení uživatelé. Všechny takto přístupné stránky jsou připravené pomocí šablonovacího systému Framewoku Laravel. Takto vygenerované stránky poskytují všechnu základní funkcionalitu. Obsluha widgetu, který je také vykreslen pomocí šablony, je však vytvořena pomocí JavaScriptu a JQuery knihovny. Zvolil jsem způsob vytváření globálních funkcí, které jsou díky tomu přístupné ve všech scriptech. Tyto funkce jsou navěšeny na příslušné objekty widgetu. na **Kód 1** vidíme ukázkovou definici globální funkce. klíčová je hlavička funkce, která díky $.fn. části definuje funkci jako globální. Jediná podmínka, která musí být splněna je, aby byly scripty k aplikaci připojeny ve správném pořadí, protože jinak by funkce nebyly dostupné ve scriptech připojených k aplikaci dříve, než script obsahující tyto funkce. Základní význam funkcí je popsán v komentářích zdrojového kódu. Zde popíši princip, na jakém fungují složitější funkce.

```
$.fn.function_name = function() {
    // Body of the function
};
```

Kód 1: Globální funkce

6.2.1 Třída Timeline

Tato třída je dílem pana Kacerovského. Jejím účelem je obsluhovat veškerou funkcionalitu spojenou s časovou osou. Spojuje dohromady komponenty, které vykreslují

časovou osu jako jedne celek. Tuto třídu jsem využil jako spojovací článek mezi stávající a novou funkcionalitou. Napojil jsem na ní novou třídu `OptionsPanel`. Díky tomu, že této nové třídě je v konstruktoru předána instance třídy `Timeline`, mohu nyní vcelku jednoduše manipulovat se zdrojem dat pro vykreslení na časové ose.

6.2.2 Třída OptionsPanel

Třída je psána ve stejném duchu jako původní třídy widgetu. Jejím úkolem je vykreslit prvek pro vyhledávání v datech časové osy a prvek pro vkládání nových dat do časové osy. Obsahuje formulář, který umožní přidat jeden nový uzel s možností napojení na ostatní uzly různými typy relací. Data z tohoto formuláře jsou následně pomocí AJAXu odeslána na server a zároveň s tím vykreslena do časové osy.

Hlavním prvkem třídy je funkce `_insertData()`. Ta získá z formuláře potřebné údaje, následně podle nich vytvoří novou instanci třídy `Entity`, která představuje uzel na časové ose. Dále pak projede všechny relace vytvořené pro teto uzel a vytvoří pro ně instance třídy `Relation`. Data jsou následně odeslána na server a přidána do zdroje dat časové osy, která je nakonec překreslena, aby reflektovala i nově vložená data.

```
var node = this._processLeftPart(left);
var edge = this._processRightPart(right, node);
```

Kód 2: Získání dat z formuláře

```
var dataSource = this._timeline.getDataSource();
```

Kód 3: Získání zdroje dat pro časovou osu

```
var entity = new Entity(node);
dataSource._entities._data.push(entity);
this._data.nodes.push(node);

if(edge !== null && edge !== undefined) {
    var relation = new Relation(edge);
    dataSource._relations._data.push(relation);
    dataSource = this._updateRelations(edge, dataSource);
    this._data.edges.push(edge);
}
```

Kód 4: Tvorba instancí Entity a Relations

```
this._sendData();
this._timeline.setDataSource(dataSource);
this._timeline.redraw();
```

Kód 5: Odeslání, uložení a vykrelsení dat

Kód 2 získává zvlášť uzly a zvlášť hrany, které jsou dále zpracovány, jak bylo výše uvedeno, do Entit a Relací pomocí **Kód 4**. Tyto nově vytvořené instance jsou následně vloženy do zdroje dat pro časovou osu funkcí `push()` volanou nad polem Entit a Relací v **Kód 4** vloženy do zdroje dat pro časovou osu získaném pomocí **Kód 3**. Nakonec se v **Kód 5** data odešlou na server funkci `_sendData()`, dále se nastaví nový zdroj dat pro časovou osu a samotná osa se překreslí. Odesílání na server probíhá pomocí standardního AJAX volání, kdy se nejprve definuje typ požadavku, data, která posíláme spolu s typem těchto dat, který je v tomto případě JSON.

```
$.ajaxPrefilter(function(options, originalOptions, xhr) {
    var token = $('meta[name="csrf-token"]').attr('content');

    if (token) {
        return xhr.setRequestHeader('X-CSRF-TOKEN', token);
    }
});
```

Kód 6: Připojení CSRF tokenu

```
var nodes = JSON.stringify(this._data.nodes);
var edges = JSON.stringify(this._data.edges);
```

Kód 7: Upravení dat pro odeslání

```
var jqxhr = $.ajax({
    url : 'http://127.0.0.1:8000/timelines/' + id,
    data : JSON.stringify({'nodes': nodes, 'edges': edges}),
    type : 'PATCH',
    contentType : 'application/json',
    processData: false,
    dataType: 'json'
});
```

Kód 8: Odeslání dat

```
jqxhr.done(function(data, textStatus, jqXHR) {
    console.log('Succeed:', textStatus);
});

jqxhr.fail(function(jqXHR, textStatus, errorThrown) {
    console.log('Failed:', errorThrown);
});
```

Kód 9: Definice funkcí pro sledování stavu požadavku

V **Kód 6** se pro AJAX připraví CSRF token, který slouží k ochraně před CSRF útokem. Tento funguje na bázi odposlechu komunikace mezi uživatelem, který je přihlášený a serverem. Útočník se tak může vydávat za uživatele a provádět autorizované operace, za předpokladu, že dokáže odhadnout url adresy a parametry pro administrativní část apliakce. Právě CSRF token coby náhodně generovaný token, který je odesílán při každé komunikaci a generován při každém novém odesílání požadavků na server znovu, zabraňje CSRF útoku, protože útočník nedokáže nikdy vygenerovat token stejný jako aplikace. Pokud tak odešle starý token, aplikace tento request ignoruje.

V **Kód 7** se data upravují tím, že JSON objekty jsou převedeny na řetězce, tyto řetězce pak mohou být jednoduše odeslány. V **Kód 8** se připravuje samotný AJAX požadavek na server. Nejprve se určí adresa, na kterou bude požadavek odeslán. Tato adresa se musí změnit podle toho, kde běží servereová část. Dále se definují data, která budou odeslána, typ požadavku a typ dat. Na `jqxhr` objekt se může navěsit nespočet stavových funkcí. Jejihc plný seznam lze nlézt na stránkách JQuery knihovny. Tyto funkce pomáhájí sledovat v jakém stavu se požadavek nachází. Například poznáme, kdy se podesílá, kdy byl odeslán a zda byl odeslán v pořádku. Pro účely apliakce postačí sledovat, zda požadavek proběhl či neproběhl úspěšně, jak je vidět v **Kód 8**.

6.2.3 Soubor test-panel.js

Obsahuje veškerou funkcionalitu pro vykreslení panelu pro tvorbu testů a také pro testy samotné. Nacházejí se v něm funkce, které uzpůsobí aplikaci tak, aby časová osa nebyla překryta testovým panelem. Dále obsahuje ovládací prvky pro tvorbu testových otázek, jako je přidávání různého počtu odpovědí, označení správné odpovědi a také validace formuláře. Při každé změně testových dat jsou data odeslána na server, aby byla práce v každém okamžiku zálohována. Komunikace se serverem probíha pomocí AJAXu.

Testový panel je součástí šablony pro vykreslení widgetu. Jeho viditelnost je vždy odvozena od módu, ve kterém je widget spušťěn. Tento mód je vždy přítomen v URL parametru **mode**. V základním zobrazení obsahuje panel dvě části. V levé části je samotný formulář, který obsahuje políčko pro otázku, jedno políčko pro odpověď spolu s zaškrtávacím políčkem pro označení správné odpovědi, tlačítko pro přidání odpovědi, tlačítko pro uložení otázky a tlačítko pro ukončení tvorby testu.

Funkce `addQuestion()` nejprve provede validaci formuláře a zkontroluje, zda jsou všechna pole vyplněná. Pokud nejsou funkce vrátí seznam původních otázek. Následně se proiterují vstupy pro odpovědi a uloží se do pole JSON objeků(**Kód 9**), ve kterém

je u každé odpovědi uložena informace, zda je tato odpvoěď správná nebo špatná. Pomocí získaných dat se vytvoří objekt pro otázku(**Kód 10**), ve kterém je text otázky z políčka pro otázku, dále pole odpovědí a ID otázky.

```
var newQuestion = {
    'text': answersString,
    'right': false/true
}
```

Kód 10: Objekt odpovědi

```
var newQuestion = {
    'id': id_otazky
    'question': questionColumn.val(),
    'answers': answers
}
```

Kód 11: Objekt otázky

Zároveň s těmito úkony je každá vložená otázka zobrazena v pravé části testovacího panelu. Pokud se na otázku klikne, vyplní se v levé části formulář údajemi z této otázky a otázka se tak může editovat. Funkce `editQuestion()` funguje na stejném principu, jako funkce `addQuestion()`, navíc se pouze získají data z pole otázek, podle kterých se vyplní formulář. Při každém přidání otázky se seznam v pravé části panelu vymaže a znovu naplní již připravenými otázkami pomocí funkce `repopulateAnswerList()`. Ta se volá i v případě vymazání některé otázky. Tato otázka se vyjme z pole otázek a zároveň se provede aktualizace seznamu v pravé části. Vždy při přidání nebo editaci nebo vymazání otázky se nová data posílají vcelku na server, který je ukládá do databáze.

6.2.4 Soubor options-bar.js

Tento soubor obsahuje funkce pro zvětšování a zmenšování prvků pro vyhledávání dat a pro přidávání dat do časové osy. Zajišťuje, aby v každou chvíli byl zvětšen pouze jeden z těcho prvků pro maximalizaci přehlednosti. Další jeho funkcí je přidávání a odebírání polí pro nové relace do formuláře pro vkládání dat do časové osy.

Funkce `resizeButton()` obsahuje základní rozměry příslušných tlačítek. Podle id tlačítka rozezná, na které bylo kliknuto a zavolá funkci `minimize()` resp. `maximize()`, která prvek zvětší nebo zmenší. Zároveň s těmito funkcemi je vždy zavolána funkce `minizeOthers()`, která která zmenší všechny ostatní prvky, na které nebylo kliknuto, které jsou maximalizované. Díky tomu je vždy maximalizován pouze jeden prvek. Další funkcionalitou tohoto scriptu je přidávání políček pro nové relace při přidávání nového prvku do časové osy. Zpracování dat formuláře probíhá ve třídě `OptionsPanel`, proto se zde pouze přidává relace(možná bude přesunuta do OptionsBar).

6.2.5 Soubor test.js

Má na starost vše kolem vyplňování testu. Protože stejně jako u tvorby testu i panel s vyplňváním testových otázek je v šabolně na pevno, odvíjí se jeho viditelnost podle módu. Při začátku testu nejprve získá data z formuláře, které uživatel vyplnil. při odkliknutí toho formuláře se začíná počítat čas startu testu. Jako odpověď na odeslaná data přijde obsah testu. Aby nemohl uživatel podvádět, není v odpověi serveru žádná informace o správnosti odpovědí, tato kontrola se děje až na konci testu na straně serveru. Tuto prvotní funkcionalitu obstarává funkce `initStartTestButton`, která se spustí ve chvíli kliknutí na tlačitko odesílající informace z formuláře. Další funkce, `initNextquestion()` zajišťuje načtení další otázky, zároveň odešle odpověď na předešlou otázku na server. Součástí je také validace, zda uživatel vybral nějaku možnost. všechny požadavky a odpovědi jsou realizovány pomocí AJAXu.

6.2.6 Soubor ui-script.js

Tento script se dá považovat za hlavní zastřešující prvek nad ostatními soubory. Podle URL získá hodnotu módu zobrazení widgetu časové osy(**Kód 11**) a následně přes konstukci `if-else` upraví viditelnost jednotlivých prvků widgetu. Dálen avěšuje funkce připravené ostatními scripty bna příslušné prvky widgetu. Protože jsou prvky dynamicky generované, musíme použít funkce `on()`. Tato funkce z knihovny JQuery dokáže proti funkci `click()` dokáže lokalizovati dynamicky vytvořené prvky a navěsit na ně funkce. V ukázce(**Kód 12**) vidíme, že pomocí identifikátoru navěšujeme funkci na celý dokument. Nicméně při akci kliknutí se najde dynamicky generovaný prvek a akce se provede v kontextu tohoto prvku.

```
jQuery(document).on('click', 'identifikator', function() {
  jQuery(this).funkce(); //Nyni je this tlacitko
});
```

Kód 12: Ukázka použití funkce on()

6.3 Server

Serverová část je realizována pomocí PHP Framewoku Laravel. Tento Framewok umožňuje tvorbu rychlých a stabilních aplikací a momentálně se řadí mezi nejlepší Framewoky, což bylo hlavním duvodem k jeho výběru. Tento framework také dodržuje architekturu **M-V-C** - Model-View-Controller, což v praxi znamená, že aplikace je rozdělena do tří vrstev. Pvní vrstva **Model** má na starosti mapování databáze a práci s daty v ní. Další vrstva **View** je tvořena systémem šablon, které vykreslují aplikaci spolu s daty z další vrstvy **Controller**. Tato poslední vrstva má za úkol komunikaci s modelem a odesílání dat získaných v něm do šablon.

Mezi další výhody patří velmi silný nástroj pro příkazovou řádku, který umožňuje

například procházení databáze a testování modelů aplikace, generování databázových tabulek spolu s vazbami a případně daty, generování tříd modelů a controllerů včetně předpřipravených základních funkcí. Veškeré informace o možnostech frameworku jsou dostupné na internetu. Důležitou součástí Framewoku je systém šablon, díky kterému se dá velmi jednoduše integrovat dynamický kód do statického HTML kontextu. Použití HTML a CSS pro stylování je u webových aplikací samozřejmostí.

Tento framework představuje celou webovou aplikaci. Obstarává přihlášení a registrování uživatelů, dále ukládání dat do databáze a jejich poskytování widgetu pomocí jednoduchého API, zobrazení widgetu a možnost odeslání zpětné vazby. Dokumentace k tomuto Framewoku je k dispozici na internetu a proto shrnu jen důležité části aplikace.

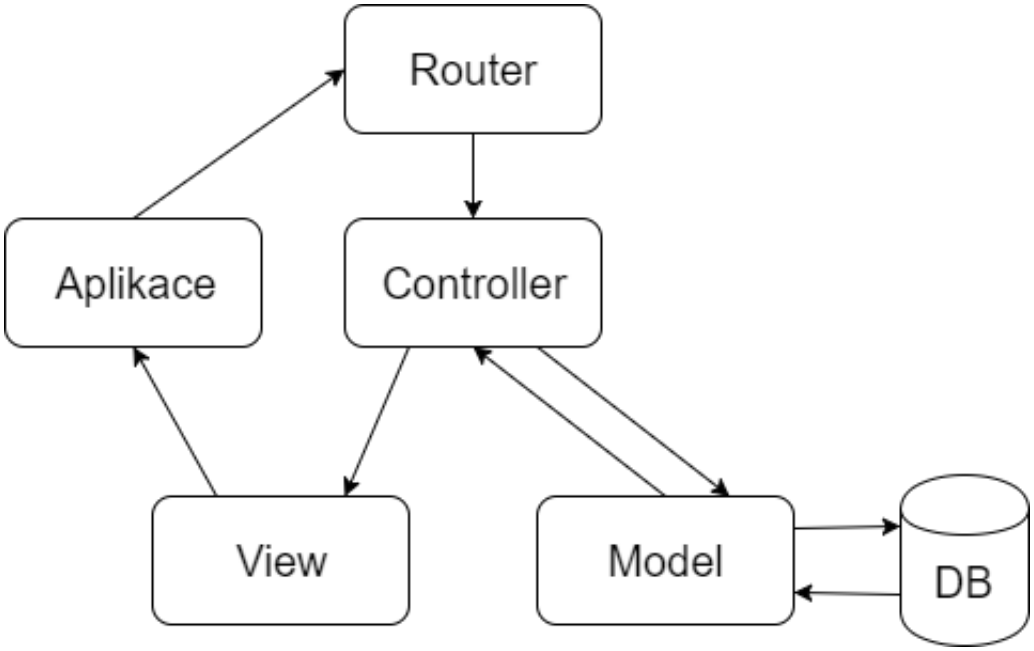
6.3.1 Implementace

Vstupní bod tvoří soubor **/routes/web.php**, který představuje router. Podle příchozího požadavku rozhoduje, která funkce z jakého controlleru se zavolá. Zde jsou také definovány routy, které jsou přístupné pouze přihlášeným uživatelům. V případě neautorizovaného přístupu potom přesměruje uživatele na domovskou stránku.

Další sada důležitých souborů se nachází ve složce **/database/migrations/**. Zde jsou definovány jednotlivé tabulky spolu s vazbami. Tyto migrace jsou vytvářeny automaticky pomocí příkazové řádky, po jejich vygenerování stačí nadefinovat pouze jednotlivá pole tabulky spolu vazbami na další tabulky a poté pomocí příkazu spustit migraci, jejíž součástí je i smazání starých tabulek. S migracemi úzce souvisí i Seedování tabulek. v **/database/seeds/** jsou obsažené skripty pro naplnění tabulek případnými výchozími hodnotami.

Na databázi navazují modelové třídy pro jedntlivé entity. Tyto třídy jsou uložené v **/app/** složce. Tvoří vrstvu mezi databází a controllery tím, že mohou definovat vazby mezi entitami. Dále mohou obsahovat mapování na data v databázi. To znamená že data jsou reprezentována třídami pro další práci s nimi vrámci frameworku. Díky tomu lze jednoduše získat data napříč tabulkami bez nutnosti ručního psaní SQL dotazů. Měla by vnich tedy být obsažena veškerá logika kolem dat a manipulací s nimi.

Nejdůležitější část Framewoku tvoří controllery. Tyto třídy se nacházejí v **/app/http/Controllers/**. Každý controller obstarává jednu logickou část aplikace, jako jsou uživatelé, tvorba testů, časové osy, zpětná vazba a průběh testování. Jednotlivé funkce v rámci controlleru by měly odpovídat požadavkům rout v routeru.



Obr. 9: Životní cyklus aplikace [1]

Podle Obr.5 životní cyklus aplikace začíná požadavkem vyvolaným aplikací, který je následně zpracován routerem. Podle požadavku router rozhodne, jaká funkce v Controlleru se spustí. V rámci funkce proběhne komunikace s modelovou vrstvou, která získá data z databáze. Ta jsou předána zpět funkci v controlleru a v případě potřeby upravena do požadovaného tvaru. Controller poskytne data šablonám, které vykreslí výsledek požadavku.

Použitá literatura

[1] Laravel architecture.
URL `http://laravelbook.com/laravel-architecture/`

[2] Laravel command line tool. 2017.
URL `https://laravel.com/docs/5.4/artisan#introduction`

[3] Laravel trending. 2017.
URL `https://trends.google.com/trends/explore?q=%2Fm%2F0jwy148,%2Fm%2F09cjcl,%2Fm%2F04n23m7,%2Fm%2F0cdvjh`

[4] Tiobe index. mar 2017.
URL `https://www.tiobe.com/tiobe-index/`

[5] Hessová, G.: *Automatické získání historických údajů z webových zdrojů.* 2015.

[6] Hrbáček, D.: *Zpracovnání časových údajů pro jejich vizualizaci.* 2015.

[7] Kacerovský, M.: *Vizuální reprezentace precedenčního grafu.* 2015.

[8] Merunko, D.: *Generování a vizualizace časové osy.* 2014.

Slovník pojmů

AJAX Asynchronous JavaScript and XML je technologie, která umožňuje asynchronně překreslovat aplkaci bez obnovení stránky a také získávat a odesílat data na server. .

API Application Programming Interface - aplikační rozhraní, které definuje komunikaci s ní..

CSRF Cross Site Request Forgery, zneužití komunikace mezi užovatelem a serverem ve prospěch útočníka za účelem změny stavu dat..

CSS Cascading Style Sheets, jazyk popisující jakým způsobem se prvky webové stránky zobrazí..

Framewok Jedná se o strukturu pomocných knihoven,nástrů a návrhových vzorů..

HTML HyperText Markup Language, značkovací jazyk pro tvorbu webových stránek..

JSON Cross Site Request Forgery, zneužití komunikace mezi užovatelem a serverem ve prospěch útočníka za účelem změny stavu dat..

M-V-C Architektura pro tvorbu moderních dynamickýchwebových aplikací..

MySQL Structured Query Language, jazky využívaný pro vytváření dotazů do databáze..

PHP Hypertext Preprocessor, původně Personal Home Page - scriptovací jazyk využívaný pro tvorbu webových aplikací..

Seed Označuje uložení počátečních dat do databáze po jejím vytvoření..

SQL Structured Query Language, jazky využívaný pro vytváření dotazů do databáze..

URL Architektura pro tvorbu moderních dynamickýchwebových aplikací..