ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Růžička** Jméno: **Tomáš** Osobní číslo: **484887**

Fakulta/ústav: **Fakulta informačních technologií**

Zadávající katedra/ústav:

Studijní program: **Informatika**

Studijní obor: **Webové a softwarové inženýrství**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Asterion - backend vizualizace časových os

Název bakalářské práce anglicky:

Asterion - timelines visualization backend

Pokyny pro vypracování:

Asterion je rozsáhlý svět s bohatou a komplikovanou historií. Cílem práce je vizualizovat historická data tohoto fiktivního světa pomocí vícera časových os vázaných na konkrétní téma, lokace, apod. Výsledný produkt bude běžet jako webová služba. Práce je součástí skupinového projektu Asterion.
1) Proveďte rešerši způsobů vizualizace historických dat.
2) Analyzujte a vhodně uspořádejte množinu časových dat světa Asterion poskytnutých zadavatelem.
3) Proveďte rešerši technologického řešení webové vizualizace.
3) Navrhněte prototyp - zaměřte se na backend a vytvoření vhodné databáze. Bezpečnost přenosu administrátorského přístupu. Možnosti privilegovaných uživatelů zadávat historická data.
4) Implementujte a nasaďte službu.
5) Důkladně otestujte bezpečnost a funkčnost služby.

Seznam doporučené literatury:

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Radek Richtr, Ph.D., katedra softwarového inženýrství FIT

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **16.02.2021** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: _____

Ing. Radek Richtr, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	doc. RNDr. Ing. Marcel Jiřina, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Bakalářská práce

ASTERION – BACKEND VIZUALIZACE ČASOVÝCH OS

Tomáš Růžička

Fakulta informačních technologií ČVUT v Praze
Katedra softwarového inženýrství
Vedoucí: Ing. Radek Richtr, Ph.D.
26. června 2021

Odkaz na tuto práci: Tomáš Růžička. *Asterion – backend vizualizace časových os.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především Ing. Radku Richtrovi, Ph.D. za nehynoucí trpělivost s námi a za nadšení do projektu. Dále bych chtěl poděkovat Michaele Zimmermannové za spolupráci na projektu a všem, kteří si tuto práci budou číst, za jejich čas a pozornost.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 26. června 2021 .....

Abstrakt

Asterion je fiktivní svět psaný hráči pen&paper her na hrdiny. Spisovatelé se musejí orientovat ve sledu událostí, pokud se chtějí vyhnout nesrovnalostem. Projekt spojený s touto prací se snaží vytvořit aplikaci pro tvorbu a vizualizaci událostí na časových osách. Cílem této práce je vytvořit API pro tuto aplikaci. API bylo implementováno pomocí Node.js, MySQL byla použita jako databáze a nginx byl použit jako webový server a pro šifrování pomocí ssl. Tento text obsahuje rešerši konkurenčních řešení, popis základních technologií a bezpečnostních principů a návrh aplikace. Aplikace byla otestována a shledána funkční a bezpečnou.

Klíčová slova asterion, časové osy, historické události, webová vizualizace, webová aplikace, backend, API, HTTP, cookies, autorizace uživatelů, databáze

Abstract

Asterion is a fictional world written by players of a pen&paper role playing games. Writers have to navigate the web of events if they want to avoid inconsistencies. Project associated with this work is supposed to make an application for creation and visualization of these events and timelines. Goal of this work is to create a web API for such application. API has been implemented with Node.js, MySQL has been used as a database and nginx has been used as a web server taking care of directing and ssl certification. This document contains research of competing applications, description of basic technologies and security principles and a application design. Application has been tested and found to be functional and safe.

Keywords asterion, timelines, historical events, web visualization, web application, backend, API, HTTP, cookies, user authorization, database

Úvod a shrnutí

Co je Asterion

Tento projekt je určen pro hráče pen&paper hry na hrdiny zasazené ve světě Asterion a spisovatele, kteří tento svět pomáhají tvořit.

Hra je v podstatě diskuze hráčů o tom, co by se mělo ve světě odehrávat. Přičemž se dějové linky drží postav, které si vybrali, a nějakého základního příběhu, který jeden z hráčů nastínil.

Motivace

Jelikož Asterion je svět s velmi dlouhou historií a velkým množstvím hráčů, je složité vymýšlet příběh, ve kterém není rozpor s již zaznamenanými fakty a událostmi v tomto světě.

Hráči nemusí dokonce ani hrát právě v aktuální době. Mohou se rozhodnout, že budou hrát jako skupina pytláků někde uprostřed doposud zaznamenané historie světa. Poté je vhodné do příběhu zakomponovat již známé události (války, svatby, narození).

Tyto události jsou ale rozprostřeny v knihách, které spisovatelé napsali, a málo komu se chce číst tři tlusté knihy, aby zjistil, kdy se vlastně mlynář Pešek z Kominíkova stal mlynářem.

Cíl projektu a práce

Cílem projektu je vytvořit webovou aplikaci, která bude na časových osách jasně a přehledně vizualizovat události, které se odehráli v herním světě Asterion.

Tato aplikace by měla uživateli pomoci zmapovat sled událostí týkajících se nějakého tématu. Například pohyb několika postav najednou a zjistit tak, zda je možné, aby se náhodou potkaly v krčmě U Fredyho v pátek 13.

Cílem této práce je pak vytvořit technické zázemí pro tento projekt a databázi událostí a webové API, čili backend.

Postup

Nejprve byla provedena rešerše konkurentů a technologií. Následně proběhl návrh domény, databáze a základního API pro provoz vizualizace. To bylo poté také implementováno. Pak proběhla příprava testovacích nástrojů a testování. Dále byl proveden návrh a implementace části API odpovědná za autentizaci a autorizaci uživatelů a její důkladné otestování. Poté byla na řadě část API schopná přidávat, editovat a mazat data, tato část byla opět navržena, implementována a otestována na separátní instanci API a databáze. Nakonec proběhlo důkladné plošné otestování a finální nasazení.

Výsledky práce

Výsledkem práce je nasazená a spuštěná aplikace pro vyhledávání a editaci historických událostí ve formátu navrženém pro snadnou implementaci vizualizace dat na časových osách.

Závěr

Byla provedena rešerše konkurence a byly nalezeny velké nedostatky v jednotlivých podobných aplikacích. API byla implementována, zpřístupněna, otestována a shledána funkční a bezpečnou.

Seznam zkratek a pojmů

Zkratky

HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
WASM	Web Assembly
HTTP	Hyper Text Transfer Protocol
HTTPS	HTTP Secure
Web API	Web Application Programming Interface
XSS	Cross-site scripting
CSRF	Cross-site request forgery

Pojmy

HTML	Jazyk určující strukturu webové stránky
CSS	Jazyk určující vzhled webové stránky
JS	Jazyk určující chování webové stránky
WASM	Strojově čitelnější, a proto rychlejší, jazyk určující chování webové stránky
HTTP	Protokol popisující komunikaci mezi počítači
HTTPS	Šifrovaná nadstavba nad HTTP
Web API	Synonymum pro webovou službu
Autentizace	Ověření, že uživatel je tím za koho se vydává
Autorizace	Ověření, že uživatel má pravomoc k využívání nějaké služby
Salt	Text zvyšující náhodnost hesla
Cookie	Data, která si stránka nebo API uložila do prohlížeče
XSS	Útok vkládající utočníkův kód na důvěryhodnou webovou stránku
CSRF	Útok z nedůvěryhodné stránky využívající stavu přihlášen na důvěryhodné stránce

Část I

Teoretická část

Kapitola 1
Události a časové osy

V této práci se zabývám aplikací, která zobrazuje časové osy, proto vysvětlím, co taková časová osa vlastně je a co jsou události, které se na ní budou zobrazovat.

Časová osa je ve své podstatě linka, ve které každý bod reprezentuje nějaký okamžik, tedy nějaký den v roce v nějaký konkrétní čas. Na takové ose lze zaznamenat **události**, tedy nějaký popis dění v čase.

1.1 Druhy časových os

Časové osy lze dělit do kategorií a to například podle tvaru os (rovné, kruhové, spirálové, …), způsobu odsazení událostí (chronologicky, chronologicky relativně k nějaké události, s libovolným odsazením, logaritmicky, …), nebo počtu a posazení os (jedna osa, více os, jedna osa rozprostřená na více řádků, …). [1]

- *Kruhové* jsou v zásadě vhodné pro zaznamenání opakujících se událostí. Pěkným příkladem je Mayský kalendář.
- *Spirálové* se hodí pro esteticky příjemné zobrazení na malém prostoru. Obě varianty se spíše hodí pro ručně vytvořené sady událostí pro sdělení nějaké konkrétní informace. [1]
- Osy uspořádané *chronologicky relativně k nějaké události* lze použít pro porovnání dvou průběhů nějaké déle trvající události. Například pro srovnání dvou sportovních výkonů které se odehrály v různý čas. Tedy na dvou osách, které jsou zobrazeny nad sebou (stejně tak události „start“), je snadno vidět, v jakých úsecích se náskok získal, nebo ztratil. [1]
- Osy s událostmi *libovolně odsazenými* od sebe zobrazují pouze pořadí událostí a zanedbávají dobu, která mezi nimi uběhla. Naopak tím získá přehlednost v případech, kdy je na nějakých místech osy mnoho událostí blízko u sebe a na jiných prázdno. [1]
- *Rovné* osy uspořádané *chronologicky* jsou pravděpodobně právě ty, které si člověk představí pod pojmem časová osa. Vzdálenost mezi událostmi odpovídá množství času mezi událostmi a čas na zobrazovacím médiu plyne jen jedním směrem. [1]

1.2 Druhy časových událostí

Nejzákladnější typ události, označme ho jako *bodové*, reprezentuje dění v jeden konkrétní okamžik [2]. Nezajímá nás jak dlouho událost trvá, pouze kdy se udála. Lze ji tedy na ose reprezentovat jako bod.

Další základní typ, označme ho jako *trvající*, je charakteristický dobou trvání [2]. Například nějaká válka, nebo něčí vláda. Je důležité vědět, kdy událost začíná a kdy končí, protože pro bodové události může být relevantní, zda se udály souběžně s událostmi trvajícími. Například pokud byl člověk A zabit strážníkem B v době, kdy bylo vyhlášeno stanné právo, či nikoli. Takové události je vhodné zobrazovat jako úsečky. Dále máme trvající události, které pro potřeby časové osy začaly na počátku věků, nebo naopak stále trvají. Tyto události lze zobrazovat jako polopřímky.

Někdy může nastat problém, kdy nemáme úplné informace o události. Nemusí být znám přesný datum bodové události, ale pouze měsíc, či rok. Nebo není známo, kdy začala, nebo skončila trvající událost. Tyto události označme jako *nepřesné* a je vhodné je zobrazovat jako úsečky (či polopřímky), jejichž konce mohou být průhledné či jinak zbarvené, aby se indikovala tolerance nepřesnosti.

Kapitola 2

Rešerše podobných aplikací

Tato kapitola se zabývá souhrnem vlastností podobných aplikací nalezených během zpracování této práce.

2.1 TimelineSetter

TimelineSetter je nástroj, který z textového souboru, obsahujícího čas a popis události, vygeneruje zdrojový kód webové stránky, obsahující daná data reprezentovaná na časové ose [3].

Tato časová osa podporuje pouze bodové události, které zobrazuje jako úzké svislé pruhy uvnitř obdélníkové reprezentace osy. Po přesunutí kurzoru myši na pruh se zobrazí detaily o události. Jednotlivé události nejsou rozlišitelné bez zobrazení tohoto detailu.

2.2 Tiki-Toki Timeline Maker

Tiki-Toki Timeline Maker [4] má několik zobrazovacích módů. V této sekci se budu zabývat Category Band módem. Ten, jak je vidět zde 2.2, zobrazuje jednu osu v dolní části obrazovky, na které je posuvné okénko reprezentující rozsah zobrazených událostí. Události mají prostor ve zbytku obrazovky.

Každá událost je zobrazena jako rámeček s popiskem, který má na své spodní straně drobnou šipku ukazující na příslušný čas. Zdá se ale, že má problém s větším množstvím událostí blízko u sebe. Prostor pro události je rozdělen mezi jednotlivé kategorie událostí. Na časové ose v dolní části obrazovky se zobrazují barevné body reprezentující jednotlivé události pro jednoduchou navigaci na ose.

2.3 Timetoast timeline maker

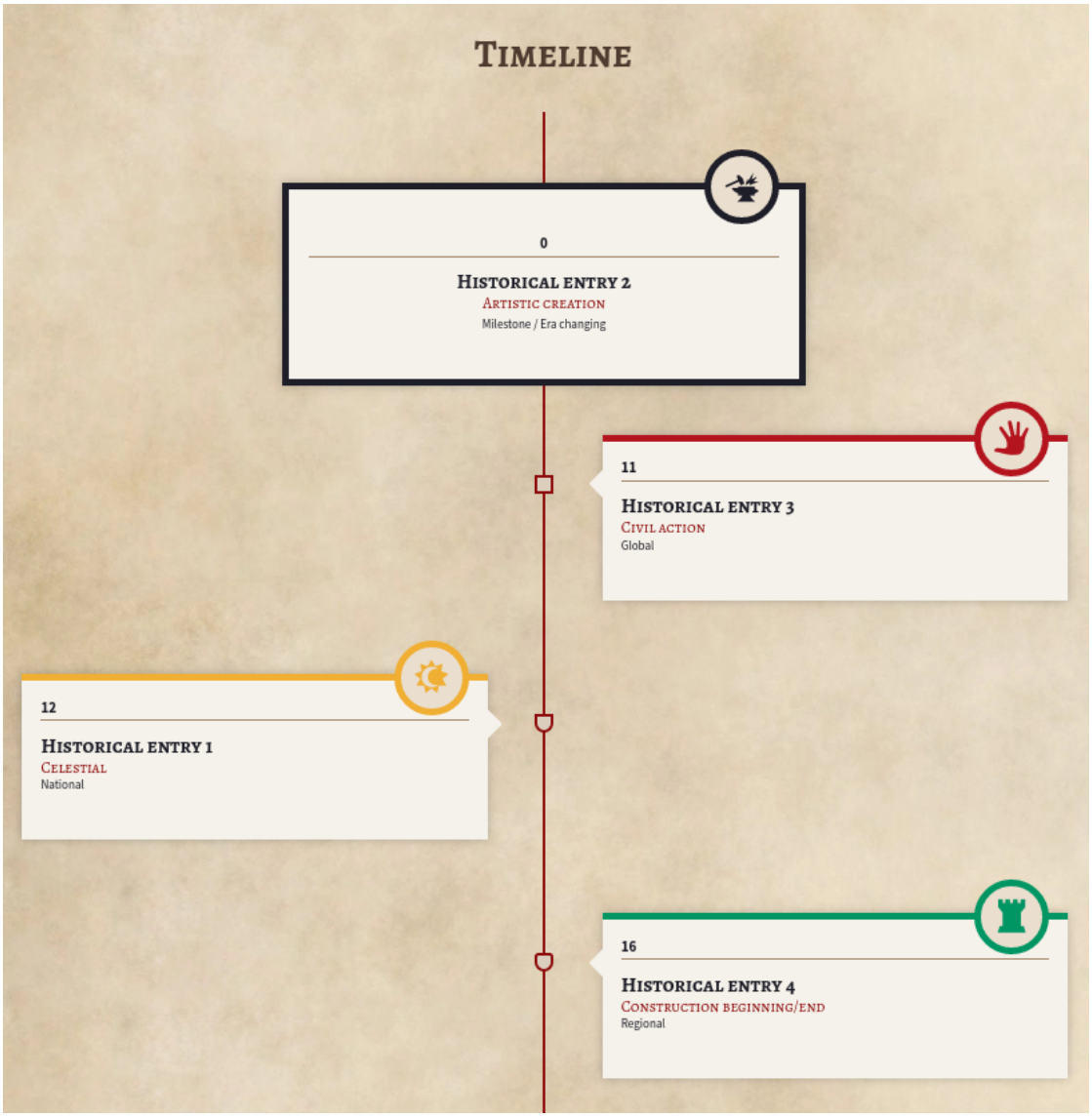
Timetoast timeline maker [5] stejně jako Tiki-Toki Timeline Maker [4] zobrazuje jednu osu v dolní části obrazovky. Ve zbylém prostoru se pak zobrazují jednotlivé události s rovnou čárou ukazující přímo dolů na konkrétní čas na ose. Viz 2.3.

Na rozdíl od Tiki-Toki Timeline Maker [4] se události rozmisťují nejdříve těsně nad osou a až při nedostatku místa se začnou přesunovat do řádků výše. Samotná osa si určí datum, před kterým, a datum, za kterým není zaznamenaná žádná událost. Mezi těmito daty pak leží posuvník, který má nastavitelný začátek i konec. Nad posuvníkem je zobrazený popisek dat časové osy, podle kterého se události zobrazují, jehož začátek a konec se řídí právě již zmíněným posuvníkem.

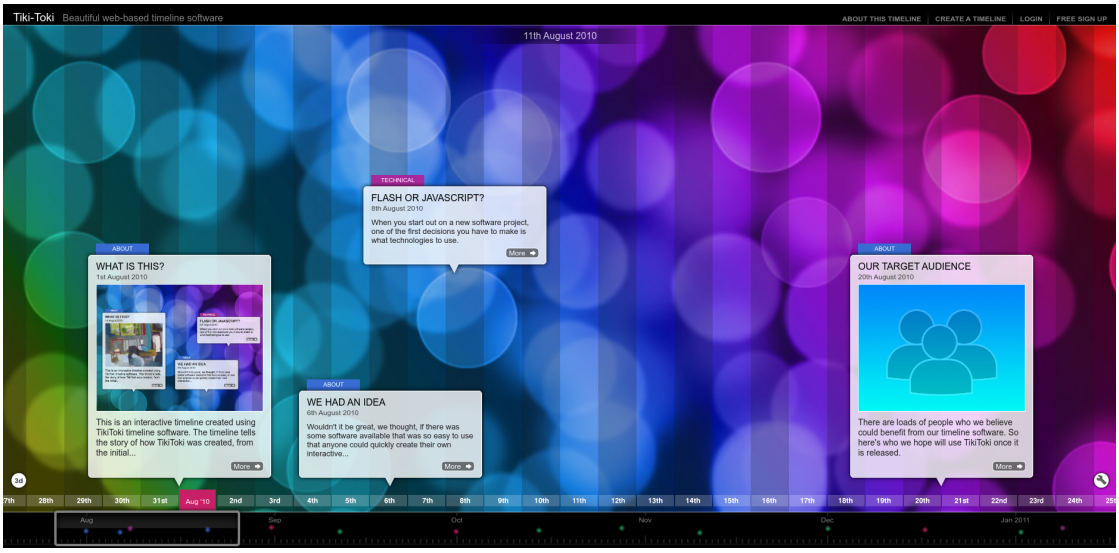
2.4 World Anvil

World Anvil [6] je nástroj, který umožňuje tvůrcům mít přehled o fiktivním světě. Tvůrce jednoduše začne tím, že si vybere co chce o světě napsat (popis postavy, státu, události, živočicha, ...). Nástroj pak tato data zobrazuje jako stránku principem podobnou Wikipedii [7].

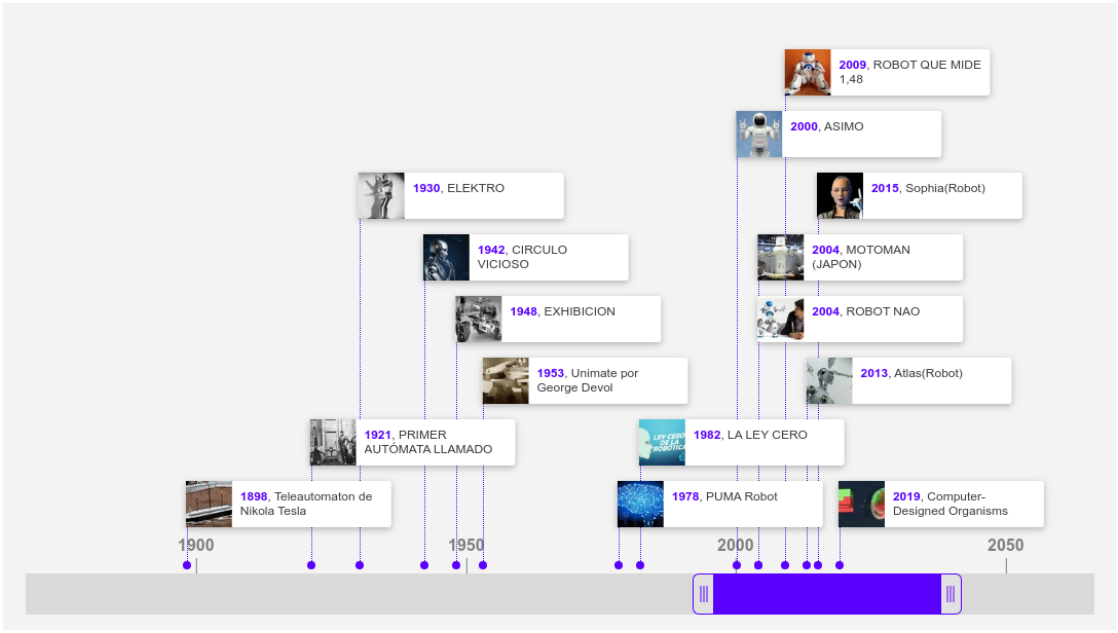
Nástroj také umožňuje tvorbu časových os 2.1. Tyto osy jsou svislé a události jsou zobrazeny z obou stran opět s malou šipkou směřující k ose. Události jsou na osách pouze seřazené a vzdálenosti mezi nimi neodpovídají délce času, který mezi nimi uplynul, ale jen výšce rámečku ve kterém je popisek události.



Obrázek 2.1 World anvil



Obrázek 2.2 Tiki-toki



Obrázek 2.3 Timetoast

Kapitola 3
Analýza podobných aplikací

TimelineSetter [3] se mi líbí snadným použitím, ale výsledná osa se mi zdá nepřehledná. Navíc se tento způsob řešení nehodí pro použití na webovém serveru.

World Anvil [6] má velmi pěkné rozhraní, jejich cíl je podobný jako cíl projektu, ale jejich časové osy jsou celkem nepřehledné pro větší množství událostí, protože každá událost zabírá velké množství místa na obrazovce a pravděpodobně z tohoto důvodu se rozhodli nepoužít chronologicky uspořádanou osu.

Tiki-Toki Timeline Maker [4] je již velmi podobný našemu cíli. Náznak množství událostí na ose v podobě teček na posuvníku velmi pomáhá přehlednosti na dlouhý osách, které nemáme zobrazené na obrazovce celé. Aplikace má ovšem problém se zobrazováním událostí co jsou velmi blízko u sebe, události místo přemístění prostě zobrazí přes sebe a to přehlednosti nepomáhá.

S tímto problémem si velmi dobře poradil Timetoast timeline maker [5], který události přesune o úroveň výše, aby se zamezilo překryvu. Z toho důvodu také používá tenkou svislou čáru, která označuje umístění události na ose.

Další zajímavý prvek této aplikace byl posuvník na časové ose. Úkol tohoto prvku byl posun osy, přiblížení osy a vizualizace, která část osy je zobrazená. Už tomu chyběla jen ta vizualizace množství událostí na ose, kterou předvedl Tiki-Toki Timeline Maker [4]

Kapitola 4
Webové technologie

Cílem projektu je vytvořit webovou aplikaci, tedy stránku. Webová stránka je text (kód), který říká internetovému prohlížeči, co a jak zobrazit a co a kdy dělat.

4.1 HTTP

Prohlížeč ovšem musí tento kód odněkud získat. Každá běžně dostupná webová stránka je umístěna na nějakém počítači, který je dostupný přes internet. Aby mohl prohlížeč získat kód webové stránky, musí nutně nějakým způsobem komunikovat s počítačem, kterému se říká server a tedy poskytuje veřejnosti onu webovou stránku ke čtení.

Základní internetové protokoly umožňují posílat sled jedniček a nul. Aplikace, které si tato data navzájem posílají, se musí dohodnout, co tato data vlastně znamenají. To není problém, pokud jsou například obě aplikace napsané stejným vývojářem. Pokud se ale bavíme o webových stránkách, pro které existuje mnoho různých prohlížečů a mnoho různých aplikací pro poskytování webových stránek, kterým se říká webové servery, začne být problém podstatně komplikovanější.

Naštěstí existuje **HTTP** protokol, který přesně tuto problematiku řeší. Jde o specifikaci formátu v jakém jsou tyto zprávy posílány. Posílané jedničky a nuly jsou v tomto případě většinou chápány jako proud textu. Tento text, dále jen zpráva, je rozdělen na několik částí: metodu, cestu, verzi protokolu, stavový kód, hlavičky a tělo.

Dotaz a odpověď

Zprávy se pak dělí na dva typy, dotaz a odpověď. *Dotaz* 4.1 obsahuje metodu, cestu, verzi protokolu, hlavičky a volitelně tělo, přičemž součástí cesty mohou být další parametry. *Odpověď* 4.2 obsahuje verzi protokolu, stavový kód, stavovou zprávu, hlavičky a opět volitelně tělo. Pokud zpráva obsahuje tělo, jedna z hlaviček určuje, jaký způsobem se má tělo číst (v jakém je tělo formátu).

Výpis kódu 4.1 Příklad dotazu

```
POST /test-api/category HTTP/1.1
Host: asterion-timelines.cz
Content-Type: application/json
Content-Length: 111

{   "name": "Category with icon",
    "description": "Loooong description of the category",
    "iconId": 1 }
```

Výpis kódu 4.2 Příklad odpovědi

```
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Thu, 24 Jun 2021 21:17:05 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 145
Connection: keep-alive
X-Powered-By: Express
Vary: Origin
Access-Control-Allow-Credentials: true
ETag: W/"91-GWIlZDJuxarzBvFOTvgTdGjTM0w"
Set-Cookie: connect.sid=abcdeXYZ; Path=/; HttpOnly; Secure

{   "id": 1,
    "name": "Místo",
    "description": "Libovolné místo, stát, ěmsto nebo kraj" }
```

Komunikující počítače si označme jako klienta a server. Cíl klienta je začít komunikovat se serverem a využívat nějakou službu. Této službě se může také říkat API. Server pak tuto službu poskytuje a odpovídá na klientovy *dotazy*. Klientem je tedy počítač s prohlížečem a server je počítač se spuštěným webovým serverem.

Klient tedy pošle na server dotaz a server na tento dotaz odpoví. Server má nadefinováno, na jaké cestě pomocí jaké metody je jaká služba, a na základě dalších informací poskytnutých klientem v těle dotazu, v parametrech jako součást cesty nebo v hlavičkách se rozhodne, co bude dělat dál. V každém případě musí odpovědět. Vzhledem k povaze protokolu je možné posílat mnoho typů neplatných dotazů, na které musí server nějak odpovědět. Z tohoto důvodu se v odpovědi posílají stavový kód a stavová zpráva, které určují, zda byla odpověď poslána bez problémů, nastala interní chyba na serveru, dotaz byl špatně zformulován apod.

HTTP a webové stránky

Webové stránky jsou obvykle dostupné pomocí metod GET a POST. Prohlížeč při zadání internetové adresy vyšle dotaz s metodou GET serveru, který se na této cestě nachází. Server pak pošle odpověď se stavovým kódem označujícím,že je vše v pořádku, a kódem webové stránky umístěným v těle odpovědi. Prohlížeč už pak díky hlavičkám ví, že má v odpovědi hledat webovou stránku.

Jelikož je webový server pouze aplikace a data poslaná v odpovědi jsou jen data, je běžná praxe, že jsou stránky částečně sestaveny na serveru, odeslány a zapomenuty. Hodí se to například v případech, kdy jsou stránky šité na míru konkrétnímu uživateli, nebo pokud je pod mnoho adresami podobná stránka (např. internetová encyklopedie) apod.

Další možností pro webové stránky je stáhnout potřebná data pomocí dalšího HTTP dotazu až po stažení webové stránky pomocí JavaScriptu. To se hodí především pro interaktivní aplikace, kdy není vhodné znovu načítat stránku při každém kliknutí na nějaké tlačítko.

4.2 Webové stránky

Základní programovací jazyky

Nejjednodušší forma webových stránek si vystačí s HTML souborem. Ten popisuje základní rozložení a prvky webové stránky (nadpisy, podnadpisy, odstavce, seznamy, tabulky, odkazy, obrázky, …) a nějaká metadata.

Pro větší kontrolu nad vzhledem zobrazených prvků vznikl jazyk CSS, který umožňuje nastavení nejrůznějších vlastností u každého prvku zvlášť nebo po skupinách (barva popředí a pozadí, rozměry, průhlednost, pozice, rozložení, změna vzhledu po přesunu kurzoru, …).

Někdy ale ani to nestačí a je třeba vyžadované chování naprogramovat ručně. V takovém případě je zde JavaScript, plnohodnotný programovací jazyk, který umožňuje vkládat, mazat a manipulovat s prvky popsanými v HTML a jejich vlastnostmi specifikovanými v CSS.

Dnes se již objevují další specifikace, které nahrazují JavaScript v místech, kde je velmi důležitá výpočetní rychlost. Například WebGL pro přímou komunikaci s grafickou kartou a WASM.

Preprocesory

Jelikož se musí mnoho prohlížečů shodnout na podobě výše zmíněných technologií, trvá adopce vylepšení celkem dlouho. Navíc ne každý se shodne na směru, kterým se webové technologie ubírají. Proto vznikají nástroje umožňující psát webové stránky v jiném jazyce a následně je přeložit (zkompilovat) do HTML, CSS a JS. Těmto nástrojům říkáme preprocesory.

Jedním z nich je například Sass. Tento nástroj kompiluje SCSS soubory (obdoba CSS s funkcionalitou navíc) do obyčejného CSS. Dalším je TypeScript. To je zase nadstavba nad JS která se do něj zkompiluje.

Aby se nemusel každý nástroj spouštět zvlášť při každé změně kódu pro každý soubor, existují nástroje, kterým se pouze nakonfiguruje, co se má kam a čím zkompilovat.

Kapitola 5

Technologické řešení webové vizualizace

Jsou tři základní způsoby jak něco zobrazit na webové stránce. Jedním je použít základní elementy HTML, nastylovat je pomocí CSS a nastavit jim chování pomocí JS. Druhou možností je použít softwarové vykreslování pomocí HTML elementu canvas (a jeho 2d kontextu). Třetí možností je vykreslovat pomocí WebGL (HTML elementu canvas a jeho WebGL kontextu).

První varianta byla navržena k zobrazování statických prvků a její použití pro neustále se měnící strukturu je velmi pomalé [8]. *Druhá varianta* je na tom už o něco lépe. Je podstatně jednodušší vykreslování základních tvarů na libovolnou pozici ve vymezeném prostoru a je výrazně rychlejší než varianta první [8]. *Třetí varianta* je sice výrazně rychlejší [8], ale má celkem složité rozhraní.

Naštěstí existují knihovny, které zjednodušují práci s WebGL. Mnoho z nich je zaměřeno na 3d zobrazování nebo na tvorbu her (*three.js* [9], *Babylon.js* [10], *TWGL: A Tiny WebGL helper Library* [11], *vtk.js* [12], *PlayCanvas* [13], *Phaser - HTML5 Game Framework* [14]), se kterými by sice šlo zobrazovat pouze 2d objekty, ale knihovny jsou pak zbytečně velké a práce s nimi je pak zbytečně složitější. Pro zobrazování časových os bude stačit 2d vykreslování, které bude ovšem potenciálně zatížené velkým množstvím událostí potřebných k vykreslení. Proto je pro naše účely vhodná knihovna *PixiJS* [15], která má relativně jednoduché rozhraní a umí využívat WebGL kontext.

Kapitola 6

Autentizace a autorizace uživatelů na webu

Tato kapitola se zabývá způsoby, jak bezpečně přihlásit uživatele. To mimo jiné znamená, že nikdo nesmí být schopen získat přístup k uživatelskému účtu bez znalosti přihlašovacího jména a hesla a my jako majitelé databáze nesmíme být schopni získat žádné heslo. Zároveň se v této kapitole vyskytují dva důležité termíny, které je třeba rozlišovat.

Autentizace je ověření identity uživatele. Tedy pokud se někdo snaží přihlásit pak si musíme být jisti, že je to opravdu majitel účtu.

Autorizace je pak ověření zda má přihlášený uživatel pravomoc k využití nějaké služby. Například pouze vybraná skupina lidí má práva k mazání cizích účtů, úpravu dat v databázi a podobně.

6.1 Autentizace

Je několik způsobů jak provést autentizaci uživatele. Jedním ze způsobů je využít služby třetí osoby, jiným je například vygenerovat kód, který se odešle pomocí sms a který se po vyzvání zadá na příslušné stránce. Tato práce se ale bude zabývat autentizací pomocí přihlašovacího jména a hesla, případně komunikací přes elektronickou poštu.

6.1.1 HTTPS

Pokud bychom nic nešifrovali, tak by pro útočníka bylo velmi snadné přihlašovací jméno a heslo odposlechnout. Proti odposlechu komunikace existuje technologie HTTPS. Je to pouze nadstavba nad protokolem HTTP, která je schopná šifrované komunikace. Po dešifrování je ovšem identická s HTTP. Tím, že zašifrujeme komunikaci mezi uživatelem a API, znemožníme útočníkům komunikaci odposlechnout.

Je zde ale hodně jiných problémů. Pokud by se útočník dostal k obsahu naší databáze, mohl by jednoduše použít heslo, které v databázi je a přihlásit se. Nejen to, pokud uživatel použil stejné heslo i jinde, stačí útočníkovi zkusit použít ukradené heslo na známých sociálních sítích, obchodech apod.

6.1.2 Hašování

Abychom tomu zamezili je třeba heslo v naší databázi takzvaně zahašovat. Jde o proces, který je deterministický a jednosměrný. To znamená, že pokaždé když zahašujeme heslo H dostaneme stejný haš H' a že když známe haš H' nejsme schopni získat původní heslo H.

Zahašujeme-li tedy každé heslo předtím než ho vložíme do databáze, můžeme správnost hesla kontrolovat tak, že heslo, které pošle uživatel zahašujeme a výsledný haš porovnáme s tím, který máme v databázi. Zároveň útočník disponující ukradenými daty z naší databáze nebude schopný jen tak použít získaná hesla. Musel by nejdříve vyzkoušet všechna možná hesla[1] a nalézt shodu s některým ze získaných hašů. Teprve poté by mohl nalezené heslo použít.

6.1.3 Útok hrubou silou a slovníkový útok

Naším cílem je tedy donutit útočníka aby použil útok hrubou silou a aby mu takový útok trval co nejdéle. Útok hrubou silou je útok při kterém útočník hádá všechny možné kombinace znaků, které se v hesle mohou vyskytovat. I tady si ale útočník může pomoci. Slovníkový útok spoléhá na to, že uživatel si nastavil lidsky čitelné heslo, které se alespoň částečně sestává ze slov nalézajících se v jeho rodném nebo anglickém jazyce[2]. Útočník pak použije software na prolamování hesel, který je naprogramovaný, aby upravoval slova ze slovníku tak, jak by to udělal průměrný člověk. Tím se výrazně zvýší poměr mezi nalezenými hesly a počtem pokusů.

Ať už způsob přihlášení zabezpečíme jakkoliv, útočník nakonec vždy může hádat hesla, která si uživatel nastavil. Proto je důležité vzdělávat uživatele o bezpečných heslech. Nejjistější metoda je hesla generovat strojově a použít správce hesel pro jejich zapamatování. Tím se do jisté míry zamezí útokům spoléhajícím na nespolehlivý generátor náhodných čísel v lidské mysli.

6.1.4 Duhové tabulky

Další problém jsou duhové tabulky. Duhové tabulky jsou ve zkratce dlouhé seznamy hesel a jejich hašů. Ty jsou samy o sobě velmi dlouhé, ale princip duhových tabulek používá postup, který jejich velikost značně snižuje. Pro naše účely stačí tyto tabulky chápat jako seznam zahašovaných hesel. Duhové tabulky spoléhají na jednoduchost uživatelských hesel a na známé postupy hašování. Každá duhová tabulka funguje jen pro nějaký konkrétní hašovací algoritmus a pro nějaké konkrétní rozmezí uživatelských hesel. Toto rozmezí může ovšem opět využít všech možných taktik, které jsou popsány výše (slovníky, předvídatelné úpravy). Jelikož jsou tyto tabulky i přes různé techniky velmi velké, jedna z nejlepších obran proti nim je použít dlouhé a komplikované heslo. [18]

6.1.5 Opakující se hesla

Uživatelům nic nebrání mít shodná hesla mezi sebou. Naopak, kdyby tomu něco bránilo, byl by to významný bezpečnostní problém. Pokaždé, když by někdo zadal heslo, které již někdo použil, tak by dotyčný měl informaci o tom, že někdo z přihlášených uživatelů takové heslo již má a to ještě předtím, než by získal obsah naší databáze.

Musíme tedy počítat s tím, že různí uživatelé mají stejné heslo. To by se projevilo tak, že by příslušní uživatelé měli i stejný haš. Toho opět může útočník využít, jelikož tím, že zjistí heslo jednoho z nich, zjistí heslo všech.

---

[1] Aby bylo nutné vyzkoušet opravdu všechny možnosti, je třeba použít kryptografický haš, který je pro takové použití vytvořený. Jako příklad uvedu *bcrypt*[16] a *argon2*[17].

[2] Mnoho stránek doporučuje při nastavení hesla použít velká písmena, čísla, speciální znaky apod. To sice pomáhá zpomalit útok hrubou silou, ale pokud útočník použije slovníkový útok, je velice předvídatelné kam uživatel vložil velké písmeno (na začátek), číslo (na konec, pravděpodobně 1-3 číslice) a speciální znaky (mezi slova a pravděpodobně _, &, ,).

6.1.6 Sůl (Salt)

S předchozími dvěma problémy nám může pomoci takzvaná sůl. Sůl je označení pro text, který se nějakým způsobem přiloží k heslu předtím, než se zahašuje. Sůl nám pomůže zajistit složitost textu, který hašujeme i přesto, že zadané heslo bylo relativně jednoduché[3]. Je totiž vhodné použít sůl s dostatečnou délkou a složitostí. Pokud navíc bude sůl unikátní pro každého uživatele, nebude možné identifikovat, kteří dva mají stejné heslo. Je ovšem důležité aby nebyla použita jedna stejná sůl vícekrát napříč všemi uživateli. Bylo by pak možné vytvořit vlastní duhovou tabulku speciálně pro naší databázi a navíc by se nezamezilo problému popsanému výše. [19]

6.1.7 Složitost hašování

Další podstatný prostředek, který lze použít pro zpomalení útočníků, je složitost samotné hašovací funkce. Čím je hašování složitější tím déle trvá vyzkoušet všechna možná hesla. Většina kryptografických hašů je právě z tohoto důvodu značně složitých. Existují i jiné hašovací algoritmy, které mají mnoho kvalit, ale jsou výrazně rychlejší a jsou proto méně vhodné pro ukládání hesel. Jsou vhodné tam kde je třeba zahašovat velké množství informací co nejrychleji. V našem případě se snažíme zahašovat malé množství informací za co nejdelší dobu.

Abychom to případnému útočníkovi moc neusnadnili, je vhodné na naše webové rozhraní nainstalovat nějaký druh omezení rychlosti přihlašování, aby se útočníkovi vyplatilo použít svůj vlastní hardware a nezatěžoval tak náš placený server. Koncový uživatel nic nepozná když přihlašování bude trvat o dvě vteřiny déle, ale útočník, který by se snažil uhodnout miliony kombinací, už by na tom byl hůře.

6.1.8 Identifikátor relace

Abychom nemuseli při každé operaci znovu a znovu (a složitě, jak jsme si již řekli) kontrolovat uživatelské jméno a heslo, nemluvě o jeho neustálém posílání, je vhodné využít k tomu náhodně vygenerovaný text (podstatně delší než heslo a vygenerovaný kryptograficky bezpečným generátorem), který po přihlášení uživateli pošleme zpět. Při každé další komunikaci již pro autentizaci bude stačit tento text (identifikátor relace). Tento identifikátor není nikde v databázi, protože je pouze v pracovní paměti webového rozhraní. Tím, že se uživatel odhlásí, smaže se identifikátor relace k němu příslušící a pro další komunikaci by se uživatel musel opět přihlásit, čímž by získal nový identifikátor.

Jelikož je tento identifikátor náhodně generovaný, není jiná možnost než použít útok hrubou silou, který zároveň nebude benefitovat z taktik zaměřených na lidský faktor. A jelikož je velmi dlouhý je nerealistické, že by někdo uhodl něčí identifikátor v nějaké rozumné době.

6.1.9 Sušenky (Cookies)

Je zřejmé, že přestože jsme autentizaci během většiny operací zjednodušili na poslání jednoho řádku textu, je nicméně stále potřeba tento řádek opakovaně odesílat. Pro zjednodušení této operace existují takzvané HTTP Cookies. Cookies je označení pro data která si aplikace komunikující přes HTTP a HTTPS pamatuje a je schopná je automaticky posílat jako součást všech dotazů. Jedním ze způsobů jak aplikace data získá je, že nějaká odpověď webového rozhraní obsahuje hlavičku, která říká aplikaci, jaká data si má zapamatovat. Příklad takové hlavičky můžeme vidět v kódu 6.1. [20] [21]

Výpis kódu 6.1 Příklad HTTP hlavičky nastavující cookie

```
Set-Cookie: session_id=some_random_number;
```

[3] Stále je třeba aby měl uživatel složité heslo kvůli přímému slovníkovému útoku.

6.1.10 Šifrování cookies

Pokud by někdo zjistil, jaký uživatel používá identifikátor relace, mohl by se za něj po určitou dobu vydávat. To teoreticky lze útokem hrubou silou, ale je opravdu velmi nízká pravděpodobnost, že by se to někomu cíleně povedlo. Další možností je cookies odposlechnout. Tomu ovšem zabráníme tím, že používáme HTTPS protokol. Abychom si byli jisti, že používáme HTTPS protokol, je vhodné použít atribut `Secure`,jak je znázorněno v kódu 6.2, který zajistí, že pokud se uživatel připojuje přes protokol HTTP, cookie se neodešle.

6.1.11 Cross-site scripting

Cross-site scripting, dále jen XSS, je typ útoku, který vloží útočníkův kód do důvěryhodné webové stránky. To je možné díky špatně ošetřené komunikaci mezi uživatelem a webovým rozhraním. Nejdříve je třeba si uvědomit, jak může útočník vložit svůj obsah na stránku, kterou nevlastní. To lze velmi jednoduše. Například při vyhledávání na webu se zadá nějaký hledaný výraz do okénka a stiskne se tlačítko. Toto tlačítko pak přesměruje uživatele na jinou webovou adresu, která mimo jiné obsahuje i hledaný výraz. Tento výraz se pravděpodobně na stránce někde zobrazí. Pokud tedy útočník vytvoří takový odkaz, pak může svůj obsah vložit například právě do této části. Nebezpečí spočívá v tom, co do této části vloží. Pokud by tento hypotetický vyhledávač nebyl proti tomuto útoku odolný a útočník vložil do hledaného výrazu nějaký kód a hledaný výraz se pak vložil do stránky, webový prohlížeč by pak tento kus kódu jednoduše spustil. Z tohoto důvodu musí webové rozhraní ošetřit každý vstup od uživatele a přepsat ten vstup tak, aby prohlížeč kód nespustil. [22]

6.1.12 Cookies a javascript

Cookies jsou přístupné i přes rozhraní v jazyce javascript. To dává možnost útočníkům použít nějaký XSS a získat tak jednotlivé cookies. Jedním ze způsobů je ošetřit webové rozhraní před XSS. Dalším je nastavit dané cookie atribut `HttpOnly`, jak je znázorněno v kódu 6.2, který řekne prohlížeči aby nikdy nedovolil čtení této cookie přes JavaScript. 6.1. [20] [21]

Výpis kódu 6.2 Set-Cookie hlavička s atributy Secure a HttpOnly

```
Set-Cookie: session_id=some_random_number; Secure; HttpOnly
```

6.1.13 Posílání cookies jenom na naše rozhraní

Jak je popsáno výše, cookies jsou pouze data, která je prohlížeč, případně jiný HTTP klient, schopen posílat automaticky s každým dotazem na webové rozhraní. Pokud by ovšem nijak nerozlišoval mezi různými rozhraními, posílal by všechny cookies na všechna webová rozhraní. To v případě identifikátoru relace určitě nechceme. Naštěstí jsou cookies nastaveny tak aby se odesílali pouze na stejnou doménu, která danou cookie nastavila. Toto chování lze upravit atributy `Domain` a `Path`. 6.1. [20] [21]

6.1.14 Cross-site request forgery (CSRF)

CSRF je velmi zákeřný typ útoku. Útočník by, v případě úspěchu, byl schopen posílat dotazy na webové rozhraní pod jménem oběti. Představme si, že jsme vše zabezpečili proti výše uvedeným útokům. Uživatel se skutečně musí přihlásit, aby mohl s rozhraním dále pracovat, a cookies se neposílají na domény, které nám nepatří. Útočník skutečně nezíská naše přístupové údaje. On je totiž nepotřebuje. Když se přihlásíme na důvěryhodnou stránku, získáme tím cookie s naším

identifikátorem. Pokud chceme pracovat s rozhraním, musíme mu spolu s HTTP dotazem poslat i cookie. Útočník tedy nepotřebuje znát žádné naše údaje, pokud je nějakým způsobem schopen zařídit, aby *z nějakého* počítače byl na rozhraní odeslán dotaz s útočníkem zadanými parametry a *s patřičou cookie oběti*. Jak je již uvedeno, patřičnou cookie má k dispozici pouze oběť a webové rozhraní. Tedy pro útočníka je nejlogičtějším krokem donutit HTTP klienta oběti odeslat jím sestavený dotaz na příslušné rozhraní. To je ovšem bez patřičné ochrany velmi jednoduché.

Útočník může například vytvořit vlastní webovou stránku. Každá webová stránka má možnost odeslat dotaz na libovolné rozhraní. Útočník na stránku umístil kód, který například odešle dotaz na webové rozhraní nějaké známé banky, který odešle nějaké množství peněz sobě na účet. Jelikož je dotaz směřovaný na rozhraní banky, tak pokud si prohlížeč pamatuje cookies poslané z rozhraní této konkrétní banky, odešle tyto cookies na rozhraní, které jednoduše uživatele ověří a odbaví dotaz.

6.1.15 SameSite atribut

Kvůli CSRF vznikl atribut `SameSite`, který prohlížeči říká, jestli má patřičnou cookie odesílat pokud se uživatel nachází na jiné doméně, než ze které cookie pochází. atribut může nabývat několika hodnot[4] (`Lax`, `Strict`, `None`). Pokud je použita hodnota `None` cookie je zranitelná vůči CSRF. Pokud ovšem zmíněná cookie není schopná způsobit žádnou škodu, může to být žádoucí chování. Hodnota `Lax` zamezí odeslání cookie pokud se uživatel nachází na stránce třetí strany. Hodnota `Strict` funguje stejně jako hodnota Hodnota `Lax`, ale navíc ještě blokuje cookie, pokud by odeslání cookie bylo způsobeno kliknutím na odkaz na stránce třetí strany, přestože odkaz vedl na naše rozhraní. Tímto by se mělo minimalizovat riziko CSRF.

Tím že atribut nastavíme jako `Strict`, může to negativně ovlivnit uživatelovu zkušenost s naší aplikací. Představte si, že na nějaké stránce kliknete na odkaz na facebook a facebook se rozhodne, že vás kvůli tomu nepřihlásí. Aby se toto nestávalo je vhodné neautentizovat uživatele hned při vstupu na stránku, ale použít následné dotazy na rozhraní až z té konkrétní stránky.

Další doporučení říká, že je vhodné při návrhu rozhraní nepoužívat typu `GET` pro potenciálně nebezpečné dotazy, jelikož `GET` dotazy se standardně dají vložit do obyčejného hypertextového dotazu a je zde možnost, že při otevření odkazu z jiné aplikace (třeba poštovního klienta) v prohlížeči nebude správně detekováno a dojde k úspěšnému CSRF útoku i přes ochranu, kterou poskytuje `SameSite=Strict` atribut. [23]

6.2 Autorizace

Autorizace je mnohem jednodušší než autentizace. Jelikož staví na předpokladu, že jsme přihlášené uživatele úspěšně autentizovali, není zde třeba žádná znalost kryptografie. Stačí když si vymyslíme nějaký systém povolení. Můžeme například vytvořit nějakou hierarchii uživatelů, nějaké stupně pověření, nebo jednotlivá práva pro jednotlivé operace či skupiny operací.

Je ovšem důležité být na pozoru před následujícím. Mějme administrátora A, který má všechna existující práva. Dále mějme správce S, který má práva potřebná ke správě jiných uživatelů. Pokud si nedáme pozor při udělování práv, může se stát, že S bude schopen uživatele A smazat nebo bude schopen vytvořit nového uživatele U, který bude mít stejná práva jako A. Tímto způsobem by S byl schopen převzít kontrolu nad systémem a kompletně odříznout administrátora A.

Proto lze implementovat pravidlo, že právo uživatele U měnit práva ostatních uživatelů je limitováno na práva, která U sám vlastní. Tedy mějme uživatele U vlastnícího taková práva spolu s právy přidávat nová data do databáze, ale už ne je mazat. Pokud U udělí uživateli V všechna práva, která je schopen udělit, pak V bude schopen měnit práva uživatelů, přidávat data do databáze, ale už ne je z databáze odebírat. Jinak řečeno V by byl oprávněný ke stejným věcem

[4] Je vhodné zmínit, že při absenci atributu `SameSite` při nastavení cookie, se většina moderních prohlížečů zachová tak, jako kdyby byla nastavena hodnota `Lax`

jako U. Dále lze právo měnit práva uživatelů dále rozdělit na více limitující práva, ovšem vždy by měla platit výše popsaná podmínka.

Tím nám ale vznikl drobný problém. Pokud bude uživatel A jediný s nějakým právem a ztratíme přístup k uživateli A pak bychom nebyli schopni takové právo nikomu udělit. Zbývá nám pak tedy jedině zavolat správci databáze aby ručně vložil nového uživatele nebo udělil nějaké právo. To s sebou ovšem přináší riziko lidské chyby, která může mít vliv na celou databázi.

Kapitola 7

Množina časových dat světa Asterion

Jak jsem již zmínil, cílem tohoto projektu je vizualizace jednotlivých událostí na časové osy. Událostí může být libovolný popis dění. Jelikož je třeba události zobrazit na časové ose, je třeba je označit nějakým časovým údajem popisujícím dobu, kdy se událost odehrála. Většina událostí je popsána tak, že je lze označit jedním časovým údajem. Některé události ovšem chápeme jako nějaké období (například války), u kterých by bylo vhodné určit jak jejich začátek tak jejich konec. Ostatní události lze dekomponovat na ty již zmíněné. Například pokud bych chtěl popsat že období vlády P. Panovníka III. se dělí na několik částí (třeba podle věku nebo životních zkušeností), bylo by možné tuto skutečnost popsat separátními událostmi následovně:[1]

- Vláda P. Panovníka III. od časové značky 0 do časové značky 500
- Vláda P. P. III. v zastoupení S. Správce Hanebného od časové značky 0 do časové značky 100
- Vláda P. P. III. ve formě diktatury od značky 100 do značky 250
- Panovník se pohřešuje od 250 do 300
- Vláda P. P. III. ve formě parlamentní monarchie od 300 do 500

V předchozím příkladu používám časové značky namísto dat. Je to z toho důvodu, že svět Asterion má vlastní kalendář. Naštěstí je mnohem jednodušší než Gregoriánský kalendář. Rok (370 dní) na Asterionu se skládá ze 12 měsíců, kde každý má 30 dnů, a 10 speciálních svátkových dnů. Svátkové dny dělají kalendář lehce složitější. Svátkový den totiž nepatří k žádnému měsíci a tak nelze namapovat všechny dny v roce ke konkrétnímu datu ve formátu měsíc a den. Pro příklad uvedu data 30. Zelenec (poslední den, 5. měsíc) a 1. Ploden (první den, 6. měsíc), mezi kterými leží dva svátkové dny (Svátek letních duchů). Pro počítačové zpracování je tedy vhodnější zaznamenat pořadí dne v celém roce. Tedy 30. Zelenec je 152. den v roce a 1. Ploden je 155. den v roce. Při použití této metody je pak třeba ve frontendu tento datový typ převést na lidsky čitelné datum. Vzhledem k potřebě takového převodu, je jen logické, zahrnout do stejného čísla i rok a to tak, že časová značka bude reprezentovat počet dní od 1. dne roku 0 v Asterionu. Díky statickému počtu dní v roce je pak velice snadné spočítat rok i den v něm.

[1] Rád bych poukázal na možnost dekomponovat každou událost se značkami od a do na dvě samostatné události. (značka 0: Vláda P. P. III. začala; značka 500: Vláda P. P. III. skončila)

Část II

Praktická část

Kapitola 8

Návrh prototypu

Aplikace se často dělí na backend a frontend. Zjednodušeně řečeno frontend je část aplikace, se kterou interaguje uživatel, a backend je zbytek. Cílem této práce je navrhnout a nasadit backend aplikace, proto se v této kapitole zaměřím na návrh databáze a webového API.

Popíši zde požadavky, případy užití a doménový model, navrhnu diagram aktivit, diagram tříd, databázový model a model nasazení.

8.1 Požadavky

- Autorizace uživatelů
- Přidávání, odebírání a editace událostí autorizovanými uživateli
- Přidávání, odebírání a editace štítků (tagů) autorizovanými uživateli
- Označení událostí jedním nebo vícero tagy
- Vyhledávání tagů
- Vyhledávání událostí podle zvoleného tagu nebo nějakého aliasu, který označuje jeden nebo více tagů
- Neautorizovaný uživatel nesmí být schopen manipulovat s daty v databázi.
- Nikdo nesmí být schopen z databáze nebo během autorizace získat něčí heslo.
- Nikdo nesmí být schopen změnit něčí heslo bez přístupu k tomuto účtu.
- Nikdo nesmí být schopen použít něčí účet bez znalosti přihlašovacích údajů.
- Snadno pochopitelná API

8.2 Případy užití

V této sekci se zaměřím na jednotlivé příklady použití navrhované API.

Vyhledání událostí podle tagů a metatagů (filtrů)

Jedním z hlavních cílů projektu je v určité podobě zobrazovat události týkající se konkrétních tagů a metatagů. (Tuto dvojici budu nadále nazývat filtry.) Ovšem pro účely úpravy dat v databázi by bylo vhodné implementovat i vyhledávání podle názvu a popisu události s případným časovým omezením.

Vyhledání filtrů

Aby bylo možné filtrovat události s konkrétním filtrem, je třeba takový filtr napřed vyhledat. Filtry by mělo být možné hledat podle jména a popisu. V nejlepším případě by se vyhledané filtry seřadily podle podobnosti s hledaným výrazem. Tyto filtry by měly být seskupeny do kategorií určujících zda se jedná o místo, osobu a podobně.

Zobrazování ikon

Součástí událostí, filtrů a kategorií může být i ikona. Odkaz na ni by měl být posílán jako součást vyžádaných dat.

Autorizace uživatele

Uživatel se bude moci do API zaregistrovat, přihlásit a odhlásit. Po přihlášení bude skrze API možné provádět operace, které jsou běžně nedostupné, předpokládaje, že k tomu byl uživatel oprávněn. Zároveň není po uživateli API vyžadováno autentizaci řešit vyjma přihlášení, odhlášení a zaregistrování, jelikož při každém volání nějaké API služby se pomocí cookies provede autentizace a následná autorizace automaticky.

Přidávání dat

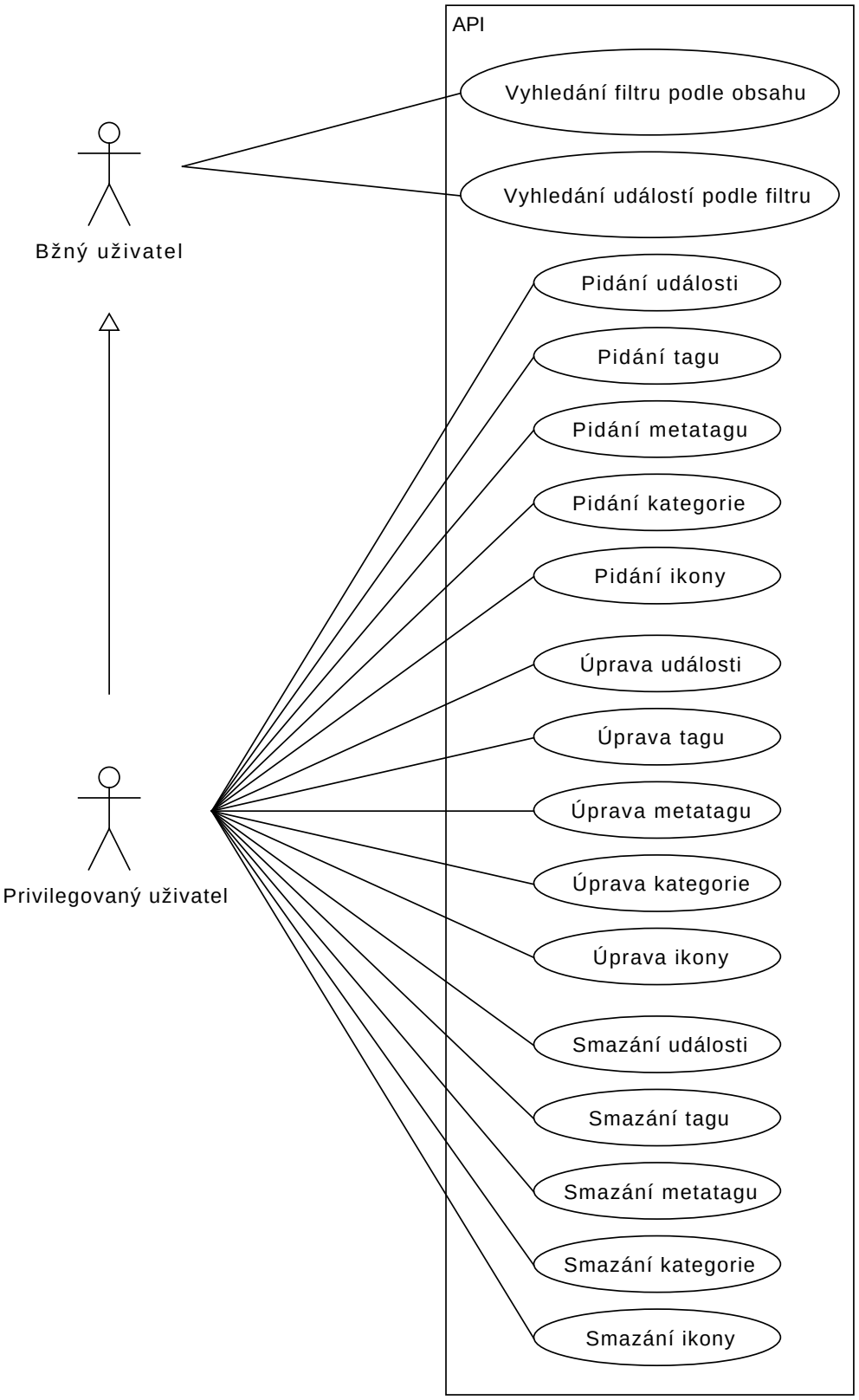
Privilegovaný uživatel by měl být schopen přidávat události, filtry, kategorie a ikony.

Úprava dat

Privilegovaný uživatel by měl být schopen upravovat události, filtry, kategorie a ikony.

Smazání dat

Privilegovaný uživatel by měl být schopen mazat události, filtry, kategorie a ikony.



Obrázek 8.1 UseCase diagram

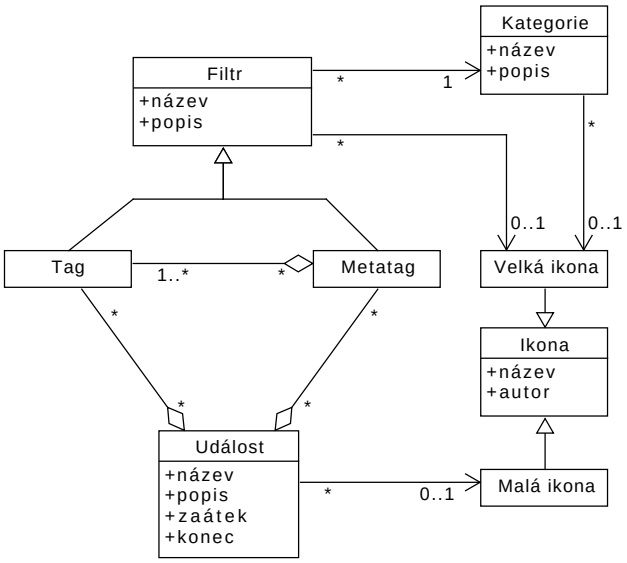
8.3 Doménový model

Historická data lze namodelovat následujícím způsobem. Mějme třídu Filtr označující subjekty zájmu (podstatná jméno ve větách) a třídu Událost označující jejich nebo jich se týkající činnosti (slovesa ve větách).

Subjekty zájmu lze ale dále seskupovat nebo přejmenovávat. Například božstvo může být označení pro skupinu více subjektů (jednotlivých bohů) a každý bůh může mít více jmen, pod kterými je znám. Rozdělme tedy třídu Filtr na podtřídy Tag a Metatag a propojme je m:n vazbou. Událost pak bude mít m:n vazbu jak s třídou Tag tak s třídou Metatag.

Dále lze jednotlivé subjekty kategorizovat do velkých skupin, jako například místa, osoby apod. Mějme tedy třídu Kategorie, která bude přiřazena každému filtru.

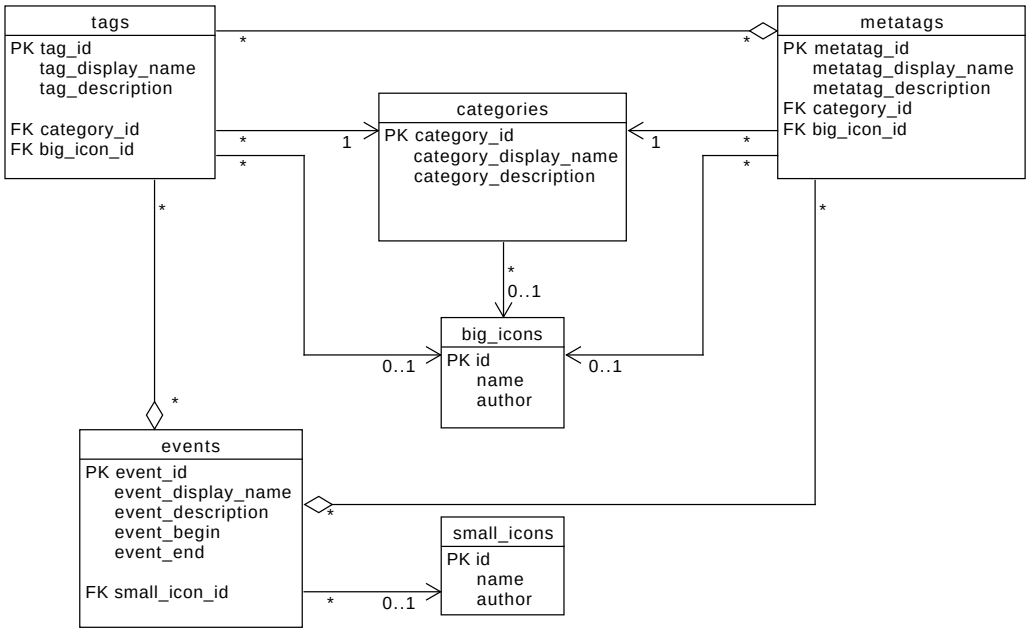
Vzhledem k tomu, že vytváříme grafickou aplikaci, je třeba jednotlivé filtry a události vizualizovat. Proto mějme třídu Ikona a její podtřídy Velká a Malá ikona. Velké ikony jsou přiřazeny k filtrům a kategoriím a malé ikony k událostem. Kategorie mají přiřazenou ikonu pro případ, že samotný filtr žádnou přiřazenou nemá a v případě, že ani kategorie nemá žádnou přiřazenou, bude vybrána jedna výchozí. Zároveň bude existovat i výchozí malá ikona pro případ, že událost nebude mít žádnou konkrétní přiřazenou.



Obrázek 8.2 Doménový model

8.4 Databázový model

Na základě předchozího doménového modelu je navržen následující model databáze, ovšem byla z něj vyjmuta dědičnost tříd, která se vyskytuje až v business logice. Z modelu je vynechána tabulka uživatelů obsahující id, uživatelská jména, emaily a haše hesel a tabulka povolení obsahující id. Tyto dvě tabulky mezi sebou mají m:n vazbu.



Obrázek 8.3 Databázový model

8.5 Návrh webového rozhraní

Rozhraní by mělo obsahovat několik hlavních sekcí. Události, tagy, metatagy, filtry, kategorie a ikony jsou sekce zodpovědné za přidávání, úpravu a mazání dat. Filtry jsou ovšem pouze pomocná sekce pro vyhledávání mezi tagy a metatagy zároveň. Další sekce by měla být zodpovědná za autentizaci uživatelů.

Události

Tato sekce by měla být přístupná na adrese `/event` (tedy `url.cz/api/event`) a pomocí http metod GET, POST, PUT a DELETE zprostředkovává funkce vyhledávání pomocí id, přidávání, úpravu a mazání událostí. Dále by měla obsahovat dotazy na vyhledání pomocí id filtru a pomocí textového obsahu události.

Tagy a metatagy

Tyto sekce by měly být přístupné na adresách `/tag` a `/metatag` a pomocí http metod GET, POST, PUT a DELETE zprostředkovávat funkce vyhledávání pomocí id, přidávání, úpravu a mazání tagů a metatagů. Dále by měla obsahovat dotazy na vyhledání podle textového obsahu a

vyhledání asociovaných protějšků, tedy v sekci Tagy vyhledání všech metatagů obsahujících daný tag a v sekci Metatagy vyhledání všech tagů obsažených v daném metatagu.

Filtry

Tato sekce by měla být zpřístupněna na adrese `/filter` a pomocí http metody GET zprostředkovávat vyhledávání jak mezi tagy tak mezi metatagy pomocí id. Dále by měla obsahovat dotaz na vyhledání tagů a metatagů podle textového obsahu.

Kategorie

Tato sekce by měla být přístupná na adrese `/category` a pomocí http metod GET, POST, PUT a DELETE zprostředkovává funkce vyhledávání pomocí id, přidávání, úpravu a mazání kategorií tagů a metatagů. Dále by měla obsahovat dotazy na vyhledání pomocí textového obsahu kategorie a na vypsání všech kategorií.

Přihlašování

Tato sekce by měla být přístupná na adrese `/sign` a pomocí dotazů `/sign/up`, `/sign/in` a `/sign/out` zajišťovat registraci, přihlášení a odhlášení uživatele. Dále by měla obsahovat dotazy pro změnu informací přihlášeného uživatele, žádost o reset hesla a žádost o odstranění účtu a údajů s ním spojených.

Uživatelé

Tato sekce by měla být přístupná na adrese `/user` a pomocí http metod GET, POST, PUT a DELETE zprostředkovává funkce vyhledávání pomocí id, přidávání, úpravu a mazání uživatelů. Dále by měla obsahovat dotazy na přidávání a odebírání oprávnění uživatelům.

Kapitola 9
Implementace aplikace

Aplikace je napsána v jazyce JavaScript a spouštěna pomocí Node.js. Pro práci s http protokolem byla vybrána knihovna express.js, pro hašování hesel knihovna argon2, pro přihlašování uživatelů knihovna passport.js a pro komunikaci s databází knihovna mysql2.

9.1 Přihlašování uživatelů

Předtím než bylo rozhodnuto pro zřízení vlastní databáze uživatelů bylo v plánu použít autentizaci poskytovanou společností Google. To se ale ukázalo být překvapivě obtížné kvůli dokumentaci, která neobsahovala řešení žádného problému, který se naskytl, a všemožné návody (přímo od googlu) byly roztroušeny napříč různými sekcemi dokumentace google api schovanými mezi mnohými jinými návody na management skupin a jiných nám nepotřebných funkcí. Postup v návodu, na který se odkazovalo nejčastěji, vyžadoval použití knihovny, která byla v jiných částech dokumentace zastaralá, a na novou knihovnu se nepodařilo najít dokumentaci zabývající se základním použitím.

Rozhodlo se tedy pro passport.js. Dokumentace passport.js sice také nebyla perfektní, byla velmi stručná a některé věci si člověk musel domyslet, ale alespoň byla konzistentní. Passport.js navíc umožňuje jednoduché začlenění jiných přihlašovacích strategií do frameworku express.js v budoucnu.

Zatímco passport.js se stará o udržování sezení přihlášeného uživatele, při použití lokální strategie je třeba naimplementovat databázi uživatelů. Ta v našem případě obsahuje uživatelské jméno, email a haš hesla. Navíc se k ní váží práva uživatelů. Hašování a ověřování hesel je implementováno pomocí algoritmu argon2, který vyhrál Password Hashing Competition (PHC) v roce 2015 a mimo jiné automaticky solí hesla náhodným řetězcem.

9.2 Code injection

Aby nedošlo k žádnému útoku pomocí code injection, je používán systém připravených příkazů (prepared statements) v mysql, který odděleně pošle SQL příkaz a poté data. Databáze tak od sebe dokáže jednotlivé části odlišit a nedojde tak záměně uživatelských dat za příkaz. Zároveň je na celou API nasazen překladač, který nahrazuje podezřelé znaky za jejich html reprezentaci (html entitu), aby nedošlo k vložení kódu na stránku ze strany nevhodných dat vložených do databáze.

9.3 Vrstvy

API je rozdělena do tří hlavních vrstev. První, databázová, vrstva má na starost komunikaci s MySQL, druhá má na starost překlad dat z databáze do správného formátu a získání doplňujících dat pomocí databázové vrstvy a třetí vrstva zprostředkovává tato data na specifických umístěních v API pomocí express.js frameworku.

9.4 Finální podoba API

Implementovaná API obsahuje následující dotazy (získané pomocí dotazu `GET::/endpoints`) zapsané v JSON formátu 9.1 tak, že každý string představuje jeden dotaz. Metoda dotazu je rovna obsahu stringu. Cesta dotazu je rovna zápisu všech názvů předků pole (s názvem `"::"`) obsahujícího daný string. Pro příklad uvedu první dotaz. Dotaz má metodu POST a je přístupný na adrese `/sign/up`, respektive `http://www.asterion-timelines.cz/api/sign/up`.

Formát parametrů

Dotazy s metodou GET jsou parametrizovány pomocí URL encoded query a dotazy s jinou metodou jsou parametrizovány pomocí těla dotazu ve formátu JSON. Výjimkou jsou dotazy obsahující jako parametr soubor (především dotazy týkající se ikon) ty jsou parametrizovány pomocí těla dotazu ve formátu multipart form data.

Použité metody

Dotazy s metodou GET slouží k získání dat, s metodou POST k vytvoření nových dat, s metodou PUT k úpravě již existujících dat a s metodou DELETE ke smazání dat. Výjimkou mohou být dotazy na adresách obsahujících prefix `/sign` a `/ping`.

Výpis kódu 9.1 Finální podoba API

```
{   "/sign": {
        "/up": { "::": [ "POST" ]},
        "/in": { "::": [ "POST" ]},
        "/out": { "::": [ "POST" ]},
        "/logged": { "::": [ "GET" ]},
        "/user": { "::": [ "GET", "PUT" ]},
        "/check_login": { "::": [ "GET" ]},
        "/remove_account": { "::": [ "POST" ]}},
    "/user": { "::": [ "GET", "PUT", "POST", "DELETE" ],
        "/byLogin": { "::": [ "GET" ]}},
    "/permit": { "::": [ "GET" ],
        "/byName": { "::": [ "GET" ] },
        "/all": { "::": [ "GET" ] }},
    "/tag": { "::": [ "GET", "POST", "PUT", "DELETE" ],
        "/byContent": { "::": [ "GET" ]},
        "/metatags": { "::": [ "GET", "PUT" ]}},
    "/metatag": { "::": [ "GET", "POST", "PUT", "DELETE" ],
        "/byContent": { "::": [ "GET" ]},
        "/tags": { "::": [ "GET", "PUT" ]}},
    "/filter": { "::": [ "GET" ],
        "/byContent": { "::": [ "GET" ]}},
    "/category": { "::": [ "GET", "POST", "PUT", "DELETE" ],
        "/byContent": { "::": [ "GET" ]},
        "/all": { "::": [ "GET" ] }},
```

```
    "/event": { "::": [ "GET", "POST", "PUT", "DELETE" ],
        "/byContent": { "::": [ "GET" ] },
        "/byFilterId": { "::": [ "GET" ] }},
    "/icon": {
        "/tags": { "::": [ "GET", "POST", "PUT", "DELETE" ],
            "/neededBy": { "::": [ "GET" ] },
            "/byPath": { "::": [ "GET" ] },
            "/byName": { "::": [ "GET" ] },
            "/all": { "::": [ "GET" ] },
            "/default": { "::": [ "PUT" ] },
            "/default_from_path": { "::": [ "PUT" ] },
            "/default_from_id": { "::": [ "PUT" ] }},
        "/events": { "::": [ "GET", "POST", "PUT", "DELETE" ],
            "/neededBy": { "::": [ "GET" ] },
            "/byPath": { "::": [ "GET" ] },
            "/byName": { "::": [ "GET" ] },
            "/all": { "::": [ "GET" ] },
            "/default": { "::": [ "PUT" ] },
            "/default_from_path": { "::": [ "PUT" ] },
            "/default_from_id": { "::": [ "PUT" ] }}},
    "/ping": {
        "/api": { "::": [ "GET" ] },
        "/logged": { "::": [ "GET" ] },
        "/db": { "::": [ "GET" ] },
        "/log": { "::": [
            "GET", "POST", "PUT", "DELETE",
            "HEAD", "CONNECT", "OPTIONS",
            "TRACE", "PATCH" ]}},
    "/endpoints": { "::": [ "GET" ],
        "/permitted": { "::": [ "GET" ] }}}
```

Hlavní dotazy

Dotazy na adresách s prefixem `/permit`, `/tag`, `/metatag`, `/category`, `/event`, `/filter`, `/icon/tags` a `/icon/events` pojmenujme jako *hlavní dotazy* a tyto prefixy jejich adres chápejme dále také jako `/prefix`. Hlavní dotazy pak dodržují následující strukturu.

Vyžadované parametry - GET

Hlavní dotazy s metodou GET na adresách `/prefix` vyžadují parametr `id` a v odpovědi vrátí JSON s daty daného objektu. Pro vyhledávání mohou existovat dotazy s metodou GET na adresách `/prefix/bySomething`, kde `something` vyjadřuje parametr, pomocí kterého vyhledáváme (všimněte si malého písmene na začátku parametru). Tyto *vyhledávací* dotazy mohou vracet pole (stále ve formátu JSON) pokud se vyhledává pomocí parametru, který není unikátní. Dotazy na adresách `/prefix/all` pak vrací všechna známá data tohoto typu.

Vyžadované parametry - PUT

Hlavní dotazy s metodou PUT jsou parametrizovány parametry téměř totožnými s těmi, které získáme dotazem `/prefix` s metodou GET. Parametr `id` je vždy povinný, tím se určí, který objekt chceme upravit. Ostatní parametry jsou nepovinné (mohou být `undefined`).

Rozdíl je ve vnořených objektech v dotazech s formátem JSON. Pokud získaný objekt z odpovědi obsahuje jiný objekt nebo pole objektů (9.2), dotaz s metodou PUT použitý k úpravě

takového objektu požaduje pouze jeho id nebo pole všech id (9.3) (všiměme si změny z `"object"` na `"objectId"` a změny z `objects` na `objectIds`).

Při úpravě zmíněných polí je třeba vložit pole všech id, které chceme vložit, API pak vytvoří dvě nová pole (co vložit a co smazat), která využije při odbavení dotazu jako změnové vektory. Pokud tedy byla na serveru uložená data 9.2 a odeslali jsme dotaz s metodou PUT a daty 9.3 budou data v databázi vypadat následovně 9.4. Pokud chceme pole zachovat nezměněné nebudeme parametr `objectIds` vůbec posílat a nebo ho nastavíme jako `undefined`.

Výpis kódu 9.2 Odpověď na dotaz GET s id 5

```
{
   "id": 5,
   ...,
   "object": {
      "id": 7,
      ...
   },
   "objects": [
      {
         "id": 4,
         ...
      }
      {
         "id": 2,
         ...
      }
   ]
}
```

Výpis kódu 9.3 Tělo dotazu s metodou PUT

```
{
   "id":5,
   ...,
   "objectId": 89,
   "objectIds": [1, 2, 8]
}
```

Výpis kódu 9.4 Odpověď na druhý dotaz GET s id 5

```
{
   "id": 5,
   ...,
   "object": {
      "id": 7,
      ...
   },
   "objects": [
      {
         "id": 4,
         ...
      }
      {
         "id": 2,
         ...
      }
   ]
}
```

Vyžadované parametry - POST

Hlavní dotazy s metodou POST jsou parametrizovány stejně jako s metodou PUT ovšem parametru `id` je nepovinný a ignorován. Dále jsou pro tyto dotazy povinné následující parametry.

- `/category`
  - `name`
  - `description`
- `/tag, /metatag`
  - `name`
  - `description`
  - `categoryId`
- `/event`
  - `name`
  - `description`
  - `begin` (počet dnů od dne 0 roku 0, `{ return (370 * rok) + den }`)
- `/icon/tags, /icon/events`
  - `name` (multipart form data text)
  - `icon` (multipart form data soubor)

Vyžadované parametry - DELETE

Hlavní dotazy s metodou DELETE jsou parametrizovány pouze parametrem `id`, který je samozřejmě povinný.

Zvláštní případ - /filter

Všiměme si, že sekce `/filter` obsahuje pouze dotazy s metodou GET. Je to proto, že objekty typu filter jsou poze ke čtení a vznikají kombinací dat typu tag a metatag. **Pozor**. Id filtru není zpětně kompatibilní s id tagu a id metatagu. Pokud potřebujete id tagu nebo metatagu je nutné [1] použít vyhledávací dotazy tagů a metatagů.

Zvláštní případ - /icon/events a /icon/tags

Ikony mají několik dotazů, které neodpovídají výše uvedené struktuře. Dotaz `/neededBy` vypíše všechny události popřípadě tagy, metatagy a kategorie, které jsou závislé na ikoně specifikované pomocí parametru `id`. Dotaz `/default` nastaví výchozí ikonu na soubor odeslaný v parametru `icon`. Dotaz `/default_from_path` nastaví ikonu (stejného typu) nacházející se na adrese specifikované parametrem `path` jako výchozí ikonu. Dotaz `/default_from_id` nastaví ikonu (stejného typu) mající id specifikované parametrem `id` jako výchozí ikonu.

[1] Pokud není jiná cesta, lze id převést následujícím způsobem, ovšem tento způsob není součástí dokumentace a může se v budoucnu změnit. Id filtru je string. Pokud má na začátku podtržítko ('_'), je třeba ho odstranit. Tím získáme id metatagu. Pokud nemá podtržítko, získali jsme id tagu. Je ovšem nutné dodat, že id tagů a metatagů jsou čísla.

Zvláštní případ - /tag/metatags a /metatag/tags

Tyto dotazy slouží k úpravě m:n vztahu mezi tagy a metatagy a dodržují konvenci ohledně vnořených polí a objektů zmíněnou výše. Pro upřesnění uvedu, že dotazy `/tag/metatags` pracují s parametrem `/metatagIds` a dotazy `/metatag/tags` pracují s parametrem `/tagIds`.

Endpoints

V sekci přístupné na adrese `/endpoints` jsou dva dotazy. Dotaz `/endpoints::GET` poskytuje seznam všech dotazů stejně jak je popsáno výše a `/endpoints/permitted::GET` poskytuje objekt reprezentující slovník mezi dotazem a hodnotou true/false označující zda má uživatel povolení využívat tento dotaz.

Ping

Sekce na adrese `/ping` jsou následující dotazy. Dotazy `/ping/api::GET` a `/ping/db::GET` odpoví statusem 200 a zprávou Ping. Pokud ne tak není v provozu api, respektive není v provozu spojení api s databází. `/ping/logged::GET` odpoví statusem 200 a zprávou Ping pokud je uživatel přihlášen, pokud ne tak status odpovědi bude 401. Dotazy `/ping/log` zapíší data dotazu do logu podezřelých. Tato skupina dotazů vznikla kvůli objeveným voláním adresy `/jsonws/invoke` z neznámé Ruské IP a je zamýšlena jako místo kam přesměrovat podezřelé dotazy pomocí web serveru nginx.

Sign

Sekce na adrese `/sign` slouží primárně k manipulaci se stavem přihlášení a manipulaci s daty přihlášeného uživatele.

Přihlašování, odhlašování a registrace

Registrace, přihlašování a odhlašování jsou zprostředkovány pomocí dotazů `/sign/up`, `/sign/in` a `/sign/out`. Všechny dotazy používají metodu POST. Registrace vyžaduje parametry `username`, `email` a `password`. Přihlášení vyžaduje parametry `login`, který může představovat jak username tak email, a `password`. Odhlášení nevyžaduje žádné paraetry.

/sign/check_login

Dotaz přijímá parametr `login`, který představuje email nebo uživatelské jméno. Pokud je email, nebo uživatelské jméno již obsazené, vrátí odpověď `{free: false}` a v opačném případě `{free: true}`. Momentálně přihlášený uživatel nemá s dotazem souvislost. Dotaz byl vytvořen za účelem zjednodušení implementace frontendu, konkrétně registračního formuláře.

/sign/remove_account

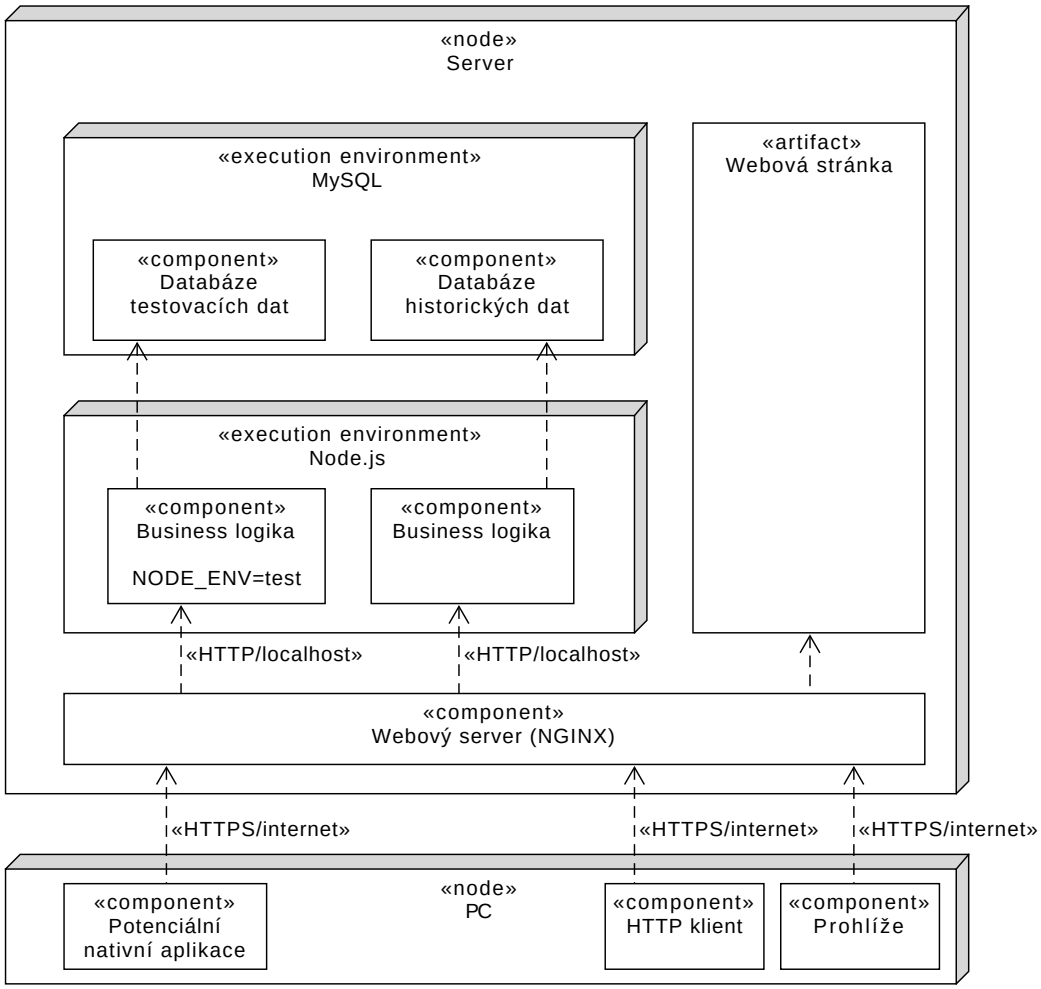
Dotaz odstraní účet přihlášeného uživatele, tedy jeho email, uživatelské jméno, haš jeho hesla a informaci o tom, jaká vlastnil oprávnění. **Frontend aplikace by měly být velmi obezřetné s použitím této operce.**

/sign/user

Dotaz `/sign/user` pomocí metody GET slouží k získání informací o přihlášeném uživateli. Dotaz byl vytvořen za účelem zjednodušení implementace frontendu, konkrétně indikaci přihlášeného uživatele. Dotaz `/sign/user` pomocí metody PUT slouží k úpravě informací o přihlášeném uživateli (uživatelské jméno, email, heslo) a pracuje s parametry `username`, `email` a `password`, pokud je parametr vynechán zůstane informace, kterou představuje nezměněna.

9.5 Nasazení aplikace

Aplikace je nasazena na zadavatelem poskytnutém serveru. Zabezpečenou komunikaci přes https s uživatelem zprostředkovává webový server nginx[24], který na adrese `/` vrací statickou webovou stránku (zkompilovaný frontend) a na adrese `/api/` přeposílá dotazy Node.js aplikaci (backend), která následně využívá MySQL databázi. Druhá dvojice Node.js aplikace a MySQL databáze je pak přístupná na adrese `/test-api/` pro účely testování.



Obrázek 9.1 Model nasazení

9.6 Testování aplikace

Testování aplikace probíhalo v několika krocích. Po implementaci nějakého dotazu se funkce ručně otestovala v aplikaci Postman. Poté byl přidán nový dotaz do sekvence automatizovaných testů (opět v aplikaci Postman), která byla spouštěna pravidelně během práce na projektu.

Byla také vytvořena webová stránka, na které se testovalo praktické použití API a odolnost vůči code injection útokům.

Dále byla otestována slabost proti útokům popsaným v kapitole 6 týkajících se technologii cookies. Cookies nejsou přístupné přes javascript, cookies se neposílají na jiné webové stránky než a cross-site request forgery se nezdařilo.

Pomocí nástroje MySQL Workbench bylo zkontrolováno, že hesla jsou zahašovaná algoritmem argon2 a uživatelé se stejným heslem mají odlišný haš.

Jsem tedy přesvědčen, že API je zabezpečena.

Bibliografie

1. BREHMER, M.; LEE, B.; BACH, B.; RICHE, N. H.; MUNZNER, T. Timelines Revisited: A Design Space and Considerations for Expressive Storytelling. *IEEE Transactions on Visualization and Computer Graphics*. 2017, roč. 23, č. 9, s. 2151–2164. Dostupné z DOI: 10.1109/TVCG.2016.2614803.
2. FULDA, J.; BREHMER, M.; MUNZNER, T. TimeLineCurator: Interactive Authoring of Visual Timelines from Unstructured Text. *IEEE Transactions on Visualization and Computer Graphics*. 2016, roč. 22, č. 1, s. 300–309. Dostupné z DOI: 10.1109/TVCG.2015.2467531.
3. PRO PUBLICA INC. *TimelineSetter* [online]. 2011 [cit. 2021-03-22]. Dostupné z: http://propublica.github.io/timeline-setter/.
4. WEBALON LTD. *Tiki-Toki Timeline Maker* [online]. 2021 [cit. 2021-03-23]. Dostupné z: https://www.tiki-toki.com.
5. TIMETOAST TIMELINES. *Timetoast timeline maker* [online]. 2021 [cit. 2021-03-23]. Dostupné z: https://www.timetoast.com.
6. THE WORLD ANVIL TEAM. *World Anvil* [online]. 2020 [cit. 2021-03-27]. Dostupné z: https://www.worldanvil.com.
7. WIKIMEDIA FOUNDATION INC. *Wikipedia: The free encyclopedia* [online]. 2004 [cit. 2021-03-27]. Dostupné z: https://www.wikipedia.org.
8. WALTERS, Abraham. *Canvas vs. DOM vs. WebGL vs. ImpactJS vs. PixiJS Particle Test* [online]. 2013 [cit. 2021-04-18]. Dostupné z: https://github.com/quidmonkey/particle_test.
9. THREE.JS AUTHORS. *three.js* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://threejs.org.
10. CATUHE, David. *Babylon.js* [online] [cit. 2021-04-03]. Dostupné z: https://www.babylonjs.com.
11. TAVARES, Gregg. *TWGL: A Tiny WebGL helper Library* [online]. 2019 [cit. 2021-04-03]. Dostupné z: https://twgljs.org.
12. KITWARE INC. *vtk.js* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://kitware.github.io/vtk-js/.
13. PLAYCANVAS LTD. *PlayCanvas* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://playcanvas.com.
14. PHOTON STORM LTD. *Phaser - HTML5 Game Framework* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://phaser.io.

15. GOODBOY DIGITAL LTD. *PixiJS* [online]. 2021 [cit. 2021-04-03]. Dostupné z: https://www.pixijs.com.

16. PROVOS, Niels; MAZIERES, David. A Future-Adaptable Password Scheme. *Proceedings of 1999 USENIX Annual Technical Conference* [online]. 1999, s. 81–92 [cit. 2021-04-22]. Dostupné z: https://archive.org/details/1999-proceedings-freenix-track-atc-monterey/page/81/mode/2up.

17. AUMASSON, Jean-Philippe. *The Password Hashing Competition* [online]. 2019 [cit. 2021-04-22]. Dostupné z: https://www.password-hashing.net/.

18. 1&1 IONOS INC. *Rainbow tables: Simply explained* [online]. 2021 [cit. 2021-06-12]. Dostupné z: https://www.ionos.com/digitalguide/server/security/rainbow-tables/.

19. AUTH0 INC. *Adding Salt to Hashing: A Better Way to Store Passwords* [online]. 2021 [cit. 2021-06-12]. Dostupné z: https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/.

20. MOZILLA; CONTRIBUTORS, individual. *Using HTTP cookies* [online]. 2021 [cit. 2021-04-27]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies.

21. MOZILLA; CONTRIBUTORS, individual. *Set-Cookie* [online]. 2021 [cit. 2021-04-27]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie.

22. MOZILLA; CONTRIBUTORS, individual. *Types of attacks* [online]. 2021 [cit. 2021-04-27]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Security/Types_of_attacks.

23. NETSPARKER LTD. *Using the Same-Site Cookie Attribute to Prevent CSRF Attacks* [online]. 2021 [cit. 2021-06-12]. Dostupné z: https://www.netsparker.com/blog/web-security/same-site-cookie-attribute-prevent-cross-site-request-forgery/.

24. NGINX, INC. *NGINX* [online]. 2021 [cit. 2021-06-12]. Dostupné z: http://nginx.org/en/.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `thesis.pdf` ..... dokument ve formátu PDF
- `src`
  - `api` ..... zdrojové kódy implementace API (JS)
  - `db` ..... zdrojové kódy implementace databáze (SQL)
  - `data` ..... zdrojové kódy pro vygenerování insertscriptu (CSV, CPP, JS)
  - `tex` ..... zdrojová forma dokumentu (LaTeX)