



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



**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**  
**ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ**

FACULTY OF INFORMATION TECHNOLOGY  
DEPARTMENT OF COMPUTER SYSTEMS

# **VIZUALIZACE SÍŤOVÝCH BEZPEČNOSTNÍCH UDÁLOSTÍ**

VISUALIZATION OF NETWORK SECURITY EVENTS

**BAKALÁŘSKÁ PRÁCE**  
BACHELOR'S THESIS

**AUTOR PRÁCE**  
AUTHOR

**PETR STEHLÍK**

**VEDOUCÍ PRÁCE**  
SUPERVISOR

**Ing. PAVEL KROBOT**

BRNO 2016

## **Abstrakt**

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém jazyce.

## **Abstract**

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

## **Klíčová slova**

Sem budou zapsána jednotlivá klíčová slova v českém jazyce, oddělená čárkami.

## **Keywords**

Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

## **Citace**

Petr Stehlík: Vizualizace síťových bezpečnostních událostí, bakalářská práce, Brno, FIT VUT v Brně, 2016

# Vizualizace síťových bezpečnostních událostí

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Pavla Krobota. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Petr Stehlík

1. května 2016

## Poděkování

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant, apod.).

© Petr Stehlík, 2016.

*Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.*

# Obsah

<b>1 Úvod</b>	<b>3</b>
<b>2 Monitorování sítě</b>	<b>4</b>
2.1 NEMEA . . . . .	4
2.1.1 Modul . . . . .	5
2.1.2 Rozhraní . . . . .	6
2.2 IDEA . . . . .	6
2.3 Další monitorovací systémy . . . . .	7
2.4 Shrnutí . . . . .	8
<b>3 Technologie</b>	<b>9</b>
3.1 Uživatelská část . . . . .	9
3.1.1 JavaScript MVC frameworky . . . . .	10
3.1.2 HTML/CSS frameworky . . . . .	13
3.2 Serverová část . . . . .	14
3.2.1 REST API . . . . .	14
3.3 Vybrané technologie . . . . .	16
3.3.1 Uživatelská část . . . . .	16
3.3.2 Serverová část . . . . .	17
3.4 Shrnutí . . . . .	18
<b>4 Architektura aplikace</b>	<b>19</b>
4.1 Případy užití . . . . .	20
4.2 Uživatelská část . . . . .	21
4.3 REST API . . . . .	22
4.4 GUI . . . . .	24
4.5 Shrnutí . . . . .	25
<b>5 Implementace</b>	<b>26</b>
5.1 Serverová část . . . . .	26
5.2 Uživatelská část . . . . .	28
5.3 Zabezpečení . . . . .	32
5.4 Distribuce . . . . .	32
<b>6 Dosažené výsledky</b>	<b>33</b>
<b>7 Závěr</b>	<b>34</b>
<b>Literatura</b>	<b>35</b>

<b>Přílohy</b>	<b>37</b>
Seznam příloh . . . . .	38
<b>A Drátěné modely</b>	<b>39</b>
<b>B Seznam použitých Python knihoven</b>	<b>42</b>
<b>C Obsah CD</b>	<b>43</b>

# Kapitola 1

## Úvod

Počítačové sítě, zejména Internet, v dnešním světě zaujímají jednu z nejvýznamnějších rolí. Počínaje výzkumem a vědeckými experimenty, konče běžným životem většiny lidí. Jen za posledních deset let se počet uživatelů Internetu více než ztrojnásobil z přibližně jedné miliardy na tři miliardy uživatelů. Počítačové sítě propoují celý svět a jsou neustále rozšiřovány, vylepšovány a modernizovány. To vede k větším nárokům na použité technologie a zdroje.

Avšak se zvyšujícím počtem uživatelů roste i počet útoků na různé počítačové sítě, kterými se útočníci snaží získat informace či poškodit oběť. Síťový útok je podle [25] definován jako záměrný akt, kde se entita snaží překonat bezpečnostní služby a porušit bezpečnost systému. Současně s tím vznikají systémy na detekci takovýchto útoků, aby správci sítí dokázali včas a efektivně reagovat na vzniklou situaci.

Jeden z těchto systémů vznikl ve sdružení CESNET s názvem NEMEA (Network Measurements Analysis). Tento systém mj. slouží pro analýzu síťového provozu a detekci neobvyklých událostí na síti. Podezřelé toky jako agregované události pak může systém zaznamenávat do databáze. Na větší síti (stovky až tisíce připojených zařízení) je takovýchto událostí vytvořeno až několik tisíc denně. S tím nastává problém jak dané události jednoduše analyzovat a rozpoznat důležité události, na které se zaměřit a na které nebrát zřetel.

Pro analýzu velkého množství dat je efektivní vizualizace dle vhodných metrik, které vyplývají z dostupných dat. Cílem této bakalářské práce je vytvořit aplikaci pro vizuální analýzu bezpečnostních událostí na síti primárně monitorované systémem NEMEA, nicméně díky IDEA formátu dat bude aplikace přenositelná na další systémy.

Důležitým aspektem vytvořené aplikace je důraz na použití moderních nástrojů podporující tvorbu dynamických webových aplikací. Společně s tím je kladen důraz na uživatelskou přívětivost a jednoduchost prostředí, ve kterém bude probíhat vizuální analýza událostí.

Celou aplikaci navíc bude možno přizpůsobit potřebám daného správce sítě. V aplikaci bude zavedena technika *drill-down*, která napomáhá rychlé a přehledné analýze velkého množství dat bez ztráty informací o analyzované události. Drill-down spočívá v postupném zvyšování rozlišení dat, která analyzujeme a postupujeme směrem shora dolů.

Aplikace bude pracovat s formátem dat nazvaný IDEA. Tento formát je specifikován sdružením CESNET a slouží jako prostředek pro sdílení dat bezpečnostních událostí mezi různými systémy. Díky tomu lze aplikaci kdykoliv přenést na jiný zdroj databáze než je systém NEMEA, např. v rámci sdružení CESNET na systém Warden nebo Mentat.

Aplikace bude integrována do současného NEMEA systému pod názvem NEMEA Dashboard a bude s ním společně distribuována jako front end celého systému.

## Kapitola 2

# Monitorování sítě

V rozlehlejších sítích jako je např. páteřní či firemní síť je téměř nutností monitorovat a analyzovat provoz na síti, abychom byli informováni o jejím aktuálním stavu, vytížení a zejména negativních vlivech na monitorovanou síť. Samozřejmě i sítě menšího rozsahu by měly být monitorované. Pokud se v malé firmě podaří útočníkovi infiltrovat síť, výsledky útoku mohou být pro firmu likvidační.

Systém pro odhalení průniku (anglicky „Intrusion Detection System“, zkráceně IDS)[21] je takový systém, který analyzuje a identifikuje ze zachyceného provozu podezřelé události. Tyto události může IDS dále klasifikovat. IDS může být dvojího typu.

Prvním typem je detekce anomálií, který má výhodu v možnosti detekce jak známých, tak neznámých útoků. Nevýhodou je, že často může označit normální provoz za útok. Tomu se snaží předejít učením a tvorbou datové sady pro rozpoznání škodlivého provozu.

Druhým typem IDS je detekce založená na pravidlech. Tyto pravidla jsou pevně daná a systém pouze porovnává síťový provoz s danými pravidly. Nevýhodou takového systému je neschopnost detekce neznámých útoků. Některé IDS kombinují oba přístupy a tvoří tak hybridní IDS, který je založen jak na pravidlech, tak na automatické detekci anomálií.

Systém prevence průniku (anglicky „Intrusion Prevention System“, neboli IPS)[21] je narozdíl od IDS aktivním prvkem v počítačové síti. Nejen že detekuje útoky na síť, ale navíc je aktivně blokuje, případně odklání na speciální uzel v síti pro hlubší analýzu útoku.

### 2.1 NEMEA

Network Measurements Analysis, zkráceně NEMEA, je systém, který umožňuje vytvořit komplexní nástroj pro automatizovanou analýzu toků získaných ze síťového monitoringu v reálném čase. Systém NEMEA je zejména monitorovacím nástrojem, ale slouží také jako IDS.

Systém se skládá z oddělených stavebních bloků nazývané moduly. Tyto jednotlivé moduly jsou následně propojeny pomocí rozhraní TRAP.

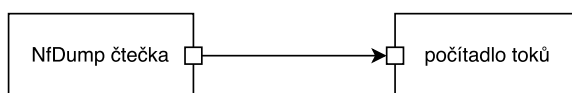
Moduly jsou nezávislé pracovní jednotky, které obecně přijímají proud dat na svých vstupech, zpracují či zanalyzují daná data a následně je odešlou ze svých výstupních rozhraní jako proud dat pro další moduly. Modul může například tvořit statistiky o přijatých datech a na základě těchto statistik detekovat určité typy síťového útoku. Detekovaný útok je popsán datovým záznamem, který je odeslán přes výstupní rozhraní dalším modulům, které s daným záznamem dále pracují, např. jej uloží v IDEA formátu (viz sekce 2.2) do databáze nebo ze získaných statistik dokáží detekovat anomálie v síťovém provozu a dokáží tak

jednotlivé pokusy od jednoho útočníka agregovat a zpracovat jej jako jediný útok skládající se z několika desítek až stovek pokusů o útok v delším časovém intervalu a administrátor sítě by je snadno přehlédl nebo ignoroval, pokud by nebyly agregované.

Jednotlivé moduly se většinou zaměřují na konkrétní typ událostí, případně činnosti. Jejich spojením je možné získat nástroj pro komplexní detekci a analýzu síťových dat schopný detekovat a identifikovat útoky na monitorovanou síť, který následně detekované události uloží do databáze a webová aplikace, kterou v této práci navrhuji, uložené útoky zobrazí.

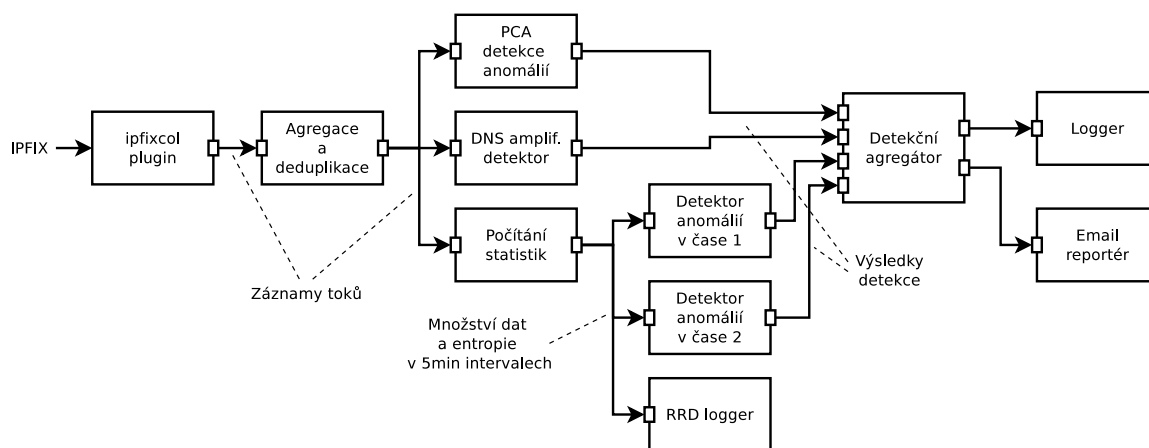
NEMEA je také schopná přístupu „store-and-ex-post“, který lze vidět na obrázku 2.1. Jsou zde dva moduly spojené jedním rozhraním. První modul čte záznamy toků ze souboru a druhý modul počítá statistiky o těchto přečtených tocích. Tento přístup je charakteristický tím jak nakládá se síťovými daty. Ty prvně uloží a až poté začíná systém s analýzou uložených síťových dat.

Oproti proudovému zpracování síťových dat je tento systém náročnější na prostor pro uložení dat, ale analýza je přesnější a méně náročná na výkon stroje, zejména v případě pokročilejší analýzy. Pokud bychom vykonávali proudové zpracování dat, je zde i velká náročnost na operační paměť, pokud chceme analyzovat velké časové rámce.



Obrázek 2.1: Minimální příklad NEMEA systému obsahující pouze dva moduly.

Z takovýchto základních bloků lze postavit i velmi komplexní systém jak je vidět na obrázku 2.2, kde jsou data přijímána v reálném čase z IPFIX[29] kolektoru. Data jsou předzpracována, analyzována několika algoritmy a následně jsou nahlášeny detekované události. Každá z těchto úloh je jeden modul, který může být znovu použitý na několika různých místech.



Obrázek 2.2: Příklad propojení modulů NEMEA systému, který shromažďuje síťová data, detekuje anomálie a útoky a následně události ukládá a reportuje.

### 2.1.1 Modul

Každý modul je samostatný program nezávislý na ostatních. To sice zvyšuje nároky na systém, ale dovoluje větší variabilitu při návrhu modulu. Ten je, díky tomuto návrhu, možno



naprogramovat v libovolném jazyce, sledovat a řídit spotřebu zdrojů každého modulu zvlášť a zejména v případě nefungujícího modulu se celý systém NEMEA dokáže zotavit z chyby naprosto bez problémů. Modulární systém navíc dovoluje přidávat a odebírat jednotlivé moduly za běhu, což je v produkčním prostředí jedna ze základních vlastností kvalitního monitorovacího systému.

Při zapojení a startu nového modulu se modul periodicky snaží připojit na definované rozhraní. Pokusy o spojení jsou sledovány NEMEA Supervisorem, kterým lze spravovat všechny moduly v systému a pokud se modulu nepodaří po několika pokusech připojit na rozhraní, je modul ukončen.

### 2.1.2 Rozhraní

Všechny rozhraní jsou výhradně jednocestná a přenos dat je realizován formou jednotlivých záznamů. Všechny záznamy poslané přes jedno konkrétní rozhraní mají vždy stejný formát, nicméně mezi rozhraními se formát může lišit.

Protokol pro dynamickou tvorbu formátu je nazván UniRec. Ten specifikuje nejen formát záznamu, ale také jak záznam vytvořit a jak zpracovat – implementuje formát pro binární reprezentaci síťového záznamu.

Pro tvorbu rozhraní je vytvořena sdílená knihovna libtrap, která využívá Traffic Analysis Platform (zkráceně TRAP) pro komunikaci mezi různými rozhraními a tyto rozhraní využívají protokolu UniRec.

## 2.2 IDEA

Pro potřebu sdílení informací o síťových událostech mezi různými skupinami a zařízeními (např. honeypoty, analyzéry systémových zpráv, analyzéry provozu na síti, netflow sondy a další) existuje několik formátů záznamu pro takovéto události. Nicméně žádný z nich není natolik univerzální, aby byl vždy a všude použitelný a pokud se k takovému formátu blíží, tak není natolik detailní, aby pokryl všechny důležité informace.

IDEA, neboli Intrusion Detection Extensible Alert, je formát záznamu síťové události specifikovaný sdružením CESNET. IDEA si klade za cíl specifikovat takový formát záznamu, který je univerzální, přenositelný, ale zároveň dost konkrétní a snadno pochopitelný bez potřeby rozsáhlé dokumentace k jednotlivým polím.

Vzorový záznam generovaný systémem NEMEA je vyobrazený ve výpisu 2.1. Jak je vidět, formát je specifikovaný jako JSON dokument, aby byl přehledný, čitelný v běžné podobě (narozdíl od binárních formátů), lehce přenositelný a efektivní (např. oproti XML[20]).

```

{
  "Format" :      "IDEA0",
  "ID" :         "73e0b136-aeb8-4aae-bb80-9bfb4f258847",
  "Category" :    [ "Availability.DDoS" ],
  "Description" : "DNS amplification",
  "EventTime" :   "2016-04-07T22:19:25Z",
  "CreateTime" :  "2016-04-07T22:34:52Z",
  "CeaseTime" :   "2016-04-07T22:34:38Z",
  "DetectTime" :  "2016-04-07T22:34:38Z",
  "PacketCount" : 393,
  "Source" : [ {
    "IP4" : [ "192.1.0.201" ],
    "Proto" : [ "udp", "dns" ],
    "OutPacketCount" : 393,
    "InPacketCount" : 767
  } ],
  "Target" : [ {
    "Proto" : [ "udp", "dns" ],
    "IP4" : [ "10.0.0.135" ],
    "InPacketCount" : 393
  } ],
  "Node" : [ {
    "SW" : [ "NEMEA", "amplification_detection" ],
    "Name" : "cz.cesnet.nemea.amplification_detection"
  } ],
  "Type" : [ "Flow", "Statistical" ],
}

```

Listing 2.1: Vzorový IDEA záznam ze systému NEMEA. Některé části byly vynechány nebo zkráceny a IP adresy změněny na lokální.

## 2.3 Další monitorovací systémy

Na trhu jsou v současné době různá dostupná řešení pro detekci a vizualizaci síťových bezpečnostních událostí, nicméně valná hromada z nich je komerční a hlavně vázaná na konkrétní hardware od daného výrobce. Klient tudíž většinou nekupuje software, ale hardware s přiloženým software.

Komerčně dostupný produkt je např. Flowmon[2] ADS[11] od stejnojmenné společnosti. Flowmon je spin-off společností z projektu Liberouter ze sdružení CESNET. Jejich sondy a kolektory využívají technologie vytvořené ve sdružení CESNET jak z hlediska hardware, tak software. Dalším komerčním řešením je Cisco Secure IDS[1], dříve známý jako Cisco NetRanger.

Open source projekty jako NEMEA jsou dostupné mnoho let, ale pouze několik z nich dosáhlo znatelnějšího rozšíření v komunitě síťových správců. Nejvýznamnějšími jsou systémy Snort[24], VERMONT[16] a framework Bro[22].

### Snort

Tento open-source projekt, od roku 2013 vlastněn firmou Cisco[3], je možno konfigurovat ve 3 hlavních režimech[4]: „sniffer“, paket logger a jako (N)IDS. V režimu IDS Snort pracuje

principiálně velmi podobně jako systém NEMEA. Zachytává síťový provoz, ukládá si důležité informace o něm a analyzuje jej. Ve výsledku ukládá záznamy o síťových událostech. Nicméně Snort není modulárním systémem a tudíž není tak flexibilní a není stavěný na vysokorychlostní rozsáhlé síti jako systém NEMEA.

## VERMONT

VERMONT (Versatile Monitoring Toolkit) je modulární monitorovací systém obsahující IPFIX kolektor, exportér, analyzátor a další moduly a grafické prostředí pro vizuální analýzu dat. VERMONT byl vyvinut v rámci projektu HISTORY[5] a evropským projektem DIADEM firewall[19]. Svou architekturou je nejbližší systému NEMEA, protože je částečně modulární. Systém NEMEA je oproti tomu modulární od samotného jádra systému, což dovoluje vyšší flexibilitu při vývoji a menší závislost na použitých technologiích.

## Bro

Dalším v komunitě rozšířeným řešením je framework Bro. Tento framework primárně určený pro síťovou analýzu není podobný systému NEMEA, ani předchozím systémům, protože je to spíše nástroj pro vytváření (N)IDS než-li ucelený systém. Bro se velmi blíží skriptovacímu jazyku (např. Perl) nebo unixovým nástrojům jako tcpdump nebo nfdump. Bro lze rozdělit na dvě vrstvy. První vrstvou je „Bro Event engine“, který analyzuje síťový provoz a generuje neutrální síťové události v podobě „byla vytvořena nějaká událost“.

Tyto neurčité události jsou následně analyzovány druhou vrstvou – „Bro Policy skripty“. V této vrstvě je naimplementovaný zmiňovaný skriptovací jazyk. V současné době existuje mnoho naprogramovaných skriptů, které jsou připraveny k okamžitému použití, včetně pokročilé analýzy síťového provozu.

V ranných fázích vývoje se systém NEMEA velmi blížil frameworku Bro, s vývojem času se ale NEMEA stala uceleným systémem připraveným k okamžitému nasazení na měřicí body.

## 2.4 Shrnutí

V této kapitole jsme prezentovali systém NEMEA a jeho architekturu. Popsali jsme jeho nejdůležitější části, zejména jak vypadá modul a jeho komunikační protokol. Dále jsme popsali formát záznamu síťové bezpečnostní události IDEA, který je opěrným bodem pro ukládání dat v systému NEMEA v rámci analýzy událostí koncovým uživatelem. V poslední sekci jsme porovnali systém NEMEA s dalšími veřejně dostupnými monitorovacími systémy.

## Kapitola 3

# Technologie

Vizualizace síťových bezpečnostních událostí může být vytvořena několika postupy. Pokud máme IDS, který je dostupný pouze z jednoho stroje, nejčastěji zvolíme tvorbu desktopové aplikace, protože máme jistotu provozního prostředí jako je např. operační systém, dostupné balíčky a jejich verze, atd. V případě vzdálené správy IDS (častější případ) jsme nejčastěji odkázáni na vzdálený přístup pomocí příkazové řádky. Tento přístup je bohužel velmi limitovaný a nelze jej využít pro vizualizaci. V současné době je pro vzdálenou správu nejvhodnější vytvořit webovou aplikaci, která je dostupná z Internetu a použitelná na většině dnes používaných zařízeních<sup>1</sup>

Pro tvorbu moderních webové aplikace je na Internetu dostupná celá řada knihoven, frameworků a systémů. Nicméně první otázkou zůstává co taková moderní webová aplikace je?

Ustáleným pojmem pro moderní webovou aplikaci je z anglického single page application[18] zkratka „SPA“. Tato zkratka je používána komunitou vývojářů těchto typů aplikací a lze se s ní setkat zejména na stránkách typu Stack Overflow.

Specifikem SPA je její vysoká interaktivita s uživatelem, vysoká dostupnost služby, kterou poskytuje a rozdělení na dva logické celky. Uživatelskou a serverovou část, často chybně nazývané frontend a backend aplikace. Více o architektuře aplikace v kapitole 4.

Většinu z těchto kvalit získává SPA díky způsobu jakým je doručována uživateli. SPA je při prvním načtení stránky v prohlížeči celá uložena v rámci vyrovnávací paměti prohlížeče a během celé doby používání SPA není nutné stránku znova načítat. SPA můžeme chápat jako univerzálního klienta pro obsluhu dané aplikace.

SPA je nejčastěji tvořena pomocí jazyka JavaScript. Ten umožňuje dynamickou změnu stránky bez nutnosti ji znova načítat a tím pádem uživatel nepřichází o dočasná data na stránce. Tento přístup navíc redukuje počet dotazů na server a snižuje tak jeho zátěž.

S přesunem logiky na uživatelskou část SPA jsou ale spojeny nemalé problémy. Zejména pak různorodá interpretace kódu. To se v posledních letech téměř eliminovalo díky moderním prohlížečům a jejich jádrům jako je např. WebKit pro Google Chrome nebo Gecko pro Firefox.

### 3.1 Uživatelská část

K vytvoření SPA a zejména uživatelské části existuje několik významných systémů. V této části si tyto technologie představíme a porovnáme mezi sebou. Nejdůležitější částí je Ja-

---

<sup>1</sup>Jedná se zejména o stolní počítače, notebooky, tablety a chytré mobilní telefony.

vaScriptová knihovna/framework, který bude nejčastěji interagovat s uživatelem. Další nedílnou součástí je knihovna pro HTML/CSS, která bude definovat vzhled aplikace.

### 3.1.1 JavaScript MVC frameworky

Bylo vybráno 5 JavaScript frameworků, které napomáhají k tvorbě SPA. Architektura MVC rozděluje aplikaci na tři celky, které mezi sebou navzájem komunikují. Více informací o MVC architektuře v kapitole 4.

Kritéria užšího výběru frameworků byla zejména následující:

- open-source projekt s MIT nebo BSD licencí,
- jednoduché použití,
- relativně nenáročný na výpočetní výkon hostitelského stroje,
- široká podpora mezi prohlížeči,
- projekt má určitou historii a je udržován některou ze známých společností (udržitelnost vývoje).

#### React[13]

Framework React je vyvíjen pod hlavičkou společnosti Facebook. Původním cílem návrhářů Reactu bylo vyřešit problémy během vývoje komplexních uživatelských rozhraní s rychle měnícími se daty. Dalším cílem bylo vytvořit takový framework, který lze distribuovat v takovém měřítku jako je Facebook.

React je framework pro uživatelskou část SPA používající tradiční MVC architekturu. Taková architektura je nejvíce znatelná v případě použití obousměrného vázání dat (anglicky two-way data binding), více v sekci 4.2.

React se zaměřuje zejména na tvorbu UI, z pohledu architektury pohledová část (view), a kvůli tomu se ostatním částem architektury nevěnuje tak detailně, jak by většinu času vývojář potřeboval. Velmi častým řešením je použít React na pohledovou část a na model a kontrolér použít jiný framework, např. AngularJS.

React pracuje s moderními přístupy k vývoji SPA a boří tak mnoho zažitých technik jak takovou PSA vyvíjet. Místo běžné manipulace s DOM elementy si React, případně vývojář, definuje vlastní DOM elementy, kterým přidává vlastnosti, váže na ně data a manipuluje s nimi bez větších výkonových ztrát. Toho docílil zejména použitím tzv. shadow DOM[30], který je ale novinkou na poli prohlížečů a je podporován pouze jádrem WebKit.

```
var HelloMessage = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
});

ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

Listing 3.1: „Hello World“ příklad v jazyku JSX, který do stránky vykreslí text „Hello John“.

Pro názornou ukázkou, jak framework React funguje, je ve výpise 3.1 vytvořen jednoduchý „Hello World“ element v JSX[7] syntaxi. Tento kód demonstruje pouze minimální možnosti frameworku, ale vypovídá o jednoduchosti použití Reactu.

## Ember.js

Ember.js je primárně zaměřen na tvorbu SPA. Toho dosahuje zejména používáním zažitých praktik v rámci komunity vývojářů, širokou funkcionalitou bez nutnosti instalace dalších doplňků a osvědčil se i vysokou stabilitou během celé doby vývoje. Ember je také vždy jedním z prvních frameworků, který implementuje novinky obsažené v nových jádrech webových prohlížečů jako jsou např. JavaScript Promises[12], Web Components[32] nebo ES6 syntaxe.

Ember se skládá celkem z pěti klíčových konceptů. Jejich názvy jsou ponechány anglicky pro větší názornost.

- **Routes** – každý stav aplikace je reprezentován unikátní cestou a této cestě náleží i odpovídající objekt, který manipuluje s danou cestou.
- **Models** – každá cesta má svůj korespondující model, který obsahuje data asociována s danou cestou a stavem aplikace. Model slouží zejména k manipulaci s daty.
- **Templates** – šablony jsou tvořeny v HTML s použitím šablonovacího jazyka HTMLBars[6].
- **Components** – komponenta je vlastní HTML značka definovaná programátorem aplikace. Její chování je implementováno pomocí JavaScriptu a její vzhled pomocí HTMLBars šablon. Komponenta se chová jako běžný DOM element.
- **Services** – servis je singleton objekt, který v sobě udržuje dlouhodobá data během používání aplikace klientem.

## AngularJS

AngularJS je jedním z nejdéle existujících frameworků pro tvorbu SPA. Je udržován společností Google a poskytuje ucelené prostředí k vývoji komplexních SPA. AngularJS poskytuje dvě metodiky návrhu architektury aplikace. MVC a MVVM (model-view-viewmodel)[26].

MVVM je rozdílný zejména ve způsobu nakládání s daty v aplikaci. Viewmodel pouze reflektuje změny modelu v pohledu a naopak. Viewmodel lze chápat jako interpret dat získaných z modelu, které jsou požadovány v pohledu.

Vzhledem k rozsáhlosti frameworku se nevyhneme tvorbě aplikace, která prolíná oba typy architektury. AngularJS je ale zejména stavěn na základech MVC architektury.

AngularJS staví na několika elementárních pilířích:

- rozdělit manipulaci DOM od aplikační logiky,
- rozdělit klientskou a serverovou stranu aplikace,
- poskytnout strukturu pro vývoj SPA.

Manipulace s DOM je zajišťována v pohledové části architektury. Aplikační logika je v tomto případě oddělena a to dovoluje simultánní tvorbu na obou částech aplikace. To dovoluje strukturovat kód do uzavřených logických celků, které mohou být znovu použity v dalších částech aplikace.

Rozdělením klientské a serverové strany aplikace získává AngularJS celkovou kontrolu nad klientskou stranou a dokáže tak abstrahovat jednotlivé vrstvy aplikace. To dovoluje hlubokou integraci MVC architektury do klientské části.

Toto rozdělení dovoluje frameworku implementovat takové možnosti jako je např. vkládání závislostí nebo pohledově závislé kontroléry.

To vše spěje ke snížení zátěže serveru, na kterém je SPA uložena. Většina, ne-li všechna, aplikační logiky totiž probíhá na klientské straně a server pro svou práci vůbec nepotřebuje. Ten je potřeba až v momentě kdy chceme získat nebo uložit data pro dlouhodobé užití.

## Backbone.js

Framework Backbone.js se mírně odlišuje od předešlých a to svou architekturou, která není MVC, ale MVP (model-view-presenter). Presenter je v tomto modelu prostředníkem, který plní logickou funkci a spojuje pohled s modelem. Veškeré akce, které se provedou v pohledu (např. kliknutí na tlačítko) jsou delegovány presenteru. Ten je navíc oddělen od pohledu a komunikuje s ním pouze přes definované rozhraní.

Tento přístup je vhodný zejména pro testování a jasné oddělení jednotlivých částí (zvláště modelu od pohledu). Avšak MVP je náročnější na tvorbu kvůli tomu, že všechny datové vazby musí vytvořit sám programátor.

Výhodou Backbone.js je jeho velikost (celková velikost distribuce je méně než 8 kB) a nezávislost na dalších knihovnách (vyžaduje pouze jednu další knihovnu<sup>2</sup>).

Z předcházejících faktů lze vyvodit, že Backbone.js je vhodný zejména pro SPA menšího rozsahu, které jsou zaměřeny na velmi konkrétní úkol. To může v pozdějších fázích vývoje SPA znamenat problémy a kompromisy při vývoji další funkcionality aplikace.

## Shrnutí

Pro rychlý a stručný přehled byla vypracována tabulka shrnující klíčové vlastnosti každého porovnávaného frameworku.

Tabulka 3.1: Porovnání klíčových vlastností JavaScript frameworků

Název	Verze <sup>3</sup>	Aktivní vývoj	Velikost <sup>4</sup>	Licence
React	15.0.1	3 roky (2013)	142 kB	BSD
Ember.js	2.5.0	4 roky (2011)	450 kB	MIT
Backbone.js	1.3.3	5 let (2010)	7.5 kB	MIT
AngularJS	1.5.3	5 let (2010)	152 kB	MIT

<sup>2</sup>Underscore.js, pokud chce programátor využít i složité funkce, musí navíc doplnit knihovnu jQuery.

<sup>3</sup>Aktuální verze v době psaní práce.

<sup>4</sup>Velikost minifikovaného produkčního kódu.

### 3.1.2 HTML/CSS frameworky

Důležitým aspektem SPA je také její vzhled, nebo spíše UX (user-experience)[8]. UX je zejména vnímání a reakce osoby, které plyne z používání nebo předpokládaného užití/chování výrobku, systému nebo služby. Důraz se musí klást zejména na předpokládané chování systému, tím pádem může uživatel procházet plynule aplikací, aniž by musel odhadovat chování té části aplikace, ve které se právě nachází.

Toho lze dosáhnout zejména uceleností vzhledu a chování jednotlivých komponent SPA. Pro tento účel existují tzv. HTML/CSS knihovny. Na Internetu je jich velké množství a výběr jsem zúžil pouze na 3 knihovny, které jsou známé; mají širokou škálu komponent, které lze v SPA využít a jsou open-source.

Poslední podmínkou jsou zdrojové kódy CSS v jednom z CSS preprocesorů[17] – LESS<sup>5</sup> nebo SASS<sup>6</sup>, aby v aplikaci byla zaručena konzistence proměnných (např. barvy, velikost, odsazení).

#### Bootstrap

Bootstrap je nejpopulárnějším frameworkem pro tvorbu vzhledu SPA. Obsahuje HTML a CSS komponenty pro ucelenou typografii, formuláře, tlačítka, navigaci a další. Bootstrap také nabízí komponenty, které jsou částečně nebo zcela napsány v JavaScriptu.

Bootstrap je udržován zejména společností Twitter, kde také vznikl na popud tvorby interních firemních aplikací s uceleným vzhledem a také UX, aby se ušetřily náklady na údržbu takovýchto aplikací.

Díky Bootstrapu se velmi rychle ujal trend responzivního designu a mobile-first přístupu. Navíc disponuje velkou komunitou vývojářů a časté problémy jsou tak mnohokrát vyřešeny.

#### Foundation

Foundation je velmi podobný framework jako Bootstrap, také nabízí HTML a CSS komponenty pro různé části SPA včetně JavaScript komponent. Nemá sice za sebou takovou společnost jakou je Twitter, ale je neméně kvalitní a udržovaný.

Hlavním rozdílem oproti Bootstrapu je jeho upravitelnost. Foundation dává větší prostor pro úpravu jednotlivých komponent.

Navíc disponuje styly, které jsou automaticky aplikované na dané HTML elementy – vývojář nepotřebuje přidávat množství tříd ke každému elementu. To však nemusí každému vyhovovat a mohou nastat neobvyklé konflikty stylů během vývoje.

#### Angular Material

Angular Material není běžným CSS frameworkem, nýbrž je velmi úzce spjat s JavaScript frameworkem AngularJS a využívá jeho mnoha možností jako jsou např. vlastní HTML elementy (direktivy).

Framework také obsahuje mnoho HTML a CSS komponent, ale ty jsou vždy velmi úzce spjaty s AngularJS funkcionalitou jako je např. two-way data binding a již zmíněné direktivy.

Společnost Google disponuje vizuálním jazykem pro všechny své produkty nazvaný Material Design<sup>7</sup>. Ten je založený na tzv. „kartičkách“, které lze skládat na sebe, řadit, orga-

---

<sup>5</sup><http://lesscss.org>

<sup>6</sup><http://sass-lang.com>

<sup>7</sup><https://www.google.com/design/spec/material-design/introduction.html>



nizovat a upravovat.

Jak již název frameworku napovídá je tento vizuální jazyk použit skrz všechny dostupné komponenty.

Z technického hlediska je Angular Material nejpokročilejším frameworkem. Pro mřížkový systém<sup>8</sup> používá flexbox[31], který je dostupný pouze v nejnovější specifikaci CSS 3 a moderních webových prohlížečích. To může způsobit vážné problémy, pokud je SPA cílena na široké publikum.

## 3.2 Serverová část

V rámci serverové části je kladen důraz zejména na výkonnost, stabilitu a efektivitu programu. Tyto faktory jsou ovlivněny zejména použitým programovacím jazykem, knihovnami a návrhem architektury programu.

Jediný požadavek na server je, aby byl typu \*nix. To nám zaručuje určité faktory a vlastnosti provozního prostředí jako např. dostupné jazyky a jejich knihovny, podpůrné programy a další.

Na serveru poběží z NEMEA Dashboard pouze REST API[10]. Jeho vlastnosti a architektura budou detailně popsány v kapitole 4. API se připojuje na NoSQL databázový systém MongoDB. Ta je využívána systémem NEMEA, který do ní ukládá detekované události. MongoDB bylo zvoleno kvůli rychlosti, jednoduchosti použití a pro záznamy ve formátu IDEA bylo potřeba NoSQL databázový systém, který dokáže pracovat s daty stejně nebo podobně jako se pracuje s JSON.

### 3.2.1 REST API

REST, neboli Representational State Transfer, je architektura rozhraní pro distribuci dat. Tato architektura nám dovoluje přistupovat ke všem zdrojům uniformním rozhraním, které definuje čtyři základní operace nad každým z nich:

**Create** – vytvoření a uložení nového datového objektu.

**Read** – čtení datového objektu.

**Update** – permanentní aktualizace datového objektu.

**Delete** – odstranění datového objektu z permanentního datového prostoru.

Toto rozhraní zpřístupňuje databázi událostí vytvořenou NEMEA systémem, která je udržována v MongoDB. Navíc tyto data agreguje, předzpracovává či jinak upravuje (funkcionalita REST API je popsána v sekci 4.3)

Pro vytvoření API s REST architekturou rozhraní existují knihovny, které vývoj ulehčují a zrychlují. Ve většině případů ale platí, že takovéto knihovny jsou náročné na zdroje a jsou tím pádem pomalejší. Pro NEMEA Dashboard je potřeba takový jazyk, který je úzce integrovaný se systémem a dokáže využívat systémové nástroje<sup>9</sup>.

Do výběru jsem zahrnul interpretované i kompilované jazyky, aby byl demonstrován rozdíl ve výkonnosti, jednoduchosti použití, množství potřebných závislostí a dalších. Jmenovitě byly zahrnuty jazyky Python, C++ a JavaScript. Pro každý vybraný jazyk existuje

---

<sup>8</sup>angl. grid system

<sup>9</sup>Využití systémových nástrojů není součástí této práce, ale byla jednou z podmínek návrhu REST API pro NEMEA Dashboard.

několik knihoven pro podporu tvorby REST API, proto pro zpřehlednění byly vybrány nejvhodnější knihovny.

Jeden z nejpoužívanějších jazyků pro tvorbu webových stránek – PHP nebyl zahrnut z důvodu vysoké režie interpretu při běhu aplikace a mnoha závislostí (zejména Apache webový server), které jsou potřeba pro tvorbu webové stránky v PHP. Navíc jazyk PHP přímo nepodporuje tvorbu REST architektury, vše je řešeno přes Apache server pomocí .htaccess konfigurace.

## Flask (Python)

Flask je mikroframework využívající Werkzeug toolkit, který slouží jako základ pro Flask. Je nazývaný mikroframeworkem, protože vývojáře nenutí využívat konkrétní knihovny nebo nástroje, nedisponuje žádnou abstrakční databázovou vrstvou, verifikací vstupů nebo kterékoliv jinou komponentou běžně dostupnou ve frameworku podobného typu.

Nicméně Flask podporuje rozšíření, kterými lze rozšířit funkcionalitu přesně dle potřeb programátora. Tím se stává Flask velmi silným nástrojem při vývoji SPA. Jelikož je navíc napsán v jazyku Python je jeho použití více než intuitivní. Například definice cesty a obslužné rutiny je realizováno pomocí dekorátorů jak je znázorněno v následujícím příkladu [3.2](#).

```
@app.route("/")
def hello():
    return("Hello World!")
```

Listing 3.2: Definování cesty a obslužné rutiny v mikroframeworku Flask.

Veškeré zde zmíněné skutečnosti dělají z mikroframeworku Flask ideální nástroj pro návrh jak rozsáhlého, tak minimálního REST API, které lze libovolně škálovat dle potřeb projektu.

S ohledem na MongoDB existuje oficiální konektor nazvaný PyMongo. Ten se použitím velmi podobá mongo shell nástroji přímo v MongoDB.

## NodeJS (JavaScript)

JavaScript je původně určen do prostředí webových prohlížečů a jejich jader. Google ale vytvořil V8[14] JavaScriptové jádro primárně určené pro webový prohlížeč, na kterém je ale postaven i framework NodeJS. V8 umožňuje spouštět kód napsaný v JavaScriptu na straně serveru.

Architektura NodeJS je založena na principu asynchronních událostí, které dovolují vytvářet vysoce škálovatelné síťové aplikace jako je např. webový server s vysokou dostupností a nároky.

NodeJS není až tak knihovnou nebo frameworkem pro vývoj webových serverů, jako spíše sbírkou modulů, které obsluhují jednotlivé části funkcionality jádra[27]. Tyto moduly využívají API, které zjednodušuje komunikaci mezi moduly. Tím se velmi podobá NEMEA systému.

## Mongoose (C/C++)

Posledním kandidátem je framework napsaný v jazyku C. Největší výhodou tohoto řešení je jeho rychlost, která je mnohonásobně vyšší než u interpretovaných jazyků. Mongoose je minimalistický framework vytvořený pro vývoj webových serverů více typů.

Jelikož jazyk C není až tak pohodlný a rychlý pro vývoj abstraktních architektur rozhraní jakou je REST, zejména kvůli práci s textovými řetězci a manipulací s JSON objekty, rozhodl jsem se napsat obálku frameworku Mongoose do jazyka C++. Ten, ačkoliv je pomalejší, je vhodnější pro práci s textovými řetězci a existuje efektivní knihovna pro práci s JSON objekty RapidJSON[28].

Za použití vytvořené obálky, která je navržena specificky pro návrh REST API, lze velmi rychle vytvořit požadovanou funkcionalitu. Obálka totiž doplňuje možnosti Mongoose, který např. nedisponuje dynamickými URL<sup>10</sup> nebo zpracováním URL parametrů do vhodné struktury.

Největším problémem je absence oficiálního konektoru pro MongoDB, ten je nezbytnou podmínkou pro toto REST API. Samozřejmě existují různá řešení nebo knihovny. Ty jsou ale buď neudržované nebo náročné na použití.

## 3.3 Vybrané technologie

V předcházející kapitole jsem uvedl užší výběr technologií, které jsou vhodné pro tvorbu SPA jak z uživatelské, tak serverové části. Každá z nich je něčím specifická a vyčnívá oproti jiným kandidátům.

### 3.3.1 Uživatelská část

Prvními aspekty při výběru knihovny pro vývoj SPA v uživatelské části byla zejména velikost knihovny, množství dostupných modulů, licence, udržitelnost a v neposlední řadě dostupné funkce. Pro rychlý přehled byla vytvořena tabulka 3.1 (viz výše), která tyto faktory porovnává.

Licence BSD a MIT jsou velmi podobné z hlediska vývojáře a jejich možnostmi použití v rámci vývoje otevřeného projektu jakým je NEMEA.

Z hlediska velikosti jasně vyčnívá EmberJS, který je násobně menší než ostatní frameworky, avšak z ostatních faktorů pozbývá funkcionalitu, kterou např. dokáže nabídnout AngularJS.

Všechny frameworky jsou v současné době aktivně vyvíjeny a pravidelně aktualizovány. Navíc kolem každého frameworku, které jsou všechny open-source, je vytvořena aktivní komunita vývojářů, kteří přispívají do daného frameworku.

Pokud se zaměříme na možnosti frameworku, jasně vybočuje AngularJS, který implementuje kompletní MVC/MVVM architekturu. Tím pádem není nutná jakákoliv další knihovna nebo vlastnoruční implementace některých komponent, které budou při vývoji NEMEA Dashboard potřeba.

Poslední faktor – možnosti frameworku – je dle mého nejdůležitějším měřítkem při výběru. Ten totiž vyvaží i větší velikost frameworku a jeho náročnost na prohlížeč, ta se navíc s každou novou verzí většinou snižuje.

---

<sup>10</sup>z angl. uniform resource locator

Tím pádem je jasným vítězem framework AngularJS, který disponuje širokou funkcionalitou. Je navržený přímo pro vývoj komplexních SPA a má opravdu mnoho dostupných rozšíření, které jsou vhodné pro naše účely.

V rámci CSS frameworků bylo rozhodování mnohem přímočarejší, ačkoliv se tak na první pohled nemusí jevit, jelikož všechny tři frameworky nabízejí téměř totožnou funkcionalitu. Ale protože jsem zvolil AngularJS jako javascriptový framework, byla volba zcela jasná, protože Angular Material je přímým rozšířením frameworku AngularJS a do útrob Angular Material je velmi hluboce integrovaný. Navíc používá nejmodernější technologie, které usnadňují vývoj.

### 3.3.2 Serverová část

Při výběru technologie pro serverovou část jsem se zejména zaměřil na rychlost výsledného REST API a jeho udržitelnost z vývojářského hlediska (modulárnost, OOP). Proto jsem vytvořil velmi jednoduché REST API v každém z kandidátů a změřil jejich výkonnost. Ostatní hlediska jsou spíše subjektivním názorem.

V každém z frameworků jsem vytvořil jednoduché statické API (fixní URL), která pouze vrátila textový řetězec „Hello World“. Pro měření jsem využil osobní virtuální server s následujícími parametry:

- CPU – Intel(R) Xeon(R) CPU W3520 @ 2.67GHz (2 jádra),
- RAM 1GB DDR3 ECC,
- konektivita – 500 Mbps.

Server nebyl v době testů nijak vytížen a běžely na něm pouze základní služby jako Apache. Konfigurace serveru není nijak výkonná a tím pádem lze odvodit kolik zdrojů jaký jazyk spotřebuje a jak je připravený na škálovatelnost.

Pro vytvoření dostatečného množství dotazů na server jsem použil nástroj wrk<sup>11</sup>, který jsem spustil na počítači Mac Mini Server (Intel i7, 8 GB RAM, 1Gbps konektivita). Nástroj wrk jsem spouštěl s následujícími parametry:

- -d 20s – délka trvání testu,
- -t 10 – počet vláken,
- -c 200 – počet otevřených připojení,
- URL.

Testy jsem spouštěl celkem třikrát pro každý framework, abych se vyvaroval chybě měření a každé měření probíhalo nezávisle na předcházejícím. Stejný server se nikdy netestoval dvakrát za sebou. V tabulce 3.2 nalezneme naměřené výsledky.

C++ obálka frameworku Mongoose a NodeJS jsou několikanásobně rychlejší než framework Flask. Tato rychlost je zejména díky použitému jazyku v případě C++. Ten je také použit v případě NodeJS, kde většina komponent je právě napsána v C++.

Ačkoliv se může zdát, že NodeJS je jasnou volbou z naměřených výsledků, není tomu tak. NodeJS totiž během testu spotřeboval téměř všechny zdroje daného stroje (zejména

---

<sup>11</sup><https://github.com/wg/wrk>

Tabulka 3.2: Naměřené vlastnosti serverových frameworků pomocí nástroje wrk

Název	Celkem požadavků	Požadavků/s	Latence	Vytížení CPU
Flask	3 373	167.9	25.82 ms	25%
NodeJS	117 579	5871.6	33.60 ms	130%
Mongoose	136 001	6787.5	29.80 ms	100%

CPU). Díky tomu je natolik výkonný. Tohoto výkonu lze dosáhnout i u frameworku Flask, ale ten by musel být doplněn dalšími nástroji pro vyšší konkurentnost serveru.

Pokud porovnáme NodeJS a Flask, zjistíme, že NodeJS potřebuje obrovské množství závislostí pouze pro svůj běh a pokud chceme jakoukoliv pokročilou funkcionalitu, musíme dodat další závislosti. Tím pádem se základní varianta REST API jeví jako velmi výkonná, ale z dlouhodobého hlediska je tato rychlost neudržitelná.

Pokud bychom pracovali v izolovaném prostředí, je nejvhodnější NodeJS. Pokud ale budeme vycházet z architektury aplikace, zjistíme, že server musí nesmí být upřednostňován před databází. V ní totiž probíhají mnohem náročnější operace, které vytvářejí prodlevu v odpovědi v rámci sekund a ne jednotek milisekund jako v případě serveru.

Tím pádem nám zůstává Flask, který díky implementaci v Pythonu je velmi jednoduchý na použití s udržitelným kódem a pohodlnou prací s JSON objekty. Pokud bychom se zaměřili na rychlost, lze dosáhnout dobrých výsledků zapojením CPython. Navíc Python nedovolí spotřebovat veškeré zdroje stroje, což se stalo při testu Mongoose a zejména pak NodeJS serveru.

### 3.4 Shrnutí

Vybrané technologie jsou vyváženou kombinací široké funkcionality, rychlosti vývoje a udržitelnosti kódu. To platí jak pro uživatelskou, tak serverovou část aplikace.

Pro uživatelskou část jsem vybral JavaScript framework AngularJS společně s CSS frameworkem Angular Material, ty se navzájemně doplňují funkcionalitou a jsou velmi úzce provázány.

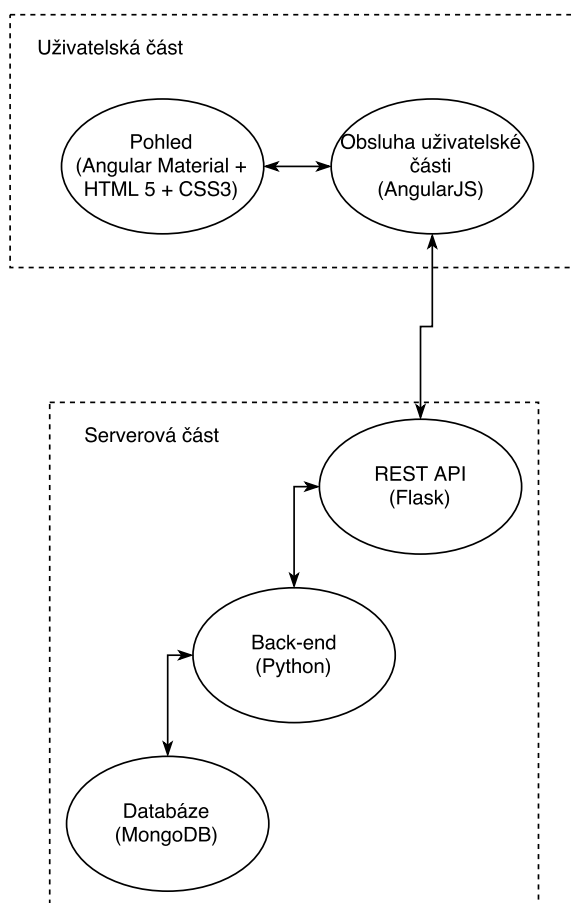
Serverová část bude realizována v jazyku Python pomocí mikroframeworku Flask. Ten disponuje všemi potřebnými funkcionalitami pro tvorbu REST API a MongoDB má oficiální konektor pro jazyk Python, který je snadný na použití a má široký repertoár funkcí a nastavení.

## Kapitola 4

# Architektura aplikace

Před samotnou implementací aplikace je potřeba navrhnout její architekturu. Ačkoliv se SPA může jevit jako jednoduchá a přímočará struktura, opak bývá většinou pravdou. Díky plynulosti práce s aplikací uživatel nevnímá změnu mezi pohledy v aplikaci. Ta se ale v pozadí může asynchronně dotazovat na server.

Celou SPA jsem již při výběru technologií rozdělil na dva logické celky. Uživatelskou a serverovou část. Schéma 4.1 znázorňuje architekturu SPA a kde jsou jaké technologie použité.



Obrázek 4.1: Rozvržení SPA na dva logické celky a jaké technologie tyto celky používají.

Díky použitým technologiím lze část rozdělit aplikaci i fyzicky na dva typy strojů. Serverová část je spuštěna na jednom počítači – serveru, který disponuje relativně vysokým výkonem a je určen pouze pro databázi a REST API. Databáze může být distribuována na dalším serveru nebo přítomna na stejném stroji. Server tedy zajišťuje funkčnost REST API a při iniciální návštěvě klienta také poskytuje stránku, kterou si klient stáhne na své zařízení.

Tím se distribuuje uživatelská část aplikace na samotná zařízení, která aplikaci zobrazují. Následkem je snížení nároků na serverovou část, která nyní obstarává pouze datovou a autentizační část SPA (vše uvnitř REST API). Jelikož je SPA z větší části uložena u uživatele, můžeme si dovolit při vývoji větší množství dat, které se při prvním načtení musejí stáhnout, protože se následně téměř žádná esenciální data nestahují (jsou tím myšlena např. data nutná pro vykreslení grafů).

Největší využití bude mít v aplikaci Dashboard (odtud i název aplikace), který bude obsahovat konfigurovatelné položky, aby si uživatel mohl Dashboard připravit přesně pro své potřeby. Dashboard je také vstupním bodem pro drill-down analýzu.

## 4.1 Případy užití

Před samotným návrhem architektury jsem sestavil tři případy užití, které budou opěrnými body pro NEMEA Dashboard. Tyto navržené případy zohledňují širokou škálu úkonů, které při běžném používání IDS mohou nastat.

Jako cílovou skupinu jsem stanovil správce větších sítí. Ti totiž musí být informováni o globálním dění a nemohou se zabývat menšími událostmi. Většinou jsou takovéto sítě stavěny velmi robustně a jsou dostatečně naddimenzovány. Nicméně i neúplné vítížení sítě může znamenat pro běžné uživatele nedostupnost některých služeb. Těmto výpadkům musí správce sítě předcházet a to zejména díky včasné analýze anomálií na síti.

### *Běžný přehled o síti a analýza provozu z několika posledních dní*

Správce sítě musí mít přehled o síti, kterou spravuje. Chce být informován o celkovém počtu útoku, které detekoval systém NEMEA a ty následně uložil jako události v IDEA formátu do databáze. Správce nechce pročitat emailové reporty nebo ručně procházet jednotlivé záznamy.

### *Detekce útoku většího rozsahu na síť*

Při běžné analýze sítě pomocí NEMEA Dashboard musí být útok většího rozsahu okamžitě vidět ve zobrazených datech. Většinou se takovýto útok nahlásí jako velké množství menších útoků. V tu chvíli je pro správce velmi těžké takový útok odhalit.

### *Analýza konkrétního útoku*

Uživatele zaujal velmi nezvyklý útok/událost a chce o ní zjistit více. Postupně se dostává k detailnějším informacím pomocí drill-down analýzy. Na konci analýzy vidí přímo událost, která daný útok způsobila a může podniknout další kroky jako např. zjišťování informací o útočníkovi, o oběti, případně o charakteru útoku.

### *Kontinuální přehled o síti*

Uživatel během své běžné pracovní činnosti Dashboard nijak aktivně nesleduje, pouze se mu autonomně aktualizuje a vidí aktuální dění na síti. Velmi jednoduchou a rychlou vizuální analýzou (běžným pohledem) identifikuje událost, která se právě vyskytla na síti s velmi krátkou časovou prodlevou (až v rámci jednotek minut).

Tyto případy užití lze rozdělit do dvou typů dat. Vizualizovaná data pomocí grafů a základní textové informace o databázi v daný časový interval. Z toho lze vyvodit několik základních typů obsahu pro dashboard:

- graf podílů (koláčový graf) – ukazuje podíly událostí dle kategorie v definovaném časovém okně,
- graf v čase (sloupcový graf) – zobrazuje množství událostí rozdělené v intervalech v definovaném časovém okně dle kategorií,
- informace o celkovém počtu (text) – celkový počet událostí v databázi v určitém časovém okně,
- informace o top událostech (text) – největší události dle statického pole v databázi v určitém časovém okně.

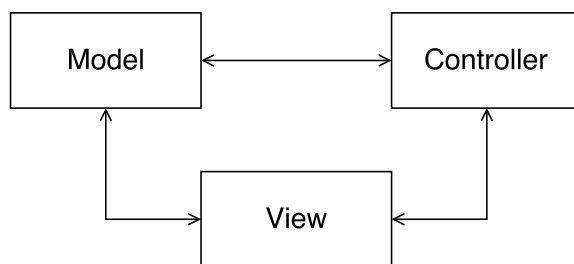
Ze všech typů boxu (kromě o celkovém počtu) bude možná drill-down analýza. V případě grafů bude možnost prokliku na výpis událostí, které jsou filtrované dle určitých metrik (např. kategorie a čas). U boxu s top událostmi bude odkaz přímo na detail dané události.

Další částí NEMEA Dashboard bude přehled všech událostí nazvaný jednoduše „Events“. Zde bude možnost vyhledávat a třídit všechny události, které se v databázi nacházejí. Všechny vyhledané a vyfiltrované události budou přehledně zobrazeny v tabulce, která bude obsahovat základních informace o události. U každé události bude možnost prokliku k detailním informacím, které jsou o události uloženy v databázi.

Poslední částí aplikace budou nastavení. Ty v tuto chvíli budou pouze pro uživatele a jejich správu.

## 4.2 Uživatelská část

Pokud se zaměříme konkrétně na architekturu uživatelské části, tak AngularJS využívá architekturu MVC. Ta spočívá v rozdělení aplikace na tři logické celky, které navzájem spolu komunikují a předávají si data.



Obrázek 4.2: MVC architektura

**Model** obstarává logiku a integritu aplikačních dat, které se v SPA nacházejí. Model také získává a odesílá data na server dle pokynů kontroléru. Tím se udržuje uniformní rozhraní pro obousměrnou komunikaci mezi klientem a serverem.

Z pohledu AngularJS jsou jednotlivé modely reprezentovány pomocí „service“. Ty jsou pomocí vkládání závislostí spojeny s jednotlivými kontroléry.



**View (pohled)** zobrazuje data, která jsou mu dodána od kontroléru. Pohled a kontrolér musí být velmi úzce propojeny, aby obě strany měly co nejaktuálnější data. To je zaručeno pomocí obousměrného vázání dat (angl. two-way data binding). Na základě požadované cesty router uvnitř AngularJS rozhodne jaký pohled má být vykreslen a předá jej jádru prohlížeče pro vykreslení.

Pohled lze rozdělit na dvě části. První z nich je šablona, která je vytvořena v HTML. Ta může obsahovat speciální značky i vlastní definované DOM elementy. Tato šablona je následně při zobrazování „kompilována“ a zobrazena uživateli. Tímto se zachová obousměrné vázání dat.

**Controller (kontrolér)** je prostředníkem mezi modelem a pohledem. Zabezpečuje aktualizaci dat na obou stranách. Avšak v rámci AngularJS lze kontrolér obejít a data z modelu může získat přímo pohled. Tím přetváříme architekturu SPA na hybridní mezi MVC a MVVM.

Jednotlivé kontroléry mohou být do sebe vnořené a navzájem spolu komunikovat pomocí událostí, kde jeden kontrolér událost vyvolá jak směrem nahoru, tak může i dolů a další kontroléry mohou na tuto událost reagovat.

Na schématu 4.3 je znázorněno jak aplikace bude pracovat s daty, které získá z REST API. Veškerá data, která lze z REST API získat, jsou JSON objekty. To velmi usnadní práci s nimi a nebude nutná téměř žádná manipulace či konverze, protože JSON je nativní struktura pro JavaScript. Jednotlivé cesty v aplikaci jsou pouhé koncepty, které reprezentují část SPA. Stejně tak komunikace s API a obousměrné vázání dat je naznačeno pouze koncepčně a vše je konkretizováno v následující kapitole 5.

Pokud bychom architekturu popisovali z pohledu uživatele, tak začínáme načtením dané cesty. Zde AngularJS zvolí šablonu a kontrolér, který je k dané cestě definován.

Poté zavolá kontrolér a spustí se kód, jenž obsahuje. Kód uvnitř kontroléru se naváže na značky specifické pro AngularJS a postupně celou šablonu „zkompiluje“<sup>1</sup>. Tím vznikne obousměrné vázání dat mezi pohledem a kontrolérem.

V kontroléru jsou pomocí vložení závilostí registrovány modely, které kontrolér chce využít (nemusí je nutně použít). Modely jsou instanciovány jako singletony v rámci celé aplikace. To snižuje nároky na výpočetní výkon jádra prohlížeče.

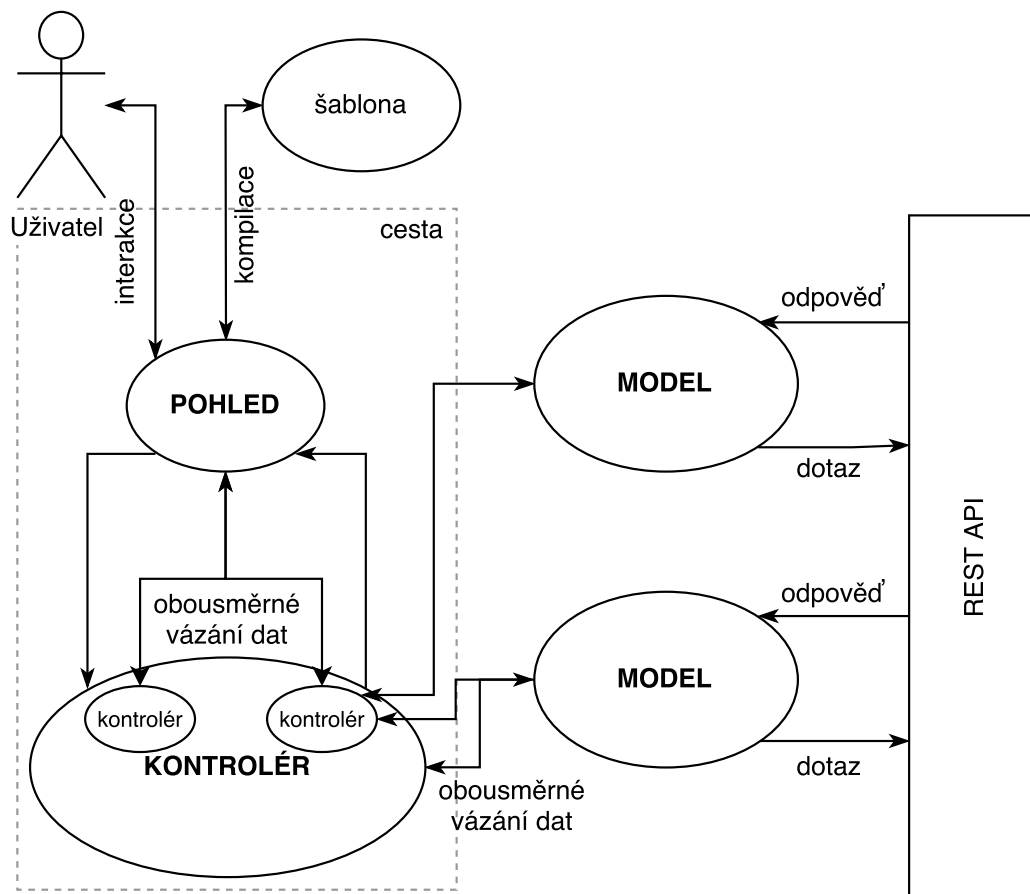
Modely mohou využívat jiných modelů. Tímto zapouzdřením funkcionality lze dosáhnout velmi vysoké abstrakce. V NEMEA Dashboard je tato abstrakce navrhnutá pro dotazy do REST API. V AngularJS je pro tvorbu HTTP požadavků vytvořen unifikovaný model `$http`, ten by bylo ale nutné při každém požadavku složitě konfigurovat. Proto bude vytvořen model `api`, který bude využívat `$http` a bude autonomně konfigurovat adresu, parametry, HTTP hlavičky a ošetří návratového hodnoty a data.

## 4.3 REST API

Representational state transfer, zkráceně REST byl definován v roce 2000 v dizertaci Roye Thomase Fieldinga [10]. Ten jej definuje jako architektonický styl pro návrh distribuovaného hypermediálního obsahu. REST je postaven na několika základních ideích, které platí i dnes.

---

<sup>1</sup>V tomto případě se HTML transformuje, přidávají potřebné třídy a identifikátory a nahrazují značky za textové řetězce.



Obrázek 4.3: Schéma uživatelské části

#### *Klient-server*

Tento návrh architektury je nejzákladnějším stylem pro síťové aplikace. Server, nabízející sadu služeb, čeká na žádosti, které následně obslouží. Klient, který se snaží vykonat určitou činnost, posílá žádosti na server skrze předem definovanou cestu. Server žádost v případě nevalidního formátu odmítne, v kladném případě provede žádost a odešle odpověď zpět klientovi.

Tímto oddělením uživatelské části od serverové získává aplikace vyšší přenositelnost, škálovatelnost a nezávislost komponent. To dovoluje částečn separátní vývoj.

#### *Bezstavovost*

Zřejmě nejdůležitějším aspektem REST API je bezstavová komunikace. Ta se zaručí tak, že každý dotaz, který je na API zaslán, obsahuje veškeré informace, které jsou potřebné pro porozumění žádosti. Takto se klientovi zabrání využit jakéhokoliv kontextu, který by na serveru mohl být uchován.

#### *Vyrovňovací paměť*

Pro zlepšení výkonu je do rozhraní vhodné implementovat vyrovnávací paměť. To však s sebou nese riziko neaktuálnosti dat, což v případě systému NEMEA je velmi nebezpečná vlastnost. Proto lze tento faktor opomenout v případě vysoce volatilních rozhraní jakým systém NEMEA rozhodně je.

### *Uniformní rozhraní*

Uniformnost rozhraní je nejdůležitější vlastností REST architektury, která jej odlišuje od ostatních architektonických stylů návrhu jakým je například RPC[23]. Generalizací rozhraní vývojář získá velmi rychlý přehled o dostupných operacích s API a jaké služby poskytují.

Aby rozhraní bylo uniformním, musí splňovat několik vlastností: identifikovatelnost zdrojů, manipulace se zdroji skrz reprezentaci, samopopisné zprávy a hypermédia jako jádro aplikačního stavu.

### *Vrstvený systém*

Aby se rozhraní dokázalo v čase přizpůsobovat potřebám vývojáře, je nutné navrhovat rozhraní vrstveně. To jej dovoluje libovolně rozšiřovat jak do šířky, tak do hloubky. Pokud se na rozhraní díváme jako na jednotlivé komponenty, tak ty, díky vrstvení, nejsou schopné interagovat s dalšími vrstvami a tím pádem jinými komponentami. Z klientské strany to znamená interakci vždy pouze s jediným zdrojem.

Při aplikaci těchto ideí do návrhu REST API pro NEMEA Dashboard jsem narazil na několik problémů, které lze vyřešit kompromisy. Jde zejména o typ API, které jsem pro aplikaci navrhl. To totiž z většinové části data pouze získává (cca 95%). Úprava či zápis dat se realizuje pouze v případě práce s uživateli nebo konfigurací dashboardu.

Tím pádem z CRUD modelu využíváme pouze jednu část, kterou je čtení. Tato část je realizována pomocí HTTP požadavku GET. Ten, ačkoliv dle RFC 2616 [9] může disponovat tělem zprávy, tělo zprávy většinou nepoužívá. Lépe řečeno jej většina webových serverů nepoužívá a tudíž by při návrhu bylo velmi nevhodnou praktikou toto chování implementovat.

Tím pádem jsem při návrhu rozhraní zvolil kompromis mezi REST a RPC, který narozdíl od REST využívá URL parametrů. Tento kompromis dovoluje při návrhu architektury použít jedno uniformní rozhraní pro více typů dotazů. To, ačkoliv porušuje základní ideu REST architektury, velmi usnadní vývoj uživatelské části.

Tento kompromis nerozděluje rozhraní dle získaných dat, ale dle typu dat, které uživatelské rozhraní získá. Rozdíl v sémantice dat je totiž daleko důležitějším měřítkem než struktura odeslané zprávy. Konkrétně jsem tento přístup zvolil u agregovaných dat, které jsou vyhodnocovány již na serveru při dotazování do databáze.

## 4.4 GUI

Při návrhu grafického uživatelského rozhraní jsem se snažil využívat co nejjednodušší drátěné modely. Ty totiž ponechávají dostatečnou flexibilitu a zároveň jsou natolik vyjadřující, že je z nich vidět základní koncept a rozložení prvků v aplikaci.

V příloze A.1 je vidět hlavní stránka aplikace, neboli dashboard. Ten se skládá z jednotlivých boxů, které jsou konfigurovatelné jak vzhledem (lze je zmenšit, zvětšit, přidat, smazat), tak obsahem. Ten je konfigurován ve zvláštním okně aplikace, které je specifické pro každý box. Na drátěném modelu A.2 lze vidět jeho návrh.

Při prokliku během drill-down analýzy se uživatel přesune na druhou část aplikace – jednotlivé události, které daný útok/anomálii způsobily. V této části, znázorněné na drátěném modelu A.3, má uživatel možnost se libovolně dotazovat do databáze událostí a získat tak z ní události, které přesně hledá. Ty jde nadále živě filtrovat přímo ve výpisu událostí (neprobíhá žádné dotazování na server).

Jednotlivé řádky tabulky obsahují pouze esenciální informace o události (ty se rozhodnout během vývoje a dle zkušeností s prací s databází událostí). Při kliknutí na daný řádek se zobrazí detailnější informace o události a pokud ani to uživateli nestačí, může přejít na detail události, který obsahuje veškeré informace o události obsažené v databázi.

Grafika samotné aplikace bude vznikat během její implementace a to v režii CSS části frameworku Angular Material. Ten má kaskádové styly pro většinu běžně používaných elementů a nevzniknou tak při vývoji výrazné odchylky od prvotního návrhu.

## 4.5 Shrnutí

NEMEA Dashboard je nyní navržený. Architekturu REST API začínaje, grafickým uživatelským rozhraním konče. Byly připraveny případy užití, které se budou implementovat. Typy dat, které se budou vizualizovat a zpracovávat. Definoval jsem REST API, které bude aplikace na uživatelské části využívat a naznačil průběh drill-down analýzy.

## Kapitola 5

# Implementace

Aplikace je navržena z architektonického hlediska a další fází je její tvorba. Prvně se bude implementovat serverová část, která je napojena na databázi událostí reportované systémem NEMEA.

Další fází bude tvorba uživatelské části, která bude dodaná data vizualizovat pomocí interaktivních grafů.

### 5.1 Serverová část

Prvním krokem při implementaci serverové části bylo vytvoření vývojového prostředí. To, díky jazyku Python, může být na jakémkoliv \*nix systému bez nutnosti cokoli upravovat. V mém případě jsem měl v prvních krocích vývoje lokální vývojové prostředí na systému Mac OS X. Kde jsem replikoval databázi událostí z jednoho ze serverů sdružení CESNET.

V pozdějších částech vývoje byla serverová část přesunuta na stejný server, kde byla umístěna databáze událostí, do které systém NEMEA ukládal události v reálném čase. Tento přesun byl nutný, abych otestoval přenositelnost serverové části na jiný stroj a díky tomu získal i databázi vyvíjející se v reálném čase.

Instalace potřebných Python knihoven proběhla pomocí nástroje `pip`. Ty zahrnují zejména mikroframework Flask, jeho závislosti a konektor pro MongoDB `pymongo`. Všechny závislosti jsou uvedeny v příloze B.1 tak, jak jsou uvedeny v souboru `requirements.txt`, který využívá nástroj `pip` pro automatickou instalaci všech závislostí.

```
/v2
  /events
    /indexes [GET]
    /:n [GET]
    /query [GET]
    /agg [GET]
    /top [GET]
    /count [GET]
    /id/:id [GET]
  /users [GET, PUT, POST, DELETE]
    /auth [POST]
    /logout [DELETE]
```

Listing 5.1: Koncové body REST API, které jsou dostupné ze serveru.

V **listing 5.1** je naznačena struktura finálního REST API, která je velmi lehce rozšiřitelná, upravitelná a zejména pochopitelná.

Během vývoje API se několikrát měnila struktura koncových bodů, proto jsem zvolil verzování API pomocí URL, aby v budoucích případech mohlo API fungovat pro více verzí současně. Tuto část URL jsem pracovní nazval prefix verze.

V době psaní této práce je API ve druhé verzi, která se od první verze liší zejména architekturním typem, kterým je REST. V první verzi se API architekturou podobalo více RPC pouze s několika rysy REST.

Za prefixem verze následuje funkcionální prefix. Ten v současném návrhu rozdluje API na dvě základní části. První část obsluhuje data událostí a je nazvána `/events`. Za URL `/v2/events` se nacházejí již jednotlivé koncové body API, ty jsou charakteristické zejména tím, že dovolují pouze jednu HTTP metodu, konkrétně `GET`.

#### `/indexes`

Tato část byla vytvořena prvně jako experimentální část API pro ověření funkcionality a zejména pak správného spojení s databází. V současné chvíli je připravena pouze jako další koncový bod pro budoucí použití.

Jak již URL napovídá, tento koncový bod pracuje s indexy v databázi. Prvně zkontroluje, zda požadované indexy v databázi existují a v případě existence vrátí nezměněný výstup z MongoDB obsahující informace o indexech.

V opačném případě pevně definované indexy (index pro seřazení databáze událostí dle klíče `DetectTime` a automatické mazání záznamů v kolekci `sessions`

#### `/:n`

URL `/:n` znamená dynamickou URL, kde `n` je číslo od 1 do 10 000 a značí kolik událostí má být vyhledáno seřazených dle času<sup>1</sup>. Tento limit je stanoven na serverové části, aby nedošlo k přetížení serveru, protože v případě čísla 0 by databáze navrátila všechny položky, což, v době psaní práce, je přibližně 5 GB textu.

#### `/query`

Tento koncový bod slouží pro pokládání předem specifikovaných dotazů do databáze. API nepřijímá konkrétní MongoDB dotaz, ale pevně mnou specifikovaná pole. Toto omezení je zejména kvůli bezpečnosti.

#### `/agg`

V tomto bodě byl zejména využit agregační framework uvnitř MongoDB, který je optimalizovaný na práci s velkými daty. Slouží pro vypočítání časových intervalů a počtů položek v databázi uvnitř časových oken.

Agregační framework v MongoDB pracuje s projekcemi. Tzn. prvně se vyhledají položky splňující zadaná kritéria (běžný dotaz), ty jsou následně projektovány dle časového intervalu (změní se čas události na začátek intervalu). Nakonec jsou všechny události se stejným časem spočteny a ty tento výsledek je vrácen.

#### `/top`

Slouží pro box typu „TOP“. Ten vybírá z databáze událost, která obsahuje nejvyšší číslo `FlowCount` u každé z kategorií.

---

<sup>1</sup>Čas se v databázi bude vždy odvozovat od hodnoty klíče `DetectTime`

#### `/count`

Box typu „COUNT“ pouze zobrazuje číslo reprezentující počet událostí, které splnily daná kritéria (časové okno a příp. kategorie).

#### `/id/:id`

Každá událost v databázi je opatřena unikátním ID, které samo jádro MongoDB každým záznamem opatří. IDEA formát specifikuje zároveň i vlastní typ ID, nicméně na tuto položku by v databázi musel být vytvořen index, aby vyhledávání bylo efektivní. Nativní databázové `_id` tento index má v základu a proto je vyhledání velmi rychlé.

Druhou částí API je práce s uživateli a jejich autentikace. O bezpečnosti aplikace a REST API pojednává poslední sekce 5.3 v této kapitole. Tyto koncové body již dovolují i ostatní HTTP metody.

#### `/users`

Tento koncový bod slouží pro veškerou práci s uživateli. Metoda `GET` vrátí záznamy o všech uživateli v NEMEA Dashboard. Tyto záznamy jsou identické k těm v databázi až na absenci hesel, která jsou odstraněna.

Metoda `PUT` slouží pro editaci uživatele. Toho je využito zejména při změně konfigurace dashboardu.

`POST` metoda vloží do databáze nového uživatele. Ten bude obsahovat předem pevně definovaný dashboard. Vše ostatní je nastavitelné v rámci aplikace.

Poslední metodou je `DELETE`. Ta uživatele nenávratně smaže z databáze.

#### `/auth`

Jediný koncový bod v API, který nevyžaduje autentikaci, protože pro ni slouží. Při dotazu na tento bod je metodou `POST` poslán přihlašovací email a heslo, dle kterých je uživatel v databázi ověřen.

Ověření probíhá ve dvou úrovních, prvním je nalezení uživatele dle emailu a následně porovnání hash hodnoty poskytnutého hesla a hesla v databázi. Pokud jedna z těchto úrovní selže, API vrací HTTP odpověď typu „401 Not Authorized“. Tuto chybu následně uživatelská část zpropaguje graficky uživateli.

V případě úspěšného ověření je odpovědí JSON web token (zkráceně JWT)[15]. O JWT více v sekci 5.3.

`/logout` Pro odhlášení uživatele slouží tento koncový bod. Ten, ačkoliv by to bylo možné, není zahrnut do koncového bodu `/auth` a to zejména kvůli bezpečnosti. Zde totiž využívám jedinou HTTP metodu `DELETE`, která v sobě nenese žádná data. Identifikace uživatele probíhá opět pomocí JWT.

REST API je v této chvíli naprogramováno. Je stabilní (ověřeno dlouhodobým provozem na serveru), rychlé a lehké na použití. Koncové body, které byly navrženy, obslouží všechny požadavky uživatelské části aplikace, které byly v případech užití navrženy.

## 5.2 Uživatelská část

Prvním krokem při vývoji uživatelské části bylo navržení stránky s výpisem posledními událostmi, které byly vloženy do databáze. Ta totiž nevyžadovala téměř žádnou konfiguraci a ověřila koncept a funkcionalitu navržené architektury.

Prvotní návrhy byly velmi strohé, bez použití knihovny Angular Material. Ta byla započena až při implementaci dashboardu. Ten je svou povahou komplexním systémem vnořeným uvnitř aplikace. Zejména proto, že každý box v dashboardu je konfigurovatelný nezávisle na ostatních a zároveň musí dashboard všechny boxy navzájem seřadit a umístit do mřížky<sup>2</sup>. První pokusy vytvořit vlastní implementaci systému mřížky se nesetkaly s úspěchem.

Původní návrh počítal s návrhem, který byl orientovaný pouze řádkově, tzn. prvně se přidala řada, do které se přidávaly jednotlivé boxy, které se automaticky přizpůsobily počtu boxů v řádce. To vytvářelo problémy hlavně pokud by uživatel chtěl box přesunout. Navíc se tento návrh nepřizpůsoboval obsahu a vznikl tak problém se zobrazováním grafů, které velmi často přetékal mimo box.

Pro další práci se tedy zvolila již existující knihovna Gridster<sup>3</sup>, která dovoluje vytvořit komplexní systém mřížky a je velmi dobře konfigurovatelná. Knihovna Gridster má navíc implementaci přímo pro AngularJS, tzn. direktivy, které lze okamžitě použít v šabloně stránky a následně v kontroléru vše spravovat (např. registrace k událostem vyvolané v pohledu po přesunu položky).

Gridster je velmi flexibilním systémem pro tvorbu mřížek. Je založen na minimální, pevně stanovené velikosti jednoho bloku, od kterého se odvíjí velikost boxu v jeho násobcích. Boxu lze následně tahem myši měnit velikost tak, jak je zvykem u běžného desktopového okna. Jediným rozdílem je, že velikost je měněna v násobcích jednoho bloku. Tato velikost je odvozena od výšky nebo šířky jednoho řádku, nebo vypočítána z počtu sloupců. V případě NEMEA Dashboard byla výška řádku stanovena experimentálně na 170 pixelů a 8 řádků. To nám dovoluje dostatečně flexibilní rozložení boxů a zároveň určitou jistotu při návrhu jednotlivých typů boxů.

Pokud uživatel chce přesunout box na jiné místo, Gridster disponuje podporou pro „drag and drop“ funkcionalitu. Ta je ještě vylepšena v rámci NEMEA Dashboard o zvýraznění cílového místa, aby uživatel nebyl z počátku překvapen, proč se jednotlivé boxy „samy hýbou“.

Při návrhu jsem opomněl výběr knihovny pro vykreslování grafů. Ta během různých experimentů s několika nejrozšířenějšími knihovnami (Google Charts, HighCharts.js, D3, NVD3 a další) byla velmi promptně vybrána. Zvolil jsem implementaci NVD3 knihovny pro AngularJS (angular-nvd3) a to zejména kvůli jednoduchosti použití, široké škále nastavení a předchozím zkušenostem s touto knihovnou. NVD3 je pouhou obálkou pro knihovnu D3, která je koncipována pro vykreslování jakéhokoliv typu dat v prohlížeči. To sebou nese vysokou flexibilitu, ale zároveň i velmi složitou tvorbu složitějších konstrukcí jakou jsou např. interaktivní grafy.

Jak již při návrhu bylo naznačeno, pro Dashboard se navrhli 4 základní typy grafů, které jsou pracovně nazvány následovně:

- top – událost s nejvyšší hodnotou FlowCount pro každou kategorii,
- sum – počet událostí dané kategorie nebo celkový počet všech událostí v daném čase,
- piechart – koláčkový graf zobrazující podíly událostí dle zadané metriky,
- barchart – sloupcový graf pro zobrazení událostí agregovaných dle stanoveného časového intervalu.

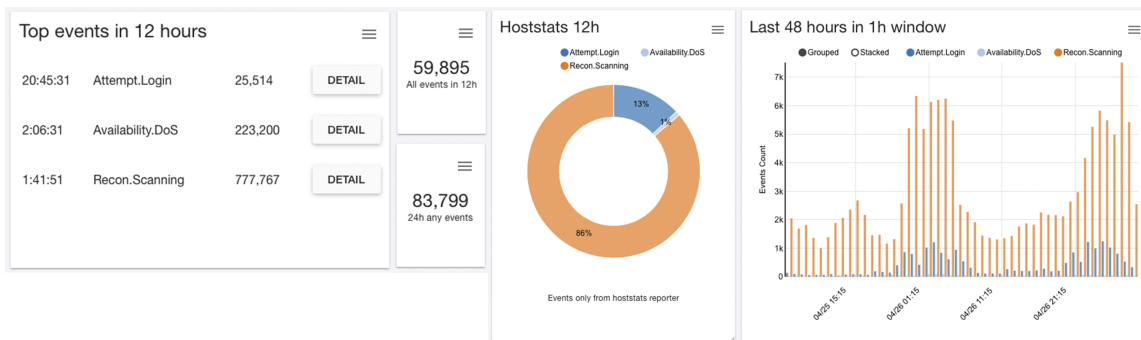
---

<sup>2</sup>angl. grid

<sup>3</sup><https://gridsterurl.com>



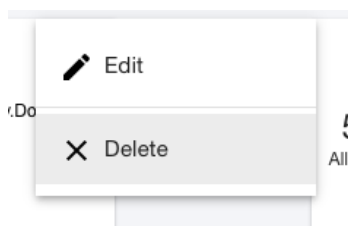
Pro poslední dva bylo využito grafového knihovny NVD3 a jejich výsledná podoba je znázorně v obrázku 5.2 ve stejném pořadí jako jsou vypsaný výše.



Obrázek 5.1: Vyobrazení jednotlivých typů boxů dostupných v NEMEA Dashboard.

Pokud se zaměříme na společné části všech boxů, nalezneme pouze 3 části. Titulek, který není vyžadovaný (lze vidět na boxu typu „sum“, který v obrázku 5.2 titulek nemá). Textový obsah boxu, který také není povinný, ale doporučený pro nahrazení titulku v boxu typu „sum“, kvůli poměru velikosti boxu (většinou minimální velikost 1x1 blok) a velikosti písma titulku. Textový obsah je vidět i u boxu typu „barchart“, kde slouží pro upřesnění obsahu daného boxu.

Posledním společným prvkem je menu, které nabízí dvě položky. Editaci a smazání boxu. Toto menu je vidět pouze po kliknutí na ikonu menu, která je u každého boxu symbolizována jako tři krátké vodorovné čáry, slangově nazýváno „burger menu“.



Obrázek 5.2: Kontextové menu přítomné u každého boxu.

Při konfiguraci boxu se dynamicky mění obsah formuláře, kterým je vše nastavováno. Ten je realizován jako vyskakovací okno v rámci stránky<sup>4</sup>. Těchto kombinací je větší množství a proto uvedu pouze ukázkový příklad.

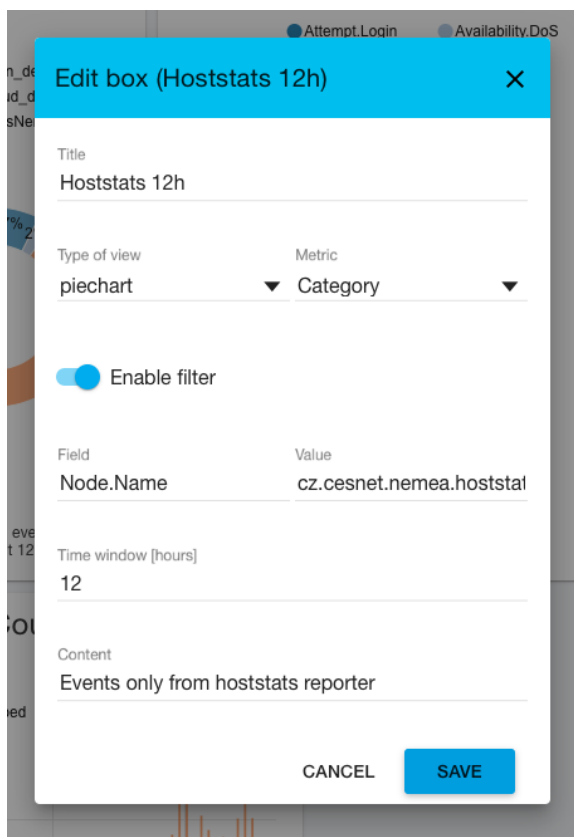
Při přidání nového boxu máme v dashboardu pouze obálku pro námi definovaný obsah. Uživatel tedy klikne na kontextové menu boxu a zvolí položku „Edit“. Zde je pouze předem definovaný titulek „New Box“, u kterého lze změnu vidět v reálném čase. Následně zvolí typ boxu, v případě typu „top“ má poté na výběr pouze časový interval, ve kterém se mají události hledat. V ostatních případech vybírá metriku (buď předdefinovanou, nebo vlastní), dle které se výsledky agregují. Uživatel v našem případě zvolí typ „piechart“, který má možnosti filtrování vyhledaných událostí. Jelikož se záznamem události v MongoDB pracuje téměř totožně jako s formátem JSON, můžeme použít tečkovou notaci pro přístup k jednotlivým hodnotám. Ve filtru volíme klíč a hodnotu, dle které se výsledek filtruje. V tomto případě, který je znázorněný na obrázku 5.3, filtrujeme události pouze z HostStats,

<sup>4</sup>angl. lightbox

což je sbírka modulů **zjistit co to přesně je**.

Posledními možnostmi je nastavení časového okna, ve kterém se událostí mají hledat a textový popis daného grafu. Poté uživatel klikne na tlačítko „Save“ a nastavení se uloží. To spustí rutinu v kontroléru, která pomocí modelu zašle dotaz na API s danými parametry a načte data do nově nakonfigurovaného boxu a vykreslí graf jakmile budou data přítomna u uživatele. Mezitím se v boxu zobrazí nápisek „loading...“.

Pokud uživatel klikne na „Cancel“, box se navrátí do původní konfigurace.



Obrázek 5.3: Editační okno boxu.

Takto je možné nakonfigurovat jakýkoliv box v dashboardu, nicméně během vývoje vznikl na požadavek existence více instancí dashboardu, každá s vlastními boxy a nastavením. Na tento požadavek jsem promptně reagoval a byl vytvořen následující systém upravený s ohledem na definové případy užití.

Dashboard je realizován jako jednotlivé pohledy na data, které se uživatel sám specifikuje. Každý takový pohled se chová a vypadá jako jeden dashboard a může mezi nima libovolně přecházet. Každý tento pohled má navíc vlastní konfiguraci, ta je dostupná přes kontextové menu pohledu, které se nachází v pravém dolním rohu.

#### Add item

Tato možnost přidá do současného pohledu nový box, který si uživatel dále přizpůsobí.

#### Clear cache

Během vývoje dashboardu jsem implementoval dočasnou paměť pro načtená data z REST API, aby se s dashboardem mohlo pracovat offline nebo na nestabilním internetovém připojení. Tato funkcionality je realizována pomocí lokálního uložště v

prohlížeči (local storage). Pokud si uživatel přeje aktualizovat data, použije toto tlačítko.

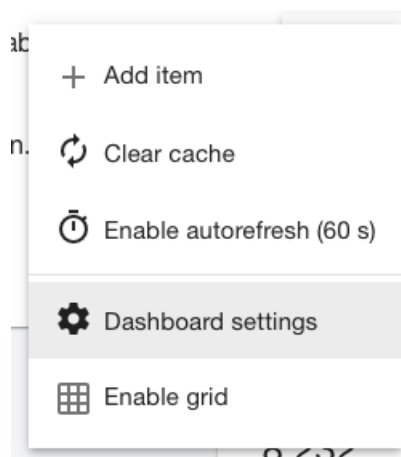
#### Enable autorefresh

Další funkcionality, která byla přidána během vývoje jako reakce na požadavek kontinuálního obnovování dat v současném pohledu. Interval obnovení dat nastavitelný pro každý pohled.

#### Dashboard settings

Zobrazí vyskakovací okno podobné tomu u boxu. V tomto okně si uživatel může nastavit obnovovací interval, název pohledu a pohled případně smazat.

**Enable grid** Během používání dashboardu se uživatelům často stávalo, že si náhodně posunuli s boxem a tím se jim celá mřížka, kterou pečlivě organizovali, přeskládala na nepoužitelnou variantu. Tato možnost vypíná nebo zapíná (záleží na předešlém stavu) „drag and drop“ funkcionalitu a změnu velikosti boxu.



Obrázek 5.4: Kontextové menu pohledu.

## 5.3 Zabezpečení

flow diagram jak probíhá autentizace – stejné jako voip volání

## 5.4 Distribuce

## Kapitola 6

### Dosažené výsledky

## Kapitola 7

### Závěr

Závěrečná kapitola obsahuje zhodnocení dosažených výsledků se zvlášť vyznačeným vlastním přínosem studenta. Povinně se zde objeví i zhodnocení z pohledu dalšího vývoje projektu, student uvede náměty vycházející ze zkušeností s řešeným projektem a uvede rovněž návaznosti na právě dokončené projekty.

# Literatura

- [1] Carter, E.; Foreword By-Stiffler, R.: *Cisco secure intrusion detection systems*. Cisco Press, 2001.
- [2] Čeleda, P.; Kováčik, M.; Koníř, T.; aj.: FlowMon Probe. *Networking Studies*, 2006: str. 67.
- [3] Cisco Systems, I.: Cisco Announces Agreement to Acquire Sourcefire. 2013.  
URL <http://www.cisco.com/c/en/us/about/corporate-strategy-office/acquisitions/sourcefire.html>
- [4] Cisco Systems, I.: Snort Users Manual. 2016.  
URL <http://manual.snort.org/node2.html>
- [5] Dressler, F.; Carle, G.: History-high speed network monitoring and analysis. In *Proceedings of 24th IEEE Conference on Computer Communications (IEEE INFOCOM 2005), Miami, FL, USA*, 2005.
- [6] Ember.js: HTMLBars. 2015.  
URL <http://emberjs.com/blog/2015/02/07/ember-1-10-0-released.html>
- [7] Facebook: JSX in Depth. 2015.  
URL <https://facebook.github.io/react/docs/jsx-in-depth.html>
- [8] FDIs, I.: 9241-210 (2009). Ergonomics of human system interaction-Part 210: Human-centered design for interactive systems (formerly known as 13407). *International Organization for Standardization (ISO). Switzerland*, 2009.
- [9] Fielding, R.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, Srpen 1999.  
URL <https://tools.ietf.org/html/rfc2616>
- [10] Fielding, R. T.: *Architectural styles and the design of network-based software architectures*. Dizertační práce, University of California, Irvine, 2000.
- [11] Flowmon Networks, a.: Flowmon Anomaly Detection System. 2016.  
URL <https://www.flowmon.com/cs/products/flowmon/anomaly-detection-system>
- [12] Foundation, M.: JavaScript Promise. 2016.  
URL [https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise)
- [13] Gackenhaimer, C.: What Is React? In *Introduction to React*, Springer, 2015, s. 1–20.

- [14] Google: Chrome V8. 2016.  
URL <https://developers.google.com/v8/>
- [15] Jones, M.: JSON Web Token (JWT). RFC 7519, RFC Editor, Květen 2015.  
URL <https://tools.ietf.org/html/rfc7519>
- [16] Lampert, R. T.; Sommer, C.; Münz, G.; aj.: Vermont-a versatile monitoring toolkit for IPFIX and PSAMP. In *Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation, MonAM*, ročník 6, 2006.
- [17] Mazinanian, D.; Tsantalis, N.: An empirical study on the use of CSS preprocessors.
- [18] Mikowski, M. S.; Powell, J. C.: Single Page Web Applications. *B and W*, 2013.
- [19] Munz, G.; Fessi, A.; Carle, G.; aj.: DIADEM firewall: Web server overload attack detection and response. *Broadband Europe (BBEurope)*, 2005.
- [20] Nurseitov, N.; Paulson, M.; Reynolds, R.; aj.: Comparison of JSON and XML Data Interchange Formats: A Case Study. *Caine*, ročník 2009, 2009: s. 157–162.
- [21] Patil, S.; Rane, P.; Meshram, D. B.: IDS vs IPS. *IRACST–International Journal of Computer Networks and Wireless Communications (IJCNC)*, ISSN, 2012.
- [22] Paxson, V.: Bro: a system for detecting network intruders in real-time. *Computer networks*, ročník 31, č. 23, 1999: s. 2435–2463.
- [23] Richardson, L.; Ruby, S.: *RESTful web services*. "O'Reilly Media, Inc.", 2008.
- [24] Roesch, M.; aj.: Snort: Lightweight Intrusion Detection for Networks. In *LISA*, ročník 99, 1999, s. 229–238.
- [25] Shirey, R.: Internet Security Glossary, Version 2. RFC 4949, RFC Editor, Srpen 2007.  
URL <https://tools.ietf.org/html/rfc4949>
- [26] Smith, J.: PATTERNS-WPF Apps With The Model-View-ViewModel Design Pattern. *MSDN magazine*, 2009: str. 72.
- [27] Teixeira, P.: *Professional Node.js: Building Javascript based scalable software*. John Wiley & Sons, 2012, ISBN 9781118240564.
- [28] Tencent: RapidJSON. 2016.  
URL <https://github.com/miloyip/rapidjson/>
- [29] Velan Petr, K. R.: Flow Information Storage Assessment Using IPFIXcol. In *Lecture Notes in Computer Science 7279*, Springer, 2012, ISBN 978-3-642-30632-7.
- [30] W3C: Shadow DOM. Working Draft. 2015.  
URL <http://www.w3.org/TR/shadow-dom/>
- [31] W3C: Flexible Box Layout Module Level 1. Candidate Recommendation. 2016.  
URL <http://www.w3.org/TR/css-flexbox-1/>
- [32] WebComponents: Web Components. 2016.  
URL <http://webcomponents.org/>

# Přílohy

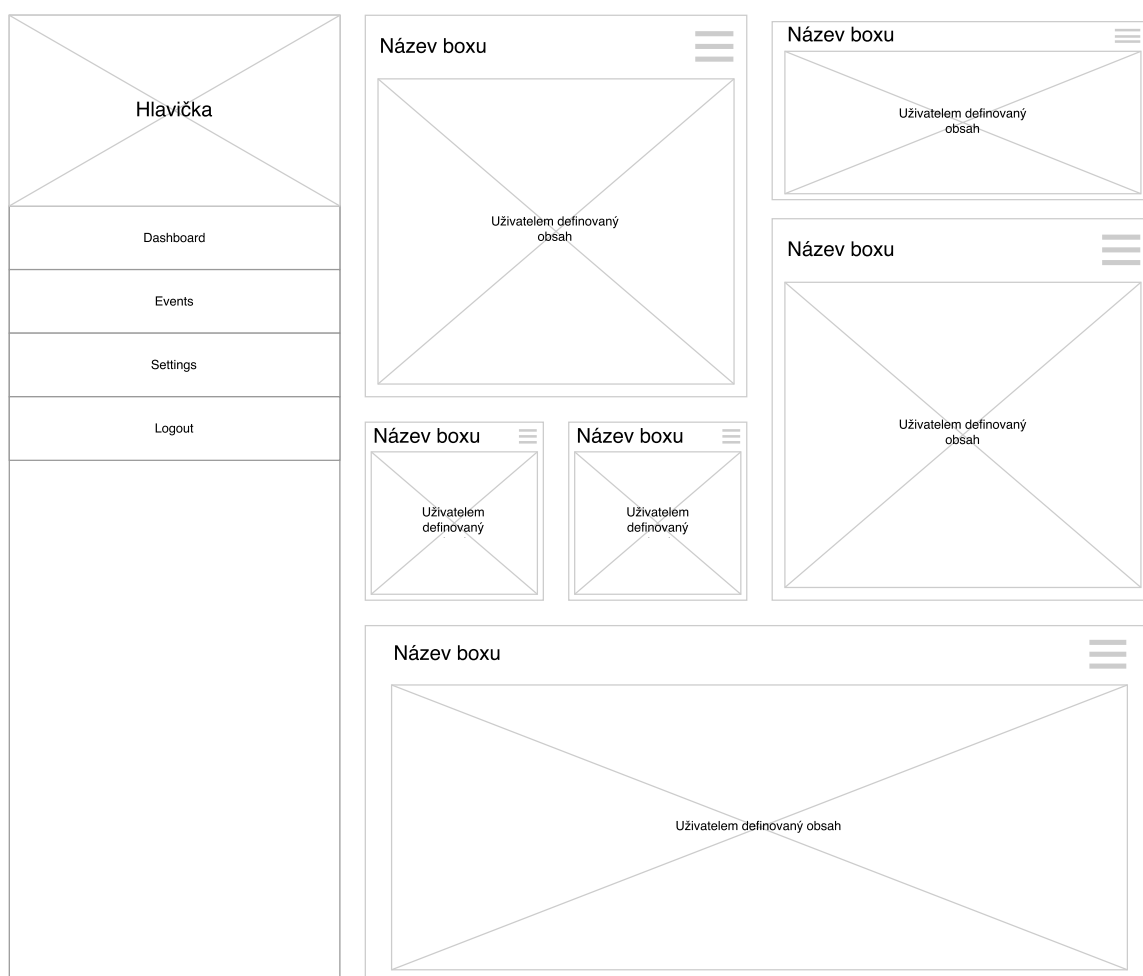


## Seznam příloh

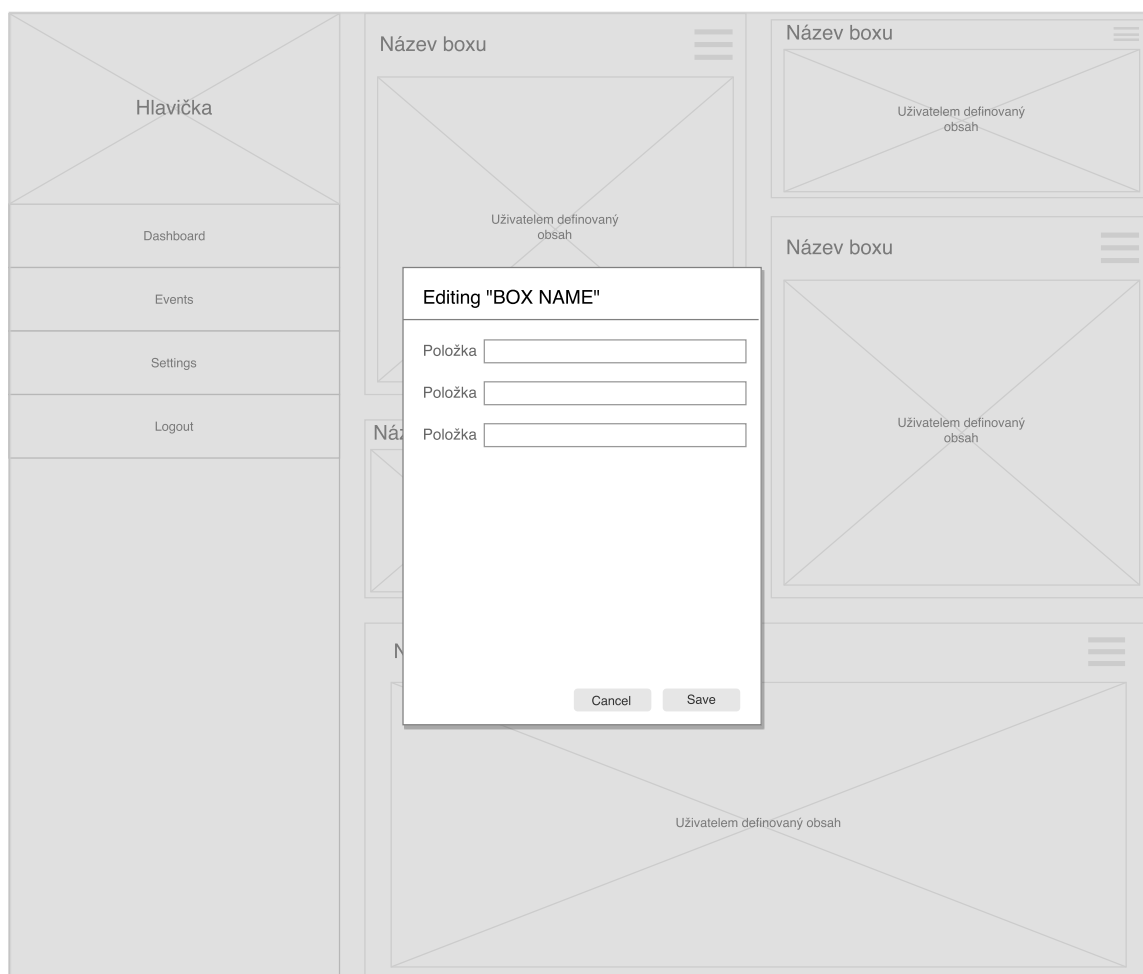
<b>A Drátěné modely</b>	<b>39</b>
<b>B Seznam použitých Python knihoven</b>	<b>42</b>
<b>C Obsah CD</b>	<b>43</b>

# Příloha A

## Drátěné modely



Obrázek A.1: Drátěný model pro dashboard, který uživatel může konfigurovat.



Obrázek A.2: Konfigurace boxu v dashboardu.

Hlavička

Dashboard

Events

Settings

Logout

time from

time to

Description

Source IP

date

category

Destination IP

LOAD

RESET

hlavicka tabulky
záznam události
záznam události
záznam události
záznam události
záznam události
záznam události
záznam události
záznam události
záznam události
záznam události
záznam události
záznam události
záznam události
záznam události
záznam události

Obrázek A.3: Přehled vyfiltrovaných událostí.

## Příloha B

# Seznam použitých Python knihoven

```
Flask==0.10.1
Flask-Cors==2.1.2
itsdangerous==0.24
Jinja2==2.8
MarkupSafe==0.23
py-bcrypt==0.4
pyparser==2.14
PyJWT==1.4.0
pymongo==3.2
six==1.10.0
Werkzeug==0.11.3
```

Listing B.1: Obsah souboru requirements.txt, který využívá nástroj pip pro instalaci Python knihoven.

**Příloha C**

**Obsah CD**