Masarykova univerzita
Fakulta informatiky



Systém pro zpracování provozních statistik videokonferenční infrastruktury

Bakalářská práce

Lukáš Kotol

Brno, jaro 2017

Místo tohoto listu vložte kopie oficiálního podepsaného zadání práce a prohlášení autora školního díla.

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Lukáš Kotol

Vedoucí práce: RNDr. Miloš Liška, Ph.D.

Poděkování

Děkuji mému vedoucímu bakalářské práce RNDr. Miloši Liškovi, Ph.D., za cenné rady a trpělivost při spolupráci. Také bych velmi rád poděkoval své rodině, která mě psychicky podporovala při vypracování mé bakalářské práce.

Shrnutí

Bakalářská práce se zabývá vývojem systému určeného ke zpracování a prezentaci provozních statistik z rezervačního systému Shongo. První část práce se věnuje analýze požadavků a návrhu systému. V druhé části práce jsou popsány všechny technologie, které byly použity při vývoji systému. Třetí část práce obsahuje samotný postup, jenž byl použit při implementaci systému pro zpracování a prezentaci provozních statistik. Čtvrtá a zároveň poslední část této práce líčí, jakým způsobem bylo vypracováno uživatelské prostředí aplikace.

Klíčová slova

statistiky, webová aplikace, videokonference, diagram, rezervační systém Shongo, Adobe Connect, H.323 MCU

Obsah

1 Úvod **1**

2 Návrh aplikace **3**
2.1 *Analýza požadavků* . . . 3
2.2 *Obecný návrh aplikace* . . . 4
2.3 *Návrh struktury datového modelu* . . . 5
2.4 *Návrh uživatelského rozhraní* . . . 7

3 Použité technologie **10**
3.1 *Technologie použité při implementaci zpracování statistik* . . . 10
3.1.1 Java . . . 10
3.1.2 Hibernate . . . 10
3.1.3 PostgreSQL . . . 11
3.2 *Technologie použité při vývoji webové aplikace* . . . 11
3.2.1 PHP . . . 11
3.2.2 Laravel . . . 12
3.2.3 Javascript . . . 13
3.2.4 Highcharts . . . 14
3.2.5 Vue.js . . . 14
3.2.6 XML . . . 15
3.2.7 HTML . . . 16
3.2.8 CSS . . . 16
3.2.9 JSON . . . 16
3.2.10 Ajax . . . 17

4 Popis implementace **18**
4.1 *Popis implementace zpracování statistik* . . . 18
4.1.1 Metody v třídě `AdobeConnectConnector` . . . 18
4.1.2 Metody v třídě `CiscoMCUConnector` . . . 20
4.1.3 Třída `AbstractProcessor` . . . 26
4.1.4 Třída `AdobeConnectProcessor` . . . 27
4.1.5 Třída `CiscoMCUProcessor` . . . 28
4.2 *Popis implementace webového rozhraní* . . . 29
4.2.1 Struktura adresového prostoru . . . 29
4.2.2 Řadiče . . . 30

4.2.3 Modely 32
4.2.4 Šablony 33
4.2.5 Komponenty 33
4.2.6 Diagramy 34

5 Uživatelské prostředí aplikace **36**

6 Závěr **41**

A Příloha **43**

1 Úvod

V současné době moderních multimediálních technologií představuje videokonference pohodlný způsob komunikace obrazem i zvukem mezi dvěma či více účastníky. Zmíněná komunikace je efektivní, neboť zúčastněným osobám šetří čas a náklady na cestování. Díky tomu, že spojení probíhá v reálném čase, představuje téměř plnohodnotnou alternativu k běžné osobní komunikaci. Z pozice těch, co se podílejí na vývoji a údržbě těchto systémů, je vhodné pracovat na tom, aby videokonference probíhaly v reálném čase a zároveň vyhovovaly narůstajícímu počtu uživatelů. Aby to bylo možné, je třeba mít dobrý přehled o stavu využívání videokonferenčních systémů. Abychom získali představu, jakým způsobem je systém používán, nestačí jen data generovaná počítačem. Obvykle jde o data nezpracovaná, která vzhledem ke své nečitelné formě neobsahují pro člověka téměř žádnou informaci. Jako pomůcka slouží strukturovaný diagram, který díky grafickému zobrazení přispěje k objasnění významu zpracovaných dat. Jednou z klíčových předností diagramu je vlastnost umožňující vnímat velké množství statistických údajů, myšlenek a relací s nadhledem.

Cílem této bakalářské práce je navrhnout a implementovat systém pro sběr, zpracování, agregaci a prezentaci provozních statistik videokonferenční a webkonferenční infrastruktury, kterou provozuje sdružení CESNET[1]. Navržený systém je úzce provázaný se stávajícím systémem Shongo [1], nasazeným na meetings.cesnet.cz, který je určený pro rezervaci videokonferečních a webkonferenčních zdrojů. Rezervační systém Shongo umožňuje administrátorům definovat zdroje, např. H.323 MCU[2] nebo Adobe Connect[3] a uživatelům umožňuje rezervaci a následné využití virtuálních místností těchto zdrojů. Virtuální místnosti jsou určeny pro zprostředkování videokonferencí, webových seminářů, sdílení obsahu dokumentů a pracovní plochy počítače.

Automaticky generované provozní statistiky obsahují informace o již proběhlých videokonferencích a jsou dostupné přes aplikační

1. CESNET, `www.cesnet.cz`
2. Multipoint Control Unit, hardwarové zařízení propojující koncové body při videokonferenci
3. Adobe Connect, `www.adobe.com/products/adobeconnect.html`

programové rozhraní (API) jednotlivých prvků videokonferenční infrastruktury. Jelikož statistiky z API pocházejí z různých zdrojů, jsou heterogenní, a tudíž bylo vhodné vytvořit jednotný databázový model, do kterého jsem nestejnorodá data transformoval. Aplikační vrstva systému disponuje logikou, jež umožňuje poskytovat různé pohledy na zpracované statistiky na základě dotazovaných parametrů, např. na základě typu zdroje. Neméně důležitou součástí systému je prezentační vrstva, která je schopna díky použitým knihovnám plně vyhovět požadavkům na moderní uživatelské rozhraní.

Výsledkem mé práce je systém zpracovávající statistiky ze zdrojů rezervačního systému Shongo do databáze s jednotným datovým modelem. Dalším výsledkem je plně funkční moderní webová aplikace, která prezentuje zmíněné zpracované statistiky. Aplikace umožňuje parametrizaci zobrazených údajů v diagramech, vyhledávání a přitom je schopna rychle reagovat na požadavky uživatele. Díky způsobu zpracování grafického rozhraní a použitým diagramům poskytuje aplikace uživatelům komplexní přehled o již proběhlých videokonferenčních spojeních.

Text této práce je členěn následujícím způsobem. Po úvodu následuje druhá kapitola, kde uvádím analýzu požadavků a návrh aplikace. Třetí kapitola seznamuje s použitými technologiemi. Ve čtvrté kapitole uvádím popis implementace. Pátá kapitola popisuje uživatelské prostředí aplikace. Poslední kapitola přináší závěr práce.

2 Návrh aplikace

Před analýzou požadavků je vhodné vysvětlit pojmy, které jsou používány při práci s rozhraním rezervačního systémemu Shongo[1]. Rezervační systém umožňuje vytvářet virtuální místnosti, ve kterých se uskutečňují dílčí sezení. Sezením rozumíme časový interval počínající připojením prvního uživatele do prázdné místnosti a končící odpojením posledního uživatele v místnosti. Pojem spojení vyjadřuje časový interval trvající od připojení do doby odpojení uživatele.

2.1 Analýza požadavků

Ze zadání a společných konzultací s vedoucím práce vyplynuly následující požadavky. První část práce tvoří zpracování provozních statistik do jednotného datového modelu (navržen v kapitole 2.3), které bude implementováno do stávajícího rezervačního systému Shongo (kapitola 4.1). Statistiky jsou uloženy v jazyce XML a pocházejí z již zmíněných zdrojů Adobe Connect a H.323 MCU.

Zdroj H.323 MCU poskytuje statistiky ve formě CDR (Call Detail Record) záznamů, jejichž struktura je tvořena chronologicky seřazenými událostmi, které nastaly na zdroji [2]. Tyto události se dělí podle typu informace, kterou nesou na začátek konference v místnosti, konec konference v místnosti, připojení a odpojení účastníka. K získání celkového přehledu ovšem stačí brát v potaz jen události s daty o odpojení účastníka, jelikož poskytují všechny potřebné informace. Klíčové informace v tomto typu události jsou pořadové číslo události, okamžik vzniku události, identifikátor účastníka a délka setrvání ve videokonferenci.

Naproti tomu zdroj Adobe Connect obsahuje statistiky z videokonferencí již organizovaně dle jednotlivých místností a sezení, což ulehčuje transformaci do jednotného datového modelu. Struktura záznamů dat z tohoto zdroje není jednotná, jelikož jsou statistiky poskytovány v závislosti na parametrizaci při používání URL rozhraní. Prostřednictvím URL rozhraní lze spouštět dotazy, jež v odpovědi obsahují požadovaná data a informaci o tom, zda bylo vyhodnocení

1. Nápověda rezervačního systému Shongo, `https://meetings.cesnet.cz/help`

dotazu úspěšné. Struktura dat v odpovědi závisí na typu dotazu, který je spuštěn. Například pokud chceme získat seznam místností se sezeními, odpověď na požadavek získání všech místností vrátí jen seznam se jmény, URL a unikátními identifikátory místností. Pokud potom chceme získat informace o sezeních vybrané místnosti, použijeme dotaz k získání všech sezení místnosti s parametrem obsahující unikátní identifikátor místnosti. K získání přehledu o využití zdroje, který lze namapovat na datový model aplikace, postačí dotazy na výpis všech místností, seznam jejich sezení a intervaly těchto sezení.

Aby získaná data mohla být jednotně zobrazena v prezentační vrstvě, musel být navržen obecný datový model, do kterého jsou statistiky transformovány. Vzhledem k množství zpracovaných dat, čítajících desetitisíce záznamů, je zpracování statistik navrženo a implementováno tak, aby bylo časově efektivní, jelikož bude probíhat několikrát denně.

Druhou část systému tvoří samotná webová aplikace využívající homogenizovaná data zpracovaná první částí systému s důrazem na parametrizaci zpracovaných statistik (implementována v kapitole 4.2). Aplikace bude schopna rychle zobrazovat data, díky použitým agregacím při zpracování statistik, i přes velké množství záznamů. Webová aplikace bude ke každému zdroji poskytovat přehled o míře jeho vytíženosti. Dále bude přehled zdroje obsahovat seznam všech místností spolu s daty o jejich využití.

Při analýze požadavků byl kladen důraz na transformaci statických dat na dynamickou webovou aplikaci, která umožňuje podávat uživateli různé pohledy na zpracované provozní statistiky, tak aby se v co největší míře využil potenciál diagramů. Takto navržený systém dokáže po implementaci poskytnout informace v kontextu a s nadhledem, oproti běžně generovaným záznamům, jejichž význam nemusí být na první pohled zřejmý.

2.2 Obecný návrh aplikace

V této sekci uvádím, jak bude koncipován celkový návrh aplikace. První část implementace bude tvořit program s nekonečnou smyčkou, která periodicky zpracuje a uloží do databáze statistiky ze zdrojů, kde probíhaly videokonference. Každý zdroj má ovšem jiné XML roz-

hraní pro dotazy nad jeho záznamy. Ke specifikaci rozhraní zdroje H.323 [3] a Adobe Connect [4] poslouží uvedené dokumentace. Jelikož bude program na zpracování statistik úzce propojen s rezervačním systémem Shongo, je vhodné využít již částečně implementované konektory ke zmíněným zdrojům. Konektory jsou třídy umožňující ovládat konkrétní zdroje v rezervačním systému Shongo. Jejich výhodou je, že usnadňují tvorbu dotazů nad zdroji a zpracování následných výsledků v XML jazyce pomocí svého rozhraní. Prostřednictvím zmíněných konektorů bude program získávat provozní statistiky zdrojů Adobe Connect a H.323 MCU. Tato funkcionalita není v konektorech dosud implementována, proto ji bude nutné v rámci této práce implementovat. Po zpracování záznamů z odlišných zdrojů do jednotné formy programem bude následovat uložení statistik do databáze podle navrženého datového modelu v kapitole 2.3.

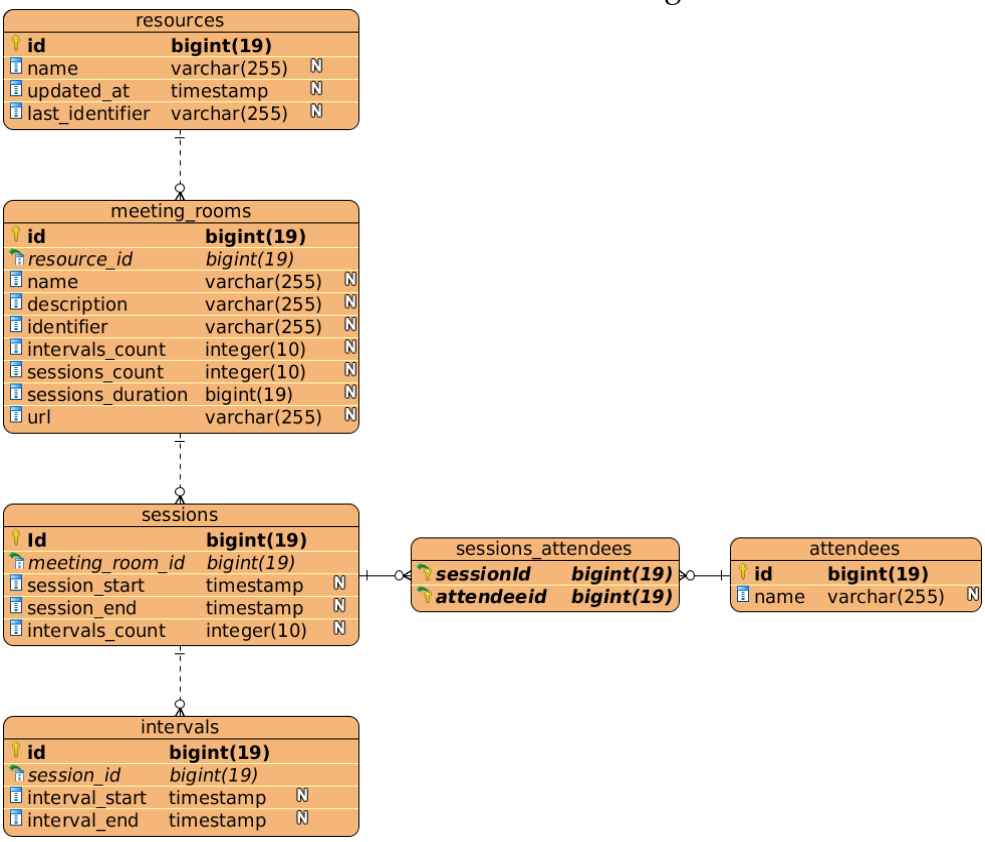
Další část implementace bude spočívat ve vytvoření webové aplikace, která poběží na serveru a bude odpovídat na požadavky uživatelů. Zmíněná webová aplikace bude poskytovat uživatelům přístup ke zpracovaným statistikám v grafické podobě pomocí diagramů a tabulek.

2.3 Návrh struktury datového modelu

Zde se budu věnovat otázce, jakým způsobem je koncipován datový model aplikace. Jak již bylo uvedeno výše, rezervační systém Shongo umožňuje vytvářet virtuální místnosti. Z hlediska využití těchto virtuálních místností je vhodné organizovat záznamy o uskutečněných spojeních také dle místností. Pokud chceme dále získat přehled o využití jednotlivých místností v čase, je žádoucí agregovat spojení uživatelů do sezení. K získání detailu sezení potom poslouží seznam jeho dílčích spojení.

Na základě popisu z předchozího odstavce jsem navrhl následující UML (Unified Modeling Language) entitně relační diagram databáze aplikace.

Obrázek 2.1: Entitně relační diagram



V obrázku diagramu 2.1 zdroj představuje entita **resources**, která si uchovává jméno a časovou značku poslední aktualizace dat. Časová značka umožní získat informaci o stáří statistik. Dále entita obsahuje identifikátor posledního zpracovaného intervalu, tak aby v případě ukončení běhu programu mohl být znovu spuštěn a začít zpracovávat statistiky od posledního zpracovaného intervalu.

Entita **meeting_rooms** reprezentuje virtuální místnost. Informace obsažené v této entitě jsou název místnosti, unikátní identifikátor pocházející ze zdroje, počet intervalů a sezení místnosti, součet doby všech sezení. Navíc entita **meeting_rooms** obsahuje volitelné položky popis a webová adresa místnosti.

Další entita **sessions** náleží místnosti a představuje sezení. Tato entita obsahuje data o začátku a konci sezení a počet intervalů v sezení.

Entita **intervals** reprezentuje interval v daném sezení a poskytuje informaci o svém začátku a konci.

Účastník videokonference je popsán entitou **attendees**, která si uchovává jen jméno účastníka.

Poslední entita **sessions_attendees** spojuje entity **sessions** a **attendees**, tak aby účastník mohl patřit do více sezení a do sezení mohlo patřit více účastníků.

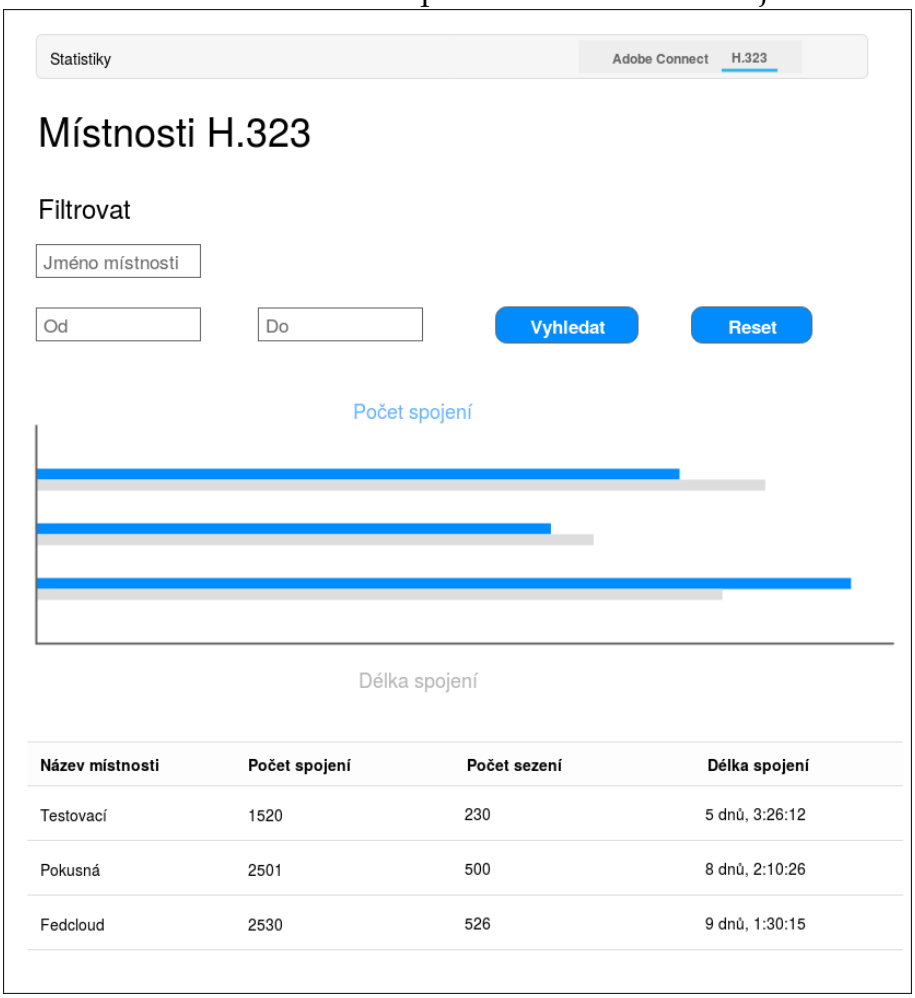
Do uvedeného datového modelu jsou statistiky ze zdrojů H.323 MCU a Adobe Connect mapovány pomocí nově implementovaných metod v konektorech rezervačního systému Shongo (kapitola 4.1.1 a 4.1.2).

2.4 Návrh uživatelského rozhraní

Nyní se zmíním o tom, jakým způsobem je navrženo uživatelské rozhraní webové aplikace. Webové rozhraní bude responzivní, tak aby bylo optimalizováno pro různá zařízení. Aby aplikace působila moderně, ovládací prvky uživatelského rozhraní budou designově konzistentní a bude je možno intuitivně ovládat.

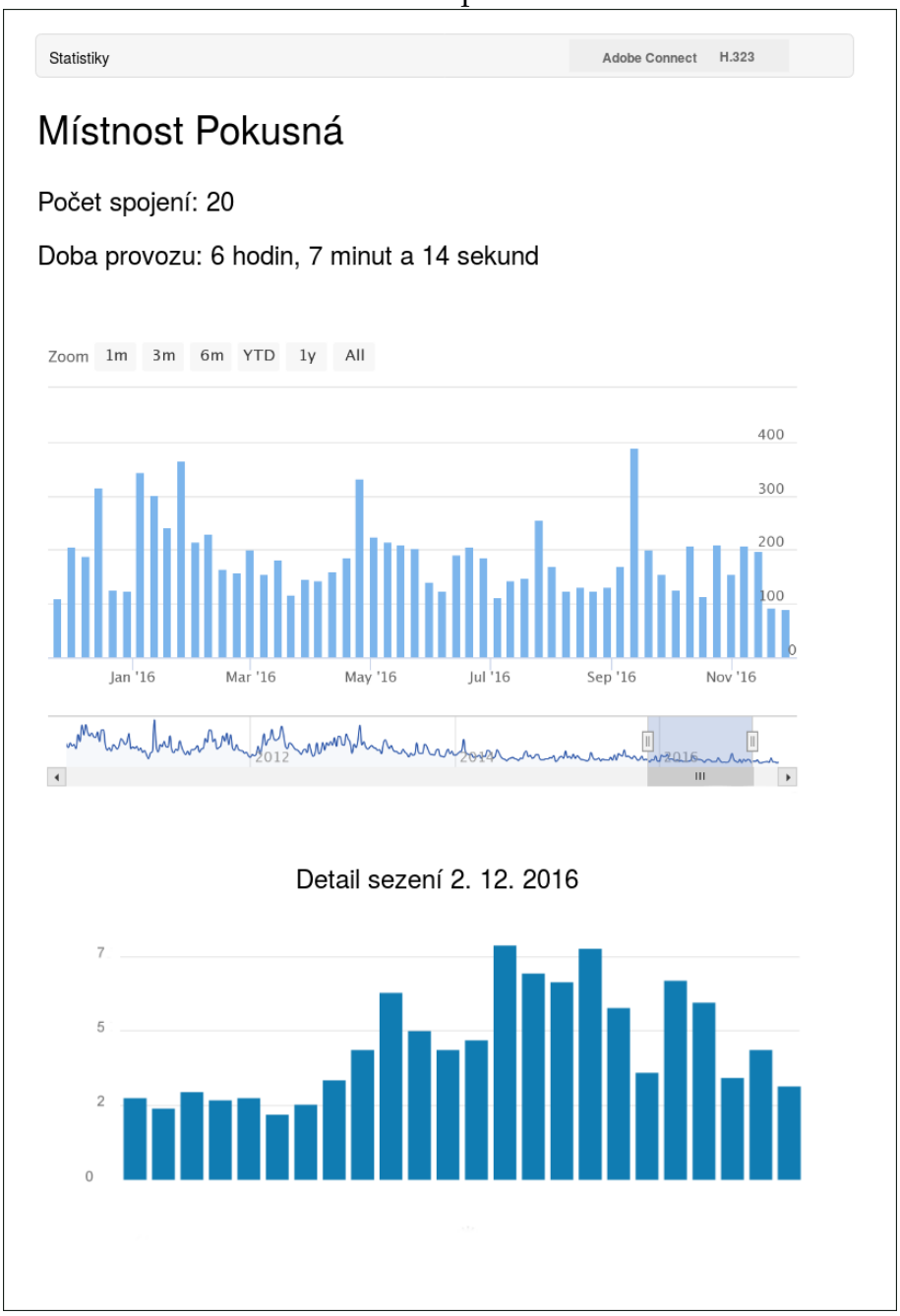
Obsah webové aplikace bude tvořit přehled jednotlivých zdrojů rezervačního systému Shongo s možností zobrazení detailu zdroje. Detail zdroje bude sestávat z agregovaných statistik zdroje, diagramu a tabulky, kde budou data členěna dle místností. Data v tabulce a diagramu spolu budou korespondovat. Místnosti v tabulce a diagramu bude možno filtrovat podle názvu a zvoleného období, kdy probíhaly videokonference v místnosti. Dále bude umožněno řadit místnosti vzestupně a sestupně podle zobrazených sloupců v tabulce. Uvedený návrh je ilustrován pomocí obrázku 2.2.

Obrázek 2.2: Návrh přehledu místností zdroje



Webová aplikace bude dále poskytovat podrobné informace o jednotlivých místnostech, kde zobrazí agregovaná data o počtu sezení a dobu využití místnosti. Přehled místnosti bude disponovat diagramem se všemi sezeními, která se odehrála v místnosti. V přehledu místnosti bude vložen další diagram, který bude podrobně zachycovat průběh jednotlivých sezení. Zmíněný přehled je graficky navržen v obrázku 2.3. Výsledná podoba grafického uživatelského rozhraní webové aplikace je popsána v kapitole 5.

Obrázek 2.3: Návrh přehledu místnosti



3 Použité technologie

3.1 Technologie použité při implementaci zpracování statistik

V této kapitole popisuji technologie, které jsem použil při implementaci mé bakalářské práce. Popis je krátce doplněn o zdůvodnění, proč jsem danou technologii použil.

3.1.1 Java

Java je silně typovaný objektově orientovaný programovací jazyk, jehož syntaxe je do značné míry odvozená z C++. Jednou z hlavních výhod Javy je nezávislost na architektuře počítače, jelikož vytváří místo skutečného strojového kódu pouze mezikód. Pro interpretaci zmíněného mezikódu se potom používá virtuální stroj JVM (Java Virtual Machine). Další výhodou Javy je, že díky automatické správě paměti, pomocí tzv. Garbage collectoru, osvobozuje programátora od uvolňování paměti.

Tyto a další vlastnosti řadí Javu k nejčastěji používaným programovacím jazykům vůbec[1]. Díky rozšířené komunitě a velké oblibě Javy existuje velké množství knihoven, které nabízejí pomoc při řešení typických problémů při vývoji programů.

Hlavním důvodem výběru tohoto jazyka byl požadavek, aby výsledný systém starající se o kontinuální zpracování statistik (implementován v kapitole 4.1) byl úzce provázán s rezervačním systémem Shongo, který je také napsán v jazyce Java.

3.1.2 Hibernate

Hibernate je aplikační rámec Javy, který slouží k objektově-relačnímu mapování entit. Jeho hlavním přínosem je tedy automatická konverze dat mezi objektově orientovaným jazykem Java a relační databází. Hibernate umožňuje vývojářům vytvářet persistentní objekty s využitím konceptů jako dědičnost, polymorfismus, asociace a kompozice.

1. Indikátor popularity programovacího jazyka Tiobe Index, `www.tiobe.com/tiobe-index`

Komunikace s databází potom probíhá s použitím API, které je transformováno do SQL příkazů. Mimo klasické mapování pomocí XML souborů umožňuje Hibernate intuitivní mapování anotacemi přímo v entitách určených k uložení do databáze.

Ve své práci jsem tuto technologii použil, jelikož velkou měrou usnadňuje transformaci dat z objektového paradigmatu do entitně-relační struktury dat.

3.1.3 PostgreSQL

PostgreSQL [5] je výkonný objektově-relační databázový systém s otevřeným zdrojovým kódem. Komunikace mezi programátorem a systémem PostgreSQL probíhá pomocí jazyka SQL (Structured Query Language). Data v databázi jsou uchovávána v tabulkách, které disponují pojmenovanými sloupci o různých datových typech a jejich velikostech. Samotné záznamy jsou pak uloženy v jednotlivých řádcích tabulek. PostgreSQL umožňuje reprezentovat vztahy mezi tabulkami pomocí primárních a cizích klíčů. Zmíněný systém dále umožňuje definovat spouště, pohledy, uživatelské datové typy, referenční integritu a provádět transakce. PostgreSQL disponuje rozsáhlou online dokumentací [6] zahrnující manuál, manuálové stránky a příklady.

3.2 Technologie použité při vývoji webové aplikace

3.2.1 PHP

PHP (Hypertext Preprocessor) [7] je slabě typovaný skriptovací programovací jazyk určený především pro tvorbu dynamických webových stránek. Syntax jazyka PHP je inspirována programovacími jazyky Perl, C, Java a Pascal. PHP je široce oblíbený jazyk s otevřeným zdrojovým kódem, který díky své jednoduchosti a značné zásobě funkcí má podporu velké komunity. To potvrzuje i fakt, že na straně serveru nejvíce současných webových stránek používá PHP.[2] Jedním z důvodů oblíbenosti PHP je podpora objektového orientovaného programování, nativní podpora různých databázových systémů a rozsáhlá dokumentace.

2. Přehled míry využití programovacích jazyků na straně serveru , `www.w3techs.com/technologies/overview/programming_language/all`

V tomto jazyce jsem napsal webovou aplikaci, která prostřednictvím svého rozhraní poskytuje přehled o využití jednotlivých zdrojů rezervačního systému Shongo (popis implementace v kapitole 4.2). PHP v aplikaci zajišťuje veškerou vnitřní logiku, směrování a v neposlední řadě zprostředkuje uživateli data z databáze. Vybral jsem si tento prostředek, protože jeho použití je velice snadné a přitom poskytuje dostatečné nástroje k tvorbě plnohodnotných webových stránek.

3.2.2 Laravel

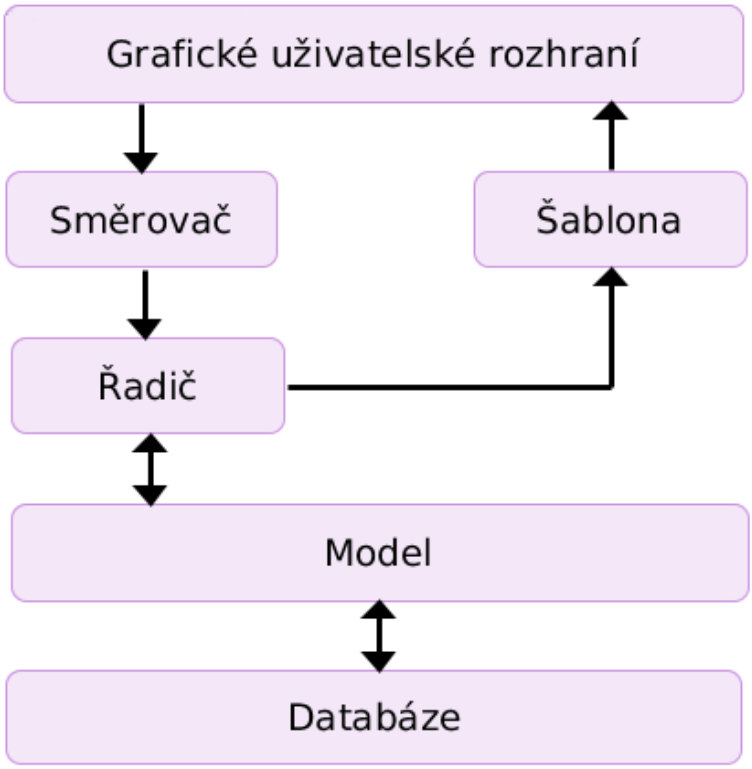
Laravel [8] je webový aplikační rámec s otevřeným zdrojovým kódem, napsaný v jazyce PHP. Tento nástroj usnadňuje řešení typických problémů při vývoji webových aplikací. Laravel využívá softwarové architektury MVC (Model-View-Controller). V architektuře MVC se model aplikace stará o aplikační data a práci s databází, šablony (View) prezentují data uživateli a řadič (Controller) reaguje na požadavky uživatele. Aplikační rámec Laravel je navržen tak, aby v případě potřeby řešil automatizovaně následující agendu:

- správa směrování požadavků,
- objektově-relační mapování a nástroje pro správu databáze,
- znovupoužití a rozšíření zobrazovaných šablon,
- správa autentizace uživatelů,
- agenda logování, testování, odesílání mailů a session.

Průběh obsluhování uživatelského požadavku Laravelem je následující: Typická interakce se serverem začíná, když uživatel navštíví jednu z URI adres definovaných v aplikačních adresách. Prohlížeč odešle požadavek, který obdrží webový server a předá jej směrovači. Směrovač přijme požadavek a přesměruje jej na vhodný řadič. Každá adresa definovaná ve směrovači koresponduje s vybraným řadičem, jenž převezme kontrolu nad během aplikace. Řadič pak interaguje s databází pomocí modelu a svůj běh ukončí vrácením odpovědi. Odpovědí většinou rozumíme vykreslení zvolené šablony, do které se vloží data zpracovaná v řadiči. Nakonec je šablona zobrazena uživateli

a aplikace je připravena k obsloužení dalšího požadavku. Zmíněný postup při obsluhování požadavku ilustruje obrázek 3.1.

Obrázek 3.1: Architektura webové aplikace Laravel



Komponenty Laravelu jsou administrovány pomocí manažeru balíků Composer[3], tudíž je možné rozšířit aplikaci o další moduly.

3.2.3 Javascript

Javascript je vysokoúrovňový objektově orientovaný skriptovací jazyk určený pro tvorbu interaktivních webových stránek. Obvykle se používá k ovládání prvků grafického uživatelského rozhraní. Syntaxe je inspirována jazyky Java, C a C++ a její zvládnutí je relativně jednoduché. Jelikož v historii vzniklo několik verzí Javascriptu, byl standardizován pomocí normy ECMAScript [9].

3. Composer, `www.getcomposer.org`

Výhodou jazyka je jeho velká rychlost. Javascript je rychlý, protože přímo interpretuje svůj kód a nevyžaduje kompilaci. Předností tohoto jazyka je také fakt, že běh jeho programů probíhá na straně klienta, a tudíž šetří síťové zdroje. Další výhodou Javascriptu je platformní nezávislost, jelikož ke běhu programu stačí běžný prohlížeč. Nejen díky těmto vlastnostem zaujal rozsáhlou komunitu vývojářů, která mezi sebou sdílí množství knihoven a balíčků. Pokud chceme využít potenciál těchto knihoven, musíme je vložit do svého kódu. Avšak s rostoucím počtem knihoven v aplikaci se může práce jevit jako nepřehledná, proto je vhodné používat k usnadnění práce s knihovnami manažer balíků NPM[4].

V tomto programovacím jazyku jsou ve webové aplikaci napsány veškeré interaktivní prvky grafického uživatelského rozhraní včetně vyobrazení diagramů.

3.2.4 Highcharts

Highcharts je javascriptová knihovna, jejíž pomocí lze vytvářet interaktivní diagramy. Tato knihovna je poskytována zdarma pro nekomerční použití a je možné si k ní dopsat další moduly. Knihovna nabízí široké spektrum typů diagramů zahrnujících sloupcový, bodový, čárový, výsečový, pruhový a jejich různé kombinace. Nastavení a implementace diagramů je velice jednoduchá, a to nejen díky rozsáhlé dokumentaci [10]. Mezi další přednosti Highcharts patří možnost exportu dat z grafu přímo do samostatných souborů PNG, JPG, PDF a SVG.

Knihovnu Highcharts jsem použil při implementaci všech diagramů, které se nacházejí ve webové aplikaci. Vybral jsem si tento nástroj, jelikož jeho použití je velice jednoduché a poskytuje rozhraní, pomocí kterého lze diagramy nakonfigurovat do detailu.

3.2.5 Vue.js

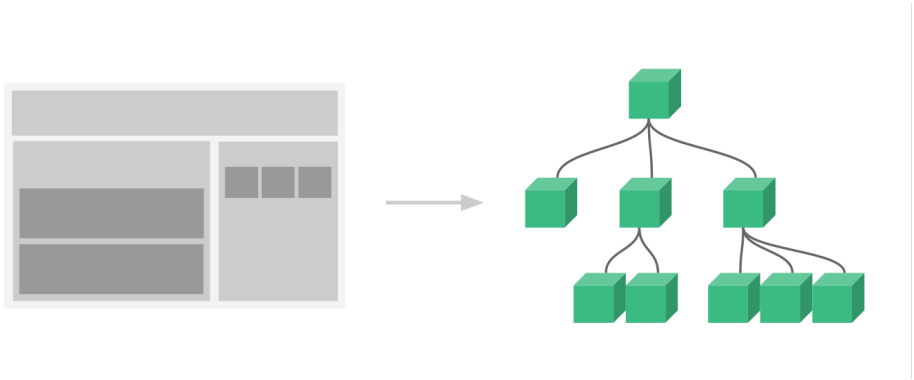
Vue.js (verze 2) je javascriptový aplikační rámec s otevřeným zdrojovým kódem, určený pro vytváření interaktivních prvků uživatelského rozhraní. Na rozdíl od ostatních monolitických aplikačních rámců je Vue.js vhodný k použití i v jednoduchých aplikacích a zároveň přináší výhody jednostránkových aplikací (Single Page Application). Hlavní

4. Node Package Manager, `www.npmjs.com/`

přínos Vue.js však je v poskytnutí benefitů reaktivního vázání dat v komponentách, které se mohou opětovně použít. Další výhodou Vue.js je fakt, že disponuje přehlednou online dokumentací [11].

Architektura Vue.js je založena na malých znovupoužitelných komponentách, které mohou interagovat mezi sebou a tvořit tak komplexnější aplikaci. Na takto vytvořené webové rozhraní lze potom nahlížet jako na hierarchickou stromovou strukturu, jejíž prvky jsou do sebe komponovány. Uvedená myšlenka je ilustrována obrázkem 3.2.

Obrázek 3.2: Abstrakce rozhraní aplikace do stromové struktury komponent, `www.vuejs.org/v2/guide/#Composing-with-Components`



Při implementaci webové aplikace jsem pomocí Vue.js vytvořil vyhledávání a filtrování místností v přehledu zdroje.

3.2.6 XML

XML (Extensible Markup Language) je obecný značkovací jazyk, který se vyvinul z jazyka SGML (Standard Generalized Markup Language). Jazyk XML obvykle slouží jako formát pro výměnu dat v prostředí internetu. Hlavní výhoda XML je snadná čitelnost jak uživatelem, tak i počítačovým programem. Další předností tohoto jazyka je, že lze snadno transformovat do jiných formátů.

Důvodem použití XML v této práci je fakt, že uložené statistiky ve zdrojích H.323 MCU a Adobe Connect jsou poskytovány právě ve formátu XML.

3.2.7 HTML

HTML (HyperText Markup Language) je standardní značkovací jazyk určený pro tvorbu webových stránek umožňující publikaci dokumentů na internetu. HTML je definován přesnými syntaktickými pravidly a slouží k popisu sémantické struktury webové stránky. Základním kamenem HTML jsou značky, pomocí kterých můžeme vytvářet text, obrázky, formuláře a další webový obsah. Velice důležitým prvkem jsou hypertextové odkazy umožňující propojení webových stránek. Použití HTML je velice jednoduché a je to jeden z fundamentálních nástrojů pro vytváření webových aplikací.

3.2.8 CSS

CSS (Cascading Style Sheets) je formálně definovaný jazyk, jímž lze popisovat způsob zobrazení elementů v dokumentech napsaných v (X)HTML nebo XML. Hlavním smyslem jazyka CSS je oddělení definice vzhledu dokumentu od jeho struktury a obsahu. Oddělení vzhledu od obsahu přináší nejen jednodušší vývoj, ale i následnou údržbu webové prezentace.

Abychom při vývoji webových stránek nemuseli vždy vytvářet vlastní styl, existuje velké množství volně dostupných knihoven s předdefinovaným CSS stylem. Ve své práci jsem použil CSS knihovnu Materializecss[5], která nabízí moderní a uniformní design podporující zobrazení obsahu na různých zařízeních.

3.2.9 JSON

JSON (JavaScript Object Notation) je jednoduchý textový formát určený především pro přenos dat. Díky jednoduchému způsobu zápisu je čitelný pro člověka, ale i snadno strojově zpracovatelný. Nespornou výhodou je fakt, že zpracování formátu JSON je podporováno ve velké většině běžných programovacích jazyků. Při implementaci jsem JSON použil jako datový formát pro asynchronní komunikaci mezi klientem a serverem.

5. Materializecss, `www.materializecss.com`

3.2.10 Ajax

Ajax (Asynchronous JavaScript and XML) je množina nástrojů, které slouží pro tvorbu interaktivních webových aplikací. Hlavním přínosem spojení Javascriptu a XML je možnost tvorby webových aplikací, které mění svůj obsah bez nutnosti jejich kompletního opětovného načtení. K výměně dat mezi serverem a klientem se místo XML může použít i prostý text, HTML nebo JSON.

Při implementaci webové aplikace jsem technologii Ajax použil pro asynchronní komunikaci s webovým serverem. Zmíněná komunikace slouží k získání dat o detailu videokonferenčního sezení z databáze a následného zobrazení těchto dat v diagramu.

4 Popis implementace

4.1 Popis implementace zpracování statistik

V této kapitole seznamuji s detailním postupem při implementaci systému, jehož úkolem je zpracování a persistence statistik. V kapitole 4.1.1 a 4.1.2 popisuji implementaci nových metod do konektorů nacházejících se v rezervačním systému Shongo v modulu `shongo-connector`, které slouží k získání statistik z příslušných zdrojů. Prostřednictvím nově implementovaných metod v konektorech jsou data ze zdrojů mapována na jednotný datový model (navržen v kapitole 2.3). V následujících sekcích uvádím implementaci tříd `AbstractProcessor`, `AdobeConnectProcessor` a `CiscoMCUProcessor`, které mají za úkol uložení získaných statistik do databáze. Uvedené třídy se nacházejí v modulu `shongo-statistics` a jejich použití je ilustrováno v spustitelném souboru `src/main/java/Main.java`.

4.1.1 Metody v třídě `AdobeConnectConnector`

`AdobeConnectConnector` je třída, která využívá webové služby zdroje Adobe Connect a pomocí webového aplikačního rozhraní poskytuje přístup k jeho datům. K získání statistik ze zdroje je použit GET HTTP (Hypertext Transfer Protocol) požadavek na webovou adresu `https://connect.cesnet.cz/api/xml`. Pro spuštění konkrétního dotazu musí být k adrese připojen řetězec ve tvaru `action=akce&parameter=hodnota_parametru`, kde `akce` je název požadavku, který chceme spustit, a za ním následuje seznam parametrů oddělený znakem &. Tímto způsobem lze jak získávat, tak modifikovat data ze zdroje Adobe Connect. Obsáhlá specifikace uvedeného rozhraní je popsána v dokumentaci [4] webové služby Adobe Connect.

V uvedené třídě `AdobeConnectConnector` jsem implementoval následující metody. Nejdůležitější metodou je `ResourceApi getResourceStatistics(ResourceApi resource)`. Hlavním úkolem této metody je mapování všech dat ze zdroje AdobeConnect do entity `ResourceApi` představující objektovou abstrakci zdroje. Na začátku metody je získán seznam všech místností pomocí dotazu `report-bulk-objects` s atributem `filter-type=meeting`. Jelikož se-

znam obsahuje objekty typu `Element` z třídy `org.jdom2.Element` a statistiky musejí být uloženy v jednotném datovém modelu, každý objekt `Element` je příslušným způsobem mapován do objektu třídy `RoomApi`. Třída `RoomApi` představuje objektovou abstrakci virtuální místnosti a uchovává obecné informace, jako je název místnosti, unikátní identifikátor v rámci zdroje Adobe Connect a seznam sezení v místnosti. Všechny zmíněné položky až na seznam sezení v místnosti lze získat přímo z posledního uvedeného dotazu. Pro získání zmíněného seznamu sezení slouží metoda popsaná v dalším odstavci. Nakonec metoda poznamená časovou značku poslední aktualizace dat do objektu `RoomApi` a vrátí jej.

K získání seznamu všech sezení místnosti slouží další implementovaná metoda `List<RoomSession> getMeetingSessions(RoomApi room)`. Sezení je reprezentováno entitou `RoomSession`, jež ve svých atributech obsahuje interval sezení, seznamy dílčích spojení a uživatelů, kteří se zúčastnili sezení. Na začátku metoda spustí dotaz na aplikační rozhraní `report-meeting-sessions` s atributem `sco-id=room-id`, kde `room-id` je unikátní identifikátor místnosti v rámci zdroje Adobe Connect. Uvedený dotaz získá seznam všech sezení ve zvolené místnosti. Dalším krokem algoritmu je spuštění dotazu `report-meeting-attendance` s totožným atributem z posledního dotazu na aplikační rozhraní. Spuštěný dotaz vrátí seznam všech spojení v místnosti do vybrané proměnné. Nyní jsou všechna potřebná data ze zdroje Adobe Connect připravena a algoritmus je může zpracovat. Dále algoritmus prochází seznam sezení a z každého prvku vytvoří odpovídající entitu `RoomSession`. Takto vytvořené entity jsou přidány do výsledného seznamu sezení, který nakonec metoda vrátí.

Při procházení seznamu všech sezení uvedeném v předešlém odstavci je opakovaně spouštěna poslední implementovaná metoda `List<RoomInterval> getSessionIntervals(List<Element> intervalsElements, RoomSession roomSession)` ve třídě AdobeConnectConnector. Hlavní funkcí této metody je vrácení seznamu spojení náležejících k vybranému sezení. Metoda dostane na vstupu seznam objektů typu `Element` představující spojení z celé místnosti a proměnnou `roomSession` reprezentující sezení. Na začátku metody je vytvořen prázdný seznam, do kterého budou vkládána spojení v sezení `roomSession`. Elementy reprezentující jednotlivá spojení obsahují atribut `asset-id`, který odkazuje na příslušné sezení. Díky tomuto

atributu je jednoduché identifikovat, jaká spojení patří do konkrétního spojení. Následně je procházen seznam `intervalsElemets` a při každém průchodu je kontrolována shodnost atributu `asset-id` se stejně nazvaným atributem u entity `RoomSession`. Jestliže je hodnota `asset-id` shodná, vytvářejí se instance objektu reprezentující spojení, do kterého se uloží začátek a konec spojení. Následně se spojení vloží do připraveného seznamu, který byl vytvořen na začátku metody. Jelikož informace o účastníkovi sezení jsou obsažena až na úrovni jednotlivých spojení, tak při průchodu seznamu `intervalsElements` se zároveň vytvářejí objektové reprezentace účastníků videokonference. Objekty představující účastníky se ukládají jako seznam v objektu `roomSession`.

4.1.2 Metody v třídě `CiscoMCUConnector`

`CiscoMCUConnector` je třída poskytující spojení k rozhraní zdroje MCU H.323. Její hlavní funkcí je komunikace se zdrojem MCU H.323 pomocí protokolu XML-RPC[1], který kombinuje značkovací jazyk XML a komunikační protokol HTTP. K získání statistik ze zdroje je nutné spustit POST požadavek se zvoleným dotazem na webovou adresu `http://mcuc.cesnet.cz/RPC2`, kde je rozhraní zdroje přístupné. V implementaci jsou použity dotazy `cdrlog.query` a `cdrlog.enumerate`. Dotaz `cdrlog.query` slouží k výpisu obecných informací o statistikách (např. celkový počet záznamů) a `cdrlog.enumerate` slouží k získání již konkrétních záznamů ze statistik. Kompletní specifikace, jak pracovat s rozhraním zdroje H.323, je popsána v referenční příručce Cisco TelePresence MCU API guide 2.9 [3].

V třídě `CiscoMCUConnector` jsem implementoval následné metody. Nejdůležitější implementovanou metodou je `ResourceApi getResourceStatistics(ResourceApi resource)`. Zmíněná metoda je jako jediná veřejná a slouží k získání statistik počínaje od posledního zpracovaného záznamu. Získané statistiky ze zdroje MCU H.323 jsou mapovány do objektu `resource`. Na začátku metody se spustí dotaz `cdrlog.query`, jenž vrátí identifikátor prvního spojení ve zdroji a celkový počet položek, které zdroj obsahuje. Tyto informace se uloží do proměnných `firstIndex` a `numEvents`. Následně je testo-

1. XML-RPC, Extensible Markup Language-Remote Procedure Call

váno, zda zdroj `resource` obsahuje poslední zpracovaný identifikátor spojení. Jestliže ano, tak se nastaví hodnota proměnné `firstIndex` na identifikátor posledního zpracovaného spojení a do proměnné `numEvents` se přiřadí zbývající počet nezpracovaných událostí. Po tomto kroku algoritmus spustí `while` cyklus, který iteruje, dokud je proměnná `numEvents` představující počet nezpracovaných elementů větší než nula. Na začátku každé iterace zmíněného cyklu se vždy na XML-RPC rozhraní spustí dotaz `cdrlog.enumerate` s atributem `index` mající hodnotu proměnné `firstIndex`. Uvedený dotaz vrátí do proměnné `events` seznam 20 událostí, počínaje událostí s identifikátorem `firstIndex`. Jelikož 20 vrácených událostí je dáno jako maximální počet událostí, které je zdroj schopen vrátit při jediném dotazu `cdrlog.query`, musí být použit tento dotaz opakovaně v cyklu. Následně se prochází seznam událostí v `events`, přičemž při každé iteraci je proměnná `numEvents` dekrementována o 1.

Jestliže je při procházení seznamu událostí nalezena událost typu odpojení účastníka, spustí se následující sekvence příkazů. Pomocí metody `RoomApi findOrCreateRoom(Map<String, Object> event, List<RoomApi> rooms)`, která je popsána na straně 25, najdeme místnost, z níž se účastník odpojil. Pokud je událost odpojení účastníka v dané místnosti nalezena poprvé, je vytvořena nová místnost. Jelikož událost odpojení účastníka obsahuje informace o délce připojení a času vzniku události, je prostřednictvím pomocné metody `Interval createInterval(Map<String, Object> event)` vytvořen nový objekt `interval` typu `org.joda.time.Interval`.

V dalším kroku následuje vytvoření objektu `newSession` reprezentující sezení. Tento nově vytvořený objekt `newSession` bude v další implementované metodě `RoomSession createSession(RoomApi room, RoomSession newSession, Interval interval)` upraven. Objekty typu `RoomSession`, jež představují sezení, obsahují informace o celkovém časovém rozsahu sezení (atribut `interval`) a seznam všech spojení, jež patří do daného sezení (atribut `intervals`). Zmíněné atributy jsou v sezení nastaveny iniciální hodnotou `interval`. Dále je v cyklu procházejícím získané události vytvořen objekt `roomAttendee` typu `RoomAttendee` představující účastníka videokonference. Jméno účastníka je získáno z položky události `h323Alias` a následně je účastník přidán do seznamu účastníků sezení patřícího objektu `newSession`. Nakonec iterace přes jednotlivé udá-

losti je do seznamu sezení proměnné `room` přidáno sezení modifikované pomocí metody `RoomSession createSession(RoomApi room, RoomSession newSession, Interval interval)`. Jakmile je cyklus procházející přes všechny získané události dokončen, je upravena hodnota proměnné `firstIndex` na identifikátor poslední zpracované události. V další iteraci `while` cyklu se díky aktualizaci proměnné `firstIndex` pokračuje opětovným spuštěním dorazu `cdrlog.query` již od poslední zpracované události. Jakmile počet nezpracovaných událostí v proměnné `numEvents` klesne pod jedna, je `while` cyklus ukončen. Po ukončení cyklu se zdroji v proměnné `resource` nastaví poslední zpracovaný identifikátor události, aktuální časová značka a technologie zdroje. Nakonec metoda v návratové hodnotě vrátí proměnnou `resource`. Výpočet metody je ilustrován následujícím pseudokódem.

Metoda 1 ResourceApi getResourceStatistics(ResourceApi resource)

```
1:  total ← execApi(newCommand("cdrlog.query"))
2:  firstIndex ← total.get("firstIndex")
3:  numEvents ← total.get("numEvents")
4:  lastId ← resource.getLastIdentifier()
5:  if lastId ≠ null then
6:      firstIndex ← lastId
7:      numEvents ← firstIndex + numEvents − lastId
8:  while numEvents > 0 do
9:      command ← new Command("cdrlog.enumerate")
10:     command.setParameter("index", firstIndex)
11:     events ← execApiEnumerate(command, "events")
12:     for each event in events do
13:         numEvents ← numEvents − 1
14:         if event.get("type").equals("participantLeft") then
15:             room ← findOrCreateRoom(event, resource.getRoom())
16:             interval ← createInterval(event)
17:             roomInterval ← new RoomInterval()
18:             roomInterval.setInterval(interval)
19:             newSession ← new RoomSession()
20:             newSession.setInterval(interval)
21:             newSession.addInterval(roomInterval)
22:             attendee ← new RoomAttendee()
23:             attendee.setName(event.get("details").get("h323Alias"))
24:             newSession.getAttendees().add(attendee)
25:             newSession ← createSession(room, newSession, interval)
26:             room.getRoomSessionList().add(newSession)
27:     firstIndex ← events.get(events.size() − 1).get("index") + 1
28: resource.setLastIdentifier(firstIndex)
29: resource.setUpdatedAt(Date.now())
30: resource.setTechnology(Technology.H323)
31: return resource
```

Statistiky jsou stahovány uvedeným způsobem, jelikož vzhledem k velkému množství uložených záznamů je efektivnější stažení jen nových dat oproti opakovanému získávání celého objemu statistik ze zdroje H.323 MCU.

Další implementovaná metoda v třídě `CiscoMCUConnector` se jmenuje `RoomSession createSession(RoomApi room, RoomSession newSession, Interval interval)`. Metoda z objektu `newSession` vytvoří sezení, které vznikne spojením dalších případných překrývajících se sezení v místnosti. Běh uvedené metody začíná `for` cyklem, který iteruje přes jednotlivá sezení obsažená v atributu sessions v objektu `room`. Na začátku `for` cyklu se proměnná `added` vždy nastaví na hodnotu `false`. Tato proměnná indikuje, jestli se v sezeních místnosti `room` již našlo sezení, jehož časový interval má neprázdný průnik s časovým intervalem sezení v objektu `newSession`. V dalším kroku je do proměnné `roomSession` uloženo aktuálně zpracovávané sezení ze seznamu zpracovaných sezení v objektu `room`. Potom algoritmus porovnává vztah existujícího intervalu sezení `session` a právě přidávaného intervalu sezení `newSession`. Mezi novým intervalem sezení `newSession` a stávajícím intervalem sezení `session` mohou nastat následující vztahy:

- interval sezení `newSession` je obsažen v existujícím sezení `session`,
- interval nového sezení `newSession` obsahuje existující sezení `session`,
- existující sezení obsahuje jen začátek nového sezení,
- existující sezení obsahuje jen konec nového sezení.

Pokud je některá z uvedených podmínek splněna, upraví se interval nového sezení tak, aby zahrnoval interval sezení v proměnné `session`. Dále se nastaví indikátor `added` na hodnotu `true` a ostatní podmínky se již nevyhodnocují. Jestliže je proměnná `added` nastavena na hodnotu `true`, na konci iterace cyklu se spustí posloupnost následujících příkazů. Odstraní se ze seznamu sezení objektu `room` právě zkoumané sezení a `newSession` získá všechna sezení a účastníky sezení, kterými disponovalo sezení `session`. Na konci metoda v návratové hodnotě vrátí proměnnou `newSession` reprezentující sezení. Postup algoritmu je znázorněn následujícím pseudokódem.

Metoda 2 RoomSession createSession(RoomApi room, RoomSession session)

```
1:  for iterator ← room.getSessions().iterator(); it.hasNext(); do
2:      added ← false
3:      session ← iterator.next()
4:      sessionInterval ← session.getInterval()
5:      interval ← newSession.getInterval()
6:      if sessionInterval.contains(interval) then
7:          newSession.setInterval(sessionInterval)
8:          added ← true
9:      if !added && interval.contains(sessionInterval)  then
10:         added ← true
11:     if !added && sessionInterval.contains(interval.getStart()) then
12:         i ← newInterval(sessionInterval.getStart(), interval.getEnd())
13:         newSession.setInterval(i)
14:         added ← true
15:     if !added && sessionInterval.contains(interval.getEnd()) then
16:         i ← newInterval(interval.getStart(), sessionInterval.getEnd())
17:         newSession.setInterval(i)
18:         added ← true
19:     if added then
20:         iterator.remove()
21:         newSession.getInterals().add(session.getIntervals())
22:         newSession.getAttendees().add(session.getAttendees())
23: return newSession
```

Další implementovanou metodou v rámci práce je `RoomApi findOrCreateRoom(Map<String, Object> event, List<RoomApi> rooms)`. Tato metoda má za úkol najít existující místnost v seznamu `rooms` nebo ji vytvořit. Metoda dostane na vstupu proměnnou `event` představující mapu, která obsahuje data o události a seznam místností `rooms`. Na začátku metoda projde seznam `rooms`. Jestliže se při průchodu objeví místnost se stejným unikátním identifikátorem, jako je obsažen v klíči mapy `event`, je vrácena aktuálně procházená místnost. Pokud při průchodu není nalezena žádná místnost, která by odpovídala identifikátoru mapy, je vytvořena místnost nová. Nově

vytvořená místnost je nakonec vrácena v návratové hodnotě a běh metody končí.

Poslední implementovanou metodou v třídě `CiscoMCUConnector` je `Interval createInterval(Map<String, Object> event)`. Tato pomocná metoda zpracuje objekt `event` představující událost a vrátí interval události. Algoritmus využije toho, že událost typu odpojení účastníka vždy obsahuje informace o aktuálním času a době, kterou účastník strávil ve spojení. Díky tomu jednoduše spočítá začátek intervalu spojení odečtením stráveného času od časové značky odpojení účastníka. Nakonec metoda v návratové hodnotě vrátí vypočtený interval.

4.1.3 Třída `AbstractProcessor`

Třída `AbstractProcessor` se nachází v modulu `shongo-statistics`, který je vytvořen pro systém zpracovávající statistiky. Zmíněná třída prostřednictvím implementace rozhraní `ProcessorService` slouží k uložení získaných homogenních statistik z konektorů `AdobeConnectConnector` (kapitola 4.1.1) a `CiscoMCUConnector` (kapitola 4.1.2). Jelikož je třída navržena jako abstraktní, nemůže být vytvářena její instance. Z toho plyne, že pokud chceme využívat její metody, musí být rozšířena jinou třídou. `AbstractProcessor` obsahuje následující atributy:

- `server` identifikuje webovou adresu zdroje,
- `port` specifikuje konkrétní port webové adresy zdroje,
- `userName` představuje přihlašovací jméno uživatele ke zdroji,
- `password` obsahuje heslo uživatele,
- `database` reprezentuje název databáze, do které jsou statistiky uloženy,
- `technology` určuje konkrétní typ zdroje,
- `connector` je instance konektoru, z kterého jsou získány statistiky,

- `entityManager` obsahuje správce entit, prostřednictvím kterého jsou ukládány entity do databáze,
- `resourceApi` představuje objekt zdroje, se kterým pracuje aplikace,
- `resource` reprezentuje zdroj jako entitu, která je uložena do databáze pomocí objektu `entityManager`,
- `attendees` je seznam entit představující uživatele, kteří budou uloženi do databáze.

Kromě metod sloužících k získání a nastavení zmíněných atributů jsem ve třídě `AbstractProcessor` implementoval níže popsané metody.

Metoda, pomocí které je vytvořeno připojení ke konektorům, se nazývá `public void connect()`. Dále metoda vytvoří manažer entit, jejž uloží do objektu `entityManager`.

Další implementovaná metoda ve třídě `AbstractProcessor` se nazývá `void createResource()`. Zmíněná metoda uloží do objektu `resourceApi` a `resource` informace o zdroji z databáze. Pokud zdroj s technologií `technology` v databázi není přítomen, pak je vytvořena nová reprezentace zdroje, která je uložena do objektů `resourceApi` a `resource`.

Třída obsahuje metody `persistRooms()`, `persistSessions()`, `persistIntervals()`, `persistAttendees()`, které postupně procházejí data uložená v objektu `resourceApi` a ukládají je do databáze. Další pomocná metoda je `getAttendees(roomAttendees, session)`, jež slouží k prohledání množiny účastníků `attendees`, zda již neobsahují účastníky z množiny `roomAttendees`.

Dále třída `AbstractProcessor` obsahuje veřejnou metodu `abstract void persistData()`, která musí být přetížena třídou rozšiřující `AbstractProcessor`.

4.1.4 Třída `AdobeConnectProcessor`

V této sekci popisuji implementaci třídy `AdobeConnectProcessor`, která rozšiřuje třídu `AbstractProcessor` (kapitola 4.1.3). Třída

`AdobeConnectProcessor` má za úkol získat a uložit data zprostředkovaná pomocí konektoru `AdobeConnectConnector` (popsán v kapitole 4.1.1). Zmíněná třída `AdobeConnectProcessor` obsahuje následující metody.

Nejdůležitější metodou je `public void persistData()`, která se stará o spouštění dalších metod vykonávajících dílčí úkoly. Na začátku metoda zahájí databázovou transakci. Tato metoda v aktuální transakci dále vymaže všechny místnosti využitím pomocné metody `private void deleteRooms()`. Dalším krokem je spuštění metody `getResourcesStatistics(getResourceApi())` z třídy `AdobeConnectConnector`, která vloží všechny statistiky ze zdroje Adobe Connect do objektu `resourceApi`. Následuje uložení získaných dat pomocí metod `persistRooms()` a `persistAttendees()`. Na konci metody `persistData()` je aktuální transakce dokončena.

Zmíněná metoda `private void deleteRooms()` slouží k vymazání všech místností, dílčích sezení a intervalů z databáze patřící zdroji Adobe Connect. Dále metoda vyprázdní místnosti v atributech `resourceApi`, `resource` a účastníky v `attendees` třídy `AbstractProcessor`.

4.1.5 Třída `CiscoMCUProcessor`

Třída `CiscoMCUProcessor` implementovaná v rámci zpracování statistik rozšiřuje abstraktní třídu `AbstractProcessor` (kapitola 4.1.3) a slouží k uložení dat získaných pomocí konektoru `CiscoMCUConnector` (popsán v kapitole 4.1.2). V této části uvádím popis, jakým způsobem jsou implementovány metody třídy `CiscoMCUProcessor`.

Nejdůležitější metoda, kterou jsem implementoval ve třídě `CiscoMCUProcessor`, se nazývá `void persistData()`. Zmíněná metoda postupně spouští ostatní metody vytvořené ve třídě `CiscoMCUProcessor` tak, aby došlo k uložení statistik. Na začátku je spuštěna pomocná metoda `flushData()`, která vyprázdní seznam místností z atributu `resourceApi` a množinu účastníků `attendees`. Poté je započata databázová transakce příkazem `getTransaction().begin()`. Dalším krokem je získání dat z konektoru `CiscoMCUConnector` pomocí metody `getResourceStatistics(ResourceApi)`, která vloží statistiky

do objektu `resourceApi`. Následuje uložení získaných statistik v objektu `resourceApi` do persistentního stavu pomocí metod `persistRoom()` a `persistAttendees()`. Na konci metody je započatá transakce ukončena příkazem `getTransaction()->commit()` a změny jsou promítnuty do databáze.

Další implementovaná metoda `protected void persistRooms()` přetěžuje stejnojmennou metodu z třídy `AbstractProcessor` (kapitola 4.1.3). Zmíněná metoda slouží k uložení místností do databáze. Jediným rozdílem mezi metodou z třídy `AbstractProcessor` a přetíženou je, že v přetížené metodě je kontrolována existence právě ukládané místnosti. Jestliže místnost existuje, jsou k ní přiřazena nová sezení a dílčí intervaly. Pokud místnost neexistuje, je vytvořena nová. Dále již metoda `persistRoom()` k uložení dílčích sezení místnosti využívá metodu `persistSessions()`, která je obsažena ve třídě `AbstractProcessor`.

Poslední implementovaná metoda v třídě `CiscoMCUProcessor` se nazývá `persistAttendees()`. Metoda přetěžuje stejnojmennou metodu a slouží k uložení získaných účastníků v atributu `attendees` třídy `AbstractProcessor`. Jedinou odlišností od původní metody je fakt, že nová metoda ukládá účastníka do databáze, jen pokud neexistuje.

4.2 Popis implementace webového rozhraní

V této kapitole uvádím popis implementace webového rozhraní, které slouží k prezentaci uložených statistik.

4.2.1 Struktura adresového prostoru

Struktura adresového prostoru směrovače, jehož interakce s ostatními komponentami aplikačního rámce Laravel je ilustrována v obrázku 3.1, je rozdělena na dvě části. První část tvoří adresový prostor vyhrazený prezentační vrstvě aplikace, jehož adresy jsou definovány v souboru `/routes/web.php`. Tato část obsahuje následující adresy:

- `/` (kořenová adresa) – zobrazí domovskou stránku aplikace,
- `/resources/{id}` – zobrazí přehled využití zdroje, kde `{id}` je primární klíč požadovaného zdroje,

- `/rooms/{id}` – vyobrazí přehled využití virtuální místnosti, kde `{id}` je primární klíč požadované místnosti,
- `/rooms/{id}/{from}/{to}` – zobrazí stránku poskytující přehled využití místnosti s primárním klíčem `{id}`, kde hlavní graf místnosti vyobrazuje detail místnosti se sezeními, která se odehrála v intervalu od časové značky `{from}` do `{to}`.

Druhou část adresového prostoru tvoří adresy aplikačního programového rozhraní, které slouží k definování adres používaných technologií Ajax. Po přístupu na vybranou adresu aplikačního programového rozhraní je vrácena odpověď v JSON formátu. Aby bylo možno takto definované adresy snadno odlišit od adres sloužících k přístupu k prezentační vrstvě, obsahují adresy aplikačního programového rozhraní prefix `api`. Adresní prostor aplikačního programového rozhraní je definován v souboru `/routes/api.php` a obsahuje následující adresy:

- `/api/session_detail/{session_id}` – vrátí podrobná data o detailu sezení s primárním klíčem `{session_id}`,
- `/api/resource_interval/{resource_id}/{from}/{to}` – vrátí seznam místností zdroje s primárním klíčem `{resource_id}`, ve kterých se v intervalu definovaném časovými značkami `{from}` a `{to}` odehrávalo alespoň jedno sezení.

Pokud je navštívena adresa, která není uvedena ve zmíněném seznamu, je automaticky zobrazena uživatelsky přívětivá stránka s chybovým kódem `404`, která značí, že uvedená adresa není nalezena.

4.2.2 Řadiče

Řadiče jsou speciální třídy sloužící ke zpracování požadavků uživatele. Jak je znázorněno pomocí obrázku 3.1, řadiče jsou odpovědné za komunikaci s databází a vytváření pohledů, do kterých připravují data. Ve webové aplikaci jsem implementoval tři řadiče, které jsou umístěny ve složce `/app/Http/Controllers`.

První řadič se nazývá `ResourceController.php` a jeho úkolem je příprava dat pro pohledy, které se zobrazí uživateli při navštívení adres `/` a `/resources/{id}`.

Při navštívení kořenové adresy / se v řadiči spustí metoda `index()`, která uživateli zobrazí statickou titulní stránku aplikace `/resources/views/main/index.blade.php`.

Pokud uživatel navštíví adresu `/resources/{id}`, kde `{id}` je primární klíč požadovaného zdroje, předá se řízení běhu aplikace metodě s názvem `show($id)`, kde `$id` je hodnota primárního klíče zdroje shodná s `{id}`. Tato metoda slouží k zobrazení přehledu využití zdroje. Na začátku metody `show($id)` je do proměnné `$resource` vložen zdroj z databáze s primárním klíčem `$id`. Poté jsou do pohledu připraveny proměnné, které v nich bude umožněno používat. Do proměnné `$resource` jsou vloženy informace o zdroji (název, datum posledního zpracování dat). Dále jsou do proměnné `$chart_data` pomocí metody `getChartData()` z modelu `Resource` vloženy statistické údaje o využití jednotlivých místností, které jsou následně použity v grafu a tabulce. Na konci metody je zobrazena šablona `/resources/views/resource/index.blade.php`, která využívá připravené proměnné.

Další implementovaný řadič se nazývá `RoomController.php`. Řadič má za úkol přípravu dat pro šablony, které se mají zobrazit, pokud uživatel navštíví adresu `/rooms/{id}` nebo `/rooms/{id}/{from}/{to}`.

`RoomController.php` obsahuje jedinou metodu `show($id, $from = null, $to = null)`, které se předá řízení, pokud uživatel navštíví jednu ze dvou zmíněných adres. Zmíněná metoda slouží k zobrazení detailního přehledu místnosti. Na začátku získá metoda všechna sezení a uživatele místnosti s primárním klíčem `$id`. Poté jsou pomocí metody `getSessionsData()` do proměnné `$sessions_data` vložena data s informacemi o sezení pro diagram v šabloně přehledu místnosti. Jestliže byly v argumentech webové adresy předány hodnoty představující interval od `from` do `to`, je spočítán interval zobrazení detailu v diagramu. V předposledním kroku jsou v proměnné `$room` zpřístupněny šabloně souhrnné informace o místnosti. Na konci metody je spuštěn příkaz, který zobrazí šablonu `/resources/views/resource/room.blade.php` s uvedenými daty.

Poslední implementovaný řadič se nazývá `ApiController.php`. Tento řadič se od ostatních liší tím, že neslouží k zobrazení šablon, ale zpřístupňuje data při asynchronní komunikaci. Zdrojový kód řadiče obsahuje metody `getSessionDetail($session_id)` a `getResourceInterval($resource_id, $from, $to)`.

První metoda `getSessionDetail($session_id)` se spustí, pokud je zahájen GET požadavek na adresu `/session_detail/{id}`, kde `{id}` je primární klíč sezení. Funkcí této metody je získání detailních dat o sezení. Zmíněná detailní data obsahují chronologicky seřazené časy s připojením nebo odpojením účastníků a s aktuálním počtem účastníků přítomných v sezení.

Druhá implementovaná metoda v řadiči `ApiController.php` je pojmenována `getResourceInterval($resource_id, $from, $to)`. Uvedená metoda se spustí, jestliže uživatel navštívil adresu `/resource_interval/{resource_id}/{from}/{to}`. Ve zmíněné adrese položka `{resource_id}` představuje primární klíč požadovaného zdroje, `{from}` a `{to}` jsou časové značky intervalu. Hlavní funkcí metody je zpřístupnění dat, určených pro šablonu přehledu zdroje. Díky použitým parametrům `{from}` a `{to}` je umožněno specifikovat časový interval, za který jsou požadovaná data vrácena.

4.2.3 Modely

Použitý aplikační rámec Laravel (popsaný v kapitole 3.2.2) umožňuje mapovat entity z databáze na třídy a zjednodušuje tak práci s daty. Entity lze mapovat vytvořením tříd, které se nazývají Modely a rozšiřují třídu `Model`. Modely umožňují dotazovat se na data v tabulkách, stejně tak jako vkládat do nich nové záznamy.

V aplikaci jsou modely uloženy ve složce `/app/Models` a ke každé tabulce z databáze je vytvořen odpovídající model. Mezi nejdůležitější modely patří `Resource.php` a `MeetingRoom.php`.

Model `Resource.php` reprezentuje zdroj a disponuje následujícími metodami. První implementovanou metodou je `getChartData()`. Tato metoda projde všechny místnosti zdroje a připraví jejich data k použití v diagramu a tabulce v šabloně přehledu zdroje. Další implementovanou metodou je `getChartDataByInterval($from, $to)`. Uvedená metoda funguje totožně jako metoda `getChartData()`, jen s tím rozdílem, že zahrnuje místnosti, jen pokud obsahují alespoň jedno sezení v intervalu tvořícím časové značky `$from` a `$to`.

Další model `MeetingRoom.php` obsahuje důležitou metodu `getSessionsData()`. Uvedená metoda slouží k přípravě dat pro diagram sezení v detailu místnosti. Metoda projde všechna sezení míst-

nosti a vhodně zformátuje jejich data, tak aby byla připravena pro použití v diagramu.

4.2.4 Šablony

Šablony jsou základním kamenem prezentační vrstvy webové aplikace používající aplikační rámec Laravel. Jak je ilustrováno v obrázku 3.1, hlavní funkcí zmíněných šablon je zobrazení požadovaného výsledku uživateli v prohlížeči. Laravel disponuje šablonovacím systémem se speciální syntaxí. Díky tomu se lze vyhnout mnohdy nepřehlednému kombinování PHP kódu a HTML značek. Šablony jsou uloženy ve složce `/resources/views` a mají příponu `.blade.php`. Ve webové aplikaci jsou implementovány níže popsané šablony.

Základní šablona `./layouts/main.blade.php` slouží jako výchozí rámec pro ostatní šablony. Dokument ve své hlavičce obsahuje vložené základní CSS styly a javascriptové soubory. Dále je v těle šablony definován navigační panel, který slouží uživateli jako navigátor ve webové aplikaci. Potom následuje místo pro samotný obsah stránky, do kterého je vložena jiná šablona. Na konci dokumentu je vložena patička stránky.

Domovská stránka webové aplikace je tvořena šablonou `./main/index.blade.php` a obsahuje odkazy na jednotlivé zdroje.

Další šablona `./resource/index.blade.php` tvoří stránku s přehledem zdroje, zahrnující diagram a tabulku všech místností vybraného zdroje.

Šablona `./resource/room.blade.php` představuje stránku, která tvoří detail místnosti. Šablona obsahuje chronologicky uspořádaný diagram se všemi sezeními místnosti a diagram detailu vybraného sezení.

Poslední šablona `./errors/404.blade.php` slouží jako stránka, která se zobrazí, pokud požadovaná adresa nebyla v seznamu definovaných adres nalezena.

4.2.5 Komponenty

Ve webové aplikaci se nacházejí různé interaktivní prvky, které jsou implementovány v Javascriptu. Kód zmíněných interaktivních prvků je přehledný, ale časem by mohlo dojít k rozšíření jeho funkciona-

lity. Z tohoto důvodu je použit javascriptový aplikační rámec Vue.js (popsaný v kapitole 3.2.5), pomocí kterého lze složitou funkcionalitu kódu udržovat v přehledném a snadno čitelném stavu. Vue.js vytváří interaktivní prvky aplikace pomocí komponent. Kód komponenty je rozdělen na HTML šablonu se speciální syntaxí a javascriptovou část. V šabloně jsou definovány prvky HTML kódu s proměnnými, které jsou provázány se stejnojmennými proměnnými v javascriptové části šablony. Komponentu lze potom vložit do běžného HTML kódu pomocí nově definované značky, nesoucí název komponenty.

Jedinou implementovanou komponentou je `RoomsTable.vue`, nacházející se ve složce `/resources/assets/js/components/`. Tato komponenta tvoří obsah stránky s přehledem zdroje, kde umožňuje vyhledávat místnosti v diagramu a tabulce, podle jejich jména a intervalu jejich sezení. Komponenta implementuje filtrování podle jména místnosti pomocí metody `filterByName()`. Filtrace pomocí časového intervalu je implementována metodou `filterByDate()`. Při filtrování pomocí intervalu získá metoda z kalendářů časový interval zadaný uživatelem. Tento interval se potom použije při asynchronním požadavku na adresu `/api/resource_interval/{resource_id}/{from}/{to}`. Data, která vrátí zmíněné asynchronní volání, aktualizují diagram a tabulku s místnostmi. Dále komponenta umožňuje řazení místností podle názvu, celkové délky spojení, počtu spojení nebo sezení. Řazení je implementováno pomocí metody `sortBy(column)`, kde `column` je objekt reprezentující sloupec tabulky.

4.2.6 Diagramy

Ve webové aplikaci jsou vytvořeny diagramy poskytující přehled o zdroji, místnosti a sezení. Zmíněné diagramy jsou implementovány pomocí javascriptové knihovny Highcharts, jež je popsána v kapitole 3.2.4. Výhodou Highcharts je, že nabízí velice jednoduché rozhraní pro práci s diagramy.

První implementovaný diagram poskytuje přehled o využití zdroje a nachází se v souboru `/resources/assets/js/resource_graph.js`. Diagram je vložen do komponenty `RoomsTable.vue`, která tvoří stránku přehledu zdroje a obsahuje seznam všech místností s počty spojení a délky jejich sezení (obrázek 5.2). Tento diagram využívá data z řadiče `ResourceController.php` uložené do proměnné

`$chart_data`. Jelikož zdroj obsahuje velké množství místností, je diagram z důvodu přehlednosti navržen jako pruhový (bar chart).

Další dva implementované diagramy ve webové aplikaci jsou sloupcového typu (column chart) a nacházejí se v souboru `/resources/assets/js/room.js`. Uvedené diagramy jsou vloženy do šablony `/resources/views/resource/room.blade.php`, která je zobrazena při navštívení detailu místnosti 5.3.

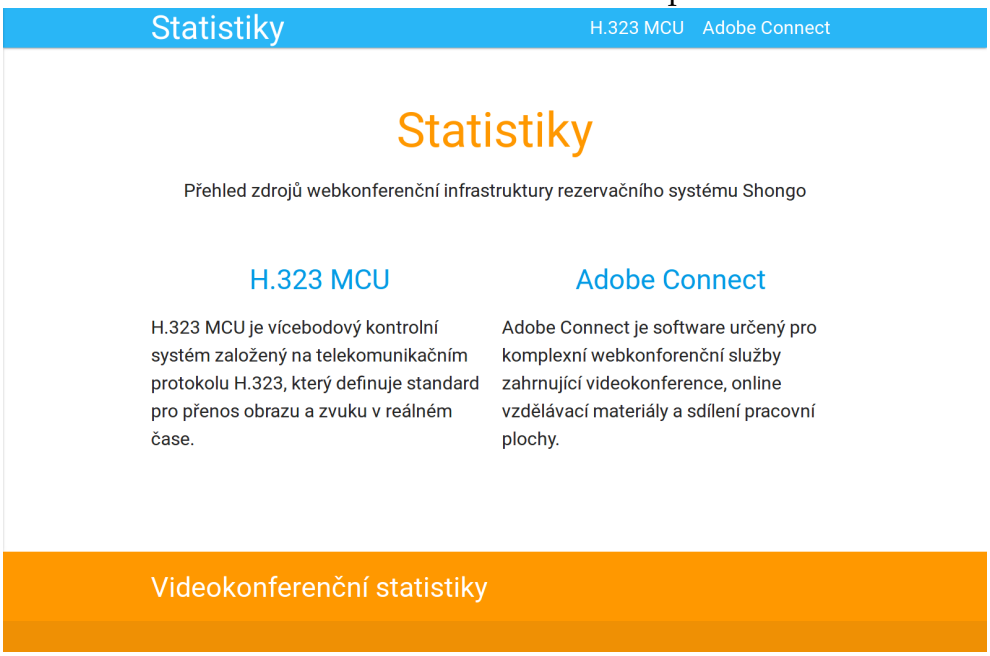
První z dvojice diagramů slouží k zobrazení chronologicky seřazených sezení místnosti s měřítkem určujícím počet jejich spojení. Diagram tedy obsahuje sezení aktuálně zobrazené místnosti s časovou osou, kde výška sloupce představuje počet spojení v sezení. Z důvodu potencionálního velkého množství sezení, jež může virtuální místnost obsahovat, umožňuje diagram zobrazit vybraný časový interval detailněji.

Druhý diagram v souboru `room.js` představuje detail sezení s časy připojení a odpojení účastníků a aktuálním počtem účastníků. Tento diagram je provázán s předchozím diagramem sezení, a to tak, že po kliknutí na sloupec představující sezení z prvního diagramu je zobrazen detail vybraného sezení v aktuálním diagramu. Po kliknutí je spuštěn asynchronní požadavek na adresu `/api/session_detail/{id}`, kde `{id}` je primární klíč požadovaného sezení. Odpověď na tento požadavek je vytvořena pomocí metody `getSessionDetail($session_id)` v řadiči `ApiController.php`, která vrátí upravený seznam spojení vybraného sezení, představující provoz v místnosti.

5 Uživatelské prostředí aplikace

V této kapitole uvádím popis uživatelského rozhraní webové aplikace, které bylo vytvořeno na základě návrhu uživatelského rozhraní (kapitola 2.4). Všechny webové stránky aplikace jsou vloženy do výchozí šablony, která obsahuje navigační panel a patičku stránky. Tímto způsobem získají všechny stránky aplikace jednotný design. První webová stránka tvoří domovskou stránku aplikace, kde jsou uživateli nabídnuty odkazy na přehled zdrojů Adobe Connect a H.323 MCU. Podobu domovské stránky zachycuje obrázek 5.1.

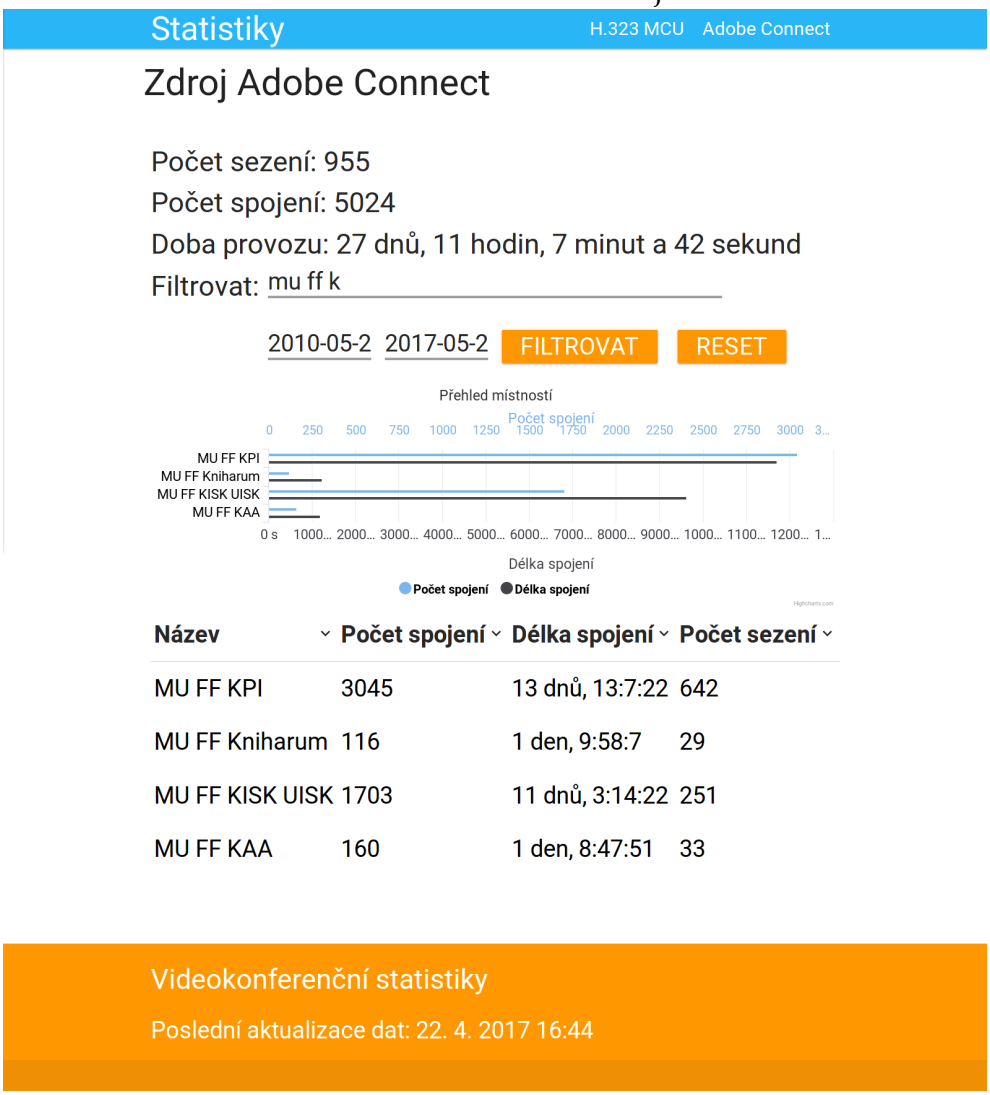
Obrázek 5.1: Domovská stránka aplikace



Další webová stránka aplikace obsahuje přehled využití zdroje. Tato stránka disponuje údaji o celkovém počtu sezení, spojení a celkové době spojení zdroje. Stránka dále obsahuje diagram a tabulku se seznamem místností zdroje. Jednotlivé pruhy diagramu a řádky tabulky představují místnosti a disponují informacemi o počtu sezení, spojení a době využití místnosti. Aplikace umožňuje uživateli pomocí

vstupního pole vyhledávat ve zmíněném seznamu místností podle jejich jména. Dále dovoluje omezit výběr seznamu místností pomocí vstupních polí s kalendářem jen na určitý časový interval, kdy se sezení v místnosti odehrávala. Další vlastností stránky je, že při vyhledávání jsou celkové hodnoty o využití místnosti automaticky aktualizovány. V neposlední řadě lze při kliknutí na vybraný sloupec tabulky seřadit vzestupně či sestupně položky v tabulce a diagramu. Pokud uživatel klikne na řádek tabulky představující místnost, je přesměrován na přehled zvolené místnosti. Zmíněný popis přehledu zdroje ilustruje obrázek 5.2.

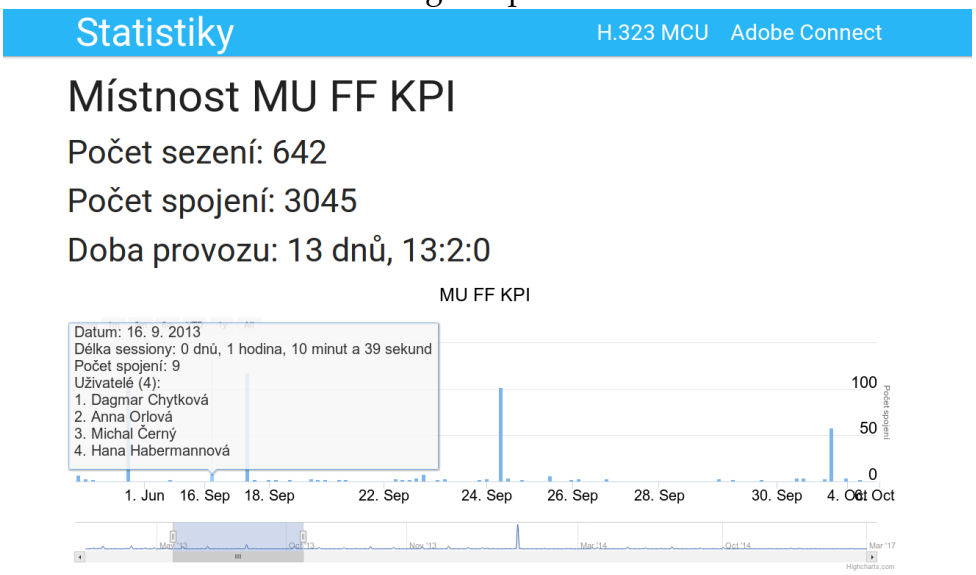
Obrázek 5.2: Přehled zdroje



Poslední webová stránka představuje přehled zvolené místnosti. Obsahuje informace o počtu sezení, počtu spojení místnosti a celkové době sezení v místnosti. Hlavním prvkem této stránky je sloupcový diagram sezení, kde výška sloupce představuje počet spojení v sezení. Zmíněné sloupce jsou chronologicky seřazeny, podle toho, kdy se sezení odehrálo. Při najetí myší na sloupec se uživateli zobrazí

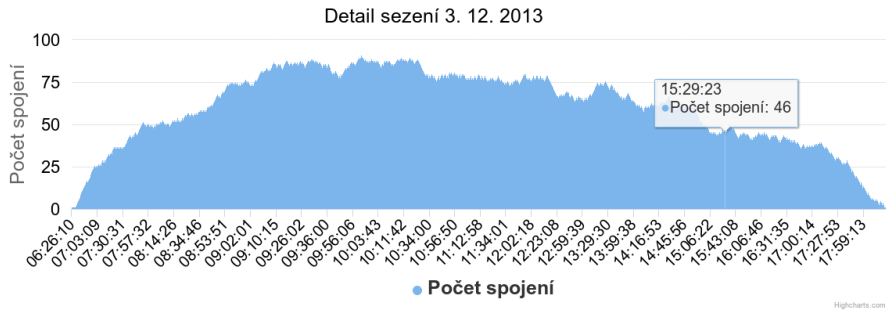
datum sezení, doba trvání sezení, počet spojení v sezení a seznam uživatelů, kteří se účastnili sezení. Tento diagram disponuje lištou, která umožňuje zobrazit detailněji určitý časový interval. Diagram přehledu místnosti popisuje obrázek 5.3.

Obrázek 5.3: Diagram přehledu místnosti



Další funkcí diagramu přehledu místnosti (obrázek 5.3) je, že pokud uživatel klikne na sloupec představující sezení, zobrazí se mu detail zvoleného sezení v druhém diagramu. Diagram detailu sezení obsahuje časové značky určující čas připojení nebo odpojení účastníka a výška sloupce značí počet aktuálně připojených účastníků v sezení. Popsaný diagram detailu sezení objasní obrázek 5.4.

Obrázek 5.4: Diagram detailu sezení



Videokonferenční statistiky

6 Závěr

Cílem bakalářské práce bylo navrhnout a implementovat systém pro sběr, zpracování a prezentaci provozních statistik z videokonferenční a webkonferenční infrastruktury, kterou provozuje sdružení CESNET. Důležitým požadavkem bylo, aby nový systém byl provázán s rezervačním systémem Shongo, který slouží ke správě zmíněné infrastruktury. Všechny uvedené cíle byly splněny a jsou dokumentovány v předešlých kapitolách.

V rámci této bakalářské práce jsem vytvořil systém určený pro sběr a prezentaci provozních statistik. Tento systém uživateli poskytuje komplexní a zároveň aktuální přehled o využití zdrojů rezervačního systému Shongo. Dále je součástí výsledné práce věcná dokumentace kódu, která poslouží při případném rozšiřování systému.

V přiloženém kódu práce je prostor pro jeho další rozšíření. Možným pokračováním této práce by mohlo být vylepšení funkcionality webové aplikace o export dat do souborů různých formátů. Dalším rozšířením by mohlo být přidání přehledu o využití kapacit z pohledu jednotlivých uživatelů systému Shongo.

Souhrnně hodnotím přínos této práce pozitivně, jelikož velkou měrou prohloubila mé znalosti v oblasti tvorby webových aplikací.

Literatura

[1] Martin Šrom, Petr Holub, Jan Růžička, Miloš Liška, Ondřej Pavelka a Ivan Novakov. *Shongo API*. 2013. URL: https://github.com/shongo/shongo/blob/master/shongo-doc/shongo/api/api.pdf (cit. 02. 05. 2017).

[2] Ray Horak. *Telecommunications and Data Communications Handbook*. 2. vyd. John Wiley & Sons, 2007. 856 s. ISBN: 0-470-12722-8. URL: https://books.google.cz/books?id=dO2wCCB7w9sC&pg=1 (cit. 13. 03. 2017).

[3] Cisco. *Cisco TelePresence MCU API guide 2.9*. 2012. URL: http://www.cisco.com/c/dam/en/us/td/docs/telepresence/infrastructure/mcu/api_guide/MCU_API_2-9.pdf (cit. 02. 05. 2017).

[4] Adobe Systems Incorporated. *Using ADOBE ® CONNECT TM 8 Web Services*. 2010. URL: http://help.adobe.com/en_US/connect/8.0/webservices/connect_8_webservices.pdf (cit. 02. 05. 2017).

[5] Andrey Volkov Salahaldin Juba Achim Vannahme. *Learning PostgreSQL*. 1. vyd. Packt Publishing Ltd, 2015. 464 s. ISBN: 9781783989195. URL: https://books.google.cz/books?id=jfKoCwAAQBAJ&printsec=frontcover (cit. 13. 03. 2017).

[6] The PostgreSQL Global Development Group. *PostgreSQL 9.6.3 Documentation*. 2016. URL: https://www.postgresql.org/files/documentation/pdf/9.6/postgresql-9.6-A4.pdf (cit. 02. 05. 2017).

[7] George Schlossnagle. *Pokročilé programování v PHP 5*. 1. vyd. Zoner Press, 2004. 640 s. ISBN: 80-8681-514-5.

[8] Martin Bean. *Laravel 5 Essentials*. 1. vyd. Packt Publishing Ltd, 2015. 144 s. ISBN: 9781785283291.

[9] ECMA International. *Ecmascript Language Specification (Open Documents Standards Library)*. 1. vyd. Iuniverse Inc, 2000. 196 s. ISBN: 978-1583482674.

[10] Highsoft AS. *Highcharts API Reference*. 2016. URL: http://api.highcharts.com/highcharts (cit. 02. 05. 2017).

[11] Evan You. *Vue.js*. 2017. URL: https://vuejs.org/v2/api/ (cit. 02. 05. 2017).

A Příloha

Součástí práce je archiv obsahující zdrojové kódy systému pro sběr, zpracování a prezentaci videokonferenčních statistik z rezervačního systému Shongo. Příloha je dostupná v elektronickém archivu práce. Popis instalace systému je obsažen v souboru `readme.md`. Část systému starající se o sběr a zpracování provozních statistik se nachází v adresáři `shongo/shongo-statistics/src/`. Prezentace získaných statistik je implementována v adresáři `shongo/shongo-statistics/statistics-web/`.