Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Informační systém pro sběr dat z výukových aplikací

Dominik Foral

Vedoucí práce: Ing. Jiří Hunka

7. května 2015

Poděkování

V první řadě bych chtěl poděkovat zadavatelům Petrovi Zavadilovi a Tomáši Sýkorovi za příjemnou spolupráci, v rámci které vzniklo zadání této práce. Dále bych chtěl poděkovat vedoucímu práce Ing. Jiřímu Hunkovi za jeho čas a cenné rady při našich konzultacích a také mé rodině a kamarádům, kteří mě při psaní této práce podporovali.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na základě níž se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ust. § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 7. května 2015 .....

Odkaz na tuto práci

Foral, Dominik. *Informační systém pro sběr dat z výukových aplikací.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Obsahem této bakalářské práce je analýza požadavků, návrh a implementace informačního systému pro sběr dat z výukových aplikací. V rešeršní části práce jsou popsány podobné systémy Google Analytics a Flurry Analytics. Výstupem praktické části této práce je funkční prototyp webové aplikace, která umožňuje data z aplikací sbírat a zobrazovat.

Klíčová slova informační systém, sběr dat, Google Analytics, PHP, Nette

Abstract

This bachelor thesis contains requirements analysis, design and implementation of information system for collecting data from education applications. In research part of the thesis are described similar systems such as Google Analytics and Flurry Analytics. Practical result of this thesis is working prototype of web application, which collects and displays data from applications.

Keywords information system, data collecting, Google Analytics, PHP, Nette

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Informační technologie se během posledních desítek let rozšířily téměř do všech oborů lidské činnosti. Navíc během posledních pár let proběhl obrovský rozmach mobilních zařízení s dotykovým ovládáním, které vnáší do práce s počítači zcela nový rozměr. Mobilní zařízení se rychle rozšiřují také do škol, kde na nich žáci pracují s interaktivními vzdělávacími aplikacemi. Tyto aplikace nejen že seznamují děti s informačními technologiemi, ale také pro ně činí výuku zábavnější.

Dále poskytují cenný zdroj informací, který lze využít na různé výzkumy populace. Například se jedná o schopnost dětí řešit různé úkoly nebo o jejich postupné zlepšování v čase. Tyto výsledky je možné poté prezentovat například rodičům, kteří tak získají přehled o schopnostech svého dítěte. Tato data slouží také vývojářům, kteří je využijí k vylepšování a ladění svých aplikací.

Tato bakalářská práce postupně popisuje jednotlivé fáze vývoje funkčního prototypu systému, jenž umožní shromažďovat a prezentovat data nejen z výukových aplikací.

Kapitola 1

Analýza požadavků

V této kapitole je v úvodu uvedeno, jako probíhal sběr požadavků se zadavateli. Dále jsou specifikovány funkční i nefunkční požadavky na systém. Tyto požadavky jsou dospecifikovány a shrnuty ve FURPS analýze. Na konci kapitoly jsou rozepsány jednotlivé případy užití.

1.1 Seznámení se základní problematikou

Zcela na začátku mé práce proběhly skypové diskuze se zadavateli (Tomáš Sýkora, Petr Zavadil), ve kterých jsme probírali zejména jejich výukovou aplikaci pro děti s názvem *Jdu do školy*, která zkouší schopnost dětí v předškolním věku řešit jednoduché úkoly. Z těchto diskuzí vzešla potřeba vytvořit informační systém, který by byl schopný přijímat, zpracovávat a zobrazovat data z této aplikace.

Takto získaná data budou sloužit k ladění aplikace, prezentaci výsledků rodičům, porovnávání v rámci populace a k dalším statistickým účelům.

Již v této fázi projektu mi došlo, že k problému sběru dat budu muset přistoupit co nejobecněji, aby byl systém připraven na případné změny ze strany aplikace. A když už budu problém řešit obecně, nabízí se možnost, aby aplikace byla schopná sbírat data nejen z jedné konkrétní aplikace, nýbrž z téměř jakékoli. Na této možnosti jsme se se zadavateli shodli a dohodli jsme se, že tato služba by mohla být v budoucnu nabízena veřejně (na internetu), tzn. kdokoliv ji bude moci použít.

Se zadavateli jsme pojmenovali aplikaci pracovním názvem Šmak, takže dále v této práci budu tento název užívat.

1.2 Funkční požadavky

1.2.1 Sběr a skladování dat

Systém Šmak bude přijímat přes dané rozhraní data ze zařízení a ukládat je do databáze. Data je nutné ukládat tak, aby je bylo možné později přiřadit k uživateli od kterého pochází a také aby byly známé parametry zařízení, ze kterého byla přijata.

Data (datové struktury) jenž bude možné sbírat, budou definovány v rámci systému.

1.2.2 Uživatelské rozhraní a role

Šmak má dále umožňovat přístup ke sbíraným datům. Se zadavateli jsme se shodli, že nejvhodnějším prostředkem bude webové rozhraní, ke kterému bude možné přistupovat hned z několika uživatelských rolí.

První rolí je *uživatel aplikace (supervisor)*, jenž bude mít možnost registrace a přihlášení. Po registraci v systému, bude možné spojit jeho účet s konkrétní instancí aplikace, kterou má (nebo měl) uživatel na svém zařízení. Po spárování bude poté uživatel schopný prohlížet svá data, která byla z jeho instance posbírána.

Další rolí je *vývojář (developer)*, který bude mít stejné možnosti jako supervisor. Dále mu aplikace umožní založit profil své aplikace (ze které budou sbírána data). V rámci takto vytvořené aplikace si bude moci vybrat struktury dat, které bude chtít sbírat. Dále mu systém vygeneruje autentizační informace, které vývojář použije při komunikaci jeho aplikace se Šmakem. K tomu ještě bude možné nastavit, která data (datové struktury) se zobrazí supervisorům. Například jim vývojář takto zakáže zobrazování výjimek a chyb z aplikace.

Třetí uživatelskou rolí bude *Administrátor*, který bude mít pravomoce dělat vše, co předchozí role (až na registraci v systému). Dále se bude starat o datové struktury v systému Šmak, tzn. jejich instalaci do systému.

1.3 Nefunkční požadavky

Mezi hlavními nefunkčními požadavky od zadavatelů byla modulárnost, která se týká hlavně jednotlivých datových struktur, které bude Šmak umožnovat sbírat. Tyto struktury by měly být řešeny jako pluginy, které bude možné do systému jednoduše instalovat.

Dále je požadováno, aby aplikace byla schopná přijímat data z více zařízení zároveň a zvládala tato data efektivně ukládat a pracovat s nimi po dobu několika let.

Tabulka 1.1: Fukční požadavky

Název	Popis
Uživatelské role	Přístup k systému z uživatelských rolí uživatel aplikace (supervisor), vývojář, administrátor
Přihlášení a registrace	Systém umožňuje přihlášení a uživatelským rolím supervisor a vývojář také registraci.
Vytvoření profilu vlastní aplikace	Vývojář může v systému vytvořit profil své aplikace
Výběr datových struktur	Vývojář zvolí z dostupných datových struktur ty, které bude chtít ze své aplikace sbírat
Nastavení zobrazovaných datových struktur uživatelům	Vývojář je schopný nastavit, které datové struktury se zobrazují uživatelům.
Poskytnutí autentizačních informací	Systém po přijetí požadavku o zaregistrovaní nové instance aplikace poskytne autentizační informace nutné k další komunikaci.
Spárování instance aplikace s jejím uživatelem	Systém umožňuje přijmout požadavek o spárování instance aplikace a jejího uživatele (supervisora)
Přijem dat	Systém umožňuje přijmout a zpracovat požadavek o vložení dat ze zaregistrované aplikace
Zobrazení dat supervisorovi	Supervisorovi je umožněno zobrazit data z jeho sprárovaných instancí aplikací.
Zobrazení dat z aplikace	Vývojář může zobrazit data posbíraná ze své aplikace
Instalace datových struktur	Administrátor má možnost instalovat nové pluginy s datovými strukturami

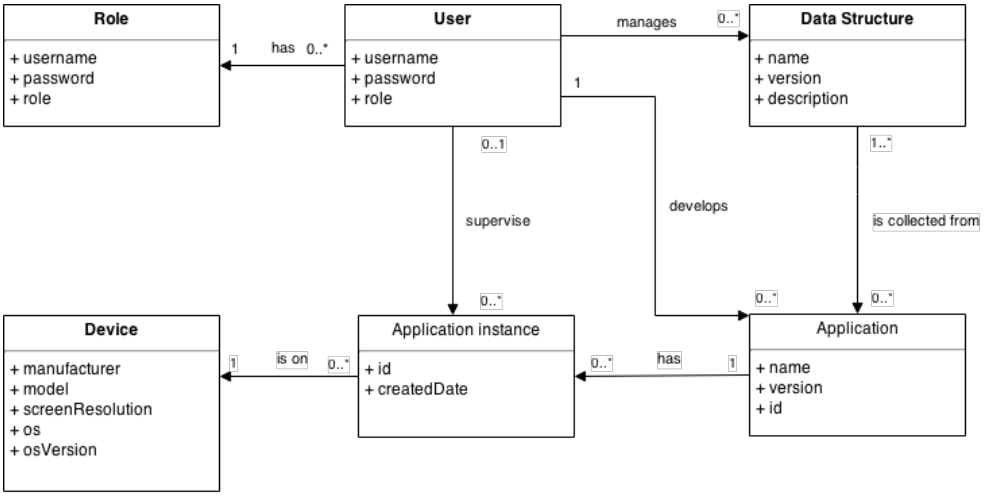
1.4 FURPS analýza

1.4.1 Funkčnost

Tabulka 1.1 přehledně zobrazuje funkční požadavky kladené na systém. V rámci systémových testů budou později ověřovány právě tyto požadavky.

1.4.2 Použitelnost

- Přístup k uživatelskému rozhraní z jakékoli platformy, která podoporuje web
- Přijem dat z jakékoli platformy, která podporuje přenosový protokol
- Nasaditelnost na běžné hostingy



Obrázek 1.1: Doménový model

1.4.3 Spolehlivost

- Dostupnost 99 % času

1.4.4 Výkon

- maximálně 100 aplikací na vývojářský účet
- řádově 100 000 uživatelů/aplikace
- řádově 1000 různých statistik/uživatele

1.4.5 Podporovatelnost

- Modularita systému - podporující budoucí vývoj, či výměnu modulů

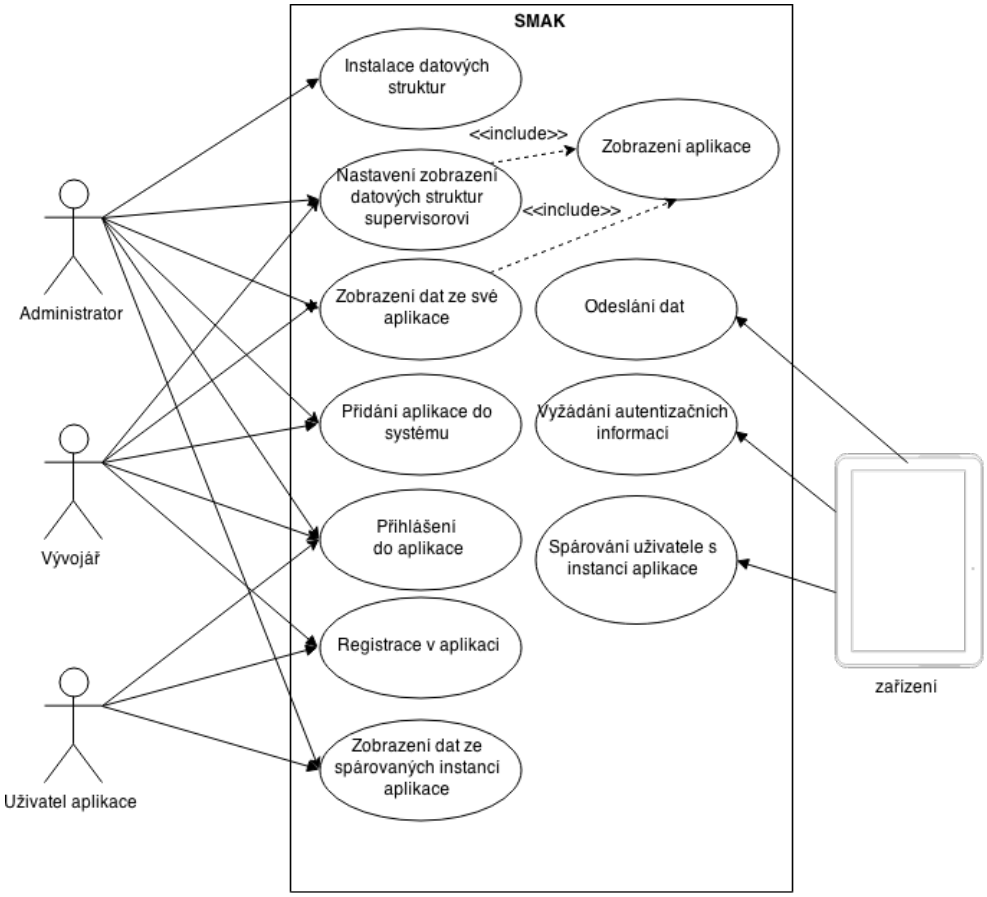
1.5 Analytický doménový model

Na obrázku 1.1 je znázorněn model problémové domény, který zachycuje vztahy mezi jednotlivými entitami.

1.6 Případy užití

1.6.1 Diagram

Na obrázku 1.2 je znázorněn diagram případů užití, který přehledně zachycuje pravomoci jednotlivých aktérů.



Obrázek 1.2: Diagram případu užití

1.6.2 PU1 Registrace v aplikaci

Umožňuje uživateli se zaregistrovat jako vývojář nebo jako uživatel

1.6.2.1 Aktéři

- Uživatel/vývojář
- Systém

1.6.2.2 Hlavní scénář

1. Uživatel zvolí možnost *Registrovat*
2. Systém zobrazí registrační formulář, který obsahuje uživatelské jméno, heslo, heslo znovu a výběr role (uživatel, vývojař)
3. Uživatel vyplní údaje a potvrdí formulář

4. Systém zvaliduje data
5. Systém zaregistruje uživatele

1.6.2.3 Alternativní scénář

Začíná v kroku 4, pokud uživatel zadá nevalidní údaje do formuláře.

1. Systém upozorní uživatele o nevalidních údajích a vyzve ho k jejich úpravě (krok 2)

1.6.3 PU2 Přihlášení do aplikace

Umožňuje se jakémukoli registrovanému uživateli se přihlásit

1.6.3.1 Aktéři

- Uživatel/Vývojář/Administrátor
- Systém

1.6.3.2 Hlavní scénář

1. Uživatel zvolí možnost *Přihlásit se*
2. Systém zobrazí přihlašovací formulář s uživatelským jménem a heslem
3. Uživatel vyplní údaje a odešle formulář
4. Systém přihlásí uživatele a zobrazí mu příslušnou domovskou obrazovku

1.6.3.3 Alternativní scénář

Začíná v kroku 3 hlavního scénáře, když uživatel vyplní neplatné přihlašovací údaje.

1. Systém upozorní uživatele na nesprávné údaje a vyzve ho k opravě (krok 2).

1.6.4 PU3 Přidání aplikace do systému

1.6.4.1 Aktéři

- Vývojář/administrátor
- Systém

1.6.4.2 Podmínky pro spuštění

Uživatel musí být přihlášen do systému.

1.6.4.3 Hlavní scénář

1. Uživatel zvolí možnost *Přidat novou aplikaci*
2. Systém zobrazí formulář, který obsahuje název aplikace, aktuální verze a seznam datových struktur
3. Uživatel vyplní údaje, vybere požadované datové struktury a odešle formulář
4. Systém zvaliduje údaje
5. Systém informuje uživatele o úspěšném přidání aplikace.

1.6.4.4 Alternativní scénář

Začíná v kroku 4, když uživatel vyplní neplatné údaje.

1. Systém upozorní uživatele a vyzve ho k opravě (krok 2).

1.6.5 PU4 Spárování uživatele s instancí aplikace

Umožňuje spárovat registrovaného uživatele s instancí aplikace registrované v systému

1.6.5.1 Aktéři

- Systém
- Zařízení
- Uživatel

1.6.5.2 Hlavní scénář

1. Aplikace v zařízení umožní uživateli zadat svoje uživatelské jméno do zařízení.
2. Aplikace pošle ze zařízení požadavek do systému na spárování své instance a uživatelského jména
3. Systém spáruje uživatele s instancí aplikace.

1.6.6 PU5 Vyžádání autentizačních informací

Umožňuje aplikaci v zařízení získat autentizační informace pro komunikaci se Šmakem

1.6.6.1 Aktéři

- Zařízení
- Systém

1.6.6.2 Hlavní scénář

1. Aplikace pošle požadavek do systému
2. Systém vygeneruje autentizační informace a zašle ho aplikaci
3. Systém zaregistruje do systému novou instanci dané aplikace

1.6.7 PU6 Instalace datových struktur

Umožní administrátorovi nainstalovat datové struktury do systému

1.6.7.1 Aktéři

- Systém
- Administrátor

1.6.7.2 Hlavní scénář

1. Administrátor zvolí v aplikaci *Správa datových struktur*
2. Systém zobrazí seznam datových struktur v aplikaci
3. Administrator zvolí *Přidat novou strukturu*
4. Systém zobrazí formulář na vložení souboru s modulem struktury
5. Administrator vybere soubor s modulem a nahraje ho
6. Systém zkontroluje validitu modulu a vytvoří novou strukturu v aplikaci

1.6.8 PU7 Zobrazení dat ze spárované instance aplikace

Umožní zobrazit uživateli data z jeho spárované instance aplikace

1.6.8.1 Podmínky pro spuštění

Uživatel musí být přihlášen v aplikaci a musí mít nějaké spárované instance aplikace.

1.6.8.2 Aktéři

- Uživatel
- Systém

1.6.8.3 Hlavní scénář

1. Uživatel zvolí možnost *Spárované aplikace*
2. Systém zobrazí uživateli seznam aplikací, u každé podseznam spárovaných instancí a u každé instance zobrazí přiřazené datové struktury
3. Uživatel zvolí datovou strukturu u požadované instance, z které chce zobrazit data
4. Systém zobrazí data

1.6.9 PU8 Odesílání dat ze zařízení

Umožňuje aplikaci v zařízení odesílat data do systému

1.6.9.1 Podmínky pro spuštění

Zařízení má získané autentizační informace

1.6.9.2 Aktéři

- Systém
- Zařízení

1.6.9.3 Hlavní scénář

1. Zařízení odešle požadavek do systému
2. Systém přijme požadavek a potvrdí zařízení jeho přijetí
3. Systém zkontroluje validitu požadavku a uloží data do databáze

1.6.9.4 Alternativní scénář

Začíná v kroku 3 hlavního scénáře, pokud je přijat neplatný požadavek.

1. Systém neuloží data do databáze.

1.6.10 PU9 Zobrazení aplikace

Umožňuje zobrazit vývojáři profil své aplikace.

1.6.10.1 Podmínky pro spuštění

Vývojář má vytvořenou alespoň jednu aplikaci

1.6.10.2 Aktéři

- Vývojář
- Systém

1.6.10.3 Hlavní scénář

1. Uživatel zvolí nabídku *Moje aplikace*
2. Systém zobrazí uživateli jeho aplikace
3. Uživatel zvolí požadovanou aplikaci

1.6.11 PU10 Nastavení zobrazení datových struktur supervisorovi

Umožňuje vývojáři nastavit u datových struktur sbíraných jeho aplikací, zdali se mají zobrazovat také supervisorovi u jeho spárovaných instancí dané aplikace.

1.6.11.1 Podmínky pro spuštění

Vývojář má vytvořenou alespoň jednu aplikaci

1.6.11.2 Aktéři

- Vývojář
- Systém

1.6.11.3 Hlavní scénář

1. Include (PU9 Zobrazení aplikace)
2. Systém zobrazí seznam datových struktur, které jsou sbírány z této aplikace a u každé struktury možnost změnit nastavení zobrazení na Ano nebo Ne.
3. Uživatel změní nastavení

1.6.12 PU11 Zobrazení dat z aplikace

Umožňuje vývojáři zobrazit data ze své aplikace

1.6.12.1 Podmínky pro spuštění

Vývojář má vytvořenou alespoň jednu aplikaci

1.6.12.2 Aktéři

- Vývojář
- Aplikace

1.6.12.3 Hlavní scénář

1. Include (PU9 Zobrazení aplikace)
2. Systém zobrazí seznam datových struktur, které jsou sbírány z této aplikace a u každé struktury možnost *Zobrazit data*
3. Uživatel zvolí možnost *Zobrazit data*
4. Systém zobrazí data z dané datové struktury a aplikace

Kapitola 2

Analýza technologií a podobných systémů

Tato kapitola analyzuje podobné existující aplikace Google Analytics a Flurry Analytics. Dále stručně porovnává technologie vhodné pro implementaci prototypu a popisuje a zdůvodňuje zvolené řešení.

2.1 Podobné existující aplikace

2.1.1 Google Analytics

Jedná se o rozsáhlou aplikaci od společnosti Google, která je schopná sbírat a vizualizovat data, jenž pochází zejména z webů a mobilních aplikací. Nad těmito daty je postavena rozsáhlá analytická část aplikace, která nabízí širokou škálu možností jak s daty pracovat.

2.1.1.1 Měření webu

Analytics dokáže sledovat uživatelské interakce s webovými stránkami a aplikacemi a to díky vložení jednoduchého javascriptového (trackovacího) kódu do stránky. Takto získaná data si poté může webový vývojář prohlédnout v přehledných tabulkách a grafech v administaci GA. Mezi shromažďovaná data patří například návštěvnost, demografické údaje návštěvníka (odkud ke stránce přistupoval), technologie návštěvníka (internetový prohlížeč, operační systém) a chování (doba strávená na stránce). GA dále umožňuje obrovskou škálu nastavení, z nichž bych rád zmínil možnost nastavit zabezpečený přenos dat, zakázání sledování některých částí webu a možnost vytvořit si vlastní metriky.

2.1.1.2 Měření mobilních aplikací

Z pohledu mé práce je zajímavější možnost získávat data z mobilních aplikací, jenž také Google Analytics umožňuje. Jedná se o software jehož klientskou část (SDK), je nutné nainstalovat do aplikace na přenosném zařízení s operačním systémem Android nebo iOS [1]. Poté je možné odesílat data na server. Podporovaná data jsou rozdělena do několika oblastí:

Event (událost) s parametry category, action, label, value reprezentuje jakoukoliv událost v aplikaci. Parametr category slouží k třídění událostí do kategorií, jenž je možné poté prohlížet zvlášť. Action určuje konkrétní akci, která může být například stisknutí tlačítka stop nebo play. Zbývajícími parametry label a value je možné doplnit událost o další data [2].

Ecommerce (elektronický obchod) umožňuje sledovat nejrůznější události a chování zákazníka v elektronickém obchodě. Konkrétně se jedná o iterakce uživatelů s produkty - jejich zobrazení, vložení do košíku, zaujetí produktem či sledování transakcí [3].

Goal (cíl) slouží ke sledování dosažení určitých cílů v rámci aplikace. Nastavení cílů je možné provést v uživatelském rozhraní GA. Jako příklad uvedu koupi produktu, či dosažení určité úrovně ve hře. Každý cíl se definuje pomocí čtyř parametrů: Goal Type (typ cíle), Category, Action, a boolean hodnoty rozhodující o tom, zdali se použije parametr value události jako hodnota cíle [4].

Custom dimension (vlastní dimenze) v GA umožňují přidat k odesílaným datům dodatečné informace (metadata). Podle těchto údajů je možné v přehledech jednotlivé záznamy třídit a seskupovat. Každá dimenze je definovaná pomocí čtyř parametrů: Index, Name, Active a Scope a nastavuje se v uživatelském rozhraní. Význam prvních 3 parametrů je zřejmý z názvu, proto bych se chtěl rozepsat o parametru Scope, jehož nastavení dělá výrazné rozdíly při zpracování těchto dimenzí. Scope (rozsah) může nabývat 4 hodnot: hit, session, user, product. Pokud bude scope nastaven na hodnotu hit, odešle se hodnota dimenze s každým odeslaným hitem [1]. Pokud je scope nastaven na session, stačí danou dimenzi nastavit pouze jednou a bude přiřazena i k ostatním hitům v daném session. Podobně je na tom user scope, avšak hodnota dimenze je přiřazena k jednomu uživateli napříč všemi session. Poslední scope product se moc neliší od hit scope avšak dimenze se nepřiřazuje k události či obrazovce, ale k produktu [5].

Custom metrics (vlastní metriky) jsou v podstatě počítadla nějakého hitu či produktu. Může to být například počet zobrazení stránky, obrazovky.

[1] Hit je právě jeden odeslaný požadavek na server

Stejně jako dimenze, je možné vlastní metriky nastavit v uživatelském rozhraní a je možné jim nastavit různý parametr scope. V tomto případě pouze hodnoty hit a product, význam těchto dvou hodnot je obdobný jako u dimenzí. V aplikaci se poté volá kód, který danou metriku zvýší o libovolnou hodnotu (obvykle o 1) [5].

Obrazovky (Screens) umožňují zjistit počty zobrazení určitých obrazovek a způsob, jak mezi nimi uživatel přechází. Obrazovka obsahuje pouze jeden parametr a to její název (Screen Name). GA dále nabízí automatické sledování obrazovek, které je možné používat, pokud každé obrazovce přiřadíte její jméno v XML konfiguračním souboru [6].

Výjimky a pády GA dokáže zpracovávat data o typu a počtu nastalých výjimek a to jak zachycených, tak i nezachycených. Každá výjimka je definovaná dvěma parametry: Description, který popisuje danou výjimku a isFatal, což je boolean hodnota popisující, zdali byla výjimka fatální (tzn. že došlo k pádu aplikace) [7].

User Timing (časové měření) je určeno parametry Category, Value, Variable, Label. Z určitého pohledu je časové měření podobné událostím, avšak nás nezajímá, zdali nějaká akce proběhla, ale jak dlouho trvala. Význam jednotlivých parametrů je stejný jako u událostí, až na value, který obsahuje časový údaj (v milisekundách) [8].

Komunikace mezi zařízením a GA je zprostředkována protokolem HTTP(S) a to pomocí jednoduchých POST nebo GET požadavků. Požadavky lze zasílat na dvě URL adresy, záleží zdali chceme využít zabezpečeného přenosu dat (HTTPS) nebo nikoliv. Každý odeslaný požadavek musí obsahovat 4 parametry - verzi, komunikační klíč, ID klienta a typ odesílaných dat (událost, časové měření, atd.). Seznam dalších parametrů lze najít v dokumentaci GA [9].

Poslední věcí o které bych se rád zmínil je způsob jak GA řeší problém zahlcení. Tato záležitost je řešena na obou stranách - na serveru i na klientu. Na serveru jsou nastaveny limity na počet požadavků celkově za měsíc a pak také na uživatele a session. Na klientské straně, pokud používáme některý z trackerů (objekt zprostředkující komunikaci s GA), je omezen počet hitů pomocí počítadla, které se v pravidelných intervalech zvyšuje o 1 a při odeslání hitu snižuje o 1 [10].

2.1.2 Flurry Analytics

Flurry Analytics (FA) je konkurenční nástroj pro sběr a analýzu dat od Yahoo, který nabízí podobné služby jako GA. Zaměřím se hlavně na rozdíly

Event (událost) je narozdíl od GA dvouúrovňová. První úroveň tvoří název události (konkrétní akce), v druhé úrovni je možné specifikovat až 10 libovolných parametrů ve formátu klíč - hodnota. Flurry dále umožňuje měření délky události, o které se, narozdíl od GA, stará samotné Flurry SDK. [11]

Tabulka 2.1: Java EE pro a proti

Výhody	Nevýhody
• rychlost • robustní technologie • enterprise technologie jsou nativní	• horší nabídka hostingů • dražší hostingy • horší podpora

Další rozdíl je v tom, jak Flurry a GA přistupují k velkým objemům dat. GA začně po 500 000 záznamech data vzorkovat zatímco Flurry používá pro reporting vždy všechna přijatá data. Vzorkování znamená, že se z přijatých dat vybere podmnožina, která reprezentuje daná data, ale jelikož se jedná o menší množinu, jsou výpočty a zobrazování mnohem rychlejší [12].

2.2 Technologie

2.2.1 Platforma

Možností v čem naprogramovat serverovou aplikaci s webovým rozhraním je několik. Mezi dva nepoužívanější jazyky patří PHP a Java [13]. V následujích sekcích rozeberu jejich pro a proti .

2.2.1.1 Java Enterprise Edition

Java Enterprise Edition nabízí již v základu širokou škálu technologií, jenž umožňují vytvořit rozsáhlou modulární aplikaci. Jedná se zejména o skvělý GUI framework Java Server Faces, který skvěle odděluje aplikační logiku od uživatelského rozhraní, čímž umožňuje jeho pohodlný a přehledný vývoj.

Dále se stojí za zmínku Java Persistent API, které zajišťuje mapování objektů na databázi (ORM) pomocí entitních tříd. Pomocí této technologie je možné se z velké části odpoutat od zvolené databáze a vůbec jejího relačního schématu a věnovat se pouze práci s objekty.

Problém s Javou je ten, že je těžší najít vyhovující webhosting. Hostingy na Javu jsou obecně dražší a méně rozšířené než pro PHP. A jelikož je nutné, aby byla aplikace snadno nasaditelná na veřejný server, stojí tato skutečnost proti použití Javy.

Tabulka 2.2: PHP pro a proti

Výhody	Nevýhody
• možnost nasazení téměř na jakémkoliv hostingu • nejrozšířenější technologie - velká podpora • mnoho frameworků	• dynamické typování - horší čitelnost kódu • slabší výkon

2.2.1.2 PHP

PHP je ve světě internetu bezkonkurečně nejpoužívanější technologií . Existuje mnoho frameworků, které umožňují používat v PHP podobné technologie jako v Javě EE (ORM, šablonovací systémy), což umožňuje rychlejší vývoj aplikací. Tyto frameworky mívají širokou základnu uživatelů, což usnadňuje jejich používání a řešení případných potíží.

PHP je však od počátku vyčítán jeho slabší výkon. Je to daň za to, že je jazyk dynamicky typovaný, což je na jednu stranu pohodlné, avšak na druhou stranu to přináší problémy s optimalizací výkonu. Dále za tuto skutečnost může fakt, že je PHP interpretovaný jazyk [14].

Problém s výkonem je samozřejmě možné omezit psaním efektivního kódu, které celou aplikaci výrazně zrychlí. K tomu se však ještě nabízí použití HipHop Virtual Machine (dále HHVM) od společnosti Facebook, která tento software vyvíjí za účelem zvýšení výkonu PHP kódu. HHVM převádí PHP kód do bytekódu, který je dynamicky převáděn do strojového kódu pomocí just in time kompilátoru. Už v posledním čtvrtletí roku 2012 se vývojářům podařilo HHVM optimalizovat tak, že je výkon srovnatelný s kódy, které byly kompilovány z PHP do C++ [15].

2.3 Zvolené technologické řešení

2.3.1 Platforma

Po zvážení všech pro a proti jsem se nakonec rozhodl využít PHP. Vedlo mě k tomu více důvodů. Jedná se zejména o možnost jednoduchého nasazení aplikace na téměř jakýkoliv server, ale také to, že jsem již v PHP v minulosti vytvořil několik aplikací, takže mám s touto platformou vcelku bohaté zkušenosti.

Dále se mi také zdálo zajímavé vyzkoušet výše zmiňovanou virtuální mašinu (HHVM) k dosažení lepších výkonů aplikace. Přestože budu systém vyví-

jet a testovat na HVVM, není problém s nasazením na běžný server, na kterém běží klasické PHP. V testovací části této práce poté porovnám výkon aplikace na běžném serveru a na serveru s HHVM.

2.3.2 Framework

Na tvorbu aplikace jsem se rozhodl využít úspěšný český framework Nette (verze 2.2.1). Nabízí se i další PHP frameworky jako je například *Symphony* či *Zend Framework*, ale k volbě právě Nette mě vedla zejména osobní zkušenost. Z té vím, že mi tento framework poskytne vše, co při vývoji takové aplikace potřebuji.

Jedná se zejména o architekturu MVP, kterou Nette využívá a která je pro takovéto systémy velmi vhodná, protože od sebe oddělí jednotlivé vrstvy aplikace a umožní jejich případnou jednoduchou úpravu nebo výměnu. Dále se tím také celá aplikace z pohledu vývojáře velmi zpřehlední.

Za zmínku také stojí podpora dependency injection a debugovacího nástroje Laděnka. Nette také dále řeší zabezpečení proti útokům nejrůznějších typů (od Cross-site scripting až po Session hijacking).

K tomu má Nette také velmi širokou komunitu (z velké části českou), na jejíž internetovém fóru lze často najít řešení k problémům, které mohou během vývoje s tímto frameworkem nastat.

Vývoj uživatelského rozhraní umožní šablonovací systém Latte, jenž se součástí Nette. Tento systém je navržen s ohledem na maximální srozumitelnost a jednoduchost, pričemž neztrácí obrovskou škálu funkcí (tzv. maker).

Spolu s Latte využiji dále i HTML, CSS a JS knihovnu Bootstrap, která poskytuje velkou množinu předpřipravených prvků uživatelského rozhraní. Na samotné zobrazení (tabulkových) dat využiji javascriptovou knihovnu W2UI.

2.3.3 Databáze

Databázových řešení se nabízí opravu mnoho a není vůbec jednoduché vybrat to, které se na daný projekt hodí nejvíce. V mém případě jsem se rozhodl pro známou a ověřenou databázi MySQL. K mému rozhodnutí mě vedl fakt, že jsem nenašel důvod proč toto řešení nepoužít. Přeci jen MySQL používají (alespoň z části) největší servery na světě jako je Facebook, Google či Youtube, což vypovídá o kvalitě tohoto řešení [16].

K tomu jsem se z literatury dočetl o mnohých způsobech optimalizace MySQL - od schématu až po nastavení hardwaru na kterém databáze běží. Takto získané znalosti mě přesvědčily o tom, že MySQL nebude určitě špatná volba.

Kapitola 3

Návrh

V této kapitole je postupně popisován návrh systému. Nejdříve popisuje komunikaci zařízení se systémem, z kterého plyne návrh aplikačního rozhraní. Dále rozebírá architekturu MVP a poté konkrétní návrhové třídy. Spolu s tím je zmíněna bezpečnost komunikace a popsány funkce klientské části této aplikace, jejíž implementace je v budoucnu plánovaná.

3.1 API a komunikace se zařízením

3.1.1 Komunikace Šmaku se zařízením

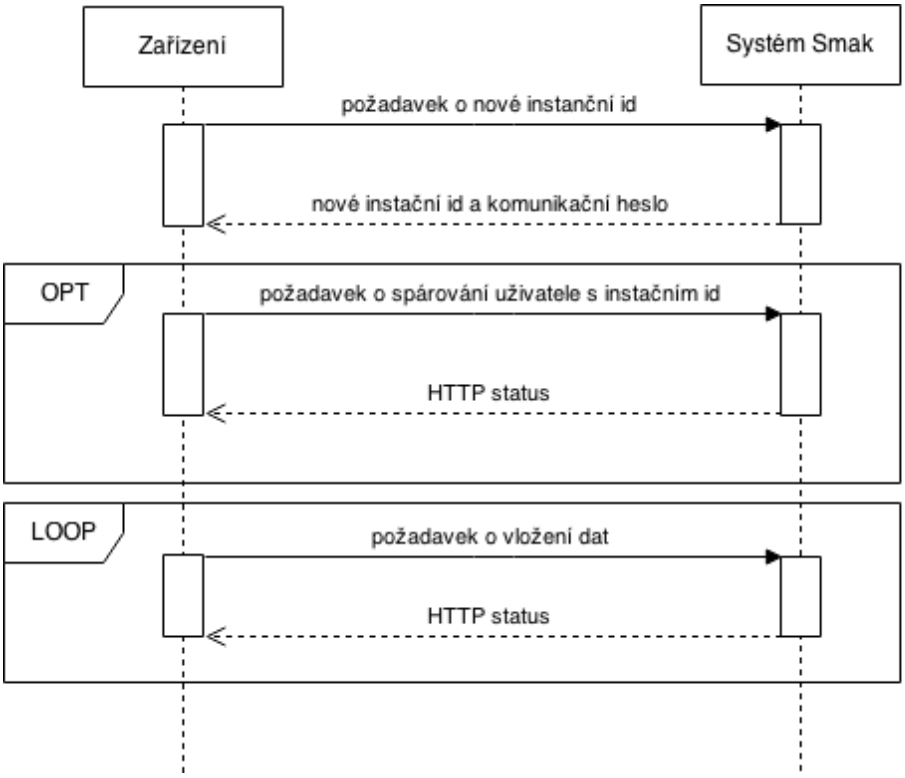
Veřejné API bude umožňovat komunikaci zařízení s touto aplikací a je velice důležité při jeho návrhu myslet na jednoduchost a srozumitelnost. Na obrázku 3.1 je zobrazen sekvenční diagram komunikace zařízení s touto aplikací. Zařízení si nejdříve vyžádá jedinečné instanční id, které je poté použito jako identifikátor při komunikaci. Každá nainstalovaná instance aplikace bude tak mít svoje instanční id, které bude nutné mít uloženo někde v paměti (např. v konfiguračním souboru). [2]

Aplikace o toto id může přijít v důsledku odinstalování ze zařízení či vymazání dat aplikace. Při opětovném nainstalování aplikace či po vymazání dat je nutné aby si aplikace požádala o nové instanční id, což dává smysl, protože se v podstatě jedná o jinou instanci.

Jakmile má instance aplikace svoje id, může ihned odesílat data do Šmaku, což znázorňuje cyklus (LOOP) na obrázku 3.1. Šmak po přijetí požadavku zpracuje (zvaliduje) data a vrátí zařízení HTTP status 200 (OK), jako informaci že data přijal (nikoliv zdali byla data validní).

Poslední a nepovinnou částí komunikace je požadavek na spárování instančního id s konkrétním uživatelem zaregistrovaným ve Šmaku. Zařízení

[2] Pokud by nás zajímala data souhrně ze všech instancí, stačilo by jako identifikátor id aplikace, avšak aplikace musí umožňovat přístup k datům z konkrétní uživatelovy instance a tak je instanční id nezbytné.



Obrázek 3.1: Sekvenční diagram komunikace zařízení se systémem Šmak

odešle svoje instanční id spolu s uživatelským jménem (pod kterým je uživatel zaregistrovaný ve Šmaku) a Šmak přiřadí danému uživateli konkrétní instanci. Tento uživatel je poté schopen zobrazit data z dané instance. Spárování může proběhnout nezávisle na vkládání dat do systému, ale musí nastat, až když má instance získané své identifikační číslo.

K zabezpečení odesílaných dat a párování instancí bude sloužit komunikační heslo, které bude aplikaci předáno spolu s novým instančním id. Tato tématika je více rozebrána v sekci 3.6.1.

3.1.2 API

Z návrhu komunikace výše jasně vyplývá, že API Šmaku potřebuje mít 3 metody:

- **getNewInstId(appKey, deviceInfo)** - poskytuje zařízením nové instanční id. Parametr *appKey* slouží k identifikaci samotné aplikace (aplikační klíč) a druhý parametr *deviceInfo* obsahuje informace o zařízení, jako je výrobce, model, rozlišení obrazovky, operační systém, verze operačního systému. Tato data budou předána jako serializované asociované pole ve formátu JSON. Klíče tohoto asociovaného pole jsou:

  - manufacturer
  - model
  - resolution
  - os
  - os_version

  Po přijetí validního požadavku vrátí Šmak asociované pole ve formátu JSON se dvěma prvky. Prvním z nich má klíč *instId* a obsahuje nové instanční id. Druhý z nich dostupný pod klíčem *psswd* obsahuje párovací heslo. Více o párovacím heslu lze nalézt v sekci 3.6.1.

- **insert(structName, data, psswd)** - přijímá data ze zařízení. Parametr *structName* určuje datovou strukturu a parametr *data* poté obsahuje samotná data. Parametr *psswd* je komunikační heslo, získané spolu se svým instančním id. Data budou předána jako serializované pole polí ve formátu JSON. Více o tomto parametru a optimalizaci lze nalézt v sekci 4.2

- **pairUserWithInstId(instId, psswd, username)** - umožnuje spárovat danou instanci s uživatelem. První parametr *instId* je instanční id aplikace z níž požadavek pochází. Druhým parametrem *psswd* je heslo, jenž instance získala při získání svého instančního id. Posledním parametr *username* určuje se kterým uživatelským účtem, který je zaregistrovaný ve Šmaku, se má daná instance spárovat.

Konkrétní návod i s příklady, jak používat metody aplikačního rozhraní lze nalézt v kapitole 5.

3.2 Architektura

Jak jsem již zmínil dříve, na tvorbu výsledné aplikace jsem vybral framework Nette. Ten je postaven na známé třívrstvé softwarové architektuře MVC (model, view, controller), respektive na její odvozenině MVP (model, view, presenter) [17].

Tato architektura rozděluje aplikaci na 3 vrstvy:

- Model - obsahuje data (umožňuje k přístup nim) a business logiku
- View - stará se o vstup a výstup aplikace
- Presenter - obsahuje aplikační logiku a ovlivňuje model a view.

Aplikaci dále rozdělím do modulů, abych zpřehlednil nejen adresářovou strukturu, ale i seskupil třídy podle jejich zameření. Rozhodl jsem se, že bude

vhodné rozdělit aplikaci na dva základní moduly - Core (Jádro) a Structures (Struktury).

Modul Core se bude starat o základní běh aplikace, tzn. přihlašování, registrace uživatelů a autorizace uživatelů, vytváření nových aplikací. Dále bude poskytovat veřejné API a vytvářet základní uživatelské rozhraní aplikace.

Druhým modulem je Structures, jenž se bude spravovat datové struktury. Jedná se zejména o jejich instalaci do aplikace a o vkládání dat. Dále bude tento modul obsahovat abstraktní třídy a rozhraní, jenž budou třídy z pluginů datových struktur rozšiřovat a implementovat. Třídy jednotlivých pluginů budou též obsaženy v tomto modulu.

3.2.1 Model obecně

Třídy v této vrstvě mají v názvu postfix *Repository* (zdroj, úložiště). Slouží primárně jako DAO objekty presenterům. Každá modelová třída má svůj interface, který implementuje. Výhodou toho je, že při dependency injection je poté požadován objekt třídy implementující daný interface a ne konkrétní objekt nějaké třídy. Tím pádem je snadné případně modelové třídy měnit, aniž by to mělo vliv na zbytek aplikace.

3.2.2 Presenter obecně

Všechny třídy presenterů musí mít postfix *Presenter* (je to určeno Nette) a měly by (alespoň podle článku od tvůrce Nette frameworku) tvořit stromovou hierarchii [18]. Tzn. že všechny presentery by měly mít společného předka a každý presenter je buď vnitřní uzel nebo list. Vnitřní uzel je myšlen jako abstraktní třída a list jako třída deklarována s klíčovým slovem *final* (tzn. že od ní již nelze dědit).

Každý presenter má metody render*(), kde je hvězdička název pohledu (view). Tyto metody slouží jak k vytvoření pohledu (deklarací metody), tak k předávání dat do daných pohledů.

Společného předka všem presenterům tvoří abstraktní třída *BasePresenter*, která dědí od *Nette\Application\UI\Presenter*. Stará se o autorizaci uživatelů a výběr bočního menu pro určitou uživatelskou roli.

K autorizaci uživatelů používá *BasePresenter* třídu *Nette\Security\Permission*. Tato třída je zaregistrována v aplikaci jako služba *authenticator* a je zavolána pokaždé, kdy je třeba zjistit, zdali má uživatel s danou rolí právo přístupu na stránku.

Funkčnost této třídy je založena na 3 základních prvcích - role, zdroj a oprávnění. Role představují jednotlivé uživatelské role, které jsou v aplikaci definované (supervisor, developer a administrator). Zdroje pak představují samotné pohledy presenterů (view). Pomocí oprávnění je poté k jednotlivým rolím přiřazeno, ke kterým zdrojům mohou přistupovat. Tyto role mohou mezi

sebou svoje oprávnění dědit, takže pokud má ke zdroji oprávnění rodič, má ho i poté jeho potomek.

V tomto případě od sebe postupně dědí role vývojář od uživatele a administrátor od vývojáře. Vzniká tak požadovaná hierarchie přístupových práv, kdy vývojář může vše co uživatel a administrátor vše co vývojář a uživatel.

Seznam těchto autorizačních údajů (Access Control List) je zčásti obsažen v souboru `acl.neon`, jenž je obsažen ve složce `config` modulu Core. Dále je automaticky doplňován o zdroje a oprávnění, které se týkají datových struktur. O toto dodefinování se stará třída *AbstractStructurePresenter* z modulu Structures, která pro každou zaregistrovanou strukturu vytváří další 3 zdroje a oprávnění (podle uživatelských rolí). Tímto způsobem je zaručeno, že nebude nutné, při instalaci nové datové struktury do aplikace, měnit zmiňovaný ACL.

3.2.3 View obecně

View v Nette je řešeno pomocí Latte šablon, které jsou uloženy v souborech s příponou `.latte`. Název soboru šablony odpovídá danému pohledu presenteru. [3]

3.3 Modul Core

3.3.1 Třídní diagram

Na obrázku 3.2 je znázorněn třídní diagram modulu Core, který zachycuje základní metody, proměnné a vztahy mezi třídami.

3.3.2 Model

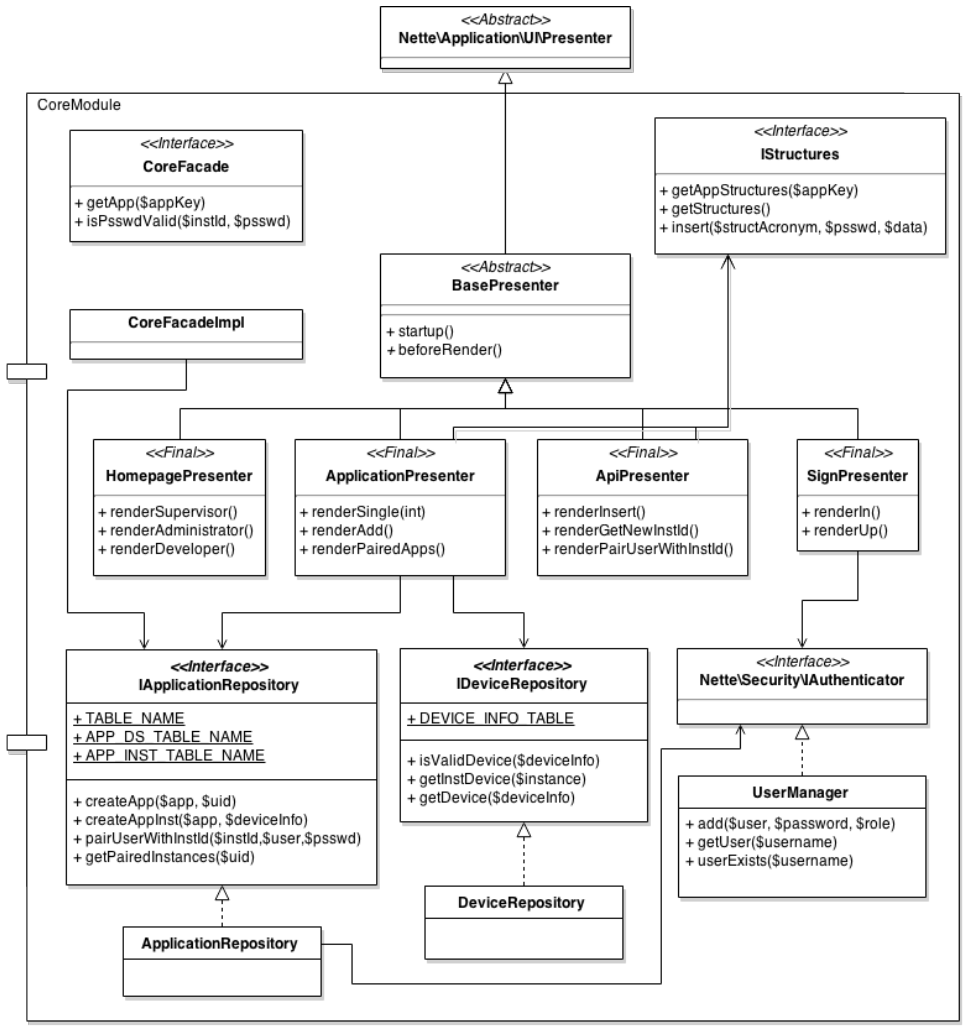
UserManager není běžná modelová třída, ale třída, která implementuje rozhraní *Nette\Security\IAuthenticator*. Jak napovídá název rozhraní, tato třída se stará o autentizaci (přihlašování a registrace) uživatelů. Její implementace je součástí Nette Sandboxu, což je základní (prázdný) Nette projekt.

Třída *ApplicationRepository* implementuje rozhraní *IApplicationRepository* a stará se zejména o vytváření nových aplikací a jejich instancí. Dále umožňuje například spárování instance s uživatelem či ověřit heslo k dané instanci.

DeviceRepository implementuje interface *IDeviceRepository* a umožňuje základní práci se zařízeními (vložení a získání).

Dále je ještě v modelové vrstvě třída *CoreFacadeImpl* implementující rozhraní *CoreFacade*, která vytváří jednoduché rozhraní nad celým modulem Core. To pak mohou využívat třídy z jiných modulů.

[3] Je ale i možné přímo nastavit danému pohledu konkrétní soubor se šablonou.



Obrázek 3.2: Třídní diagram modulu Core

Pro práci s datovými strukturami využívá modul Core interface *IStructures*. Ten předepisuje metody pro vložení dat do dané datové struktury, získání datových struktur konkrétní aplikace a podobné. Implementace tohoto rozhraní je obsažena v modulu *Structures*. Tímto způsobem je dosaženo menší provázanosti mezi moduly (tzv. princip dependency inversion) a také tím zanikne nežádoucí cyklická závislost mezi moduly.

3.3.3 Presenter

ApplicationPresenter má 4 pohledy. Výchozí (default) pohled slouží k zobrazení aplikací daného vývojáře. Další pohled *single* zobrazuje konkrétní jednu aplikaci, pohled *add* obsahuje formulář pro přidání aplikace a poslední pohled *PairedApps* zobrazuje spárované instance aplikací.

ApiPresenter se stará o zpracování HTTP požadavků od zařízení - vytváří tedy API aplikace. Každá metoda API odpovídá jednomu pohledu presenteru, tzn. že presenter má následující pohledy: *getNewInstId*, *pairUserWithInstId* a *insert*. Tyto pohledy mají společnou šablonu *response.latte*, která je prázdná a obsahuje pouze výpis jedné proměnné, která je do šablony předána (odpověď metody API).

HomepagePresenter presenter zjistí roli uživatele a zobrazí mu odpovídající domovskou stránku. Pro každou roli je domovská stránka jiná, takže tento presenter má tři pohledy podle uživatelských rolí: *supervisor*, *developer* a *administrator* a jeden pohled *default* sloužící k přesměrování na odpovídající domovskou stránku.

SignPresenter vytváří a obsluhuje přihlašovací a registrační formuláře uživatelů. Má dva pohledy. Pohled *in* slouží k zobrazení přihlašovacího formuláře a *up* k zobrazení registračního formuláře.

3.4 Modul Structures

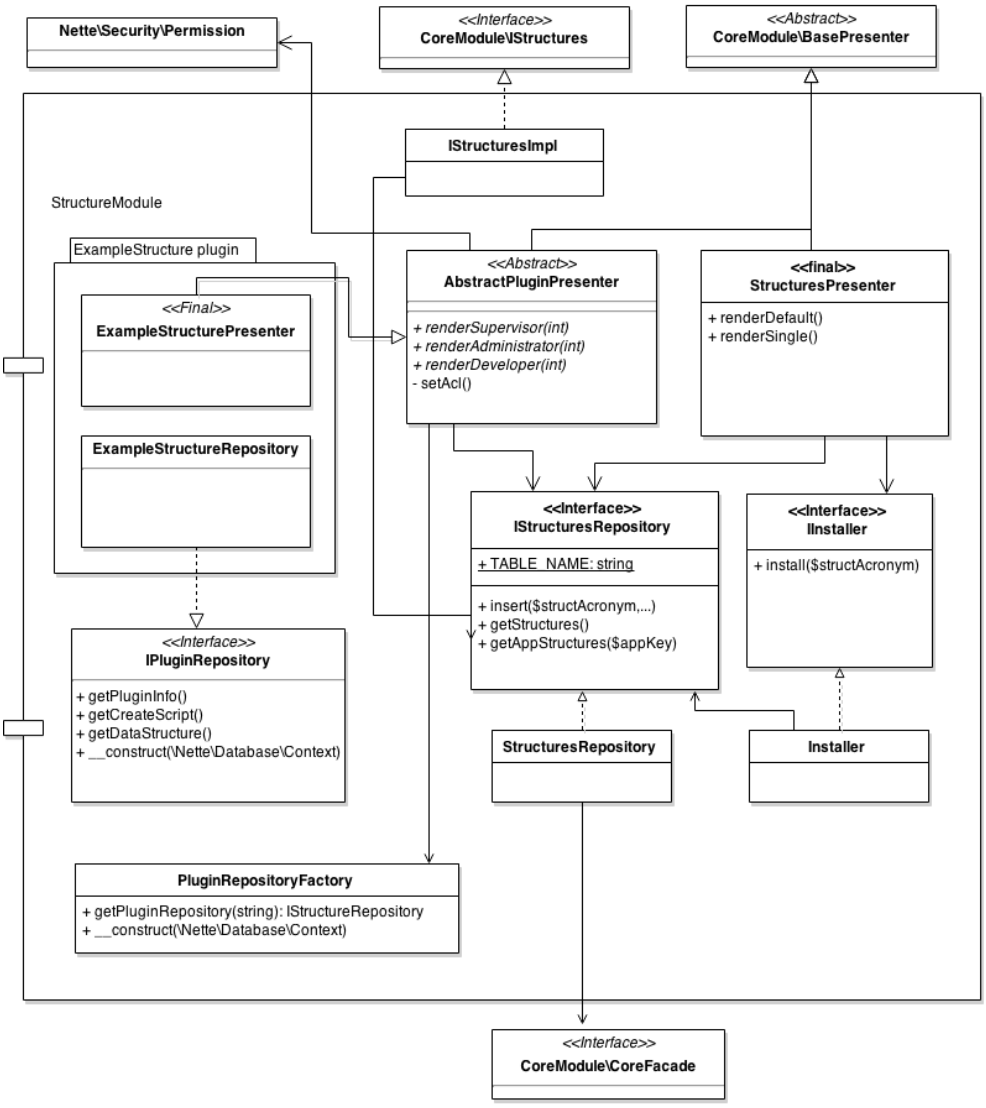
3.4.1 Třídní diagram

Na obrázku 3.3 je zachycen třídní diagram modulu Structures, který zachycuje základní metody, proměnné a vztahy mezi třídami. Zobrazuje také třídy vzorového pluginu datové struktury (*ExampleStructure*) a jejich závisloti na třídách tohoto modulu.

3.4.2 Plugin datové struktury

Každý plugin musí obsahovat určité údaje se kterými bude aplikace pracovat. V první řadě se jedná o název, popis, zkratku, název databázové tabulky a verzi pluginu. Tyto údaje musí být uloženy v indexovaném poli s následujícími indexy:

- name



Obrázek 3.3: Třídní diagram modulu Structures

- description
- acronym
- tablename
- version .

Nejdůležitejším údajem je zkratka (acronym) pluginu. Jedná se o jednoslovné označení (bez diakritiky), od kterého se odvozují názvy modelové a presenterové třídy.

Dále plugin musí obsahovat *DDL* skript, kterým je možné vytvořit v MySQL databázi tabulku. Každá pluginová databázová tabulka musí obsahovat atribut *inst_id* (integer), který identifikuje z které instance daný řádek pochází a dále atribut *datetime* (timestamp), který určuje, kdy byl daný řádek vložen. Z toho plyne, že *DDL* každého pluginu musí obsahovat tyto dva řádky:

```
‘inst_id‘ int(11) NOT NULL,
‘datetime‘ timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
```

Poslední informaci, kterou musí plugin obsahovat je indexované pole, které popisuje datovou strukturu. Klíče tohoto pole jsou názvy atributů v databázové tabulce a hodnoty jsou slovní popisy významu těchto atributů. Toto pole však v sobě neobsahuje povinné atributy tabulky (instanční id a časovou známku).

Stejně tak jako zbytek aplikace, plugin se bude skládat ze tří vrstev.

V modelové vrstvě bude třída, jejíž název se skládá ze zkratky pluginu s postfixem *Repository* (např. *ExampleRepository*), jenž musí implementovat rozhraní *IPluginRepository*. Toto rozhraní uděluje třídě povinnost implementovat 3 get metody - pro získání create skriptu tabulky, informací o pluginu a datové struktury. Třída tedy musí obsahovat informace zmíněné v odstavích výše. Dále rozhraní předepisuje konstruktor s parametrem typu *Nette\Database\Context*, kterým třída dostane objekt umožňující přístup k databázi.

V prezentační vrstvě je třída s názvem zkratky pluginu s postfixem Presenter (např. *ExamplePresenter*), jenž je potomkem třídy *AbstractStructuresPresenter*. Tento presenter musí implementovat 3 abstraktní metody *renderAdministrator()*, *renderSupervisor()*, *renderDeveloper()*, které představují pohledy na datovou strukturu ze třech uživatelských rolí. Svůj modelový objekt presenter získá z objektu *PluginRepositoryFactory*, jenž dědí od rodiče.

Plugin dále musí obsahovat 3 Latte šablony pro každou metodu render*() presenteru, tzn. soubory administrator.latte, supervisor.latte a developer.latte.

3.4.3 Model

Stejně jako u modulu Core, každá modelová třída implementuje svůj interface.

Třída *StructuresRepository* implementující *IStructuresRepository* slouží primárně ke vkládání dat do databáze, dále pak například k získání datových struktur přiřazených k dané aplikaci.

Installer je třída, která slouží k instalaci datových struktur do aplikace. Implementuje právě jednu metodu *install()* z rozhraní *IInstaller*.

PluginRepositoryFactory je továrna na modelové objekty pluginů. Presentery datových struktur by mohly získávat svoje modelové objekty přes dependency injection, jenže by se muselo při každé instalaci pluginu měnit nastavení dependency containeru. Tato továrna celý problém vyřeší a presentery tak mohou pomocí metody *getPluginRepository()* svůj modelový objekt jednoduše získat.

Poslední třídou v modelové vrstvě je třída *IStructuresImpl*, která implementuje již zmiňovaný interface *IStructure* z modulu Core.

Modelová vrstva pak ještě obsahuje rozhraní *IPluginRepository*, které předepisuje metody, které musí implementovat modelové třídy pluginů.

3.4.4 Presenter

Presenterová vrstva obsahuje pouze dva presentery.

Prvním z nich je abstraktní presenter *AbstractPluginPresenter*, který je rodič všech presenterů z pluginů datových struktur. Více jsem se o něm již zmiňoval v popisu pluginu v sekci 3.4.2 a také v sekci 3.2.2 o presenterech.

Druhým je *StructuresPresenter*, který zobrazuje datové strutkury a také obsluhuje a vytváří formulář pro instalaci pluginu.

3.5 Databáze

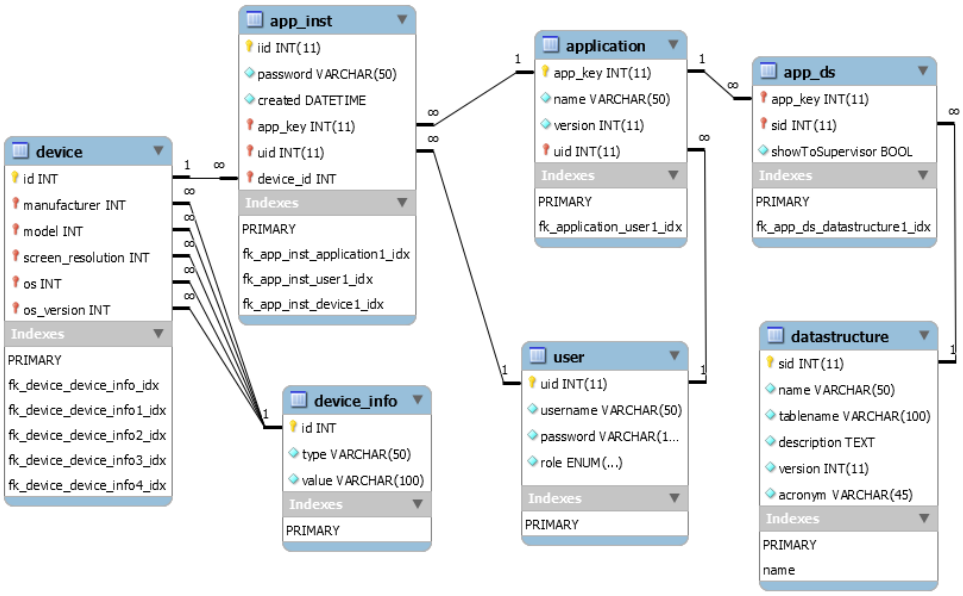
3.5.1 Schéma

Na obrázku 3.4 je znázorněno schéma databáze. V této sekci stručně popíši jednotlivé tabulky.

datastructure obsahuje informace o datových strukturách nainstalovaných do Šmaku jako je název datové struktury (atribut *name*), název tabulky datové struktury (*tablename*) a její verze a popis.

app_ds reprezentuje vztah n ku n mezi tabulkami *application* a *datastructure*. Dále obsahuje také atribut *showToSupervisor*, který říká, zdali se daná struktura má zobrazit supervisorovi.

application obsahuje data o aplikaci jako je její název, verze a dále má cizí klíč *uid*, který určuje uživatele, který danou aplikaci v systému zaregistroval.



Obrázek 3.4: Databázový model

app_inst reprezentuje instanci aplikace. Cizí klíč *app_key* určuje o jakou aplikaci se jedná, dále *uid* určuje uživatele, jenž si danou instanci spároval. Posledním klíčem je *device_id*, který odkazuje na zařízení, ze kterého instance pochází.

user je tabulka obsahující informace o uživateli zaregistrovaném v systému. Obsahuje uživatelské jméno (*username*, hash jeho hesla a roli (výčtový typ ENUM s 3 hodnotami Administrator, Developer a Supervisor).

device je tabulka reprezentující zařízení s danými vlastnostmi (ne tedy jedno konkrétní). Jeden typ zařízení tedy může mít více instancí. Vlastnosti tohoto zařízení jsou určeny cizími klíči k tabulce *device_info*.

device_info se skládá ze tří atributů. Kromě primárního klíče *id* obsahuje ještě atribut *type*, který určuje o jakou informaci o zařízení se jedná (výrobce, model ap.), v atributu *value* je poté hodnota.

Informace o zařízení by mohly být uloženy rovnou v tabulce *device*, ale protože se budou jednotlivé údaje často opakovat (například výrobce, či operační systém), rozhodl jsem se pro řešení zmíněné výše. Ušetří se tak zejména místo v databázi a také vyhledávání podle těchto parametrů bude efektivnější.

Dále bude databáze obsahovat tabulky jednotlivých datových struktur, jejich vlastnosti jsou popsány v kapitole 3.4.2.

3.6 Návrh klientské části aplikace

Stejně jako má Google Analytics tzv. Tracker, který vytváří rozhraní pro komunikaci s GA na straně klienta, je v plánu vytvořit podobný program také pro Šmak. V této části nebudu dělat konkrétní softwarový návrh, ale spíše navrhnu funkce, které by měl klientský Šmak mít.

Hlavní funkcí bude odstínění komunikace se Šmakem od protokolu HTTP. Aplikace na zařízení nebude muset sama vytvářet HTTP požadavky a posílat je na server. O to se bude starat právě klient Šmaku a volání metod API ze strany aplikace se tím pádem zjednoduší pouze na běžné volání metod objektu.

Pokud bude posíláno v kratším časovém období více insert požadavků (stejné datové struktury), bude klient tyto požadavky shromažďovat a posílat je pouze v jednom požadavku. Tím se docílí zejména menší zátěže na straně serveru (více v sekci 4.2).

Dále by ještě mohla být implementovaná ochrana proti zahlcení, tzn. že by klient omezoval počet požadavků odeslaných na server za nějaké časové období (podobně jako to má GA).

Tato klientská část bude samozřejmě potřeba naprogramovat pro každý operační systém zvlášť. Počítá se hlavně s verzí pro Android a iOS, v budoucnu ale může vzniknout také verze pro Windows či nějaký další operační systém.

3.6.1 Bezpečnost

Veškerá data mezi Šmakem a zařízením se přenáší klasickým protokolem HTTP, který nepoužívá žádné šifrování. Přenášená data tedy mohou být třetí stranou přečtena. V případě tohoto systému se nejedná až o tak velký problém, protože se nepřenáší citlivá data, která by mohla být třetí stranou zneužita. Avšak pokud třetí strana bude přenášená data nějakým způsobem měnit či sama vytvářet, budou poté data v systému nepřesná, či uplně jiná od skutečnosti, což už problém je.

Taková situace může nastat ve chvíli, kdy si chce uživatel (supervisor) spárovat svou instanci aplikace v zařízení se Šmakem. Pokud by se ke spárování nevyužívalo heslo získané při vytvoření nové instance, mohl by takový požadavek zaslat kdokoliv (instanční id náhodně vygeneruje, uživatelské jméno zadá své). Pokud by instanční id shodovalo s nějakým v systému, které ještě není spárované, tak by třetí strana takto mohla snadno ukradnout cizí instanci a uživatel, kterému opravdu patří, by neměl šanci ji získat zpět.

Je samozřejmé, že dané párovací heslo je možné odposlechnout při vytváření či párování instance, jen je to méně pravděpodobné, protože k těmto událostem bude docházet méně často.

Podobný problém nastává při vkládání dat. Kdokoliv, kdo vytvoří validní požadavek, může vložit data k cizí instanci aplikace. V tomto případě také částečně tento problém řeší heslo předávané při vkládání, jen s tím rozdílem,

že vkládání bude častějším jevem a tím pádem bude možné s větší pravděpodobností třetí straně heslo získat.

Problémy tohoto typu vyřeší použití zabezpečeného (šifrovaného) protokolu HTTPS, který umožňuje ověřit identitu obou stran a zamezuje podvržení, či změně přenášených dat. Pokud bude komunikace probíhat pomocí tohoto protokolu, bude komunikační heslo znát pouze systém Šmak a dané zařízení a nebude možné ho odposlechnout během přenosu. Tím pádem budou dva výše zmíněné problematické požadavky zabezpečeny a nebude prakticky možné je zfalšovat nebo změnit.

3.6.2 Porovnání s Google Analytics

Jelikož je tento projekt teprve v počáteční fázi, je těžké ho srovnávat s takovým kolosem jako je Google Analytics. Co se týká základních funkcí (a těch nejdůležitějších) jako je vytvoření nové aplikace a sběr dat s té aplikace, poskytuje Šmak i Google Analytics v principu to samé. V obou případech je základní používání aplikace a nabízené rozhraní velmi jednoduché.

V čem je Analytics každopádně lepší a propracovanější je samotná práce s daty, protože poskytuje širokou škálu možností, jak si data zobrazit a pracovat s nimi. Systém Šmak ve svém prototypu nabízí pouze jednoduché zobrazení přijatých dat s jednoduchými možnostmi filtrování. Avšak pokročilejší práce s daty je samozřejmě do budoucna plánovaná (více v sekci 5.7.2).

Na druhou stranu Šmak umožňuje sbírat data přesně taková, jaká si zadavatelé přejí. GA je omezený na určitou množinu typů dat (podrobněji rozebraná v sekci 2.1.1.2), která je sice navržena tak, aby pokryla požadavky co nejširší skupiny zákazníků, ale svou obecností je také trochu svazuje.

Další výrazný rozdíl, který mezi těmito dvěma aplikacemi stojí, je o tom, kdo vlastně ke sbíraným datům přistupuje. GA je primárně určen pro vývojáře, kteří zkoumají, jak se chovají uživatelé jejich webových stránek nebo aplikací. Nikdo jiný k těmto datům přístup nemá a v případě GA by to nejspíše nemělo ani žádný praktický význam.

Oproti tomu Šmak poskytuje data také uživatelům od kterých pochází. V praktickém případě se bude jednat o rodiče, kteří budou mít přehled nad daty z aplikací svých dětí, nebo učitele, jenž budou mít možnost kontrolovat své žáky.

Kapitola 4

Testování a optimalizace

Tato kapitola se zabývá výkonnostním testováním a optimalizací aplikace, kde jsou porovnány například různé úložné databázové enginy a také výkon na běžném PHP a PHP HVVM. Na konci kapitoly je popsáno a vyhodnoceno také průběžné systémové testování.

4.1 Databáze

Prvním krokem při optimalizaci databázové tabulky je především vybrat správný úložný engine. Každý z nich se hodí na trochu jiné použití, takže použití nesprávného způsobí na více zatížených databázích propastné rozdíly ve výkonu.

Dotazy nad tabulkou v databázi lze optimalizovat pomocí indexů. Jedná se většinou o stromové struktury (B-stromy), postavené nad jedním, či více sloupci v tabulce. Pomocí indexů může poté databáze velmi rychle data řadit či v nich vyhledávat [19].

4.1.1 Optimalizace tabulek datových struktur

Nejvytíženějšími tabulkami v databázi budou ty, jenž obsahují data z aplikací. Do těchto tabulek se bude velmi často zapisovat, méně často číst a v podstatě nikdy se nebudou data měnit, či mazat. Je tedy nutné aby úložný engine zvládal souběžné zapisovaní s občasnými dotazy (selecty). Podle [19] by na toto použití měl být vhodný úložný engine MyISAM, což také potvrdily mé testy [4].

Testů těchto tabulek jsem provedl několik druhů. První testem jsem měřil přímo databázi (nikoliv aplikaci). Na začátku testu byla testovaná tabulka (instId (int), apid (int), datetime (datetime), text (text)) prázdná. 10 vláken se staralo o neustálý zápis dat s náhodně generovaným sloupcem instId do databáze a 5 vláken provádělo výběry dat z této tabulky podle náhodného

[4] Testy jsem prováděl pomocí programu Apache JMeter

Tabulka 4.1: Výsledky testu 1

Úložný engine	Počet insertů	Počet selectů
MyISAM	2757093	2597
InnoDB	34029	6060

Tabulka 4.2: Výsledky testu 2

Úložný engine	Počet insertů	Počet selectů
MyISAM	3265803	10069
InnoDB	30567	9052

instId. Pokud by instId nebylo náhodně generováno, ale bylo pevné, prováděly by se pokaždé stejné dotazy což neodpovídá reálné situaci a navíc by byla ve velké míře používána cache dotazů, což by výrazně jednotlivé dotazy zrychlovalo. Každé z 5 vláken určených k výběrům dat po dotazu 100-200ms čeká a poté provede další.

Jeden test trval 10 minut, pouštěl jsem je 3x za sebou a výsledky zprůměroval. Výsledky tohoto testu jsou zobrazeny v tabulce 4.1 a jasně z nich vyplývá, že engine MyISAM zvládá souběžný zápis mnohem lépe než InnoDB avšak na druhou stranu InnoDB zvládl udělat více selectů.

Druhý test byl shodný s tím prvním, ale na sloupečku instId, podle kterého probíhají v testu selecty, jsem vytvořil index. Výsledky jsou zaznamenány v tabulce 4.2. Engine MyISAM si výrazně polepšil jak v insertech tak i v selectech, kde dokonce překonal i InnoDB, který byl v minulém testu v této položce lepší. Druhý engine se zato mírně pohoršil v počtech insertů a jak se dalo čekat, počet selectů se stejně jako u prvního enginu zvýšil.

4.2 Aplikace

4.2.1 Optimalizace API

Metody API *getNewInstId()* a *pairUserWithInstId()* nebudou v praxi tak často volány jako metoda *insert()*. A právě u této metody se nabízí velmi dobrý způsob jak ji zefektivnit.

Pomocí zabudované Nette knihovny Tracy (Laděnka) lze jednoduše v aplikaci měřit, jak dlouho trvá určitý požadavek a navíc lze i změřit, kolik času z tohoto požadavku zabere samotný dotaz do databáze. Pomocí Laděnky jsem změřil 10 požadavků o vložení dat a výsledky zprůměroval. Samotné vykonání dotazu (execution time) trvalo v průměru 220ms avšak samotný dotaz do databáze trval kolem 40ms, takže tvořil jen zhruba šestinu celého požadavku. Zbytek času zabralo vytvoření instance aplikace pro daný požadavek, dále kontroly validity dat a také samotné vytvoření připojení k databázi. V praxi

Tabulka 4.3: Výsledky testu PHP a PHP-HHVM

Verze PHP	Počet insertů
PHP	20420
PHP HHVM	25456

by to aplikaci, která chce poslat 5 požadavků o vložení do jedné datové struktury, zabralo zhruba 1100ms.

Jak by se ale změnila situace, pokud by v jednom požadavku bylo možné poslat více řádků dat? Upravil jsem aplikaci tak, aby byla schopná takovéto požadavky přijímat a provedl stejný test jako v předchozím odstavci s tím rozdílem, že jsem posílal 5 řádků dat najednou. Doba vykonání požadavku vzrostla v průměru o 15ms na 235ms a doba dotazu vzrostla na zhruba 45ms. Rozdíl v časech je opravdu minimální (až zanedbatelný) a bylo dosaženo mnohem vyššího výkonu.

Donutit ale aplikace aby posílaly takovéto hromadné požadavky je v praxi nemožné i když je to pro obě strany výhodnější. Možným řešením se zabývam v kapitole 3.6.

4.2.2 Testování na HipHop Virtual Machine

Otestoval jsem výkon aplikace na běžném PHP a pak také na PHP běžícím na HHVM. Testy probíhaly na stejném hardwaru, operačním systému (Linux Mint 17.1) a na stejném serveru Apache (verze 2.4.7). V testu trvajícím 2 minuty jsem měřil, kolik dokáže aplikace přijmout insert požadavků. Zprůměrované výsledky 3 testů jsou uvedené v tabulce 4.3 Z výsledků jasně plyne, že s HHVM opravdu poskytuje vyšší výkon, v tomto případě je rozdíl téměr 25 %.

4.2.3 Systémové testy

Tyto testy jsem průběžně prováděl během celé fáze implementace. Slouží k otestování, jak aplikace funguje jako celek. Testoval jsem všechny funkční požadavky uvedené v tabulce 1.1 podle scénářu případů použití v sekci 1.6. Výsledky jsou uvedeny v tabulce 4.4. Jak je vidět, až na jeden požadavek jsou všechny splněny. Požadavek na instalaci datových struktur je splněn částečně, protože měl nižší prioritu a zadavateli současná implementace prozatím stačí.

Tabulka 4.4: Výsledky systémových testů

Požadavek	Splněno
Přihlášení a registrace	Ano
Uživatelské role	Ano
Vytvoření profilu vlastní aplikace	Ano
Výběr datových struktur	Ano
Nastavení zobrazovaných datových struktur uživatelům	Ano
Poskytnutí autentizačních informací	Ano
Spárování aplikace s jejím uživatelem	Ano
Přijem dat	Ano
Zobrazení dat supervisorovi	Ano
Zobrazení dat z aplikace	Ano
Instalace datových struktur	Částečně

Kapitola 5

Dokumentace

V této sekci popíši základní používání aplikace Šmak. Zaměřuji se zejména na používání API, kde popíši požadované parametry a metody požadavku. K dosažení maximální příkladnosti uvádím také u každé metody API vzorový HTTP požadavek. Dále zde také uvádím návod na nasazení aplikace na server a systémové požadavky.

5.1 Datové struktury v prototypu

V prototypu, který jsem vypracoval v rámci této práce, jsou implementovány dvě vzorové datové struktury. Jedna umožňuje sbírat výjimky a chyby a druhá je určena pro měření a zaznamenávání událostí. Tyto pluginy slouží pro demonstrační účely a je velmi pravděpodobné, že budou v budoucnu změněny.

5.2 Registrace do aplikace

První, co je nutné udělat před tím, než začnete aplikaci Šmak používat, je registrace do aplikace. Navštivte adresu *název-serveru/www* kde by se měl zobrazit přihlašovací formulář. V horní černé liště zvolte odkaz *Registrovat se* a vyplňte registrační formulář. Ve spodní části formuláři vyberte jakou roli chcete mít. Pokud chcete zobrazovat pouze data z nějaké své instance aplikace, zvolte roli *Uživatel*. Jinak pokud chcete využívat i další funkce Šmaku, jako je registrace své aplikace, zvolte možnost *Vývojář*. Potvrzením formuláře dokončíte registraci a již se můžete do Šmaku přihlásit.

5.3 Přihlášení do aplikace

Přihlášení je možné provést pomocí přihlašovacího formuláře, který se zobrazí, pokud nejste přihlášeni na adrese *název-serveru/www*. Do formuláře zadejte údaje, které jste uvedli při registraci. Pokud jsou údaje správné, zobrazí se

vám po potvrzení formuláře domovská stránka odpovídající vaší roli. K přihlašovacímu formuláři se lze také dostat pomocí odkazu *Přihlásit se* v horní černé liště.

5.4 Vytvoření nové aplikace

Pokud je uživatel přihlášen jako vývojář (nebo administrátor), má možnost zaregistrovat do Šmaku svou aplikaci, ze které bude chtít sbírat data. V levé liště stačí zvolit možnost *Přidat novou aplikaci*. Uživateli se zobrazí jednoduchý formulář, kde vyplní základní údaje o aplikaci a zvolí, které datové struktury bude chtít ze své aplikace sbírat. Jakmile potvrdí formulář, aplikace se uloží do systému a je možné ji najít na stránce *Moje aplikace* (odkaz taktéž v lévém bočním menu).

5.5 Zasílání požadavků

5.5.1 Registrace nové instance aplikace

Registrace nové instance aplikace (pomocí metody *getNewInstId* je možné provést pomocí GET nebo POST HTTP požadavku. Tento požadavek je nutné poslat na adresu *název-serveru/www/core.api/getnewinstid*. Prvním parametrem požadavku je *appKey*, což je číslo aplikace, které je aplikaci přiděleno při registraci a dá se zjistit v sekci *Moje aplikace*. Dalším parametrem je *deviceInfo*, ve kterém se předávají informace o zařízení. Výsledný požadavek může vypadat například takto:

```
POST /www/core.api/getnewinstid HTTP/1.1
Host: localhost
appKey=1&
deviceInfo={
"manufacturer":"Apple",
"model":"iPhone",
"screen_resolution":"1280x1024",
"os":"iOS",
"os_version":"7.1.2"
}
```

Pokud je požadavek v pořádku odpověď serveru může vypadat například následovně:

```
{"instId":31,"psswd":"2E9gtHMW1G"}
```

Server vrátil pole o dvou prvcích. Nové instanční id (*instId*) a heslo (*psswd*), které od teď bude aplikace používat při komunikaci se Šmakem. Tyto údaje

je nutné uložit někam, kde budou k dispozici i při dalším spuštění aplikace, tedy například do konfiguračního souboru nebo databáze. Od teď už je možné posílat do Šmaku data, či instanci spárovat s nějakým uživatelem.

5.5.2 Párování instance aplikace s uživatelským účtem

Požadavek o spárování instance s uživatelem se posílá GET nebo POST požadavkem na adresu *název-serveru/www/core.api/pairuserwithinstid*. Parametry tohoto požadavku jsou instanční id (instId), komunikační heslo (psswd) a uživatelské jméno uživatele zaregistrovaného v systému Šmak (username). Odpověď serveru je buď true nebo false v závislosti na tom, zdali spárování proběhlo úspěšně. Požadavek (v tomto případě POST) může vypadat například takto:

```
POST /www/core.api/pairuserwithinstid HTTP/1.1
Host: localhost
instId=13&
psswd=SjnCRtekvN&
username=supervisor
```

5.5.3 Odesílání dat

Odesílání dat probíhá taktéž pomocí POST nebo GET požadavků (vzhledem k tomu, že data odesíláme, hodí se spíše POST požadavek). Adresa na kterou se posílá požadavek je *název-serveru/www/core.api/insert*. Parametry požadavku jsou název struktury (*structName*), komunikační heslo (*psswd*) a *data*. Požadavek pak může vypadat například následovně:

```
POST /www/core.api/insert HTTP/1.1
Host: localhost
structName=Exceptions&
psswd=tajneheslo&
data=[{
"inst_id":"Číslo instance aplikace",
"exception":"Text výjimky"
}]
```

Pokud bude pole v parametru *data* obsahovat nějaké položky navíc, budou v průběhu zpracování zahozeny. Naopak pokud budou některé povinné položky chybět, nebude vůbec požadavek zpracován. Dále je také při vícenásobném požadavku o vložení testováno, zdali jednotlivé řádky neobsahují různé hodnoty *inst_id*. Pokud by k takové situaci došlo, bude požadavek taktéž zamítnut.

Ukázka validního POST požadavku je uvedena u každé datové struktury, která je k aplikaci přiřazena. Lze ji najít po zvolení konkétní aplikace v sekci *Moje aplikace*.

5.6 Instalace nového pluginu datové struktury

Vkládání nových pluginů datových struktur je zatím částečně prováděno ručně a částečně automatizovaně. V budoucnu je v plánu plné zautomatizování této procedury, avšak v tuto chvíli to od zadavatelů vyžadováno není, takže jsem ve své práci implementoval pouze tu část, která by mohla případně dělat větší problémy, pokud by automatizována nebyla.

Pro tento příklad uvedu plugin, který se skládá z následujících souborů:

- ExampleRepository.php
- ExamplePresenter.php
- supervisor.latte
- developer.latte
- administrator.latte

V první fázi je nutné nahrát příslušné třídy a šablony pluginu do správných adresářů v aplikaci. Modelová třída (*ExampleRepository*) patří do adresáře *app/StructuresModule/model/PluginRepositories*. Třída presenteru (*ExamplePresenter* patří do složky *app/StructuresModule/presenters* k ostatním presenterům. Pro šablony (.latte soubory) je nutné vytvořit vlastní složku s názvem zkratky pluginu (*Example*) v adresáři *app/StructuresModule/templates*.

Jakmile jsou soubory na svém místě, je již snadné datovou strukturu do aplikace přidat. Stačí se jako administrátor přihlásit do systému a na stránce *Správa datových struktur* v sekci *Instalace nového pluginu* zadat do formuláře zkratku této datové struktury (*Example*) a kliknout na instalovat. Aplikace zkontroluje, zdali jsou požadované soubory na svém místě, zaregistruje plugin do databáze a vytvoří mu jeho databázovou tabulku. Od této chvíle je možné datovou strukturu normálně používat.

5.7 Instalace systému Šmak na server

5.7.1 Softwarové požadavky

PHP je velmi platformě nezávislé, takže by mělo bez problémů běžet na různých operačních systémech i webových serverech.

Systém Šmak jsem vyzkoušel jak na operačním systému Windows (7) tak i Linux (Mint 17.1 KDE), kde bez problémů fungoval. Neměl by být problém ani s jinými verzemi těchto systémů.

Jako webové servery jsem vyzkoušel, jak velmi rozšířený Apache verze 2.4.7 tak i Nginx verze 1.7.10, taktéž bez jakýchkoliv problémů.

Před samotnou instalací Šmaku je nutné zjistit, zdali jsou splněny požadavky na server pro Nette. K tomu slouží nástroj *Requirements-checker*, který

je umístěn na přiloženém CD ve složce install, jenž stačí nahrát na webový server. Jakmile je tento nástroj nahrán, stačí přistoupit na stránku *název-server/Requirements-checker/checker.php*, kde se zobrazí přehledná tabulka, která obsahuje názvy požadavků a zdali jsou splněny. Nette vyžaduje PHP verze 5.3.1 a vyšší, dále je z velké části kompatibilní s PHP HHVM (verze 5.6.99), na kterém jsem systém Šmak vyzkoušel a běžel bez nejmenších problémů [20].

Jak jsem již zmiňoval v analytické části této práce, databázi jsem zvolil MySQL. Při mém vývoji jsem využíval verzi 5.6.15, ale neměl by být problém ani s jinými verzemi, kde by jediný rozdíl mohl nastat ve výkonu databáze.

5.7.2 Instalace

Pokud je server vhodně nastaven, může se pokračovat v instalaci. Dále je nutné importovat databázové schéma (například pomocí adminstračního nástroje *phpMyAdmin* ze souboru *src/smak-db.sql*.

Pokud i to proběhne v pořádku, může se přejít k instalaci systému Šmak. K tomu je potřeba nahrát obsah adresáře *src/smak* na webový server (například pomocí FTP klienta *Total Commander*. Po nahrání je potřeba nastavit přístupová práva ke složkám *temp* a *log*, tak aby do nich bylo možné zapisovat a to ze všech rolí. Na unixových systémech to lze provést následujícím příkazem:

```
chmod -R a+rw temp log
```

Po provedení těchto kroků by již měl systém bez problémů fungovat.

Závěr

Během tvorby této bakalářské práce jsem získal mnoho nových znalostí a zkušeností. V rešeršní části práce jsem vcelku podrobně nastudoval Google Analytics, které jsem do té doby používal pouze na sledování webových stránek a nevěděl jsem, že umožňuje také sbírat data z mobilních aplikací. Poznal jsem také lehce jeho konkurenta Flurry. Tato část práce mi pomohla uspořádat myšlenky k vytvoření podobného systému, což jsem velmi ocenil během návrhu.

Dále jsem vcelku podrobně nastudoval databázi MySQL, přičemž jsem získal nové znalosti ohledně efektivního používání tohoto úložiště. Některé získané vědomosti šly i mimo rámec této bakalářské práce, ale alespoň mi pomohly získat větší přehled v této problematice.

V rámci návrhu jsem si ověřil zkušenosti a znalosti získané během mého studia. Některé z nich, jako je například princip dependency inversion, poprvé v praxi.

Během implementace jsem nenarazil na žádné větší problémy a tak tato část práce patřila k těm jednodušším. K tomu přispěl zejména softwarový návrh, na kterém jsem si dal hodně záležet, protože vím, že je to jedna z nejdůležitějších (ne-li nejdůležitější) částí softwarového projektu.

Pokud bych to měl shrnout, jsem s výslednou prací vcelku spokojen, protože splnila požadavky zadavatelů a já během tvorby získal mnoho nových vědomostí. Mám také dobrý pocit z toho, že moje práce bude v budoucnu prakticky využita.

Další vývoj projektu

Tento projekt po odevzdání této bakalářské práce ani zdaleka nekončí a bude se pokračovat na jeho vývoji. Fukční prototyp vzniklý během této bude dále konkrétněji rozšiřován a vylepšován. Bude probíhat zejména vytváření dalších pluginů datových struktur podle požadavků zadavatelů, které je plánované na léto 2015. Poté bude aplikace nasazena do několikaměsíčního testovacího

provozu, kdy bude již přijímat data z opravdových zařízení. Během tohoto testovacího provozu si budou moci vyzkoušet aplikaci i případní zájemci.

Dále je v plánu vytvořit klientskou část Šmaku (více popsanou v (3.6), která usnadní komunikaci na straně klienta a také umožnit zabezpečenou komunikaci pomocí protokolu HTTPS.

Literatura

[1] Google: Platform Overview [online]. Dostupné z: https://developers.google.com/analytics/devguides/platform/, [cit. 8.2.2015].

[2] Google: Event Tracking - Android SDK v4 [online]. Dostupné z: https://developers.google.com/analytics/devguides/collection/android/v4/events, [cit. 9.2.2015].

[3] Google: Enhanced Ecommerce Tracking - Android SDK v4 [online]. Dostupné z: https://developers.google.com/analytics/devguides/collection/android/v4/enhanced-ecommerce, [cit. 9.2.2015].

[4] Google: Custom Goals - Android SDK v4 [online]. Dostupné z: https://developers.google.com/analytics/devguides/collection/android/v4/enhanced-ecommerce, [cit. 9.2.2015].

[5] Google: Custom Dimensions & Metrics - Feature Reference [online]. Dostupné z: https://developers.google.com/analytics/devguides/platform/customdimsmets, [cit. 9.2.2015].

[6] Google: Screens - Android SDK v4 [online]. Dostupné z: https://developers.google.com/analytics/devguides/collection/android/v4/screens, [cit. 9.2.2015].

[7] Google: Crashes & Exceptions - Android SDK v4 [online]. Dostupné z: https://developers.google.com/analytics/devguides/collection/android/v4/exceptions, [cit. 9.2.2015].

[8] Google: User Timings - Android SDK v4 [online]. Dostupné z: https://developers.google.com/analytics/devguides/collection/android/v4/usertimings, [cit. 10.2.2015].

[9] Google: Measurement Protocol Developer Guide [online]. Dostupné z: https://developers.google.com/analytics/devguides/collection/protocol/v1/devguide, [cit. 10.2.2015].

[10] Google: Google Analytics Collection Limits and Quotas [online]. Dostupné z: https://developers.google.com/analytics/devguides/collection/android/v4/limits-quotas, [cit. 10.2.2015].

[11] Yahoo: Custom Events [online]. Dostupné z: https://developer.yahoo.com/flurry/docs/analytics/gettingstarted/events/android/, [cit. 5.2.2015].

[12] Lewandowski, M.: Help For Choosing Your Mobile Application Analytics Platform [online]. Dostupné z: http://www.degdigital.com/blog/Help-For-Choosing-Your-Mobile-Application-Analytics-Platform, [cit. 10.2.2015].

[13] W3Techs: Historical trends in the usage of server-side programming languages for websites [online]. Dostupné z: http://w3techs.com/technologies/history_overview/programming_language, [cit. 18.11.2014].

[14] Ed Lecky-Thompson, S., Nowicki: *PHP 6: programujeme profesionálně*. ISBN 8025131270.

[15] Paroski, D.: Speeding up PHP-based development with HHVM [online]. Dostupné z: https://www.facebook.com/notes/facebook-engineering/speeding-up-php-based-development-with-hiphop-vm/10151170460698920, [cit. 2.2.2015].

[16] Oracle: MySQL Customers [online]. Dostupné z: http://www.mysql.com/customers/, [cit. 2.2.2015].

[17] Neuveden: Model-View-Presenter (MVP) [online]. Dostupné z: http://doc.nette.org/cs/0.9/model-view-presenter, [cit. 14.3.2015].

[18] Grudl, D.: Návrh struktury presenters/views [online]. Dostupné z: http://phpfashion.com/navrh-struktury-presenters-views, [cit. 14.3.2015].

[19] Baron Schwartz, V. T. J. D. Z. A. L. D. J. B., Peter Zaitsev: *MySQL profesionálně - optimalizace pro vysoký výkon*. ISBN 978-80-7413-035-9.

[20] Neuveden: Požadavky Nette Framework [online]. Dostupné z: http://doc.nette.org/cs/2.3/requirements, [cit. 5.2.2015].

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

HTTP Hypertext Transfer Protocol

GA Google Analytics

PHP Hypertext Preprocessor

HHVM HipHop Virtual Machine

MVC Model, View, Controller

MPV Model, View, Presenter

GUI Graphic User Interface

EE Enterprise Edition

HTML Hypertext Markup Language

CSS Cascading Stylesheet

JS Javascript

DAO Data Access Object

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `smak` ..... zdrojové kódy implementace
  - `smak-db` ..... DDL databázový skript
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS
- `doc` ..... dokumentace
  - `source` ..... dokumentace zdrojového kódu
  - `app` ..... dokumentace aplikace
- `install` ..... instalační nástroje
  - `Requirements-checker` ..... kontrola požadavků Nette