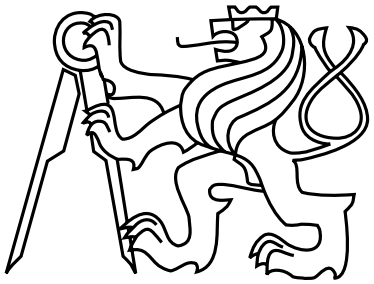
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky



Bakalářská práce

Systém pro monitorování průmyslových strojů

Jakub Dundálek

Vedoucí práce: Ing. Tomáš Richta

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Inteligentní systémy

7. ledna 2011

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 7. ledna 2011 .....

Abstract

The goal of this Bachelor thesis is to design and implement a server-side application for industrial machines monitoring. Data are presented to users via web interface.

The work describes possibilities and design of communication protocol between monitoring unit and server. Then it contains description of application design and implementation with emphasis on security and extensibility.

Abstrakt

Tato práce se zabývá návrhem a implementací serverové aplikace systému pro monitorování průmyslových strojů. Server přijímá a ukládá data z monitorovací jednotky. Zobrazení dat uživatelům je řešeno pomocí webového rozhraní.

Práce nejdříve popisuje možnosti návrhu komunikačního protokolu mezi jednotkou a serverem. Dále obsahuje návrh samotné aplikace a popis implementace s důrazem na bezpečnost a rozšiřitelnost.

Obsah

1 Úvod **1**

2 Popis problému, specifikace požadavků **2**
2.1 Hlavní kritéria . . . 2
2.2 Specifikace požadavků . . . 2
2.2.1 Aplikační rozhraní . . . 3
2.2.2 Role uživatelů . . . 3
2.2.3 Uživatelské rozhraní . . . 3

3 Analýza a návrh komunikace **5**
3.1 Vlastnosti spojení . . . 5
3.1.1 Network address translation (NAT) . . . 5
3.1.2 Navazování komunikace . . . 5
3.2 Formát přenosu dat . . . 6
3.2.1 Data . . . 6
3.2.2 Délka odesílacího intervalu . . . 7
3.3 Protokol . . . 7
3.3.1 TCP . . . 7
3.3.2 HTTP . . . 7
3.3.3 UDP . . . 8
3.3.4 Proxy server . . . 8
3.3.5 Komprese . . . 8
3.3.6 Šifrování . . . 8
3.4 Shrnutí . . . 9
3.5 Ukázka komunikace . . . 9

4 Analýza a návrh aplikace **11**
4.1 Architektura aplikace . . . 11
4.2 Použité technologie . . . 11
4.2.1 Programovací jazyk . . . 11
4.2.2 Knihovny a frameworky . . . 12
4.3 Uložení dat . . . 13
4.3.1 Data ze strojů . . . 13
4.4 Datový model . . . 13
4.4.1 Databázové schéma . . . 14

5 Implementace **17**
5.1 Architektura aplikace . . . 17
5.2 Implementace senzorů . . . 18
5.2.1 Dekódování dat . . . 18
5.2.2 Převod do jiných jednotek . . . 19
5.2.3 Zobrazení grafů . . . 19
5.3 Geografické zobrazení . . . 20
5.4 Zabezpečení . . . 21
5.4.1 Přihlašování . . . 21
5.4.2 Cross-site scripting . . . 23
5.4.3 SQL Injection . . . 23
5.4.4 Cross-site request forgery . . . 23
5.5 Komunikace s jednotkou . . . 24
5.6 Lokalizace . . . 24

6 Testování **26**
6.1 Testování přenosu dat . . . 26
6.2 Testování aplikace . . . 26

7 Závěr **27**
7.1 Zhodnocení . . . 27
7.2 Možnosti do budoucna . . . 27
7.3 Shrnutí . . . 28

Literatura **29**

A Seznam použitých zkratek **30**

B Obsah přiloženého CD **31**

Kapitola 1

Úvod

S rozvojem výpočetních a komunikačních technologií se objevují nové možnosti dohledu nad průmyslovými stroji. Díky technologii GPS je možné určit polohu stroje kdekoli na světě a díky GSM je možné se strojem téměř odkudkoli komunikovat. Spojení těchto technologií umožní monitorování stroje na dálku.

Mezi hlavní důvody pro monitorování stroje patří:

- přehled o provozu stroje pro zvýšení produktivity,
- kontrola využití stroje, která umožní efektivnější provádění servisu stroje,
- kontrola spotřeby paliva (pokud je namontován měřič spotřeby) umožnující lepší hospodárnost a snížení nákladů,
- zobrazení historie jízd,
- kontrola ujetých kilometrů,
- pomoc při lokalizaci zcizeného stroje,
- hlídání neoprávněného užití stroje.

Existující systémy jsou většinou úzce zaměřeny na konkrétní požadavek a neposkytují možnost dodatečného rozšíření podle speciálních potřeb. Cílem tohoto systému je obecné řešení pro nejrůznější druhy strojů, aby mohla být aplikace snadno rozšířena o další funkcionalitu bez nutnosti rozsáhlého zásahu.

Kapitola 2

Popis problému, specifikace požadavků

Cílem je vytvořit serverovou aplikaci systému pro monitorování průmyslových strojů. Z hlediska jednoduchosti přístupu pro uživatele bude vytvořena za pomoci webových technologií. Uživateli proto bude pro přístup stačit pouze internetový prohlížeč.

2.1 Hlavní kritéria

Při návrhu a implementaci je třeba se držet následujících zásad:

Důraz na bezpečnost – Minimalizovat možnosti napadení aplikace a obrana před známými útoky.

Snadné rozšíření – Při potřebě přidání nové funkcionality by mělo být potřeba zasahovat do aplikace na co nejméně místech.

Jednoduché nasazení – Zvolené technologie musí umožňovat bezproblémovou instalaci na server (webový hosting).

Nenáročné na údržbu – Po nasazení bude vyžadovat minimální úsilí na udržení systému v chodu.

Možnost jazykové mutace – Systém by měl umožňovat snadný překlad uživatelského rozhraní do cizího jazyka.

2.2 Specifikace požadavků

Aplikace bude mít dvě rozhraní, jedno pro komunikaci s jednotkou a druhé pro přístup uživatelů. Uživatelské rozhraní bude grafické a mělo by zajistit co největší komfort při práci za použití moderních webových technologií.

2.2.1 Aplikační rozhraní

Rozhraní pro komunikaci s jednotkou bude záviset na komunikačním protokolu. Monitorovací jednotka bude zřejmě postavena na vestavěné architektuře s mikroprocesory, které dosahují nižšího výkonu než procesory na serverech. Proto bude při návrhu nutné zajistit, aby byl protokol ve výpočetních možnostech jednotky.

Dále bychom chtěli oboustranný přenos, tedy že jednotka vysílá požadavky na server a server může také vysílat požadavky na jednotku. Jak se ukáže v rozboru, toto může představovat problém, na který se musí použít dodatečné řešení.

Samozřejmostí je zabezpečený přenos, aby nemohlo dojít ke zneužití přenášených dat. Bude nutné vybrat vhodný šifrovací mechanismus.

2.2.2 Role uživatelů

Před vlastním popisem uživatelského rozhraní si nejdříve popíšeme role uživatelů.

Uživatel
: Jedná se o běžného uživatele. K těmto uživatelům patří vlastník stroje nebo zaměstnanci firmy vlastnící stroj. Uživatel může prohlížet stroje a data, ke který je oprávněn, případně měnit základní nastavení strojů.

  Pokud k tomu má uživatel oprávnění, tak může vytvářet pod-uživatele na neomezené či časově omezené období.

Pod-uživatel
: Jedná se u uživatele další strany. Kupříkladu firma vlastnící stroj zprostředkovává zakázku za pomoci daného stroje. Firma může zákazníkovi po dobu realizace zakázky umožnit přístup k informacím o stroji, aby například mohl zákazník vidět, že stroj byl ve smluveném čase na místě určení.

Administrátor stroje
: Ke každému stroji je přidělen jeden administrátor. Ten může přidělit administrátorská práva pro daný stroj i jiným uživatelům. Tento přístup může poté odebrat, ale žádný jiný uživatel s administrátorskými právy nemůže odebrat práva hlavnímu administrátorovi.

Hlavní Administrátor
: Jedná se o hlavního administrátora systému. Má práva pro provádění veškerých akcí.

2.2.3 Uživatelské rozhraní

Aplikace bude podporovat následující případy užití.

Zobrazení seznamu strojů
: Zobrazí seznam strojů, ke kterým má uživatel práva k přístupu.

Zobrazení senzorů pro vybraný stroj
: Po vybrání určitého stroje zobrazí seznam senzorů.

Zobrazení dat ze senzoru

Zobrazí data k senzoru. V závislosti na typu senzoru může zobrazení vypadat různě. Například graf pro veličiny jako teplota tlak apod., či mapa pro geolokaci. Pro všechny druhy senzorů je možné si data zobrazit v tabulce nebo si je nechat vyexportovat do souboru CSV pro použití v tabulkovém procesoru (např. MS Excel nebo OpenOffice Calc).

Dále je možné vybrat zobrazení dat za určité období, například měsíční či denní výpis apod. Po zatrhnutí volby je možné automatické načítání nových dat, pro přehled co se strojem děje v aktuálním okamžiku.

Je také možné zvolit přepočítávání dat do jiných jednotek.

Administrace

Následují požadavky pro možnosti nastavení.

Změna uživatelských údajů

Uživatel si může změnit a nastavit základní údaje jako jméno, heslo a email.

Administrace pod-uživatelů

Možnost vytváření a rušení dočasných uživatelských účtů.

Administrace oprávnění

Nastavení oprávnění ke strojům. Součástí je i výběr časového období, po které je toto oprávnění přiděleno.

Administrace strojů

Možnost volby jména stroje, výchozí barvy grafů a jednotky.

Kapitola 3

Analýza a návrh komunikace

3.1 Vlastnosti spojení

S jednotkou chceme komunikovat přes internet. Kromě toho že jednotka posílá data na server, chceme také odesílat příkazy ze serveru jednotce, komunikace tudíž musí být oboustranná.

Při návrhu musíme uvažovat kromě plnohodnotného připojení, např. v rámci místní sítě (LAN) či v případě internetu s uzly s veřejnými adresami, také omezené připojení s překladem adres (NAT) či mobilní internet. V případě mobilního internetu budeme uvažovat také omezení doby připojení u vytáčeného spojení.

3.1.1 Network address translation (NAT)

Jako NAT uvažujeme situaci, kdy více zařízení v privátním rozsahu adres navenek sdílí jednu veřejnou adresu. Většina takovýchto sítí je navíc zabezpečena tak, že nelze navázat připojení z vnějšku dovnitř. Pokud ale navážeme spojení zevnitř a budeme pravidelně posílat TCP keep-alive pakety, potom zařízení provádějící překlad bude spojení udržovat a bude možné kdykoli odeslat data směrem k jednotce.

V některých případech jako např. u mobilního internetu s účtováním podle času není ovšem udržování trvalého připojení žádoucí. V tomto případě je potřeba nějak klienta vyrozumět, že se má ohlásit a server ve své odpovědi pošle potřebné příkazy. K tomuto účelu můžeme využít v případě GSM prozvonění. Abychom udrželi implementaci serveru dostatečně obecnou a nečinili ji zbytečně komplexní kvůli těmto velmi specifickým požadavkům, zavedeme v případě potřeby transparentní proxy server, který bude mechanismus notifikace zprostředkovávat.

3.1.2 Navazování komunikace

Jednotka bude navazovat spojení na server a dle potřeby odesílat nashromážděná data. V odpovědi může server odeslat data pro jednotku jako například řídící příkazy. Pokud bude potřebovat server odeslat data a připojení nebude navázané, použije se mechanismus notifikace. Poté se jednotka ohlásí a server v odpovědi odešle potřebná data. U plnohodnotného

internetového připojení bude jako notifikace prosté navázání spojení. U mobilního internetu můžeme jako notifikaci využít prozvonění za využití proxy.

3.2 Formát přenosu dat

Abychom mohli uvažovat nad návrhem protokolu, musíme se nejdříve ujasnit, jaká data je nutno přenášet.

Jednotka odesílá

- data – základem jsou data naměřená ze senzorů,
- odpověď na stav – v případě že se server dotázal na aktuální stav jednotky.
- přihlašovací údaje – pro přihlášení a autentizaci jednotky

Server odesílá

- dotaz na stav – vyžádá si informace o aktuálním stavu jednotky; dotaz může být vyvolán např. uživatelskou akcí,
- příkazy (např. čas dalšího ohlášení).

V každé zprávě musíme rozlišovat

- ID stroje – jednoznačná identifikace stroje nutná pro uložení do databáze,
- typ zprávy (např. data nebo odpověď),
- délku zprávy.

3.2.1 Data

Jednotka odesílá hodnotu všech namontovaných senzorů v daném čase.

Při určení času se spokojíme s přesností v jednotkách sekund. Můžeme využít unix timestamp [5], což je celé číslo udávající počet sekund, které uplynuly od 1.1.1970. Na většině systémů se využívá 32bitové celé číslo se znaménkem, které má kvůli omezenému rozsahu tu vlastnost, že v roce 2038 přeteče a nelze reprezentovat pozdější data. Abychom se tomuto problému vyhnuli, můžeme používat 64bitové znaménkové celé číslo, které poskytuje dostatečný rozsah.

Na každém stroji mohou být namontovány jiné senzory, proto si informace o senzorech musíme pamatovat na serveru pro každý stroj zvlášť. Pro každý typ senzoru musíme mít na serveru uložený způsob, jakým z dat získáme použitelnou hodnotu. Například zda se jedná o celé číslo či číslo s plovoucí řádkou, jaký je rozsah veličiny apod. Formát zprávy bude určen posloupností bytů dat jednotlivých senzorů.

Například pro stroj, který má 16bitový senzor otáček motoru a 32bitový senzor pro tlak v hydraulickém válci může být formát následující:

8B	2B	4B
timestamp	otáčky motoru	tlak v hydraulickém válci

Při jednom přenosu můžeme poslat více zpráv najednou pro ušetření konektivity. Protože známe délku jedné zprávy, můžeme data jednoznačně rozdělit.

3.2.2 Délka odesílacího intervalu

V případě že bude aktuálně přihlášený uživatel do systému, budeme chtít aby byla data přenášena například každých 30 sekund, aby měl uživatel aktuální přehled. V běžném provozu stroje není nutný takto krátký interval a data mohou být posílána po delším čase např. 2-3 minuty. V případě nečinnosti stroje nebo v nočních hodinách je možné interval odesílání prodloužit třeba na hodinu.

Tohoto chování docílíme tak, že server ve své odpovědi může jednotce sdělit interval příštího přihlášení.

3.3 Protokol

Vyžadujeme přístup přes internet, tudíž protokol bude postaven na IP. Komunikaci můžeme postavit buďto na TCP (Transmission Control Protocol) nebo UDP (User Datagram Protocol).

3.3.1 TCP

TCP je protokol transportní vrstvy, který mezi dvěma síťovými uzly navazuje spojení. V rámci tohoto spojení jsou posílána data a protokol zajišťuje jejich spolehlivé doručení a že dojde k doručení ve správném pořadí [9].

3.3.2 HTTP

Hypertext Transfer Protocol [2] je protokol postavený na TCP a tvoří základní prvek dnešního webu. Jeho použití je oproti TCP výhodné v tom, že je ho možné velice jednoduše integrovat do webové aplikace na serveru. Byla by tak využita stávající infrastruktura bez nutnosti implementovat obsluhu dodatečných protokolů.

Další výhodou je fakt, že HTTP jako základní služba není ve většině případů blokována firewally omezujícími datový provoz. Data takto projdou bez nutnosti zajišťovat dodatečné spojení, tunelované přes jiný protokol.

HTTP posílá řídící data v prostém textu, z čehož plyne další výhoda a zároveň nevýhoda. Výhodou je možnost jednoduchého rozšíření aplikačního protokolu bez narušení zpětné kompatibility. Nevýhodou je větší datová režie.

3.3.3 UDP

UDP je protokolem transportní vrstvy pro posílání dat mezi uzly bez předchozího navazování spojení [8]. Oproti TCP nezaručuje, že odeslaná data budou doručena a že dojdou ve správném pořadí.

Z vlastností protokolu plyne velmi nízká datová režie. Proto tento protokol bude vhodné zvolit v případě, že bude naším požadavkem co nejmenší datový provoz. Nevýhodou je nutnost dodatečné implementace mechanismů pro zaručení správného doručení dat.

3.3.4 Proxy server

V dřívější kapitole jsme uvedli důvody pro zavedení proxy serveru u omezených internetových připojení. Jestliže bude připojení konkrétní jednotky postaveno na GSM, můžeme využít k notifikaci jednotky prozvonění. V nejnutnějších případech lze také přenést kritická data pomocí SMS.

Bránu do GSM sítě můžeme realizovat připojením GSM zařízení (mobilní telefon, modem) k sériové lince serveru. Ovládání těchto zařízení probíhá pomocí AT příkazů. Sada těchto příkazů se mezi výrobci zařízení liší, proto můžeme využít knihovnu Gammu[1], která poskytuje sjednocené rozhraní pro ovládání mobilních telefonů různých výrobců.

Dále by bylo technicky možné též použít softwarové brány mobilních operátorů či programové rozhraní VOIP operátorů. Průzkum trhu a možných řešení je ovšem nad rozsah práce a proto se jím nebudeme dále zabývat.

Proxy server také můžeme využít pokud budeme chtít použít pro přenos jiný protokol než HTTP. To bude výhodné v případě že zvolíme jako protokol HTTP z výše uvedených důvodů, ale později se ukáže, že datové přenosy jsou příliš vysoké. Bude tedy nutné přidat podporu pro např. pro UDP. Využitím proxy serveru bude nutné pouze implementovat překlad mezi protokoly, ale ušetříme úsilí, protože se nebudeme muset zabývat reimplementací aplikační logiky.

3.3.5 Komprese

Pokud to bude ve výpočetních schopnostech jednotky a budeme chtít minimalizovat množství přenesených dat, můžeme volitelně využít kompresi. Přenášená data můžeme zapouzdřit na nižších vrstvách, tudíž volba kompresního algoritmu návrh protokolu neovlivní.

Výběr kompresního algoritmu bude záviset zejména na možnostech implementace na monitorovací jednotce a může se v konkrétních případech nasazení velmi lišit.

3.3.6 Šifrování

Přenášená data chceme zabezpečit šifrováním. Abychom snížili množství šifrovaných dat použitelných pro kryptografický útok a tudíž snížili riziko možného prolomení, je vhodné šifrovací klíče používat po omezenou dobu a poté vygenerovat nové [6]. Typicky se za tuto krátkou dobu považuje spojení, po jehož ukončení dojde k zapomenutí veškerých možných stop vedoucích k prolomení.

[1] domovská stránka na http://wammu.eu/libgammu/

SSL / TLS

Secure Sockets Layer a jeho nástupce Transport Layer Security jsou kryptografické protokoly, které poskytují možnost bezpečné komunikace přes internet [4]. SSL/TLS se běžně používá pro zabezpečení webových služeb, proto se v případě HTTP využití přímo nabízí. TLS má při vyjednávání spojení možnost použít režim kompatibilní s SSL, proto budu dále v textu používat pouze označení TLS.

TLS při navazování spojení využívá mechanismu podepsaných certifikátů pro ověření identity (autentizace). K tomu se nejčastěji používá protokol RSA. Poté následuje vyjednávání šifrovacích protokolů a klíčů. Při samotné komunikaci se již využívá symetrické šifry jako například RC4 či AES.

TLS je poměrně komplexní balík a protokol RSA je výpočetně velmi náročný. Není vyloučeno, že tato náročnost bude mimo výpočetní možnosti monitorovací jednotky.

OTR

Alternativní možností je Off-the-Record Messaging. OTR je protokol, který zaručuje zabezpečený přenos a autentizaci [3]. Pro výměnu klíčů využívá Diffie-Hellmanův algoritmus, který je obohacený o autentizační mechanismus. Vlastní přenos je poté šifrován pomocí šifry AES. Pro snížení rizika napadení je potřeba vyměňovat klíče co nejčastěji, nejlépe po každé zprávě. Naštěstí je Diffie-Hellmanův výpočet poměrně nenáročný, takže dle [3] by i zařízení s omezenou výpočetní silou měla být schopná přepočítávat klíče dostatečně často. Proto můžeme OTR využít jako alternativu v případě že jednotka nebude zvládat nasazení TLS.

3.4 Shrnutí

Probrali jsme možnosti komunikačního protokolu pro zadaný systém. Pro volbu protokolu máme dvě hlavní možnosti: využít HTTP nebo protokol postavit nad UDP.

Pro HTTP hovoří rozšířenost, dostupné hotové knihovny a jednoduchost integrace do webové aplikace, což ušetří značné úsilí při implementaci. Pokud se spokojíme s vyšší datovou režií, pak je HTTP doporučenou volbou.

Pokud by režie HTTP byla v reálných podmínkách příliš vysoká, jako základ protokolu by se dalo využít UDP. Ovšem v tomto případě by bylo potřeba navíc implementovat základní vlastnosti jako mechanismus zaručení přenesených dat a integrace do serverové aplikace by byla obtížnější.

Zároveň jsme popsali řešení problémů při omezeném internetovém připojení a možnosti zabezpečeného šifrovaného přenosu dat.

3.5 Ukázka komunikace

Zde uvádím příklad, jak by mohla vypadat komunikace postavené na HTTP. Dodatečné HTTP hlavičky byly pro jednoduchost vypuštěny.

V ukázce č. 1 posílá jednotka, jejíž identifikační číslo je 123, binární data ze senzorů o délce 100 bytů.

```
POST /?id=123&type=data HTTP/1.0
Content-Type: application/octet-stream
Content-Length: 100

... binární data ...
```

Ukázka 1: Jednotka → Server

```
HTTP/1.0 200 OK
Content-Type: application/x-www-form-urlencoded
Content-Length: 11

interval=60
```

Ukázka 2: Server → Jednotka

V ukázce č. 2 server odpovídá na předchozí požadavek. Status 200 OK značí úspěšné uložení dat na serveru. Server navíc jednotce oznamuje, že se má znovu ohlásit za 60 sekund.

Kapitola 4

Analýza a návrh aplikace

4.1 Architektura aplikace

Aplikaci chceme postavit na čistém návrhu, kdy bude aplikační logika oddělená od zobrazovací. Toho dosáhneme použitím architektury na bázi Model-View-Controller (MVC). Pro implementaci je vhodné využít objektově orientované programování, které dovolí funkcionalitu správně zapouzdřit. Výběr technologií je tedy tomuto požadavku nutno přizpůsobit.

4.2 Použité technologie

4.2.1 Programovací jazyk

Při výběru programovacího jazyka musíme přihlédnout k požadavku, že má jít o webovou aplikaci, a vybrat takový programovací jazyk, který nabízí dobrou podporu pro vývoj webových aplikací. Jelikož budu aplikaci sám implementovat, musím výběr omezit na programovací jazyky, se kterými jsem se již alespoň někdy setkal. Takovými kandidáty jsou: C++, Java, Python a PHP.

C++

Výhodou je velká rychlost výsledné aplikace. Nevýhodou je zejména delší čas potřebný na vývoj aplikace. Toto řešení není ve světě webových aplikací příliš rozšířené.

Java

Java nabízí robustní prostředí pro vývoj aplikací včetně objektového programování a jednoduchého využití MVC. Výhodou je velká stabilita aplikací. Nevýhodou jsou větší systémové nároky a nutnost běhu Java serveru, což znesnadňuje nasazení na běžná hostingová řešení.

Python

Python je objektový interpretovaný jazyk. Možnosti jazyka jsou velké, bohužel v něm nemám zkušenosti s vývojem webových aplikací a proto si nemůžu dovolit v něm tuto aplikaci vyvíjet.

PHP

Jedná se o nejrozšířenější řešení s dlouhou tradicí. PHP od verze 5 podporuje objektové orientované programování a bude tedy možné využít vzor MVC. Výhodou je snadné nasazení aplikace a velké množství dodatečných knihoven. PHP splňuje všechny požadavky a také mám s touto technologií nejvíce zkušeností.

Pro implementaci jsem zvolil PHP verze 5.2 a vyšší. Vývoj probíhal na verzi 5.2.10.

4.2.2 Knihovny a frameworky

Nette Framework

Abychom urychlili a usnadnili vývoj, tak použijeme soubor podpůrných knihoven, takzvaný framework. Existuje nepřeberné množství nejrůznějších frameworků a jejich porovnání by vydalo na celou práci. Já jsem zvolil Nette Framework[1], neboť s ním mám pozitivní zkušenosti. Pokusím se stručně shrnout nejdůležitější přednosti:

- Vysoká bezpečnost – používá techniky, které eliminují výskyt bezpečnostních děr. Podrobněji budou tyto techniky zmíněny při popisu implementace v kapitole 5.
- Velký výkon – podle nezávislého testu[2] dosahuje v porovnání s ostatními frameworky velkého výkonu.
- Čistý objektový návrh – využívá nových vlastností PHP 5 a vede k čistému návrhu aplikace.
- Open-source licence – je k dispozici zdrojový kód, framework je zdarma a to i pro komerční využití.

Toto jsou jen některé z výhod, které považuji za klíčové. Další výhody je možné nalézt na http://nette.org/cs/hlavni-prednosti.

dibi

Pro zjednodušení práce s databází používám knihovnu dibi[3]. Knihovna pochází od stejného autora jako Nette, takže zde existuje určitá provázanost. Hlavním přínosem je usnadnění psaní SQL dotazů a zvýšení bezpečnosti.

[1] domovská stránka na http://nette.org/
[2] http://www.root.cz/clanky/velky-test-php-frameworku-zend-nette-php-a-ror/
[3] domovská stránka na http://dibiphp.com/

jQuery

Pro zlepšení uživatelského komfortu se na straně klienta využívá Javascript. jQuery[4] je javascriptová knihovna, která sjednocuje funkcionalitu existujících webových prohlížečů, které se v implementaci standardu více či méně odlišují. Díky tomu je urychlen a zjednodušen vývoj aplikace, neboť není nutné tyto odlišnosti složitě obcházet a ladit.

4.3 Uložení dat

Pro uložení dat zvolíme databázi z důvodu udržování konzistence a jednoduchého způsobu manipulování s daty.

Jako databázi zvolíme MySQL, neboť plně dostačuje našim požadavkům, má dostupnou finanční politiku (zdarma) a jedná se o rozšířené snadno nasaditelné řešení.

Aplikace byla vyvíjena a testována na verzi 5.1.41, ale aplikace by měla být funkční na jakékoli verzi 5. řady.

4.3.1 Data ze strojů

Zvláštní pozornost si zaslouží data ze strojů. Těch bude velké množství a možná by se dosáhlo velkého urychlení při výpisu, pokud by byla binárně uložená v obyčejných souborech na disku.

Toto uložení by ovšem bylo náročnější na implementaci. Museli bychom zvolit vhodné rozdělení do souborů podle období (po dnech, týdnech či měsících) a dopsat funkce pro výběr zvolených dat. Komplikace by mohly nastat, pokud by data nepřicházela z jednotky popořadě, což se v případě výpadku signálu ale může stát.

Proto zvolíme v zájmu méně náročné implementace taktéž uložení do databáze, která za nás předchozí problémy řeší. Pokud se při testovacím provozu ukáže, že tento způsob nemá dostatečný výkon, tak teprve potom bude vhodné uvažovat o dodatečných optimalizacích. Například by šla zavést cache nebo ukládat data do binárních souborů.

4.4 Datový model

Ze zadání nám vyplynou tři hlavní entity aplikace, jsou to: Uživatel, Stroj a Senzor.

Uživatel

O uživateli potřebujeme evidovat přístupové údaje, jméno, roli a oprávnění ke strojům.

[4] domovská stránka na http://jquery.com/

Stroj

Potřebujeme evidovat název, typ, přístupové heslo a s ním spjaté senzory. Model stroje se bude starat o uložení dat. Předpokládáme, že budeme mít více strojů podobného typu, které budou mít namontované stejné senzory. Tento předpoklad se projeví při návrhu databázového modelu.

Je třeba ošetřit platnost na určité období, když se např. po určité době přidá dodatečný senzor, tak se změní formát, ale stará data nesmí být zneplatněna.

Senzor

Senzor zprostředkovává prezentaci dat uživatelům. Jeho funkce je převzít binární data od modelu stroje a prezentovat v uživatelsky čitelné formě. To znamená přepočítat binární data do smysluplných jednotek, vypsat údaje do tabulky, grafu nebo mapy, atd. To jak budou data prezentována záleží na typu senzoru.

V aplikaci mohou být základní typy senzorů:

- analogové veličiny - např. teplota, otáčky motoru
- boolean – např. stav stroje - vypnuto/zapnuto
- GPS souřadnice - vizualizace trasy na mapě

Pokud budeme chtít přidat senzor s odlišným způsobem vizualizace, tak bude pouze potřeba napsat funkce pro vizualizaci, protože funkce pro získávání dat jsou společné. Toto flexibilní řešení umožňuje velmi jednoduchou rozšiřitelnost podle budoucích potřeb, které nejsou přesně známy v době psaní aplikace.

4.4.1 Databázové schéma

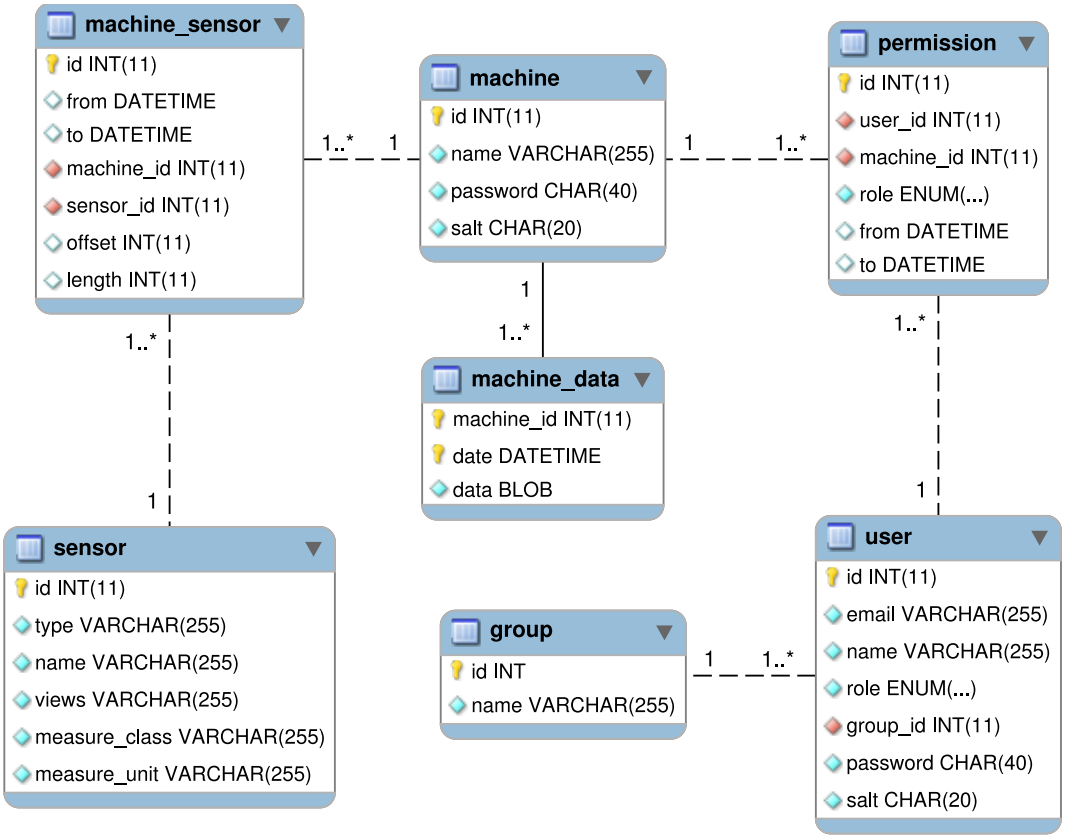
Výše popsaný model převedeme do relační databáze a dostaneme následující schéma v UML notaci, viz obrázek 4.1.

Schéma obsahuje tři hlavní tabulky – `user`, `sensor` a `machine`. Ty odpovídají třem hlavním entitám.

Tabulka pro uživatele obsahuje základních údaje jako jméno, email a heslo (ukládán je pouze jeho otisk za pomoci hashovací funkce). Dalším sloupcem navíc je `salt`, jehož potřeba je vysvětlena v kapitole 5 v sekci zabývající se zabezpečením.

Tabulka obsahuje také atribut `role`, který nabývá následujících hodnot:

- `user` – běžný uživatel
- `subuser` – pod-uživatel, který byl vytvořen běžným uživatelem
- `serveradmin` – správce s neomezenými pravomocemi



Obrázek 4.1: Databázové schéma

Tabulka `group` slouží pro reprezentaci skupiny uživatelů. Podle příslušnosti ke skupině budeme vyhledávat pod-uživatele. V praxi může být skupina chápána jako firma, ke které patří daní uživatelé. Tabulka obsahuje jen jeden sloupec pro jméno, ale může být v budoucnu rozšířena potřebné údaje.

K určení relace mezi tabulkou `user` a `machine` je zde tabulka `permission`. Ta kromě cizích klíčů obsahuje časy od kdy do kdy toto oprávnění platí a roli oprávnění. Role oprávnění nabývá následujících hodnot:

- `machineadmin` – hlavní administrátor stroje
- `admin` – administrátorská práva
- `view` – práva pro prohlížení

Tabulka `machine` obsahuje základní údaje stroje a není potřeba ji dále rozvádět.

V tabulce `sensor` si kromě jména udržujeme typ, který je reprezentován jako řetězec, aby bylo možné v budoucnu přidávat nové typy senzorů bez nutnosti měnit databázové schéma. Typ vyjadřuje například celé číslo (int) nebo číslo s plovoucí řádkou (float). Sloupec `measure_class` představuje o jakou se jedná veličinu (např. rychlost, vzdálenost)

a `measure_unit` v jakých jednotkách je uložena. Posledním sloupcem je `views`, ve kterém je uložena informace, jaké druhy pohledů senzor podporuje.

Relaci mezi strojem a senzorem zajišťuje tabulka `machine_sensor`, která obsahuje čas od kdy a do kdy je senzor na stroji namontován. Pro určení jaká data senzoru patří slouží hodnota pozice dat `offset` a délka v bytech `length`.

Poslední tabulkou je `machine_data`, která slouží pro ukládání dat z jednotky. Obsahuje sloupec kdy byla data naměřena a jejich hodnotu.

Kapitola 5

Implementace

5.1 Architektura aplikace

Aplikaci chceme pojmout podle architektury Model-View-Controller (MVC). Architektura MVC dělí aplikaci na 3 logické části tak, aby je šlo upravovat samostatně a dopad změn byl na ostatní části co nejmenší. Tyto tři části jsou Model, View a Controller. Model reprezentuje data a business logiku aplikace, View zobrazuje uživatelské rozhraní a Controller má na starosti tok událostí v aplikaci a obecně aplikační logiku [1].

MVC lze brát i pouze jako přístup k návrhu a konkrétní implementace se může lišit. Jednou z těchto variací je Model-View-Presenter (MVP) [10]. MVP formalizuje a ještě důsledněji odděluje reprezentaci dat (Model) a uživatelské rozhraní (View-Controller). Kombinace View-Controller je nazývána prezentací. Uživatel poté ovládá aplikaci pomocí Presenteru, který ovládá Model a předává jeho data do View.

Logice Nette Frameworku daleko lépe odpovídá MVP [13]. View je potom zajištěn šablonovacím systémem, který poskládá výslednou HTML stránku. Presenter zpracovává HTTP požadavky, manipuluje s Modelem a předává data do šablonovacího systému. Model je pak zjednodušeně vrstva zapouzdřující data, která jsou nejčastěji uložena v databázi.

Dále budou popsány základní presentery, ze kterých se aplikace skládá.

HomepagePresenter
: Tento presenter obsluhuje úvodní webové stránky a slouží k seznámení návštěvníka se systémem a jeho přínosy. Dále obsahuje odkaz na přihlášení do aplikace.

AuthPresenter
: Zde je obsluhováno přihlašování a odhlašování uživatele z aplikace.

MachinesPresenter
: Slouží k zobrazení seznamu strojů. Po výběru stroje ze seznamu zobrazí podrobnější informace o daném stroji a jeho senzory.

SensorPresenter
: Zobrazuje data ze senzoru. Uživatel má možnost zvolit si různé druhy vizualizace, například tabulku, grafy apod.

AdministrationPresenter
Tento presenter slouží k nastavení, změna hesla, údaje o uživatele, správa pod-uživatelů a strojů.

UnitPresenter
Obsluhuje komunikaci s monitorovací jednotkou a stará se o ukládání dat.

Následující podpůrné presentery nesouvisí přímo s aplikační logikou, přesto plní důležitou roli a zaslouží si zmínku.

ErrorPresenter
Pokud v aplikaci nastane chyba, tak se vyhodí výjimka. Není žádoucí obtěžovat uživatele s přesným číslem chyby a dalšími matoucími podrobnostmi. Aplikace nejdříve chybu a informace pro ladění zapíše do logovacího souboru. Poté dojde k přesměrování na `ErrorPresenter`, který uživatelsky přívětivou formou oznámí, že došlo k chybě.

BasePresenter
Tento presenter je předkem pro všechny výše uvedené presentery a obstarává funkce, které jsou pro ně společné. To je výhodné, neboť kód je na jednom místě a nedochází k jeho duplikaci. Před předáním řízení potomkům dochází například ke připojení k databázi.

Dále je zde ověřováno, zda je uživatel přihlášen. Pokud ne, tak dojde k přesměrování na `AuthPresenter`. Po úspěšném ověření uživatele dojde k přesměrování zpět na původní stránku.

5.2 Implementace senzorů

5.2.1 Dekódování dat

Abychom zachovali obecný přístup tak zavedeme rozhraní `ISensorDecoder`, které zavádí jednu metodu `decode`. Pro různé druhy senzorů definujeme třídy, které budou toto rozhraní definovat, například `SensorDecoderInt` pro celá čísla a `SensorDecoderInt` pro čísla s plovoucí desetinou řádkou.

```
interface ISensorDecoder {
    function decodeData($data);
}
```

Ukázka 3: Rozhraní ISensorDecoder

Třída `SensorModel` reprezentující senzor se podle typu uloženého v databázi postará o zvolení odpovídajícího dekodéru.

5.2.2 Převod do jiných jednotek

Pro převod jednotek jsem se rozhodl využít již hotové řešení `Zend_Measure`, které je součástí Zend Frameworku[1]. `Zend_Measure` je soubor tříd, které poskytují robustní řešení a podporují mnoho veličin[2]. Použití je velmi jednoduché, viz ukázka níže.

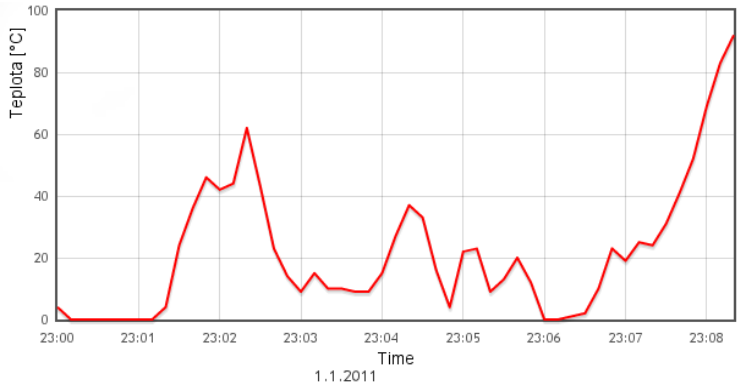
```
// vytvoř novou veličinu s hodnotou 100 kilometrů
$value = new Zend_Measure_Length(100, Zend_Measure_Length::KILOMETER);

// převeď na míle
$value->convertTo(Zend_Measure_Length::MILE);

// vypiš se zaokrouhlením na 2 desetinná místa
echo $value->getValue(2);

// vypíše 62.14
```

Ukázka 4: Použití Zend_Measure pro převod hodnot do jiných jednotek

Třída `SensorModel` se postará, že se k převodu použije správná třída.

5.2.3 Zobrazení grafů

Zobrazení grafů jsem se rozhodl řešit na straně klienta ve webovém prohlížeči. Výhodou tohoto řešení je vysoký uživatelský komfort, neboť se při změně vybraného období nemusí znovu načítat celá stránka, ale pouze se na pozadí přenesou potřebná data a graf se překreslí.



Obrázek 5.1: Ukázka zobrazení grafu pomocí knihovny flot

[1] domovská stránka na http://framework.zend.com/
[2] seznam veličin je na http://framework.zend.com/manual/en/zend.measure.types.html

K vykreslování grafu je použita knihovna `flot`[3] napsaná v javascriptu. Pro načítání dat se používají AJAX (Asynchronous JavaScript and XML) požadavky.

5.3 Geografické zobrazení

V době tvorby této práce nebylo známo jaké veličiny bude mít jednotka k dispozici a jaká bude jejich přesnost. Pro referenční implementaci jsem zvolil dva základní údaje: zeměpisnou délku (angl. longtitude) a zeměpisnou šířku (angl. latitude). Hodnoty budou uvedeny ve stupních a reprezentovány 64bitovým číslem s plovoucí řádovou čárkou dle normy IEEE 754 (tento typ je ve většině programovacích jazyků známý jako `double`);

Pro zobrazení mapových dat budu používat open source knihovnu OpenLayers[4]. Její výhodou kromě otevřené licence je také podpora různých mapových poskytovatelů a formátů v jednotném API. Dále velmi dobře podporuje vykreslování vektorových útvarů, které je využito pro zobrazení trasy.

Jako zdroj mapových podkladů poslouží OpenStreetMap[5]. OSM je otevřená editovatelná mapa celého světa šířená pod svobodnou licencí CC-BY-SA [6]

Obrázek 5.2: Ukázka zobrazení mapy pomocí OpenLayers s datovým podkladem OpenStreetMap

[3] http://code.google.com/p/flot/
[4] domovská stránka na http://openlayers.org/
[5] domovská stránka na http://www.openstreetmap.org/
[6] Creative Commons Attribution-ShareAlike 2.0 http://creativecommons.org/licenses/by-sa/2.0/

5.4 Zabezpečení

Při implementaci byl kladen důraz na bezpečnost aplikace. V této sekci popíšeme nejznámější bezpečnostní rizika a jak se proti nim bránit.

5.4.1 Přihlašování

Pro přístup do aplikace používáme ověřování pomocí hesla. Pro každého uživatele musíme mít v databázi uloženo jeho uživatelské jméno a heslo. Pro uživatele, který se přihlásí pod daným jménem, ověřujeme, zda se zadané heslo shoduje. Pokud ano, tak předpokládáme, že uživatel je ten za koho se vydává a umožníme přistup do aplikace.

Uživatelské jméno musí být samozřejmě pro každého uživatele unikátní. Aby si uživatel nemusel pamatovat další údaj navíc, tak se jako uživatelské jméno používá jeho e-mailová adresa. To také zabrání zbytečnému úsilí, kdy uživatel musí opakovaně zkoušet volit různá jména tak, aby nebyla zabrána již zaregistrovanými uživateli.

Ukládání hesel

Nejjednodušší způsob jak ukládat hesla je v jejich původní podobě. Tento způsob má ovšem několik nevýhod. Hesla jsou přístupná správcům serveru, na kterém aplikace běží. Zálohování databáze představuje další bezpečnostní riziko, neboť hesla jsou v zálohách přístupná v původní podobě. V neposlední řadě pokud dojde ke kompromitování databáze, tak se útočník dozví hesla uživatelů, což může mít dalekosáhlé následky.

Raději než ukládat hesla v původní podobě ukládáme hodnotu jednosměrné funkce, které je předáno původní heslo. Postup ověřování (schéma na obrázku 5.3) potom probíhá tak, že aplikace vypočítá hodnotu jednosměrné funkce pro zadané heslo, a tuto hodnotu poté porovná s uloženým záznamem. Potenciální útočník nedokáže ze znalosti hodnoty funkce zjistit původní heslo.

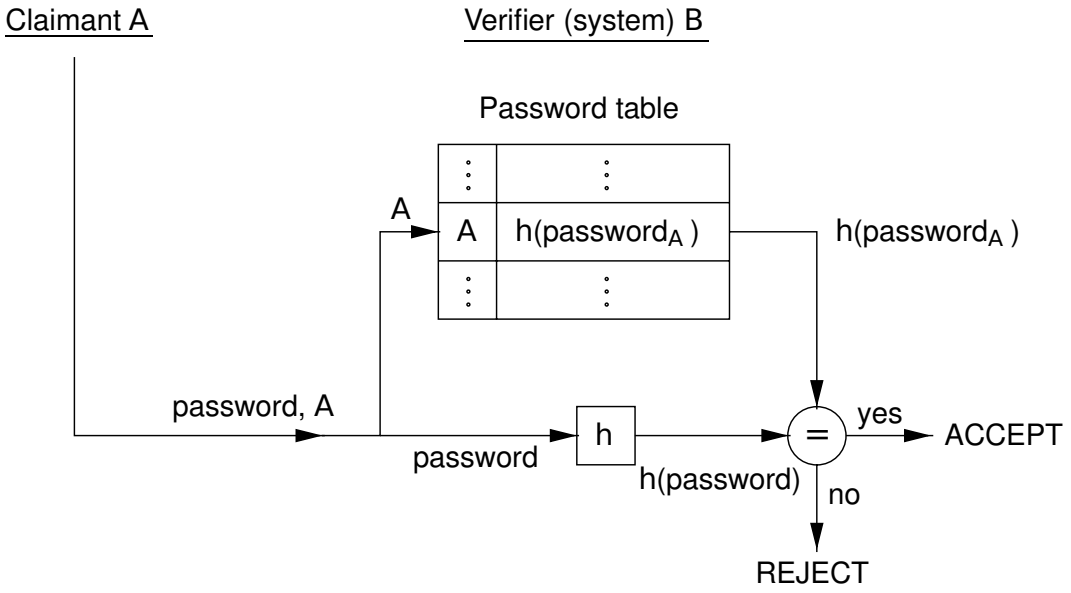
V praxi se pro tyto účely využívají hashovací funkce, pro které je velmi obtížné získat původní hodnotu. V aplikaci je použita funkce SHA-1, která má výstup o délce 160 bitů. Pro uložení se používá textová reprezentace v hexadecimální soustavě, tudíž je délka ukládaného řetězce 40 bytů.

Útok hrubou silou

Velmi naivní útok spočívá v postupném zkoušení různých hesel (volených náhodně nebo systematicky) v naději, že se podaří najít správné heslo. Těmto útokům se dá bránit dvěma hlavními způsoby:

1. Omezení počtu neúspěšných pokusů za určitý čas

2. Zpomalení procesu ověřování hesla



Obrázek 5.3: Použití jednosměrné funkce pro ověření hesla, zdroj [6]

Slovníkový útok

Dle [6] většina uživatelů volí hesla pouze z malé podmnožiny všech možných hesel. Do této podmnožiny patří například hesla s malým počtem znaků, slova přirozeného jazyka, názvy (osob, zvířat, atd.) a řetězce obsahující pouze malá písmena. Tato hesla s nízkou entropií se dají celkem snadno uhodnout. Studie ukazují, že velké procento uživateli zvolených hesel lze typicky nalézt ve středně velkém slovníku o 150 tisíci slovech.

Pro zvýšení očekávané pravděpodobnosti úspěchu hrubého prohledávání zkouší útočník všechna slova z takového slovníku. Tato technika se nazývá *slovníkový útok*.

Jelikož jsou slovníkové útoky účinné proti slabým heslům, lze zavést omezení, která uživatele povedou k používání silnějších hesel. Například určit nejmenší možnou délku nebo povinnost obsáhnout minimálně jeden znak z každé skupiny (velká písmena, čísla, interpunkční znaky). Na druhou stranu je potřeba mít na paměti, že se může stát, že u přísnějších pravidel si uživatelé nebudou hesla schopni pamatovat. Pokud si je napíšou na někam na lísteček, tak bezpečnost trpí ještě více.

Domnívám se, že rozumný kompromis je heslo minimálně 6 znaků dlouhé a kromě písmen musí obsahovat alespoň jednu číslici a jedno velké písmeno.

Salting

Abychom zmírnili účinnost slovníkových útoků tak ke každému heslu přidáme náhodný řetězec a teprve poté aplikujeme jednosměrnou funkci. Tento řetězec se nazývá *salt* a ukládá se do databáze ke každému záznamu. Salting nezvyšuje náročnost pro prolomení jednoho hesla, ale zvyšuje náročnost pro prolamování více hesel najednou, neboť pro každé heslo je potřeba přepočítat celý slovník.

Za pozornost stojí, že uživatelé se stejným heslem budou mít v uložen databázi rozdílný hash.

5.4.2 Cross-site scripting

Cross-site scripting (XSS) [7] je útok na webové aplikace, při kterém má útočník možnost vložit do stránky škodlivý obsah, zejména javascriptový kód. Obvykle jsou tyto útoky využívány k odcizení cookies, které často obsahují citlivá data, jako například přístupové údaje. Základní obrana proti tomuto útoku spočívá v ošetření všech uživatelských vstupů před jejich výpisem na stránky.

Obrana je za pomoci Nette Frameworku velmi snadná, neboť jeho součástí je šablonovací systém, který automaticky výstup *escapuje*[7] v závislosti na kontextu [11].

5.4.3 SQL Injection

SQL Injection je útok, při kterém má útočník možnost spustit libovolný SQL dotaz na databázi. Dochází k němu tehdy, pokud není při skládání dotazu ošetřen uživatelský vstup. Stejně jako u XSS je obrana velmi snadná, neboť použitá databázová vrstva dibi automaticky vstup escapuje.

5.4.4 Cross-site request forgery

Podstata Cross-site request forgery (CSRF) útoku spočívá v tom, že uživatele přimějeme navštívit stránku aplikace, která provádí nějakou akci, aniž by o tom uživatel věděl [12]. Pokud je uživatel v dané chvíli do aplikace přihlášen nebo využívá funkci trvalého přihlášení, tak aplikace provede útočníkem stanovenou akci. Útok je účinný proti aplikacím, které uchovávají informace o uživatelově přihlášení pomocí cookies.

Navštívení aplikace bez vědomí uživatele se dá snadno zařídit. Použije se například HTML značka `img`, která slouží pro vkládání obrázků. Do parametru `src` vložíme URL stránky, kterou chceme, aby uživatel nevědomě navštívil. Prohlížeč odešle požadavek na dané URL a spolu s ním odešle i autorizační cookie. Obrázek se nepodaří načíst a prohlížeč místo něj zobrazí ikonu pro nefunkční obrázek. Pokud je obrázek vhodně skryt, tak uživatel nic nepozná.

Nebezpečnost tohoto útoku spočívá v tom, že aplikace nemá možnost rozlišit, který požadavek je od skutečného uživatele a který je jen nastražen. Obranou proti této formě útoku je provádět akce měnící citlivá data metodou POST, která slouží k odesílání formulářů. Ovšem i požadavek POST lze podvrhnout, například použitím JavaScriptu.

Proto je nutné formuláře ještě dále zabezpečit. Jedním ze způsobů je použít *autorizační token*. Princip je takový, že před provedením každé operace vygenerujeme náhodný řetězec a při jejím provedení tento řetězec zkontrolujeme. Token chránící před CSRF útokem má

[7] Escapování je zrušení významu speciálních nebo řídících znaků. Například ve většině jazyků se zapisuje textový řetězec uzavřený do dvojitých uvozovek. Pokud bychom chtěli mít uvozovku jako součást textu, tak musíme určit, že uvozovka nemá být interpretována jako ukončení řetězce, ale jako prostý znak. To se většinou dělá pomocí zpětného lomítka \. Například takto: *"dnes je \"pěkný\" den"*. Při vypsání se zobrazí: *dnes je "pěkný" den.*

platnost po dobu existence session. Díky tomu nebrání použití ve více oknech najednou (v rámci jedné session) [14]. Tato obrana je implementována v Nette Frameworku. Její použití je velmi snadné a je demonstrováno na ukázce 5.

```
$form = new AppForm;

...
  nastavení prvků formuláře
...

$form->addProtection();
```

Ukázka 5: Způsob zabezpečení formuláře proti CSRF

5.5 Komunikace s jednotkou

V době implementace serverové aplikace bohužel ještě neprobíhal vývoj jednotky, jak bylo původně plánováno. V kapitole 3 byly dopodrobna probrány možnosti komunikace. Zároveň bylo poznamenáno, že konkrétní podoba protokolu je závislá na výpočetních možnostech jednotky a dalších technických omezeních.

Proto jsem implementoval pouze základní metody pro ukládání dat, které budou použity pro testování a naplnění databáze testovacími daty. O obsloužení se stará `UnitPresenter`.

To znamená, že jsem neimplementoval pokročilé mechanismy (kupříkladu udržování spojení nebo notifikace), které byly rozebrány v návrhu. Ovšem aplikace je navržena tak, že tuto funkcionalitu bude možné implementovat s minimálním úsilím, až budou známy podrobnosti o jednotce. K autentizaci jednotky je prozatím použita HTTP metoda Basic access authentication. Její výhodou je snadná implementace, nevýhodou že se uživatelské údaje přenáší při každém spojení.

5.6 Lokalizace

Aplikace podporuje jazykové mutace. K tomu používá doplněk NetteTranslator[8]. Jeho použití a způsob ukládání je obdobné jako pro nástroj `gettext`[9], který je nepsaným standardem pro lokalizaci unixových aplikací a je zabudován do PHP. Využití NetteTranslatoru oproti nativní implementaci má následující výhody:

- V nativní implementaci jsou překlady udržovány v cache a pokud je aktualizován soubor s překlady, je potřeba restartovat celý web server.

[8] popis na http://forum.nette.org/cs/4758-nettetranslator-gettexttranslator-nette-translation-panel
[9] domovská stránka na http://www.gnu.org/software/gettext/

- NetteTranslator obsahuje panel, ze kterého je možné aplikaci překládat přímo z prohlížeče, což výrazně celý proces překladu ulehčuje.

Přidání podpory pro nový jazyk je velice jednoduché, spočívá pouze v přeložení textových hlášek do daného jazyka.

Kapitola 6

Testování

6.1 Testování přenosu dat

Před nasazením do provozu je nutné řádně otestovat správnou funkčnost protokolu. Základní funkčnost budeme schopni otestovat i v případě, že nebudeme mít k dispozici hardwarovou monitorovací jednotku. Přenos můžeme otestovat odesíláním testovacích dat v lokální počítačové síti. V případě HTTP to bude velmi jednoduché, neboť můžeme využít již existujících programů pro odesílání testovacích dat.

Pro test jsem napsal krátký skript v jazyce Python, který odesílá náhodná data a zjednodušeně simuluje přenos mezi jednotkou a serverem. Takto získaná data poté slouží pro demonstraci možností zobrazení v aplikaci.

6.2 Testování aplikace

Funkčnost aplikace byla testována a funkčnost ověřena v těchto prohlížečích:

- Chrome 8 a 9
- Firefox 3.6
- Opera 11
- Internet Explorer 7

Prohlížeč Internet Explorer 6 není plně podporován.

Aplikace prošla testem validity podle specifikace HTML5.

Kapitola 7

Závěr

7.1 Zhodnocení

Hlavním výstupem práce kromě tohoto textu je funkční aplikace. Implementována byla pouze obecná základní funkčnost. Ovšem aplikace je navržena tak, aby chybějící funkce bylo možné v budoucnu přidat bez většího úsilí a přizpůsobit potřebám konkrétních zákazníků.

Implementace pro mě byla osobním přínosem, neboť jsem si v praxi vyzkoušel vývoj za pomoci zvolených technologií (PHP, HTML, Nette Framework, Javascript). Při implementaci jsem měl možnost zmíněné technologie poznat do hloubky a získal jsem mnoho nových vědomostí a zkušeností.

Při reálném nasazení se může při velkém množství dat ukázat jako úzké hrdlo ukládání dat. Pokud se za rok pro jeden stroj nasbírají data o velikosti v řádech stovek megabajtů, tak samozřejmě nechceme například při ročním výpisu přenášet všechna data. Řešením by bylo zavést cache, případně ještě přepočítávat data pro daná období (ročně, měsíčně, týdně, atd.) již při ukládání.

7.2 Možnosti do budoucna

Před nasazením bude potřeba dořešit otázku komunikačního protokolu, která je ovšem velmi podrobně rozepsána a řešení bude již přímočaré. Toto řešení bude záviset na výpočetních možnostech hardwarové jednotky a možnostech komunikačního kanálu.

V aplikaci není implementována pokročilá administrace, neboť je závislá na konkrétní případy užití. Povaha webových aplikací je taková, že se jedná o živý organismus, který se neustále vyvíjí a přizpůsobuje aktuálním potřebám. Proto se nejedná o neměnný produkt a existuje mnoho možností pro rozšíření. Pro začátek mě napadá implementace mechanismu pro upozornění na změnu kritických veličin či výpis knihy jízd, který bude odpovídat právním požadavkům. Zde se zase jedná o funkcionalitu závislou na konkrétním právním prostředí.

7.3 Shrnutí

Nyní si shrneme požadovaná kritéria.

- Důraz na bezpečnost – součástí práce je podrobný popis možných způsobů napadení aplikace a implementace
- Snadné rozšíření – tento požadavek provází celou práci a kroky k jeho dosažení jsou popsané na mnoha místech
- Jednoduché nasazení a Nenáročnost na údržbu – aplikace vyžaduje webový hosting s podporou PHP 5.2 a vyšší, mod_rewrite a databáze MySQL. V ČR tyto podmínky splňuje drtivá většina poskytovatelů.
- Možnost jazykové mutace – rozhraní aplikace je pro demonstraci k dispozici v českém a anglickém jazyce. Přidání dalších překladů nepředstavuje žádný problém.

Tato práce vyhovuje požadovaným kritériím a splňuje zadání.

Literatura

[1] B. Bernard. Úvod do architektury MVC, publikováno 7. 5. 2009. http://zdrojak.root.cz/clanky/uvod-do-architektury-mvc/.

[2] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational), May 1996.

[3] Borisov, N., Goldberg, I. and E. Brewer. Off-the-record communication, or, why not to use pgp, 2004.

[4] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFC 5746.

[5] IEEE. 1003.1 posix time specification, 1988.

[6] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[7] R. Miller. PHP Security Guide, stav z 25. 12. 2009. http://php.robm.me.uk/.

[8] J. Postel. User Datagram Protocol. RFC 768 (Standard), Aug. 1980.

[9] J. Postel. Transmission Control Protocol. RFC 793 (Standard), Sept. 1981. Updated by RFCs 1122, 3168.

[10] M. Potel. MVP: Model-View-Presenter. The Taligent Programming Model for C++ and Java, 1996.

[11] J. Vrána. Context-aware HTML escaping, Month of PHP Security, publikováno 5. 5. 2010. http://php-security.org/2010/05/05/mops-submission-02-context-aware-html-escaping/.

[12] J. Vrána. Cross-Site Request Forgery, PHP triky, publikováno 24. 4. 2006. http://php.vrana.cz/cross-site-request-forgery.php.

[13] Model-View-Presenter (MVP), Nette Framework – Příručka programátora, stav z 18. 5. 2010. http://doc.nettephp.com/cs/model-view-presenter.

[14] Nette\Forms, Nette Framework – Příručka programátora, stav z 18. 5. 2010. http://doc.nettephp.com/cs/nette-forms.

Příloha A

Seznam použitých zkratek

CSRF Cross-site request forgery

CSV Comma-separated values

GPS Global Positioning System

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

LAN Local area network

MVC Model–View–Controller

MVP Model–View–Presenter

NAT Network address translation

OTR Off-the-Record Messaging

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

UML Unified Modeling Language

VOIP Voice over Internet Protocol

XSS Cross-site scripting

Příloha B

Obsah přiloženého CD

```
Kořenový adresář
|---- readme.txt                  instrukce k použití
|---- src                         adresář se zdrojovými kódy
|    |---- app                    samotná aplikace
|    |---- document_root          adresář dostupný z webového serveru
|    |---- libs                   pomocné knihovny
|    \---- database.sql           SQL skript pro vytvoření databáze
|---- text
|    \---- bp-dundalek.pdf        text této práce
\---- tools
    |---- cesta.json              data fiktivní trasy
    \---- data-load.py            program k simulaci přenosu dat
```