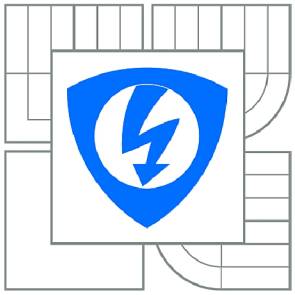
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

ZABEZPEČENÍ PŘENOSU DAT MEZI KLIENTY A SERVEREM PŘES HESSIAN PROTOKOL

SECURED DATA TRANSFER BETWEEN CLIENT AND SERVER USING HESSIAN PROTOCOL

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

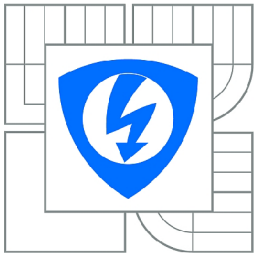
AUTOR PRÁCE — JAN HOŘEJŠ

AUTHOR

VEDOUCÍ PRÁCE — Ing. RADIM BURGET, Ph.D.

SUPERVISOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Jan Hořejš	***ID:***	129352
Ročník:	3	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Zabezpečení přenosu dat mezi klienty a serverem přes hessian protokol

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s aplikačním serverem Tomcat a protokolem Hessian. Popište a srovnejte s obdobnými technologiemi. Realizujte datovou komunikaci prostřednictvím protokolu hessian a zabezpečte komunikaci pomocí protokolu HTTPS. Výsledek popište a zhodnoťte jeho výhody a nevýhody.

DOPORUČENÁ LITERATURA:

[1] V. Chopra, S. Li, J. Genender, Professional Apache Tomcat 6 (WROX Professional Guides), Wrox
[2] R. Johnson, Expert One-on-One J2EE Design and Development, Wrox

Termín zadání:	6.2.2012	***Termín odevzdání:***	31.5.2012

Vedoucí práce: Ing. Radim Burget, Ph.D.
Konzultanti bakalářské práce:

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

ABSTRAKT

Cílem práce byla volba a vytvoření zabezpečené komunikace mezi serverem a klientem tak, aby bylo možné přenášet citlivá lékařská data skrze nezabezpečenou síť. Teoretická část se zaměřuje na popis a vlastností jak klientské, tak serverové aplikace. Dále jsou popsány metaprotokoly, jejich účel a vhodné použití. Praktická část se zabývá vytvořením zabezpečené komunikace a vytvořením demonstrační aplikace pro přenos zabezpečených dat. Je demonstrováno nastavení serverové, vytvoření klientské aplikace a sestavení certifikátu včetně jeho instalace.

KLÍČOVÁ SLOVA

Hessian, metaprotokol, Vzdálené volání procedur, HTTPS, Apache Tomcat, zabezpečení komunikace, zdravotnictví, klientská aplikace, serverová aplikace, TLS/SSL, vytvoření certifikátu, keytool, JAVA, aplikace, přenos dat

ABSTRACT

The objective of this work was the choice and creating of a secure communication protocols between server and a client in order to transmit sensitive medical data through an unsecured network. The first part is focused on a theoretical description and the properties of both client and server applications, metaprotocols, their purpose and thein appropriate use. The second part describes creation of secure communication and creation of demonstration application for secure data transmission. It is demonstrated by setting the server, creation of client applications and compilation of the certificate, including its installation.

KEY WORDS

Hessian, metaprotocol, Remote call procedure, HTTPS, Apache Tomcat, secure communication, health care, client application, server application, TLS/SSL, creation of certificate, keytool, JAVA, application, data transmission

HOŘEJŠ, J. *Zabezpečení přenosu dat mezi klienty a serverem přes Hessian protokol*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. 49 s. Vedoucí bakalářské práce: Ing. Radim Burget, Ph.D..

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma Zabezpečení přenosu dat mezi klienty a serverem přes Hessian protokol jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne 31.05.2012 .....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce Ing. Radim Burget, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne 31.05.2012 .....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Vzhledem k rozmachu komunikačních technologií, které se za poslední dekádu rozrostly exponenciálně do všech odvětví, také ovlivnily naše myšlení. Už jsou pryč doby, kdy jsme stavěli mohutné osobní počítače s velkým množstvím datového prostoru. Toto řešení bylo v té době výhodné, měli jsme absolutní kontrolu nad uloženými daty, ale jen stěží jsme mohli svá data mít přístupná z více míst, která by nám umožnila mobilitu, také pokud jsme chtěli získat nějaký výsledek z těchto dat, použil počítač celý svůj výpočetní výkon na získání této informace, což zpomalilo naši prací a i přes to, výpočet trval neúměrně dlouho.

Právě s rozmachem Internetu a dalších technologií, uživatelé a firmy spatřili potencionální výhody klienta-serveru. Přesun dat na server umožnilo snížení požadavků na pracovní stanice, které už nemusí skladovat velká množství dat, jež jsou pro všechna zařízení společná, využijí serveru, ze kterého si stáhnou jenom data, která jsou pro ně aktuálně potřebná. Tímto se docílilo mobility. Pokud si chceme vytvořit další pracovní stanici, stačí ji pouze připojit do sítě. Správa celého systému se výrazně usnadnila, není nutné obcházet jednotlivé stanice, ale vše se děje na straně serveru včetně zálohování, které je jednou z hlavních předností serveru. Rozhodnutí, která varianta je vhodnější, záleží samozřejmě z velké části na ceně. Ta je vyšší pro serverové řešení, naopak s vyšším počtem uživatelů je právě toto řešení ekonomičtější a vhodnější.

Velkou nevýhodou této komunikace je to, že data jsou přenášena mimo jiné skrze síť Internet, kde mohou uživatelé čelit potencionálním útokům. Abychom zamezili zcizení citlivých údajů či jejich pozměnění, je nutné cestu mezi klientem a serverem zabezpečit.

Hlavní přínos této práce spočívá ve vytvoření aplikace v jazyce JAVA, která zabezpečuje přenos citlivých lékařských dat mezi klientskou aplikací a vzdáleným serverem. Tato aplikace zpracovává data a výsledky předává skrze internetovou síť zpět k lékaři tak, aby byla tato data chráněna před neoprávněným zásahem zvenčí. Dalším přínosem této práce je zmapování různých metaprotokolů a vhodném výběru mezi nimi. Také je zde popsána tvorba vlastního certifikátu do klientské, tak i serverové aplikace.

Zbytek této práce je členěn následovně. Následující kapitola je zaměřena na důvody zabezpečení dat, jejich pozměňování, zneužití a souvislosti s projektem zdravotnictví. Třetí kapitola popisuje jednotlivé prvky, které jsou nutné pro komunikaci. Jsou zde zmíněny různé typy klientských a serverových aplikací. Následně jsou popsány a porovnány metaprotokoly, které se využívají pro komunikaci mezi serverem a klientem. Čtvrtá kapitola, se zaměřuje na implementaci zabezpečení dat a to za použití protokolu HTTPS, šifrujících protokolů SSL/TLS a binárního protokolu Hessian. Po té přejdeme na samotnou konfiguraci spojení jak na straně aplikace, tak na straně serveru a tím sestavíme zabezpečenou komunikaci. V páté kapitole je popsána implementace samotné aplikace „Negativ app“ která slouží k demonstraci vzdálené komunikace mezi serverem a klientem přes protokol Hessian. Její funkci je možné v budoucnu nahradit libovolným jiným algoritmem pro zpracování a analýzu obrazu.

1 Zdravotnictví a citlivá data

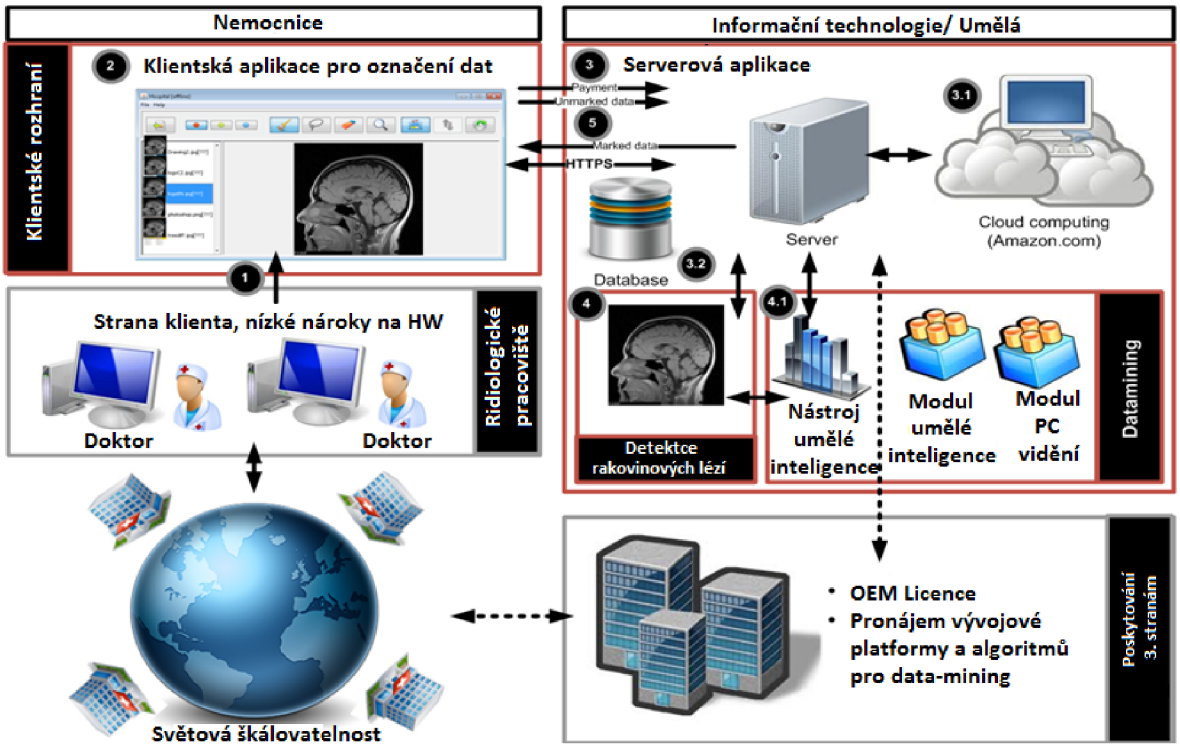
1.1. Souvislost s projektem zdravotnictví

Rozdíl mezi minulostí a dnešní dobou spočívá v moderních technologiích. Tyto technologie změnily naše životy a usnadňují nám každodenní život. Mezi tyto technologie se řadí Internet, chytré telefony, přes automobily až po automatické nastavování teploty v pokoji. S moderními technologiemi také vznikají nové otázky a problémy, které se snažíme vyřešit. Jednou z otázek je, co se dá dělat s velkým množstvím různých dat, která chytrá zařízení nasbírají. Využití těchto dat je opravdu široké například pro zjištění optimálních dopravních cest, přes cílenou reklamu, až po lékařství.

Jelikož se nacházíme v moderní době, která využívá nových technologií všude o kolo nás, jako chytré telefony, Internet, předpověď počasí a mnohé další. S používáním těchto technologií vzniká velké množství dat, která se učíme zpracovávat a vhodně využívat. Tato data mohou pomoci například v lékařství.

Medicína se za poslední desetiletí výrazně rozvíjí, tomu pomohly moderní přístroje a technologie. V současnosti už je jen stěží představitelné nevyužití těchto moderních zařízení pro léčbu, například laserové nože, kardiostimulátory a jiná zařízení, která ulehčí provádění obtížných zákroků. Další nedílnou součástí péče je samotná diagnostika. Jestliže budeme pomocí těchto moderních zařízení včas a přesně diagnostikovat různé nemoci, zvýšíme tím možnost léčby, ne-li přežití pacienta.

K tomuto účelu slouží aplikace pro lékaře, která pomocí velkého výpočetního výkonu, rozpoznáváním objektu, dokáže jednodušeji, přesněji a rychleji identifikovat například tumory ze snímků magnetické rezonance.



Obr. 1.1: Schéma aplikace, zdroj: poskytl vedoucí práce Ing. Radim Burget, Ph.D.

Ze schématu je patrný princip aplikace. Lékař si zobrazí výsledky z magnetické rezonance, které se automaticky zašlou na vzdálený server, zde jsou data roztříděna, zpracována a následně navrácena lékaři, který pomocí těchto dat lépe diagnostikuje pacienta a může navrhnout příslušnou léčbu.

1.2. Proč zabezpečujeme, získávání citlivých údajů

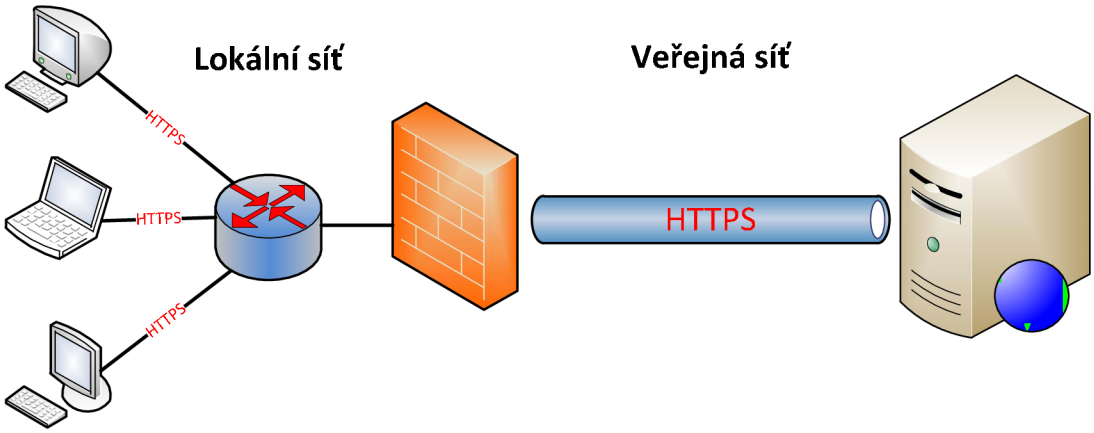
Aplikace pro lékaře je především určena ke zpracovávání dat. Tato data se vztahují k určitým osobám, proto se tedy jedná o osobní data. Tyto informace podléhají přísné kontrole podle zákona na ochranu osobních dat a informací dle zák. č.101/2001 Sb., v platném znění [2]. V tomto zákonu jsou přísně popsána pravidla pro zacházení s těmito údaji. Dále se na ně také vztahuje lékařská tajemství. To jsou hlavní důvody, proč je vyžadováno zabezpečení těchto citlivých dat.

Jelikož aplikace je určena pro přenos dat, musí být již od počátku navržena a přizpůsobena pro tyto bezpečnostní požadavky. Jestliže chceme přenášet informace na vzdálený server, musíme si přenos rozdělit na tři základní segmenty:

1. První segment je tvořen lokální sítí, kterou spravuje lokální administrátor. Ten má za úkol ochránit danou síť před útoky zvenčí, ale hlavně i zevnitř. Musí dohlédnout na fyzické umístění zařízení, aby nebyla dostupná všem. Dále nastavuje, omezuje a zaznamenává přístupová práva uživatelů. Proto může být vhodnější data uchovávat mimo tuto síť, což i potencionálnímu útočníkovi sníží šance na zjištění dat, jelikož nebude vědět, kde se data nachází. V neposlední řadě je nutné proškolit samotné uživatele, aby správně zacházeli se systémem a programy.

2. Druhou část tvoří samotný přenos dat. Ten může napadnout útočník za pomocí podvrhování zpráv, dDOS útoků a pod.. Tyto problémy by měli být odstraněny pomocí správně nastavených firewallů, monitoringem od síťového poskytovatele a samozřejmě samotnou aplikací využívající zabezpečený přenos dat.

3. Posledním segmentem je samotný server. Pro něj platí obdobná pravidla jako pro lokální síť, ale mnohem přísněji. Je zde striktně omezován pohyb nepovolaných osob, velmi dobře nastavena síť a její politika. Datová centra jsou i lépe fyzicky chráněna pomocí ostrahy, umístění atd.. I kdyby útočník pronikl do těchto prostor, nebude v množství serverů schopen identifikovat svůj cíl. Posledním a nejdůležitějším důvodem zabezpečení je záloha. Všechna data v serverových místnostech jsou redundantně zálohována na více médií, lokací a podléhají neustálé kontrole, před poruchami a vnějšími vlivy.



Obr. 1.2: Schéma síťové komunikace

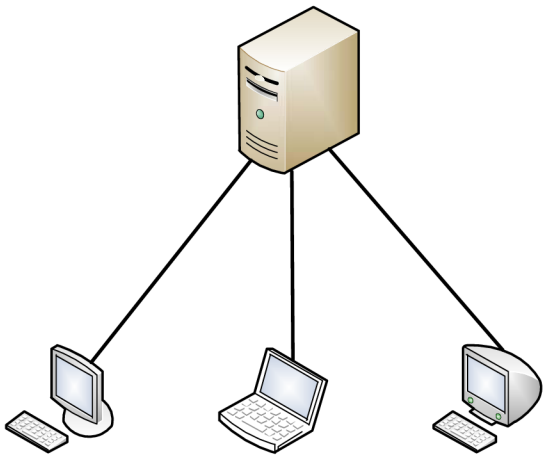
Úniky citlivých dat, podle výzkumu znaleckého ústavu APOGEO [13], jsou z 50% způsobeny nedbalostí, 13% ztrátou datových nosičů, dále 12% krádeží, útoky hackerů jsou tvořeny 14%. Z výzkumu také vyplývá, že pokud organizace vlastní citlivá data, čelí až šestinásobnému riziku útoku. Výsledný systém zabezpečení je komplexním balíkem všech nástrojů na všech vrstvách komunikace a celý systém je zabezpečený tak dobře, jako ten nejslabší prvek.

2 KLIENTSKÁ A SERVEROVÁ APLIKACE, METAPROTOKOLY

2.1. Koncept klient-server

Koncept klienta-serveru popisuje vztah mezi dvěma počítačovými programy. První je zvaný klient, který nesdílí své zdroje a pro zjištění dalších informací se dotazuje druhého programu, zvaného server. Server se skládá z propojení více serverových programů, které vzájemně sdílejí své zdroje, pomocí nichž server odpovídá klientovi na jeho dotaz. Tento koncept může sice běžet na jediném počítači, ale s rostoucím požadavkem na informace, které jsou distribuovány skrze různá místa, se uchytil způsob distribuce přes Internet.

Komunikace mezi klientem-serverem je velmi častá aniž si to uvědomujeme. Příkladem může být poštovní klient, kterého používáme každý den. Klient zašle dotaz na emailový server, ten aby splnil požadavek, musí klienta ověřit, proto se dotáže na databázový server, který ověří přihlašovací údaje a následně vrátí emailovému serveru zprávu, že se klient úspěšně přihlásil a může činit další požadavky. Mnohé aplikace jsou napsány za pomoci modelu klient-server. Stejně jako hlavní Internetové aplikační protokoly, jako HTTP, sloužící pro přenos WWW stránek, SMTP, přenos pošty, FTP a další.



Obr. 2.1: Topologie klienta-serveru

Výhody systému klient – server:

- Přístup k velkému množství informací, která jsou umístěna na různých místech.
- Podpora obsluhy více klientů a sdílení společných zdrojů.
- Multiplatformní klientská aplikace, může být napsána pro různé operační systémy.

- Bezpečnost uložených dat z důvodu poruch (lepší zálohovatelnost).
- Nenáročnost na klientský hardware, v případě nedostatku se navýší výkon serverů.

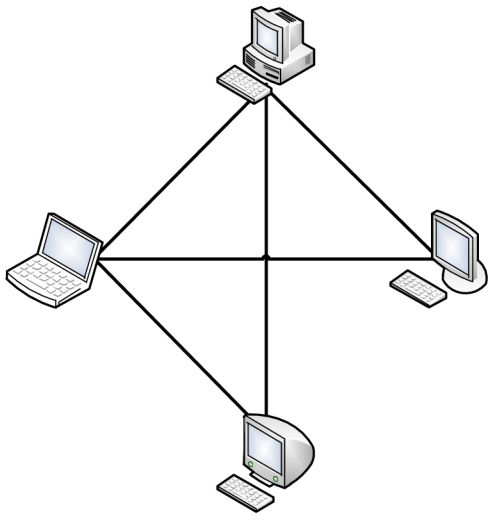
Nevýhody:

- Nutnost neustálého připojení k Internetu (datové poplatky, zatížení linky).
- Možná nedostupnost v případě poruch.
- Zachytávání a pozměňování přenášených dat.

Porovnání s peer-to-peer (P2P, klient-klient) architekturou:

Klient-server využívá pro připojení více klientů jeden centrální server. To má za následek, že tento server je velmi vytěžovaný a je zde nutnost vysoké datové propustnosti, výkonu a diskového prostoru.

Jinou možností je Peer-to-Peer (P2P), která využívá princip sdílení. Dva a více počítačů fungují zároveň jako klient a server. To umožňuje sdílení dat, tiskáren atd., která jsou dostupná všem zařízením. Výhodou je decentralizovaná struktura, ta zajišťuje bezpečnost v případě útoků. Dále jednoduchost instalace, nízké pořizovací náklady. Nevýhodou je sdílení vlastního výkonu, nízká možnost administrace a konfigurace. V případě větších sítí je lepší uvažovat o serverovém řešení, které umožňuje standardizovaní softwaru, hardwaru, jednodušší konfiguraci a administraci.



Obr. 2.2: Topologie typu Peer-to-Peer

2.2. Popis klientské aplikace

V předchozí kapitole byl představen koncept klient-server. Nyní bude představena blíže klientská část. Pod tímto pojmem se může jednat o aplikaci nebo o samotný počítač. V dnešní době rozdělujeme klienty do tří skupin. První je tlustý klient (fat, thick client) opakem k tlustému je tenký klient (thin client) a poslední hybridní. Nyní si rozebereme vlastnosti jednotlivých klientů a porovnáme je.

2.2.1. Tenký klient

Takzvaná bezdisková stanice je velmi jednoduchý počítač sloužící pro připojení k serveru, na kterém se provádí veškerá práce klienta. Stanice většinou obsahuje pouze webový prohlížeč nebo program vzdálené plochy.

Výhody bezdiskových stanic:

- Nízké pořizovací náklady. Ve stanici nejsou potřeba disky, výkonné procesory či grafické karty. Výpočetní životnost těchto zařízení se prodlužuje z důvodu nenáročnosti, lze provozovat i na zastaralých zařízeních.
- Spotřeba energie je nízká, tím se šetří náklady za elektřinu a není nutné aktivní chlazení.
- Zabezpečení je plně přeneseno na server, kde funguje lepší správa a ochrana dat, včetně zálohování.
- Jednoduchá opravitelnost. V případě poruchy stačí klienta vyměnit za jiného. Veškerá data jsou stažena ze serveru a okamžitě použitelná pro práci.
- Při nedostatečném výkonu stačí modernizovat serverovou část.

2.2.2. Tlustý klient

Poskytuje plnou funkčnost nezávislou na serveru. Tlustý klient, oproti „tenkému“, využívá v co největší míře svůj vlastní výpočetní výkon. Tím se stává nezávislým na síťové komunikaci se serverem, kterou využívá jen v omezené míře.

Výhody tlustého klienta

- Nízké nároky na síťovou komunikaci.
- Více volného výpočetního výkonu serveru, který je volný pro další uživatele.

- Větší multimediální výkon, který je nutný pro střih videa, kreslení v grafických programech, hraní her a jiné.
- Lepší funkčnost aplikací. Většina aplikací je psána pro desktopy a ne vždy funguje na serverech.
- Možnost práce i při ztrátě připojení.
- Lokální ukládání dat a aplikací, která není nutno při každém spuštění stahovat.

2.2.3. Hybridní klient

Je aplikace ležící mezi dvěma již zmíněnými a využívá vlastnosti obou. Hybridní klient je velmi podobný tlustému klientovi, který zpracovává data lokálně, využívá svůj výpočetní výkon, tím může spouštět i náročné multimediální aplikace, ale pro ukládání dat využívá server, díky němuž jsou data lépe chráněna a zpracovatelná z více míst.

Každý klient má své výhody a nevýhody. Nedá se plošně určit kde použít „tlustého“ či „tenkého“ klienta. Je nutné individuálně posoudit, k jakému účelu budou jednotlivé stanice určeny a následně se rozhodnout. Jistým indikátorem může být množství uživatelů. Tlustý klient je vhodný tam, kde není mnoho stanic, proto není nutno budovat výkonný server, naopak pozitiva převažují pro tenkého klienta tam, kde je velké množství stanic, příkladem jsou školy, úřady a nemocnice.

2.3. Popis serverové aplikace

V minulých odstavcích byla blíže popsána architektura klienta-serveru a probrána klientská část. Jako poslední je serverová část tedy aplikační server. Aplikační server je spojován s třívrstvou architekturou, případně architekturou tenkého klienta. Ten je součástí tzv. třívrstvé architektury. První vrstva je tvořena aplikací uživatele, druhá je aplikační server a poslední je databáze, kde jsou uložena data.

Aplikační server vytváří vrstvu mezi operačním systémem a aplikacemi, pomocí nichž je psaní jednodušší a praktičtější. Toho nejčastěji využívají firemní (enterprise) aplikace, které jsou běžnými aplikacemi, ale jsou na ně kladeny mnohem větší nároky na spolehlivost, výkon, dostupnost a také by měly být navrženy v souladu se servisně orientovanou architekturou (SOA), která má za cíl maximalizovat efektivitu a flexibilitu. Klasickým příkladem je aplikace, která musí obsloužit mnoho uživatelů a jejich požadavky v reálném čase.

Valná většina dnešních aplikačních serverů vychází z programovacího jazyka Java, nejen proto, že Java je multiplatformní a velmi rozšířená, ale také pro svoji podporu standardu Java Enterprise Edition (JEE), která je pomocí hromadné spolupráce různých firem v rámci Java Community Process (JCP), určena pro vývoj a provoz podnikových aplikací. Všechny servery nejsou JEE kompatibilní. Pro získání certifikátu je nutné, aby prošly zátěžovými testy kompatibility. Některé aplikace využijí pouze část platformy nebo ji dokonce ignorují. Může se stát, že specifikace splňují, ale nemají certifikaci.

Aplikační server JEE zahrnuje:

- Application programming interface (API) - Specifikace, podle kterých se může softwarový program řídit při komunikaci mezi sebou.
- Vývoj webových aplikací - Java Servlets, Java Server Pages (JSP), Java Server Faces (JSF)
- Vývoj sdílené podnikové logiky - Enterprise Java Beans (EJB), Spring
- Přístup k dědičným systémům - Java Connector Architecture (JCA), Hibernace
- Přístup ke zprávovému middleware - Java Messaging Services (JMS)
- Komponenty zajišťující integraci webových aplikací a portálů - Portlety
- Podpora technologií Webových služeb

Java EE aplikační servery je možno dělit do dvou hlavních kategorií:

- Open Source aplikační servery (aplikační servery s otevřeným zdrojovým kódem)
    - Jboss
    - Apache Tomcat (částečná implementace)
    - JOnAS

- Komerční aplikační servery
    - IBM WebSphere
    - Oracle AS
    - Sun Java System Application Server
    - BEA WebLogic

Jedním z hlavních rozdílů mezi open sourcem a komerčními programy je bezpečnost. Open source využívá otevřených zdrojových kódů a mnohem větší skupinu lidí, kteří mohou vyřešit bezpečnostní a jiné chyby v programu, avšak tyto informace má také potencionální útočník. Komerční aplikace jsou uzavřené, nepracuje na nich tolik lidí a opravy jsou složitější. Útočník se ale nedozví o případných chybách. Obecně je však vhodnější, aby informace byly dostupné všem, protože větší skupina lidí má více prostředků a informací pro vytvoření opravy než útočník.

Zvláštním případem je Apache Tomcat, který byl zařazen do aplikačních serverů s poznámkou o částečné implementaci. Pro pochopení pozice Tomcatu je nutné si ujasnit, co je webový server a jaký je rozdíl mezi ním a aplikačním serverem.

Primární funkcí webového serveru je zobrazení webových stránek na základě požadavku uživatele (HTTP requests/responses). Web server je tedy určen k zobrazení HTML dokumentů, obrázků a přesměrování, tedy zobrazení statického obsahu. Aplikační server dělá v podstatě to stejné, avšak čeká na jakémkoliv portu, využívá

jakýkoliv protokol a zobrazuje dynamický obsah. Můžeme tedy říci, že webový server je součástí aplikačního serveru.

Pro ujasnění pozice Tomcatu je nutné se podívat na jeho vývoj. Tomcat začal jako JServ modul pro Apache HTTPD (dnes pod názvem Apache web server). Byl to Servlet container (programovací jazyková třída Javy, pomoci nichž jde vytvořit dynamický obsah), který nebyl schopný sám zvládat webové požadavky. Skombinováním došlo k tomu, že všechny statické požadavky zpracoval webový server a dynamické požadavky postoupil JServu. Časem JServ byl nahrazen Tomcat aplikačním serverem, byl mu přidán konektor, což umožnilo Tomcatu fungovat samostatně jako webový/aplikační server, ale bez podpory SSL, tu využíval opět spojením s Apache web serverem. Postupně přibyla Tomcatu plná SSL podpora viz kapitola 3.1.1. a hlavně se výrazně zlepšil výkon, kvůli kterému je dnes velmi populární.

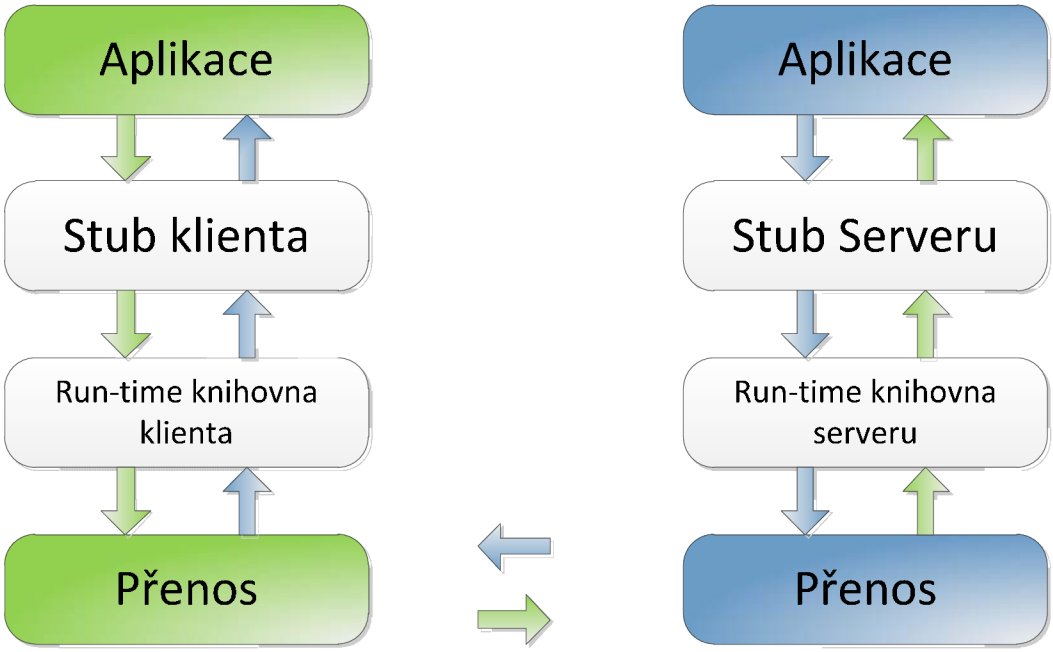
Jak můžeme vidět, je velmi těžké určit, do které skupiny patří. Tomcat je tedy webový server a Servlet/Serverový stránkový kontejner. Často je používán jako aplikační server výlučně pro webově založené aplikace, ale neobsahuje kompletní balíček JavaEE, který by měl webový server obsahovat. Tomcat je velmi používaným serverem z důvodu jeho jednoduchosti, která v případě nutnosti se dá velmi jednoduše rozšířit. Ostatní aplikační servery přicházejí jako kompletní balíky, které jsou nabobtnalé a pomalé, samozřejmě se dají zrychlit odstraněním nepoužívaných součástí.

Minoritní část tvoří aplikační server využívající platformu .NET framework, která je především určená pro společnost Microsoft a která je použita v operačním serveru Windows Server. Operační systém Microsoft Server vytvoří novou roli, pomocí aplikace psané v .NET frameworku. Velkou nevýhodou aplikací využívající .NET je, že fungují pouze na operačních systémech společnosti Microsoft.

2.4. Vzdálené volání procedur

Remote Procedure Call (RPC), v češtině Vzdálené volání procedur, je protokol aplikační vrstvy, který může požadovat služby programu umístěného na jiném počítači v cizí síti, bez nutnosti, aby programátor musel složitě psát kód pro tuto komunikaci a znal síťovou strukturu. RPC využívá modelu klient-server. Stejně jako lokální volání procedur, RPC je synchronní operace požadující program, aby se zastavil, než je výsledek vzdálené procedury navrácen zpět klientovi. Nicméně použití lightweight procesů (umožňující provádět několik procesů současně tzv. multitasking) nebo vláken (threads), které sdílejí stejný adresní prostor, umožňuje několikanásobné volání RPC současně.

Postup volání je velmi jednoduchý. Klient zavolá lokální službu včetně všech parametrů a identifikátorů. Tyto informace jsou převedeny do formy vhodné pro přenos mezi počítači (tzv. marshalling) využívající External Data Representation format (XDR). Následně jsou data odeslána a na straně serveru opět přijata. Provede se unmarshalling, tedy rozbalení dat z formátu XDR, která se následně použijí pro samotné zavolání a provedení služby. Výsledek volání se znovu převede do formátu XDR a celý postup se opakuje. Klientem přijatá hodnota je předána příslušné proceduře.



Obr. 2.3: Vzdálené volání procedur

RPC využívá pro přenos protokoly transportní vrstvy, tedy nespolehlivou nepotvrzovanou službu UDP nebo spolehlivou pomalejší potvrzovanou službu TCP. Dále přiřazuje různým relacím programová čísla, podle kterých mohou být identifikovány. Jelikož RPC nevyužívá rezervovaných portů, není zde žádná garance, že originální port je dostupný a nebyl zabrán jiným procesem. Z toho důvodu si RPC vybere náhodný port, který si zaregistruje u speciálního programu zvaného Portmapper daemon. Portmapper funguje jako zprostředkovatel pro všechny RPC servery běžící na zařízení. Klient, který chce začít komunikovat, nejdříve osloví Portmappera na serveru s přiděleným programovým číslem, ten mu navrátí TCP a UDP číslo portu, na kterém je dostupný.

Vzdálené volání procedur je velmi dobrý protokol, který umí odlehčit výpočetnímu výkonu klienta a přenést jej na server. Nevýhodou je vyžadování neustálého internetového připojení a spolehlivosti bezchybného chodu. Klient v důsledku poruchy v síti je nucen se vyrovnat s takto vzniklými chybami.

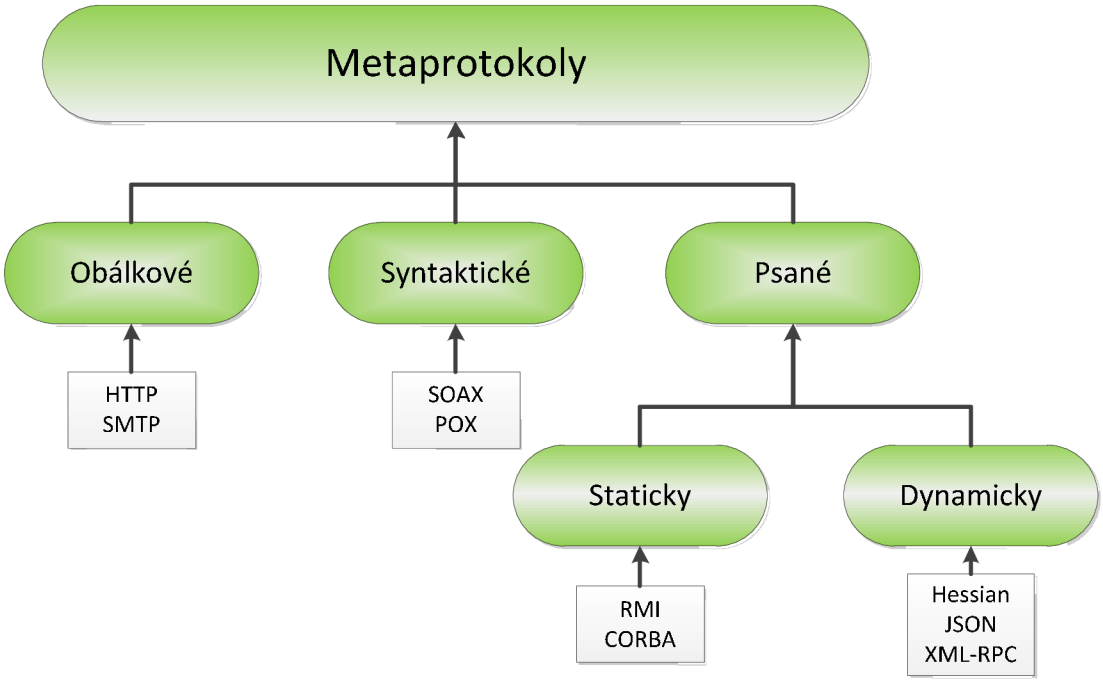
2.5. Metaprotokoly

Metaprotokoly jsou zvláštními případy protokolů, které pro svůj běh potřebují plnohodnotný protokol, bez kterých by samostatně nemohli fungovat. Tyto metaprotokoly mají za účel zajištění různých druhů komunikace mezi aplikacemi skrze síť. Dále také usnadňují programátorovi kódování tohoto spojení.

Vybrání správného metaprotokolu je hlavní stavební rozhodnutí, které ovlivní výkon, spolehlivost, udržovatelnost a rozvojové úsilí. Existují různé metaprotokoly jako

Hessian, který může být spolehlivý a jednoduchý a dobře zapadne do aplikací pro proudové vysílání (streaming) nebo RPC. Dalším třeba SOAP, který používá existující WSDL (Web Services Description Language) specifikaci, dále si rozebereme a zařadíme různé protokoly, abychom mohli zvolit ten správný pro lékařskou aplikaci.

Metaprotokoly jako SOAP, CORBA, JSON nebo Hessian jsou specificky navrženy, aby například vytvořili NFS (Network File System), protokol pro vzdálený přístup, nebo Atom Publishing Protocol, umožňující vytvářet a aktualizovat webové zdroje ve formátu Atom pomocí WWW. Protože metaprotokoly požadují využití typů a specifikací, nejedná se o kompletní protokoly. Příkladem budiž NFS ve verzi 3, který využíval ONC-RPC jako svůj metaprotokol.



Obr. 2.4: Metaprotokoly

Metaprotokoly jsou odlišné od obálkových protokolů jako HTTP,SMTP, REST atd. Ty mohou být použity pro jejich přenos, ale oni sami nejsou metaprotokoly.

Dynamicky definované Metaprotokoly:

- Příklad: Hessian, JSON, Burlap, XML-RPC.
- Jazyk skriptovací a objektově orientovaný: Flash, JavaScript, Java, C#, Ruby, PHP, atd.
- Specifikace IDL (Interface Description Language) definuje modelovou komunikaci konec-konec (end-to-end).
- Vhodné pro streaming, RPC, REST a Messaging

Staticky definované metaprotokoly:

- Příklad: CORBA, RMI.
- Objektově orientované:Java, C#, C.
- Specifikace IDL definuje modelovou komunikaci konec-konec.
- Vhodné pro streaming a RPC.

Syntaktické metaprotokoly:

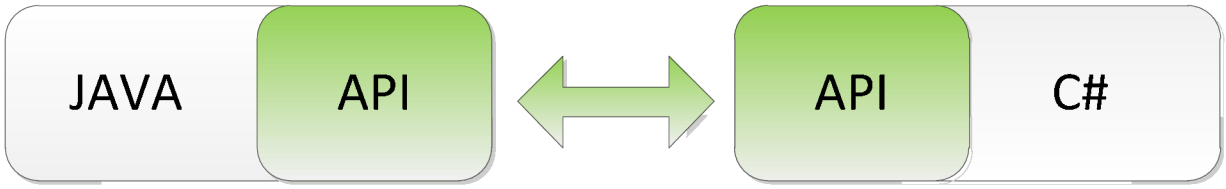
- Příklad: SOAP, POX.
- Jazyk skriptovací a objektově orientovaný: Flash, Java, C#, C, PHP.
- Specifikace IDL (WSDL).
- Mnoho SOAP implementací potřebuje k běhu WSDL.
- Vhodné pro REST a Messaging

Rozdíl mezi dynamickým a statickým metaprotokolem spočívá v tom, že dynamický přenáší všechny datové typy, jako součást protokolu. Například Hessian, Burlap, JSON a XML-RPC. Statický metaprotokol potřebuje externí definici datového typu. V případě CORBA protokolu se použije Interface Definition Language (IDL), u RMI zase třídy Javy, které poskytují vnější definici pomocí zrcadlení.

Syntaktický metaprotokol se zabývá syntaxí přenášených dat.

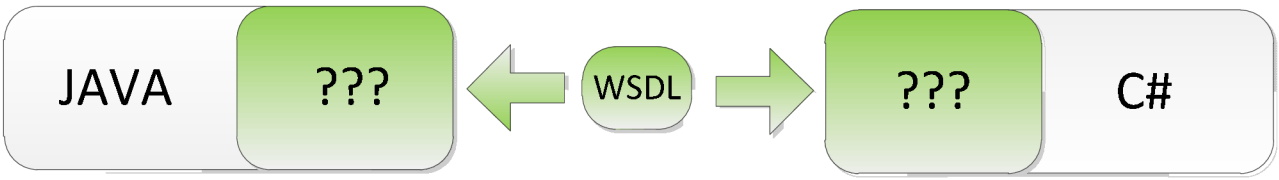
Schopnost docílit vzájemné spolupráce je důležitou součástí pro úspěch aplikace a udržovatelnost.

Datově definované metaprotokoly používají pro vzájemnou komunikaci API, buď použitím IDL pro CORBA protokol nebo objektově orientovaný jazyk schopný zrcadlení jako Java a C# pro Hessian, či Java pro RMI. Cílem je nadefinování koncové specifikace mezi serverem a aplikací.



Obr. 2.5: Hessian protokol

Syntaktické metaprotokoly definují syntaxi přenášeného dokumentu. V případě SOAP nebo POX je dokument definován XML a navíc definován WSDL. Syntaktické metaprotokoly nedefinují API použité v aplikacích. V podstatě na obou stranách je použit Document Object Model (DOM) nebo kombinace DOM a XPath nebo XQuery.



Obr. 2.6: SOAP

Servisně Orientovaná Architektura (SOA)

SOA nepředstavuje konkrétní technologii, ale slouží jako soubor pravidel a postupů pro návrh informačních systémů. Základní dělení je na poskytovatele služby (Service providera) a konzumenta služby (Service consumera). Jednotlivě fungují jako uzavřený celek, který je na venek prezentován svým rozhraním. Jednotlivé služby je tedy možno skládat nezávisle do větších bloků. Nejčastěji je SOA implementována do webových služeb a je charakterizována modularitou, interoperabilitou, opětovném použití, volnými vazbami. Rozhraní je jazykově nezávislé na platformě.

Hessian

Může použít jakýkoliv staticky definovaný objektově orientovaný jazyk jako Java, C#, PHP nebo Ruby. Protože jazyk, ve kterém píšeme, je zároveň aplikačním jazykem, vývojové nástroje a dokumentace poskytují jasné a spolehlivé specifikace. Specifikace API může být mechanicky přeložena z jednoho jazyka do druhého za pomoci zrcadlení. Protože je metaprotokol dynamický, poskytuje koncové specifikace služby.

CORBA

Protokol používá jako své rozhraní IDL. Nástroj pro překlad vytváří API pro specifické jazyky i přes mnohá CORBA řešení jako JavaEE použije Java API pro odvození IDL. Jako staticky psaný metaprotokol také poskytuje koncové API specifikace.

SOAP

Využívá WSDL. Ten nedefinuje objektový model XML. Mnoho SOAP řešení vyžaduje WSDL, aby byla dostupná za chodu, přidávajíc další službu nutnou pro SOA. WSDL dokumenty jsou složité pro člověka ke čtení, proto je nutný nástroj navíc. SOAP aplikace poskytují objektově orientované API mimo protokolové specifikace, což dělá vzájemnou komunikaci složitou.

RMI

Podobně jako Hessian používá standardní programovací jazyky pro API, přestože je omezen na jazyk Java. Díky jazykovým omezením RMI nesplňuje požadavky pro SOA.

JSON

Přestože JSON nepoužívá popisující jazyk a tím se technicky nekvalifikuje jako SOA protokol, muže být nadefinován standard, aby JSON kopíroval objektově orientovaný jazyk podobný Hessianu. Takovéto namapování může umožnit SOA aplikacím použít JSON jako svůj komunikační protokol.

Pro různé druhy komunikací jako RPC, REST, Messiging a Streming může být použit jakýkoliv metaprotokol, přesto některé vyhovují lépe. Pro aplikace vyžadující neustálé aktualizace ze serveru je vhodné použití proudové komunikace. Pro aplikace, které čtou stavy webové služby, jako například RSS čtečka, je vhodnější REST.

Messaging (zasílání zpráv):

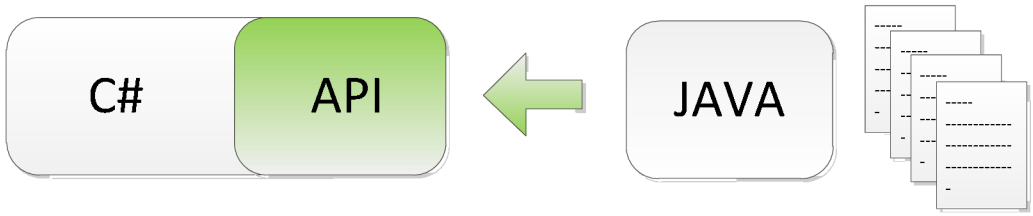
- Řazení ulož a pošli (store and forvard).
- Kompletní dokument požadavků.
- Obálky/hlavičky – SMTP mail.
- Zvýšená bezpečnost.
- Syntaxní a dynamicky definované protokoly: SOAP,POX,Hessian.

Messaging protokol zasílá kompletní dokument požadavku. Druhá strana využívá princip fronty a skladuje požadavky do té doby, než na požadavek dojde řada, tedy frontu typu FIFO (First In, First Out). Jelikož se data ukládají, je nutné zajistit zvýšenou bezpečnost, proto obsahuje šifrování a kontrolu podpisů. Na tento protokol můžou být použity jak syntaktické, tak dynamicky definované metaprotokoly jako POX,JSON nebo Hessian.

Obr. 2.7: Zasílací (Messaging) protokol

REST:

- Určen především pro čtení a mnoho klientů.
- Webový model – HTTP GET.
- Dotazovatelný, možné uložení do vyrovnávací paměti.
- Syntaktický nebo dynamický metaprotokol: POX,JSON,Hessian.



Obr. 2.8: REST protokol

Representational State Transfer (REST) je architektura rozhraní navržená pro distribuované prostředí. REST používá základy webového modelu HTTP GET. Veškerá data jsou dotazována a stahována klientem, jako například výsledky sportovních zápasů nebo RSS čtečka.

Pro model REST mohou být použity dynamické nebo syntaktické metaprotokoly Hessin, JSON, nebo XML. Výběr závisí na účelu aplikace. Pokud je aplikace objektově orientovaná je vhodné použít dynamicky metaprotokoly, ale jestli je aplikace založená na dotazech je vhodné použít XML tedy POX metaprotkol, jelikož SOAP jakožto obálkový protokol by při dotazu HTTP s hlavičkou byl redundantní.

RPC:

- Programovatelné API/model volání.
- Flexibilní, lehce navržitelný.
- Dobré specifikace – IDL, OO programovatelné API.
- Statické a dynamické metaprotkoly: JSON, CORBA,RMI,Hessian

Obr. 2.9: RPC protokol

Vzdálené volání procedur bylo podrobně vysvětleno v kapitole 2.4.. RPC je jednoduchý a dobře popsatelný. RPC aplikace by měly používat statické, nebo dynamické metaprotokoly, které přesně odpovídají programovacímu jazyku. Použití syntaktických metaprotokolů je náročné, protože je nutné vybudovat další vrstvu JAXB na XML nebo starší SOAP-RPC na XML. Jelikož protokolové specifikace jsou syntaxí a ne API, znamená to, že API není blíže specifikováno.

Streaming (proudové vysílání):

- Neustále přenášení paketů, objektů, zpráv.
- AJAX, monitorování, internetová videa, online hry atd..
- Dynamické metaprotokoly: JSON, Hessian,
- CORBA a RMI pouze při simulaci dvousměrného RPC.

Obr. 2.10: Proudové vysílání

Proudové vysílání je vhodné pro aplikace, které neustále monitorují změnu stavů nebo vyžadují další data. Příkladem AJAXové aplikace, online přenosy, hry atd.. Při proudovém vysílání jsou data ze serveru asynchronně posílána jako pakety při změně stavu. Může se použít řízení klient-server nebo RPC, kdy server vyzve klienta, aby aktualizoval svůj stav. Tento model dvoucestného RPC využívají CORBA a RMI. Protože je však streaming objektově založený, dynamické mataprotokoly jako JSON a Hessian jsou vhodné k použití.

3 IMPLEMENTACE ZABEZPEČENÉ KOMUNIKACE MEZI KLIENTEM A SERVEREM

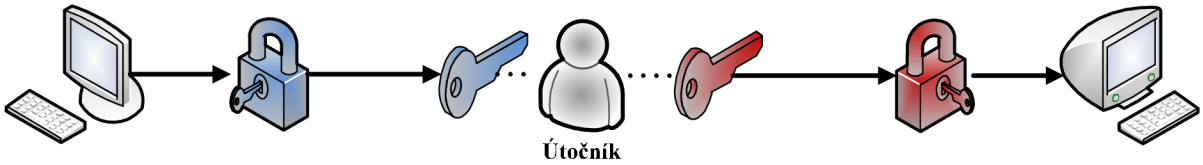
3.1. Zabezpečení přenosu mezi klientem-serverem prostřednictvím HTTPS

HTTPS je zkratka z anglického Hypertext Transfer Protocol Secure. Jedná se v podstatě o klasický protokol HTTP, běžícím na portu 80, přes který například navštěvujeme WWW stránky. HTTPS je jeho nadstavbou, která přidá jednu vrstvu navíc, tedy zabezpečení. Pomocí této vrstvy můžeme zabezpečit komunikaci mezi klientem a serverem, která se dá využít pro přenos citlivých dat, jako čísla kreditních karet, bydliště atd.. HTTPS využívá pro šifrování dat protokoly SSL nebo TLS a mění port obvykle na 443.

Použití HTTPS se dá rozdělit na dva hlavní účely. Tím prvním je šifrování. To má na starost skrytí přenášených dat, změnou posloupnosti různých znaků, což stíží nebo znemožní přečtení originální zprávy. Druhým účelem je identifikace, pomocí které jsme schopni ověřit, s kým komunikujeme a jestli dané osobě či počítači můžeme důvěřovat.

Nejdříve se podíváme na šifrování. HTTPS používá tzv. asymetrické šifrování. Pro představu bude nejlepší příklad. Řekněme, že jsme uživatelé emailové schránky, kteří se chtějí připojit na poštovní server a zkontrolovat si poštu. Aby naše komunikace byla bezpečná, použijeme klíč od serveru, pomocí kterého zabezpečíme zprávu. Server si tuto zprávu pomocí téhož klíče dešifruje. Takto můžeme zabezpečit zprávu, ale je to nepraktické, obtížné a nebezpečné. V případě milionu uživatelů by nebyl žádný problém použít jeden klíč k dešifrování zpráv jiných. Nemluvě o obtížném a bezpečném přenosu tohoto klíče. Z toho důvodu se používá asymetrické šifrování. Server poskytne nám a milionům dalších uživatelům veřejný klíč, jenž je pro všechny stejný. Pomocí něj zprávu zašifrujeme a pošleme serveru. Server je vlastníkem jediného osobního klíče, kterým je schopen zprávu dešifrovat. Princip navázání a ověření probereme dále.

Při komunikaci jak zabezpečené, tak nezabezpečené, můžeme narazit na útočníka, tedy člověka uprostřed, z angličtiny Man in the middle. Útočník odchytává komunikaci mezi dvěma uživateli. Těchto dat může využít ke zjištění hesel nebo také může měnit obsah těchto zpráv. Obvyklým řešením je použití již zmíněného veřejného klíče, ale i to nemusí být dostatečné. Útočník může zachytit veřejný klíč vyslaný jednou stranou a nahradit ho svým vlastním veřejným klíčem, to stejné provede na druhé straně. Oba uživatelé se pak milně domnívají, že spolu navázali komunikaci. Jakákoliv data, která si mezi sebou zašlou, jsou zaslána přes útočníka druhému uživateli. Ochranou před takovýmto útokem je výměna klíčů jiným kanálem (třeba telefonicky) nebo využití certifikační autority.



Obr. 3.1: Útok typu „Člověk uprostřed“

Tím se dostáváme k druhému účelu HTTPS tedy identifikaci protější strany. Veřejný klíč si může vytvořit kdokoliv. V další části bude uveden podrobný návod a následně i demonstrace použití. Uživatelé, ale musí mít jistotu, že ví s kým komunikují, protože např. sdělení čísla své kreditní karty komukoliv je extrémně nebezpečné. Z toho důvodu existují tzv. Certifikační Autority (CA). Jedná se o komerční společnosti, které se zaobírají kontrolou a ověřením daných uživatelů a společností.

Postup ověření je následující:

1. Společnost požádá certifikační autoritu o certifikát. Existují různé společnosti po celém světě příkladem VeriSign, RapidSSL, GeoTrust a jiné. Těmto CA předá název webového serveru, o jakou společnost se jedná, kde se nachází atd..
2. CA ověří tyto informace v různých rejstřících a zašle dopis na udanou adresu. Z těchto dat CA vytvoří certifikát a podepíše ho tak, aby nebylo možné ho změnit. Certifikát obsahuje verzi, sériové číslo, ID algoritmu, platnost, informace o společnosti, veřejný a osobní klíč, identifikátory, podpisový algoritmus z hashe všech těchto informací.
3. Společnost obdrží vytvořený certifikát a nainstaluje ho na svůj server.
4. Webový prohlížeč si ověří certifikát u jedné z těchto společností, jestli je pravý.
5. Prohlížeč důvěřuje certifikátů.

Uživatelé se mylně domnívají, že pokud se jim v adresním řádku zobrazí HTTPS a zámeček, tak je jejich komunikace bezpečná. Webové prohlížeče mají od vydavatele předem nainstalované různé certifikační autority, kterým důvěřují. Vyskakovací okno o problému se zobrazí, pouze pokud daný web neprojde kontrolou. Pokud máme ale velké množství CA a byť jen jedna je nepravá, zdiskreditujeme tím veškeré certifikáty. Certifikáty vydané podvodnou CA jsou podepsané, doména odpovídá a vše se tváří důvěryhodně. Pro uživatele, kteří by chtěli převzít plnou kontrolu nad ověřováním, je nutné smazat záznamy CA. Tím dosáhneme toho, že každý jednotlivý certifikát vlastnoručně ověříme. Nesmíme se však spolehnout na dané stránky a telefonní čísla na nich obsažená.

3.1.1. TLS/SSL

V úvodu bylo řečeno, že protokol HTTPS používá pro šifrování dva protokoly. Ten první je SSL, tedy Secure Socket Layer. Tím druhým je jeho nástupce TLS (Transport Layer Security) kryptografický protokol. Rozdíl mezi protokoly je jen velmi malý. V SSL v3.0 byl použit jako základ pro TLS v1.0., a proto jsou často zaměňovány a uváděny jako SSL/TLS. TLS má silnější šifrovací algoritmus a může pracovat i na jiném portu než 443, umí se také přepnout na SSL komunikaci. Oba protokoly podporují jak asymetrický, tak symetricky klíč. Pro navázání zabezpečeného spojení slouží takzvaný handshake v češtině potřesení rukou. Jedna se o posloupnost zpráv, které slouží k ustanovení spojení.

Pro první fázi klient a server využijí následující kryptografické algoritmy:

- Pro kryptografii s veřejným klíčem: RSA, Diffie-Hellman, DSA.
- Pro symetrické šifrování: RC2, RC4, IDEA, DES, Triple DES, AES, Camellia.
- Pro jednosměrné hashování: Message-Digest algorithm (MD2, MD4, MD5), Secure Hash Algorithm (SHA-1, SHA-2).

Inicializace probíhá následovně:

1. Klient zašle „Hello“ zprávu společně se všemi podporovanými kryptografickým algoritmy, jako verzi, podporované šifry a hash metodu. Zpráva taky obsahuje 28 bytové náhodné číslo.
2. Server odpoví také „Hello“ zprávou, která obsahuje serverem vybrané kryptografické algoritmy, další náhodné číslo a číslo vytvořeného spojení. Server musí podporovat alespoň jednu společnou šifru, aby mohli komunikovat. Server také vybírá nejsilnější šifru ze všech společných.
3. Dále server zašle svůj digitální certifikát (například X.509 digitální certifikát pro SSL). Pokud se použije oboustranné ověření, server zašle „digital certificate request“, tedy dotaz na certifikát klienta a zasílá seznam certifikátů a CA které podporuje.
4. Server odpoví „Hello done“ zprávou a čeká na odpověď.
5. Zatím co server zasílá tuto zprávu, klient kontroluje platnost certifikátu u CA a platnost „Hello“ zprávy serveru. Pokud si server vyžádal certifikát, klient ho zašle nebo odešle zprávu, že certifikát nevlastní „no digital certificate“. Tato zpráva slouží jako upozornění pro server, ale může znamenat ukončení spojení.

6. Klient pošle zprávu na výměnu klíčů „client key exchange“. Zpráva obsahuje pre-master secret, 46 bytové náhodné číslo, které se používá pro generování šifrovacího klíče a Message Authentication Code (MAC) klíč, zašifrovaný veřejným klíčem.
   Jestliže klient zašle digitální certifikát serveru, tak zašle zprávu „digital certificate verify“ zprávu podepsanou klientským osobním klíčem. Ověřením podpisu této zprávy, může server ověřit vlastnictví klientova digitálního podpisu.
7. Klient použije sérii kryptografických operací, aby změnil pre-master secret na master secret, ze kterého jsou všechny klíče pro šifrování a autentifikaci zpráv odvozeny. Následně klient zašle zprávu „change cipher spec“, která řekne serveru, že mění šifrování na nově sjednané. Další zpráva „finished“ je první zpráva zašifrovaná domluvenou metodou a klíči.
8. Server odpoví stejně jako klient zasláním zpráv „chase cipher spec“ a „finished“.
9. Tímto je handshake proces u konce a zařízení mohou začít zabezpečeně výměnu dat.

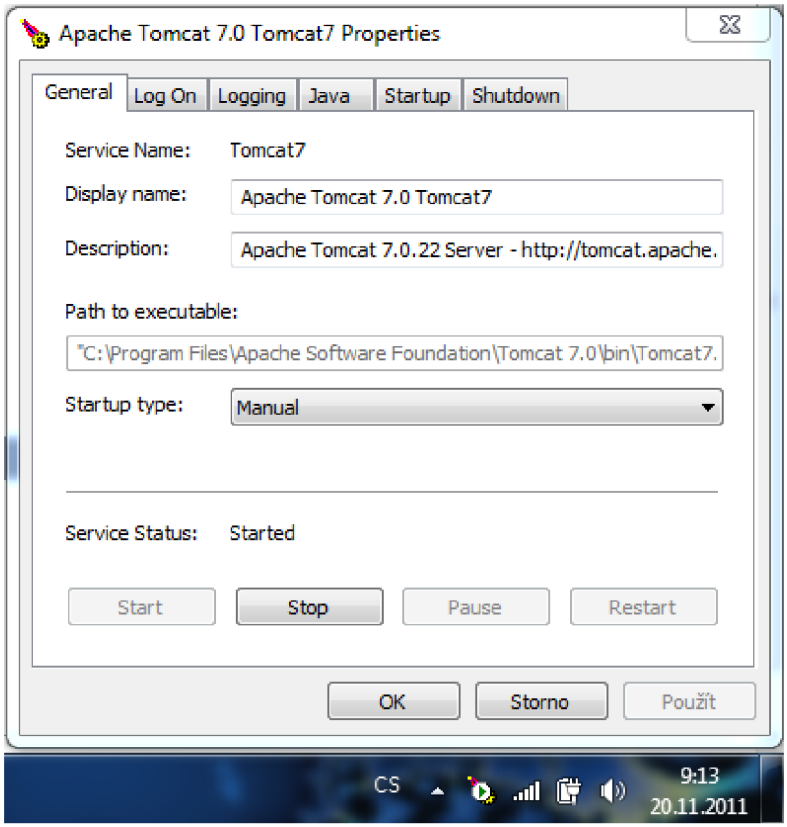
3.2. Popis vybrané varianty a její implementace

Pro nemocniční aplikaci byl zvolen aplikační server Apache Tomcat. Ten bude využívat pro komunikaci s aplikací protokol Hessian a jako zabezpečení použijeme přenos přes HTTPS. Toto spojení umožní vytvořit jednoduchý nenáročný server, který nebude mít problém s průchodem přes síť a firewally. Zároveň všechny použité technologie jsou pod otevřenou licencí, což poskytuje otevřený systém, který neporušuje vlastnická práva a není nutné platit licenční poplatky.

Jelikož byly probrány a vysvětleny všechny pojmy, bude ukázáno, jakým způsobem lze vytvořit a zabezpečit komunikaci pro nemocniční aplikaci. Celý postup je rozdělen na dvě části a to serverovou a aplikační.

3.2.1. Serverová část:

Před samotnou instalací Apache Tomcatu je nutné mít nainstalovanou Javu Standart Edition ve verzi 6 nebo vyšší. Následně je možné zvolit jednu z různých instalací na webu Tomcatu, jak pro 32 bitové systémy, tak i pro 64 bitové systémy. Tomcat je multiplatformní, takže může běžet i na Linuxu nebo Mac OS X. Pro instalaci na Windows je také možno volit z varianty rozbalení do složky nebo instalace, kde je instalace provedena pomocí instalačního programu s volbou umístění, nastavení portů atd.. Další výhodou je grafická aplikace, která se zobrazuje v dolní části lišty a je zde možné zapnout/vypnout Tomcat, nastavit automatické zapnutí a mnohá další užitečná nastavení.



Obr. 3.2: Okno grafického nastavení Apache Tomcat

Tato verze je vhodná především pro domácí použití, třeba při běhu domácího serveru na Windows. Pro serverové využití je vhodnější jedna z neinstalačních verzí, která je méně náročná a nezatěžující zbytečně výkon serveru. Spouštění je prováděno pomocí dávkového souboru startup.bat a vypínání pomocí shutdown.bat v podsložce \\Tomcat\bin\.

Další nastavení se především nastavuje pomocí editace souborů XML v podsložce \\Tomcat\conf\.

Prvním ze souborů je server.xml, kterým se nastavuje vše, co se týká serverové části Tomcat serveru. Především tedy konektory portů. Nejdůležitější jsou konektory pro HTTP a HTTPS.

```
<Connector port="8080" protocol="HTTP/1.1"
connectionTimeout="20000" redirectPort="8443" />
```

Pomocí tohoto konektoru nastavíme, že protokol HTTP bude dostupný na portu 8080. V případě přesměrování bude cílový port zvolen 8443.

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
maxThreads="150" scheme="https" secure="true"
 clientAuth="false" sslProtocol="TLS" keystoreFile=
 ="conf/myCertificate.cer" keystorePass="HoriCZ" />
```

Tento druhý konektor slouží pro HTTPS na standardním portu 8443 s povoleným SSL. Dále je nutné přiřadit cestu k uloženému certifikátu, tedy "conf/myCertificate.cer" a heslo k certifikátu. Popis vytvoření certifikátu je níže.

Dalším souborem je tomcat-users.xml. Jak z názvu vyplývá, soubor slouží pro vytvoření účtů pro správu serveru. Příkladem:

```
<role rolename="tomcat"/>
<role rolename="role1"/>
<user username="tomcat" password="tomcat" roles="tomcat"/>
<user username="both" password="tomcat" roles="tomcat,role1"/>
<user username="role1" password="tomcat" roles="role1"/>
```

Posledním souborem je web.xml sloužící pro nastavení aplikací. Do tohoto souboru byl přidán následující kód:

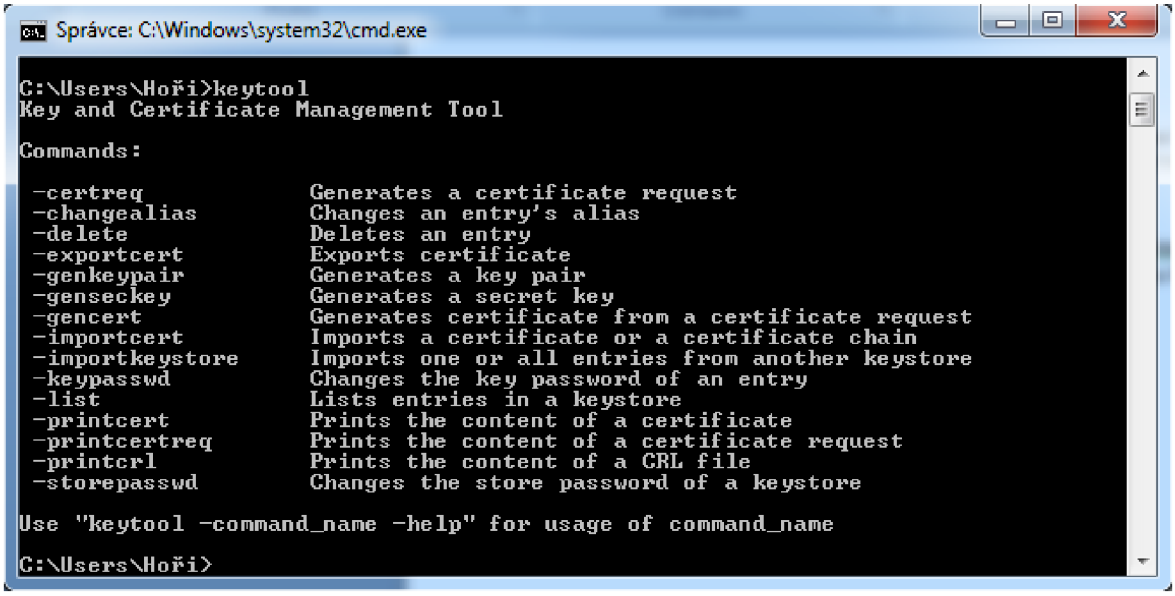
```
<security-constraint>
      <web-resource-collection>
            <web-resource-name>Protected Context</web-resource-name>
            <url-pattern>/*</url-pattern>
      </web-resource-collection>
      <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
      </user-data-constraint>
</security-constraint>
```

Pomocí těchto kódů je zajištěno, že pokud aplikace bude přistupovat na Tomcat přes HTTP na portu 8080 bude automaticky přesměrován na zabezpečenou komunikaci HTTPS na portu 8443.

Při instalaci Tomcatu se může objevit chyba, kdy nebude možné zapnout zabezpečené spojení. Tato chyba se dá jednoduše opravit a to odstraněním souboru tcnative-1.dll ze složky \\Tomcat\bin.

Pro zabezpečenou komunikaci je nutné použít certifikát. Ten vystaví Certifikační Autorita viz kapitola 4.1.. Další možností pro vyzkoušení funkčnosti je vytvoření vlastnoručně podepsaného certifikátu.

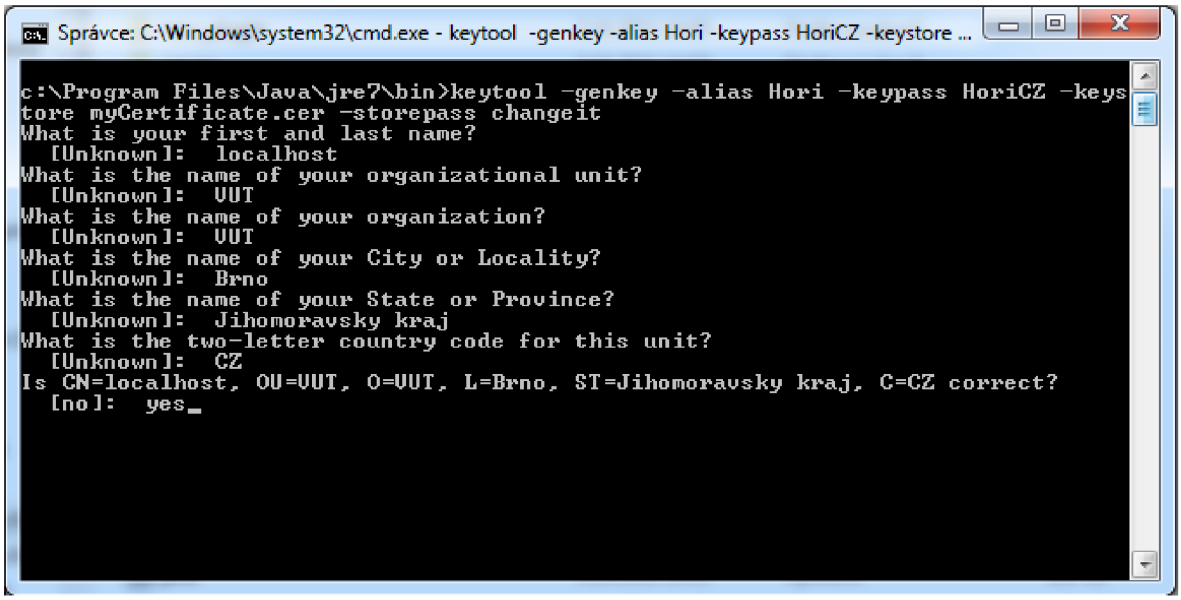
K tomu slouží aplikace keytool, která je součásti balíčku Java, nainstalovaném na každém počítači využívajících Java programů. Aplikace se spouští pomocí příkazového řádku a zadáním potřebných parametrů.



Obr. 3.3: Výpis příkazů aplikace keytool

Jak je z obr. 3.3 patrné, můžeme vytvářet, mazat, importovat a tisknout certifikáty. Pro náš certifikát je využito následujících příkazů:

„keytool -genkey -alias Hori -keypass HoriCZ -keystore myCertificate.cer -storepass changeit“



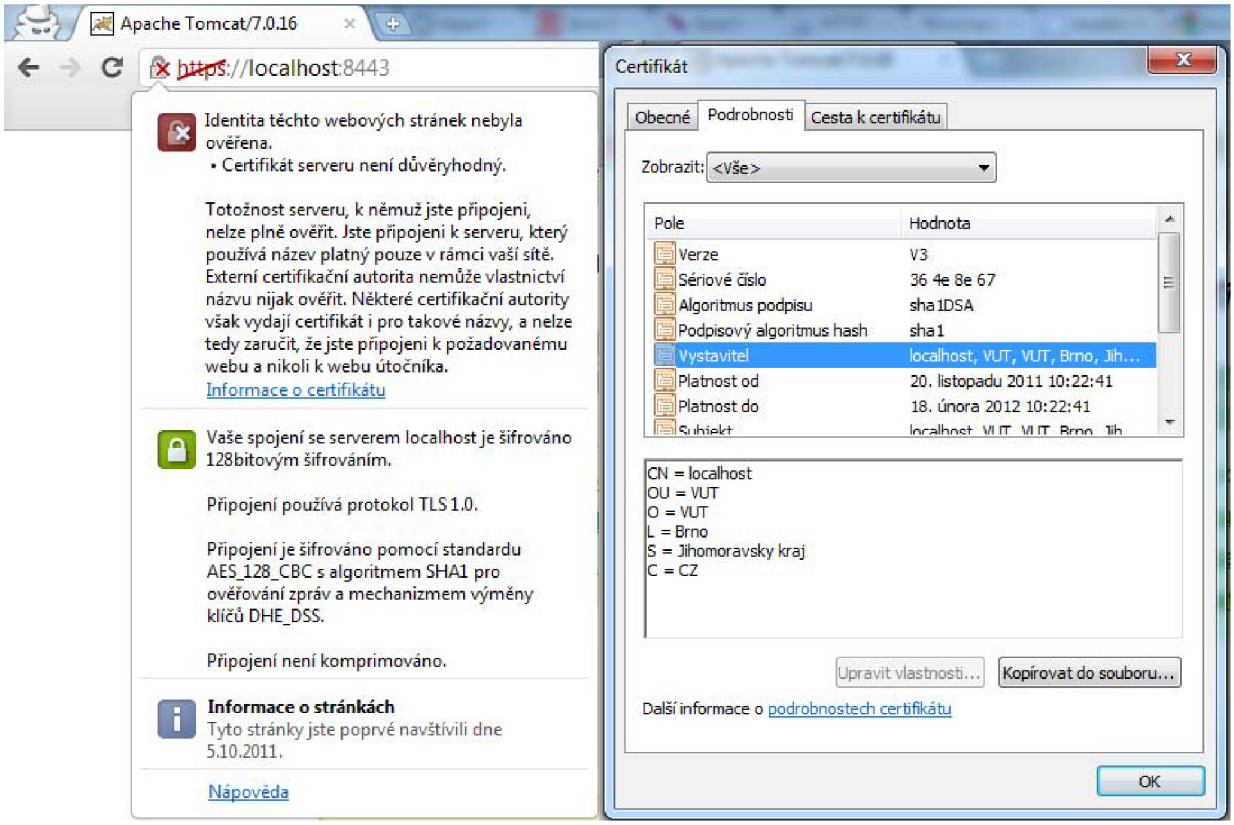
Obr. 3.4: Vytvoření certifikátu

Těmito příkazy se vygeneruje nový certifikát s názvem Hori, heslem HoriCZ, jménem certifikátu myCertificate.cer a heslem do uložiště certifikátů. Výchozí heslo, pokud nebylo změněno je „changeit“. Následně jsme vyzváni zadat jméno a příjmení, což je nešťastně definované. Je tím myšleno Canonical Name (CN) tedy název hostitele nebo počítače. Příkladem pro www.vutbr.cz by bylo CN jméno „vutbr.cz“. Toto je

nejčastější chyba u vlastnoručně podepsaných certifikátů. Jméno domény musí být obsaženo v CN. V našem případě je certifikát určen pro zkoušku na lokálním PC, použijeme tedy CN jméno „localhost“. Pakliže zadáme místo doménového jména IP adresu, pro localhost je to 127.0.0.1, tento certifikát nebude fungovat. Jedinou možností, je pak upravení DNS záznamu v operačním systému ve složce C:\Windows\System32\drivers\etc a přidáním IP adresy a doménového jména do souboru hosts. Tím je dosaženo toho, že je možno použit libovolné doménové jméno například vutbr a přesměrovat jej na adresu 127.0.0.1. Je nutné mít na paměti, že takto upravený DNS záznam funguje pouze na daném PC a neovlivňuje DNS jméno v síti.

Námi vytvořený certifikát je následně přidán do složky s Apache Tomcatem. V souboru server.xml je nadefinována cesta „conf/myCertificate.cer“ a heslo. Po spuštění Tomcatu se tento soubor načte a je připraven k použití.

V prohlížeči je možno si ověřit funkčnost certifikátu. Prohlížeč zobrazí varování o tom, že certifikát není správně podepsán a ověřen CA.



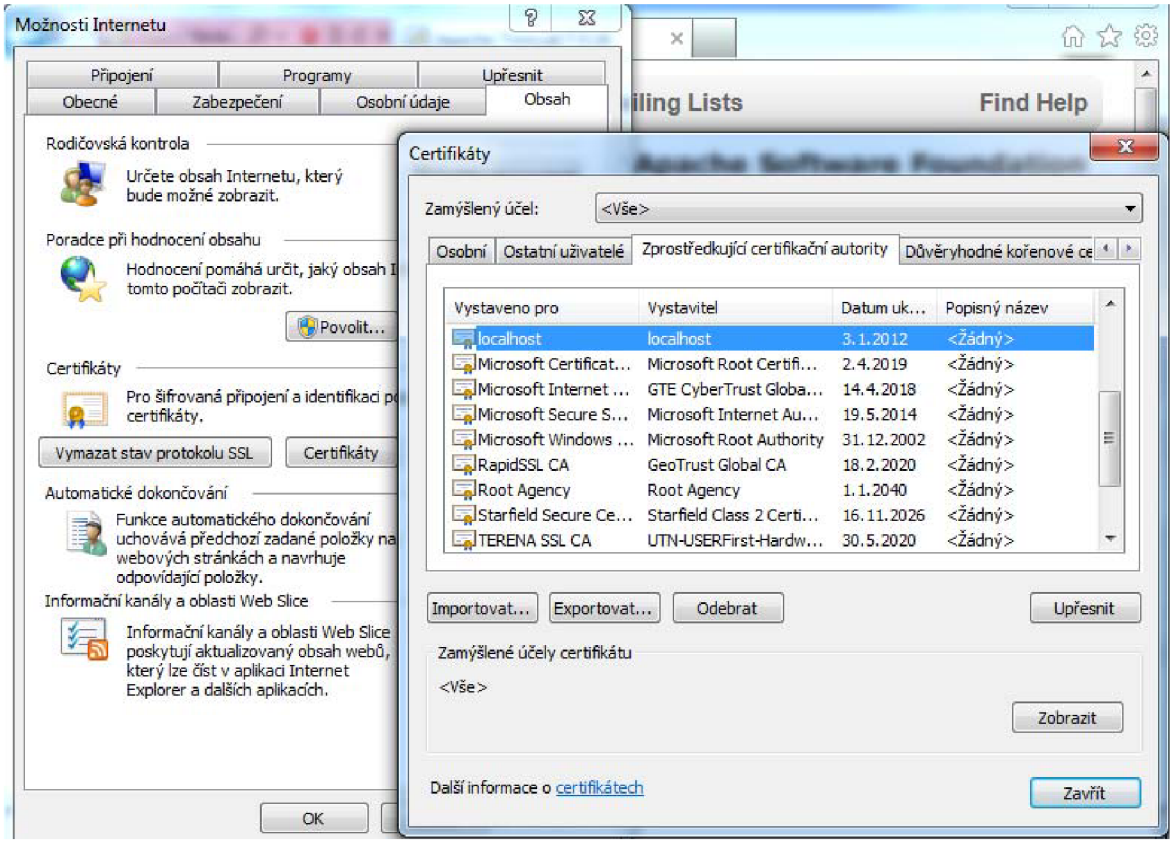
Obr. 3.5: Zobrazení informací o certifikátu

Z obr. 3.5 můžeme vyčíst informace o použitém certifikátu. Ten je ve verzi SSL V3, který také odpovídá protokolu TLS V1. Dále můžeme odečíst jedinečné sériové číslo, algoritmus CE, informace o vystavovateli, použitý veřejný klíč DSA o délce 1024 Bitů a blokovou 128 bitovou šifru AES. Tento vlastnoručně vytvořený certifikát poskytuje dostatečnou úroveň zabezpečení. Je zde možnost použít místo klíče DSA klíč RSA, který podporuje maximální délku 2048 Bitů, má rychlejší ověřování, ale oproti

DSA pomalejší generaci podpisu. Pro použití RSA je nutné přidat k parametrům programu keytool „-keysize 2048 a -keyalg RSA“.

Aby mohla probíhat komunikace s klientskou aplikací, je potřebné získat veřejný klíč. Ten lze opět získat použitím aplikace keytool s parametry „-export -class -file“. Tento postup se však neosvědčil, proto byla použita druhá metoda, která je složitější, ale spolehlivější.

Nejdříve je nutné zapnout Internet Explorer, ten jako jediný podporuje instalaci certifikátů. Klikneme na certifikát, vybereme záložku obecné a zvolíme nainstalovat certifikát. Zobrazí se instalační program, všechny hodnoty necháme nastaveny na výchozí a dáme dokončit. Tím byl nainstalován veřejný klíč, který nyní exportujeme. Kliknutím na možnosti Internetu, záložku Obsah a zvolením položky certifikáty.



Obr. 3.6: Exportování certifikátu

Vybereme požadovaný certifikát a exportujeme jej. Zobrazí se průvodce ve kterém ponecháme zvolený výchozí formát Binarní X.509, kódování DER, pojmenujeme klíč a vyexportujeme. U takto vyexportovaného veřejného klíče je jisté, že je to ten, který používá Apache Tomcat.

3.2.2. Klientská část

Než bude blíže rozebrána klientská aplikace, je nutné dokončit instalaci certifikátů, která byla započata v části pojednávající o serveru.

Vytvořený klíč je nutnou součástí aplikace, aby mohla probíhat zabezpečená komunikace, je nutné na klientském zařízení tento klíč nainstalovat. K tomu opět poslouží program keytool, tentokrát s parametry „-import -alias -keystore -file“ a zadáním hesla k úložišti certifikátů. Protože tato aplikace je určená do nemocnic, je přepokládáno, že v nemocnici budou počítače připojeny na lokální server a každý uživatel si stáhne po přihlášení svůj profil. Proto uživatel nebude muset takto instalovat veřejný klíč. Jeho instalaci zařídí lokální správce sítě. Tento postup je v rámci zabezpečení výhodný. Pokud by uživatel nebo jakákoliv jiná neznalá neoprávněná osoba vlastnila program, nebude schopna se přihlásit do aplikace, ani se znalostí jména a hesla. Dále je možno vytvářet a měnit klíče pro jednotlivé účastníky, tím je docíleno separování jednotlivců a v případě změny klíče není nutná reinstalace.

Aplikace je vyvíjena v jazyce Java s použitím programu Eclipse. Do tohoto vývojového programu existuje velmi jednoduchý doplněk s názvem Keytool [11], pomocí kterého lze jednoduše přidávat a odebírat certifikáty z úložiště. Tento program ulehčí správu certifikátu pro programátora.

Pokud nebude možné instalovat klíč do úložiště certifikátů, je možné přidat do aplikace cestu k veřejnému klíči a to následovně. Do aplikace k podprogramu LoginDialog přidáme řádek:

```
System.setProperty("javax.net.ssl.trustStore",
 "C:/workspace/Client/myCertificate.cer");
```

Tím určíme cestu k externímu klíči, který můžeme být uložen např. v nainstalované složce.

Vývoj je rozdělen na dvě části, na klientskou aplikaci a na aplikaci běžící na serveru. Pro samotnou komunikaci je nejdůležitější, aby obě obsahovaly knihovnu Hessian. Výsledná aplikace funguje na principu tenkého klienta, z čehož vyplývá, že uživatel pomocí aplikace inicializuje spojení se serverem a vzdáleně zpracovává požadavky.

Aplikace pro komunikaci se serverem využívá tří konektorů loginService, dataService a projectService, kde jejich nastavení komunikace je definováno v app.xml následovně:

```
<bean id="loginService" class="org.springframework.remoting.caucho.
     .HessianProxyFactoryBean">
 <property name="serviceUrl"
      value="https://localhost:8443/hospital/remoting/LoginService"/>
 <property name="serviceInterface"
      value="cz.vutbr.feec.hospital.app.ILoginService"/>
</bean>
```

```
<bean id="dataService" class="org.springframework.remoting.caucho.
     .HessianProxyFactoryBean">
 <property name="serviceUrl"
     value="https://localhost:8443/hospital/remoting/DataService"/>
 <property name="serviceInterface"
     value="cz.vutbr.feec.hospital.app.IDataService"/>
</bean>

<bean id="projectService" class="org.springframework.remoting.caucho.
     .HessianProxyFactoryBean">
 <property name="serviceUrl"
    value="https://localhost:8443/hospital/remoting/ProjectService"/>
 <property name="serviceInterface"
     value="cz.vutbr.feec.hospital.app.IProjectService" />
</bean>
```

Jak je ze zdrojového kódu patrné, aplikace se připojuje na adresu: *https://localhost:8443/hospital*

Po bližším prozkoumání zjistíme, že využívá protokol HTTPS na portu 8443 a jako adresa je zvolena doména, která má shodné jméno jako CN ve veřejném klíči.

Při tvorbě byla také ověřena funkčnost aplikace bez použití zabezpečení, tedy HTTP na portu 8080. To dovolovalo použití IP adresy, třeba 127.0.0.1, místo doménového jména, což při zabezpečené komunikaci použít nelze.

V případě, že proxy klienta, využívající Hessian, není schopné kontaktovat server je zobrazeno chybové hlášení o nekompatibilním protokolovém problému. Pakliže není problém způsoben Hessianem, ale například poruchou sítě nebo pádem serveru je zobrazena informace, že server neodpovídá.

Pro implementaci aplikace pro serverovou část do Apache Tomcat slouží build.xml, který při spuštění vytvoří soubor hospitál.war a umístí jej do složky \\tomcat\webapps\. Po spuštění Tomcat rozbalí a načte soubor hospitál.war.

4 IMPLEMENTACE APLIKACE PRO VZDÁLENOU KOMUNIKACI PŘES PROTOKOL HESSIAN

4.1. Zaměření aplikace

V předchozí kapitole bylo popsáno vytvoření zabezpečení mezi klientem a serverem, které vytváří základ pro aplikaci využívající protokol Hessian. Cílem této aplikace je demonstrace protokolu Hessian, který umožňuje lokální volání objektů, které se nacházejí v jiné síti a jiném zařízení. Jeho bezproblémový průchod skrze firewally, díky jeho binární podobě a přenosu po běžném zabezpečeném portu 443.

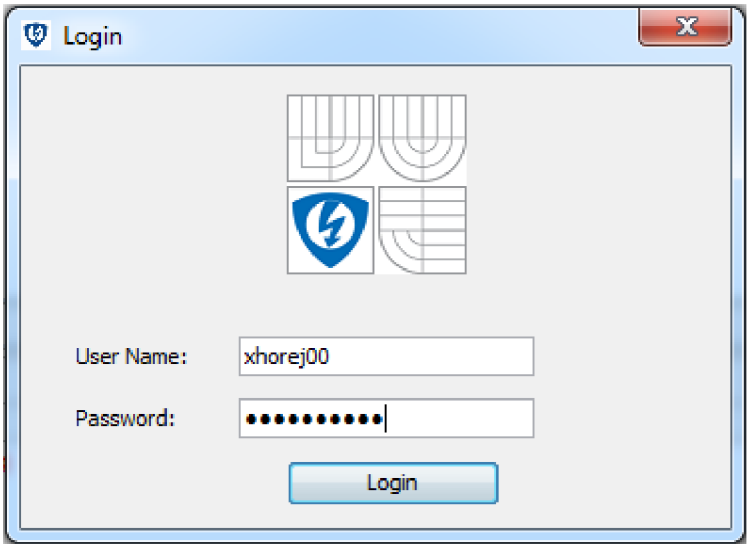
Z tohoto předpokladu byla navržena aplikace tenkého klienta, která načte z lokálního disku obraz, který je zaslán na server, kde je zpracován a následně navrácen uživateli, aniž by zatížil výpočetní jednotku uživatele nebo jeho datový prostor.

Tato aplikace bude později upravena a implementována do většího projektu, který se zabývá zpracováváním lékařských dat, která usnadní a zpřesní práci lékařům.

4.2. Popis uživatelského prostředí a funkcí

Pro vytvoření aplikace bylo opět využito vývojového programu Eclipse za použití programovacího jazyku JAVA.

Po spuštění programu pojmenovaného „Negativ app“ se zobrazí úvodní obrazovka, která je určená k ověření přístupu autorizovaných uživatelů k citlivým lékařským datům.



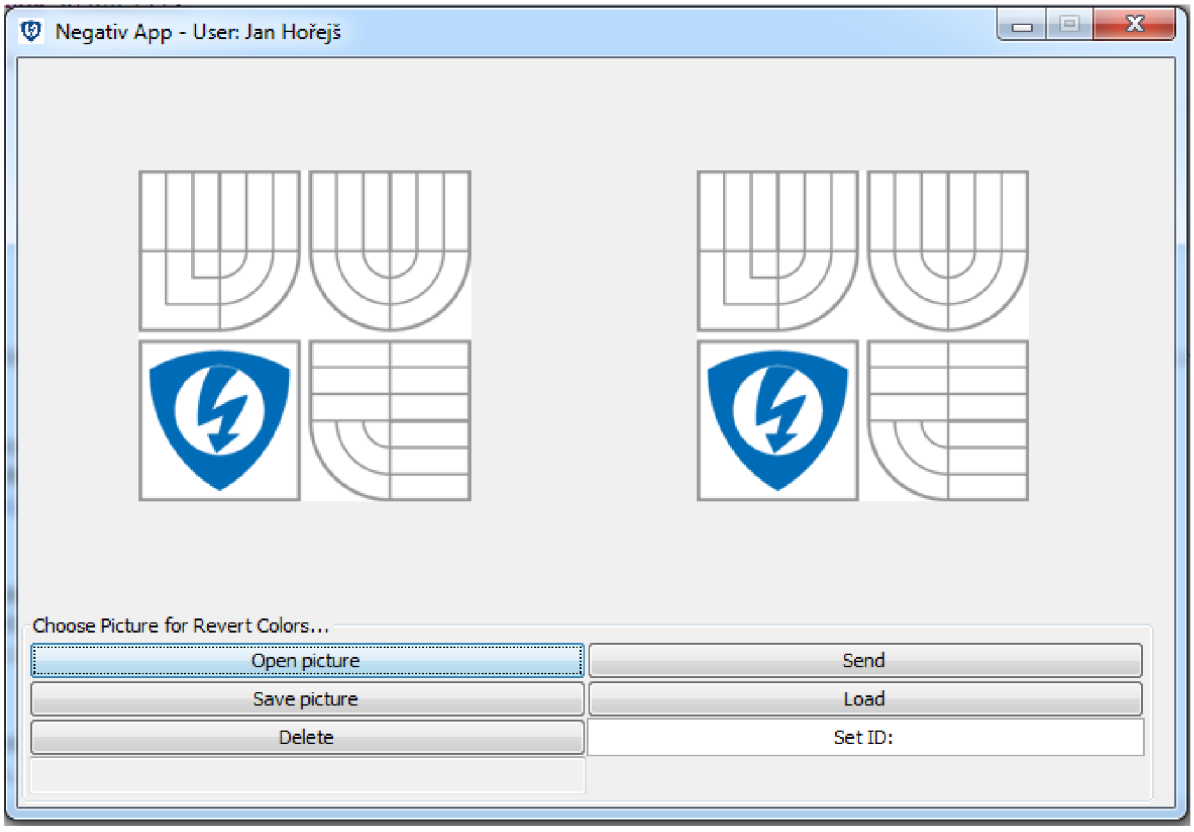
Obr. 4.1: Přihlašovací dialog

Heslo pro aplikaci jsou následující:

User Name: xhorej00

Password: 123456

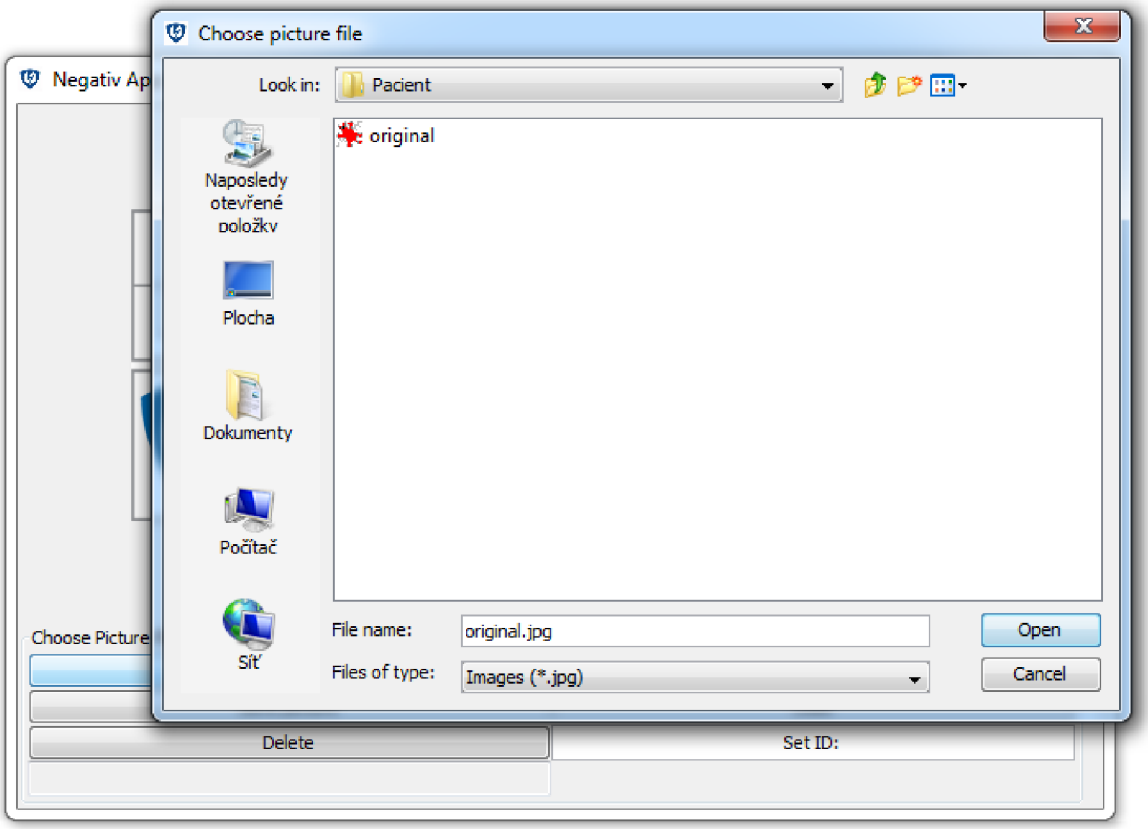
Ověření uživatele probíhá zadáním uživatelského jména a hesla, která jsou šifrovaně odeslána na server, kde jsou porovnána s uživatelskou databází. Pokud je přistupující uživatel ověřen, je navrácena hodnota session, která identifikuje uživatele po celou dobu, co je přihlášen. Při dalším přihlášení je tato hodnota znovu vygenerována a je platná po dobu přihlášení. Tím docílíme lepšího zabezpečení, i když se podaří útočníkovi tuto identifikační hodnotu získat. Po úspěšném přihlášení se nám zobrazí hlavní okno programu.



Obr.4.2: Hlavní okno programu

Hlavní okno se skládá ze dvou polí určených k zobrazení originálního a zpracovaného obrazu. Dále z ovládacích tlačítek, pole pro zadání čísla zpracovávaného případu (ID) a informačního pole. V horní části vedle názvu aplikace se zobrazuje jméno aktuálně přihlášeného uživatele.

4.2.1. Princip a funkce tlačítka „Open picture“



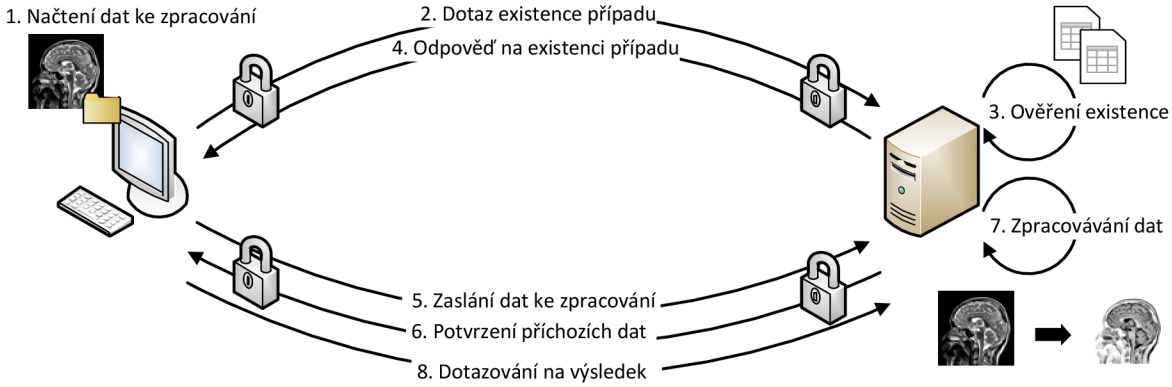
Obr. 4.3: Načtení obrazu pro zpracování

Po zvolení tlačítka „Open picture“ program otevře průzkumníka souborů, pomocí něhož je možné vybrat cestu ke zdrojovému obrázku, se kterým chceme nadále pracovat. Po načtení obrázku se vykreslí do levého pole a v informačním poli se zobrazí zdrojová cesta tohoto souboru.

4.2.2. Princip a funkce tlačítka „Send“

Tlačítko „Send“ zastává v aplikaci klíčovou funkci. Slouží k úpravě dat před odesláním, samotnému přenosu, inicializaci zpracovávání na vzdáleném serveru a kontrole postupu zpracovávání serverem.

Pokud nebyl vybrán pomocí předchozího tlačítka obraz ke zpracování, jsme k tomu pomocí stavové zprávy vyzváni, dále je také nutné zadat číslo zpracovávaného případu (ID), které slouží k identifikaci jednotlivých případů. Pokud byl vybrán obraz ke zpracování a zadáno ID, program zahájí odesílání souboru. Princip tlačítka je následující:



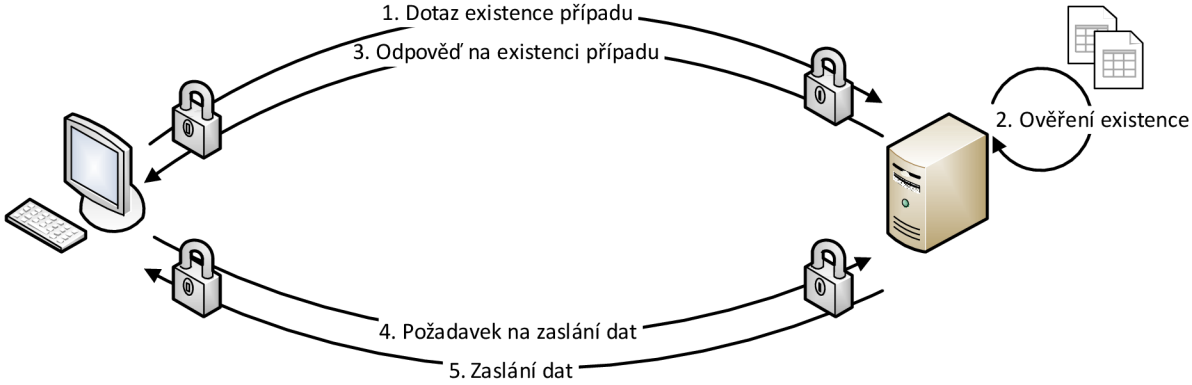
Obr. 4.4: Průběh komunikace tlačítka „Send“

V prvním kroku dojde ke kontrole načtených dat před zahájením odesílání. Následně je zasláno ID případu na server, kde je porovnáno s databází. Pokud již tento případ existuje, jsme vyzváni k zadání nového ID. Jestliže server odpoví, že tento případ neexistuje, může aplikace dále pokračovat a to zasláním načtených dat na server. Po ukončení přenosu jsme informováni ve stavové liště o úspěšném přenosu. Mezi tím dochází k inicializaci serveru, který začne zpracovávat příchozí data a podprogramu aplikace, který se začne v určitém časovém intervalu dotazovat, jestli je případ serverem zpracován.

V této demonstrační aplikaci je pod pojmem zpracování, použit kód určený k vytvoření negativu ze zaslaného obrazu a časového zpoždění, které simuluje složité časově náročné výpočty. Tuto funkci je později možné nahradit libovolnou metodou pro analýzu a zpracování obrazu, která může pomoci při identifikaci nádorů a jiných objektů ze zaslaných snímků.

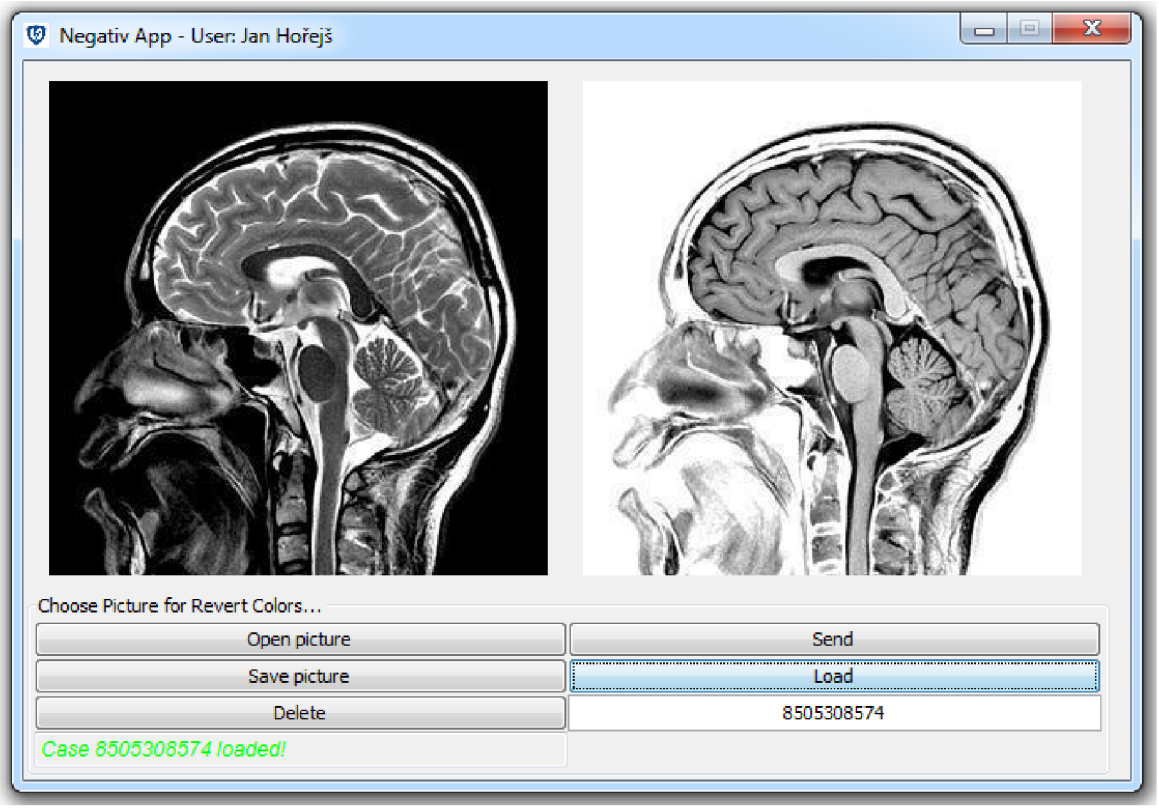
Pokud přenesení a zpracování proběhlo úspěšně, dojde k zápisu projektu do databáze serveru. Program klienta po tomto zjištění oznámí, že tento projekt je zpracován a je připraven k načtení. Toto řešení představuje jeden z hlavních přínosů celé práce, neboť jsou zde zabezpečeně přenesena data skrze komunikační síť a využito vzdáleného volání funkcí. Další výhodou tohoto řešení je, že i v případě pádu či ukončení aplikace jsou data nadále zpracovávána a následně připravena k opětovnému načtení.

4.2.3. Princip a funkce tlačítka „Load“



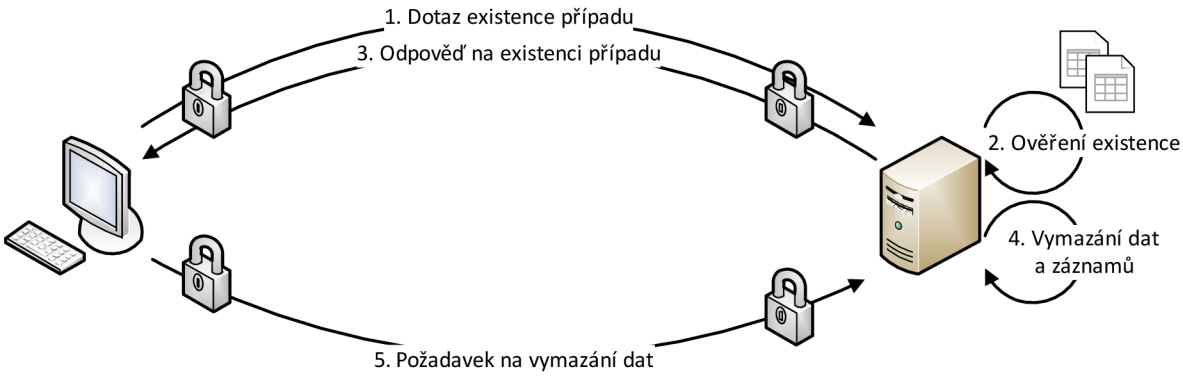
Obr. 4.5: Průběh komunikace tlačítka „Load“

V případě požadavku k načtení právě zpracovaného projektu nebo načtení jiného, bude k tomuto účelu použito tlačítko „Load“, které v první části ověří, zdali případ existuje na serveru. Pokud ne, aplikace nás informuje stejně, jako u předchozího tlačítka o této informaci, ve stavové liště. V případě existence si klientská aplikace tato data vyžádá. Server odešle jak původní přijatý obraz, tak i zpracovaný. Díky tomu je možné obrazy, například CT mozku, zároveň porovnávat. Obrazy jsou následovně uloženy do dočasné složky a zobrazeny v příslušných polích v hlavním okně programu.



Obr. 4.6: Hlavní obrazovka po načtení projektu ze serveru

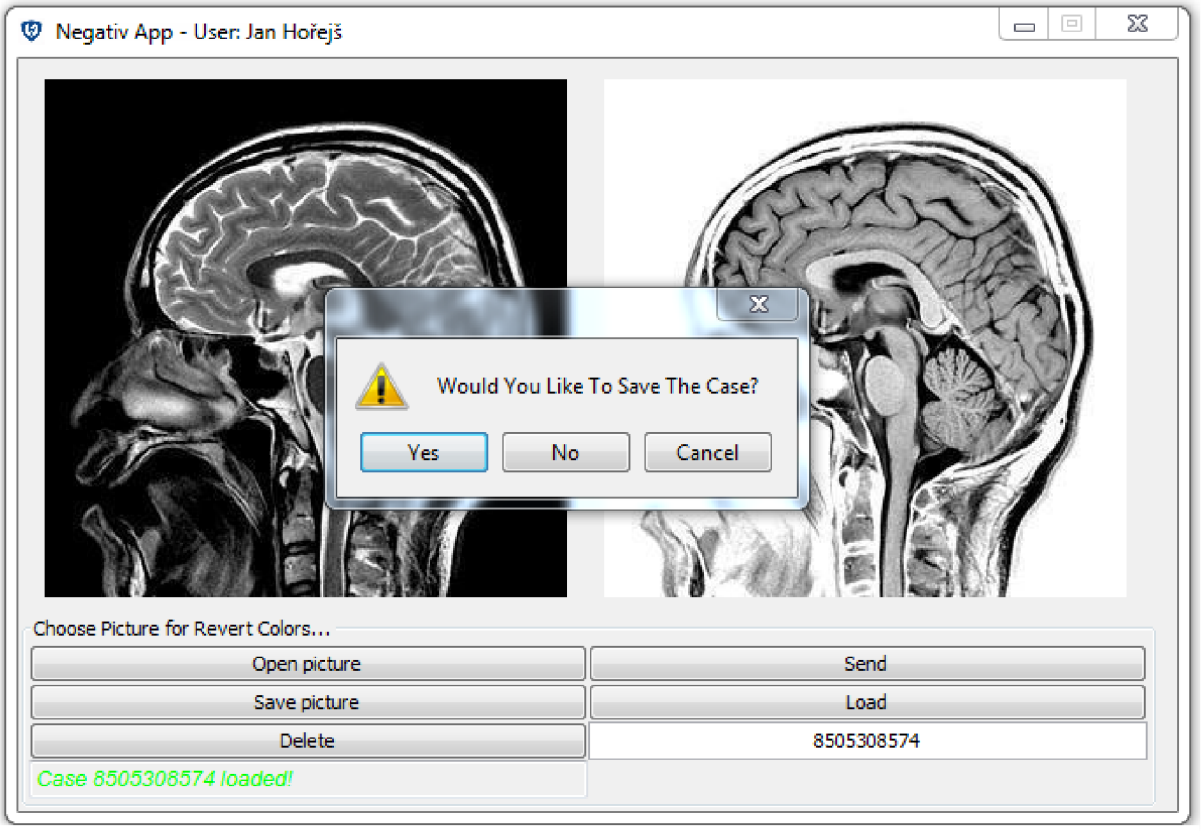
4.2.4. Princip a funkce tlačítka „Delete“



Obr. 4.7: Průběh komunikace tlačítka „Delete“

Účel tohoto tlačítka je k odstranění určitého případu ze serveru a jeho databáze. Ověření existence je shodné jako v předchozích případech. Následně je spuštěna na serveru funkce k odstranění souborů a záznamů. V uživatelské aplikaci je zobrazeno vyskakovací okno, které se dotáže před odstraněním k potvrzení příkazu.

4.2.5. Princip a funkce tlačítka „Save Picture“ a tlačítka „Exit“



Obr. 4.8: Dotaz na uložení projektu

Pokud byl načten případ ze serveru a chceme jej uložit, využijeme k tomu tlačítko pro uložení, které dočasně uložený zpracovaný obraz pomocí průzkumníka, na zvolené místo uloží.

Na stejném principu funguje také tlačítko „Exit“ v pravé horní části okna. V případě zmáčknutí tlačítka se aplikace dotáže, jestli opravdu chceme ukončit aplikaci a případně zpracovaný neuložený projekt uložit. Po následovném potvrzení ukončení, nebo uložení, je program ukončen a dočasná složka se soubory je vymazána.

5 ZÁVĚR

Tato práce se zabývala zabezpečením komunikace mezi aplikací pro lékaře a serverem. Jako aplikační server byl zvolen Apache Tomcat, který vyniká jednoduchostí, účelností, nízkými systémovými požadavky a má otevřený zdrojový kód. Dále byl vybrán binární metaprotokol Hessian pro své přednosti. Ty spočívající v jednoduchosti, multiplatformnosti, využití pro vzdálené volání procedur a bezproblémového průchodu skrze zabezpečení různých sítí.

Aby bylo možné tuto komunikaci uskutečnit, byla realizována instalace a zprovoznění Apache Tomcat ve kterém bylo demonstrováno jak nastavit konektory pro komunikaci, přesměrování a instalaci certifikátu pro protokol HTTPS. Tento certifikát byl vytvořen pomocí aplikace keytool a následně implementován jak do klientské aplikace a serverové aplikace Tomcat, tak i v případně potřeby do operačního systému. V klientské aplikaci byl použit protokol Hessian. Pro jeho potřeby byly vytvořeny konektory pro směrování komunikace na server a byl nainstalován potřebný veřejný klíč.

Dále byla vytvořena aplikace „Negativ app“, která provádí jednoduchou operaci v podobě invertování barev, kterou bude v budoucnu možné nahradit libovolnou metodou pro analýzu, zpracování obrazu a která slouží k ověřenému vzdálenému přístupu na server. Za pomoci protokolu Hessian bylo možné volat vzdálené funkce serveru, což nám umožnilo zabezpečené přenášení dat od tenkého klienta na server, kde tato data jsou zpracována a následně navrácena uživateli, aniž by bylo nutné použít jeho lokálních systémových prostředků.

Hlavní přínos této práce spočívá v zabezpečení přenosů citlivých lékařských dat mezi klientskou aplikací a vzdáleným serverem, který bude tato data zpracovávat a výsledky předávat skrze internetovou síť zpět k lékaři, tak aby byla tato data chráněna před neoprávněným zásahem z venčí. Tato data jsou zabezpečena protokolem HTTPS využívající 128 bitové šifrování. Dalším kladem této práce je vytvoření aplikace, která se stane součástí většího lékařského projektu určeného k odhalovaní tumorů a jiných objektů ze zaslaných dat. Práce také mapuje různé metaprotokoly a vybírá mezi nimi ty nejvhodnější pro různé účely. Také bylo popsáno vytváření a implementování vlastního certifikátu jak do klientské, tak i serverové aplikace.

LITERATURA

[1] *Apache Tomcat* [online]. c2009 [cit. 2011-12-04]. SSL Configuration HOW-TO. Dostupné z WWW: <http://tomcat.apache.org/tomcat-4.1-doc/ssl-howto.html>.

[2] ČR. Zákon o ochraně osobních údajů a o změně některých zákonů. In *Sbírka 101/2000*. 2000, 32, s. 1521-1532. Dostupný také z WWW: <http://aplikace.mvcr.cz/sbirka-zakonu/ViewFile.aspx?type=c&id=3420>.

[3] FLANAGAN, David. *Java Foundation Classes in A Nutshell* [online]. Sebastopol: O'Reilly Media, Inc., 1999 [cit. 2012-04-15]. ISBN 1565924886. Dostupné z: <http://docstore.mik.ua/orelly/java-ent/jfc/index.htm>

[4] HANSON, Jeff. *Java World : Solutions for Java developers* [online]. 22.1.2008 [cit. 2011-12-04]. Is Tomcat enterprise ready?. Dostupné z WWW: <http://www.javaworld.com/javaworld/jw-01-2008/jw-01-tomcat6.html>.

[5] HARKEY, Dan; ORFALI, Robert; EDWARDS, Jeri. *Client/Server Survival Guide*. 3rd ed. Canada : Wiley, 1999. 800 s. ISBN 0471316156.

[6] *Hessian binary web service protocol* [online]. c2011 [cit. 2011-12-04]. Metaprotocol Taxonomy. Dostupné z WWW: <http://hessian.caucho.com/doc/metaprotocol-taxonomy.xtp> .

[7] How to Make Dialogs. In: *Oracle Documentation* [online]. © 1995-2012 [cit. 2012-04-15]. Dostupné z: <http://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html>

[8] How to Use File Choosers. In: *Oracle Documentation* [online]. © 1995-2012 [cit. 2012-04-15]. Dostupné z: <http://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html>

[9] HUMPOLÍK, Jan. *Zabezpečený přístup pro webové aplikace* [online]. Brno : Vysoké učení technické v Brně (VUT), 2010. 56 s. Bakalářská práce. Vysoké učení technické v Brně (VUT). Dostupné z WWW: <https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=30757>.

[10] Java Image Processing : Negative of Input Image. In: TIWARI, Ganesh. *GT's Blog* [online]. [June 2011] [cit. 2012-04-15]. Dostupné z: <http://ganeshtiwaridotcomdotnp.blogspot.com/2011/06/java-image-processing-negative-of-input.html>

[11] Keytool plugin for Eclipse. FUST, Patrick. [online]. [cit. 2012-05-12]. Dostupné z: <http://keytool.sourceforge.net/>

[12] KRZYZANOWSKI, Paul. *Lectures on distributed systems* [online]. [s.l.] : Rutgers University, 2003-02-10 [cit. 2011-12-04]. Remote Procedure Calls, s. . Dostupné z WWW: <http://www.cs.rutgers.edu/~pxk/rutgers/notes/content/04-rpc.pdf>.

[13] MOCZAR, Lajos. *Tomcat 5 unleashed*. Indianapolis, Ind. : SAMS, 2005., 742 s., ISBN 0-672-32636-1.

[14] *MSDN Microsoft* [online]. 7.9.2011 [cit. 2011-12-04]. How RPC Works. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/windows/desktop/aa373935(v=vs.85).aspx>.

[15] RESCORLA, Eric. *SSL and TLS : Designing and Building Secure Systems*. 1. Boston : Addison-Wesley Professional, 2000. 528 s. ISBN 0201615983.

[16] Za krádeží a únikem citlivých informací stojí nejčastěji zaměstnanci. *ICT Security*: *Nezávislý odborný on-line magazín* [online]. 08.07.2011[cit. 2012-04-15]. Dostupné z: <http://www.ictsecurity.cz/security-bezpenost/za-kradei-a-unikem-citlivych-informaci-stoji-nejastji-zamstnanci.html>

SEZNAM ZKRATEK

AES Advanced Encryption Standart – Symetrická bloková šifra

API Application Programming Interface Rozhraní pro programování

C# Programovací jazyk

CA Certificate Authority – Certifikační autorita

CN Canonical Name – Hostitelské jméno hosta, počítače nebo serveru

CORBA Common Object Request Broker Architecture – umožňuje běh na více PC

CT Computed Tomography – způsob rentgenového vyšetření

dDOS distributed Denial of Service – Distribuované odmítnutí služby, internetový útok

DER Distinguished Encoding Rules – Šifrovací pravidla X.509

DES Data Encryption Standart – Symetrická šifra

DOM Document Object Model – API, které umožňuje měnit styly, obsah a strukturu dokumentů XML a HTML

DSA Digital Signature Algorithm – Standard americké vlády pro digitální podpis

FIFO First In First Out – Řazení dat - první přijde, první odejde

FTP File Transfer Protocol – Internetový protokol pro přenos souborů

HTML Hyper Text Markup Language – program. Jazyk pro vytváření WWW stránek

HTTP Hypertext Transfer Protocol – Internetový protokol

HTTPD Hypertext Transfer Protocol Daemon – první webový server

HTTPS Hypertext Transfer Protocol Secure – Nadstavba HTTP

ID Identity Document – označení pro identifikaci osob či věcí

IDEA International Data Encryption Algorithm – Mezinárodní algoritmus pro šifrování dat využívající symetrickou blokovou šifru

IDL Interface Description Language – Definuj rozhraní pro komunikaci různých programovacích jazyků

JCP Java Community Process – Sdružení firem, které mohou ovlivnit vývoj platformy Java

JEE Java Enterprise Edition – Platforma pro serverové programování v Javě

JMS Java Messaging Services – API pro firemní software

JSF Java Server Faces – Vývojáři definované uživatelské rozhrání

JSP Java Server Pages – Umožňuje dynamické generování obsahu stránek

MD2 Message Digest algorithm – Hashovací funkce vytvářející otisk vstupních dat, 8 bitová

MD4 Message Digest algorithm – Nástupce MD2, 128 bitová

MD5 Message Digest algorithm – Nástupce MD4

NFS Network File System – Internetový protokol pro vzdálený přístup

ONC-RPC Open Network Computing RPC – Část NFS projektu

P2P Peer to Peer – Architektura klient-klient

PHP Personal Home Page – Skriptovací programovací jazyk

POX Plain Old XML – Základní XML

REST Representational State Transfer – Protokol pro mnohonásobné čtení

RMI Remote Method Ivocation – Java API pro běh RPC

RPC Remote Procedure Call – Vzdálené volání procedur

RSA Rivest, Shamir, Adleman – Šifra s veřejným klíčem

RSS RDF Site Summary – Čtení novinek na webových stránkách z XML

SHA Secure Hash Algorithm – Hashovací funkce vytvářející otisk vstupních dat

SMTP Simple Mail Transfer Protocol – Internetový protokol pro přenos el. pošty

SOA Service Orientyd Architecture – Sada pravidel a principů pro tvorbu aplikací

SOAP Simple Object Access Protocol – Protokol pro výměnu zpráv založených na XML

SSL Secure Sockets Layer – Vrstva zabezpečení mezi transportní a aplikační vrstvou

TCP Transmission Control Protocol – Spojová potvrzovaná služba

TLS Transport Layer Security – kryptografický protokol, nástupce SSL

UDP User Datagram Protocol – Nepotvrzovaná nespojová služba

WSDL Web Services Description Language – Popisuje způsob a funkce webové služby

WWW World Wide Web – Aplikace internetového protokolu propojující dokumenty

XDR External Data Representation format – Změna dat pro kompatibilní přenos

XML Extensible Markup Language – Rozšířitelný značkovací jazyk

OBSAH PŘILOŽENÉHO DVD

\\Bakalářská práce\	- Složka s elektronickou verzí práce
\\eclipse\	- Vývojové prostředí Eclipe s doplňkem Keytool
\\workspace\	- Složka se zdrojovými kódy
\\workspace\tomcat\	- Složka s aplikačním serverem Apache Tomcat
\\ReadMe.pdf	- Soubor pro orientaci v přiložených materiálech