Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Klient-server aplikace využívající REST API

Miloš Havránek

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Miloš Havránek

Informatika

Název práce

Klient-server aplikace využívající REST API

Název anglicky

Client-server application using REST API

Cíle práce

Bakalářská práce se zabývá návrhem, vývojem a implementací REST API. Hlavním cílem je vytvořit server a klient využívající RESTful služby jako prostředek pro komunikaci. Dílčím cílem bude popsat použité technologie a postupy.

Metodika

Při zpracování teoretických východisek práce bude využito informací získaných studiem odborné literatury.

Praktická část práce bude spočívat v návrhu a implementaci serverové a klientské aplikace využivající RESTful API. K tvorbě aplikací bude využit vybraný programovací jazyk a související technologie. Při tvorbě budou použity standardní metody a nástroje softwarového inženýrství.

Doporučený rozsah práce

35-40 stran

Klíčová slova

Java EE, Rest API, Tomcat, Klient vs. Server

Doporučené zdroje informací

HEROUT, P. Učebnice jazyka Java. České Budějovice: Kopp, páté vydání, 2010. ISBN ISBN 978-80-7232-398-2.
Jersey [online]. [cit. 2016-06-12]. Dostupné z: https://jersey.java.net/documentation/latest/index.html
Oracle [online]. [cit. 2016-06-12]. Dostupné z: http://www.oracle.com/technetwork/java/index.html
SCHILDT, H. Java 8. Brno: Computer Press, 2016. ISBN 978-80-251-4665-1

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 12. 11. 2016

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Klient-server aplikace využívající REST API" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15. března 2017 _____

Podpis autora práce

Poděkování

Rád bych touto cestou poděkoval:

- panu Ing. Jiřímu Brožkovi, Ph.D., jakožto vedoucímu mé bakalářské práce za pevnou ruku a trpělivost při vedení této práce
- panu Ing. Luboši Klímovi za poskytnuté znalosti ohledně zabezpečení
- panu Martinu Hamplovi za pozitivní a kolegiální přístup
- firmě Neit Consulting s.r.o. za praxi a znalosti potřebné k dokončení této práce

Klient-server aplikace využívající REST API

Souhrn

Bakalářská práce je zaměřena na komunikační rozhraní mezi klientskou a serverovou aplikací využívající technologie REST API. V bakalářské práci jsou vymezeny základní pojmy z oblasti programovacího jazyka Java a jeho rozšíření Java EE ve formě balíčků běžící na aplikačním serveru Tomcat a šifrování samotné komunikace. Práce se zabývá samotným komunikačním rozhraním a komunikací.

Klíčová slova: Java EE, REST API, Tomcat, Klient vs. Server

Client-server application using REST API

Summary

This Bachelor's Thesis focuses on communication interface between client and server application via REST API specification. In Bachelor's thesis are determined basic notions from Java programming language and its extension Java EE in form of packages running on Tomcat application server and traffic encryption.

Keywords: Java EE, REST API, Tomcat, Client vs. Server

Slovník použitých zkratek a pojmů

3rd party	Třetí strana
API (Application Programming Interface)	Abstrakce aplikačního rozhraní pro programování aplikací
Asymetrická šifra	Zpráva je zašifrována veřejným klíčem a dešifrována privátním klíčem.
C (Country)	Kód státu
CA (Certificate Authority)	Certifikační autorita
Certifikát	Obsahuje klíč a dodatečné informace
CN (Common Name)	Identifikátor / Doménové jméno
Doména	Doménové jméno
E / EMAIL (Email)	Email
Hash	Výstup hashovacího algoritmu
Hashovací algoritmus	Vytvoření jedinečného otisku vstupních dat, kdy nelze reverzním inženýrstvím získat data původní.
HTML (HyperText Markup Language)	Značkovací jazyk používaný pro tvorbu webových stránek
IDE (Integrated Development Environment)	Softwarové vývojové prostředí
Keypair (Priv_Pub)	Pár klíčů (veřejný a privátní klíč)
KeyStore	Úložiště vlastních keypairů
Klient	Počítač připojující se na server s různými požadavky pro zpracovaní nebo poskytnutí dat.
Kryptografie	Věda zabývající se šifrováním zpráv.
L (Location)	Místo (město)
O (Organization)	Organizace (firma)
OU (Organization Unit)	Oddělení / Dodatečný popis
Private key	Privátní klíč
Public key	Veřejný klíč
REST (Representational State Transfer)	Architektura rozhraní pro distribuované prostředí
RSA	Asymetrická šifra s veřejným klíčem
SAN (Subject Alternative Name)	Alternativní jméno objektu
Server	Vzdálený počítač zpracovávající požadavky klienta, poskytující a zpracovávající data, tvoří hlavní logiku celé aplikace.
Session	Relace připojení
SHA-512 (Secure Hash Algorithm)	Hashovací algoritmus s délkou 512 bitů

SSH (Secure Shell)	Zabezpečený komunikační protokol, pro navázání zabezpečeného připojení v nedůvěryhodné síti (internet).
ST (State)	Stát
Symetrická šifra	Vyžaduje stejný klíč pro šifrování i dešifrování zprávy.
TCP	Protokol transportní vrstvy TCP/IP, garantovaný přenos
TCP/IP (Transmission Control Protocol/Internet Protocol)	Primární přenosový protokol / protokol síťové vrstvy
TrustStore	Úložiště formátu KeyStore tvořené důvěryhodnými veřejnými certifikáty (CA/Self-signed)
UDP	Protokol transportní vrstvy IP, negarantovaný přenos

OBSAH

1 Úvod

Java je jedním z nejpoužívanějších a nejlépe ohodnocených programovacích jazyků roku 2017, nejvíce využití nachází v Android a Enterprise aplikacích. Mezi klíčové znalosti a schopnosti vývojáře patří nejen analytické myšlení, schopnost programovat a znalost samotného programovacího jazyka, ale i znalost ostatních technologií, platforem i jiných programovacích jazyků.

V této bakalářské práci je popsáno a naprogramováno jednoduché komunikační rozhraní mezi klientem a serverem využívající webových technologií, konkrétně REST API pomocí HTTP. Základem serveru je Java EE Web Application běžící na Tomcatu od Apache, využívající Jersey (JAX-RS) pro RESTové služby. Klientem je jednoduchá CLI aplikace využívající Jersey pro komunikaci se serverem přes REST.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem této práce je přiblížit současné technologie programovacího jazyka Java a překonat počáteční robustnost, složitost a rozmanitost technologií s ním spojeným. Tato práce sestává z aplikačního serveru Java poskytující REST služby klientské aplikaci. Účelem není obsah zprávy, ale komunikace samotná.

Hlavním cílem je popsat a postavit komunikační rozhraní z pohledu základní funkcionality a následným zabezpečením pomocí HTTPS, tedy SSL/TLS šifrováním komunikace. Vedlejším cílem pak je poskytnout funkční ukázkové řešení s příklady použití pro snadnější pochopení problematiky.

2.2 Metodika práce

Pro vyhotovení práce je sestaven následující postup:

- **studium odborné literatury**

Nedílnou součástí této bakalářské práce je studium teoretických východisek, které objasní základní pojmy v oblasti programování v prostředí jazyka JAVA. Studium odborné literatury probíhá souběžně se zpracováním analytické části bakalářské práce. Tato vymezení jsou stěžejní pro kvalitní zpracování celé bakalářské práce. Teoretická východiska jsou zpracována zejména z literárních zdrojů a z části ze zdrojů internetových. Použité zdroje jsou vyjmenovány v kapitole Seznam použitých zdrojů.

- **vývoj aplikace**

Praktická část pojednává o návrhu, naprogramování, nastavení a sestavení funkčního řešení aplikací s jednoduchými příklady použití za pomoci znalostí získaných ze studia odborné literatury a praktických zkušeností z vývoje software.

- **souhrn dosažených výsledků, diskuse a závěr**

V kapitole Závěr je zhodnoceno naplnění kladeného cíle bakalářské práce.

3 Teoretická východiska

3.1 Jazyk Java

Historie jazyka Java

S prvním vydáním jazyka Java 1.0. přišla společnost Sun Microsystems, Inc. V roce 1995. Vytvoření jazyka Java znamenalo pro programování revoluci, jelikož tento jazyk rozšířil atraktivitu webového prostředí. (Schildt, 2016)

Původním cílem vývoje jazyka Java ještě pod názvem „Green project“ byla potřeba vytvořit programovací jazyk vhodný pro zařízení spotřební elektroniky. James Gosling, který byl vedoucím tohoto projektu vytvořil programovací jazyk Oak. Tento název, jak se později ukázalo, již existoval, a tak vznikla Java. Programy pro elektrické spotřebiče nakonec vůbec nevznikly, za to se však v roce 1993 rozrostla služba World Wide Web mezi širokou veřejnost. (Keogh, 2005)

Verze jazyka Java

V průběhu let jazyk Java prošel mnoha aktualizacemi (edicemi). Od první aktualizace Java 1.1., která přinesla mnoho významných knihovních prvků přes J2SE 5, která zásadně rozšířila možnosti i oblasti uplatnění jazyka Java. Od vydání Java SE 7 byla již Java vlastněna společností Oracle, v jejíž vlastnictví je dodnes. Nejnovější verzí (k r. 2016) je Java SE 8. (Schildt, 2016)

3.1.1 Hlavní atributy jazyka Java

Hlavními požadavky při vývoji jazyka Java byly přenositelnost a bezpečnost, ale při formování výsledné podoby sehrály roli i tyto faktory: (Schildt, 2016)

- **Jednoduchost –** stručná sada funkcí,
- **Bezpečnost,**
- **Přenositelnost –** programy lze spouštět v libovolném prostředí podporující Java Virtual Machine,
- **Objektová orientace,**

- **Robustnost –** pomáhá programovat bez chyb, má typovou kontrolu,
- **Více vláken,**
- **Neutralita vzhledem k architektuře,**
- **Interpretovaný kód –** díky využití bajtového kódu je kód jazyka Java přenositelný mezi platformami,
- **Vysoký výkon,**
- **Distribuované prostředí –** jazyk Java navržen s ohledem na distribuované prostředí internetu,
- **Dynamičnost (Reflexe) –** programy Java obsahují informace o typech runtime, které slouží ke kontrolám a zpřístupnění objektů za běhu.

3.1.2 Vývojářská sada

- **JRE (Java Runtime Environment) –** běhové prostředí Javy,
- **JDK (Java Development Kit – sada pro vývoj v Javě),** někdy označován jako **SDK (Software Development Kit – sada pro vývoj softwarů)** je vlastně řada JRE doplněná o vývojové nástroje. (Pecinovský, 2009)

3.2 JVM (Java Virtual Machine)

Java sama o sobě je nezávislá na platformě, co je na platformě závislé je pak JVM, které jako virtuální stroj slouží k interpretaci byte kódu do strojového jazyka počítače. Dále jsou závislá různá specifika kódu, kdy je potřeba někdy kód upravit pro potřeby jednotlivých operačních systémů. (Schildt, 2016)

3.3 Java SE (Standard Edition)

Java SE obsahuje základní Java Core API a JVM. (Herout, 2010)

Java Core API

U vyšších programovacích jazyků, jako je Java je uživateli umožněno se částečně oprostit od nižšího programování, a to využitím již před naprogramovaných tříd. Java Core

API je tvořeno těmito základními balíčky tříd. V praxi se ale narazí na potřebu psát takové knihovny vlastní. (Herout, 2010)

3.4 Java EE (Enterprise Edition)

Java EE je specifikace rozšíření Javy SE o další technologie, vyžaduje tedy základní Java Core API z Java SE. (Hanel, 2016)

Java EE lze provozovat 2 způsoby: (Hanel, 2016)

- Java EE Complaint application server
  - Obsahuje všechny oficiálně dostupné knihovny Java EE.
  - Lépe optimalizované jako celek
- Java EE Lightweight application server
  - Obsahuje vlastní knihovny pro běh základního prostředí Java SE s podporou Java EE knihoven bez jejich zahrnutí do aplikačního serveru. Java EE knihovny jsou distribuovány se samotným balíčkem programu. Lze tak dosáhnout vyššího výkonu samotného aplikačního serveru a aplikace.
  - Většinou Open Source řešení vyvíjené komunitou.

3.4.1 Java EE technologie

Java EE obsahuje např. tyto technologie: (Čápka, 2014)

- **JSP (Java Server Pages) -** pomocí níž lze vkládat speciální direktivu do HTML kódu, která spustí Java kód,
- **JSF (Java Server Faces)** – jedná se modernější a konkurenční technologii k JSP, webová stránka je reprezentována jako XML soubor, web se skládá z již připravených komponent, které lze plnit daty z Javy,
- **JDBC (Java Database Connectivity) -** JDBC je standardní rozhraní pro práci s různými typy databází v jazyce SQL,
- **JPA (Java Persistence API) -** JPA je rozhraní, které umožňuje objektovou práci s daty, konkrétní implementací je Hibernate,

- **EJB (Enterprise Java Beans**) - komponenty obchodní logiky,
- **JAX-RS (Java API for RESTful Web Services) –** Java API pro RESTové webové služby.

3.5 Maven

Maven je software vyvinutý neziskovou organizací Apache Software Foundation, slouží ke správě, automatizaci buildů aplikací, umožňuje jak přenositelnost projektu mezi jednotlivými IDE, tak i build projektu se samotným Maven. S tím souvisí také stahování potřebných knihoven ze vzdálených repositářů nebo vlastních. (Hordějčuk)

Koncept

Principem systému Maven je vytvoření objektového modelu nad zdrojovým kódem, se kterým lze provádět různé operace (kompilace, kontrola nebo vytvoření balíků). Model projektu je definován v souborech *pom.xml*, které se nachází v kořenovém adresáři každého projektu. Důležitou funkcí systému Maven je řešení závislostí. Není tedy nutné kopírovat knihovny a umisťovat je na classpath. (Hordějčuk)

Projekt

Každý soubor *pom.xml* představuje jeden **projekt** (**artefakt**). Artefakt je jednoznačně identifikován skupinou (groupId), názvem (artifactId) a verzí (version). (Hordějčuk)

Repositář

Repositář systému Maven obsahuje katalog artefaktů a systém pro jejich vyhledávání a stahování. Repositáře jsou **lokální** a **vzdálené**. Lokální repositář je umístěn na lokálním počítači a cachují se v něm použité závislosti. (Hordějčuk)

Závislosti (Dependencies)

Závislostmi jsou myšlené jiné artefakty spravované systémem Maven, které nějaký artefakt vyžaduje ke své kompilaci či funkci. Závislosti mají tzv. *scope*, který specifikuje míru a okamžik potřeby dané závislosti. (Hordějčuk)

Výjimky ze závislostí (Dependency Exclusions)

Pokud projekt obsahuje závislost na dalším artefaktu, může explicitně zakázat některé jeho tranzitivní závislosti. (Hordějčuk)

Soubor pom.xml

Soubor *pom.xml* (POM = project object model) je XML soubor obsahující model jednoho artefaktu (projektu), který obsahuje například závislosti artefaktu a jeho návaznosti na jiné artefakty. (Hordějčuk)

3.6 Intellij IDEA

Intellij IDEA je komerční IDE pro programovací jazyk Java (Community Edition – zdarma) s podporou Java EE (Ultimate Edition – placená) a 3rd party pluginů. Tento software je vyvíjený společností JetBrains a nabízí studentům licence zdarma pro všechny produkty k nekomerčnímu prostředí. IDE obsahuje řadu stejných funkcí, jako konkurence (NetBeans, Eclipse), ale funkce také řešené jinak a mnoho dalších funkcí, které konkurenční řešení nemá. (JetBrains)

IDE se těší popularitě, díky provázanosti a ucelenosti celého řešení, které je po krátkém čase snadné na osvojení.

3.7 Tomcat

Apache Tomcat je aplikační server pro webové aplikace a servlet kontejnery. Jedná se o lightweight application server, open-source software s podporou komunity a spoustou návodů.

Aplikační server ***Tomcat*** využívá dvě implementace SSL. **APR** (Apache Portable Runtime) implementaci, která využívá knihovny ***OpenSSL*** a **JSSE** (Java Secure Socket Extensions) implementaci, což je součást Java runtime. (Bouška, 2014)

3.8 REST (Representational State Transfer)

REST je Architektura rozhraní pro distribuované prostředí. REST jako takový je velice abstraktní, neukládá, jak má samotná implementace RESTu vypadat, ale základní principy, co musí tato architektura splňovat.

REST je orientován datově, nikoli procedurálně. Webové služby definují vzdálené procedury a protokol pro jejich volání, REST určuje, jak se přistupuje k datům. Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (resources). Zdrojem mohou být data, stejně jako stavy aplikace. Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim. (Malý, 2009)

Metody pro přístup k datům: (Katedra informatiky)

- Přístup reprezentovaný zkratkou CRUD:
    - Create (C) – Vytvoření dat
    - Retrieve (R) – Získání požadovaných dat
    - Update (U) – Změna, aktualizaci dat
    - Delete (D) – Smazání dat
- Metody jsou implementovány pomocí metod HTTP protokolu

Metody pro přístup ke zdrojům: (Malý, 2009)

REST implementuje čtyři základní metody, známé pod označením **CRUD**, tedy vytvoření dat (Create), získání požadovaných dat (Retrieve), změnu (Update) a smazání (Delete). Tyto metody jsou implementovány pomocí odpovídajících metod HTTP protokolu.

- **GET (Retrieve)**

Základní metodou pro přístup ke zdrojům je **získání zdroje**. Jedná se o požadavek na stránku. Každý zdroj má podle rozhraní REST vlastní identifikátor (URI). Pomocí HTTP GET požadavku se získávají data konkrétního zdroje.

Pokud potřebujeme získaná data nějak upravit či filtrovat, můžeme je předat jako součást požadavku, jako tzv. „Query parametry“, tedy za otazníkem.

- **POST (Create)**

Získání dat je jednoduché a přímočaré. Pro vytvoření dat slouží metoda POST, známá z HTML formulářů. U metody POST není ve chvíli volání známý přesný identifikátor zdroje. Proto se pro POST používá domluvený společný identifikátor („endpoint“).

Po odeslání by měl server vrátit patřičný návratový kód – HTTP má k tomu účelu stavový kód 201 – Created, v němž lze předat URI nově vytvořeného zdroje.

- **DELETE**

Zdroj lze smazat pomocí volání URI HTTP metodou DELETE. V praxi bývá někdy problematické vyvolat HTTP metodu DELETE – spousta HTTP nástrojů či HTML formuláře jsou omezeny pouze na metody POST a GET. V praxi se proto u REST rozhraní používají náhradní způsoby – např. volání pomocí POST s parametrem, který sděluje, že má být ve skutečnosti použita metoda DELETE.

- **PUT (Update)**

Operace změny je podobná operaci vytvoření (metoda POST), s tím rozdílem, že voláme konkrétní URI konkrétního zdroje, který chceme změnit a v těle předáme novou hodnotu. Na rozdíl od POST je u úprav zdroje jeho URI už známá, takže ji lze zadat.

REST vs. RESTful

- REST je název technologie.
- RESTful (volným překladem RESTové) je využití technologie (implementace).

3.8.1 API

API (Application Programming Interface). Volným překladem je API **rozhraní pro programování aplikací**. Lze jej nalézt na úrovni operačního systému, konkrétních programů nebo na úrovni webových služeb. Na webových stránkách uživatel může využívat různé funkce a aplikace. K těmto funkcím je možné se dostat pomocí API. (Rybařík, 2015)

3.8.2 REST Server a Klient

REST Server

REST Server poskytuje RESTful Services, tedy RESTové služby běžící třeba na aplikačním serveru, co však musí REST služby splňovat je, že každý samostatný resource musí mít vlastní unikátní URL. Každý resource vyjadřuje stav aplikace a chování. Za každým stojí různá logika, chování a samotné odpovědi serveru (například pomocí HTTP status kódů). Resource může být různě reprezentován, nejčastěji se jedná o HTML, JSON, XML. Samotná reprezentace je ekvivalentní odeslání požadavku či přijetí odpovědi. (Rybařík, 2015)

REST Klient

REST klient využívá REST API jako vrstvu spojení se serverem na výměnu dat pomocí požadavků na server. Nejčastěji je přitom dosahováno pomocí 4 základních metod pro přístup ke zdrojům (CRUD). (Rybařík, 2015)

3.8.3 JAX-RS (Jersey implementace)

Jersey je open source framework implementace JAX-RS (JSR 311 & JSR 339 specifikace součástí Java EE 6 od verze 1.1) pro vývoj RESTových webových služeb. Jedná se o rozšíření samotného JAX-RS frameworku nabízející množství rozšíření pro zjednodušení vývoje aplikací. (Jersey, 2016)

Framework obsahuje v základu od JAX-RS API spoustu anotací pro implementaci REST API do zdrojového kódu aplikace: (Jersey, 2016)

- @Path – relativní cesta třídy, metody nebo přímo služby na úrovni webové adresy
- @GET, @PUT, @POST, @DELETE, @HEAD – specifikace HTTP požadavku nad metodou REST služby
- @Produces – návratový typ Contentu
- @Consumes – příjmový typ Contentu
- @Context – anotace využívaná pro Injection tříd, nejčastěji „@Context HttpServletRequest request“ jako parametr metody

3.9 HTTP / HTTPS

3.9.1 HTTP (Hyper Text Transfer Protocol)

HTTP je internetový protokol, který slouží pro přenos hypertextových dokumentů HTML, umí samozřejmě přenést i jiné typy dokumentů a zaobalit jiné protokoly, defaultně běží na TCP protokolu transportní vrstvy na portu 80, alternativně na 8080 nebo dle nastavení. Nejrozšířenější verze 1.1 bude brzy postupně nahrazena verzí 2.0. Tento protokol je nezabezpečený a samotná komunikace může být odposlouchávána pomocí MITM (Man In The Middle) metodou odposlouchávání komunikace, dále zachycena nebo pozměněna na jakémkoliv prvku v síti, i když se nejedná přímo o MITM, také je možné komunikaci odposlouchávat třeba na Wi-Fi. (Bouška, 2014)

3.9.2 HTTPS (Hyper Text Transfer Protocol Secured)

HTTPS rozšiřuje HTTP o SSL/TLS spojení, které nešifrované HTTP zaobaluje do šifrovaného spojení. HTTPS běží také na TCP protokolu transportní vrstvy na portu 443, alternativně 8443 nebo také dle nastavení. (Bouška, 2014)

3.10 SSL/TLS

SSL/TLS se obvykle používá při zabezpečení koncových bodů a průchozí komunikace, nejčastěji při HTTPS, ale třeba i SSH terminálovém připojení. (Bouška, 2014)

3.10.1 Verze SSL/TLS

SSL končí na verzi 3.0 a TLS začíná na verzi 1.0 (SSLv3). Mezi SSLv3 a TLSv1 jsou drobné rozdíly, nicméně jsou tak drobné, že pokud se mluví o TLSv1.0 často se myslí SSLv3 a naopak, avšak SSLv3 a starší jsou dnes nebezpečné (POODLE vulnerability), a tedy i na drobných rozdílech záleží. Tyto rozdíly jsou v inicializaci zabezpečeného spojení, následné možnosti šifrování jsou stejné. Následují verze TLSv1.1 a TLSv1.2. TLS lze považovat za bezpečnější variantu. (Bouška, 2014)

3.10.2 Self-signed / CA (Certificate Authority)

Self-signed

O tomto termínu se mluví v případě, že certifikát je podepsán sám sebou. CA může být podepsána jinou CA nebo podepsána sama sebou. Self-signed certifikáty jsou často používány v aplikačních prostředích běžících na LAN nebo vývojovém a testovacím prostředí. (Bouška, 2014)

Certificate Authority

CA je certifikační autorita, která by měla být důvěryhodná z ohledu toho, jaké certifikáty vydá, komu je vydá a jakým stylem je vydává. Privátní klíče CA jsou nebo by měly být dobře střeženy. Pokud by se útočník dostal k tomuto privátnímu klíči, tak by mohl snadno podepisovat vlastní keypairy a účel certifikační autority, v kterou klient vkládá důvěru byl narušen bez vědomí klienta. (Bouška, 2014)

3.10.3 Šifrované spojení SSL-TLS

Prohlížeč nebo počítač klienta obsahuje databázi důvěryhodných CA a jejich veřejných klíčů, na které se vztahují jistá pravidla pro zajištění jejich důvěryhodnosti. Celá důvěryhodnost je tedy postavená na důvěře CA a správnosti jimi vydanými certifikáty. (Bouška, 2014)

Při inicializaci spojení se vytvoří šifrované stavové spojení (session). Na začátku takového připojení je nejdříve potřeba stanovit základní parametry spojení, jako třeba jaká šifra bude použita, jaká verze SSL-TLS je podporována, takový úkon se nazývá handshake, při němž dochází k ověření identit. (Bouška, 2014)

Nejčastěji při handshake, kdy se připojuje klient se ověřuje identita serveru, ale existuje i vzájemné ověření identit. Ověření identity serveru je důležité, abychom neodesílali data na nesprávný server nebo server vydávající se za správný. Ověření identity klienta (vzájemné ověření) je pak použito v uzavřených systémech, třeba při připojení na interní API aplikace nebo jako uživatel přes SSH na server. Není-li veřejný certifikát, který server poskytuje podepsán důvěryhodným CA je v případě důvěry serveru tento veřejný certifikát uložit do TrustStore klienta při ověřování serveru, při vzájemném ověření identit toto platí

jak pro server, tak pro klienta. (Pokud ověření identit selže, je spojení na straně, kde došlo k selhání ověření ukončeno.) (Bouška, 2014)

Klient vygeneruje „pre-master secret“, jež slouží pro inicializaci následovaného šifrovaného spojení. Pre-master secret je zašifrován veřejným klíčem serveru a jen server s privátním klíčem může pak pre-master secret dešifrovat. Jedná se o asymetrickou kryptografii. Po dešifrování se pre-master secret použije na straně serveru i klienta pro aplikování série kryptografických úkonů, dle domluvené šifry, tím vznikne „master secret“, který je použit pro session key. Session key se vytvoří hashovacím algoritmem z master secret. Daný session key slouží k aplikaci symetrické šifry pro samotnou komunikaci mezi serverem a klientem, tedy k šifrování a dešifrování dat zprávy. (Bouška, 2014)

4 Vlastní práce

4.1 Úvod do projektu IntelliJ IDEA s použitím Mavenu

Pro Java projekt bylo vybráno IDE IntelliJ IDEA z především důvodu uživatelské preference. Jiné IDE samozřejmě nabízejí také základní funkcionality potřebné a pohodlné pro Java vývoj. Pracuje-li na projektu více lidí s různými IDE, vyžaduje projektová struktura konvertibilitu do jiného prostředí nebo portabilitu mezi prostředími. Ačkoliv IntelliJ IDEA podporuje převod projektu do Eclipse a Eclipse podporuje převod projektu do Netbeans není řešení konvertibility projektu vyhovující. Dnes se proto používají nástroje umožňující portabilitu projektu, kdy je možné projekt zkompilovat a sestavit i bez jakéhokoliv IDE. Pro vývoj na platformě Android se hojně využívá Gradle. Pak je tu Maven, který byl použit v tomto projektu. Oba nástroje jsou do velké míry v základní funkcionalitě stejné.

4.1.1 Maven

Jakmile se naučí člověk s Mavenem, tak zjistí všechny jeho výhody a bude je moci využít naplno. Po počáteční konfiguraci projektu se o nic není potřeba starat, maximálně o úpravy dle požadavků na projekt.

pom.xml

Každý Maven modul (nebo celý projekt) má vlastní pom.xml, obsahující základní informace o modulu či projektu. Nastavit zde lze závislost na dalších Maven modulech, knihovnách a projektech, které mohou být lokální nebo vzdálené.

Obsah pom.xml lze rozdělit na:

- Základní informace o modulu potřebné pro build
- Properties – Vlastní konstanty
- Dependencies – Použité externí knihovny
- Repositories – Použité externí repositáře
- Build – Nastavení celého build procesu za pomoci konfigurace Maven pluginů

4.1.2 Strukturální rozdělení modulů

Projekt sestává ze 2 programů, a to klient a server. Projekt tedy zákonitě musí obsahovat minimálně 2 moduly pro oddělení těchto částí programu. Pro utilizaci kódu společného pro obě části programu je potřeba ještě jedna knihovna. Pokud by byla vyžadována automatizace procesů, bylo by možné ještě celý projekt zaobalit do jednoho obecného modulu, pro účely této práce stačí moduly 3.

Vztahy jednotlivých modulů mezi sebou lze dvěma způsoby. Vztahem parent x child moduly, kde každý child modul může mít maximálně jeden parent modul. Pro potřeby projektu byl zvolen způsob druhý nazývaný agregace. V agregaci se určuje pouze závislost na ostatních modulech, a tedy jeden child modul může mít 2 parent moduly.

Závislosti modulů pro klienta:

- rest-client (jar)
    - rest-commons (jar)
        - dependencies (externí knihovny)
    - dependencies (externí knihovny)

Závislosti modulů pro server:

- rest-server (war)
    - rest-commons (jar)
        - dependencies (externí knihovny)
    - dependencies (externí knihovny)

4.1.3 Potřebné balíčky knihoven (Maven Dependencies)

rest-client

- rest-commons
- Jersey Client
- Jargs

rest-server

- rest-commons
- Jersey Server
- Javax servlet

rest-commons

- Jersey Jackson
- Log4J
- ICU4J

4.1.4 Maven pluginy

rest-client

- Maven Clean plugin
- Maven Compiler plugin
- Maven Dependency plugin
- Maven Resources plugin
- Maven JAR plugin
- Maven ANTrun plugin

rest-server

- Maven Clean plugin
- Maven Compiler plugin
- Maven Dependency plugin
- Maven Resources plugin
- Maven JAR plugin

rest-commons

- Maven Clean plugin
- Maven Compiler plugin
- Maven Install plugin

4.1.5 Práce s Mavenem

V IDE Intellij IDEA jsou předpřipravené nastavení sestavení mimo Maven i přímo využívající Maven. Projekt lze tedy sestavit vícero možnostmi, v jiném IDE je však nutné projekt uzpůsobit danému IDE nebo využívat právě Maven, k čemuž také slouží. Díky Mavenu je také možné projekt sestavit i mimo jakékoliv IDE, a to s vlastní Maven aplikací.

Sestavení z IDE přes Maven:

1. Vybereme modul (rest-client, rest-server)
2. Otevřeme konzoli mavenu
3. Zadáme příkaz: mvn clean package

Sestavení v standalone Mavenu (Centos7):

- mvn -B -f rest-app/rest-client/pom.xml clean package
- mvn -B -f rest-app/rest-server/pom.xml clean package

4.2 Tomcat a SSL/TLS

Doporučené verze Tomcatu pro tento projekt je 7 a vyšší, pro testovací účely byla použita verze Tomcat 9.0.0.M17. SSL/TLS se u serveru řeší na úrovni Connectoru v Tomcatu. Klasický HTTP port 80 bude přesměrován na port 443, tedy Connector s HTTPS. Connector podporuje pouze TLSv1 a vyšší, ověření identity klienta je možné, ale nevyžadováno. Pokud nemá klient privátní klíč, kterému server věří, tak proběhne pouze klasické ověření serveru ze strany klienta, a pokud by takový klient chtěl následně získat přístup k zabezpečenému obsahu, měl by smůlu. Toto nastavení je vhodné pro webové aplikace s RESTful API, kde zabezpečený obsah je řešen na úrovni Servletu a je možné na klasický web přistupovat ze strany klienta normálně a na API s autentizací.

Server.xml

Nastavení aplikačního rozhraní:

```
<Connector
    port="80"
```

```
        protocol="HTTP/1.1"
        connectionTimeout="20000"
        redirectPort="443"
        enableLookups="false"
/>
<Connector
        port="443"
        protocol="org.apache.coyote.http11.Http11NioProtocol"
        SSLEnabled="true"
        maxThreads="200"
        scheme="https"
        secure="true"
        SSLProtocol="TLSv1+TLSv1.1+TLSv1.2"
        clientAuth="want"
        SSLVerifyClient = "optional"
        SSLVerifyDepth = "2"
        URIEncoding="UTF-8"
        keystoreFile="/opt/tomcat/conf/keystore.jks"
        truststoreFile="/opt/tomcat/conf/truststore.jks"
        keystorePass="changeit"
        truststorePass="changeit"
        keystoreType="JKS"
        truststoreType="JKS"
        keyAlias="rest-app server (rest-app ca)"
        keyPass="rest"
        disableUploadTimeout="true"
        enableLookups="false"
/>
```

4.3 Java EE Web Application s SSL/TLS

4.3.1 Deployment Descriptor a Servlet

Celý deployment descriptor se nastavuje v souboru web.xml, kde se také nastavují servlety a v případě tohoto řešení s Jersey potom navíc ve třídě ApplicationConfig.

web.xml (Deployment Descriptor)

```
<!-- Init core -->
<listener>
  <listener-class>cz.czu.restapp.core.CoreServletContextListener</listener-class>
</listener>
```

K základnímu nastavení je potřeba nastavit Listener, tedy třídu implementující interface ServletContextListener, kde při startu serveru a načtení artefaktu je vytvořena instance této třídy a zavolána metoda contextInitialized. Tato funkcionalita by se dala pro pochopení přirovnat ke statické metodě main v klasické Java SE aplikaci, tedy inicializační metoda aplikace.

CoreServletContextListener

Celá funkcionalita této třídy bude zavolat metodu init() instance jádra serveru ze singletonu jádra serveru.

Pro pochopení: Ve třídě ServerCore je ve statickém konstruktoru vytvořena instance sama sebe, která je uložena ve statické proměnné a dostupná přes getter. Instanci třídy ServerCore je možné pouze v daném statickém konstruktoru díky privátnímu konstruktoru třídy. Při práci s jádrem server už jen získáme referenci na danou instance jádra pomocí:

ServerCore.getInstance()

A následně pracuje s danou instancí.

4.3.2 Servlet s SSL/TLS

Celé zabezpečení zajišťuje aplikační prostředí serveru, které autentizuje klienta na danou roli.

V souboru tomcat-users.xml je nutné nastavit role a uživatele mající přístup k daným rolím. Server s API pro jeden typ aplikace upotřebí jednu roli. Pro klienty byla vybrána role „client“ a k tomu jeden uživatel. Při nastavení uživatele, který se autentizuje klientským certifikátem je username výpis údajů certifikátu a password nastavený na „null“ s přiřazenou rolí.

```
<role rolename="client"/>
<!-- Clients -->
<user username="CN=rest-app Client, OU=Milos Havranek
xhavm030@studenti.czu.cz, O=KII pef.czu.cz, L=Prague, ST=Czech republic, C=CZ,
EMAILADDRESS=spli-t@seznam.cz" password="null" roles="client"/>
```

Klientský certifikát bohužel nemůže obsahovat české znaky a některé speciální znaky, jinak autentizace nebude fungovat (Příklad:“ěščřžýáíéůúťďň“).

Ve web.xml potom řešíme, jaká role má kam přístup a které stránky budou zabezpečeny.

Zapnutí SSL/TLS pro celý Servlet

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

Zapnutí SSL/TLS autentizaci pro API a přiřazení rolí

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>API</web-resource-name>
    <url-pattern>/api/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>client</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>

<login-config>
  <auth-method>CLIENT-CERT</auth-method>
</login-config>

<security-role>
  <role-name>client</role-name>
</security-role>
```

4.4 Jersey (JAX-RS) REST backend

Celé REST API běží na Jersey verze 2. Jersey je rozšíření Java EE knihovny JAX-RS. Vytvoření REST API díky Jersey je jednoduché, problém nastává při potřebě TLS-SSL

šifrovaného připojení, kde u serveru toto zajišťuje z 90 % aplikační prostředí a Java, tedy Tomcat, nicméně zabezpečeného REST klienta je nutné vytvořit vlastního, chceme-li složitějšího a propracovanějšího klienta. O TLS-SSL se Java stará nativně, avšak bez rozumného nastavení.

4.4.1 Server Servlet

Existuje více způsobů, jak nastavit na serveru servlet, lze jej nastavit ručně ve web.xml, což je nastavení pro javax.servlet, v tomto nastavení se následně nastavuje i zabezpečení, filtry pro webové služby a jiné. Použit byl způsob rozšíření třídy Application. ApplicationConfig tedy staví na třídě Application, instance této třídy je vytvořena automaticky při nastartování serveru. V konstruktoru této třídy je nutné přidat do resources REST služby, které mají být aktivní.

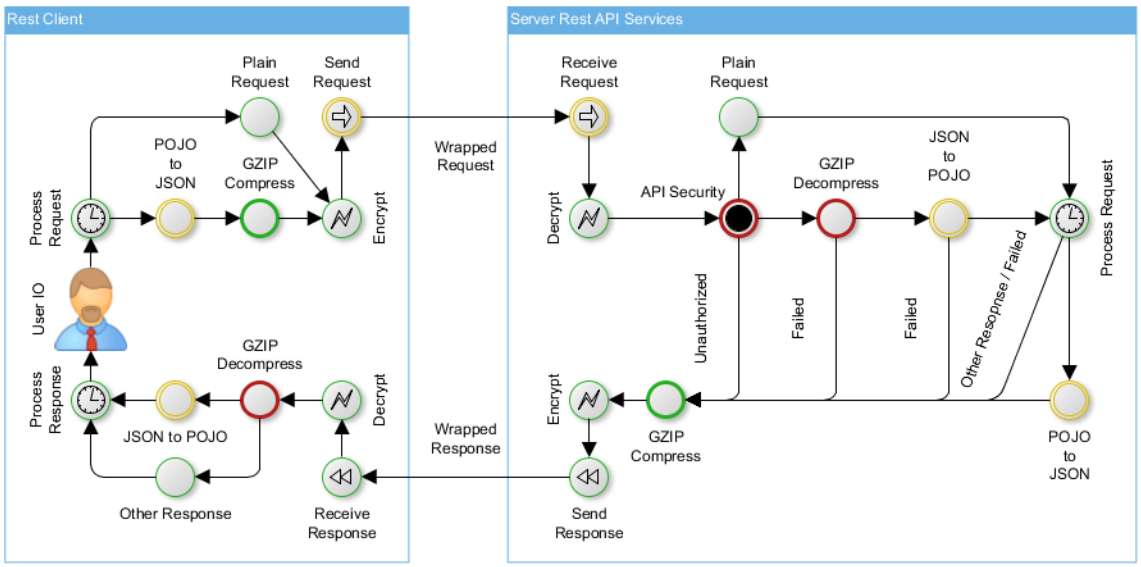
V Tomcatu je potřeba nastavit cache pro MultiPart Feature, jinak by byla omezena velikost posílaného obsahu. Nastavení se provede v souboru: „context.xml“. Dovnitř XML tagu <Context> a </Context> se vloží tag s atributy:

```
<Resources cachingAllowed="true" cacheMaxSize="100000" />
```

4.5 REST logika

Při komunikaci je důležité si uvědomit role komunikujících stran, ve většině případů schématu klient vs. server, klient zásobuje daty server, a naopak server zásobuje daty klienta, samotné zpracování a uchování všech dat probíhá na samotném serveru.

Obrázek 1 REST diagram



Zdroj: vlastní zpracování

Klient vyžaduje interakci uživatele poskytnutím vstupních parametrů, dat nebo úkonů klientovi, klientská aplikace pak ve zkratce zpracuje takový požadavek a odešle jej na server. Při zpracování samotného požadavku může dojít ke konverzi datových formátů a komprese samotných dat, takový požadavek pak je odeslán na server pouze v případě, že je navázáno zabezpečené připojení přes HTTPS, kde se vytvoří šifrovaná session.

Server po navázání zabezpečeného připojení HTTPS požadavek dešifruje a zjistí, zda má klient příslušná oprávnění zasílat na server požadavky. Požadavek je nadále zpracován, může být dekomprimován a převeden do klasického Java objektu. Server následně zpracuje odpověď, která obsahuje nějaký HTTP kód v případě chyby nebo jednoduché odpovědi, jinak se Java Objekty převedou do formátu JSON. Při streamování binárních dat nebo textu samozřejmě ke konverzi do JSON nedochází. Odpověď serveru je dále komprimována a v té samé šifrované session šifrovaně odeslána.

Klient odpověď přijme a dešifruje, před zpracováním odpovědi případně dekomprimuje a převede na klasický Java objekt. Odpověď je zpracována a podle logiky klienta se s daty dále pracuje. Uživatel dostane požadovaná data nebo odpověď, v případě selhání podle HTTP kódu je možné s chybovou odpovědí dále naložit.

V tomto schématu vždy klient zasílá požadavky serveru a od serveru musí vždy obdržet odpověď, ať už chybovou nebo očekávanou. Naopak klient se může kdykoliv během

celého procesu odmlčet a ukončit spojení, od serveru tedy vyžadujeme odpověď za každé situace.

4.5.1 HTTP status kódy

Enum RestStatus zajišťuje přesné definování obsahu a významu zpráv HTTP status kódů. Nastane-li jakákoliv situace při dotazování serveru, server vždy odpoví odpovídajícím status kódem. RestStatus obsahuje jak definované základní, tak široce používané HTTP kódy, které jsou nedefinované, ale jen v komentáři pro jejich nepoužívání. HTTP kódy samotné aplikace jsou pak využity ve volném pásmu. Je-li HTTP kód 200, tak aplikace přijme obsah odpovědi, je-li nějaký, jinak všechny ostatní kódy řeší ClientResponseHandler, který při obdržení jakéhokoliv HTTP kódu, vypisuje HTTP kód s popisem z enumu RestStatus. Obsahuje-li odpověď serveru i zprávu o konkrétní chybě, tak ji vypíše.

Tabulka 1 - Nejčastěji používané HTTP kódy a jejich význam

Kód	Popis nebo alternativní význam přímo v aplikaci
200	Požadavek úspěšně zpracován, odpověď + data
256	PingService odpověď znamenající to samé, co kód 200, jiný kód zvolen z důvodu unikátnosti od klasických webových aplikací
302	Přesměrování serverem, znamenající použití špatného protokolu HTTP místo HTTPS
401	Neautorizovaný přístup na API (Servlet + Tomcat zabezpečení -> client-cert)
403	Neautorizovaný přístup na danou službu REST API nebo ke konkrétním datům
404	Špatný server nebo startující server s nenačteným obsahem
500	Nespecifikovaná chyba serveru, většinou nastane při vyhozené Exception

Zdroj: vlastní zpracování

4.6 Jackson Interceptor

Data přenášená přes REST API jsou ve formátu JSON. Jackson Interceptor zajišťuje automatickou konverzi klasických Java objektů do JSON a zpět bez nutnosti manuálního převodu formátů dat.

4.6.1 POJO to JSON

Při odesílání dat přes REST API nastavíme typ odesílaných dat (MediaType) na JSON a odešleme klasický Java objekt. Před odesláním dat Jackson Interceptor zachytí objekt před odesláním a nahradí jej objektem typu JSON, který je snadno převeden při odesílání do textové podoby.

4.6.2 JSON to POJO

Při přijímání dat přes REST API nastavíme typ přijímaných dat v případě MediaType na JSON a v případě EntityType či konstruktoru na požadovanou třídu. Přijatá data ve formátu JSON jsou z textového formátu převedena na JSON objekt a zachycena Jackson Interceptorem, který vytvoří novou instanci dané třídy, tedy objekt obsahující původní data. U normálních tříd stačí konstruktor a settery, na ENUM je bohužel potřeba napsat vlastní deserializer v jeho vlastní třídě. Setkat se lze i s případem, kdy bude potřeba deserializovat více objektů jedné třídy obsažené v Listu, tam by EntityType vypadal takto:

List<Book> books = response.readEntity(new GenericType<List<Book>>() {});

A u normálního objektu takto:

Book book = response.readEntity(Book.class);

4.7 GZIP Interceptor

Obsah přenášený mezi 2 aplikacemi je možné komprimovat pomocí GZIP Interceptorů, kde odesílaný obsah se automaticky komprimuje a po přenosu dekomprimuje.

U klienta je potřeba zaregistrovat třídu GZIPWriterInterceptor a GZIPReaderInterceptor při vytváření REST HTTP klienta (třída RestClient), vše pak probíhá automaticky.

U serveru se ty samé třídy přidají do „resources“ v ApplicationConfig, stejně jako služby REST Resources, pak už jen v dané službě na třídu přidáme anotaci @Compress nebo @Decompress dle potřeby, lze i obojí.

4.7.1 GZIP komprese

Ke kompresi obsahu přenášených dat slouží GZIPWriterInterceptor, který před odesláním dat zaobalí posílaný obsah do GZIPOutputStream a přidá HTTP hlavičku "Content-Encoding" s nastavením na "gzip" kompresi. GZIP dekomprese.

K dekompresi obsahu přenášených dat slouží GZIPReaderInterceptor, který při přijímání dat dekomprimuje přenášený stream na původní obsah, pokud HTTP hlavička obsahuje nastavení GZIP kódování obsahu, neobsahuje-li tuto hlavičku, stream je předán dál bez dekomprimace. Je tedy možné přijímat komprimovaný i nekomprimovaný obsah.

4.8 Příklady použití (klient vs. server)

Projekt obsahuje také příklady použití v podobě využití popisovaných technologií této bakalářské práce. Jedná se konkrétně o stavbu samotného klienta a jeho funkcionalit potřebných pro komunikaci včetně samotných tříd a metod obsahující konkrétní příklady. Server obsahuje základní kostru potřebnou pro svůj běh, zabezpečení a komunikaci se zabudovanými konkrétními příklady ve formě REST Services, tedy poskytovaných služeb.

4.8.1 PingService

Nejjednodušší ze všech příkladů použití je PingService mající jednu metodu typu GET, která vrací HTTP status kód 256 (OK). Přes tuto jednoduchou metodu lze určit například, zda jsme připojeni na správný server přes správný port a správný protokol s TLS-SSL zabezpečením a zda máme na API přístup. Vše záleží jen na úpravě této služby dle potřeb.

Funkce klienta

- Odeslat požadavek typu GET serveru
- Přijmout odpověď serveru

Funkce serveru

- Vrátit odpověď klientovi

Požadavek na server:

GET https://localhost:443/api/ping

Odpověď serveru:

256

Cache-Control: private

Content-Length: 0

Date: Sun, 05 Mar 2017 20:11:25 GMT

4.8.2 SudokuService – SudokuSolver

Za normální situace by ve všech případech bylo sudoku generováno nebo řešeno přímo na klientovi, avšak je to dobrý příklad použití pro přenos, transformace, sběr a zpracování dat. Klient načte zadání Sudoku ze souboru, vytvoří instanci SudokuEntity, převede do JSON formátu, zkomprimuje, zašifruje a odešle. Server dešifruje, dekomprimuje, převede JSON na POJO, ověří, zda je zadání Sudoku validní a vyřeší jej, s daty může dál nakládat. Odpoví chybovým kódem nebo zpět vrátí vyřešené Sudoku v podobě objektu SudokuEntity (při zachování přechozích procedur transformace, komprimace, šifrování). Klient v případě obdržení neočekáváného kódu Vypíše kód a případně přiloženou zprávu, jinak uloží vyřešené Sudoku a vypíše na obrazovku.

Funkce klienta

- Načíst Sudoku zadání
- Odeslat požadavek typu POST serveru
- Přijmout odpověď serveru
- Uložit Sudoku řešení

Funkce serveru

- Přijmout data v požadavku
- Ověřit řešitelnost matice
- Vyřešit Sudoku
- Logování proběhlých operací

- Vrátit odpověď klientovi

Požadavek na server:

POST https://localhost:443/api/sudoku

Content-Type: application/json

{

"sudoku":[[7,0,0,0,4,5,8,0,1],[8,0,0,0,0,1,0,0,5],[0,5,1,8,3,6,4,7,0],[3,0,6,5,7,2,0,0,4],[0,2,0,6,1,9,7,3,8],[1,7,0,0,8,0,0,0,6],[0,9,0,0,0,0,1,8,0],[0,1,0,9,0,0,6,0,3],[2,3,8,1,6,7,5,0,0]]

}

Odpověď od serveru:

200

Cache-Control: private

Content-Encoding: gzip

Content-Type: application/json;charset=utf-8

Content-Length: 153

Date: Sun, 05 Mar 2017 20:04:48 GMT

{

"sudoku":[[7,6,3,2,4,5,8,9,1],[8,4,2,7,9,1,3,6,5],[9,5,1,8,3,6,4,7,2],[3,8,6,5,7,2,9,1,4],[5,2,4,6,1,9,7,3,8],[1,7,9,3,8,4,2,5,6],[6,9,5,4,2,3,1,8,7],[4,1,7,9,5,8,6,2,3],[2,3,8,1,6,7,5,4,9]]

}

4.8.3 SudokuService – SudokuGenerator

Princip funkčnosti je stejný, jako u SudokuSolver, avšak server přijímá GET požadavek a vrací pokaždé jinou odpověď ve formě SudokuEntity obsahující řešitelnou matici.

Funkce klienta

- Odeslat požadavek typu GET serveru
- Přijmout odpověď serveru
- Uložit Sudoku

Funkce serveru

- Přijmout požadavek
- Generovat Sudoku
- Logování proběhlých operací
- Vrátit odpověď klientovi

Požadavek na server:

GET https://localhost:443/api/sudoku

Odpověď serveru:

200

Cache-Control: private

Content-Encoding: gzip

Content-Type: application/json;charset=utf-8

Content-Length: 148

Date: Sun, 05 Mar 2017 20:04:48 GMT

{

"sudoku":[[0,0,7,4,5,1,0,0,0],[4,0,5,9,0,0,7,0,1],[0,3,0,8,7,2,0,9,4],[9,0,0,3,0,8,4,7,5],[1,7,0,2,0,5,9,6,0],[0,0,0,6,9,0,1,0,0],[7,6,8,5,3,4,0,0,9],[5,0,0,7,0,6,8,4,3],[3,4,0,0,0,0,0,0,7]]

}

4.9 Návrh REST klienta

REST klient byl naprogramován podle následujících požadavků:

- Konfigurovatelnost klienta

- Snadné šíření zabezpečeného klient bez dodatečné konfigurace
- Možnost sestavení klienta s obejitím zabezpečení pro testovací účely
- Implementovatelnost dodatečných Interceptorů

4.9.1 InsecureHostnameVerifier a InsecureTrustManager

InsecureTrustManager je třída pro třídu RestClient, která by měla být využívána jen na testování. Normální TrustManager věří pouze certifikátům podepsaným důvěryhodnou certifikační autoritou či certifikátům majícím veřejný certifikát v TrustStore. InsecureTrustManager ověří všechny certifikáty poskytnuté klientovy jako důvěryhodné, což samo o sobě zanechává možnost šifrované komunikace, která ovšem může nemusí být se serverem, na který se klient chtěl připojit, ale prostředníkem (MITM), který pak přeposílá danou komunikaci serveru, s odposloucháváním komunikace, její úpravy či dokonce posílání falešných zpráv tvářící se jako klient nebo server protistraně.

InsecureHostnameVerifier je také třída pro třídu RestClient a slouží pro ignorování hostname či IP adresy při ověřování identity serveru, s tím že je vyžadován stále platný a důvěryhodný certifikát. Využití této třídy je také jen pro testování či plně kontrolované prostředí sítě.

4.9.2 RestClient

Třída RestClient má privátní konstruktory a nelze vytvořit její instanci. Hlavním účelem třídy RestClient je pomocí statických metod vytvořit instanci třídy Client (javax.ws.rs) s dodatečným nastavením, jako je zabezpečení SSL/TLS, registrace Interceptorů a ostatní nastavení klienta.

Zabezpečený klient (Client) může být vytvořen bez externích keystorů a je tedy možné celou aplikaci snadno přenášet bez potřeby rekonfigurace, tento princip je často využíván u Android aplikací u veřejného API (Proto bude také certifikát klienta v další kapitole self-signed a nepodepsán certifikační autoritou, kvůli vystavení privátního klíče veřejně). Takový klient pak využívá keystorů zabalených v .jar archivu aplikace, kde vývojář nastaví vše potřebné a po sestavení a deploymentu se není třeba o nic starat. V jiném případě je potřeba poskytnout keystory manuálně s přístupovými údaji.

Jako další nastavení REST klienta bylo potřeba nastavit:

- Kódování na utf-8
- Přenos Contentu pomocí chunků (po částech), aby nedošlo k přetečení paměti při přenosu Contentu
- Registrace JacksonJSONProvider
- Registrace GZIPWriterInterceptor
- Registrace GZIPReaderInterceptor

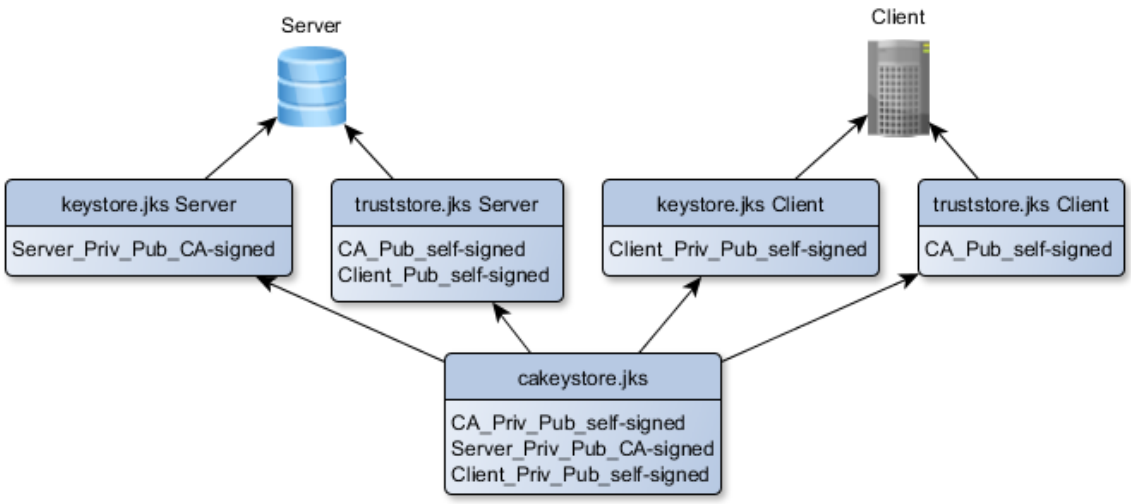
4.9.3 ClientResponseHandler

ClientResponseHandler se stará o překlad známých HTTP status kódů při neočekávané odpovědi serveru. Kódy interpretuje pomocí třídy RestStatus a známých kódů, není-li nalezen známý kód je číselný kód vypsán s varováním bez interpretace. Obsahuje-li odpověď serveru Content je vypsán jako text, většinou obsahuje dodatečné chybové hlášky nebo upřesnění odpovědi.

4.10 Generování certifikátů

Vygenerované certifikáty je potřeba uložit či importovat do keystorů. Pro každý keystore byl zvolen formát JKS s defaultním heslem:“changeit“.

Obrázek 2 KeyStore / TrustStore hierarchie certifikátů



Zdroj: vlastní zpracování

Celkem bude potřeba 5 keystorů.

1. Privátní keystore pro uchování CA privátního klíče mimo dané aplikace
2. Keystore serveru pro ověření identity serveru a zabezpečení připojení
3. Truststore serveru pro ověření identity klientů
4. Keystore klienta pro autentizaci na serveru a zabezpečení připojení
5. Truststore klienta pro ověření identity serveru

4.10.1 Privátní KeyStore

Privátní keystore je potřeba jen pro bezpečného uchování CA privátního klíče pro podepisování nový klíčů, také zde lze uchovat všechny ostatní keypairy. Privátní keystore v projektu byl pojmenován: “cakeystore.jks“ a nebude distribuován s aplikacemi.

4.10.2 CA

Certifikační autorita v tomto případě byla použita vlastní, je tedy potřeba vygenerovat self-signed certifikát, tedy certifikát podepsaný sám sebou. Certifikát se stane certifikační autoritou ve chvíli, kdy podepíše nějaké další certifikáty svým privátním klíčem a klienti takové autoritě věří, tedy mají veřejný certifikát takové autority ve svém truststore.

Pro potřeby této bakalářské práce byl vygenerován keypair verze 3 algoritmem RSA (4096bit) s SHA-512 o platnosti 100 let a údaji:

CN: rest-app CA

OU: Miloš Havránek xhavm030@studenti.czu.cz

O: KII pef.czu.cz

L: Prague

ST: Czech republic

C: CZ

E: spli-t@seznam.cz

Certifikační autorita nepotřebuje ověření identity pomocí hostname či IP adresy, a tak CN bude jen název certifikátu.

Pokud by byl certifikát jen self-signed a nebyl CA, tak by bylo možné do CN dát IP adresu nebo hostname, pro ověření identity serveru nebo nechat CN jako název certifikátu a identitu vztahující se na tento certifikát doplnit pomocí extension SAN k certifikátu. Subject Alternative Name má tu výhodu, že je do této extension možné dát více IP adres a doménový jmen zároveň. Samozřejmě lze použít v CN a SAN wildcard, tedy zástupný znak * pro domény třetího řádu a taky vlastně podepsat všechny nebo nějaké subdomény vaší domény druhého řádu pomocí jednoho záznamu, pro IP adresy toto nefunguje.

Certifikát lze vygenerovat v aplikaci s GUI (KeyStore Explorer) či v terminálu pomocí programu: “keytool“.

4.10.3 Generování podepsaných certifikátů pro server

Keypair pro server je potřeba vygenerovat podobně, jako CA keypair s tím rozdílem, že tento keypair musí být podepsaný danou certifikační autoritou (rest-app CA) a je potřeba ověřovat identitu serveru či klienta. Při ověřování identity serveru se ověřuje hostname či IP adresa serveru podle certifikátu, neshoduje-li se údaj serveru s údajem poskytnutým certifikátem, ověření selže.

CN serveru: rest-app Server

SAN serveru:

- ***DNS Name: localhost***
- ***IP Address: 0:0:0:0:0:0:0:1***
- ***IP Address: 127.0.0.1***

4.10.4 Generování certifikátů pro klienta

Keypair pro klienta je potřeba vygenerovat stejně, jako CA keypair z toho důvodu, že vystavujeme API veřejným klientům, kteří by mohli použít certifikát k podepsání dalších a tím získat nepřímý přístup k podúrovni CA.

CN klienta: rest-app Client

Ostatní údaje u tohoto certifikátu jsou bez speciálních znaků a diakritiky, Tomcat autentizace jinak selže.

5 Výsledky a diskuse

Mnoho bakalářských a diplomových prací bylo napsáno, a ještě více dokumentace na dané technologie a témata je volně dostupné na internetu. Vize autora této práce bylo věnovat méně prostoru samotné abstrakci teoretických východisek a místo toho vytvořit jednoduché a reálně funkční řešení. Aplikace serveru a klienta je možné využít jako základ pro jiné aplikace. Komunikační rozhraní obsahuje jednoduché ukázkové příklady pro lepší pochopitelnost.

Rozsahem u takto komplexního tématu (použité technologie atd..) nebylo možné se do hloubky věnovat všem okruhům, u kterých by si to autor přál a bylo nutné některé další možné aspekty práce vynechat.

Při úvaze o možných změnách současného řešení problému by autor příště zvolil místo JAX-RS implementace Jersey REST od Springu, a to z čistě důvodu ucelenějšího řešení a větší robustnosti a oblíbenosti Springu.

Jako možné rozšíření autor uvažoval o přidání příkladu použití s využitím JPA implementace Hibernate, jelikož je to žádaná technologie v dnešní době, avšak by bylo nutné i částečné odchýlení od tématu spolu se zabraným prostorem pro samotnou práci jako takovou.

Další možné rozšíření týkající se práce samotné by byla vlastní autentizace klienta v neobvyklé formě spolu s dodatečným maskováním autentizačních údajů pomocí hashovacích algoritmů a ochranou proti přeposílání zachycených paketů.

6 Závěr

Dle stanovených cílů autor postupoval dle metodiky. Při zpracování teoretických východisek práce bylo využito informací získaných studiem odborné literatury pro vymezení základních pojmů.

V teoretické části byl popsán programovací jazyk Java a jeho rozšíření Java EE, seznámení s Mavenem, vývojovým prostředím IntelliJ IDEA a aplikačním serverem Tomcat. Dále jsou popsány základní principy architektury REST s popisem specifikace Java řešení JAX-RS a jeho implementace Jersey. Teoretická část je zakončena popisem HTTP a HTTPS s hlavní částí zabývající se šifrováním SSL/TLS.

Pro vývoj aplikace bylo použito praktických znalostí spolu se znalostmi získanými z odborné literatury. Vlastní práce začíná úvodem při vytváření projektu ve vývojovém prostředí IntelliJ IDEA s použitím Mavenu a strukturou celého projektu pro jeho funkční sestavení. Navazuje konfigurace aplikačního serveru Tomcat pro SSL/TLS a nastavení Java EE Web Application deployment descriptoru se servletem spolu s šifrováním SSL/TLS. Následně se praktická část věnuje RESTovému API, tedy Jersey se samotným JAX-RS a logice přenosu, zpracování požadavků klienta a odpovědí serveru vlastním diagramem s popisem logiky a úskalí problémů, spolu s převodem Java objektů z/do formátu JSON a de/komprimace Contentu komunikace.

Praktická část je obsahuje jednoduché příklady použití demonstrující funkčnost a efektivitu celého řešení obsahující návrh a naprogramování RESTových služeb serveru a RESTového klienta klientské aplikace se zakončením příklady generování certifikátů pro server a klienta obsahující diagram struktury celého řešení.

7 Seznam použitých zdrojů

7.1 Knižní zdroje

HEROUT, Pavel. *Učebnice jazyka Java.* 5. rozšířené vydání České Budějovice: Kopp nakladatelství, 2010. ISBN 978-80-7232-398-2.

KEOGH, James Edward. *Java bez předchozích znalostí: průvodce pro samouky*. Brno: CP Books, 2005. ISBN 80-251-0839-2.

PECINOVSKÝ, Rudolf. *Myslíme objektově v jazyku Java: kompletní učebnice pro začátečníky*. 2. aktualiz. a rozš. vyd. Praha: Grada, 2009. ISBN 978-80-247-2653-3.

SCHILDT, Herbert. *Java 8: výukový kurz*. Přeložil Jakub GONER. Brno: Computer press, 2016. ISBN 978-80-251-4665-1.

7.2 Internetové zdroje

BĚHÁLEK, Marek. *Hibernate.* [online]. *FEI VŠB-TU Ostrava, 2005.* [cit. 2016-12-19]. Dostupné z: http://www.cs.vsb.cz/behalek/frvs/2005/java/hibernate/hibernate.html

BOUŠKA, Petr. *Protokol SSL/TLS - vypnutí slabých šifer a zabezpečení serverů*. [online]. samuraj.cz, 2014. [cit. 2016-12-18]. Dostupné z: http://www.samuraj-cz.com/clanek/protokol-ssl-tls-vypnuti-slabych-sifer-a-zabezpeceni-serveru/

ČÁPKA, David. *1.díl – Úvod do Java Enterprise Edition (JEE).* [online]. itnetwork.cz , 2014. [cit. 2016-12-18]. Dostupné z: http://www.itnetwork.cz/java/jee/java-enterprise-edition-uvod-do-jee-j2ee

HANEL, David. *Java EE platforma.* [online]. VŠE. [cit. 2016-12-13]. Dostupné z: https://java.vse.cz/jsf/chunks/ch02.html

HORDĚJČUK, Vojtěch. *Maven.* [online]. voho.cz [cit. 2016-12-18]. Dostupné z: http://voho.cz/wiki/maven/

JERSEY. *Jersey 2.25.1 User Guide.* [online]. Jersey.java.net. [cit. 2016-06-12]. Dostupné z: https://jersey.java.net/documentation/latest/index.html

JETBRAINS. *Intellij IDEA.* [online]. jetbrains.com [cit. 2016-05-05]. Dostupné z: https://www.jetbrains.com/idea/features/

MALÝ, Martin. *REST: architektura pro webové API.* [online]. Zdroják.cz, 2009. [cit. 2016-12-19]. Dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

PODLESÁK, Jakub. *Jersey: referenční implementace JAX-RS (JSR 311).* [cit. 2017-02-25]. Dostupné z: http://java.cz/dwn/1003/8290_REST-JAX-RS-Jersey-CZJUG.pdf

KUČERA, František. *Java na webovém serveru: píšeme REST API*. [cit. 2017-02-25]. Dostupné z: https://www.zdrojak.cz/clanky/java-na-webovem-serveru-piseme-rest-api/

KATEDRA INFORMATIKY. *RESTful API.* [online]. VŠB-TU Ostrava. [cit. 2016-12-18]. Dostupné z: http://wiki.cs.vsb.cz/images/f/fd/TAMZ-cv-11-CZ.pdf

RYBAŘÍK, Jan. *Co je API a jak jej používat ve vašem podnikání.* [online]. Onemark, 2015. [cit. 2016-12-19]. Dostupné z: http://www.onemark.cz/clanky/co-je-api-a-jak-jej-pouzivat-ve-vasem-podnikani

8 Seznam ilustrací

8.1 Seznam obrázků

8.2 Seznam tabulek

9 Přílohy