VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2017

Petr Kuča

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Petr Kuča**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Olivka, Ph.D.**

Konzultant bakalářské práce: Mgr. Zdeněk Dřizga

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. dubna 2017 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

Tieto Czech s.r.o.
28. října 3346/91
702 00 Ostrava - Moravská Ostrava
IČO 64608051 DIČ CZ64608051

V Ostravě 28. dubna 2017 .....

Rád bych na tomto místě poděkoval společnosti Tieto Czech s.r.o., za umožnění absolvování individuální odborné praxe, zvláště pak Mgr. Zdeňku Dřizgovi. Dále také kolegům, kteří se mnou spolupracovali.

Mé poděkování také patří mému vedoucímu bakalářské práce Ing. Petru Olivkovi, Ph.D. za konzultace a věcné připomínky k vypracování.

Abstrakt

Tato Bakalářská práce popisuje mé působení ve společnosti Tieto Czech, kde jsem absolvoval individuální odbornou praxi na pozici Java Software Developer.

V tomto dokumentu popisuji společnost samotnou spolu s historií, interní aplikace se kterými jsem pracoval a úkoly, které jsem dostával a řešil. Také popisuji některé technologie a strategie jenž jsem používal. Dále se detailněji věnuji deployment problematice a Single Sign-On implementaci pomocí moderních autentizačních protokolů. Na konci práce představuji projekt Locker, jeho cíle, návrh a vzhled. V závěru se zmiňuji o zkušenostech a vědomostech, které jsem během praxe nabyl, použité znalosti získané výukou na vysoké škole a shrnutí praxe celkově.

Klíčová slova: Bakalářská práce, Tieto, Java, Spring Boot, Git, Deployment, Locker, Web

Abstract

This Bachelor thesis describes my activities in Tieto Czech company, where I completed the individual professional practice like Java Software Developer.

In this work I describe the company, along history, internal applications which I met and my tasks which I have got assigned to solve. Also I describe used technologies and strategies. I focus on the details of deployment and Single Sign-on implementation with modern authentication protocols. I introduce Locker project, his goals, proposal and appearance. In conclusion I mention the experiences and knowledges that I gained during my practice, applied skills obtained from the university and a summary of practice.

Key Words: Bachelor thesis, Tieto, Java, Spring Boot, Git, Deployment, Locker, Web

Obsah

Seznam použitých zkratek a symbolů

XML	–	Extensible Markup Language
OS	–	Operation system
LDAP	–	Lightweight Directory Access Protocol
SPN	–	Service Principal Name
JSF	–	JavaServer Faces
HTTP	–	Hyper Text Transfer Protocol
REST	–	Representational state transfer
HTML	–	HyperText Markup Language
CSS	–	Cascading Style Sheets
SSL	–	Secure Socker Layer
TLS	–	Transport Layer Security
SSO	–	Single Sign-on
KDC	–	Key Distribution Center
API	–	Application Programming Interface
JSON	–	Java Script Object Notation
WAR	–	Web Application Archive

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Při výběru bakalářské práce se naskytla možnost absolvovat odbornou praxi v IT firmě. Tuto možnost jsem využil a jako firmu jsem si vybral Tieto v Ostravě. Po úspěšném pohovoru jsem do firmy nastoupil na konci druhého ročníku bakalářského studia. Hlavními důvody pro výběr odborné praxe bylo nabytí praktických zkušeností v odvětví programování a vývoje informačních systému. Další nesmírnou výhodou odborné praxe je seznámení se s mnoha pracovními postupy, kulturou ve společnosti a kolektivem zkušených lidí v oboru.

Očekával jsem, že se seznámím s prací v IT firmě, protože jsem zatím neměl s žádnou takovou prací zkušenosti, kromě studia ve škole. Také, že budu součástí vývoje komplexních informačních systémů, vývoje v týmu a tak podobně. V neposlední řadě mě zaujalo poznat severoevropskou firmu, kde se mluví výhradně v anglickém jazyce a která zaměstnává více něž 13 000 zaměstnanců.

Na začátku práce bych chtěl uvést krátký popis firmy, pozici na které jsem pracoval a interní projekty. V další části části bych chtěl popsat zadané úkoly, které jsem řešil a věnovat část textu technologiím s jimiž jsem pracoval. Snažím se zaměřit na praktické zkušenosti, které jsem získal a podělit se o mé postupy. Na konci uvádím své poznatky z práce na jednotlivých projektech a technologie použité při vypracovávání úkolů. V závěru práce shrnuji své pocity, nabyté zkušenosti na odborné praxi a zhodnocení mého očekávání.

2 O společnosti Tieto Czech s.r.o.

2.1 Zaměření firmy

Tieto je největší severoevropská společnost poskytující komplexní IT služby. Poskytuje služby v oblasti vývoje produktů pro firmy působící v odvětví komunikací a integrovaných technologií. Na základě svých znalostí, technologických vizí a inovativního myšlení se společnost aktivně snaží inspirovat a zapojit své zákazníky do hledání nových způsobů, jak zefektivnit jejich podnikatelskou činnost.

Tieto spojuje globální možnosti s přítomností na lokálních trzích, a klade velký důraz na profesní růst a výsledky. Společnost se svým sídlem ve finských Helsinkách zaměstnává přes 13 000 expertů z téměř více než 20 zemích světa. Obrat činí přibližně 1,5 miliard eur [7].

Tieto Czech s.r.o.

Do České republiky společnost Tieto vstoupila v roce 2001 a v roce 2004 otevřela své softwarové centrum v Ostravě. S více než 2 200 zaměstnanci je jedním z největších zaměstnavatelů v oblasti IT služeb v České republice a největším v Moravskoslezském kraji. Z hlediska počtu kmenových zaměstnanců je česká pobočka třetí největší pobočkou Tieto korporace na světě (první dvě místa zaujímají mateřské země Finsko a Švédsko.)

V roce 2012 se zaměstnanci společnosti Tieto Czech, kteří doposud v Ostravě pracovali ve čtyřech různých budovách přesídlili do nově vybudovaného komplexního sídla v centru Ostravy [12].

2.2 Pracovní pozice

Před nástupem do firmy jsem musel projít třemi koly přijímacího řízení. V prvním kole jsem se setkal s personální pracovnicí a předmětem schůzky bylo probírání mého dosavadního vzdělání, obecných zkušeností s IT odvětvím a jazykových znalostí. Ve druhém kole jsem byl přizván na pohovor s manažerem Mgr. Zdeňkem Dřizgou, kde jsme se bavili konkrétně o zkušenostech v programovacím jazyce Java, vývoji aplikací, projektech na kterých jsem pracoval apod. V posledním kole jsem již dostal nabídku na pozici Software Developer. Mé zaměření v průběhu praxe bylo vyvíjení v Javě a Spring Frameworku. Kromě toho jsem se krátce zabýval vývojem stránek v JavaScriptu. Pro práci v týmu naše oddělení používalo verzovací systém Git a systém JIRA pro správu úkolu *(tzv. tasks)*. Protože Tieto je mezinárodní společnost, komunikuje se zde hlavně v anglickém jazyce. Taktéž všechny aplikace, dokumentace kódů, prezentace, manuály musí být psány v angličtině. Právě až na této pozice jsem využil znalosti angličtiny v praxi.

Obrázek 1: Ostrava Tieto Towers [7]

2.3 Interní aplikace

Po nástupu do firmy mi byly představeny aplikace, na kterých se pracuje nebo které již jsou v provozu.

Place Reservation slouží pro rezervaci míst zaměstnancům a stážistům v kancelářích ve firmě. Zaměstnanci mají statické místo *(static place)*, stážisté a externisté si rezervují sdílené místo *(shared places)* dopředu. V nejnovější verzi Place Reservation 3 je také umožněno rezervovat parkovací místa, meeting room, či číst data ze senzorů (teplota, vlhkost vzduchu apod.)

Hardware Index je určená pro skenování firemních počítačů a ukládání informací o hardware do webové aplikace, ta pak slouží manažerům jako přehled.

My Internship slouží pro zaznamenávání odpracovaných hodin stážistům. Kromě hodin se zde zaznamenává i typ směny a komentář. Data z této aplikace se poskytují HR a dalším oddělením firmy.

Code Storm slouží pro sdílení a poskytování kódu. Aplikace je rozdělena na tři části: *Modules* - obsahuje databázi kódů, které představují zásuvný modul, čili dají se jednoduše vložit do projektu a s určitou konfigurací využívat, sám o sobě je bezpředmětný. *Component* - představuje databázi komponent, které lze využít při vývoji aplikací. Zde můžou být již funkční celky aplikace vhodné ke kustomizaci. *Snippet* - obsahuje jen útržky kódu, je určena jen pro sdílení mezi dvěma uživateli.

3 Zadané úkoly

V průběhu mé praxe jsem dostával různé úkoly. Stážisté pracují na nekomerčních projektech, to jsou projekty, které nejsou určené pro koncové zákazníky, ale jen pro firemní užití. Tyto aplikace jsou umístěny na serverech, které mají stážisté k dispozici a o které se sami starají. Jedním z mých úkolů bylo právě dělat údržbu těchto serverů a provádět nasazování aplikací na ně. Také jsem se podílel na vývoji aplikace Locker a rozvoji My Internship a Place Reservation projektů.

3.1 Časová náročnost

Podstatnou část své praxe jsem strávil vyvíjením webových Java aplikací ve Spring Frameworku a problematikou deploymentu do produkčního prostředí. Další větší část praxe jsem věnoval implementaci zabezpečení OAuth 2 protokolu ve Springu pro autentizaci a následně propojení s Kerberos síťovým autentizační protokolkem. Zbylou část praxe jsem se setkal nebo pracoval s Hibernate (JPA), StringTemplate, SMTP, Linuxovými systémy, LDAP protokolem, Jboss Wildfly aplikačním serverem, JavaScript knihovnou React, konfigurováním SSL připojení, NGINX webovým serverem, Apache webovým serverem, Piwik webovým analyzátorem, JUnit testy ve Spring frameworku, GIT verzovacím systémem, Jenkins software pro automatizaci, Bash skripty, SSH, MongoDB, PostgreSQL, Apache Maven.

3.2 Deployment

Deployment *(v překladu nasazení)* je souhrn všech činností, které instalují aplikaci do určitého prostředí (*například na server).* V našem týmu se prováděl deploy přibližně jednou až dvakrát za měsíc. Většina našich aplikací běží na Tomcat aplikačním serveru, vyjma My Internship, která beží na Jboss Wildfy serveru. Aplikace se skládají ze dvou základních částí:

backend Výkonná část aplikace, kterou uživatel nevidí. Může mít třívrstvou architekturu, může být rozdělen do více komponent.

frontend Část aplikace sloužící pro interakci s uživatelem, čili část aplikace kterou uživatel vidí a kde zadává data.

V závislosti na použitých technologiích při vývoji se proces deploye může lišit. Obecně lze ale shrnout do dvou kroků, a to deploy backednu a frontendu. Před samotným procesem nasazení je nutné vytvořit instanci aplikačního serveru.

3.2.1 Vytváření instance Apache Tomcat server

Před samotným deployem je potřeba vytvořit prostředí pro běh aplikací, tím se myslí vytvořit instanci aplikačního serveru. Jak již bylo zmíněno, používáme open source webový server a servlet kontejner Apache Tomcat ve verzi 8.5.11. Při vytváření instance je potřeba stáhnout balíček a ten rozbalit a umístit na server. Jelikož je multiplatformní, je k dispozici ve formátech:

zip, tar.gz případně *exe*. Pro spuštění serveru je potřeba mít nainstalovaný balíček Java SE 7 nebo vyšší. Konfigurace Tomcatu je založena na úpravě *xml* souborů ve složce *conf*. Konfiguroval jsem soubory *server.xml* - pro nastavení služeb a portů, *tomcat-users.xml* - pro nastavení rolí a vytvoření uživatelů [2]. Příklad vytváření uživatelů a rolí lze vidět níže.

```
<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users ...>
   <!-- Create role for GUI manager on url localhost:8080/manager/html -->
   <role rolename="manager-gui" />
   <!-- Create role for using manager script, can be use for automation deploy
       (i.e. Jenkins) -->
   <role rolename="manager-script" />
   <!-- Create custom role -->
   <role rolename="role1"/>
   <!-- Create user admin with access to GUI manager -->
    <user username="admin" password="ctuVhuCQg4TUWpZz" roles="manager-gui" />
   <!-- Create user deployer, can be use like service account for Jenkins -->
    <user username="deployer" password="ctDvbKMOEgwe" roles="manager-script" />
</tomcat-users>
```

Výpis 1: Příklad konfigurace souboru tomcat-users.xml [2], *upraveno*

Jako výchozí port má Tomcat nastavený port 8080. V případě, že nám poběží na serveru více instancí Tomcatu, je potřeba ho změnit, jinak by došlo ke konfliktu. V konfiguračním souboru *server.xml* v elementu `Connector` lze výchozí port změnit.

```
<?xml version="1.0" encoding="UTF-8"?>
    <Server port="8005" shutdown="SHUTDOWN">
        <Service name="Catalina">
        ...
        <!-- Configuration port number -->
        <Connector port="8080" protocol="HTTP/1.1"
            connectionTimeout="20000"
            redirectPort="8443" />
        ...
</Server>
```

Výpis 2: Změna portu Tomcat serveru [2], *upraveno*

Tomcat server se spouští pomocí skriptu *startup.sh*, případně *startup.bat*, ve složce *bin*. O tom jestli se server podařilo spustit se dozvíme z logů *catalina.out* (složka *logs*). Do tohoto logu se také mimo jiné zapisují logy a výjimky, které vznikají jak při deploy, tak za běhu aplikace. Pro vypnutí serveru použijeme skript *shutdown.sh* [2].

Kromě konfigurace, kterou jsem popsal výše, je také klíčové vytvořit pro každý aplikační server vlastního uživatele v OS pod kterým se bude server spouštět, omezit přístup do kořenové složky Tomcat serveru ostatním uživatelům a vymezit mu také diskovou kvótu. Vymezení diskové kvóty považuji za nezbytné, neboť několikrát se stalo, že v aplikaci došlo k selhání konektivity mezi komponenty. Tato situace nebyla programátorsky ošetřena) a vypisovala se několikrát za sekundu výjimka do logů. Než se na takovou chybu přijde a opraví se, může se mezitím zaplnit celý disk a serveru dojde místo na disku, čímž se stanou nefunkční všechny aplikace, které na něm běží.

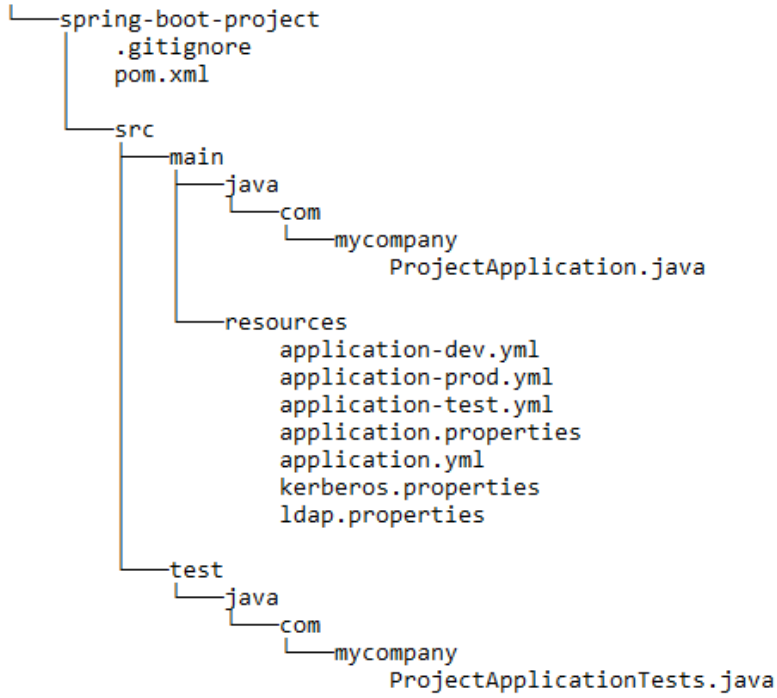
3.2.2 Deploy backednu

Deploy backednu se provádí po ukončené fázi vývoje aplikace. Lze ho shrnout do 4 kroků:

1. Korektně nastavit všechny properties soubory v projektu.
2. Provést Maven build projektu v produkčním profilu. V případě úspěšného buildu vznikne soubor WAR.
3. Na produkčním serveru vytvořit novou prázdnou databázi pokud již neexistuje. Aplikace obsahuje skript *schema.sql*, který se provede při spuštění a vytvoří, nebo aktualizuje databázové schéma.
4. Nahrát WAR soubor na server.

Protože aplikace sama o sobě spolupracuje s mnoha dalšími systémy jako je LDAP (nebo Actice Directory), SPN pro Single Sign-On, připojení k databázi apod., je potřeba udržovat adresy, jména a hesla servisních účtů a další údaje k těmto službám. Udržování těchto údajů přímo ve zdrojovém kódu by bylo neefektivní a obtížně vyměnitelné. Spring Boot projekt má danou strukturu projektu a umožňuje udržovat tyto údaje přehledně v properties nebo YAML souborech ve složce *src/main/resources*, jak můžeme vidět na obrázku 2. Properties představují nastavení ve formátu `klíč=hodnota`, jeden řádek souboru představuje jedno konkrétní nastavení.

Klíčový je souboru *application.yml* kde se nastavuje aktivní profil, který vytváří různé prostředí pro běh aplikace. Například pokud bude aktivní *dev* (development) profil může aplikace bežet v debug módu (což umožňuje bohatější výpisy logů a usnadní tak vývoj), používat jinou databázi atd. Konkrétní ukázku konfigurace v YAML souboru můžeme vidět níže [8].



Obrázek 2: Ukázka struktury Spring boot projektu

```
spring:
  datasource:
    driver-class-name: org.postgresql.Driver
    url: jdbc:postgresql://localhost:5432/placereservation-core
    username: username
    password: password
    initialize: false
    max-active: 3
    initial-size: 2
```

Výpis 3: Ukázka konfigurace v *application-dev.yml* pro Spring Boot

Jak si můžeme všimnout z výpisu 3, properties se v YAML souborech zadávají pomocí syntaxe `klíč: hodnota`. Jednotlivé property se shlukují do daných namespace. Například nastavení týkajících se připojení k databázi se nachází v namespace **spring** a sub-namespace **datasource.**

3.2.3 Deploy frontendu

Deploy frontendu závisí na použité technologii. Frontend aplikace může být součásti WAR souboru, který se nahrává na Tomcat server. Například u technologie JSF pro webové aplikace, nebo

může být fyzický oddělený od aplikačního serveru, pouze s ním komunikovat přes REST rozhraní HTTP protokolem (např. při použití technologie React JavaScript). V prvním případě je deploy frontendu součástí deploye backendu, v druhém případě je potřeba učinit několik kroků. V poslední době se náš tým zaměřuje na vývoj frontendu právě v Reactu. Produktem takového vývoje v Reactu jsou dynamické webové stránky v HTML, JavaScriptu a CSS.

3.2.3.1 Webové servery Pro provoz takových dynamických webových stránek je potřeba webový server. Na trhu existují dva nejznámější multiplatformní produkty: **Apache HTTP server** a **Nginx**. Ve firmě používáme oba v Unix verzích, kde na testovacím serveru běží verze Apache 2 a v produkčním prostředí pak Nginx. Poslední verze Apache a Nginx poskytují srovnatelnou funkcionalitu. Kromě drobných odlišností se hlavně liší ve formátu konfigurace. Apache konfigurace připomíná více značkovací jazyk XML, kdežto Nginx připomíná více formát programovacího jazyka C, viz obrázek 3.

```
############################
### Nginx configuration #####
############################

server {
  listen 80 default_server;
  listen [::]:80 default_server;

  root /usr/share/nginx/html;
  index index.html index.htm;

  server_name myserver.company.com;

  location / {
    try_files $uri $uri/ /index.html;
  }

  location /endpoint-one/ {
    proxy_pass http://127.0.0.1:8000/backend/;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
  }
}
```

```
##############################
### Apache 2 configuration #####
##############################

<VirtualHost *:80>
  ServerAdmin webmaster@localhost
  DocumentRoot /var/www

  <Directory /var/www/>
    Options Indexes FollowSymLinks MultiViews
    AllowOverride None
    Order allow,deny
    allow from all
  </Directory>

  <Location "/endpoint-two/">
    ProxyPass "http://127.0.0.1:8000/backend/"
    SetEnv force-proxy-request-1.0 1
    SetEnv proxy-nokeepalive 1
  </Location>
</VirtualHost>
```

Obrázek 3: Srovnání Nginx a Apache 2 konfiguračních souborů [3][13], *upraveno*

Postup pro deploy na produkční server je tedy následující. Nejdříve je potřeba umístit vytvořené a sestavené webové stránky v Reactu do složky na serveru, vhodná je k tomu složka `/var/www/<název aplikace>`. Důležitá je správná konfigurace Nginxu. Protože náš tým má k dispozici jen jeden produkční server a několik DNS domén, které jsou na něj registrovány, využíváme virtuálních hostů, které umí právě Nginx nebo Apache vytvořit. V praxi to pak funguje

tak, že při pokusu o přístup k aplikaci pomocí internetového doménového jména pomocí webový prohlížeč, jdou všechny požadavky na jeden fyzický stroj a Nginx rozhodne podle hlavičky `Host` v HTTP protokol, na který virtuální host bude požadavek směrován. Aby směrování fungovalo správně je klíčové nastavit atribut **server_name: <název internetové domény>;**. Podle [3] mezi další atributy, které je potřeba nastavit, patří:

listen *<port>;* Určuje port na kterém Nginx naslouchá, typicky 80 respektive 443 (HTTP/S).

root *<cesta k webu>;* Absolutní cesta k adresáři se zdrojovými kódy webu.

index *<index.{html|php|xhtml...}>;* Definuje množinu výchozích vstupních bodů webu.

ssl_certificate *<cesta ke .crt souboru>;* Určení cesty k podepsanému certifikátu, formát CRT. Certifikát by měl být podepsaný platnou certifikační autoritou.

ssl_certificate_key *<cesta ke .key souboru>;* Určení cesty k soukromému klíči certifikátu, formát KEY.

ssl_protocols *<{TLSv1|TLSv1.2|...}>;* Definuje množinu omezení na připojení pouze za podpory určených verzí SSL/TLS.

Poznámka: Pro bezpečnost je důležité zajistit, aby žádný nepovolaný uživatel neměl přístup k .crt souboru, .key souboru ani dalším souborům certifikátům, nebo je jinak modifikoval. Jeden z dobrých postupů může být vytvoření složky např: `ssl` *přímo v konfiračním adresáři Nginx a do něj umístit všechny certifikáty a soubory tvořící certifikát. Na složku pak aplikovat příkaz:*

```
sudo chmod -R /etc/nginx/conf.d/ssl 000
```

Tímto příkazem upravíme přístupová práva k adresáři tak, že žádný uživatel nebude mít práva ho číst, zapisovat do něj, nebo ho procházet (pokud se pohybujeme v Unix systému).

Veškeré konfigurace se zapisují do souboru, jak můžeme vidět na obrázku 3, a ukládají do konfigurační složky. Nginx při startu prohledá danou složku a aplikuje nastavení. Umístění složky s konfiguračními soubory pro virtuální hosty se může lišit v závislosti na distribuci systémů. Například pro CentOS je složka umístěna v `/etc/nginx/conf.d/`, ale v Ubuntu Xenial Xerus je to `/etc/nginx/sites-enable/`. Pro zachování přehlednosti a škálovatelnosti je dobré vytvářet pro každé doménové internetové jméno virtuálního hosta konfigurační soubor zvlášť, tedy vyvarovat se jednomu rozsáhlému souboru kde budou obsaženy všechny. Pokud máme například aplikaci Place Reservation a k ní internetové doménové jméno places.int.tieto.com, která navíc používá SSL, vzniknou nám tak dva konfigurační soubory: **places.conf** a **ssl-places.conf**. Jak názvy souborů napovídají places.conf bude sloužit jako nastavení pro web http://places.int.tieto.com a ssl-places.conf pro https://places.int.tieto.com. Protože se ale uživatel musí do aplikace vždy nejdříve přihlásit, bude vždy hned přesměrován na https://places.int.tieto.com. Toho lze docílit konfigurací viz výpis 4. Většina prohlížečů provede po obdržení HTTP status kódu 301 automatické přesměrování na určenou URL pokud je k dispozici.

```
...
Location / {
    # Represent returned HTTP status code 301 - moved permanently
    # And target URL with all URL parameters
    return 301 https://places.int.tieto.com$request_uri;
}
...
```

Výpis 4: Nginx redirect na HTTPS [3], *upraveno*

3.3 Single Sign-on

Jedním z úkolů, kterým jsem se delší dobu zabýval, byla implementace SSO do již fungujících aplikací. Požadavek vznikl od mého manažera na základě zefektivnění využití našich aplikací, protože se uživatel vždy přihlašuje jednotným uživatelským jménem a heslem. Cíl byl tedy takový, aby po přihlášení do firemní domény pomocí pracovního počítače již nebylo potřeba zadávat stejné přihlašovací údaje do jednotlivých aplikací, ale autentizace probíhala na pozadí. Po konzultaci se zkušeným kolegou a na jeho doporučení jsem zvolil SSO pomoci protokolu Kerberos. Dle [5] jsou níže uvedeny hlavní důvody:

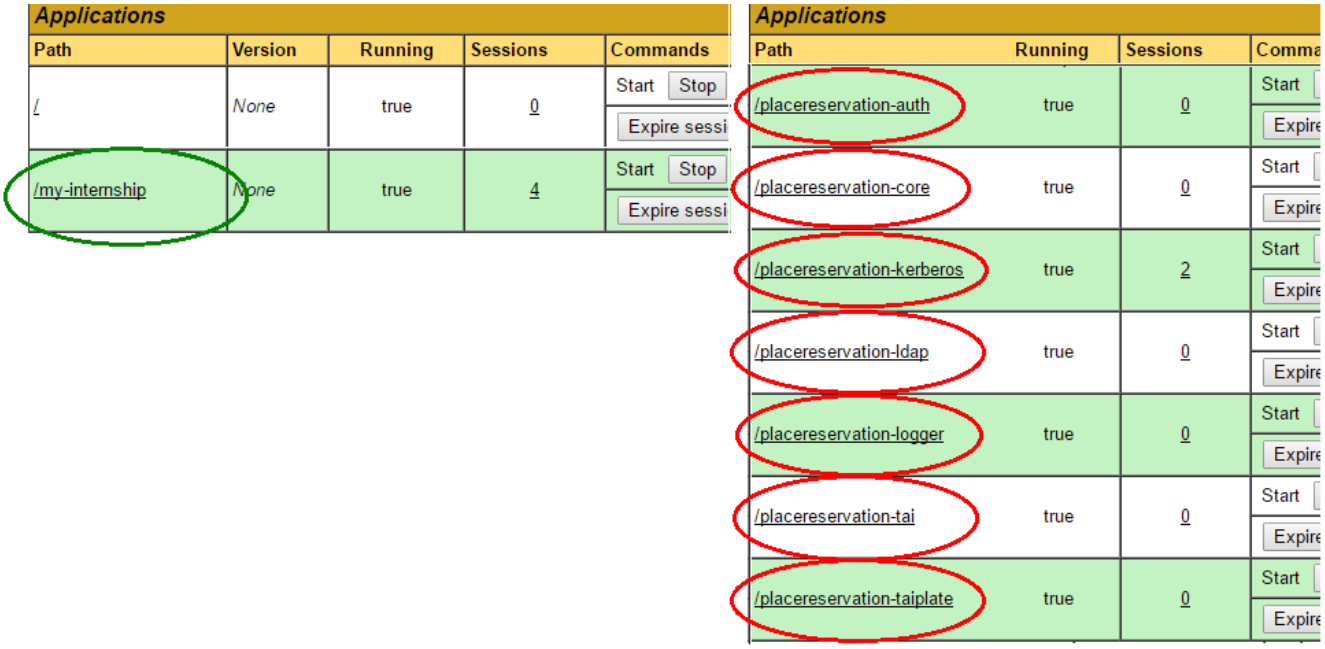
- podpora ze strany firmy,
- integrace v OS Windows[1] *(nejčastěji používán ve firemních PC)*,
- podpora ve Spring Frameworku *(Spring Security Kerberos Extensios)*.

Kromě samotné implementace SSO pomocí Kerberos protokolu, bylo potřeba ještě vyřešit jeden problém. Některé naše aplikace jsou postaveny na architektuře microservice, tzn. že se jedná o aplikaci složenou z komponent, které mezi sebou komunikují. Srovnání mezi microservice aplikací a klasickou monolitickou aplikací z pohledu aplikačního serveru můžeme vidět na obrázku 4. Protože jednotlivé microservice musí být samy o sobě zabezpečeny, využíváme k tomu protokol OAuth 2. Bylo tedy potřeba sloučit funkci protokolů OAuth 2 a Kerberos.

3.3.1 Kerberos

Předtím, než se začnu věnovat konkrétně SPNEGO, je potřeba popsat Kerberos protokol samotný. Jde o autentizační protokol, který používá kryptografii pro bezpečné ověření klienta přes nezabezpečenou síť. Kerberos protokol ve verzi 5 popisuje **RFC 2478**. Jeho princip spočívá v tom, že se uživatel neověřuje vůči resource serveru (tam, kde má zájem o službu), ale vůči

[1]Microsoft Windows využívá Integrated Windows Authentication, která používá protokol Kerberos a je primárně spojovaná s webovými prohlížeči, což dovoluje využívat SSO pro webové aplikace



Obrázek 4: Srovnání mezi monolitickou *(zeleně)* a microservices architekturou *(červeně)* z pohledu aplikačního serveru. Jeden řádek přestavuje jeden WAR soubor

centrálnímu autentizačnímu serveru *(tzv. KDC)*. KDC obsahuje databázi šifrovacích klíčů všech klientů, kteří se vůči němu autentizují. Poté, co je uživatel ověřen, získává přístup ke službě. Celý proces ve stručnosti popíšu na příkladě. Ještě je ale potřeba dodat, že existence takové prostředníka při autentizaci přináší své výhody i nevýhody. Centrální autentizační server zvyšuje bezpečnost, může poskytovat autentizaci pro více aplikací zároveň, používá šifrování atd. Na druhou stranu se může jednat o tzv. *single point of failure* při komunikaci, pokud se KDC stane z nějakého důvodu nedostupným *(chyba systému, výpadek sítě, apod.)* způsobí to výpadek celého systému. Pro ošetření této situace můžeme zajistit spolupráci více KDC. Při výpadku převezmu roli centrálního prvku jiný dostupný KDC (problém se synchronizací, redundantnost). KDC musí být také velmi dobře zabezpečen jak na fyzické tak síťové úrovni [5].

3.3.1.1 Průběh autentizace

Průběh autentizace můžeme najít v literatuře [5]. Ve zkrácené podobě ho můžeme popsat takto.

1. Autentizace uživatele vůči KDC.
   (a) Uživatel zadá své přihlašovací údaje (například jméno a heslo pro přihlášení do domény).
   (b) Nad heslem se provede jednosměrná hash funkce - tím získáme `secret key`.
   (c) Potřebné uživatelské údaje se posílají na Autentizační server, který je součástí KDC *(heslo ani otisk hesla se neposílá)*.

2. Získání Ticket-Granting Ticketu.

   (a) Autentizační server ověří uživatele a vytvoří si `secret key` z jeho hesla *(heslo uživatele má uloženo ve své databázi).*

   (b) Uživatel obdrží od KDC `session key`, který je zašifrován pomocí `secret key`.

   (c) Uživatel obdrží také Ticket-Granting Ticket - zkráceně `TGT`, který je zašifrovám pomocí `KDC secret key` (uživatel ho nemůže dešifrovat).

   (d) Uživatel umí dešifrovat jen `session key`, tím došlo k prokázání jeho totožnosti.

   (e) `TGT` dále slouží k identifikaci uživatele, obsahuje jeho uživatelské údaje a informace o ticketu, má omezenou platnost.

3. Získání Service Ticket

   (a) Pro přístup ke službě potřebuje uživatel získat `service ticket`.

   (b) Uživatel pošle žádost na Ticket Granting Service - zkráceně `TGS`, ve dvou zprávách. Tyto dvě zprávy tvoří credentials uživatele).

   (c) `TGS` je také součást KDC.

   (d) V první zprávě se posílá již získané `TGT` a ID služby na kterou chceme přístup - Service Principal Name, zkráceně `SPN`.

   (e) Druhá zpráva se nazývá Authenticator, obsahuje identifikaci uživatele a časové razítko - šifrováno pomocí `session key`.

   (f) `TGS` dešifruje příchozí `TGT` od uživatele, z kterého se dozví `session key` - to umožní dešifrovat druhou Authenticator zprávu a ověřit totožnost.

   (g) `TGS` odesílá odpověď uživateli rozdělenou na dvě části: (`klientskou` a `serverovou`).

   (h) `Klientská část` obsahuje `session key` - ten slouží pro komunikaci uživatele a serveru se službou.

   (i) `Serverová část` obsahuje `service ticket` - identifikace uživatele, adresa, platnost ticketu, session key. Data jsou šifrována pomocí `secret key`).

4. Oveření služby na straně serveru.

   (a) Na server, kde se chceme pomocí SSO přihlásit se posílá `service ticket` - šifrovaný pomocí `secret key`.

   (b) A také Authenticator - uživatelova identifikace a časové razítko - šifrované pomocí `session key`.

   (c) Server dešifruje `service ticket`, následně authenticator zprávu pomocí získáného `session key`.

   (d) Po oveření posílá zašifrovanou zprávu, která obsahuje časové razítko od klienta + 1, šifrovaná pomocí `session key`.

5. Získání session

   (a) Uživatel dešifruje potvrzení a zkontroluje časové razítko, jestli odpovídá - tím došlo k úspěšné autentizaci.

   (b) Oveření proběhlo obousměrně.

   (c) Dojde k vytvoření a uložení session.

3.3.2 SPNEGO protokol

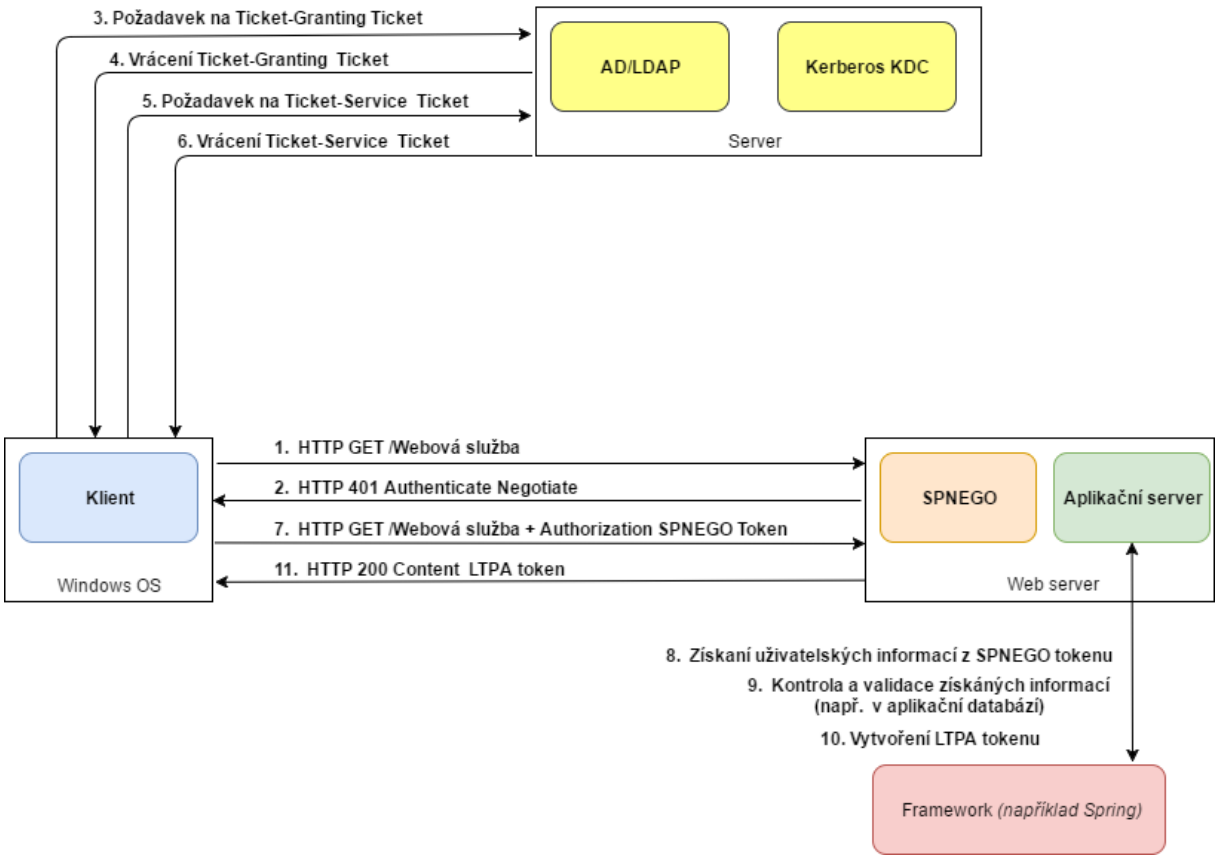
Simple and Protected GSSAPI Negotiation Mechanism zkráceně `SPNEGO`, je protokol, který definoval Microsoft a je popsán v **RFC 2478**. Dle [4] SPNEGO protokol slouží pro:

- vyjednávání, zda se použije Kerberos (pokud web server podporuje SSO), případně jiný protokol,
- výměnu informací o autentizaci, *tzv. credentilas*, mezi klientem a web serverem pomocí protokolu HTTP.

Průběh procesu protokolu lze znázornit obrázkem 5. Uživatel chce přistoupit na chráněný webový obsah, ten vyžaduje autentizaci a web server umožňuje SSO. Pokud není uživatel dosud přihlášený, webový server odpoví HTTP kódem 401 (Unauthorized) spolu s hlavičkou `Authorization: Negotiate`. Uživatel, který má nakonfigurovanou podporu IWA rozpozná negotiate hlavičku, rozparsuje URL, čímž získá internetové doménové jméno, tzv. host name. Na základě získaných atributů zašle požadavky na získání Kerberos tiketů, které po úspěšném získání vloží do SPNEGO tokenu. Uživatel nyní vyšle na základě 401 Unauthorized a hlavičky negotiate nový HTTP požadavek a SPNEGO token vloží do hlavičky s prefixex Authentication: Negotiate. Hlavička bude tedy vypadat takto: `Authorization: Negotiate <SPNEGO token>;`. Server rozpozná poslanou hlavičku, vyextrahuje Kerberos ticket a provede jeho ověření. Poté co je dokončeno ověření, provede se autorizace a uživateli se povolí přístup. V opačném případě se vrátí HTTP odpověd 403 - Access Denied [4] [5].

3.3.3 OAuth 2 protokol

Jak již bylo řečeno, aplikace, které využívají architekturu microservices, jsou zabezpečeny pomocí protokolu OAuth 2. Jedná se o moderní autorizační protokol pro zabezpečení webových a také RESTových služeb. Cílem je poskytnout přístup k chráněným webových zdrojům *(resources)* a službám třetí straně, aniž by bylo nutné poskytnout přímo přístupové údaje. Jako příklad si můžeme uvést uživatele, který využívá služeb Google a potřebuje poskytnout přístup ke svým e-mailům na Gmail účtu jiné aplikaci. Jednou z možností je zadat přímo uživatelovy přístupové údaje, kterými se přihlašuje, tím ale umožní aplikaci prakticky neomezený přístup ke svému účtu a dalším službám Google, což není zcela v pořádku. Další možností je využít služeb OAuth 2 a poskytnout přístup na základě Bearer tokenu (dále jen tokenu). Uživatel vytvoří

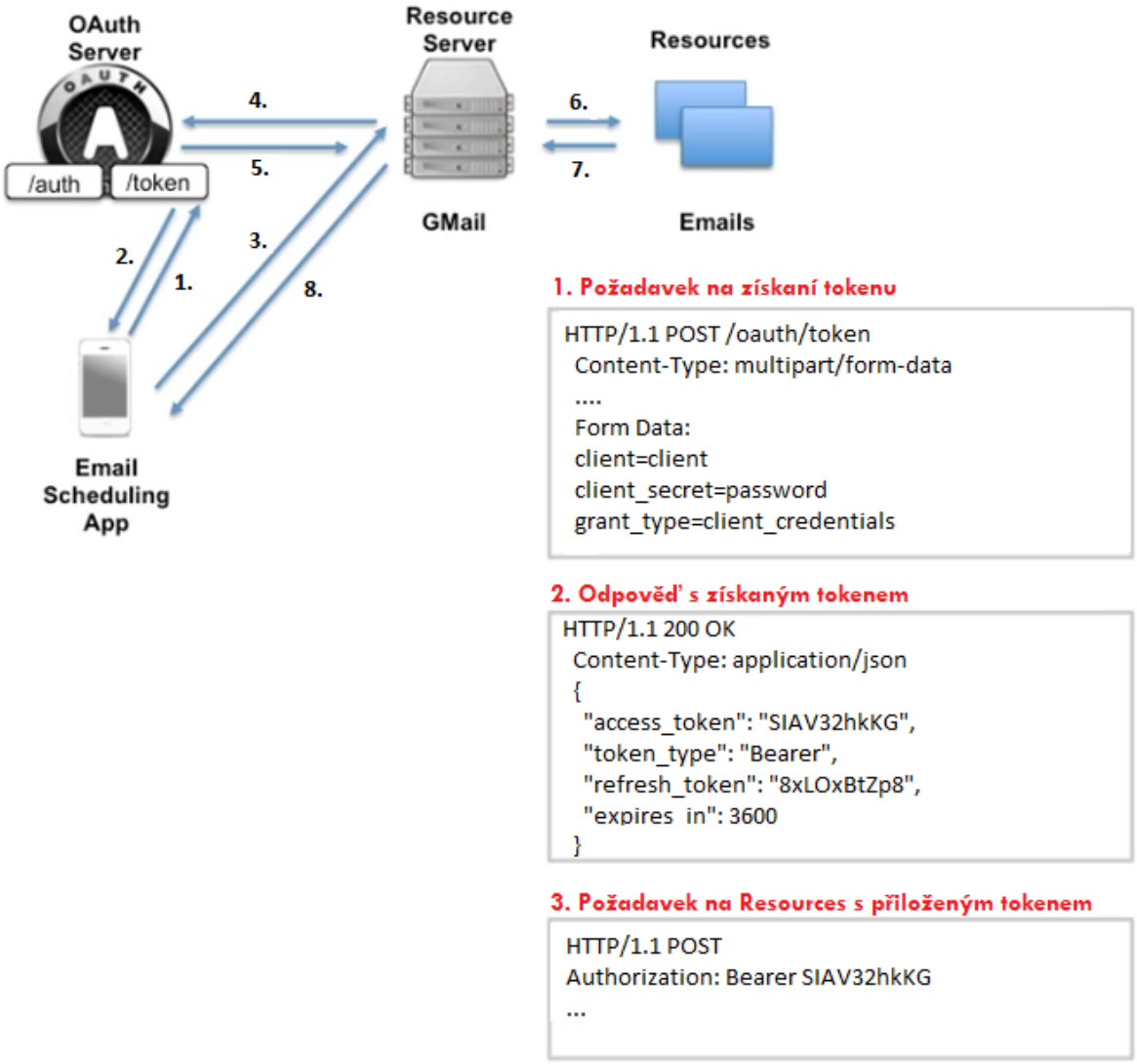


Obrázek 5: Znázornění procesu SPNEGO a Kerberos protokolů [4]

klienta na autorizačního serveru, určí požadovanou službu a definuje tzv. scope *(pouze pro čtení, pouze pro zápis nebo čtení i zápis)*. Pomocí tohoto klienta se zmiňovaná aplikace autorizuje a obrží token, kterým získá omezený přístup pouze k emailovému účtu. Token má navíc omezenou dobu platnosti, kterou můžeme zjistit z pole `"expires_in"`. Udávaná hodnota je v sekundách. OAuth 2 protokol je již velmi rozšířen a podporován známými službami, jako například GitHub, Facebook, Twitter, Google a další. Jedna z nevýhod je různorodost implementace mezi společnostmi, kde se může výrazně lišit poskytované API. Protokol je popsán v **RFC 6749**. Příklad který jsem popsal výše, znázorňuje obrázek 6. Uvažujeme, že veškerá komunikace musí probíhá šifrovaným spojením, aby nebylo možné odposlechnout na sítí údaje klienta nebo přístupový token [11].

3.3.4 Implementace SSO

Po nastudování jednotlivých protokolů jsem se pustil do implementace SSO pro aplikaci Place Reservation. Na úplném začátku bylo potřeba vytvořit servisní účet v doméně, který musí mít vyplněný Service Principal Name (SPN) s danou službou (jeden záznam pro každou internetovou doménu) ve tvaru `HTTP/<internetová doména>`. Například pro aplikaci Place Reservation to bude HTTP/places.int.tieto.com) Taková žádost se v Tietu vyřizuje přes Service Desk vytvoře-



Obrázek 6: Znázornění komunikace OAuth 2 protokolu [6]

ním ticketu a její vyřízení trvá přibližně 3-7 dní. Na řadu se dostala samotná úprava zdrojového kódu. Spring Boot umožňuje rozšíření webové aplikace o zabezpečení pomocí Spring Security. To je framework, který se zaměřuje na zabezpečení a poskytuje autentizaci i autorizaci, mnoho tříd pro konfiguraci, obrané mechanizmy, například proti *session fixation útoku*, *clickjacking*, *cross site podvrhům* apod. Také v sobě integruje Servlet API nebo *Spring Web MVC* [10].

Pokud chceme do našeho projektu zahrnout Spring Security, postačí vložit následující Maven dependency do našeho Project Object Model (**POM.xml**).

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
    [...]
    <dependencies>
    [...]
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <!-- Last stable version -->
            <version>4.2.1</version>
        </dependency>
    </dependencies>
</project>
```

Výpis 5: Spring Security Web dependency [9], *upraveno*

Nyní už jen stačí implementovat několik konfiguračních tříd. Zde je důležité podotknout, že Spring Boot obsahuje sadu autokonfiguračních tříd, které provedou mnohé nastavení za nás. Nicméně autokonfigurace sama o sobě není dostačující. Spring se postará o vytvoření Security Contextu, který udržuje objekty související s zabezpečením, např. jaký uživatel je aktuálně přihlášený, jeho role případně další podrobnosti o autentizaci. V aplikaci Place Reservation máme implementovaný svůj autentizační proces. Ve zdrojovém kódu to znamená vytvořit třídu, která implementuje rozhraní `AuthenticationManager`.

```java
public interface AuthenticationManager {

    public Authentication authenticate(Authentication authentication)
        throws AuthenticationException;
}
```

Výpis 6: Rozhraní `AuthenticationManager` [10]

Dle [9] má rozhraní jen jednu metodu `authenticate(...)`, kdy při provádění metody můžou nastat tři situace:

1. vrácení plně autentizovaného objekt, který navíc obsahuje credentials,
2. vyhození výjimky `AuthenticationException` - pokud se autentizace nepodaří,
3. vrácení `null` literálu - značí neznámý výsledek *(autentizace nemohla být provedena)*.

Uvažujme první příklad, tedy že autentizace proběhne v pořádku a metoda vrátí autentizovaný objekt. Pokud bychom se podívali do dokumentace, zjistíme, že typ `Authentication`

není nic jiného než zase rozhraní. Chceme tedy najít typ, který bude implementovat rozhraní `Authentication` a zároveň bude nositelem credentials. Jedním z vhodných kandidátů může být třída `UsernamePasswordAuthenticationToken`.

```
    public class UsernamePasswordAuthenticationToken extends
        AbstractAuthenticationToken {

        private final Object principal;
        private Object credentials;

        public UsernamePasswordAuthenticationToken(Object principal, Object
            credentials) {...}

        public UsernamePasswordAuthenticationToken(Object principal, Object
            credentials, Collection<? extends GrantedAuthority> authorities)
            {...}

        public Object getCredentials() {...}

        public Object getPrincipal() {...}

        public void setAuthenticated(boolean isAuthenticated) throws
            IllegalArgumentException {...}

        public void eraseCredentials() {...}
}
```

Výpis 7: Třída `UsernamePasswordAuthenticationToken` [9]

Třída je poměrně jednoduchá a kromě konstruktorů, getterů, setterů, obsahuje také dvě instanční proměnné `principal` a `credentials`. Obě tyto třídy by měly být nastaveny jako property pomocí `toString()` metody. Ideálním typem je `String`.

Víme už tedy skoro všechno potřebné k dokončení implementace. Finalní zkrácená podoba metody `authenticate(...)` je zobrazena ve výpisu 8.

```
@Service
public class AuthServiceImpl implements AuthService {
    [...]

   @Override
   public Authentication authenticate(Authentication authentication) {

       // FIRST PART
      String login = authentication.getPrincipal().toString();

      // if was used Kerberos it is encryption key, otherwise
      // password is user's domain password
      String password = authentication.getCredentials().toString();

       // if was used Kerberos:
       //  - login contains kerberos\username format,
       //  - kerberosLogin is not null
       // if was used OAuth 2:
       //  - login contains simple username
       //  - kerberosLogin is null
      String kerberosLogin = extractLoginName(login);

      if (kerberosLogin != null) {
         // Check if the encryption key is not incorrect
         if(!password.equals(assembleEncryptionKey())) {
             authentication.setAuthenticated(false);
             // If so return no-authenticate
             return authentication;
             }
            // Otherwise set acquired login for next proceeding
            login = kerberosLogin;
      }
```

```
 // SECOND PART
try {
   // According used protocol will be contact right
   // LDAP micro-service endpoint
   // UserPasswordAuthenticateDTO - wrapper object
   UserPasswordAuthenticateDTO = restTemplate.postForObject(
         kerberosLogin != null ? authRequestKerberosURL : authRequestURL,
         new UserPasswordAuthenticateDTO(login, password),
         UserPasswordAuthenticateDTO.class
   );
} catch (HttpStatusCodeException e) {
    // Authentication process failed
    [...]
}
  // Check if the LDAP module response positively
if (UserPasswordAuthenticateDTO == null || !UserPasswordAuthenticateDTO.
    isAuthenticated()) {
    // Authentication process failed
   [...]
}

  // THIRD PART
// Get AuthUserDTO object with user details from response
AuthUserDTO authUser = authUserMapper.mapFromObject(
    UserPasswordAuthenticateDTO.getDetails());

try {
   // Core module return more informations (eg. roles, photo..)
   AuthUserDTO result = restTemplate.postForObject(accessRequestOrigin,
      authUser, AuthUserDTO.class);

   // Amend user by received Core database information
   authUser.setId(result.getId());
   authUser.setPermissions(result.getPermissions());
   [...]
} catch (HttpStatusCodeException e) {
   // Authentication process failed
   [...]
}
```

```
        // FOURTH PART
      UsernamePasswordAuthenticationToken token = new
          UsernamePasswordAuthenticationToken(
            user.getName(),
            authenticate.getCredentials(),
            parseAuthorities(user)
          );
      return token;
   }
```

Výpis 8: Implementace metody `authenticate(Authentiation authentication)`

Metodu lze rozdělit na několik částí. V první části se určuje zdali uživatel použil přihlášení pomocí SSO nebo manuálně formulářem podle toho, v jakém formátu přijde login uživatele (ten získáme pomocí `authentication.getPrincipal().toString()`). Pokud je získáný login ve formátu `kerberos\username` znamená to, že byl použit Kerberos protokol a nemusíme tedy přenášet doménové heslo uživatele a ověřovat vůči LDAPu, namísto toho lze pracovat s *encryption key* a použije se servisní účet pro stažení informací o uživateli z LDAPu. Pokud se SSO z nějakého důvodu nezdaří, uživatel použije přihlašovací formulář a login přijde ve formátu `username`. V druhé části se kontaktuje LDAP microservices a získají se informace o uživateli. Pro přenos dat mezi microservises se využívá objekt typu `UserPasswordAuthenticateDTO`. Příklad instance je vidět ve výpisu 9.

```
{
    "principal": "kucabpet",
    "credentials": "x",
    "authenticated": true,
    "details": {
        "id" => null,
        "login" => "kucabpet",
        "firstName" => "Petr",
        "lastName" => "Kuca",
        "workLocation" => "Ostrava",
        "manager" => "edzgzird",
        "permissions" => null
    }
}
```

Výpis 9: Příklad JSON reprezentace objektu `UserPasswordAuthenticateDTO`

Atributy `id` a `permission` jsou nastaveny na `null` a budou doplněny z aplikační databáze. Tím se dostáváme ke třetí části popisované metody, kontaktování core microservices. Core je hlavní modul aplikace, který obsahuje všechny REST endpointy, byznys logiku a perzistentní vrstvu. Pro komunikaci s ním se používá jiný objekt, protože již není potřeba nést informace jako *principal, credentials, authenticated*. Namísto toho je použit objekt typu AuthUserDTO, viz výpis 10.

```
{
    "id": 1,
    "login": "kucabpet",
    "firstName": "Petr",
    "lastName": "Kuca",
    "email": "petr.a.kuca@tieto.com",
    "workLocation": "Ostrava",
    "manager": "edzgzird",
    "photo": [...],
    "permissions" : [
        USER
    ]
}
```

Výpis 10: Příklad JSON reprezentace objektu `AuthUserDTO`

Při kontaktování core microservice dojde k ověření, jestli uživatel v aplikační databázi již existuje - provádí se autorizace. Pokud ne, dojde k vytvoření a přidělení permission a databázového id. Na tomto místě bych chtěl popsat, jakým způsobem se přidělují role. Můžeme uvažovat tři běžné role: ADMIN, MANAGER, USER. Firemní LDAP neposkytuje jednotný způsob, jak rozeznat uživatele a manažera, proto je zde algoritmus pro určení:
Při požadavku na ověření uživatele v core se zkontroluje, jestli se login uživatele nevyskytuje v databázové tabulce, ve sloupci `manager`. Pokud je nejméně jeden záznam nalezen, uživatel se stává manažerem a je mu přidělena permission MANAGER. Aby byla ošetřena i situace kdy se manažer přihlásí dříve než některý jeho podřízený, je potřeba provádět kontrolu při každém požadavku na autorizaci. Permission ADMIN může udělovat jen uživatel, který již je administrátorem. První takový administrátor se vytváří fyzickým přepsáním řádku v databázové tabulce. Ve čtvrté a poslední části se vytváří `UsernamePasswordAuthenticationToken`, popsaný již dříve. Dle [9] jeho konstruktor přijímá tři parametry:

1. `Object principal` - nastavuje se username,
2. `Object credentials` - nastavuje se přihlašovací heslo případně encryption key,
3. `Collection<? extends GrantedAuthority> authorities` - permissions uživatele.

Vytvoření a vrácení instance UsernamePasswordAuthenticationToken jsme uživatele přihlásili a proces autentizace je tak u konce. Pokud v kterékoli části dojde k chybě je nutné nastavit `authentication.setAuthenticated(false)` a vratít neautentizovaný `autentication` objekt.

3.3.4.1 Popis modulů projektů

api-model knihovna, která obsahuje sadu objektů pro komunikaci mezi moduly *(AuthUserDTO, UserPasswordAuthenticateDTO ...)* - využívá se vzor Data Mapper.

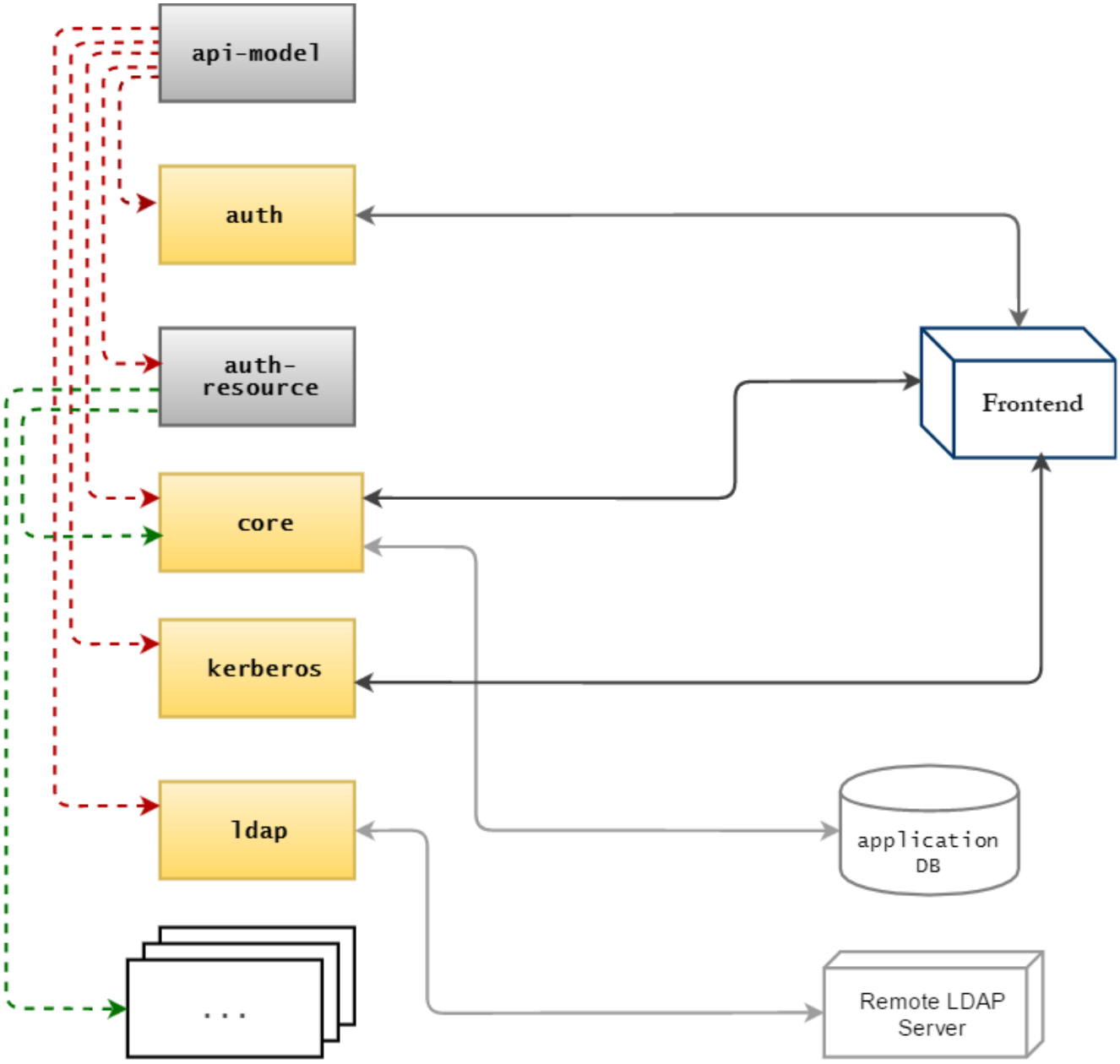
auth Implementace OAuth 2 a Spring Security (metoda `Authentication authenticate(..)`).

auth-resource Knihovna, která obsahuje část konfigurace Spring Security. Její hlavní podstatou je distribuovat zabezpečení endpointů mezi moduly. Vkládá se všude tam, kde existují endpointy a je potřeba je zabezpečit. Není potřeba ji vkládat do auth modulu pro zabezpečení /oauth/$*$ endpointů, protože ten již zahrnuje zabezpečení na úrovni tokenů.

core Hlavní modul aplikace, obsahuje REST endpointy, logiku, perzistentní vrstvu, zprostředkovává data z databáze.

kerberos Implementace SSO, je nutné dodat, že uživatele pouze autentizuje ale nepřihlašuje - samotné přihlášení do aplikace je přenecháno `auth` modulu.

ldap Modul, který zprostředkovává data z firemního LDAP serveru.



Obrázek 7: Náhled na moduly aplikace, znázornění závislostí a komunikací

Pozn.: čárkované šipky přestavují komunikaci na doménové vrstvě, plné šipky značí komunikaci pomocí protokolů (HTTP, TCP, UDP...), moduly označené šedou barvou představují knihovny (tzn. na serveru se nevytváří jako běžící služba, pouze se přidají jako dependency v POM.xml), moduly označené žlutou barvou představují microservices (tzn. vytvoří se jako běžící služba).

3.4 Locker projekt

Mým posledním projektem, na kterém jsem v rámci praxe pracoval, byl Locker. Na projektu jsme spolu s kolegyněmi a kolegy začal pracovat na popud mého manažera, který přišel s požadavkem na aplikaci pro správu skříněk ve firmě. V budově firmy jsou na každém patře k dispozici uzamykatelné skříňky, které slouží zaměstnancům, stážistům nebo externistům pro úschovu věcí. Tyto skříňky jsou zpracovávány jen papírově a chybí aplikace, kde by se uživatelé mohli podívat, které skříňky jsou volné a případně si nějakou vybrat a rezervovat.

3.4.1 Terminologie

Při vývoji projektu jsme se s kolegy shodl na používání následujících výrazů:

Box Jedná se o velkou skříň, která má v sobě vestavěné lockery (průměrně 6 až 8). Nese jedinečný identifikátor.

Locker Značí samotnou uzamykatelnou skříňku *(buňku)*. Každý locker je součástí jednoho boxu. Nenese žádný identifikátor.

3.4.2 Požadavky aplikace

Nejdříve bylo potřeba aplikaci správně navrhnout. Začal tedy sběr požadavků aplikace a zodpovězení otázek Zachmanova Frameworku - *Co? Jak? Kde? Kdo? Kdy? Proč?* [1]

- **Co?** Informační systém pro správu skříněk ve firmě, možnost přihlásit se do něj pomocí doménového jména a hesla. Přihlášený uživatel si bude moci vytvořit rezervaci, zobrazit si již rezervované skříňky a prohlížet si je.
- **Jak?** Procesy s informacemi budou jen jejich zobrazovaní a ukládání.
- **Kde?** S informacemi se bude pracovat pouze na serveru. Uživatel bude s aplikací pracovat pomocí webového prohlížeče.
- **Kdo?** Do aplikace se bude moci přihlásit každý kdo je zaměstnancem, stážistou nebo externistou firmy. Bude mít tři role:
  1. **Employee** - uživatel aplikace, nemá žádné zvláštní oprávnění. Bude mu umožněno prohlížet volné skříňky, vytvářet rezervace, stornovat své rezervace.
  2. **Manager** - představuje manažere, má vyšší oprávnění než Employee. Navíc bude moci prohlížet skříňky svých podřízených, vytvářet skupiny uživatelů.
  3. **Admin** - uživatel s nejvyšší oprávněním, celková správa aplikace.
- **Kdy?** Předpokládá se, že s aplikací se nebude pracovat každý den ale podle potřeby v pracovní době. Odhadem ji bude každý uživatel využívat 1-2x.

- **Proč?** Cílem je vytvořit rezervační systém s následujícími pravidly:

    1. Každý uživatel s rolí Employee bude mít možnost rezervovat max 2 skříňky,
    2. uživatel s rolí Manager bude mít možnost rezervovat si maximálně 5 skříněk,
    3. rezervace bude vložena do systému a bude mít omezenou platnost, za kterou žadatel musí zajít na recepci a fyzicky převzít klíček od skříňky, pak se rezervace stává trvalou,
    4. pokud bude chtít uživatel rezervaci stornovat, musí jít klíček opět vrátit na recepci, teprve až pak bude rezervace v systému stornována. Rezervace nebudou odstraňovány z databáze z důvodu zachování historie,
    5. uživatel s rolí Admin bude moci vytvářet rezervace pouze jiným uživatelům, nikoliv sám sobě. Bude také provádět správu aplikace.

3.4.3 Technologie

Použité technologie budou standardní jako u jiných aplikací, tedy Java 8, Spring Boot Framework, relační databáze PostgreSQL, React (JavaScript).

3.4.4 Případy užití

Protože se nejedná o žádnou rozsáhlou aplikaci, při návrhu jsme zahrnuli do případu užití jen základní procesy uživatele. V budoucnu se může aplikace ještě dále rozšiřovat. Diagram případu užití je vidět na obrázku 8.

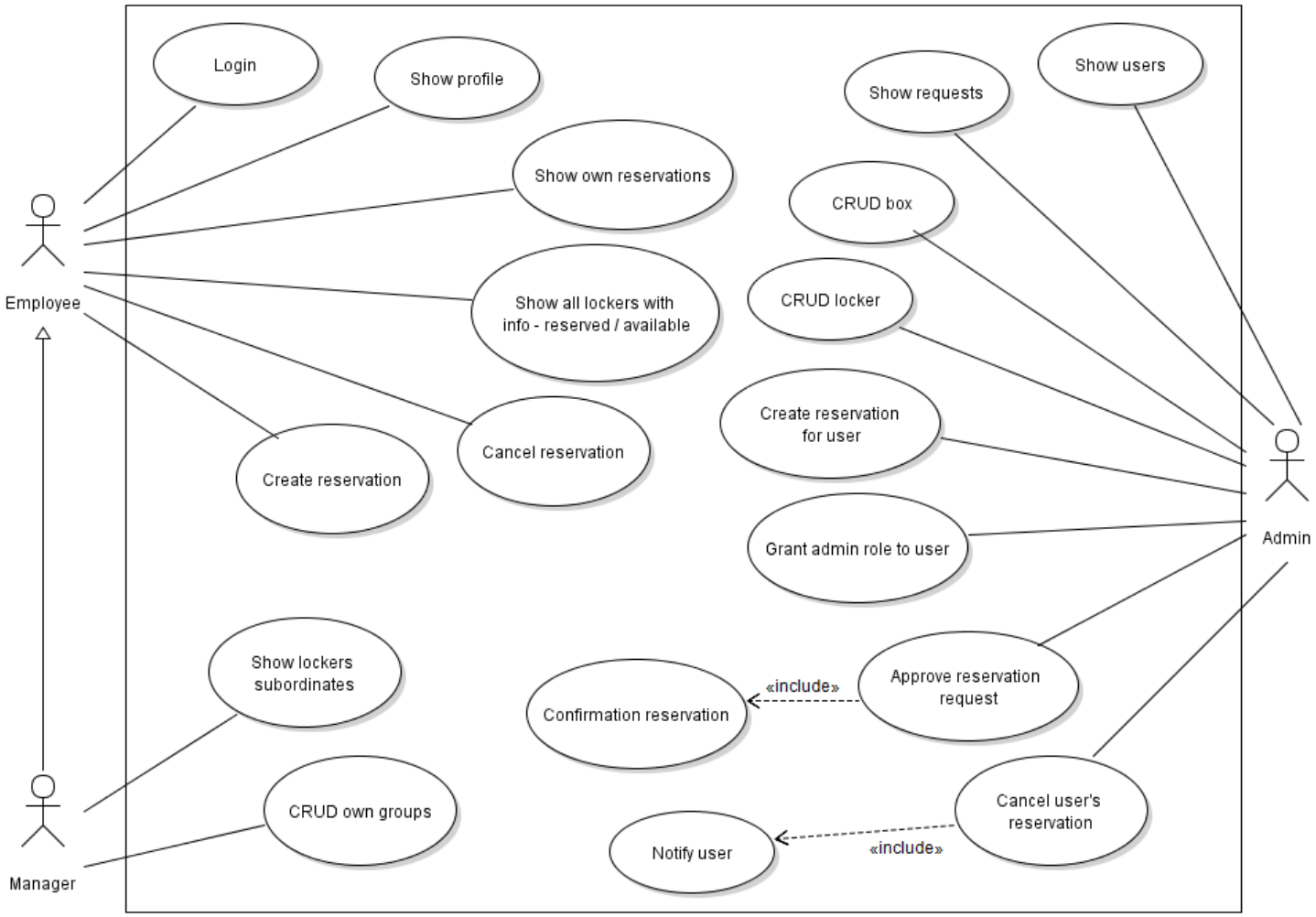
3.4.5 Model databáze

Pro návrh databáze jsme použili relační datový model. Kromě entit jako *locker*, *box*, *user* se v modelu objevují také tabulky reprezentující stav a *request* entita. Request, který reprezentuje požadavek uživatele na rezervaci lockeru, je svázána dvěma vazbami s tabulkou *user*. První reprezentuje uživatele který požadavek zadává - sloupec `created_by`, druhý představuje administrátora, který bude požadavek vyřizovat - sloupec `closed_by`. Poslední dvě vazby k tabulkám *request_states* a *request_types* symbolizuji, jak názvy napovídají, stav a typ požadavku.
Požadavek může nabývat tyto stavy *([název - popis]*):

OPENED Vytvoření nového požadavku.

CANCELED Zrušení požadavku buďto uživatelem, nebo administrátorem. To upřesňuje tabulka `request_types`).

CLOSED Úspěšné uzavření požadavku.



Obrázek 8: Locker - případy užití

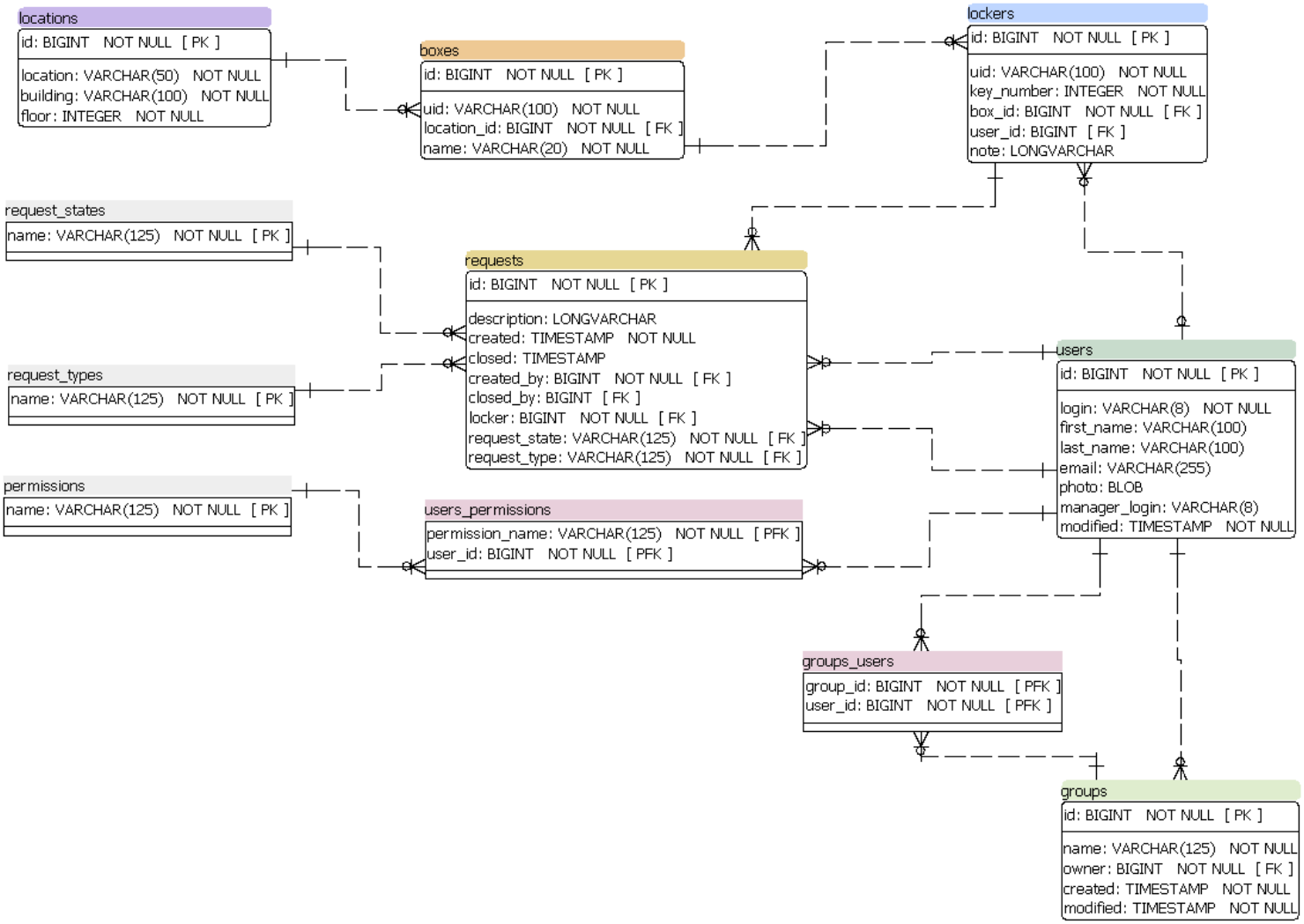
Požadavky můžou být také několika typu *([název - popis]*):

LOCKER_RESERVATION Vytvoření požadavku.

LOCKER_CANCEL_RESERVATION_BY_ADMIN Stornování rezervace administrátorem.

LOCKER_CANCEL_RESERVATION_BY_EMPLOYEE Stornování rezervace uživatelem.

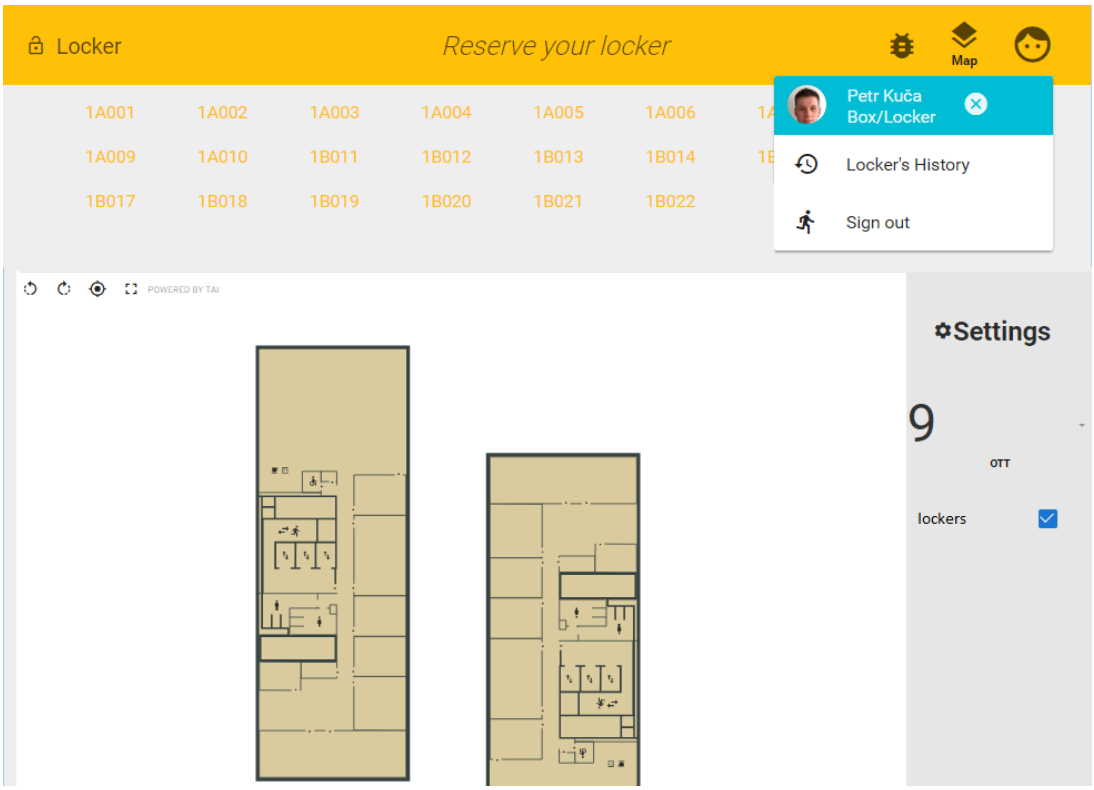
LOCKER_CLOSED_RESERVATION_BY_ADMIN Vyřízení požadavku adminstrátorem.



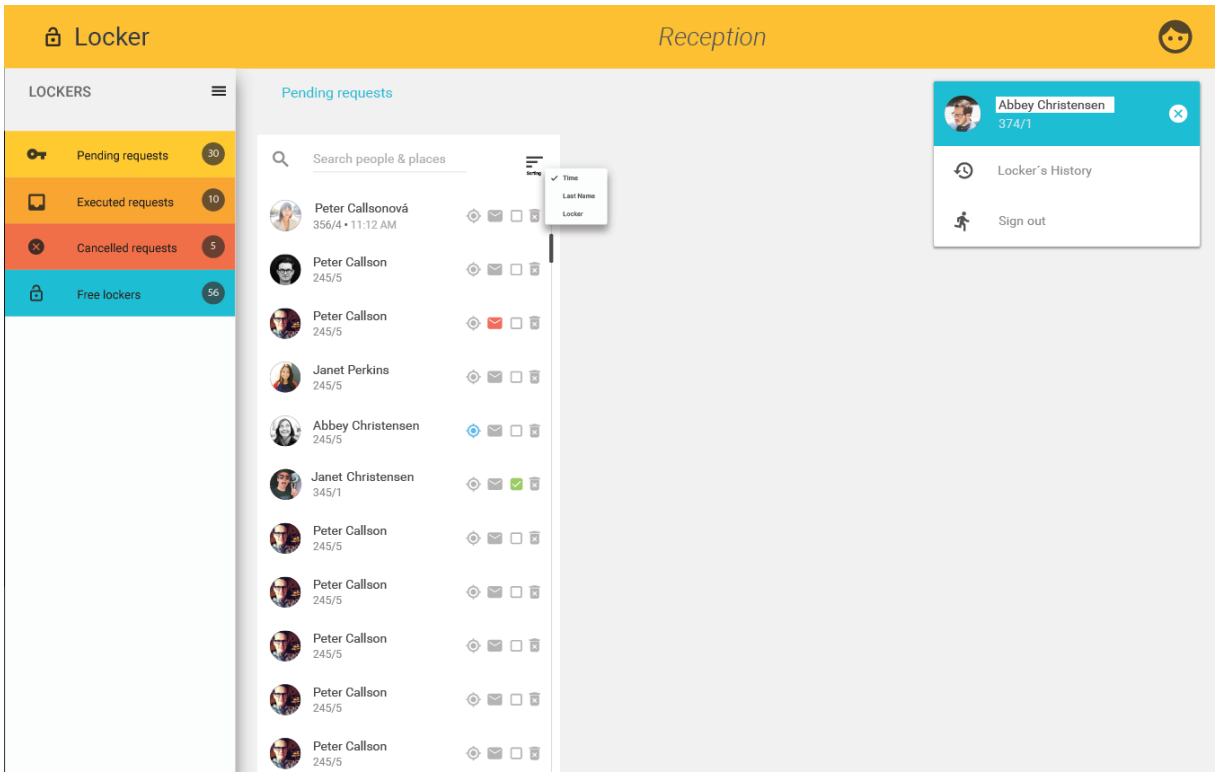
Obrázek 9: Locker - databázový model

3.4.6 Vzhled aplikace

Na obrázcích 10 a 11 je vidět beta verzi frontendu aplikace Locker.



Obrázek 10: Locker - hlavní stránka po přihlášení



Obrázek 11: Locker - hlavní administrační stránka

4 Uplatněné a chybějící znalosti

4.1 Uplatněné znalosti z vysoké školy

Na praxi jsem uplatňoval zejména znalosti získané na vysoké škole. Při programování pro mě bylo přínosné absolvování předmětů Algoritmy II, Programování I a Programování II. Při navrhování architektury a tvorby dokumentace jsem využil UML jazyk, který byl předmětem kurzu Úvod do Softwarového Inženýrství. Předmět Vývoj Informačních Systémů, kde jsem získal přehled o návrhových vzorech mi nejen poskytl návod na řešení některých problémů při vývoji, ale dovolil mi také porozumět některým rysům různých frameworků. Musím také zmínit předměty Úvod do Databázových Systémů a Databáze a Informační Systémy, které mi daly mnoho důležitých informací o relačních databázích, jenž jsem použil při tvorbě architektury projektu Locker. Při vývoji ve Spring frameworku jsem čerpal ze znalostí z předmětu Java Technologie, díky kterému jsem mohl snadněji porozuměl jeho chování. Částečně mi pomohl i předmět Jazyk anglický b/I pro FEI, který mi dovolil zdokonalit se v anglickém jazyce.

4.2 Chybějící znalosti

I přes získané znalosti ze studia jsem postrádal znalost mnoha technologií a musel jsem se je během praxe doučit. Jednou ze základních věcí téměř každého vývoje rozsáhlejšího informačního systému je používání verzovacího systému GIT. Tento systém a práci s ním jsem před praxí neznal a neuměl s ním pracovat. Ocenil bych kdyby se tento dnes nejpoužívanější verzovací software byl součástí některého z povinných předmětů bakalářského studia a učili se v něm kromě základů také pokročilé techniky správy repositářů. Mezi další chybějící zkušenosti bych zařadil práci v týmu nebo praktické zkušenosti s nasazováním aplikací do produkčního prostředí.

5 Závěr

Za dobu své bakalářské praxe jsem se naučil deployovat aplikace, vytvářet instance aplikačního serveru, nastavovat a obsluhovat různé reverzní proxy a webové servery. Za úspěch také považuji to, že jsem prostudoval i porozuměl mechanismu Single Sign-On a autentizačním protokolům jako Kerberos nebo OAuth 2, které se dnes ve velké míře používají. Kromě toho jsme dokázal obě tyto technologie naimplementovat ve Spring frameworku. V průběhu praxe jsem se také seznámil a dokázal pracovat s rozsáhlým Spring Boot frameworkem a Spring Security frameworkem. Dále jsem si vyzkoušel vyzkoušel práci v týmu při návrh a vyvíjení projektu Locker. Získal jsem mnoho zkušeností z návrhu architektury moderních informačních systému pro web v programovacím jazyce Java i obecně. Na závěr musím svoji odbornou praxi hodnotit velmi přínosně a pozitivně, neboť mi dala možnost nahlédnout do odvětní IT světa, vyzkoušet si reálně práci vývojáře na projektech. Kromě toho jsem poznal politiku jedné z největších firem IT v České republice, kde se navíc mluví cizím jazykem. Dokázal jsem tedy získat mnoho nových znalostí a zkušeností v docela krátkém čase, a rád bych v budoucnu v takovéto firmě pracoval.

Literatura

[1] About the Zachman Framework. Zachman International - Enterprise Architecture [online]. Copyright © 2008 John A. Zachman, [cit. 11.04.2017]. Dostupné z: https://www.zachman.com/about-the-zachman-framework

[2] Apache Tomcat 8 - Documentation Index. Apache Tomcat [online]. Copyright © 1999 [cit. 22.03.2017]. Dostupné z: https://tomcat.apache.org/tomcat-8.5-doc/index.html

[3] Configuring HTTPS servers. Nginx news [online]. Copyright © 2017 NGINX Inc [cit. 21.02.2017]. Dostupné z: http://nginx.org/en/docs/http/configuring_https_servers.html

[4] IBM Knowledge Center. [online]. Copyright © Copyright IBM Corp. 2015 [cit. 15.03.2017]. Dostupné z: https://www.ibm.com/support/knowledgecenter/en/SS9H2Y_6.0.1/com.ibm.dp.xb.doc/spnego_protocol.html

[5] Kerberos protokol a Single sign-on. SAMURAJ-cz.com - počítačové sítě, Cisco, Microsoft, VMware, administrace [online]. Copyright © 2005 [cit. 10.03.2017]. Dostupné z: http://www.samuraj-cz.com/clanek/kerberos-protokol-a-single-sign-on

[6] Never Get Confused by OAuth Again. API-University – APIs your consumers will love [online]. Copyright © 2017 [cit. 22.03.2017]. Dostupné z: http://api-university.com/oauth-cheat-sheet

[7] Ostrava Tieto Towers. Contractis [online]. c2013 [cit. 2017-04-10]. Dostupné z: http://www.contractis.cz/cs/reference/tdi/ostrava-tieto-towers-cs

[8] Spring Boot Reference Guide. [online]. Copyright © 2012 [cit. 22.03.2017]. Dostupné z: http://docs.spring.io/spring-boot/docs/1.5.2.RELEASE/reference/htmlsingle

[9] Spring Security Reference. [online]. Copyright © 2004 [cit. 22.03.2017]. Dostupné z: http://docs.spring.io/spring-security/site/docs/4.2.2.RELEASE/reference/htmlsingle

[10] Spring Security. [online]. Copyright © 2017 Pivotal Software [cit. 11.04.2017]. Dostupné z: http://projects.spring.io/spring-security

[11] The OAuth 2.0 Authorization Framework. [online]. Copyright (c) 2012 IETF Trust and the persons identified as the document authors. [cit. 22.03.2017] Dostupné z: https://oauth.net/2

[12] Tieto - IT, výzkum a vývoj a poradenství. Tieto [online]. Copyright © 2016 Tieto [cit. 22.03.2017]. Dostupné z: https://www.tieto.cz

[13] The Apache Software Foundation. The Apache HTTP Server Project [online]. Copyright © 1997 [cit. 13.04.2017]. Dostupné z: https://httpd.apache.org