Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Implementace a konfigurace řešení pro automatickou autentizaci do wifi síťě pomocí mobilní aplikace pro Android

Jan Kusý

Vedoucí práce: Ing. Alexandru Moucha, Ph.D.

11. května 2015

Poděkování

Děkuji své rodině a přátelům za podporu po celou doby studia. Dále bych rád poděkoval Ing. Alexandru Mouchovi za jeho ochotu vést moji bakalářskou práci.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 11. května 2015 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Kusý, Jan. *Implementace a konfigurace řešení pro automatickou autentizaci do wifi sítě pomocí mobilní aplikace pro Android.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Práce se zabývá problematikou možnosti automatického přihlašování do wifi sítě pomocí mobilní aplikace Android.

Cílem práce je navrhnout základní kostru řešení obsahující minimální funkční konfiguraci a implementaci všech potřebných komponent.

Výsledné řešení je složeno z mobilní aplikace, webové administrátorské aplikace, implementace webových služeb a konfiguračních souborů serveru.

Klíčová slova OpenWrt, mobilní aplikace, RADIUS, Android, Java

Abstract

Thesis deals with the ability to automatically authenticate to the wifi network using mobile Android application.

The aim is to propose a basic framework of solutions containing the minimum functional configuration and implementation of all necessary components.

The final solution is composed of mobile application, web administration application, implementation of web services and configuration files of server.

Keywords OpenWrt, mobile appliaction, RADIUS, Android, Java

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Seznámení s tématem

Když se řekne captive portal, pravděpodobně se málo komu vybaví význam tohoto slovního spojení. Nicméně dle mého názoru téměř každý uživatel mobilního zařízení či přenosného počítače se již s tímto pojmem nevědomky setkal. Captive portal je definován jako směrovač či hostitelská brána, která nepropustí žádný síťový provoz dokud není uživatel autentizován.[1]

Funkce, které captive portal zajišťuje:

- umožňuje mobilnímu zařízení získat IP adresu z DHCP[2] serveru přes wifi připojení
- přesměrovává veškerý síťový provoz z mobilního zařízení do captive portalu
- zobrazuje webovou stránku obsahující podmínky užití, platební informace nebo přihlašovací formulář
- zpřístupňuje internetové připojení uživateli, který souhlasí s podmínkami užití nebo který je ověřen přihlašovacími údaji

Z mé zkušenosti je captive portal hojně využíván jako brána sloužící k přijetí podmínek používání free wifi hotspotu a zároveň tato úvodní webová stránka informuje uživatele wifi sítě komu tento hotspot patří. S takovým to typem využití captive portalu se můžeme setkat například v různých řetězcích rychlého občerstvení a dalších obdobných místech. S řešením, kde je nutná uživatelská autentizace či tvorba dočasných uživatelský přístupů, se nejčastěji setkáme v hotelech, kde při registraci návštěvník obdrží dočasné přihlašovací údaje pro přístup do wifi sítě po dobu jeho ubytování.

Cílem této práce je zabývat se zjednodušením přihlašování do takovéhoto captive portalu pomocí mobilní aplikace, která nebude sloužit pouze jako jakýsi autorizační klíč, ale navíc bude umožňovat přijímat notifikace různého informačního charakteru, které budou ve střetu se zájmy uživatele.

Práce zahrnuje komplexní nástin řešení služby od konfigurace serverových a síťových prvků až po implementaci mobilní a webové administrátorské aplikace. Cílem je vytvoření funkčního prototypu ukazující především potenciální funkčnost autentizace pomocí mobilní aplikace.

Motivace

Důvody, proč jsem se rozhodl vybrat si toto téma v rámci mé bakalářské práce, tkví ve špatné zkušenosti s jedním takovým captive portalem, který byl umístěn v nejmenovaném hotelu, kde se jsem se neustále musel přihlašovat, což vedlo v konečném důsledku k tomu, že jsem se nakonec raději rozhodl využívat vlastní mobilní datové připojení.

Po této zkušenosti jsem přemýšlel jak by se dal tento problém vyřešit a zároveň nabídnout návštěvníkovi více informací o pobytu, které by mohl hotel poskytovat formou notifikací do mobilní aplikace. V koncovém důsledku toto řešení hotelu umožní rychlejší a jednodušší propagaci svých služeb. Hotel tím pádem získá možnost nabízet návštěvníkům třeba i nějaké výhody a další užitečné informace, například v podobě denních jídelních menu a tak podobně.

Díky studiu oboru informační technologie jsem získal potřebné znalosti nutné ke konfiguraci a instalaci serverových i síťových komponent. Zároveň jsem absolvoval všechny předměty týkající se programovacího jazyka Java což mi dalo znalosti potřebné k implementaci RESTful[3] webových služeb sloužících jako backend[1] server pro mobilní aplikaci a implementaci webové administrátorské aplikace, která slouží pro správu uživatelů, notifikaci a dalších implementovaných vlastností.

Jelikož jsem si téma vymyslel sám, mojí hlavní motivací a cílem je vytvořit komplexnější službu, kde si propojím všechny své znalosti do jednoho celku a výsledkem bude jádro služby, kterou bych do budoucna mohl zdokonalovat a případně ji i produkčně nasadit.

[1] Backend je název pro serverovou aplikaci, která zajišťuje nějakou podporu (například přístup k datům) pro klientskou aplikaci

Kapitola 1

Analýza

Tato kapitola je zaměřená na analýzu nejčastějších modelových situací, které jsou spojené s poskytováním internetového připojení, kde je využit captive portal pouze jako brána sloužící k souhlasu podmínek užití wifi sítě. Kapitola obsahuje rozbor výhod ale i nevýhod našeho řešení z různých hledisek. Důležitou součástí analýzy je vhodný výběr firmwaru[2] pro cenově dostupný domácí router. Nakonec se podíváme na existující captive portaly a možnosti jejich využití pro naše řešení.

1.1 Specifikace a popis služby

Pro moji službu jsem si vymyslel pracovní název „WifiPoint“, proto v implementační části práce se lze setkat s názvy Java balíčků, které obsahují výše zmíněný název „WifiPoint“ či někde jsem použil jako prefixy názvů Java tříd zkratkou **WP**. Zároveň tak v návrhové a dále pak i v implementační části uživatelského rozhraní jsem volil tento název jako název reprezentující službu. V následujících částech mojí práce ho budu občas používat jako zástupný název pro službu. V následující sekci popisuji modelovou situaci, ve které nastiňuji koncept nasazení služby do několika restaurací, které vlastní jeden majitel. Předpokládejme, že majitel si nazval svoji síť wifi míst taktéž jako WifiPoint.

1.1.1 Modelová situace původního řešení

Představme si úspěšného podnikatele, který podniká v oblasti gastronomie a hotelnictví. Tento podnikatel vlastní 3 restaurace rozmístěné v okruhu 10 km, tedy v rámci jednoho města či okresu. Aktuální situace a je taková, že v každém podniku má wifi router s jednoduchým captive portalem, který po připojení zobrazuje souhlas s podmínkami s užití po jejichž odsouhlasení může návštěvník využívat zdarma internetové připojení. Toto řešení se může jevit

[2]Firmware je jednoduchý software sloužící k řízení a konfiguraci vestavných zařízení

jako jednoduché, bezproblémové a ideální. Nicméně když si představíme, že jsme návštěvníci restaurace, kteří se v některé z těchto restaurací stravují několikrát do týdne a máme zájem využívat zdarma internetové připojení, může nás časem omrzet opakované souhlasení s podmínkami užití. Potenciál popsaného řešení končí u toho, že maximálně nabídne návštěvníkům internetové připojení zdarma.

1.1.2 Popis naší služby

Naše řešení spočívá v tom, že použijeme captive portal stejně jako v modelové situaci, avšak budeme ho využívat jako bránu pro přihlášení se do wifi sítě. Náš captive portal by tedy měl splňovat funkci průchozí brány, pomocí které si lze stáhnout mobilní aplikaci. Naše řešení navíc nabízí webovou administrátorskou aplikaci, pomocí které lze spravovat wifi stanice podporované službu, spravovat uživatele a sledovat statistiky připojování na jednotlivé wifi stanice. Administrace bude umožňovat správu „nabídek“ různých typů. Lze typy nabídek nastavit tak, aby vyhovovaly potřebám provozovatele služby. Typy nabídek mohou být například „Polední menu“, „Zábava“ nebo „Aktualita“. Tyto typy slouží k označení informačního charakteru nabídky. Smyslem nabídek je možnost posílat uživatelům mobilní aplikace informace různých typů pomocí notifikací.

1.1.3 Výhody služby z hlediska provozovatele služby

Předpokládejme, že majitel přešel na naše řešení. Co tím získal? Získal tím možnost sám sebe propagovat a dělat si cílenou reklamu pro své klienty. Díky administrační aplikaci má možnost rozesílat polední menu uživatelům nebo je třeba zvát na různé společenské akce. Informační charakter typů nabídek záleží pouze na provozovateli služby.

1.1.4 Výhody služby z hlediska návštěvníka

Návštěvník restaurace získá mobilní aplikaci, která ho vždy sama přihlásí do wifi sítě služby, pokud bude poblíž. Dále pak má možnost přijímat notifikace různých nabídek, ze kterých si může vybrat ty, které chce přijímat. Samozřejmě nechceme uživatele mobilní aplikace zahltit různými notifikacemi, o které ani nemá zájem, proto je zde možnost vybrat si to co ho bude konkrétně zajímat.

1.1.5 Možné nevýhody našeho řešení

Bohužel každé řešení má své nevýhody, které by mohly zapříčit neúspěch celého řešení. Nastíním hlavní z nich a popíši jejich možná řešení.

První nevýhodou může být nutnost instalace mobilní aplikace. Bohužel někteří uživatelé si nechtějí instalovat žádné další aplikace do svých telefonů, a proto nevyužijí službu. Já osobně si dokážu představit člověka, který se velmi často pohybuje v prostorech, kde bude mít wifi připojení zdarma a pravděpodobně si aplikaci nainstaluje a bude ji v budoucnu využívat. Na druhé straně je zde návštěvník, který nenavštěvuje tak často danou lokalitu a nebude mít zájem si instalovat aplikaci.
Tato nevýhoda může být částečně řešena například instalací a investicí do vytvoření dalších wifi stanic, které budou úplně mimo lokalitu, než kde se nachází daná restaurace, avšak budou to místa s vysokou koncentrací osob. Ideálním místem pro možné rozšíření je například hlavní autobusová zastávka, kde v případě umístění wifi stanice by se mohla rapidně zvýšit používanost mobilní aplikace a tím i celé služby.

Další z nevýhod je podpora připojení pouze smartphony[3]. Uživatel, který by se chtěl připojit do wifi sítě pomocí nepodporovaného zařízení(například notebook) nemá momentálně v aktuálním návrhu možnost se žádným způsobem připojit.
Lze toto vyřešit ponecháním možnosti přihlášení a registrace přes webové rozhraní jako při klasickém využití captive portalu.

1.2 Analýza síťových prvků

Abychom mohli vybrat některý z domácích wifi routerů, pro který budeme konfigurovat naše řešení, nejprve musíme zvolit vhodný firmware. A následně poté budeme moci porovnat dostupné routery. Cílem je vybrat dostatečně výkonný domácí router, který by měl zvládnout plynulý běh captive portalu a jeho cena by se neměla vyšplhat k extrémním částkám.

1.2.1 Analýza dostupných firmwarů pro bezdrátový router

Na začátku této analýzy se na nám může naskytnout otázka, proč vlastně potřebujeme nějaký jiný alternativní firmware pro náš router, který si mohu koupit běžně v jakémkoliv obchodě. Odpověď je velmi jednoduchá. Výrobci bezdrátových routerů používají pro domácí routery velmi jednoduché firmwary, které jsou přesně optimalizované pro danou čipovou sadu a bohužel neobsahují žádné, pro nás potřebné nástroje, pomocí kterých bychom mohli nakonfigurovat router. Tyto originální firmwary mají naštěstí možnost aktualizace, díky které jsme schopni celkem snadno originální firmware nahradit naším alternativním.

[3]Smartpohone je takzvaně chytrý telefon, který využívá pokročilý mobilní operační systém. Dále jen **smartphone**

Alternativních firmwarů pro bezdrátové routery můžeme v dnešní době najít hned několik. Ale mezi nejznámější patří OpenWRT. Neméně známým je další firmware DD-WRT. Z mého pohledu nejzajímavějším alternativním firmwarem je OpenWRT, jelikož většina všech další firmwarů má základy a nebo dokonce je založena právě na OpenWRT.

1.2.1.1 Gargoyle

Gargoyle[4] je volně šiřitelný alternativní firmware založený na platformě OpenWRT podporující dvě nejběžnější čipové sady Atheros a Broadcom.

Kromě standardních funkcí, které má běžný tovární firmware, Gargoyle umožňuje přístup do a konfiguraci přes SSH. Další z rozšiřujících funkcí byla možnost konfigurace OpenVPN a pár dalších rozšíření.

Cílem toho projektu bylo vytvoření firmwaru s možností pokročilejší konfigurace pro zkušenější uživatele.
Gargoyle je stále živým projektem a vývoj se odvíjí podle aktualizací OpenWRT. Pravděpodobně vývojáři Gargoyle pouze udržují aktuální jádro firmwaru.

1.2.1.2 DD-WRT

DD-WRT[5] firmware je téměř na stejné úrovní jako je OpenWRT firmware z hlediska vývoje a dostupného rozšiřitelného software.

Momentálně je rozdělený na dvě verze. Jedna komerční – placená a druhá komunitní, která je zdarma. Nevím, zda-li je rozdíl ve funkčnosti verzí, ale na oficiálních stránkách DD-WRT lze pořídit i některé síťové prvky přímo pro DD-WRT.

1.2.1.3 OpenWRT

OpenWRT[6] je velmi rozšířená GNU/Linux distribuce pro různé bezdrátové routery.
Vlastnosti:

- zdarma a jeho všechny kódy budou vždy otevřené pro kohokoliv, kdo by je chtel měnit, upravovat a dále publikovat jako nové dílo
- o celý projekt se stará komunita, není zde žádná společnost, která by udávala směr, kam se bude vývoj ubírat
- obsahuje správce balíčků, který momentálně obsahuje přes 2000 různých balíčků pro rozšíření

- podporuje mnoho čipových sad, proto je možná ho nainstalovat na velké množství routerů

Jednou z největších výhod OpenWRT je správce balíčků, díky kterému lze instalovat další rozšiřující komponenty podobným způsobem jakým jsme zvyklí instalovat balíčky v jiných linuxových distribucích.

1.2.2 Shrnutí poznatků a výběr konkrétního z nalezených alternativních firmware

I přesto, že OpenWRT a DD-WRT se jeví jako dva velmi konkurenční firmware. Pro mě byla volba jednoduchá. Stoprocentně se přikláním k OpenWRT, jelikož je zde rozsáhlá komunita starající se o dokumentaci a vývoj firmwaru pro nejrůznější konkrétní typy routerů.

Velmi dobře propracovaná dokumentace mi pomohla vyřešit problémy, které vznikli během mých experimentálních pokusech konfigurace. Další výhodou dobré dokumentace bylo usnadnění výběru bezdrátového routeru, na kterém určitě bez větších problémů bude fungovat captive portal.

1.2.3 Výběr a porovnání vhodných wifi routerů pro OpenWRT

Na základě studování dokumentace o podporovaných zařízeních jsem vydedukoval, že nejoblíbenější u ostatních uživatelů OpenWrt jsou wifi routery značky TP-LINK. Nabídka routerů od tohoto výrobce se cenově pohybuje od cca 400 Kč až po téměř 5000 Kč, což nám dává poměrně velký výběr. Všechny orientační ceny použité v této práci byly přebrány z internetového obchodu `www.alza.cz`.

Podle hardwarových specifikací routerů jsem usoudil, že úplně nejlevnější modely wifi routerů pro nás nejsou vhodné, jelikož jsme zde limitování interní pamětí a u levných routerů lze pouze nainstalovat firmware bez dalších doplňků. Dokonce některé firmwary zkompilované přímo pro konkrétní wifi routery nemají v základu ani webové uživatelské rozhraní[4], na které jsme běžně zvyklí a OpenWRT má v těchto případech pouze možnost konfigurace pomocí SSH[5] a nebo Telnetu[6].

[4] Uživatelské rozhraní = User Interface – dále v práci bude použita zkratka UI

[5] Secure Shell – šifrovaný síťový protokol využívaný ke správě systému pomocí příkazové řádky

[6] Telecommunication Network – nezabezpečený síťový protokol umožňující připojení se ke vzdálenému počítači

Asi to nejlepší, co TP-LINK nabízí v oblastí domácích wifi routerů, se nachází v nabízené řadě Archer[7]. Zde jsou v nabídce 4 modely:

- TP-LINK Archer C9 cca 5000 Kč
- TP-LINK Archer C8 cca 4200 Kč
- TP-LINK Archer C7 cca 3800 Kč
- TP-LINK Archer C5 cca 2800 Kč

Všechny wifi routery v této řadě nabízí Dual-Band (2.4GHz + 5GHz) s různými rychlostmi. Čím dražší router, tím vyšší rychlost v obou pásmech. Každý router je vybaven třemi anténami, 2x USB[8] 2.0, 1xGWAN a 4xGLAN porty.

Mně se 2800 Kč za router zdálo příliš mnoho a proto jsem se porozhlédnul ještě v nižší cenové kategorii do 2000 Kč. Zde jsou v nabídce následující dva routery:

- TP-LINK TL-WDR4300 cca 2000 Kč
- TP-LINK TL-WR1043ND cca 1600 Kč

1.2.3.1 Tabulkové porovnání routerů značky TP-LINK

Bezdrátové standardy IEEE 802.11 specifikace[7]:

- 802.11a – 5GHz, 54Mb/s
- 802.11b – 2.4GHz, 11Mb/s
- 802.11g – 2.4GHz, 54Mb/s
- 802.11n – 2.4/5.0GHz, 100Mb/s a více
- 802.11ac – 2.4/5GHz, 600Mb/s a více

Tabulka 1.1: Porovnání hardwarových specifikací routerů

Typ	Flash (MB)	Ram (MB)	CPU (MHz)	Standardy IEEE
`Archer C5`	16	128	720	a/b/g/n/ac
`TL-WDR4300`	8	128	560	a/b/g/n
`TL-WR1043ND`	8	64	720	b/g/n

[7]Produktové označení série routerů značky TP-LINK

[8]Universal Serial Bus – standardizovaná sběrnice pro připojování rozšiřující periferií

Z tabulky 1.1, ve které porovnáváme TP-LINK routery, můžeme usuzovat, že se jedná o lepší routery s lepší hardwarovou výbavou. Pro představu uvedu hardwarovou specifikaci obyčejnějšího routeru v cenové kategorii do 800 Kč.

TP-LINK WR740N CPU = 400MHz, Flash = 4MB, Ram = 32MB

Z čehož vidíme, že tento levný router má poloviční parametry. Pro nás je důležitá velikost FLASH paměti, jelikož zde je uložen firmware, který vyplňuje celou kapacitu paměti. OpenWRT je navržený tak, že vyhrazuje prostor pro interní aplikace kam lze instalovat požadovaná rozšíření. Ideální router by tedy byl ten nejdražší `Archer C5`, nicméně cena tohoto routeru mně přijde dost vysoká vzhledem k tomu, že se jedná o testovací zařízení a proto jsem nakonec zvolil nejlevnější router `TL-WR1043ND`, který bude dle mého názoru plně dostačovat.

1.3 Analýza a výběr dostupného captive portalu pro OpenWRT

OpenWRT uvádí ve své dokumentaci[8] několik captive portalů, které jsou dostupné přímo z centrálního repozitáře[9].

NoCatAuth je jednoduchá implentace captive portalu v jazyku Perl nabízející jednoduchou úvodní obrazovku, která slouží k přihlašování uživatelů do wifi sítě. Toto řešení již není dlouhou dobu vyvíjeno a nemá tak žádnou podporu pro nejnovější verzi OpenWRT.

NoCatSplash je obdobné řešení jako výše zmíněný Nocatauth s tím rozdílem, že neposkytuje žádné možnosti přihlašování uživatelů. Slouží pouze jako jednoduchá webová stránka, na které musí uživatel sítě přijmout podmínky užití, aby se mohl využívat internetové připojení.

NoDogSplash je další alternativou k NoCat řešení. Nodogsplash nabízí více možností nastavení jako například nastavení hranic rychlostí připojení uživatele a nebo nastavení povolených portů.

WifiDog vychází z NoCatAuth a NoCatSplash řešení. Jedná se captive portal, který je rozdělený do dvou částí. První část je implementována v jazyky C a její funkce je přesměrovávat uživatelská připojení v závislosti na to, zdali byli autentizováni či přijali podmínky užití sítě. Druhá část implementuje

[9] Datové úložiště pro balíčky programů

autentifikační server, který slouží jako uživatelské rozhraní pro správu. Autentifikační server je implementován v PHP[10] pro webové rozhraní a pro uchování dat uživatelských účtů je zde PostgreSQL[11] databáze.

ChilliSpot je byl vytvořen jako open source captive portal, ze kterého vychází řada dalších řešení. Pro autentizaci, autorizaci a správu uživatelských účtů (AAA) lze využít podpory RADIUS[13] serveru. Bohužel tento captive portal již momentálně nemá žádnou podporu a proto může být zastaralý.

PepperSpot implementuje open source jádro ChilliSpotu a rozšířujeho o podporu IPv6[14]

CoovaChilli je pro mně nejzajímavější implementací ChilliSpotu, jelikož ho rozšiřuje o řadu užitečných funkcí. Stejně jako ChilliSpot pro jako AAA protokol je zde podpora napojení na RADIUS serveru. CoovaChilli implementuje jednoduché JSON API, které umožňuje komunikovat s tímto rozhraní jiným službám.

Captive portalů je celá řada, ale jen pár z výše popsaných je podporováno OpenWRT. Některé z těch co jsem našel i mimo nejsou již vyvíjeny a proto jsem se o nich vůbec nezmiňoval. Z mého hlediska k naším účelům nejlépe bude vyhovovat CoovaChilli, který dosud stále vyvíjen pro různé linuxové distribuce a právě i pro OpenWRT.

1.4 Shrnutí analýzy

Analýza nám ukázala, že nejvhodnějším řešením z hlediska rozšířenosti a jednoduchosti je použít firmware OpenWRT. Již dříve jsem si zkoušel tento firmware instalovat na svůj domácí router, nicméně můj zájem byl čistě ze zvědavosti a nikdy své znalosti neprohluboval hlubší konfigurací.
Bohužel jsem zjistil fakt, že aktuálně v repozitáři OpenWRT je zastaralá verze CoovaChilli captive portalu, což mě po hlubším zkoumání vedlo k tomu, že jsem musel využít vývojářského testovacího nástroje OpenWRT a zkompilovat si nejnovější verzi captive portálu sám.

[10] PHP je skriptovací programovací jazyk, který je využíván pro implementaci funkční logiky pro webové aplikace

[11] PostgreSQL je open source[12] objektově-relační databázový systém

[13] RADIUS je zabezpečená implementace AAA protokolu sloužící k ověřování uživatelských přístupů

[14] IPv6 je IP protokol verze 6

Kapitola 2

Návrh

V této kapitole se budeme věnovat již vlastnímu návrhu celého řešení z hlediska výběru technologií a popisu logiky fungování sytému. Nemalá část bude pojednávat o návrhu datového modelu a k němu připojených RESTful webovým službám. V závěru kapitoly představím navrhovaná uživatelská rozhraní pro mobilní a webovou administrátorskou aplikaci.

2.1 Výběr technologií

Technologie, které volím pro tuto práci, budou z velké části vycházet z mých předešlých zkušeností.

2.1.1 Výběr databázového systému

Pro tuto práci jsem zvolil databázový systém MySQL. Jedná se o objektově-relační databázový multiplatformní systém vyvinutý firmou MYSQL-AB, kde je využit pro komunikaci jazyk SQL[15]. Konfigurace MySQL serveru není nijak složitá a pro naše řešení jeho možnosti budou dostačující.

2.1.2 Volba aplikačního serveru

Jelikož webová aplikace a RESTful webové služby budou implementovány v jazyku Java, i aplikační server musím zvolit s podporou toho jazyka. K dispozici je řada takovýchto webových serverů. Nejrozsáhlejší praktické zkušenosti mám s aplikačním serverem Red Hat JBoss aktuální verze 8.1 s označením WildFly. Tento aplikační server implementuje nejnovější standardy Java EE verze 7, Java API a také HTTP/JSON REST API.

[15]SQL je standardizovaný strukturovaný dotazovaný jazyk, který se využívá v relačních databázích

2.1.3 Výběr RADIUS služby

RADIUS[9] je protokol, nikoliv program, definující rozhraní, které by měla jeho implementace splňovat. Vlastnosti RADIUS protokolu jsou přesně definovány dokumentem RFC 2865[16]. Klíčové vlastnosti implementace RADIUS serveru jsou následující:

Model klient/server

Network Access Server (dále jen NAS) musí operovat jako klient RADIUS serveru. NAS je odpovědný za předání uživatelských údajů příslušnému RADIUS serveru.

Síťová bezpečnost

Všechny transakce mezi klientem a RADIUS server jsou autentizovány skrze sdílený klíč, který není nikdy posílán samostatně přes síť v nešifrované formě. Všechny uživatelské informace musejí být zasílány šifrovaně.

Flexibilita autentizačních mechanizmů

RADIUS server může podporovat různé autetifikační metody. Pokud je autentizace prováděna pomocí uživatelského jména a hesla, mohou být implementovány PPP PAP nebo CHAP, UNIX[17] uživatelské účty a další.

V dnešní době existuje několik rúzných implementací RADIUS protokolu. Pro tuto práci jsem zvolil FreeRADIUS, což je momentálně jedena z nejrozšířenějších implementací. Například je využit v akademické síti eduroam.

2.1.4 Výběr frameworků pro J2EE aplikace

2.1.4.1 Maven

Maven není úplně až tak framework, jako standardizovaný nástroj pro vývoj aplikací sloužící ke správě projektu a k lepšímu pochopení struktury. Tento nástroj obsahuje rozsáhlý repozitář, ve které najdeme různé knihovny, které je tedy velmi jednoduché importovat do naše vlastního projektu. Struktura konfiguračních souborů, které definují závislosti a další parametry, je ve formátu XML. Definovaným názvem pro konfigurační soubor je `pom.xm`.

2.1.4.2 Hibernate JPA framework

Hibernate ORM framework slouží k mapování a následné komunikaci s relační databází. Tento framework je nezávislý na databázi, kterou použijeme,

[16] RFC normy jsou dokumenty definující implementace internetových protokolů

[17] Ochraná známka pro operační systémy, které odpovídají svojí charakteristikou operačním systémům Version 7 Unix nebo UNIX System V

jelikož komunikace probíhá pomocí JDBC řadiče. Tento framework nám velmi usnadní práci, protože bez něho bychom museli psát všechnu komunikaci ručně a pokud bychom přešli na jinou databázi, museli bychom opravovat naši implementaci aby odpovídala použité databázi.
Hibernate implementuje funkčnost frameworku JPA, jehož hlavní komponentou je takzvaný EntityManager, který má následující hlavní metody pro práci s entitami:

- PERSIST – uloží objekt do databáze – INSERT
- REMOVE – smaže objekt z databáze – REMOVE
- MERGE – v případě, že objekt již byl pesistován, změní jeho proměnné a uloží do databáze – UPDATE
- FIND – nejde požadovaný objekt v databázi a vrátí ho v případě, že byl nalezen – SELECT

2.1.4.3 JavaSever Faces – JSF

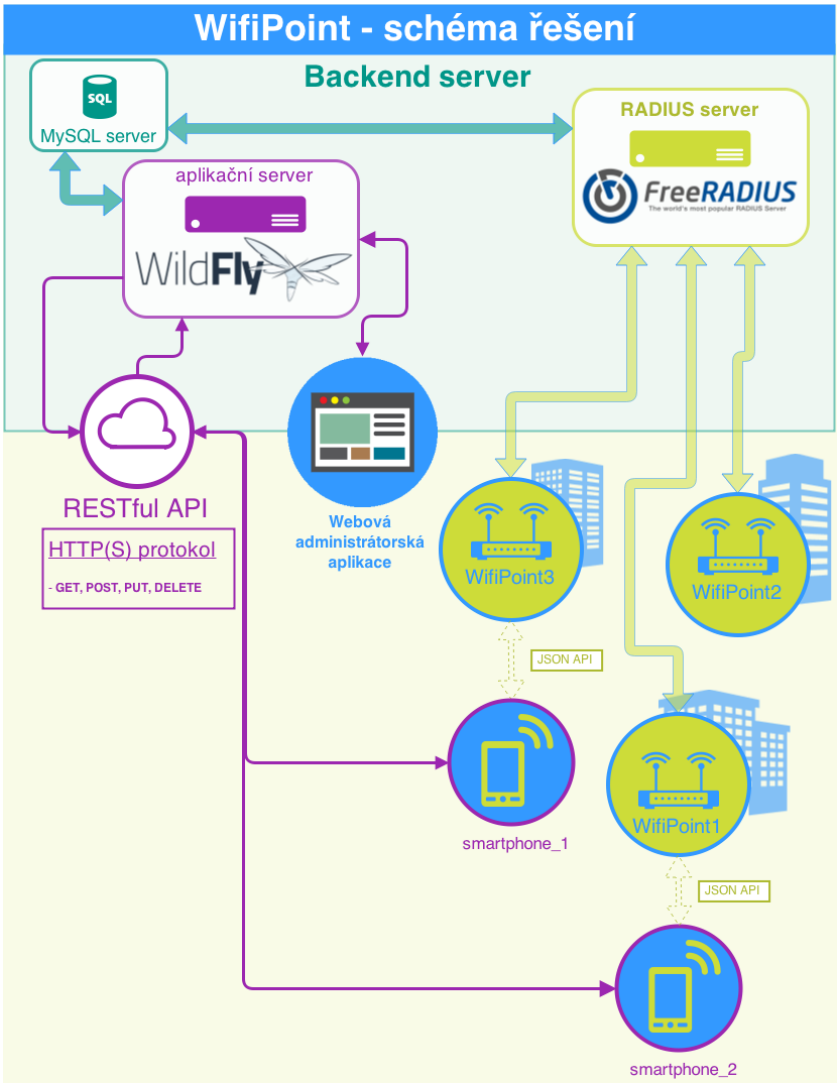
JavaServer Faces je framework pro vývoj webových aplikací. Díky tomuto frameworku je možné snadno oddělit webovou aplikaci na implementaci uživatelského rozhraní a aplikační logiky. Pro přístup k datům z aplikační logiky slouží speciální XML značky, kterými jsou předávány data pro zobrazování či editaci ze standardních Java Beans[18]. Existuje několik implementací například Primefaces, MyFaces nebo RestFaces, díky kterým můžeme měnit vzhled a funkčnost výsledné webové aplikace.
V této práci jsem vyžil pro několik prvků implementaci Primefaces.

2.1.4.4 RESTEasy framework

RESTEasy je implementace poskytující Java API RESTful webové služby. Je to nativní implementace webových služeb pro JBoss aplikační server. Tento framewok je plně certifikovaná implementace JAX-RS podle Java API specifikace.

[18]Java Beans jsou třídy, díky kterým můžeme přistupovat k instančním třídám pomocí referencí definovaných pomocí getterů a setterů

2.2 Návrh schématu struktury řešení služby



Obrázek 2.1: Schéma struktury celé služby

Obrázek 2.1 blíže nastiňuje strukturu výsledné služby jako celku včetně zobrazení wifi stanic a uživatelských smartphonů. Struktura je rozdělena na dvě části – backend server a ostatní části řešení.

Backend server je postaven na linuxovém operačním sytému Debian „Wheezy“[19], kde je konfigurován databázový server MySQL.

[19] Wheezy je označení verze 7.0 distribuce operačního systému Debian. Zajímavostí je to, že verze systému jsou pojmenovávány po postavičkách z filmu Příbeh hraček (Toy story)

Jako autentizační službu jsme výše zvolili implementaci RADIUS protokolu „FreeRADIUS“. Aplikační server pro webovou administrátorskou aplikaci a webové služby je JBoss WildFly server.

Ostatní části jsou všechny různé komponenty zajišťující funkčnost celého řešení. Níže popíši jejich význam a funkčnost v řešení.

- **RESTful API** implementuje webové služby pro mobilní aplikace, případně pro kterékoliv jiné klientské aplikace.
- **Webová administrátorské aplikace** slouží k celé správě řešení z pohledu koncového zákazníka, který využívá tuto službu. Funkčnost bude rozebráná sekci 2.5.2.
- **WifiPoint(1,2,3)** jsou routery, kde je nainstalován firmware OpenWRT a captive portal CoovaChilli. Router je nakonfigurován tak, aby komunikoval s backend serverem (konkrétně s RADIUS serverem). Název wifi sítě (SSID) je v tomto případě WifiPoint(1,2,3) a musí být shodný s názvem, který jsme zadali přes administrátorskou aplikaci do databáze. S mobilní aplikací komunikuje přes jednoduché JSON API, které již implementuje captive portal CoovaChilli. Struktura tohoto API bude upřesněna v sekci 2.4.
- **Smartphone(1,2)** jsou koncová uživatelská mobilní zařízení, která se připojují do wifi stanic. Aby se takové zařízení mohlo připojit musí mít nainstalovánu mobilní aplikaci. Detailní funkčnost aplikace bude rozebrána v následujících kapitolách.

2.3 Návrh datového modelu služby

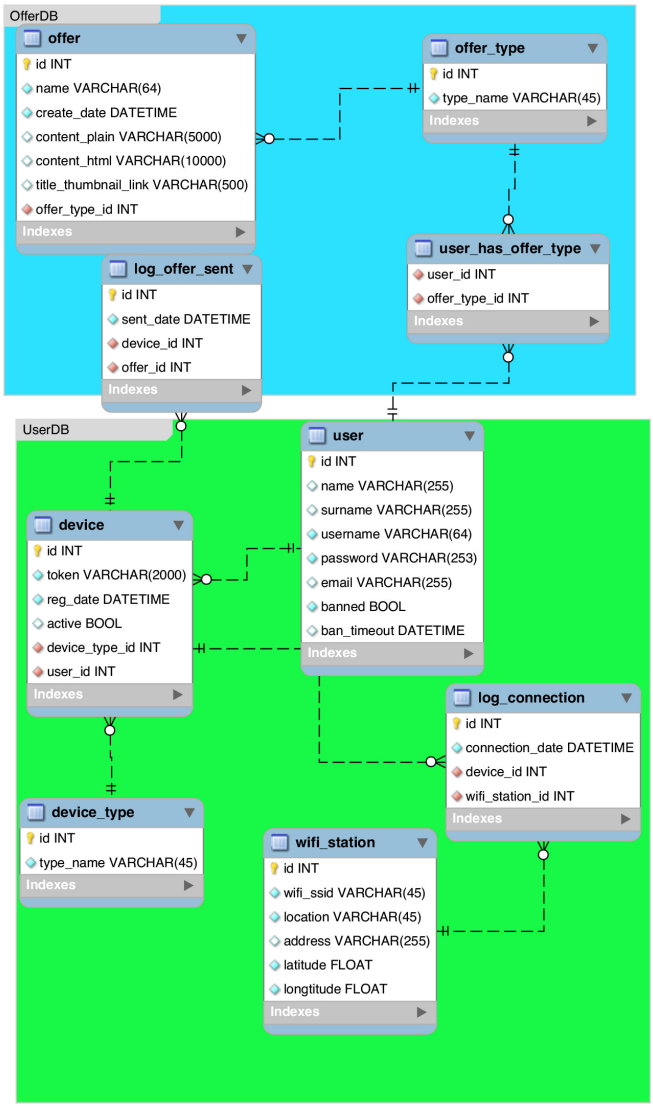
Pro návrh datového modelu a následnou tvorbu databáze, která bude sloužit pro ukládání veškerých dat aplikace, jsem pro jednoduchost vytvořil pomocí grafického nástroje MySQL Workbench. Návrh datového modelu jsem rozdělil na dvě části, jelikož část, která se týká databáze RADIUS serveru je již přesně navržená. Mým cílem však bylo sloučit tyto dva modely do jedné databáze tak, aby k databázi RADIUS serveru měla přístup i webová část řešení pomocí frameworku JPA.

2.3.1 RADIUS server datový model

Obrázek 2.2: Datový model RADIUS serveru

Obrázek 2.2 zobrazuje relační model databáze, který ukazuje strukturu tabulek, které jsou nutné pro běh RADIUS serveru nad databází MySQL. Operace nad těmito tabulkami standardně provádí RADIUS server, nicméně pro tvorbu uživatelský účtů využijeme framework JPA, který bude vytvářet uživatelské účty jak v části pro RADIUS server, tak i aplikační části.

2.3.2 Aplikační datový model



Obrázek 2.3: Návrh datového modelu pro naši aplikaci

Vlastní aplikační datový model se skládá z celkem 8 entitních tabulek a jedné vazebné. V mém modelu jsou momentálně navržené některé tabulky, které jsem nastínil k možnému rozšíření, avšak nejsou implementovány.

2.3.2.1 Definice tabulek

Tabulka offer je takzvanou „nabídkou“, což může být jakákoliv informace poskytnutá uživateli. Aktuální návrh obsahuje základní proměnné pro představení možné funkčnosti.

- **id** – primární klíč tabulky
- **name** – název nabídky
- **create_date** – datum vytvoření nabídky
- **content_plain** – jednoduchý textový popis nabídky, zkrácená verze
- **content_html** – detailní popis nabídky podporující HTML značky možné formátovaní
- **title_thumbnail_link** – hypertextový odkaz na nějaký obrázek, který může být u nabídky zobrazen
- **offer_type_id** – cizí klíč pro definování typu nabídky

Tabulka offer_type slouží k definování informačních typů nabídek. Pro představu může obsahovat data jako „Jídelní lístek“, „Novinka“, „Kultura“ a další.

id – primární klíč tabulky

type_name – název typu nabídky

Tabulka user definuje strukturu uživatelského účtu, který bude vytvořen po registraci uživatele. S vytvořením záznamu do této tabulky se zároveň vytvoří záznam do tabulky `radcheck`, která je definována strukturou datového modelu RADIUS serveru.

- **id** – primární klíč tabulky
- **name** – jméno uživatele
- **surname** – příjmení uživatele
- **username** – uživatelské jméno pro přihlášení
- **password** – heslo uživatele pro přihlášení
- **email** – email uživatele
- **banned** – hodnota typu pravda/nepravda pro nastavení zakázání přihlášení uživatele
- **ban_timeout** – datum, kdy bude uživatel mít možnost znovu využívat službu (pouze v případě, že byl zakázán přístup)

Tabulka device je nutná pro uchování unikátního údaje o zařízení, které bylo zaregistrováno do služby. Slouží pro rozesílání notifikací do mobilních telefonů uživatelů.

- **id** – primární klíč tabulky
- **token** – unikátní klíč vygenerovaný mobilní aplikací před žádostí o registraci do služby
- **reg_date** – datum registrace zařízení
- **active** – hodnota typu pravda/nepravda sloužící o deaktivaci zařízení – díky této hodnotě nebudou do toho zařízení rozesílány notifikace
- **device_type_id** – cizí klíč typu zařízení (definice tabulky níže)
- **user_id** – cizí klíč id uživatele, ke kterému je zařízení připojeno

Tabulka device_type slouží k definování typu zařízení. Předem definované záznamy ANDROID, IOS a WINDOWSPHONE. V aktuální implementaci bude pouze záznam ANDROID, jelikož aplikace pro ostatní typy mobilních operačních systému nebudou implementovány.

- **id** – primární klíč tabulky
- **type_name** – název typu zařízení

Tabulka wifistation nám poskytuje uchovávání informací, kde se která wifi stanice nachází. Zároveň tato tabulka slouží k tomu, aby mobilní aplikace dokázala učit, zda-li se má pokoušet přihlašovat do captive portalu. Postup přihlašování a ověřování bude popsán v dalších kapitolách.

- **id** – primární klíč tabulky
- **wifi_ssid** – SSID[20] wifi stanice
- **location** – lokace naší wifi stanice – může specifikovat námi zadané údaje o poloze – patro, název restaurace a další
- **address** – adresa polohy wifi stanice
- **latitude** – souřadnice pozice zeměpisné šířky, kde se wifi stanice nachází
- **longtitude** – souřadnice pozice zeměpisné délky, kde se wifi stanice nachází

Následující dvě tabulky, které budu definovat, v naší verzi projektu nebudou zahrnuty. Do budoucna budou sloužit pro ukládání informací o různých událostech. Data z těchto tabulek budou moci sloužit k zobrazování statistik.

[20]SSID je jedinečný identifikátor wifi sítě, který je zobrazen jako ASCII řetězec

Tabulka log_offer_sent slouží k uložení informace odeslané notifikaci s konkrétní nabídkou do konkrétního mobilního zařízeni.

- **id** – primární klíč tabulky
- **sent_date** – datum odeslání nabídky do zařízení
- **device_id** – cizí klíč id zařízení (definice tabulky níže)
- **offer_id** – cizí klíč id odeslané nabídky

Tabulka log_connection slouží k uložení informace o pokusu přihlášení konkrétního zařízení do wifi stanice

- **id** – primární klíč tabulky
- **connection_date** – datum odeslání nabídky do zařízení
- **device_id** – cizí klíč id zařízení
- **wifi_station_id** – cizí klíč id wifi stanice

Tabulka user_has_offer_type definuje vazbu mezi tabulkami offer_type a user. Díky této tabulce může uživatel definovat, které nabídky chce přijímat a které ne. Výchozí nastavení je takové, že uživatel přijímá všechny typy nabídek.

- **user_id** – cizí klíč id uživatele
- **offer_type_id** – cizí klíč id typu nabídky

2.4 Návrh logiky autentizace

Obrázek schématu 2.4 přesně popisuje průběh autentizace uživatele do wifi sítě. Schéma je rozděleno do následujících tří částí.

Android aplikace
Tato část implementuje hlavní funkčnost a smysl automatického přihlašování do captive portalu. Jsou zde nastíněny dvě API rozhraní.

- **REST API WifiPoint** – API webových služeb, které implementuje náš aplikační server. Toto rozhraní poskytuje CRUD[21] operace díky kterým jsme schopni bezpečně komunikovat s databází.
- **API CoovaChilli** – jednoduché JSON rozhraní pro základní komunikaci s wifi routerem.
  Rozhraní je implementováno ve skriptovacím jazyku JavaScript a poskytuje 3 typy HTTP požadavků. URL adresa těchto požadavků je odvozena z aktuální konfigurace routeru.
  Ve výchozí konfiguraci vypadá formát následovně:
  `http://<ip_adresa_výchozí_brány>:3990/json/<typ_požadavku>`
  Router přijímá pouze požadavky typu GET. Odpovědi vrácené routerem jsou vždy ve formátu JSON, což nám umožňuje snadně mapovat tyto odpovědi do Java objektů. Typy požadavků:
  - **status** – vrátí informaci o tom, zda-li je připojené zařízení autentizováno
  - **refresh** – aktualizuje proměnné v aktuálním spojení, které jsou používány pro autentizaci a jejich odposlechnutí by znamenalo bezpečností chybu
  - **logon** – slouží k přihlášení uživatele do sítě přes RADIUS server. Požadavek vyžaduje následující 2 URL parametry:
    * **username** – uživatelské jméno
    * **response** – šifrovaná odpověď odesílána RADIUS serveru při pokusu o přihlášení uživatele.

WIFI router
Router, který chceme začlenit do naší služby, musí splňovat následující:

- **OpenWRT** – nutná instalace a konfigurace routeru tak, aby byl schopný komunikovat s RADIUS serverem
- **Captive portal CoovaChilli** – nutná instalace a konfigurace tohoto softwaru, přes který se autentizují uživatelé do wifi sítě

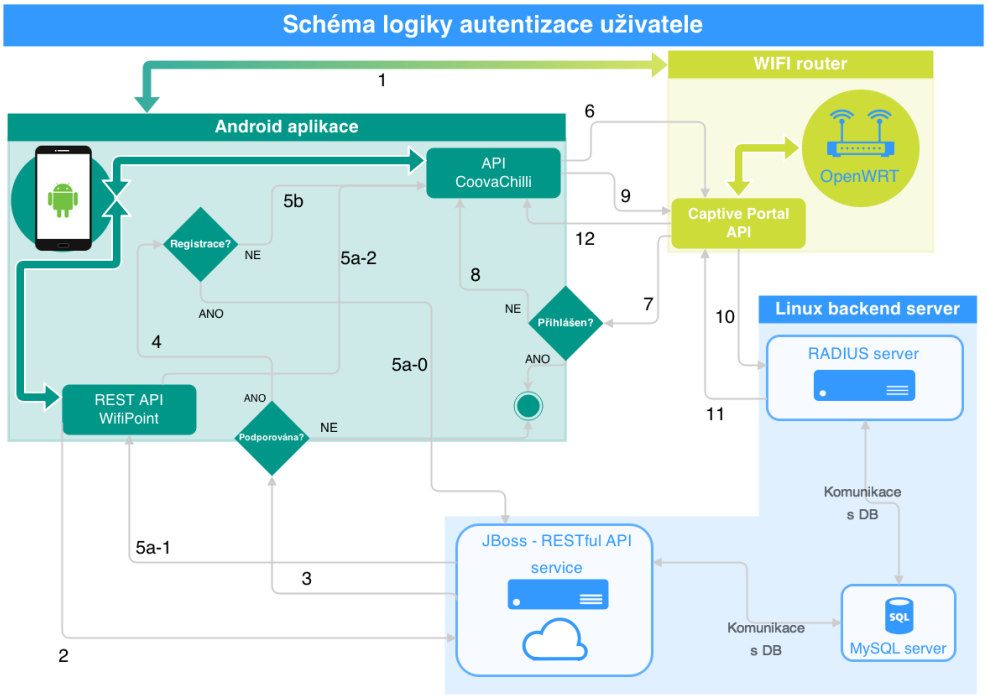
[21] CRUD operace – čtení, zápis, editace a mazání

V případě splnění požadavků, musíme ještě tento router přidat databáze přes webovou aplikaci. Pokud tento krok neučiníme, router se bude jevit při autentizaci jako neznámé wifi zařízení a mobilní aplikace se ani nepokusí přihlásit.

Linux backend server
Backend server při autentizaci uživatele slouží jako webová služba pro mobilní aplikaci a jako autentizační služba pro wifi router.

2.4.1 Popis schématu autentizační logiky řešení

Vysvětlivky ke schématu
V schématu 2.4 jsou jednotlivé procesy značeny popsanými hranami grafu. Nepopsané hrany grafu nejsou pro nás v případě popisu autentizace podstatné. Hrany mezi JBoss – RESTful API service, RADIUS server a MySQL server vyjadřují komunikaci služeb s databází. Detailnější popis není v tomto případě nutný. Schéma je pro jednoduchost minimalizováno na klíčové kroky autentizace a detailnější průběh procesů je nastíněn v popisu.



Obrázek 2.4: Schéma autentizační logiky řešení

Proces 1

Přípojení smartpohonu k wifi routeru. Aplikace zpracuje informace o změně internetového připojení a vyhodnotí, zda-li je zařízení připojeno přes wifi anténu. Pokud ano pokračuje v autentizaci. V případě mobilního připojení nemá smysl se pokoušet autentizovat uživatele. Při tomto procesu je získáno SSID wifi sítě, které slouží k určení začleněných wifi sítí.

Proces 2

Zpracování informací o wifi připojení. Z předchozího procesu jsme získali informaci o SSID wifi sítě a aplikace odesílá HTTP[10] požadavek pro zjištění zda-li wifi síť je podporována. V případě chyby připojení proces autentizace končí.

Proces 3

Aplikační server přijme požadavek a vyhledá danou wifi stanici podle SSID v databázi. Pokud najde wifi stanici, pak odesílá klientovi odpověď s nalezeným objektem ve svém těle ve formátu JSON. Pokud nenajde, odpověď je prázdná.

Rozhodovací proces „Podporována?“

Aplikace přijímá odpověď.

- **ANO** – nachází v těle odpovědi objekt wifi stanice a pokračuje na další krok
- **NE** – daná wifi není podporována, autentizace končí

Proces 4

Aplikace zjistí, zda-li její vnitřní databáze má již uložené nějaké přihlašovací údaje. Předává informaci dále.

Rozhodovací proces „Registrace?“

- **ANO** – aplikace nachází přihlašovací údaje a pokračuje na krok 5b
- **NE** – aplikace nenašla žádné přihlašovací údaje – pokračuje na krok 5a-0. Odesílá požadavek na vytvoření nového uživatelského účtu pro přístup do služby.

Tento proces **nenastává** v případě, že se aplikace snaží přihlásit na pozadí automaticky bez uživatelské interakce. Proto se tento proces spouští pouze do té doby, než se uživatel zaregistruje.

Proces 5a-0

Server přijímá požadavek a vytváří v databázi uživatele. Nejprve v tabulce `user` a následně v tabulce `radcheck`. Zpět je vrácena odpověď, kde tělo obsahuje objekt nově vytvořeného uživatele.

Proces 5a-1

Aplikace uloží nově vytvořeného uživatele do své interní databáze a pokračuje na další proces.

Proces 5b + 5a-2

Oba procesy získají z předešlých kroků přihlašovací údaje.

Proces 6

Rozhraní API CoovaChilli odesílá požadavek na router za účelem získání informací o stavu přihlášení uživatele.

Proces 7

Router přijímá požadavek o zjištění stavu konkrétního zařízení. A zpět odesílá formátované informace v těle odpovědi.

Rozhodovací proces „Přihlášen?“

Aplikace vyhodnocuje, na základě hodnoty `clinetState`, zda-li je již zařízení přihlášeno.

- **ANO** – proces končí
- **NE** – aplikace přechází na další krok

Proces 8

Aplikace přijímá informaci o stavu připojení a vytváří šifrovanou odpověď pro API routeru. Šifrovací metoda přijímá parametry `heslo` a `challenge`. `Challenge` je náhodný, ale vhodný, unikátní řetězec vygenerovaný rozhraním routeru.

Proces 9

Z předchozího procesu aplikace získává vypočtenou `response`, která slouží pro autentizaci. Sestavuje požadavek obsahující uživatelské jméno a `response`, který je následně odeslán na rozhraní wifi routeru.

Proces 10

Router přijímá požadavek o autentizace. Předává získané parametry RADIUS klientovi, který se pokouší provést autentizaci uživatele.

Proces 11

RADIUS server vyhodnocuje požadavek o přihlášení a odpovídá RADIUS klientovi, který zpracovává odpověď a vyhodnocuje, zda-li uživatel byl autentizován. Po zpracování předešlých informací vytváří odpověď, kterou odesílá zpět mobilní aplikaci.

Proces 12

Aplikace přijímá odpověď a vyhodnocuje aktuální stav. Pokud je uživatel úspěšně přihlášen zobrazuje notifikaci o úspěšném přihlášení do wifi sítě služby. V případě že nebyla autentizace úspěšná čeká na další pokus o autentizaci.

2.5 Návrh uživatelského rozhraní

Standardem při návrhu uživatelského rozhraní pro aplikace je tvorba wireframů[22], které pouze zobrazují rozmístění prvků aplikace a definují funkčnost aplikace. Slouží především jako pracovní návrh, na kterém se demonstruje logika chování aplikace. Zjišťuje se na nich, zda-li rozhraní je navrženo vhodně pro koncové uživatele nebo jestli vyžaduje nutné úpravy.

Jakmile je návrh dostatečně otestován ve formě wireframů, je předán grafikovi uživatelského rozhraní, který z wireframů vytvoří grafický návrh aplikace. Grafický návrh aplikace již zobrazuje to, jak by měla aplikace vypadat ve finální podobě.

Ve své práci používám rovnou grafické návrhy, které vznikly částečně před, ale i během implementace, jelikož mi nepřišlo nutné se momentálně zabývat wireframy. V případě budoucího rozvoje této práce by bylo vhodné předělat uživatelské rozhraní a otestovat vhodnost návrhu rozhraní.

Cílem této práce nebylo vytvořit hotové aplikace a proto funkční řešení ukazují pouze klíčové vlastnosti, čemuž odpovídají i grafické návrhy.

[22]Wireframe je takzvaný „drátěný“ model návrhu uživatelského rozhraní

2.5.1 Návrh vzhledu mobilní aplikace

Při návrhu uživatelského rozhraní mobilních aplikací vím ze své zkušenosti, že je dobré držet se standardů pro danou platformu. Mobilní aplikaci budu programovat pro Android, budu se tedy držet standardů, na které jsou uživatelé zvyklí.
Všechny grafické ikony či obrázky jsem buďto sám vytvořil a nebo získal zdarma z veřejně dostupných zdrojů.

Obrázek 2.5: Návrh vzhledu Android aplikace

Z obrázku 2.5 vidíme, že mobilní aplikace se skládá ze 4 pohledů.

Pohled registrace
Obrazovka registrace se bude uživateli zobrazovat pouze do té doby, než se zaregistruje. Po registraci se nadále nezobrazuje. Obsahuje standardní pole pro vyplnění přihlašovacích údajů. Při zaškrtnutí pole pro anonymní přihlášení jsou vygenerovány náhodné údaje a uživatel bude přihlášen.

Pohled přihlašování
Tato obrazovka indikuje uživateli, že probíhá nějaké ověřování údajů. Může to být buď registrace uživatele a nebo ve většině případů přihlašování uživatele.

Pohled příspěvky
Hlavní obrazovka, kam by se náš uživatel měl nejčastěji vracet a sledovat ji. Jedná se o zobrazení nabídek, které můžeme rozesílat uživatelům. Aktuální funkčností této obrazovky je pouze prohlížení všech příspěvků a zobrazení jejích detailů. Řazení příspěvků je chronologické od nejnovějšího nahoře až po nejstarší. Na tomto pohledu to není zřejmé, ale uživatel by měl mít možnost aktualizovat příspěvky pomocí „pull to refresh“, což dotykové gesto, kdy se snažíme „stáhnout“ seznam příspěvků směrem dolů. Momentálně je to hojně využívaná metoda uživatelské aktualizace dat. Každý zobrazený příspěvek má svůj náhledový obrázek, název, zkrácený popis nabídky a datum. Barevnost jednotlivých položek na této obrazovce se liší podle typu nabídky.

Pohled detail příspěvku
Každý příspěvek si můžeme v tomto návrhu „přiblížit“ zobrazením detailu celé nabídky, jelikož hlavní obrazovka příspěvků neobsahuje rozšířený popis nabídky. Detailní popis nabídky může být ve formátu HTML kódu, díky čemuž máme možnost různě formátovat text už při vytváření nabídky v administrátorské aplikaci.

2.5.2 Návrh vzhledu webové administrační aplikace

Pro webovou aplikaci budeme využívat framework Primefaces[23], který nám velmi zjednoduší implementaci a zároveň k tomuto frameworku jsem použil volně dostupnou šablonu pro webové stránky „Admin LTE“[11], která je určena pro tvorbu administračních rozhraní. Zde jsem tedy ve většině případů využil standardních prvků, které nám šablona nabízí. Pro některé komponenty se mi zdálo vhodnější použít prvky z frameworku Primefaces – například pro tabulková zobrazení dat.

[23] PrimeFaces implementace Java server faces

Použitím již existující šablony nelze považovat tvorbu vzhledu webové aplikace za můj vlastní návrh. Proto i pro zobrazení jednotlivých částí aplikace použiji screenshoty[24] z již hotové aplikace.

Šablona aplikace se skládá ze tří částí. Horní nástrojová lišta, levý navigační panel a centrální obsah. Obsah nástrojové lišty a navigačního panelu je v našem návrhu neměnný, obě části jsou staticky definovány ve zdrojovém kódu.

Finální implementace této webové aplikace bude velmi jednotvárná. Jednotlivé části administrace se budou ve všech případech téměř podobat, proto si dovolím v návrhu detailněji popsat pouze některé z nich. Případně všechny screenshoty webové aplikace jsou umístěny v elektronické příloze této práce.

Přihlášení
Obrázek 2.6 zobrazuje přihlašovací formulář pro správce aplikace. Momentálně zde není žádná možnost registrace uživatelů do systému, jelikož aktuální konfigurace aplikačního serveru nepodporuje správu uživatelských účtů.

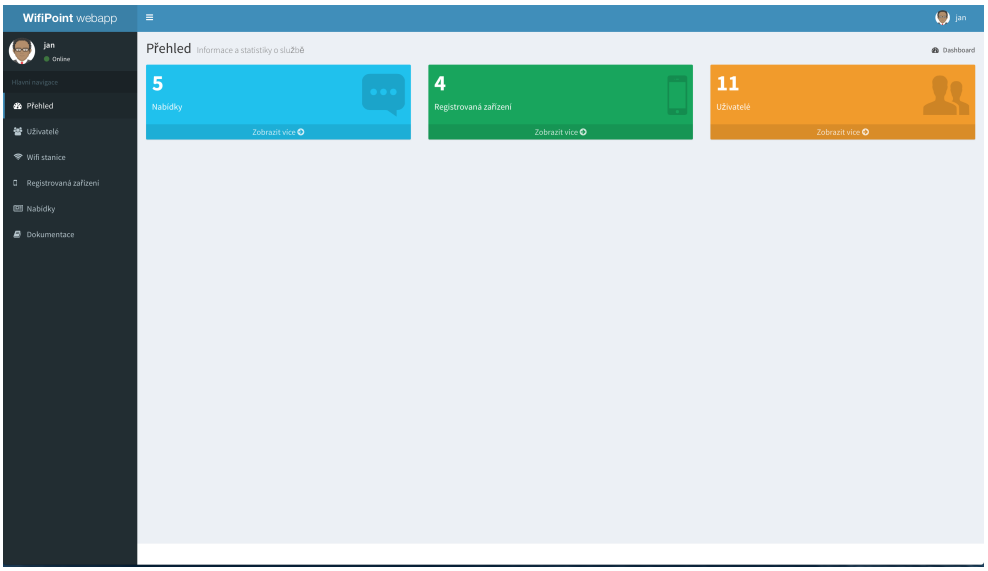
Obrázek 2.6: Návrh vzhledu administrátorské aplikace – přihlášení

[24]Screenshot je snímek obrazovky – tento termín je běžně používán jako termín, proto dále budu používat

Přehled

Po úspěšném přihlášení je administrátor přesměrován na úvodní obrazovku „Přehled (2.7)“, která v takovém to typu aplikací slouží jako souhrnný pohled na všechny důležité či užitečné informace. Náš přehled zobrazuje pouze informace o celkovém počtu nabídek, registrovaných zařízení a o celkovém počtu uživatelských účtů. Avšak smyslem by mělo být především zobrazení různých grafů orientovaných na statisticky zajímavé údaje a mnoho dalších informací, které by se měli objevit v administračních systémech. Takovéto úpravy mohou být jedním z vhodných rozšíření webové aplikace.

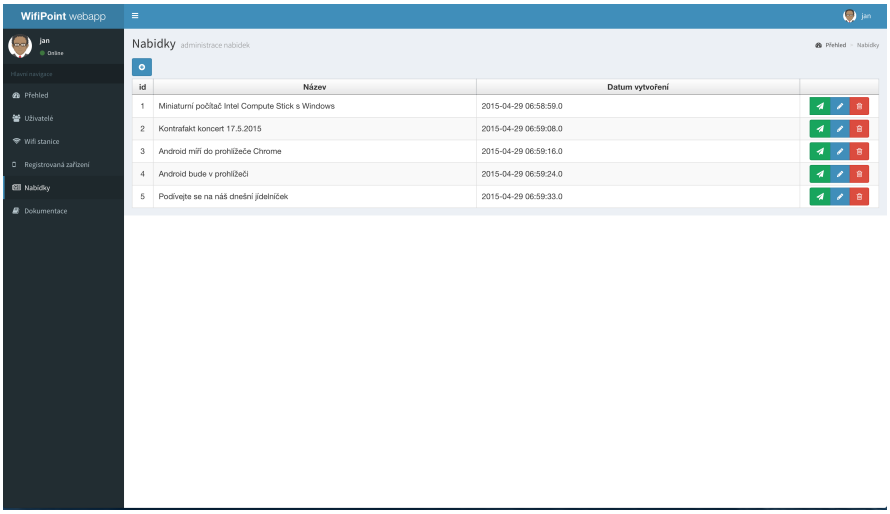
Obrázek 2.7: Návrh vzhledu administrátorské aplikace – přehled



Správa nabídek

Screenshot na obrázku 2.8 ukazuje jednoduchý správce nabídek. Nabídky můžeme přidávat, editovat, mazat a rozesílat. Standardní operace nebudu dále rozvádět. Zajímavým prvkem v je dle mého názoru tlačítko pro rozesílání notifikací na jednotlivé nabídky. Toto tlačítky by jako základní funkci mělo umožňovat rozeslat vybranou nabídky jako notifikaci do mobilních zařízení. Zároveň rozeslání vybrané nabídky určitého typu by mělo být doručeno pouze těm uživatelům, kteří mají zájem dostávat tento typ upozornění.

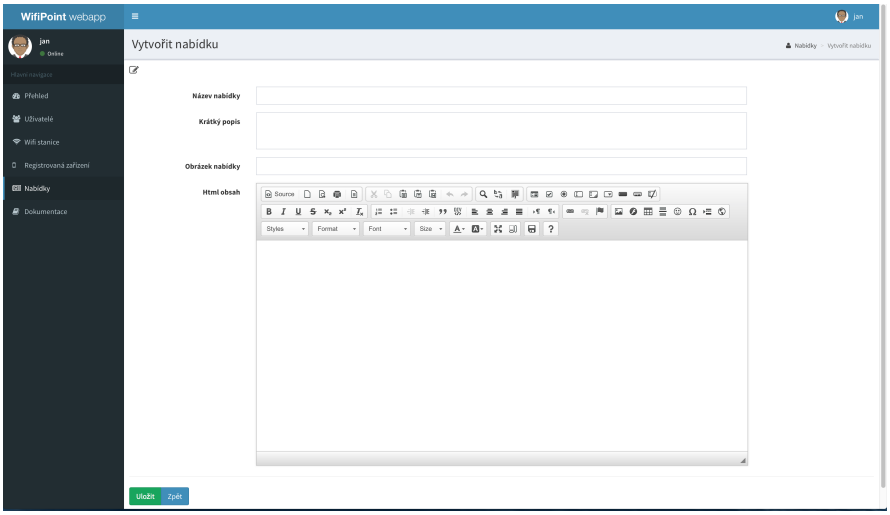
Obrázek 2.8: Návrh vzhledu administrátorské aplikace – nabídky



Tvorba nové nabídky

Vytvoření nové nabídky je zobrazeno v obrázku 2.9. Jedná se o formulář, kde vyplní administrátor požadovaná pole a následně uloží nabídku. Nabídka se pouze přidá do databáze, neodešle se notifikace. Možným rozšířením můžou být různé parametry co udělat po uložení nabídky. Například okamžité odeslání notifikace uživatelům a nebo zajímavým rozšířením by bylo načasování rozeslání notifikací na určitý čas a datum.

Obrázek 2.9: Návrh vzhledu administrátorské aplikace – nová nabídka



Ostatní komponenty aplikace
Zbývající pohledy mají stejnou strukturu a logiku jako celý pohled na správu nabídek. Pro účely této práce bude stačit jejich stručné popsání významu a funkčnosti v aplikaci.

- **Správa uživatelů** slouží převážně k přehledu aktivních uživatelských účtů či k jejich případným úpravám. Užitečná funkcionalita v této správě je nastavení zakázání přístupu do wifi sítě. Například pokud uživatel poruší podmínky užívání wifi sítě, může administrátor aplikace dočasně zakázat přístup.

- **Správa stanic** umožňuje vytvářet, mazat a upravovat wifi routery, které jsou podporovány a lze se do nich přihlašovat pomocí mobilní aplikace za účelem využívání služby a internetového připojení.

- **Správce zařízení** v aktuální návrhu pouze zobrazuje registrovaná zařízení, která budou přijímat notifikace. Význam tohoto pohledu byl pouze za účelem testování a ověřování správné komunikace mobilní aplikace se serverem.

- **Dokumentace** je pohled obsahující dokumentaci pro používání celého řešení. Aktuálně je dokumentace prázdná.

Kapitola 3

Instalace a konfigurace

Jako backend server jsem zvolil linuxovou distribuci Debian „Wheezy“. Před konfigurací jednotlivých programů, jsem zvyklý si všechny nainstalovat a pak je postupně konfigurovat. Pro instalaci jsem vytvořil jednoduchý bash[25] skript (Zdrojový kód [src: 1]).

```
1  #!/bin/bash
2  #aktualizace seznamu dostupných programů v nastavených
3  #   repozitářích balíčkovacího systému
4  sudo apt-get update
5  #instalalce MySQL serverovské a klientské aplikace
6  sudo apt-get install mysql-server mysql-client
7  #instalace implementace RADIUS serveru FreeRadius
8  sudo apt-get install freeradius
9  #instalace ovladačů pro komunikaci freeradiusu a MySQL databáze
10 sudo apt-get intall freeradius-mysql
11 #instalace vývojového balíku pro Javu
12 sudo apt-get install openjdk-7-jdk
13 # instalace příkazového editoru vim -
14 #   vhodný pro editaci konfiguračních souborů
15 sudo apt-get install vim
16
17
18 #Instalace jboss wildfly serveru
19 JBOSS_HOME="/usr/local/share/jboss_8_2"
20 mkdir -p $JBOSS_HOME
21 cd /tmp/
22 wget http://download.jboss.org/wildfly/8.2.0.Final/wildfly-8.2.0.Final.tar.gz
23 tar -xpvzf wildfly-8.2.0.Final.tar.gz -C $JBOSS_HOME --strip 1
```

Zdrojový kód [src: 1]: Instalační bash scrip

[25] Bash – příkazový „procesor“ interpretující příkazový řádek, lze tvořit skripty vykonávající zadané příkazy

3.1 Konfigurace databázového serveru

Instalace databázového serveru MySQL vytvoří standardní schéma struktury spouštěcích a konfiguračních souborů, které je definované pro Debian. Na konci instalace je MySQL server spuštěn a lze se k němu připojit pomocí MySQL klienta. Výchozí konfigurace serveru je nastavená tak, aby server přijímal požadavky pouze z lokální adresy `127.0.0.1`, což znamená, že se k němu lze připojit pouze ze serveru, kde je nainstalován. Pro naše účely nemusíme nic měnit, jelikož chceme, aby se do databáze mohli připojovat pouze aplikace běžící na stejném serveru.

Na databázovém serveru vytvoříme databází podle našeho návrhu z MySQL Workbench. Z MySQL Workbench aplikace vyexportujeme SQL skript, který nám vytvoří navrženou datábázi na serveru. Skript spustíme pomocí následujícího příkladu :

```
mysql -u root -p < create_scheme.sql
```

kde „root“ je uživatel databázového serveru. Tento příkaz vyžaduje zadání hesla užizvatele. Po ukončení SQL skriptu máme připravenou databázi pro napojení ostatních aplikací, kde se nám vytvořilo schéma s názvem „wifipointdb“.

3.2 Konfigurace webového serveru pro J2EE aplikace

Instalace aplikačního serveru JBoss Wildfly[26] spočívala pouze ve stažení archivu z oficiálních zdrojů. Následně již můžeme aplikační sever spustit, ale nepoběží nám jako služba, například stejně jako MySQL server. Pro konfiguraci jako službu musíme provést následující příkazy[src: 2].

[26] Dále jen „aplikační server“

```
# přidání nového uživatele pro spuštění serveru
$ adduser --system --group --disabled-login wildfly
$ JBOSS_HOME="/usr/local/share/jboss_8_2"
# změna vlastníka a skupiny domovského adresáře aplikačního serveru na
 ↪  uživatele, pod kterým bude spouštěn server
$ chown -R wildfly:wildfly /usr/local/share/jboss_8_2
# vytvoření symbolického odkazu na spouštěcí skript serveru
$ ln -s /usr/local/share/jboss_8_2/bin/init.d/wildfly-init-debian.sh
 ↪  /etc/init.d/wildfly
# zkopírování spuštěcího konfiguračního souboru pro aplikační server
$ cp /usr/local/share/jboss_8_2/bin/init.d/wildfly.conf /etc/default/wildfly
# pomocí editoru vim v konfiguračním souboru nastavím hodnotu proměnné
 ↪  JBOSS_HOME="/usr/local/share/jboss_8_1/"
$ vim /etc/default/wildfly
# pomocí programu update-rc.d přidáme spoutěcí skript aplikačního serveru do
 ↪  systémových skriptů
$ update-rc.d /etc/init.d/wildfly defaults
```

Zdrojový kód [src: 2]: Konfigurace aplikačního serveru jako systémové služby

Vlastní konfigurace aplikačního serveru je umístěna v souboru s názvem `standalone.xml`.

Konfigurační soubor `standalone.xml` obsahuje rozsáhlou celého aplikačního serveru. V tomto případě budu konfigurovat pouze nejdůležitější části.

HTTPS spojení

```
# vygenerování self-signed certifikátů
$ keytool -genkey -keyalg RSA -alias selfsigned -keystore server.jks
 ↪  -storepass password -validity 360 -keysize 2048

#editace konfigurčního souboru standalone.xml
.....
<security-realms>
    ...
    <ssl protocol="TLS">
        <keystore path="server.jks" relative-to="jboss.server.config.dir"
         ↪  keystore-password="password" alias="self"
         ↪  key-password="password"/>
    </ssl>
    ...
</security-realms>
...
```

Zdrojový kód [src: 3]: Konfigurace protokolu HTTPS v souboru standalone.xml

Připojení k databázi

```
...
<datasource jta="true" jndi-name="java:jboss/datasources/wifipointdb"
 ↪  pool-name="wifipointdbPU" enabled="true" use-ccm="false">
    <connection-url>
        jdbc:mysql://127.0.0.1:3306/wifipointdb?
            useUnicode=true&amp;characterEncoding=UTF-8amp;
    </connection-url>
    <driver>mysql</driver>
    ...
    <security>
        <user-name>root</user-name>
        <password>heslo123</password>
    </security>
    ...
</datasource>
<drivers>
...
    <driver name="mysql" module="com.mysql">
        <driver-class>com.mysql.jdbc.Driver</driver-class>
    </driver>
...
</drivers>
...
```

Zdrojový kód [src: 4]: Konfigurace datového zdroje v souboru standalone.xml

Konfigurace síťového rozhraní

```
...
<interfaces>
    ...
    <interface name="public">
        <inet-address value="${jboss.bind.address:0.0.0.0}"/>
    </interface>
    ...
</interfaces>
<socket-binding-group name="standard-sockets" default-interface="public"
 ↪  port-offset="${jboss.socket.binding.port-offset:0}">
...
    <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
    <socket-binding name="http" port="${jboss.http.port:8080}"/>
    <socket-binding name="https" port="${jboss.https.port:8443}"/>
...
</socket-binding-group>
```

Zdrojový kód [src: 5]: Konfigurace síťového rozhraní v souboru standalone.xml

Uvedený příklad [src: 5] konfiguračního souboru nastavuje server na to, aby

přijímal požadavky ze všech příchozích spojení a jako server je tedy dostupný na přiřazené IP adrese, která komunikuje s okolním světem. U aplikačních serverů, které podporují Java webové aplikace, je zvykem, že HTTP port je nastaven jako `8080` a HTTPS port jako `8443`. Toto nastavení není z hlediska uživatele webové aplikace příjemné, nicméně většinou tomuto aplikačnímu serveru předchází klasický web server typu Apache či Nginx, který přesměrovává požadavky na aplikační server. V případě potřeby můžeme nastavit oba zmíněné porty na standardní.
Nyní lze spustit aplikační server běžným způsobem jako systémovou službu příkazem `service wildfly start`. Po spuštění je vytvořen aplikační „log“, který je umístěn v `/var/log/wildfly/server.log`. Do logu se zapisují všechny konzolové výstupy serveru, stejně jako u jiných serverových aplikací.

3.3 Konfigurace autentizačního serveru

Konfigurace freeradius serveru je složena z několika různých souborů. Všechny jsou umístěny standardně v `/etc/freeradius`.

```
# ---- soubor radius.conf
listen {
        type = auth
        ipaddr = 5.175.194.155 # nastavení adresy serveru
        port = 0 # pokud je 0, jsou použity výchozí porty freeradius serveru
}
...
modules {
        ...
        \$INCLUDE sql.conf # konfigurace pro připojení k databázovému
        ↪  serveru
        ...
}
# ---- soubor client.conf
# všechny příchozí požadavky z různých adres budou akceptovány
client 0.0.0.0/0{
        secret                  = testing123
        shortname       = wifipoint_routers
}
# ---- soubor sql.conf
sql {
        database = "mysql" # nastavení typu databáze
        driver = "rlm_sql_${database}"
        server = "localhost"
        login = "root"
        password = "heslo1234"
        radius_db = "wifipointdb"
        $INCLUDE sql/${database}/dialup.conf
}
```

3.4 Instalace a Konfigurace OpenWRT na bezdrátový router

Výchozí nastavení bezdrátového routeru TP-Link TL-WR1043ND je originální tovární firmware, který zdaleka nesplňuje požadavky pro naše řešení. V první fázi musíme přehrát originální firmware za vhodný OpenWRT firmware. Na webových stránkách[12] stáhneme z odkazu binární soubor určený pro tento router a přes standardní webové rozhraní původní konfigurace routeru provedeme instalaci. Instalace se provádí v sekci „Aktualizace softwaru“, kam nahrajeme stažený soubor a spustíme aktualizaci. Aktualizace provede kompletní přehrání firmwaru po a restartu již máme na routeru OpenWRT.
Nově aktualizovaný router má následující výchozí kofiguraci:

```
1 IP konfigurace 192.168.1.1/24
2 wifi - rozhraní wlan0, deaktivováno
3 instalovaný software:
4 dropbear - implementace SSH pro vestavná zařízení - dokud není nastaveno
  ↪ heslo pro uživatele root je deaktivován
5 telnet
6 LuCI - webové rozhraní pro OpenWRT
```

Pro připojení přes SSH musíme natavit „root“ heslo přes Telnet klienta.

```
1 telnet 192.168.1.1
2 root@openwrt:~$ passwd
3 Changing password for root
4 New password: heslo123
5 Retype password: heslo123
6 Password for root changed by heslo123
```

Po několika neúspěšných konfigurací jsem zjistil, že v současných repozitářích balíčkovacího systému OpenWRT je stará implementace CoovaChilli captive poratlu. Nicméně lze zkompilovat nejnovější implementaci pomocí vývojářské nástroje OpenWrt Buildroot. Jeho konfigurace se provádí lokálním zkompilováním zdrojových kódů.

Nástroj slouží především pro kompilaci OpenWRT na konkrétní wifi router. Já jsem ho použil pouze pro zkompilování nejnovější verze CoovaChilli. Po kompilaci získáme instalační soubor, který nahrajeme přes SCP[27] na router a nainstalujeme ho.

[27] Secure copy je nástroj sloužící ke kopírování souborů přes SSH

```
# kopírování instalačního souboru na router
$ scp coovacholli.ipk root@192.168.1.1:/tmp/
# instalace CoovaChilli na router
$ opkg install /tmp/coovachilli.ipk
```

Konfigurační soubory a CoovaChilli jsou v `/etc/chilli`. Instalace bohužel nevytvoří spouštěcí skript, proto je ho nutné vytvořit.

```
# aktivace wifi rozhraní
 uci set wireless.@wifi-device[0].disabled=0; uci commit wireless; wifi
#-------------------------
# spuštěcí skript /etc/init.d/chilli
#!/bin/sh /etc/rc.common
. /etc/chilli/functions
start() {
        /sbin/modprobe tun > /dev/null 2>&1
        echo 1 > /proc/sys/net/ipv4/ip_forward
        writeconfig
        radiusconfig
        iptables -F POSTROUTING -t nat
        iptables -I POSTROUTING -t nat -o $HS_WANIF -j MASQUERADE
            #chilli --fg -d
            chilli
}
stop() {
        killall chilli
}
#-------------------------
cd /etc/chilli
# výchozí konfigurační soubor defaults zkopírujeme jako config, který budeme
 ↪  konfigurovat
cp defaults config
#-------------------------
# ostatní konfigurační soubory
config
defaults
#inicializační skript, který obsahuje fuknce potřebné k načítání proměnných
 ↪  konfigurace
functions
newmulti.sh
up.sh
#automaticky vygerované soubory inicializačními skripty
down.sh
hs.conf
local.conf
main.conf
#adresář s webovým rozhraním CoovaChilli
www/
```

Konfigurace ChoovaChilli

```
1  ...
2  HS_WANIF=eth0            # rozhraní připojené k internetu
3  HS_LANIF=wlan0                  # bezdrátové rozhraní
4  HS_NETWORK=10.1.0.0           # Adresa sítě
5  HS_NETMASK=255.255.255.0   # Maska sítě
6  HS_UAMLISTEN=10.1.0.1          # Výchozí adresa routeru
7  HS_UAMPORT=3990             # Port pro
8  HS_UAMUIPORT=4990           # Port, kam budou přesměrovávány webového
   ↪  rozhraní
9  ...
10 HS_RADIUS=5.175.194.155 # IP adresa radius serveru
11 #Dostupné webové stránky a rozsahy adresa, kam může neautentizovaný uživatel
12 HS_UAMALLOW=www.zive.cz,play.google.com,10.1.0.0/24,5.175.194.155
13 HS_RADSECRET=testing123    # tajný klíč pro šifrování
14 HS_UAMDOMAINS=".zive.cz,play.google.com" # dostupné domény neautentizovaných
   ↪  uživatelů
15
16 HS_UAMFORMAT=http://\$HS_UAMLISTEN:\$HS_UAMUIPORT/www/status.chi
17 # výchozí webová stránka, kam se přesměrovávají neautentizované požadavky
18 HS_UAMHOMEPAGE=http://\$HS_UAMLISTEN:\$HS_UAMPORT/www/coova.html
19 ...
20 HS_WWWDIR=/etc/chilli/www
21 HS_WWWBIN=/etc/chilli/wwwsh
22 ...
```

Zdrojový kód [src: 6]: Konfigurace souboru config

Kapitola 4

Implementace

4.1 Nástroje pro vývoj webové a mobilní aplikace

Před samotným vývojem aplikací je důležité zvolit nástroje, které budeme využívat. Moje volba bude hlavně na základě předchozích zkušeností s danými nástroji, abych předešel případným problémům, které by mohli vzniknout neznalostí nástrojů.

Android Studio
Pro vývoj mobilní aplikace jsem použil vývojový nástroj Android Studio, což je upravená verze IntelliJ IDEA Community Edition, který je zdarma ke stažení a jeho zdrojový kód je uvolněn volně k úpravám. Android Studio je považováno jako oficiální Android IDE[28]. Prostředí obsahuje všechny potřebné komponenty pohromadě a bez nutnosti instalace dalších nástrojů pro vývoj.

Netbeans IDE 8.0
Vývojové prostředí Netbeans je jedním z nejpoužívanějších IDE nejen pro vývoj aplikací v jazyce Java, ale nabízí i další moduly, které lze nainstalovat pro vývoj v jiných jazycích. S tímto vývojovým prostředím mám již zkušenosti a jsem zvyklí ho používat.

Verzovací systém
Verzovací systém je při vývoji jakékoliv aplikace dnes již standardem. Díky němu máme náš kód chráněn proti případnému kolapsu našeho zařízení, na kterém vyvíjíme a následné ztrátě dat. Verzovací systém slouží k týmové práci na celém projektu, kde každý z týmu může pracovat v jednu chvíli na stejné aplikaci. Další výhodou verzovacího systému je možnost vracet se zpět ve svém kódu a v případě fatální chyby vývojové chyby jsme schopni se vrátit

[28] IDE je zkratka pro vývojové prostředí.

do určitého místa a pokračovat znovu. Já jsem si jako verzovací systém zvolil open-source software git, který je v dnešní době nejpopulárnější.

4.2 Implementace aplikací pro aplikační sever

Pro správu a navržení struktury aplikací pro aplikační server jsem použil již zmíněný nástroj Maven, ve kterém jsem si vytvořil takzvaný POM projekt pojmenovaný „wp_server“. POM projekt bude hlavním projektem a bude sdružovat všechny ostatní komponenty pomocí závislostí na nich. Díky POM projektu se při kompilaci celého projektu sestaví všechny jednotlivé závislé komponenty pro následné nasazení na aplikační server. Do POM projektu jsem si tedy vytvořil další dva závislé projekty „RESTWS“ a „WEBApp“. Obě dvě výsledné aplikace budou nasazeny na jeden aplikační server do jednoho aplikačního kontejneru, kde každá z nich bude mít svůj oddělený kontext. K jednotlivým aplikacím budeme moci přistupovat různými URL adresami na serveru.

Před samotným programováním je vhodné si připravit strukturu jednotlivých aplikací. Jelikož obě aplikace poběží na aplikačních serverech, vyžaduje to konfiguraci souboru `web.xml`, který je načítán při spouštění na serveru a nastavuje aplikaci běhové prostředí. Názorný příklad je naznačen v ukázce kódu [src: 7].

```
1  <web-app>
2  ...
3      <security-constraint>
4      ...
5          <web-resource-collection>
6              ...
7              <url-pattern>/restws/resources/*</url-pattern>
8          </web-resource-collection>
9          <auth-constraint>
10             <role-name>intranet</role-name>
11         </auth-constraint>
12     ...
13     </security-constraint>
14     <login-config>
15         <auth-method>BASIC</auth-method>
16         <realm-name>ApplicationRealm</realm-name>
17     </login-config>
18     <security-role>
19         <role-name>wifiappservices</role-name>
20     </security-role>
21     ...
22 </web-app>
```

Zdrojový kód [src: 7]: Konfigurace souboru web.xml

Podle výše uvedeného příkladu konfigurace se k tomuto URL kontextu aplikace budou mít přístup pouze uživatelé ve skupině „wifiappservices“ a aplikace bude jako autentizační parametr „realm“[13] používat hodnotu „ApplicationRealm“, který je již definován na serveru pro všechny aplikace běžící na serveru. Typ autentizace je zde pouze BASIC, nicméně celý kontext aplikace běží pouze pod šifrovaným připojením.

Obě aplikace budou přistupovat k databázi přes takzvanou „perzistentní jednotku“[14][29]. Tuto PU implementuje framework JPA. Perzistentní jednotka je definována v souboru `persistence.xml` způsobem, který je vidět v příkladu [src: 8].

```
1  ...
2  <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" ...>
3  ...
4    <persistence-unit name="wifipointdbPU" transaction-type="JTA">
5      <jta-data-source>java:/jboss/datasources/wifipointdb</jta-data-source>
6      ...
7    </persistence-unit>
8  ....
9  </persistence>
```

Zdrojový kód [src: 8]: Konfigurace perzistentní jednotky v souboru persistence.xml

Z ukázky je vidět, že perzistentní jednotka je definována názvem „wifipointdbPU“ a datovým zdrojem. Datový zdroj musí být poskytován aplikačním serverem, jehož konfiguraci jsme provedli v ukázce konfigurace [src: 4]. Takto definovanou perzistentní jednotku lze načíst objektu `EntityManager`[30]. Pomocí tohoto objektu jsme schopni komunikovat s databází přes metody `find`, `merge`, `persist` a `remove`. Pokud chceme vytvářet složitější dotazy nad databází jsou k dispozici dva možné styly přístupu:

- **Nativní dotazování** – SQL dotazy odpovídající dialektu databáze, na kterou je napojen server. Toto řešení komplikuje přechod na jiný typ databáze. Zároveň zde při špatném ošetření uživatelských vstupů hrozí riziko „SQL injection“[31] útoku.
- **JPQL dotazování** – Java Persistence Query Language je jazyk, který implementuje framework JPA a hlavní výhodou je dotazování se do databáze již na úrovni entitních objektů. Dotazy jsou obdobné klasickým SQL, avšak jsou nezávislé na dialekt databáze.

[29] Perzistentní jednotka = Persistence Unit, dále již jen jako PU

[30] EntityManager je třída frameworku JPA

[31] SQL injection je útok, kdy útočník vloží místo například námi očekávaného jména vhodný SQL dotaz, který databáze vykoná a na výstup vrátí data z dotazu.

4.2.1 Aplikace RESTWS

Tato aplikace bude implementovat RESTful webové služby a její kontextová URL adresa bude definovaná ve tvaru:
`https://<adresa_web_serveru>:<https_port>/restws/resources/*`.
Po zadaní kontextové adresy pro tuto aplikaci získáme přístup k jednotlivým definovaným zdrojům, ze kterých lze získávat data. Aplikace **restws** poskytuje pro všechny entitní třídy metody pro vytvoření, editaci a vyhledávání záznamů v databázi.

Popis webových služeb

Podporované metody HTTP protokolu jsou:

- **GET** – vrátí data na zadaném zdroji. Akceptovatelný formát přijatých dat může typu XML a nebo JSON.
- **POST** – slouží k odeslání nových dat na server. Aplikace po jeho přijetí vytváří nový objekt. Přijatelné formáty pro serverovou aplikaci jsou XML a JSON.
- **PUT** – editování záznamu. Tento požadavek vyžaduje další parametr a to „id“ entity, kterou budeme chtít editovat. Přijatelné formáty pro serverovou aplikaci jsou XML a JSON.
- **DELETE** – mazání záznamu. Aplikace po přijetí toho požadavku vyhledá podle zadaného parametru „id“ požadovaný objekt a smaže ho z databáze.

Všechny webové zdroje této aplikace podporují CRUD (Create, Read, Update, Delete) operace. Pro některé z webových zdrojů jsem implementoval vlastní rozšířené operace, které jsem následně využil při napojení na mobilní aplikaci.

4.2.2 Aplikace WEBApp

Webová administrační aplikaci je složena ze tří implementačních částí. Jedná se tedy o třívrstvou architekturu.

- **Datová vrstva** je implementována pomocí frameworku Hibernate JPA a jejím účelem je poskytovat data logické vrstvě.
- **Logická vrstva** implementuje funkce komunikující s datovou vrstvou a celou logiku aplikace.
- **Prezentační vrstva** je implementována pomocí JSF 2 frameworku. Zobrazuje výstupy nižších vrstev aplikace uživateli.

Díky výše zmíněnému návrhovému vzoru je zdrojový kód aplikace rozdělen do dvou částí z nichž každá má vlastní význam a případné konfigurační soubory načítané při nasazování na aplikační server. Následné rozdělení v různých vývojových prostředích je různě pojmenované. Já použiji pojmenování nástrojem Netbeans IDE.

Web Pages
Tato část obsahuje všechny částí kódu, které se týkají prezentační vrstvy. Struktura musí obsahovat adresář s přesným názvem `WEB-INF`, který obsahuje konfigurační soubory definující chování projektu po nasazení. Ostatní složky a soubory jsou již samotnou implementací grafického rozhraní.

Source Packages
Zdrojové balíčky jsou strukturovány běžným standardním způsobem známým pro Java aplikace. Java balíčky jsou rozděleny do třech částí.

- **balíček integration** obsahuje implementaci komunikace s databází
- **balíček view** sdružuje implementaci JavaBeans, které slouží k zobrazování dat do grafického rozhraní
- **balíček business** implementuje návrhový vzor „fasáda“, díky kterému můžeme v jednotlivých částech uživatelského rozhraní zobrazovat datové výstupy skrze zmíněné JavaBeans

4.2.3 Nasazení aplikací na aplikační server

Výsledná implementovaná aplikace je sestavena a zkompilována do `war` souboru, což je archiv webové aplikace určený pro nasazení na aplikačním serveru. Na JBoss WildFly aplikační server je velmi jednoduché nasadit takovouto aplikace pomocí webového rozhraní pro správu a částečnou konfiguraci serveru.

4.3 Vývoj mobilní aplikace pro platformu Android

Hlavním požadavkem na mobilní aplikaci je implementovat její funkčnost tak, aby po zjištění přítomnosti například wifi stanice „WifiPoint“ se automaticky připojila k sítí a přihlásila uživatele aplikace. V implementaci jsem se tedy především zaměřil na tuto funkčnost a zbylé funkcionality, které by v produkční aplikaci měly být, jsem vynechal.

4.3.1 Struktura

Stejně jako ve webových aplikacích i při vývoji Android aplikace je vhodné použít nástroj pro správu a kompilování projektu. V Android Studiu je použit pro automatickou kompilaci nástroj Gradle[15]. Gradle, stejně jako Maven,

umožňuje přidávat do projektu další knihovny rozšiřující funkčnost aplikace. Každá android aplikace se obsahuje ze tři části:

- **Manifest** – soubor definující používaná oprávnění pro přístup k uživatelským datům a ke komponentám zařízení. Dále manifest definuje aplikační kontext celé aplikace a jeho hlavní komponenty jako jsou „aktivity“, „service“ a „receiver“.
- **Java balíčky** – stejně jako u jiných Java aplikací struktura odpovídá konvencím Java.
- **Res – zdroje UI** – struktura XML souborů pro definici UI

4.3.2 Použité knihovny třetích stran

Pro implementaci některých částí aplikace jsou využil již vytvořených a otestovaných knihoven. Dle mého názoru používání knihoven při implementaci Android projektů je velmi vhodné, jelikož urychlují práci a většina je dostatečně otestována. V této práci jsem použil hned několik takových to knihoven, které si já osobně myslím, že by měl znát každý vývojář pro Android platformy.

Sprinkles

Sprinkles[16] je velmi užitečná knihovna velmi usnadňující práci s interní databází Androidu. Android nativně umožňuje ukládání uživatelských dat do integrované SQLite databáze. Nicméně její konfigurace a implementace je poměrně zdlouhavá. Knihovna sprinkles elegantně řeší tento pomocí anotací tříd, stejně jako například Hibernate a další robustní frameworky. Jednoduchým anotováním a děděním běžné entitní třídy získáme metody, které nám umožní objekt jednoduše vytvořit, upravit a nebo smazat z databáze.

Retrofit

Knihovna Retrofit[17] implementuje REST klient rozhraní. Retrofit umožňuje mapovat data získaná z webových služeb přímo na entitní objekty, jsou-li vhodně navrženy.

OttoBus

Ottobus[18] knihovna implementuje funkčnost rozesílání událostí mezi jednotlivými komponentami aplikace. Jednoduchým vytvořením „softwarové sběrnice“ získáme možnost registrovat různé komponenty k této sběrnici, což umožňuje jednoduše implementovat komunikaci mezi komponentami a definovat interakce na dané události.

4.3.3 Přístup k REST API v platformě Android

Komunikaci s webovými službami zajistíme implementací Retrofit knihovny. Nejprve je ale nutné implementovat entitní třídy odpovídající objektům, které získáváme z webových služeb. V podstatě budeme implementovat datový model webových služeb. Příklad [src: 9] odpovědi získané z webových služeb:

```
"/////// HTTP - GET ////////

http://<srv>:8443/restws/resources/wifistation/find/WifiPoint
Accept: application/json
Authorization:Basic dGVzdDp0ZXN0dGVzdA==

////////////////////////////////////"

{
  "location" : "blok4,339",
  "address" : "Vanickova 7",
  "id" : 2,
  "wifiSsid" : "WifiPoint",
  "latitude" : 50.08,
  "longtitude" : 14.3933
}
```

Zdrojový kód [src: 9]: Příklad odpovědi webových služeb

Důležité při implementaci modelu aplikace je zachování přesných názvů a datových typů proměnných výsledné třídy. Pokud tak neučiníme jsou dvě možné chyby:

- proměnou objektu JSON nelze namapovat k žádné proměnné z entitní třídy – žádná hodnota
- proměnnou se podařilo nalézt, avšak nejsou stejné typy proměnných – skončí chybou mapování

Výsledná správná implementace objektu může vypadat jako v následujícím příkladě[src: 10], kde je zároveň ukázáná implementace knihovny Sprinkles.

```
//importované knihovny

@Table("WifiStation")
public class WifiStation extends Model implements Serializable {
    @PrimaryKey
    @Column("id")
    private long id;
    @Column("location")
    private String location;
    @Column("address")
    private String address;
    @Column("wifiSsid")
    private String wifiSsid;
    @Column("latitude")
    private float latitude;
    @Column("longtitude")
    private float longtitude;
    // kontruktror
    // + getters, setters
}
```

Zdrojový kód [src: 10]: Příklad implementace entitní třídy

Jednotlivé webové zdroje implementuje veřejné rozhraní „WPApiInterface“. Zde definujeme metody, které implementují RESTful komunikaci.

```
...
public interface WPApiInterface {
    // Vytvoření nového uživatele
    @POST("/user")
    void createNewUser(@Body User user, Callback<User> cb);
    // Registrace nového zařízení k uživateli
    @PUT("/user/{id}/register/{token}/{type}")
    @Headers("Content-type: application/json")
    void registerDevice(@Path("id") long id,@Path("token") String
     ↪  token,@Path("type") String type,Callback<Object> cb);
    // Vyhledání wifi stanice podle wifissid způsob 1
    @GET("/wifistation/find/{wifissid}")
    void findWifiStationBySsid(@Path("wifissid") String ssid,
     ↪  Callback<WifiStation> cb);
    // Vyhledání wifi stanice podle wifissid způsob 2
    @GET("/wifistation/find/{wifissid}")
    WifiStation findWifiStationBySsid(@Path("wifissid") String ssid);
   ...
}
```

Zdrojový kód [src: 11]: Příklad implementace klientského rozhraní webových služeb

Příklad [src: 11] nastiňuje implementaci rozhraní jaké metody budou využívány pro komunikaci s webovými službami. Jsou zde dva možné přístupy k definování rozhraní z pohledu návratových hodnot.

Metoda vrací void
Pokud je návratovým typem Void, jedním z parametrů metody musí být objekt „Callback“. Celá metoda se po zavolání vyvolává na pozadí asynchronně, ne na vlákně zpracovávající uživatelské rozhraní (UI thread). „Callback“ potom funguje jako objekt, který je zavolán z pozadí aplikace a lze tak provést změny na UI vlákně. Například aktualizace seznamu nabídek. Tato metoda může být bez problémů volána na UI vlákně.

Metoda vrací konkrétní objekt či kolekci objektů V tomto případě není definovaný „Callback“, což znamená, že metoda by měla být volána pouze v asynchronním vlákně. Pokud bude vyvolána na hlavním UI vlákně, systém vyvolá výjimku „NetworkOnMainThreadException“ a pravděpodobně způsobí pád celé aplikace.

4.3.4 Využití notifikací pro Android

Rozesílání notifikací do mobilních telefonů je v dnešní době standard, který je naprosto běžný. Google pro Android implementuje platformu nazývanou „Google Cloud Messaging“. Zkráceně GCM je služba, která se stará o rozesílání notifikací do mobilních telefonů. V jednoduchosti to celé funguje tak, že každé zařízení se na začátku spuštění aplikace zaregistruje na GCM server a zároveň tak na nějaký další server, který se stará o správu a rozesílání notifikací (v našem případě řídícím serverem je implementace *RESTws* a *WEBapp*). Pokud chceme odeslat notifikaci do mobilního telefonu postupujeme následovně:

1. Vytvoříme obsah rosezílané notifikace.
2. Vybereme registrovaná zařízení, kterým chceme poslat notifikaci.
3. Odešleme všechny zařízení a notifikaci GCM službě, která rozešle notifikace do mobilních zařízení.

Díky mnoha nově vzniklých online služeb pro rozesílání notifikací bychom mohli vynechat implementaci v naší webové aplikaci. Já jsem ve webové aplikaci implementaci notifikací nevynechal, jelikož mě tato problematika zajímala a chtěl jsem si jí vyzkoušet.

Přijímání notifikací v mobilní aplikaci vyžaduje implementaci takzvané „IntentService“, což je komponenta „spící“ na pozadí aplikace (i v případě, že úplně zavřeme aplikaci a není viditelná ani v minimalizovaných aplikacích). Tato komponenta se „probudí“ pouze v případě, když ji zavolá komponenta „receiver“. „Receiver“ musí být zadefinován v Manifestu Android aplikace a musí specifikovat událost, kterou přijímá. Víz ukázka [src: 12].

```
...
<receiver
    android:name=".services.NetworkBroadcastReceiver"
    android:permission="android.permission.ACCESS_WIFI_STATE"
    android:label="NetworkConnection">
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    </intent-filter>
</receiver>
<service android:name=".services.NetworkNotificationService" />
....
```

Zdrojový kód [src: 12]: Příklad definice receiveru v Manifestu aplikace

4.3.5 Implementace logiky automatického přihlašování

Tato část implementace je dle mého názoru alfa omegou této bakalářské práce. Hlavní myšlenkou této bakalářské práce je vytvoření Android aplikace, která v případě dostupnosti podporované wifi stanice automaticky přihlásí do sítě a uživatel bude moci rovnou používat neomezené připojení k internetu.

4.3.5.1 Komunikace s routerem pomocí JSON API

Velkou výhodou captive portalu CoovaChilli je implementace velmi jednoduchého JSON API, které poskytuje pouze 3 webové zdroje, které jsem popsal v sekci 2.4. Stejně jako pro webové služby namapujeme a vytvoříme entitní třídu „CPRequest“. „CPRequest“ je objekt, který vrací router v různých stavech autentizace. Tento objekt obsahuje dvě proměnné, které jsou pro nás aktuálně nejdůležitejší. Proměnná `clientState` definuje zda-li je klient autentizovaný (hodnota 1) či není (hodnota 0). Druhá proměnná `challenge` obsahuje unikátní retězec, vygenerovaný wifi routerem v závislosti na MAC adrese.

4.3.5.2 Implementace logiky přihlášení

Proces přihlášení je rozložen do několika samostatných na sobě závisejících kroků.

1. Zjištění, zda-li mám internetové připojení typu wifi a zda-li jsem příhlášen k podporované wifi síti.
2. Zjištění stavu autentizace (přihlášen/nepříhlášen). Odeslán pořadavek na JSON API. Zpět je vrácen objekt `CPRequest`.
3. Získání informace o uživatelském účtu z interní databáze Android aplikace, zda-li byl již uživatel registrován. Pokud ne, je nutná registrace.
4. Výpočet URL parametru `response` a následné odeslání požadavku o přihlášení na router.

5. Router zpracuje požadavek a přepošle ho RADIUS serveru, který ověří zadané údaje.

6. Zpět do Android aplikace je odeslán CPRequest objekt, ze kterého aplikace určí aktuální stav.

4.3.5.3 Princip CHAP protokolu

Hlavní myšlenka CHAP protokolu je autentizace uživatele na základě sdíleného klíče, který není nikdy posílán přes síť v „prostém“ tvaru. Na začátku každé komunikace ChoovaChilli router vygeneruje unikátní `challenge`. Challenge slouží k výpočtu parametru `response`. Při standardním použití se `response` vypočítá po zadání hesla a jména do webového rozhraní routeru. Následně se vypočítá jednocestný hash, který je složen z `challenge` a sdíleného klíče, což je v našem případě heslo uživatele. Následně se hash, uživatelské jméno, původní `challenge` a další parametry odesloušlou RADIUS serveru. Ten podle zadaných parametrů a svého sdíleného klíče vypočítá také hash a porovná ho s přijatým hashem. Pokud se shoduj, RADIUS server povolí přístup do sítě, pokud se neshodují spojení je ukončeno a CoovaChilli generuje nové spojení s novou `challenge`.

Při implementaci se objevil problém a to jak vypočítat na Android zařízení parametr `response`, aniž bychom museli posílat heslo v „plaintext“ formátu. Napadla mě dvě možná řešení tohoto problému.

1. Implementace jednoduché služby, která bude mít jako URL parametry `heslo` a `challenge`. Ze zadaných parametrů vypočítá parametr `response`, který by vrátila jako odpověď. Samozřejmě v tomto případě je nutné zabezpečit komunikaci HTTPS protokolem.

2. Výpočet parametru `response` implementovat přímo v mobilní aplikaci.

Mně připadá druhá varianta velmi elegantní a bezpečnější. Všechny parametry pro výpočet máme k dispozici již při prvním požadavku na wifi router. Algoritmus výpočtu nastiňuje část implementace [src: 13]:

```
1 String calculateResponse(String ident, String password, String challenge) {
2     String hex_password = str2hex(password);
3     String concatenate = ident + hex_password + challenge;
4     concatenate = concatenate.toLowerCase();
5     return  md5Hex(concatenate);
6 }
```

Zdrojový kód [src: 13]: Ukázka výpočtu parametru RESPONSE

4.3.5.4 Automatické přihlašování

K automatizaci přihlašování do wifi routeru jsem využil komponenty `IntentService` a `BroadcastReceiver`. Definice těchto dvou komponent je zobrazena v ukázce Manifestu [src: 12].

`NetworkBroadcastReceiver` poslouchá změny konektivity a jakmile detekuje přepnutí připojení na wifi, následně pomocí „intentu“[32] probudí `NetworkNotificationService` komponentu.

`NetworkNotificationService` implementuje výše uvedený postup přihlašovací logiky. Pokud celý proces proběhne úspěšně a uživatel je přihlášen do wifi sítě, aplikace vyvolá notifikaci informující o proběhlé akci.

[32]Intent je jednoduchý objekt, do kterého lze nastavit hodnoty, které budou předány jinému procesu

Kapitola 5

Testování

Součástí všech projektů by mělo být testovaní. Cílem testování je včasné odhalení chyb, které by mohli v budoucnu způsobit nepříjemnosti jak uživatelům, tak i vývojářům. Testování by mělo zahrnovat automatizované testy prováděné testovacími nástroji a zároveň ruční testování, které častokrát odhalí chyby v návrhu uživatelského rozhraní. Ideálním přístupem k testování je vytváření průběžných částečných testů, které odhalí chyby již na počátku.

5.1 Automatizované testování

Velkou výhodou oproti ručnímu testování je velká časová úspora a především omezení lidského faktoru během automatizovaného testování. Minimalizováním lidského faktoru se zároveň minimalizuje chybovost v prováděných testech vzniklá lidskou chybou.

Automatizované testování je v vhodné pro často se opakující soubory testů. Před nasazením automatizovaných testovacích nástrojů je dobré zvážit, zdali jsou tyto robustní testy vhodné. Jestliže již předem víme, že náš projekt bude mít pouze jednu verzi a nepředpokládáme budoucí vývoj, automatizované testy jsou zbytečně časově náročné.

Nevýhoda automatizovaného testování je velká počáteční časová investice, která někdy může být stejně velká jako vývoj celé aplikace. Bohužel se o tyto testy musíme „starat“ a udržovat je aktuální. Jakákoliv změna v projektu musí být zanesena i do testů.

5.1.1 Automatizované testovací nástroje

Existuje celá řada testovacích nástrojů pro různé programovací jazyky. Tato práce se zabývala implementací Java aplikací a mobilní Android aplikace, takže uvedu zástupce pro obou z nich.

5.1.1.1 Unit testy

Tyto testy se zaměřují na testování jednotlivých částí aplikace či samostatně testovatelných částí kódu. Z pohledu objektově orientovaného programování se většinou jedná o testování jedné třídy.

Nejznámějším testovacím nástrojem na principu unit testů pro Java aplikace je JUnit[19] framework. Jedná se o jednoduchý framework, který lze snadno přidat jako Maven závislost.

Android již ve svém základu nabízí možnost využití testování po mocí unit testů. Jsou zde implementovány třídy zvané „Test Case“, které implementují metody vhodné pro testování typický komponent platformy Android.

5.1.1.2 Robotium

Velmi oblíbený mezi Android vývojáři je robotium[20] open-source framework. Jedná se o automatizovaný nástroj pro testování Android aplikací. Robotium umožňuje jednoduše napsat komplexní a robustní automatické testy pro Android aplikaci.

5.2 Ruční testování

Ruční testování produktu spočívá v manuálním zkoušení funkčnosti a mělo by simulovat používání produktu koncovým uživatelem. Ruční testování v některých případech může odhalit chyby, které automatizované testy neodhalí. Například nelogické chování uživatele a další.

Při vývoji mobilních aplikací je ruční testování důležité již při návrhu uživatelského rozhraní, jelikož můžeme už na začátku odhalit možné problémy například s nevhodným ovládáním aplikace.

Pro ruční testování je dobré vytvořit seznam úkonů, které mají být vykonány a následně ohodnoceny, zda-li splňují požadovanou funkčnost, která byla navržena.

5.3 Testování této práce

V této bakalářské práci jsem pro testování použil pouze ruční metodu testování. Testoval jsem pouze celkovou funkčnost z různých hledisek.

Nejprve jako administrátor jsem testoval funkčnost přidávání, editace a mazání položek ve webové administrátorské aplikaci. Následně jsem testoval mobilní aplikaci a její automatické přihlašování do wifi sítě.

Závěr

Zhodnocení práce a výsledného řešení

Cílem práce bylo vytvoření návrhu řešení pro automatickou autentizaci do wifi sítě pomocí mobilní aplikace Android. I přes různé problémy, které jsem měl během vývoje si myslím, že se mi podařilo naplnit zadání práce a výsledné řešení splňuje všechny klíčové funkce.

Aktuální verze celého systému má k produkčnímu řešení velmi daleko, nicméně si myslím, že současná řešení podobného typu nezahrnují do své funkcionality mobilní aplikaci jako možný autentizační klíč. Nevím, zda-li je to správný směr, kterým měly captive portal ubírat, ale každopádně tato práce může být brána jako jiný pohled na aktuální problematiku.

Plány rozvoje služby

Moje velmi futuristická představa konečného řešení by bylo implementovat jednotné řešení, které by nabízelo upravený a již nakonfigurovaný firmware OpenWRT, který by si mohl zákazník nechat nasadit na svůj router.

Následně bych rád vytvořil mobilní SDK implementující funkčnost automatické autentizace, které by mohlo být do implementováno do stávající či nové aplikace zákazníka.

Zákazník by tímto způsobem mohl integrovat naše řešení do svého a rozšířil tím svoje možnosti působení díky možnosti rozesílání notifikací svým uživatelům aplikace.

Literatura

[1] Hole, K. J.; Dyrnes, E.; Thorsheim, P.: Securing wi-fi networks. *Computer*, ročník 38, č. 7, 2005: s. 28–34.

[2] Droms, R.: Dynamic host configuration protocol. 1997.

[3] Richardson, L.; Ruby, S.: *RESTful web services*. "O'Reilly Media, Inc.", 2008.

[4] Gargoyle: Wiki OpenWRT [online]. 2014, [cit. 2015-03-05]. Dostupné z: `http://www.gargoyle-router.com/wiki/doku.php`

[5] DD-WRT: What is DD-WRT? [online]. 2014, [cit. 2015-03-04]. Dostupné z: `http://www.dd-wrt.com/wiki/index.php/What_is_DD-WRT`

[6] OpenWRT Community: Gargoyle Documentation [online]. 2013, [cit. 2015-03-05]. Dostupné z: `http://wiki.openwrt.org/about/start`

[7] *Linková vrstva - metody přístupu [online].*

[8] OpenWRT Community: Wiki OpenWRT - wireless overview [online]. 2013, [cit. 2015-03-05]. Dostupné z: `http://wiki.openwrt.org/doc/howto/wireless.overview`

[9] Rigney, C.; Willens, S.; Rubens, A.; aj.: Remote Authentication Dial In User Service (RADIUS). RFC 2865 (Draft Standard), Červen 2000, updated by RFCs 2868, 3575, 5080, 6929. Dostupné z: `http://www.ietf.org/rfc/rfc2865.txt`

[10] Fielding, R.; Gettys, J.; Mogul, J.; aj.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), Červen 1999, obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585. Dostupné z: `http://www.ietf.org/rfc/rfc2616.txt`

[11] Almsaeed, A.: AdminLTE Dashboard & Control Panel Template [online]. Březen 2015, [cit. 2015-20-04]. Dostupné z: https://almsaeedstudio.com

[12] community, O.: TP-Link TL-WR1043ND [online]. 2015, online wiki manuálnová stránka konfigurace OpenWRT. Dostupné z: http://wiki.openwrt.org/toh/tp-link/tl-wr1043nd

[13] Franks, J.; Hallam-Baker, P.; Hostetler, J.; aj.: HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), Červen 1999, updated by RFC 7235. Dostupné z: http://www.ietf.org/rfc/rfc2617.txt

[14] Troníček, Z.: BI-EJA - Java Persistence API [online]. Duben 2012, [cit. 2015-22-04]. Dostupné z: https://edux.fit.cvut.cz/archive/B122/BI-EJA/_media/lectures/eja5.pdf

[15] GRADLE, INC.: Gradle - Build Automation Evolved. [online]. 2015, [cit. 2015-22-04]. Dostupné z: https://gradle.org/docs/current/userguide/userguide

[16] Emil Sjölander: Sprinkles [online]. 2015, [cit. 2015-09-05]. Dostupné z: https://github.com/emilsjolander/sprinkles

[17] Square, Inc.: Retrofit [online]. 2013, [cit. 2015-03-05]. Dostupné z: http://square.github.io/retrofit/

[18] Square, Inc.: Otto [online]. 2014, [cit. 2015-03-05]. Dostupné z: http://square.github.io/otto/

[19] JUnit-team: JUnit [online]. 2014, [cit. 2015-03-05]. Dostupné z: http://junit.org

[20] Robotium-Team: Robotium [online]. 2014, [cit. 2015-03-05]. Dostupné z: https://code.google.com/p/robotium/

Příloha **A**

Seznam použitých zkratek

GUI Graphical User Interface

IDE Integrated Development Environment

JSON JavaScript Object Notation

HTML HyperText Markup Language

XHTML Extensible Hypertext Markup Language

XML Extensible Markup language

JPA Java Persistence API

API Application Programming Interface

JSF Java Server Faces

REST Representational State Transfer

J2EE Java Platform, Enterprise Edition

GCM Google Cloud Messaging

IP Internet protocol

DHCP Dynamic Host Configuration Protocol

REST Representational State Transfer

WP WifiPoint

wifi Wireless Fidelity

SSH Secure Shell

Telnet Telecommunication Network

PHP Hypertext Preprocessor

WPA Wireless Protected Access

RADIUS Remote Authentication Dial In User Service

AAA Authentication, authorization and accounting

SQL Structured Query Language

ORM Object/Relational Mapping

JDBC Java database connectivity

JAX-RS Java API for RESTful Web Services

RFC Request For Comments

CHAP Challenge-Handshake Authentication Protocol

PAP Password Authentication Protocol

PPP Point-To-Point Protocol

SSID Service Set Identifier

ASCII American Standard Code for Information Interchange

CRUD Create Read Update Delete

URL Uniform Resource Locator

BASH Bourne Again Shell

IDE Integrated Development Environment

POM Project Object Model

JPU Java Persistence Unit

JPQL Java Persistence Query Language

MVC Model View Controler

WAR Web Application Archive

UI User Interface

MAC Media Access Control

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `conf` ..... konfigurační soubory
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF