FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Mobilní aplikace pro monitorování síťového připojení

Jan Kubala

PLZEŇ

2024

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Mobilní aplikace pro monitorování síťového připojení

Jan Kubala

Vedoucí práce
Ing. Ladislav Pešička

PLZEŇ 2024

© Jan Kubala, 2024.

Všechna práva vyhrazena. Žádná část tohoto dokumentu nesmí být reprodukována ani rozšiřována jakoukoli formou, elektronicky či mechanicky, fotokopírováním, nahráváním nebo jiným způsobem, nebo uložena v systému pro ukládání a vyhledávání informací bez písemného souhlasu držitelů autorských práv.

Citace v seznamu literatury:
KUBALA, Jan. *Mobilní aplikace pro monitorování síťového připojení*. Plzeň, 2024. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Ing. Ladislav Pešička.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jan KUBALA**
Osobní číslo:	**A22B0307P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Mobilní aplikace pro monitorování síťového připojení**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Prozkoumejte vybrané mobilní aplikace pro monitorování rychlosti připojení a dostupnosti síťových služeb.
2. Vyberte vhodné parametry síťového připojení, které bude mobilní aplikace monitorovat. Dále navrhněte vhodný způsob zobrazení naměřených hodnot.
3. Navrhněte mobilní aplikaci, která bude monitorovat vlastnosti síťového připojení. Součástí řešení bude i server pro ukládání naměřených hodnot a konfiguraci plánovaných měření.
4. Navržený systém realizujte, ověřte jeho funkcionalitu v místech s odlišnou úrovní mobilního signálu a navrhněte další vhodná rozšíření.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Ladislav Pešička**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **2. října 2023**
Termín odevzdání bakalářské práce: **2. května 2024**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 25. října 2023

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Plzni dne 31. dubna 2024

. . . . . .
Jan Kubala

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Abstrakt

S rostoucím významem internetu je důležité sledovat kvalitu připojení. Pro tento účel byla vyvinuta aplikace pro platformu Android sloužící pro měření a analýzu kvality internetového připojení k předdefinovaným přístupovým bodům. Aplikace umožňuje konfigurovat testovací sady s různými protokoly a nástroji a zaznamenává detailní výsledky s informacemi o protokolu a poloze. Data se synchronizují se serverem pro online prezentaci a filtrování. Řešení tak poskytuje možnost sledovat a analyzovat kvalitu připojení v různých lokalitách a při různých konfiguracích a identifikovat případné problémy. Aplikace je užitečným nástrojem pro vývojáře aplikací, síťové administrátory a běžné uživatele ke sledování a řešení problémů s připojením a nízkým výkonem internetových služeb.

Abstract

With the growing importance of the Internet, it is important to monitor the quality of the connection. For this purpose, an Android application has been developed to measure and analyse the quality of the Internet connection to predefined access points. The app allows configuring test sets with different protocols and tools and records detailed results with protocol and location information. The data is synchronized with the server for online presentation and filtering. The solution thus provides the ability to monitor and analyze connection quality in different locations and configurations and identify potential problems. The application is a useful tool for application developers, network administrators and general users to monitor and troubleshoot connection issues and poor Internet service performance.

Klíčová slova

Android • `HTTPS` • `SFTP` • `SSH` • `Ping` • měření kvality připojení • `API`

Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce Ing. Ladislavovi Pešičkovi za odborné vedení, pomoc a cenné rady při zpracování této práce.

Jan Kubala, (květen 2024)

Obsah

Úvod 1

S rychlým rozvojem internetových služeb a jeho zvětšující se důležitostí je nutné pozorovat i jeho chování. Chování síťových služeb je často stěžejním požadavkem, při kterém je vyžadována maximální efektivita. Tato efektivita se ovšem může lišit v závislosti na mnoha faktorech, jako je kvalita síťového připojení, efektivita komunikačního protokolu a podobné. Pomalé načítání webových stránek může být pro uživatele nepříjemné, nekvalitní internetové připojení může degradovat výkon videokonference a našli bychom celou sadu dalších případů, kdy kvalita internetového připojení ovlivňuje kvalitu používané služby.

Pro účely měření a pozorování měřených výsledků bude potřeba stanovit několik požadavků. Hlavním požadavkem bude poskytnutí několika druhů měření kvality připojení k předdefinovaným přístupovým bodům a následného srovnání naměřených výsledků na platformě Android.

Z vytvořených aplikací volně publikovaných na `Obchod Play` bylo zjištěno, že žádná z poskytovaných aplikací nedisponuje možností konfigurace přístupových bodů využitých pro měření.

Z tohoto důvodu bude vytvořena aplikace pro měření kvality připojení k předdefinovaným přístupovým bodům s využitím různých typů měření. Jednou z hlavních vlastností má být provádění a analýza testů síťového připojení na základě předdefinovaných sad konfigurací, které jsou testovány za pomoci různých protokolů a nástrojů. Jednotlivé testy nejenže budou poskytovat detailní informace o výsledcích měření, podle využitého protokolu, ale také budou zaznamenávat polohu kde byly provedeny. Díky tomu budou moci uživatelé srovnávat výsledky měření pro různé lokality a konfigurace. Naměřená data by bylo vhodné prezentovat i přes internet pro jejich širší využití.

Z důvodu požadavku prezentace dat bude vytvořen server, který bude poskytovat `API` pro filtrování v naměřených datech. S tím je samozřejmě spojena i synchronizace měřených dat a konfigurací mobilního zařízení se serverovou aplikací.

Cílem celého řešení je tedy poskytnout spolehlivé řešení pro měření kvality připojení ke vzdáleným bodům a následná prezentace výsledků uživateli.

Potřeby a požadavky na aplikaci — 2

Cílem aplikace je nabídnutí komplexního nástroje pro monitorování a hodnocení kvality síťového připojení. Z toho plynou požadavky na přesnost měření rychlosti a latence sítě. Očekává se, že aplikace poskytuje věrohodné a přesné informace o kvalitě připojení. Měřená data budou následně srozumitelně prezentována pro hlubší analýzu a vyhodnocování měřených výsledků.

Důležitým aspektem je také snadná ovladatelnost a intuitivní uživatelské rozhraní. Aplikace bude poskytovat přehledné uživatelské prostředí takové, že uživatelé budou moci aplikaci ovládat bez problémů a efektivně využívat její funkce bez potřeby hlubšího technického porozumění. To zahrnuje i jasnou a přehlednou vizualizaci výsledků, která umožní snadné porozumění prezentovaným datům.

Další funkcionalitu, kterou bude aplikace poskytovat, je ukládání historie výsledků měření. Tato funkce uživatelům umožní sledovat vývoj kvality svého připojení v čase a lépe tak identifikovat případné problémy nebo změny ve svém síťovém prostředí.

2.1 Poskytované funkce

Tato sekce se zaměřuje na podrobné představení všech klíčových funkcionalit vytvořené aplikace pro měření kvality síťového připojení. Každá funkce je pečlivě popsána a vysvětlena, aby bylo jasné, jak ji lze využít a jak přispívá k celkové hodnotě aplikace.

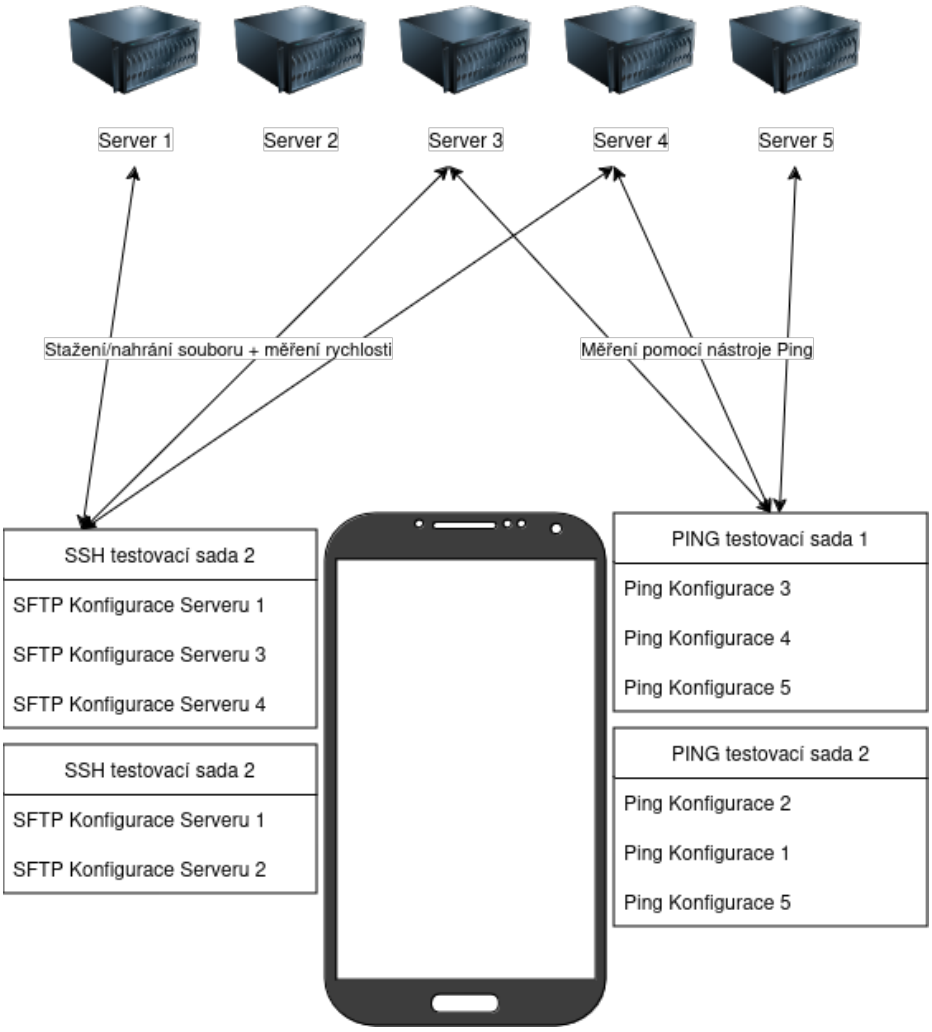
2.1.1 Předdefinování testovacích sad

Aplikace bude poskytovat možnost nadefinovat sadu konfigurací nad poskytovanými protokoly. Konfigurace představuje určitý typ šablony, podle které se budou provádět testy. Jednotlivé konfigurace si uživatel může sám upravit přímo v aplikaci. Konfigurace mohou být přidávány do tzv. testovacích sad, kde je možné spustit test nad více konfiguracemi najednou (sériově za sebou). Tato funkcionalita testovacích sad je vhodná pro měření, při požadavku, aby bylo měření prováděno vždy za

stejných podmínek. Další výhoda tohoto přístupu je, že usnadní uživateli testování více konfigurací najednou. Uživatel může vytvořit tolik sad, kolik potřebuje, a pro každou z nich může definovat různé parametry. Díky vytváření a správě předdefinovaných testovacích sad, tato aplikace poskytuje uživatelům flexibilní a efektivní nástroj pro monitorování a analýzu výkonnosti připojení.

2.1.2 Měření kvality připojení

Aplikace bude poskytovat možnost měření kvality připojení pomocí různých protokolů a nástrojů. Nad testovací sadou lze spustit měření, které poskytne srovnání kvality připojení k testovaným vzdáleným zařízením, viz obrázek 2.1.



Obrázek 2.1: Funkcionalita předdefinování testovacích sad spolu se spuštěním testů

Z obrázku 2.1 lze vidět, že aplikace bude ukládat více testovacích sad. Každá testovací sada obsahuje množinu konfigurací, pomocí kterých je prováděno testování

kvality připojení.

2.1.3 Modul pro měření kvality připojení

Aplikace bude poskytovat modul pro platformu Android, který bude přenositelný i do jiných aplikací. Tato funkcionalita může sloužit hlavně pro měření kvality připojení i v jiných aplikacích než aktuálně vyvíjené aplikaci.

2.1.4 Historizace a lokalizace měřených výsledků

Další důležitou vlastností bude ukládání naměřených hodnot včetně lokace, kde bylo měření prováděno. U měření pro mobilní zařízení je velice důležitou složkou, kde se aktuálně nacházíme. S lokalitou se obvykle mění i kvalita a rychlost připojení. Pro každý test je tedy nutné ukládat polohu, kde byl test proveden. Zároveň data musí být ukládána pro účel zpětné vyhledatelnosti prováděných testů. Ukládání naměřených výsledků a jejich lokalizace umožní dohromady jedinečný pohled na sbírané údaje. Bude možné pro danou lokalitu udělat jednorázový test a zjistit která konfigurace je pro nás nejvýhodnější, nebo naopak dělat dlouhodobé měření a zjišťovat jak se vyvíjí rychlost připojení dané lokality v čase. Dalšími využitími může být například zjišťování stability připojení, odezvy, a další.

2.1.5 Synchronizace dat

Aplikace bude poskytovat možnost registrace uživatele pro rozšířenou práci s daty. S registrací je spojeno několik výhod:

- **Synchronizace dat skrze více zařízení** - Ukládání naměřených dat, ale i ukládání konfigurací a testovacích sad není prováděno jen lokálně, ale vše je synchronizováno se vzdáleným serverem. Data jsou skrze registrovaný účet zálohována a slouží pro obnovu dat při změně mobilního zařízení.
- **Dostupnost dat** - Zvyšuje se dostupnost naměřených dat. Data mohou být prezentována i volně přes internetové služby například programům třetích stran pro další zpracování.

2.1.6 Webová dostupnost naměřených dat

Poslední, ale neméně důležitou funkcionalitou je poskytnutí `API`, popřípadě webové stránky, kde uživatel může k měřeným datům přistupovat přes internet. Tato funkcionalita je navržena především kvůli využitelnosti celého systému. Umožňuje zformátovat výstupní data do podoby následně využitelné k externí programové analýze a zpracování naměřených výsledků.

Srovnání s existujícími řešeními

3

Tato kapitola pojednává o konkurenčních aplikacích v oblasti monitorování kvality síťového připojení, nebo celkového monitorování připojení.

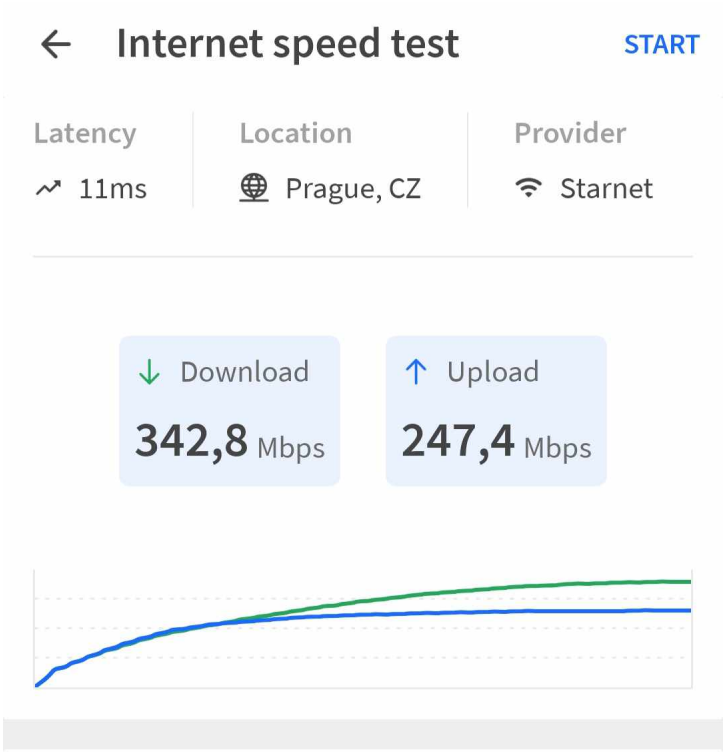
Aplikace je porovnávána se vzorkem několika nejpoužívanějších aplikací pro platformu Android. Snahou porovnání je vymezení funkcionalit implementované aplikace vůči ostatním aplikacím.

3.1 Srovnání aplikací

Tato kapitola se bude zabývat srovnáním aplikací určených pro měření kvality připojení pro platformu Android. Pro každou aplikaci jsou stručně shrnuty její vlastnosti a ceník.

3.1.1 Fing

Hojně využívaná aplikace, poskytující nejen měření rychlosti sítě, ale také kompletní diagnostiku aktuální sítě [Kri21]. Velkou výhodou této aplikace je její přehlednost a široká škála nástrojů pro analýzu a sledování sítě. Ke zpřístupnění všech funkcionalit aplikace poskytuje verzi `Premium`, která je placená. Verzi `Premium` lze platit buď měsíčně za cenu 75Kč, nebo při roční platbě 645Kč. Neplacená verze poskytuje užitečné nástroje jako jsou `Ping`, `Trace route`, `Wi-Fi Skener`, `Netstat` (sledování otevřených portů) a samozřejmě měření rychlosti stahování a nahrávání s využitím aktuálního připojení, viz obrázek 3.1. `Wi-Fi Skener` poskytuje podrobné informace o aktuálním `Wi-Fi` připojení, jeho nedostatky, informace o poskytovateli a bezpečnostní rizika. Placená verze je následně rozdělena na dva balíčky - `Starter` a `Premium`. `Starter` verze přidává například rozšířenou analýzu sítě `Wi-Fi` a zvětšení limitů pro počet monitorovaných zařízení. `Premium` již verze poskytuje plnohodnotnou správu sítě `Wi-Fi` a monitoring neomezeného počtu zařízení.



Obrázek 3.1: Ukázka měření rychlosti stahování a nahrávání pomocí aplikace `Fing`

3.1.2 Speedtest by Ookla

Jedna z nejspolehlivějších aplikací pro měření rychlosti připojení pro platformu Android [gsm20]. Hlavními funkcemi, které aplikace poskytuje jsou:

- Měření rychlosti stahování, nahrávání, doby obrátky a `Jitteru` (nestability).
- Měření kvality připojení pomocí video benchmarku. Dokáže určit přenosovou rychlost videa v různých kvalitách a vyvodit vhodnou konfiguraci pro sledování videa na různých typech zařízení.
- Historie měření a možnost jejich exportu ve formátu CSV.
- Sledování dostupnosti známých serverů, jako jsou YouTube, Facebook, Google a podobné.
- Mapa poskytující informace o kvalitě připojení pro poskytovatele internetových služeb.
- `VPN`, skrze které je možné se bezpečně pohybovat na internetu.

Aplikace existuje ve verzi zdarma a placené verzi, avšak hlavním rozdílem těchto verzí je využití `VPN`, kdy při neplacené verzi je možné využít pouze 2GB za jeden měsíc.

3.1.3 Network Analyzer Pro

Aplikace poskytující sadu nástrojů, jako jsou `Ping`, `Trace Route`, `Netstat` (sledování otevřených portů), sledování připojených zařízení k `Wi-Fi`, včetně jejich vlastností, možnost kontroly DNS, a měření rychlosti stahování a nahrávání. Další podstatnou výhodou je vizualizace výsledků a možnost zobrazení jednotlivých skoků (`Traceroute`) v mapě. Na druhou stanu, již prvotní verze této aplikace je placená. Placená verze se dá na `Google Play` zakoupit za 100Kč na dobu neomezenou.

3.2 Shrnutí

Ačkoliv implementovaná aplikace nebude poskytovat tak široké spektrum nástrojů a funkcionalit jako výše zmiňované nástroje, bude poskytovat hned několik unikátních vlastností. Hlavní výhodou vytvářené aplikace bude vytvoření modulu pro platformu Android, který bude volně přenositelný do jiných projektů a bude sloužit pro měření kvality připojení za využití různých technik. Dalším odlišujícím aspektem je, že aplikace bude ukládat a sledovat geografickou polohu pro každé měření. Tato funkce bude užitečná pro výběr vhodných konfigurací v konkrétních lokalitách. Přestože bude potřeba nastavit testovací sady a konfigurace, což může být pro některé uživatele složitější, aplikace bude poskytovat větší flexibilitu porovnání síťových připojení. Navíc přinese inovativní prvek ve formě `API`, které umožňuje přenos výsledků do jiných aplikací a platforem.

Měření kvality připojení

4

Tato část se zabývá popisem vhodných metrik, které lze použít pro měření kvality síťového připojení. Cílem je obsáhnout co nejširšího pokrytí možných testů tak, aby uživatel mohl posoudit celkový stav jeho připojení.

4.1 Postupy měření

Měření výkonu sítě pomáhá identifikovat úzká místa, diagnostikovat problémy a optimalizovat výkon sítě. [Man23] Zde je popsáno několik postupů, které lze využit k měřeni kvality připojení.

4.1.1 Využití reálné sítě

Použití skutečné sítě je nejčastější postup k testování kvality připojení. Hlavní výhodou tohoto přístupu je to, že poskytuje výsledky odpovídající skutečným podmínkám. Pomocí využití reálné sítě lze navíc odhalit problémy, které pomocí simulované nebo jiné sítě nelze odhalit. Na druhou stranu je měření pomocí reálné sítě náročnější, protože může být obtížné kontrolovat všechny atributy, které mohou ovlivnit naměřené výsledky.

4.1.2 Využití testovacího prostředí

Dalším přístupem, který lze použít k testování kvality připojení, je využití testovacího prostředí. Testovací prostředí je zjednodušená verze skutečné sítě, kterou lze zřídit v laboratorním prostředí. Hlavní výhodou použití testovacího prostředí je, že umožňuje kontrolovat topologii sítě a vzorce provozu, což může přinést přesnější výsledky než testování v reálné síti. Výsledky měření se však využívají spíše k diagnostickým účelům, jako třeba předpověď chování sítě. Kromě toho mohou být testovací zařízení levnější a snadněji se nastavují než skutečné sítě. Testovací zařízení však nemusí přesně odrážet složitost skutečné sítě a může být obtížné replikovat určité síťové podmínky v laboratorním prostředí.

4.1.3 Využití simulace

Simulace je oblíbeným přístupem k testování kvality připojení. Simulace zahrnuje vytvoření modelu sítě a spuštění měření na tomto modelu. Hlavní výhodou simulace je, že umožňuje testování v kontrolovaném prostředí, kde lze snadno modifikovat vlastnosti prostředí. Kromě toho může být simulace levnější a rychlejší než použití skutečné sítě nebo testovacího prostředí. Přesnost simulace závisí na kvalitě modelu a vytvořit přesný model složité sítě může být obtížné. Kromě toho může být obtížné přesně modelovat některé aspekty chování sítě, jako je časování a ztráta paketů.

4.1.4 Využití formálního modelu

Dalším přístupem, který lze použít k testování kvality připojení, jsou formální modely. Formální modely používají rovnice k popisu chování protokolu a jeho výkonnosti za různých podmínek. Hlavní výhodou tohoto přístupu je, že může poskytnout vhled do chování připojení a může být použit k předpovědi jeho výkonnosti v různých scénářích. matematické modely mohou být navíc méně nákladné a snadněji se nastavují než simulace nebo testovací prostředí. Matematické modely však nemusí přesně odrážet složitost reálné sítě a nemusí brát v úvahu určité aspekty chování sítě, které mohou ovlivnit výsledky měření.

4.1.5 Vyhodnocení

Podle požadavků na aplikaci je zapotřebí měřit připojení ke vzdáleným zařízením v reálném čase a prostředí. Nejvhodnější variantou pro měření je využití reálné sítě, avšak definice ostatních postupů měření může být důležitá pro získání co nejspolehlivějších výsledků. Například pomocí využití testovacího prostředí nebo simulace lze získat předpokládané výsledky měření, které lze následně srovnat s výsledky naměřenými za pomoci reálné sítě.

4.2 Metriky měření

Existuje několik různých metrik pro měření kvality připojení. Cílem práce je obsáhnout co nejvíce z nich, protože každá metrika má jinou výpovědní hodnotu a lze z ní vyvozovat jiné závěry.

4.2.1 Rychlost stahování

Jednou z nejdůležitějších a obvykle nejvyužívanějších metrik je rychlost stahování. Určuje, jak rychle lze přijímat data z určitého zařízení umístěného v počítačové síti (v našem případě Internetu) na používané zařízení. Rychlost stahování se měří v jednotkách dat za sekundu. Tato metrika ovlivňuje mnoho důležitých aspektů, jako

je rychlost načítání webových stránek, přenos videí, stahování souborů, hraní online her a jakékoliv další činnosti, které vyžadují příjem dat z internetu [KR16].

4.2.2 Rychlost nahrávání

Úzce související metrika s rychlostí stahování. Určuje, kolik dat za jednotku času je možné přenést směrem ven na cílové zařízení v Internetu. Uplatní se pro účely nahrávání souborů na cloudové úložiště, sdílení fotek nebo videí na sociálních médiích, odesílání e-mailů s přílohami, hraní online her nebo provádění video hovorů [KR16].

4.2.3 Jednosměrná latence

Metrika označující dobu, za kterou se datový paket dostane od odesílatele k příjemci v síti. Tato veličina je neměřitelná [Gid] a používá se místo toho `RTT`, viz sekce 4.2.4.

4.2.4 Doba obrátky (RTT)

Doba od odeslání prvního bitu paketu, který příjemce nejprve celý přijme a pak jej ihned odešle zpět, až do příjmu posledního bitu paketu. U této veličiny se předpokládá, že paket není nijak zpracováván příjemcem a je rovnou odeslán zpět. Tato metrika se také označuje jako obousměrná latence [KR16].

4.2.5 Paketová ztráta

Tato metrika měří, kolik paketů, které jsou odeslány z iniciačního zařízení, nedorazí do cílového zařízení. Tento problém je často způsoben špatným fyzickým připojením, síťovými zařízeními, nebo zahlcením síťových zařízení a případným zahazováním rámců a paketů. V `TCP` protokolech není ztráta paketů tolerována, a tak jsou zasílány opětovně. To může vést k většímu vytížení sítě [DL10] [KR16].

4.2.6 Jitter

`Jitter`, nebo spíše `cyklický jitter`, je vypočítáván z více měření latence. Jedná se o odchylku časů od průměrné hodnoty latence. Jitter je obzvláště důležitý pro real-time aplikace, jako jsou hry, videohovory a podobné oblasti vyžadující vysokou rychlost připojení. Pro sítě s vysokou rychlostí připojení je `Jitter` obzvlášť důležitý, protože i malé změny doby doručení mohou mít velký vliv na výkon [KR16] [Pet24].

4.2.7 Průchozí kapacita

Průchozí kapacita je množství dat, které může vaše internetové připojení přenést za určitý časový úsek. Jednotkou jsou data přenesená za sekundu. Je vhodná pro určení

rychlosti stahování nebo nahrávání souborů, přenos videí, prohlížení webových stránek a provádění dalších online aktivit. Rozdíl mezi rychlostí stahování/rychlostí nahrávání (4.2.1) a průchozí kapacitou je ten, že `průchozí kapacita` je maximální možná rychlost přenosu dat, zatímco rychlost stahování a nahrávání jsou skutečné rychlosti, kterých připojení dosahuje v konkrétním čase [KR16] [Pet24].

4.2.8 Kvalita služby - QoS

Míra, která ukazuje, jak efektivně síť zvládá různé druhy provozu. Může být ovlivněna jakoukoliv kombinací výše uvedených faktorů [KR16].

4.3 Protokoly

Tato kapitola se bude zabývat principy výběru vhodného internetového protokolu pro měření kvality připojení. Dále budou popsány jednotlivé protokoly, nebo nástroje nad nimi postavené, včetně jejich charakteristických vlastností a využití při měření kvality připojení.

Protokol je konvence nebo standard, podle kterého probíhá elektronická komunikace a přenos dat mezi dvěma koncovými body. Mezi hlavní protokoly využívané internetem se řadí rodina protokolů nad protokolem `TCP/IP`. Další hojně využívané protokoly jsou například `HTTP`, `DHCP`, `SMTP` a mnoho dalších.

Volba protokolu s vhodnými vlastnostmi je klíčovým prvkem a signifikantně ovlivňuje výsledky měření. Měření stejné metriky pomocí jiných protokolů může poskytovat jiné výsledky, například kvůli odlišnému šifrování dat. Zde jsou vyjmenovány požadavky, které by protokoly měly obsáhnout:

- **Rozmanitost** - Pomocí různých protokolů je možné měřit různé metriky provozu sítě, je tedy vhodné využít takové protokoly, aby bylo možné obsáhnout co nejširší spektrum měřených metrik.
- **Bezpečnost** - Pro přenos citlivých dat by protokol měl splňovat určité bezpečnostní standardy.
- **Využitelnost** - Vybrané protokoly by měly být hojně používány v praxi. Cílem tohoto požadavku je zajistit, že výsledky měření budou prakticky využitelné a relevantní pro většinu uživatelů.
- **Stabilita** - Tento požadavek úzce souvisí s využitelností. Znamená, že by akce nad protokolem měly být deterministické (předvidatelné).
- **Dostupnost v programovacím prostředí** - Implementace protokolu musí být dostupná v programovacích jazycích dostupných pro vývoj na platformě Android.

Pro případ stabilního a reprodukovatelného měření je potřeba, aby byl protokol stabilní. Dalšími důležitými atributy pro všechny vybírané protokoly jsou jejich využitelnost a dostupnost v programovacích jazycích. Vybraná sada protokolů by poté měla mít co největší rozmanitost měřených metrik.

4.3.1 ICMP (Internet Control Message Protocol)

ICMP je protokolem nacházejícím se na síťové úrovni OSI/ISO modelu. ICMP je povinnou součástí každé implementace protokolu IP, to znamená IPv4 i IPv6 [24]. Samotný protokol ICMP avšak nehraje roli při měření metrik sítě tak, jako nástroje implementované nad tímto protokolem. Zde jsou uvedeny dva nejpoužívanější nástroje pro práci s protokolem ICMP:

- **Ping** - Ping je založen nad protokolem ICMP a jedná se obvykle o malý paket dat, který slouží k diagnostice dostupnosti vzdálených zařízení. Velikost posílaných dat je od 8 bajtů až po 65535 bajtů. Větší velikosti však bývají často blokované z důvodu útoku `ping of death` [Zol24]. Další častou nevýhodou je, že protokol ICMP může být cílovým zařízením buď zakázán úplně, nebo může být snížena priorita obsluhy, nebo může být řízen podle pravidel definovaných cílovým zařízením. Omezení provozu ICMP obvykle probíhá kvůli útoku `ping flood`, který se řadí do DDoS útoků [Zol24]. Kvůli spolehlivosti přenosu se tedy obvykle volí velikost ICMP paketu 56 nebo 64 bytů a intenzita posílání zpráv taková, aby nezapříčinila přetížení systému. Díky své jednoduchosti poskytuje užitečný údaj o tom, za jak dlouhou dobu se vrátí malý nešifrovaný paket dat při poslání na vzdálené zařízení [SF11]. Pomocí nástroje Ping lze měřit primárně doba obrátky (4.2.4), jitter (4.2.6), a paketovou ztrátu (4.2.5).

  Příklad použití a výstupu nástroje Ping na Linuxovém prostředí může vypadat viz kód.

Zdrojový kód 4.1: Ukázka spuštění nástroje Ping na Linuxovém prostředí

```
1 user@user:~$ ping portal.zcu.cz
2 PING portal4.zcu.cz (147.228.52.136) 56(84) bytes of data.
3 64 bytes from portal4.zcu.cz (147.228.52.136): icmp_seq=1 ttl=52 time=10.6
  ↪ ms
4 64 bytes from portal4.zcu.cz (147.228.52.136): icmp_seq=2 ttl=52 time=10.4
  ↪ ms
5 64 bytes from portal4.zcu.cz (147.228.52.136): icmp_seq=3 ttl=52 time=10.7
  ↪ ms
6 64 bytes from portal4.zcu.cz (147.228.52.136): icmp_seq=4 ttl=52 time=13.4
  ↪ ms
```

```
64 bytes from portal4.zcu.cz (147.228.52.136): icmp_seq=5 ttl=52 time=36.2
↪  ms
--- portal4.zcu.cz ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 10.426/16.281/36.190/10.015 ms
```

Kód 4.1 ukazuje použití a výstup nástroje `Ping`. `Ping` využívá pro přenos paketu velkého 64B a pro každý odeslaný požadavek zaznamenává dobu obrátky (`RTT`). Při ukončení volání nástroje `Ping` jsou následně vypsány statistiky, jako je minimální, maximální a průměrný čas doby obrátky.

- **Traceroute** - `Traceroute` je jako `Ping` nástrojem, který je založený nad protokolem `ICMP`. Slouží k diagnostice sítě tak, že vypíše všechny uzly (směrovače), přes které jde požadavek z lokálního až na vzdálené zařízení. Zároveň poskytuje informaci o tom, jaké zprávy přišly z jednotlivých směrovačů. Pomocí `traceroute` lze detekovat, kde se v komunikačním schématu vyskytuje zvětšení latence [She24]. Na druhou stranu je obvykle těžší na implementaci, protože některé směrovače nemusí odpovídat správně a tím pádem může celý proces zabrat více času a být méně konfigurovatelný než `Ping`. Pomocí nástroje `traceroute` lze měřit primárně doba obrátky (4.2.4), jitter (4.2.6), a paketovou ztrátu (4.2.5). Může však poskytovat detailnější informace než `Ping`.

4.3.2 FTP (File Transfer Protocol)

Tento protokol je jeden z nejstarších standardních protokolů používaných na internetu. Je navržen k přenosu souborů mezi cílovým zařízením a klientem v síti, přičemž může být použit jak pro stahování, tak pro nahrávání souborů. `FTP` nabízí jak stahování a nahrávání souborů, tak rozšířené možnosti manipulace se soubory, jako je správa oprávnění vzdálených souborů a snadná navigace ve vzdáleném souborovém systému [Ker24].

4.3.3 SFTP (Secure File Transport Protocol)

Protokol `SFTP` je šifrovanou verzí protokolu `FTP` a poskytuje navíc šifrovaný přenos dat, včetně samotných přihlašovacích údajů. Pro přenos souboru je nejčastěji využíváno protokolu `SSH-2`, ale může využívat i protokolu `SSH` [ssh].

Pomocí protokolu `SFTP` lze měřit primárně rychlost stahování a nahrávání souborů využívajících šifrování pomocí `SSH-2`.

4.3.4 SCP (Secure Copy Protocol)

Jedná se o síťový protokol, který poskytuje zabezpečený způsob kopírování souborů mezi lokálním a vzdáleným zařízením nebo mezi dvěma vzdálenými zařízeními. Pro přenos využívá protokol SSH-2. Na rozdíl od SFTP neposkytuje protokol možnost rozšířené manipulace se souborovým systémem, jako například listování podsložek a nastavení oprávnění [Vil].

Pomocí protokolu SCP lze měřit primárně rychlost stahování a nahrávání souborů využívajících šifrování pomocí SSH-2.

4.3.5 HTTP(s) (Hypertext Transfer Protocol Secure)

Aplikační protokol navržený pro přenos hypertextových dokumentů, jako jsou webové stránky. V dnešní době velice rozšířený a standardizovaný protokol. HTTP umožňuje nahrávání i stahování souborů. Protokol HTTP(s), což je zabezpečená verze HTTP, šifruje komunikaci mezi klientským zařízením a cílovým zařízením [Gou+02].

Pomocí protokolu HTTP(s) je možné měřit především rychlost stahování a nahrávání, což může simulovat například rychlost načítání webových stránek, které také využívají protokol HTTP(s).

4.3.6 iPerf3

iPerf3 je vysoce konfigurovatelný a efektivní nástroj určený pro měření rychlosti síťového připojení. iPerf3 dokáže měřit propustnost, rychlost stahování a nahrávání, ztrátu paketů a jitter [Dug]. Hlavní nevýhodou je, že je tento nástroj navržený architekturou klient-server a tak se pro každé měření je potřeba spustit iPerf3 server na cílovém uzlu.

4.4 Volba velikosti bufferu

Buffer, neboli vyrovnávací paměť označuje paměť, do které jsou dočasně ukládány data před jejich odesláním nebo přijetím po síti. Pomáhá vyrovnávat výkyvy v rychlosti odesílání nebo přijímání dat a zajišťuje, že odesílatel a příjemce mohou pracovat svým vlastním tempem bez přerušení [Kum23].

Velikost přenosového bufferu je pro měření klíčová vlastnost, která dokáže výrazně ovlivnit výsledky měření [Kan04]. Velikost bufferu záleží na mnoha faktorech, avšak hlavními faktory jsou topologie sítě a vytížení sítě. Pro zjednodušení by se však dalo říct, že čím větší je propustnost sítě, tím větší buffer můžeme nastavit a dostáváme lepších výsledků. Pro srovnání je zde ilustrován jednoduchý příklad

několika měření pro různé velikosti bufferů s využitím protokolu SFTP, mobilním 5G připojením a velikostí stahovaného/nahrávaného souboru o velikosti 50MB.

Tabulka 4.1: Srovnání rozdílů velikosti bufferu při 5G připojení

Velikost bufferu	Rychlost stahování	Rychlost nahrávání
512B	67,452 kB/s	18,373 kB/s
1kB	139,462 kB/s	34,851 kB/s
8kB	623,180 kB/s	282,338 kB/s
64kB	4,331 MB/s	2.055 MB/s
128kB	7,794 MB/s	8.036 MB/s
512kB	19,545 MB/s	51,235 MB/s
1MB	15,426 MB/s	48,141 MB/s

Z tabulky 4.1 lze vidět, že se zvětšující se velikostí bufferu se zvyšuje i efektivita přenosu. Tato hodnota se nemůže ale zvyšovat do nekonečna a tak v posledním řádku je vidět, že došlo ke snížení přenosové rychlosti. V určitém bodě začne velikost bufferu zhoršovat kvalitu přenosu a může docházet k častějšímu zahazování jednotlivých paketů [Lap19], tím následně klesá i přenosová rychlost.

Jako kontradikci tohoto příkladu však lze uvést i druhé měření, při podobné konfiguraci, avšak kvalitě mobilního připojení EDGE, viz tabulka 4.2.

Tabulka 4.2: Srovnání rozdílů velikosti bufferu při E připojení

Velikost bufferu	Rychlost stahování	Rychlost nahrávání
512B	835 B/s	613 B/s
1kB	1.394 kB/s	835 B/s
8kB	1.594 kB/s	1.009 kB/s
64kB	1.309 kB/s	1.049 kB/s
128kB	0B/s - Vypršení časového limitu	0B/s - Vypršení časového limitu
512kB	0B/s - Vypršení časového limitu	0B/s - Vypršení časového limitu
1MB	0B/s - Vypršení časového limitu	0B/s - Vypršení časového limitu

Velikost bufferu hraje klíčovou roli v optimalizaci přenosu dat po síti. Zvětšování bufferu vede k růstu rychlosti stahování a nahrávání, avšak do určitého bodu. Dále

dochází ke zhoršení kvality přenosu a klesání rychlosti. Výběr optimální velikosti bufferu závisí na typu sítě (5G, EDGE), topologii sítě a vytížení sítě.

Tabulky 4.1 a 4.2 demonstrují vliv velikosti bufferu na rychlost stahování a nahrávání pro 5G a EDGE připojení. V obou případech je patrný nárůst rychlosti s rostoucím bufferem do určitého maxima, po kterém dochází k poklesu.

4.5 Shrnutí

Po analýze protokolů a nástrojů pro měření kvality síťového připojení byly vybrány pro aplikaci protokoly HTTP(s), SFTP a nástroj Ping. Pomocí vybraných protokolů a nástrojů lze dosáhnout měření rychlosti stahování, rychlosti nahrávání, oboustrané latence (RTT) a při hlubší analýze z ní vypočítaného Jitteru.

Využité nástroje 5

V této kapitole jsou popsány různé technologie, nástroje a postupy, které budou využity při tvorbě výsledné aplikace. Konkrétně je zde probrán způsob připojení, uživatelské rozhraní a část související se serverovým připojením. U způsobu připojení byly prozkoumány jeho možnosti a vyhodnocení souvisejících nástrojů. U uživatelského rozhraní je prozkoumán přístup ke tvorbě Android aplikace s ohledem na výběr programovacího jazyka a ukládání dat. V serverové části bylo třeba prozkoumat složitost a rozsáhlost projektu, správu a přístupy serveru a volba `API` pro komunikaci mezi serverem a aplikací.

5.1 Měření kvality připojení

Pro realizaci měření kvality připojení je vhodné využít standardních implementací daného protokolu nebo nástroje. Vhodným řešením jsou tedy buď standardní knihovny pro daný protokol/nástroj, nebo funkce a nástroje poskytované samotným operačním systémem.

Dalším důležitým aspektem je zpoždění měření vnitřními vlivy. Vnitřními vlivy je myšlena efektivita dané implementace. Byly provedeny testy rychlosti stahování pomocí dvou implementací protokolu `SFTP`. Jeden set měřených testů zobrazuje rychlost stahování s využitím knihovny `SSHJ` vzhledem k velikosti bufferu v tabulce 5.1 a druhý set znázorňuje stejné hodnoty v tabulce 5.2, ale pro knihovnu `libssh2`.

Tabulka 5.1: Rychlosti stahování s využitím knihovny `SSHJ`

Velikost bufferu	Rychlost stahování
512B	17.8206 kB/s
1kB	33.110 kB/s
8kB	158.972 kB/s
64kB	449.431 kB/s
128kB	748.712 kB/s

(tabulka pokračuje na další stránce)

Tabulka 5.1 (*pokračování z předchozí stránky*)

Velikost bufferu	Rychlost stahování
512kB	3.899 MB/s
1MB	7.289 MB/s

Tabulka 5.2: Rychlosti stahování s využitím knihovny `libssh2`

Velikost bufferu	Rychlost stahování
512B	67,452 kB/s
1kB	139,462 kB/s
8kB	623,180 kB/s
64kB	4,331 MB/s
128kB	7,794 MB/s
512kB	19,545 MB/s
1MB	15,426 MB/s

Při porovnání efektivity měření tabulek 5.1 a 5.2 je na první pohled zřejmé, že implementace `libssh2` je výrazně efektivnější. Z tohoto srovnání lze tedy předpokládat, že pro přesnější měření rychlosti je výhodnější využít nízkoúrovňových knihoven.

5.1.1 Měření protokolu SFTP

Pro měření rychlosti přenosu pomocí protokolu `SFTP` byly porovnány čtyři knihovny, `JSCH` [JSC], `SSHJ` [SSH], `libssh` [Liba] a `libssh2` [Libb].

Zpočátku analýzy knihovny `JSCH` bylo zjištěno, že knihovna neposkytuje volbu velikosti bufferu, proto byla jako možnost pro implementaci zamítnuta. `SSHJ` volbu velikosti bufferu poskytuje a tak byl vytvořen jednoduchý příklad pro měření rychlosti stahování pomocí knihoven. Následně byly výsledky knihovny `SSHJ` porovnány s jednoduchou implementací knihovny `libssh2`. Měření bylo prováděno na mobilní síti 5G při velikosti stahovaného souboru 100`MB`. Porovnání výsledků měření je uvedeno v tabulce 5.3.

Tabulka 5.3: Porovnání rychlostí knihoven SSHJ a libssh2

Velikost bufferu	Rychlost stahování SSHJ	Rychlost stahování libssh2
512B	17.8206 kB/s	67,452 kB/s
1kB	33.110 kB/s	139,462 kB/s
8kB	158.972 kB/s	623,180 kB/s
64kB	449.431 kB/s	4,331 MB/s
128kB	748.712 kB/s	7,794 MB/s
512kB	3.899 MB/s	19,545 MB/s
1MB	7.289 MB/s	15,426 MB/s

Kvůli signifikantně lepším výsledkům knihovny libssh2 oproti SSHJ byla vybrána z těchto dvou knihovna libssh2. Jako poslední byly porovnávány knihovny libssh a libssh2. Obě knihovny jsou psané v jazyce C a poskytují široké spektrum operací nad protokoly SSH a SFTP. Zároveň jsou obě knihovny dlouhodobě udržované s velkou komunitou. Největším rozdílem těchto knihoven je to, že libssh umožňuje na rozdíl od libssh2 operace nad SSH serverem, jako jeho vytvoření správu klientských připojení. Z tohoto důvodu je i velikostně větší a proto byla vybrána knihovna libssh2.

5.1.2 Měření nástrojem Ping

Pro měření metrik pomocí nástroje Ping byla nalezena pouze jedna dominantní knihovna. Knihovna se jmenuje Icmp4a [Icm] a je implementována v Kotlinu. Přestože je Kotlin jazyk vysokoúrovňový, efektivita implementace knihovny je vysoká, protože knihovna byla vyvinuta pouze pro platformu Android a využívá android.system.OS třídy, která poskytuje funkce obvykle dostupné pouze pro nízkoúrovňové jazyky, jako jsou C nebo C++, viz kód 5.1.

Zdrojový kód 5.1: Ukázka implementace Icmp4A

```
1
2     private fun createSocketForDestination(destination: InetAddress):
      ↪  FileDescriptor {
3         try {
4             val fd = when (destination) {
5                 is Inet4Address -> Os.socket(OsConstants.AF_INET,
                  ↪  OsConstants.SOCK_DGRAM, OsConstants.IPPROTO_ICMP)
6                 is Inet6Address -> Os.socket(OsConstants.AF_INET6,
                  ↪  OsConstants.SOCK_DGRAM, OsConstants.IPPROTO_ICMPV6)
7                 else -> throw IllegalStateException("...")
```

```
8               }
9
10              if (!fd.valid())
11                  throw Icmp.Error.SocketException(message = "Created file
                    ↪  descriptor is invalid")
12              return fd
13          } catch (e: ErrnoException) {
14              throw Icmp.Error.SocketException(
15                  message = "Socket creation failed",
16                  cause = e)
17          }
18      }
```

V kódu 5.1 lze vidět vytváření socketů pro přenos ICMP paketů. Tento kód je obdobou pro standardní kód C, jak je vidět v ukázce kódu 5.2.

Zdrojový kód 5.2: Vytvoření ICMP socketu v C

```
1   sockfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_ICMP);
```

Struktura a komunikační schéma protokolu ICMP nejsou složité a tak je možné měření implementovat i samostatně v nízkoúrovňovém jazyce C++ bez využití knihoven. Přestože by tedy bylo možné využít implementaci jazyka C++ bez knihoven, byla pro tuto práci zvolena knihovna Icmp4a.

5.1.3 Měření protokolu HTTP(s)

Pro implementaci měření pomocí protokolu HTTP(s) byly porovnány tři možnosti. První možností byla implementace pomocí třídy java.net.URL. Tato třída je standardně obsáhlá v jazyce Java a nepotřebuje žádné další závislosti. Bohužel není tato třída dobře konfigurovatelná a lze měřit přenos pouze jako celek, to znamená včetně navázání spojení.

Druhou možností byla knihovna AsyncHttpClient psaná taktéž v jazyce Java. Tato knihovna již poskytovala možnost měření jednotlivých iterací, avšak stále postrádala možnost rozšířené konfigurace [Cli24].

Poslední možností byl knihovna libcurl. Knihovna libcurl je určena pro jazyk C/C++. Poskytuje snadnou práci s protokolem HTTP(s) a velké množství jeho konfigurací. Knihovna poskytuje také údaje, jako jsou čas posledního odeslaného bitu, nebo čas prvního přijatého bitu, což může značně zlepšit výsledky měření. Dále má knihovna velikou komunitu a kvalitní dokumentaci [cur24].

5.2 Uživatelské rozhraní

Pro výběr `programovacího jazyka/frameworku` na Android platformě je důležité brát v úvahu hned několik kritérií, jako jsou dostupnost knihoven a nástrojů, podpora pro mobilní platformy, výkonnost a také produktivita a snadnost vývoje. Dle toho lze zvážit několik programovacích jazyků, nebo frameworků:

- **Java** - Robustní jazyk, velké množství knihoven a nástrojů, široká podpora pro Android, silné komunitní zázemí, podporuje `JIT` kompilaci pro optimální výkon. Pomalejší vývoj oproti moderním jazykům.
- **Kotlin** - Speciálně navržený pro Android, poskytuje vysokou produktivitu vývoje a vylepšený syntax pro větší čitelnost a jednoduchost. Kotlin je kompatibilní s Javou, je avšak kompaktnější než Java v Androidu, takže k dosažení stejného výsledku je třeba méně kódu.
- **React Native** - `Framework` nad `JavaScriptem` a `TypeScriptem`. Umožňuje multiplatformní a rychlý vývoj, má velkou komunitu a mnoho dostupných nástrojů a knihoven. Oproti tomu poskytuje menší výkon než nativní jazyky.
- **Flutter** - Nástroj pro vývoj pracující nad jazykem `Dart`. Podpora pro multiplatformní vývoj, rychlý renderovací stroj. Nevýhodami jsou méně poskytovaných knihoven než v nativních jazycích a horší komunikace s nativními funkcemi.

Jazyky `Java` a `Kotlin` mají navíc jednu klíčovou výhodu, kterou je snadná integrace volání nativních funkcí (`C/C++`). Po analýze byly zvoleny jazyky `Java` a `Kotlin` pro implementaci uživatelského rozhraní.

5.2.1 Ukládání dat

Pro aplikaci je důležité mít data uložena pro jejich prohlížení `offline`. Je tedy nutné využít databázi, která je poskytována pro platformu Android v jazycích `Java` a `Kotlin`. Nalezeny byly dvě implementace databáze: `SQLite` a `Room`. `SQLite` je knihovna, která poskytuje systém správy relačních databází. Její nevýhodou je ovšem to, že veškeré operace nad databází, jako je tvoření tabulek, relací, atd. je nutné vytvářet ručně. Z tohoto důvodu byla zvolena databáze `Room`, která je abstrakcí nad `SQLite`. Databáze `Room` je speciálně navržena pro platformu Android. Poskytuje jednoduché a automatické mapování objektů do databáze, zjednodušenou tvorbu `SQL` dotazů a jejich validaci během kompilace programu [Zin].

5.3 Serverové řešení

Při výběru nástrojů a jazyků pro serverové řešení je třeba pečlivě zvážit několik faktorů. Hlavními faktory jsou volba `API` a složitost projektu.

5.3.1 API

Pro tvorbu `API` je nutné posouzení a následné vyhodnocení výhod a nevýhod několika nejznámějších a nejvíce využívaných `API`:

- **REST** - Starší a zavedený standard, který je využíván většinou webových aplikací. Je jednoduchý, což usnadňuje jeho udržitelnost a škálovatelnost. Nicméně, `REST` může být v některých situacích neefektivní, protože klient nemá možnost specifikovat, jaké údaje od serveru potřebuje, což může vést k přenosu nadbytečných dat [Big].
- **SOAP** - Obsahuje přísná pravidla a pevné standardy pro zasílání zpráv, díky nimž může být bezpečnější. Na druhou stranu postrádá snadnou rozšiřitelnost a formát přenášených zpráv je omezen pouze na `XML`.
- **GraphQL** - Nejmodernější technologie z vyjmenovaných, která zároveň poskytuje větší flexibilitu a efektivitu. Umožňuje klientovi specifikovat, jaké údaje od serveru potřebuje, což může významně snížit objem přenášených dat. Nicméně, může být složitější a vyžadovat více času na implementaci a udržování.

Z výše vyjmenovaných `API` bylo jako nejvhodnější vybráno `REST API` kvůli jeho jednoduchosti a škálovatelnosti.

5.4 Správa identit a přístupů

Jak bylo psáno v části 2.1.5, vytváření aplikace bude poskytovat přihlášení uživatele pro synchronizaci dat. Tato část práce se věnuje využití nástrojů pro správu identit a přístupu. Nástroje slouží k zabezpečení komunikace mezi dvěma body a poskytují spolehlivou autentizaci a autorizaci uživatele [FP12].

Pro analýzu nástrojů je potřeba nejdříve stanovit tři hlavní požadavky, které budou důležité pro implementaci a funkcionalitu celého systému. Prvním požadavkem je bezpečnost daného nástroje, protože nástroje ukládají uživatelské údaje, které by neměly nikdy uniknout. Druhým důležitým faktorem při výběru je poskytnutí typu ověřování (autentizace heslem, tokenem, atd.) [FP12] [Rao] a posledním důležitým faktorem je podpora daného nástroje a jeho jednoduchá integrace pomocí přidružených knihoven do aplikace Android.

Pro analýzu byly vybrány pouze nástroje, které splňují první požadavek a tedy standardně se používají pro firemní řešení [Aut][Ide][Key]:

- **Keycloak** - Keycloak je open-source řešení pro správu identity a přístupu, které zajišťuje autentizaci a autorizaci uživatelů. Je flexibilní a snadno konfigurovatelný. Neposkytuje ovšem žádnou knihovnu pro integraci pro platformu Android a jediná možnost komunikace s nástrojem je přes `REST API`.
- **Google Cloud Identity** - Řešení poskytované společností `Google`. Poskytuje širokou škálu konfigurací a služeb. Má ovšem složitější integraci pro platformu Android než `KeyCloak`.
- **Auth0** - Řešení od společnosti `Okta`. Poskytuje jednoduché rozhraní pro konfiguraci projektu a širokou škálu poskytovaných funkcionalit. Dále poskytuje knihovnu pro platformu Android, která výrazně usnadňuje správu účtu v zařízení.

Vzhledem k vlastnostem jednotlivých nástrojů byl vybrán pro další implementaci `Auth0`, jelikož poskytuje nejlepší řešení správy identit a přístupů pro platformu Android.

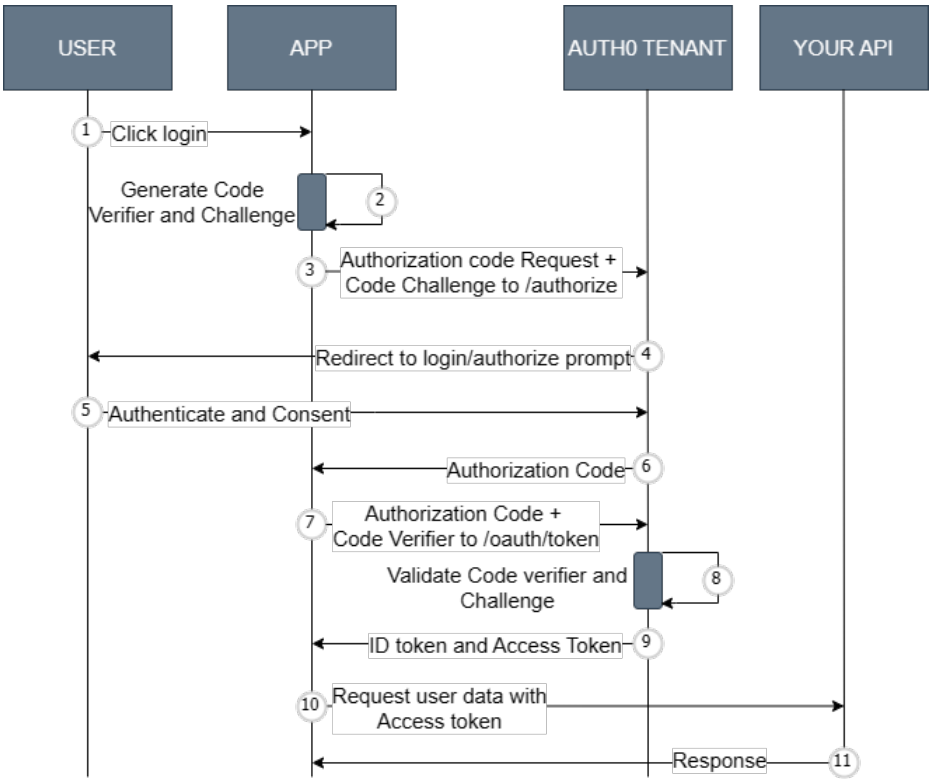
Návrh komunikačního schématu — 6

V této kapitole bude navrženo komunikační schéma s využitím nástrojů z předchozí kapitoly (5).

Komunikační schéma by mělo být co nejrobustnější a škálovatelné pro případné rozšíření projektu a přidání nových funkcionalit. Pro návrh schématu je důležité vědět, že se skládá ze tří složek, klienta (mobilní aplikace), serveru a nástroje pro správu identit a přístupů. V kapitole správa identit a přístupů (5.4) byl vybrán jako nejvhodnější nástroj `Auth0`.

`Auth0` poskytuje protokoly zabezpečení `OpenID` a `OAuth 2.0`. Protokol `OpenID` je standard, který umožňuje uživatelům přihlašovat se na různé webové stránky pomocí stejného poskytovatele identity [kon23]. `OAuth 2.0` je protokol, který umožňuje aplikacím a rozhraním API třetích stran bezpečný přístup k uživatelským datům. Protokol `OAuth 2.0` zároveň poskytuje několik postupů, kterými může autorizace probíhat [aut23]. Jako nejvhodnější postup pro autorizaci a autentizaci uživatele byl vybrán autorizační kód s ověřovacím klíčem pro výměnu kódů (PKCE). Do jisté míry dokáže řešit knihovna `Auth0` pro Android tyto věci implicitně a uživatel obdrží pouze tokeny, které jsou nutné pro volání aplikací třetích stran. Tato vlastnost byla plně využita a bude popsána dále v části 10.3.2. Obrázek 6.1 znázorňuje tok dat při použití tokenů pro autentizaci a autorizaci uživatele.

Z obrázku 6.1 lze vidět postup autentizace a udělení tokenů uživateli. Jako první proběhne přihlášení uživatele pomocí mobilní aplikace. Následně `Auth0` knihovna v Android zařízeni vygeneruje verifikační kód, z něj vytvoří výzvu (`Code Challenge`) a přesměruje uživatele na autorizační server `Auth0` spolu s výzvou. Autorizační server `Auth0` přesměruje uživatele na výzvu k přihlášení a autorizaci. Uživatel se ověří pomocí jedné z nakonfigurovaných možností přihlášení (jméno/heslo, Facebook, Google, ...) a může se mu zobrazit stránka se souhlasem, na které jsou uvedena oprávnění, která aplikace `Auth0` získá. Autorizační server `Auth0` uloží kód výzvy a přesměruje uživatele zpět do aplikace s autorizačním kódem, který je vhodný pro jedno použití. Aplikace pošle zpět autorizační kód spolu s ověřovacím kódem na `Auth0` autorizační server. Autorizační server ověří zaslaný ověřovací kód a výzvu a



Obrázek 6.1: Udělení tokenů a jejich využití [aut23]

pokud je ověření správné zašle aplikaci `ID`, přístupový a obnovovací token. Přijatý přístupový token je následně využit k autorizaci vzdálenému serveru, který po jeho ověření vrátí požadovaná data.

Nastavení nástroje pro správu identit a přístupů

7

V této kapitole je popsáno nastavení a konfigurace nástroje `Auth0`, který slouží k zabezpečení komunikace a ověření autentizace a autorizace uživatele. Aby byl `Auth0` kompletně nastavený, musí se nejprve vytvořit instance aplikace pro platformu Android na stránkách Auth0. Po vytvoření se musí upravit její nastavení a následně je potřeba vytvořit instanci `API` pro serverovou část řešení.

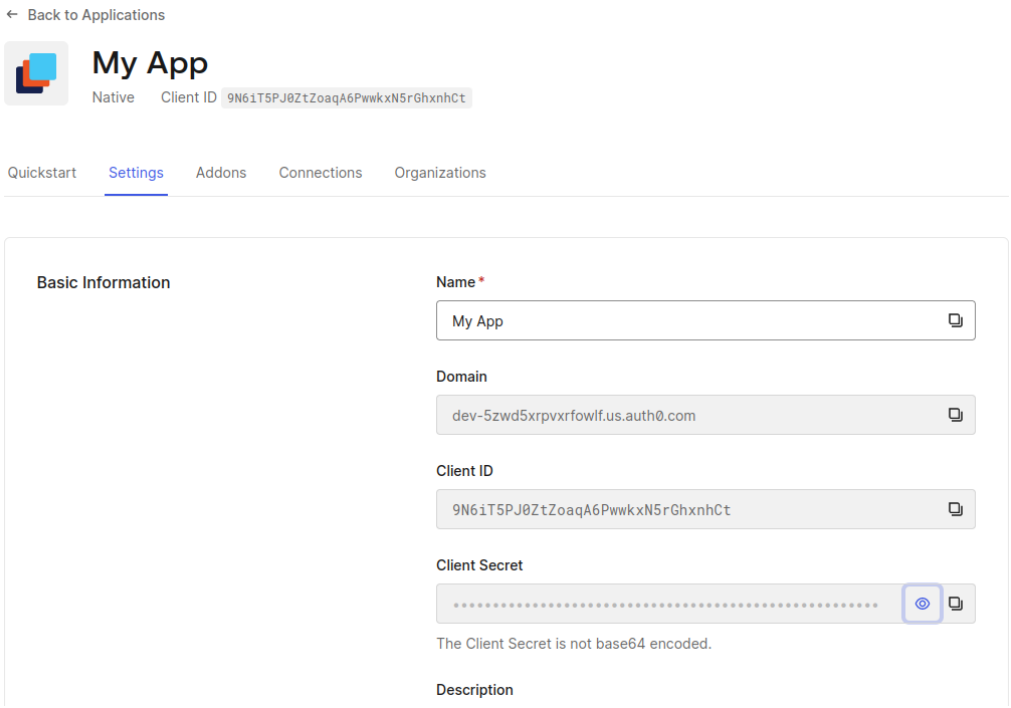
Je nutné uvést, že je povinné se před používáním `Auth0` registrovat na Auth0 stránce. Po úspěšné registraci bude provedeno přesměrování na domovskou stránku.

Důležitými pojmy pro využití nástroje `Auth0` jsou:

- **Client ID** - Jednoznačný identifikátor aplikace.
- **Scope** - Alternativa rolí, používá se při využití ověření pomocí tokenů a definuje oprávnění uživatele. Aplikace musí požádat o oprávnění k přístupu k informacím jménem uživatele. Rozsahy definují konkrétní akce, které mohou aplikace jménem uživatele provádět [Aut24c].
- **Callback** - Adresa, která je volána vždy po úspěšném přihlášení/odhlášení. Tato adresa je složena jako `SCHEME://DOMAIN/android/ID_ANDROID_APLIKACE/callback`, viz obrázek 7.2. Podrobnější vysvětlení `ID_ANDROID_APLIKACE` je v části 10.3.1, jde o identifikaci balíčku Android aplikace z `Manifest.xml`.
- **Scheme** - Řetězec vyžadovaný pro přihlášení a odhlášení Android zařízení pomocí `Auth0`. Uživatel si může zvolit libovolný název.
- **Domain** - Doména, na které se nachází `Auth0` autentizační server.
- **Audience** - `URL`, která nemusí to být veřejně dostupná, [Aut24b]. Definuje zamýšleného zprostředkovatele pro využívané tokeny. V případě Android aplikace je využívaným zprostředkovatelem `Auth0 API`.

7.1 Nastavení nástroje pro platformu Android

Na domovské stránce se v levém menu klikne na `Applications` > `Applications`. Následně se na nové stránce objeví tlačítko `Create Application`, které umožní vytvoření nové aplikace. Po stisku tlačítka je zobrazena nabídka, v níž je nutné zvolit možnost `Native Application` (pro platformu Android) a nakonec vybrat jméno aplikace. Tímto způsobem je založena aplikace, která bude využívána na platformě Android. Po založení lze procházet nastavení této aplikace, viz obrázek 7.1.

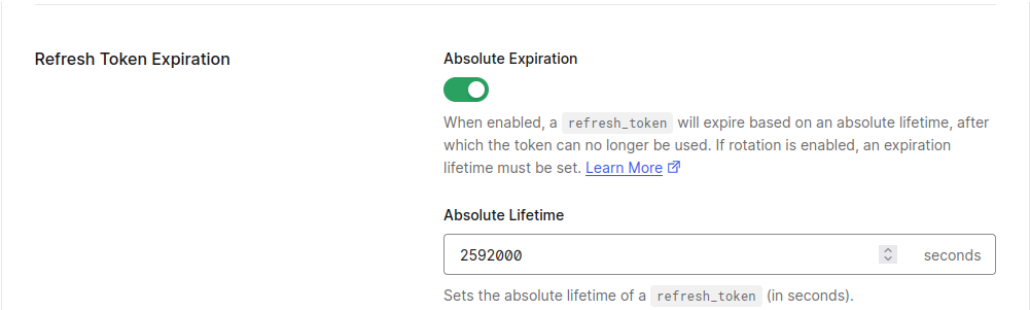


Obrázek 7.1: Ukázka `Domain`, `Client ID` a `Client Secret` hodnot pro Android aplikaci

Jako další krok je důležité nastavit trvanlivost obnovovacího tokenu, viz obrázek 7.3. Nastavení trvanlivosti obnovovacího tokenu je důležité hlavně z důvodu, že přístupový token [Aut24a] nemá obvykle z bezpečnostních důvodů dlouhou životnost (řádově desítky minut). Je ovšem důležité z hlediska uživatelské přívětivosti, uživatele pokaždé po vypršení přístupového tokenu neodhlašovat.

Z toho důvodu existuje obnovovací token, pomocí kterého lze dostat od autorizačního serveru nový obnovovací a přístupový token.

Obrázek 7.2: Nastavení `callback` adres pro přihlášení a odhlášení na platformě Android

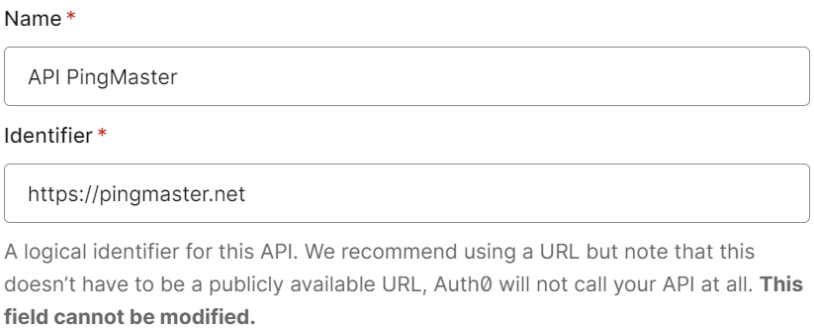


Obrázek 7.3: Nastavení trvanlivosti obnovovacího tokenu

7.2 Nastavení nástroje pro přístup k API

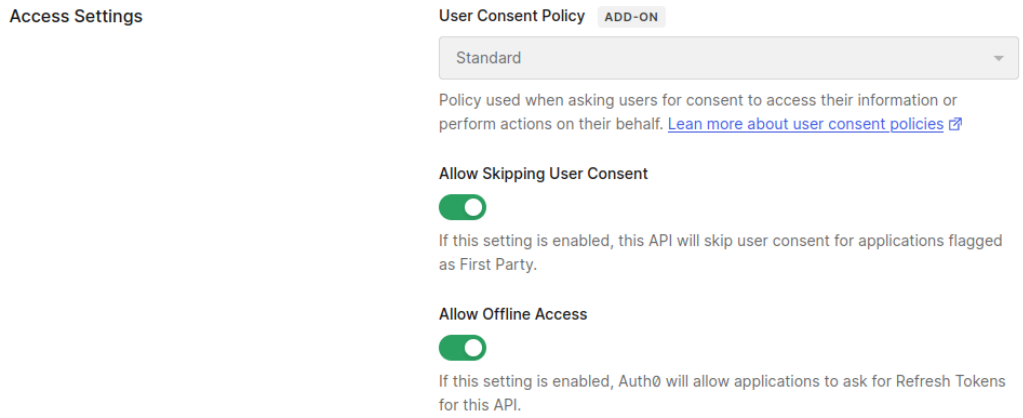
Dalším krokem pro nastavení nástroje `Auth0` je jeho nastavení pro komunikaci klientského zařízení (Android) s `API`.

Na domovské stránce se v levém menu klikne na `Applications` > `APIs`. Na nové stránce se klikne na tlačítko s textem `Create API`. Po stisknutí tlačítka vyskočí dialog, ve kterém se vyplní název aplikace a `URL` adresa. Vyplňovaná `URL` adresa nemusí vůbec existovat a není nijak volána ze strany `Auth0`, ale je nutné ji vyplnit, viz obrázek 7.4. Vyplňovaná `URL` adresa je dále používána jako `Audience`.



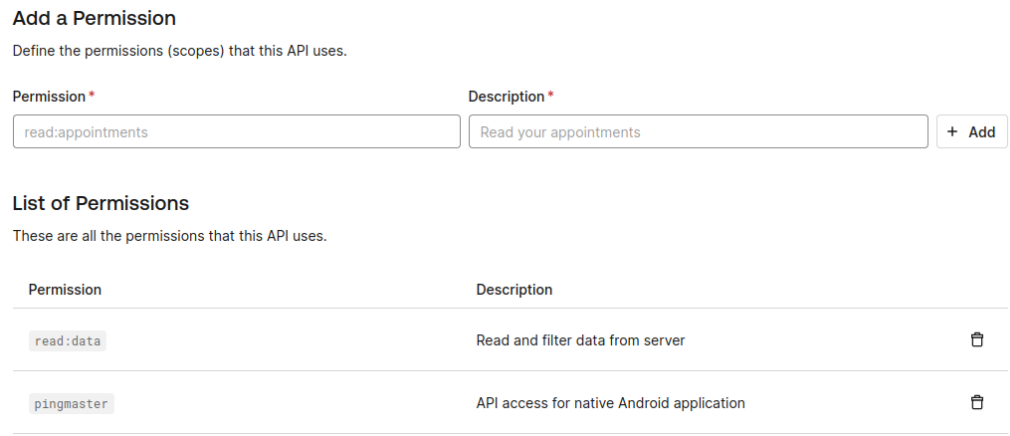
Obrázek 7.4: Dialog pro vytvoření `Auth0 API`

Po založení lze upravovat nastavení nově vytvořeného `API`. Jediným důležitým nastavením v sekci `Settings` nově založeného `API` je povolení využití obnovovacích tokenů, viz obrázek 7.5.



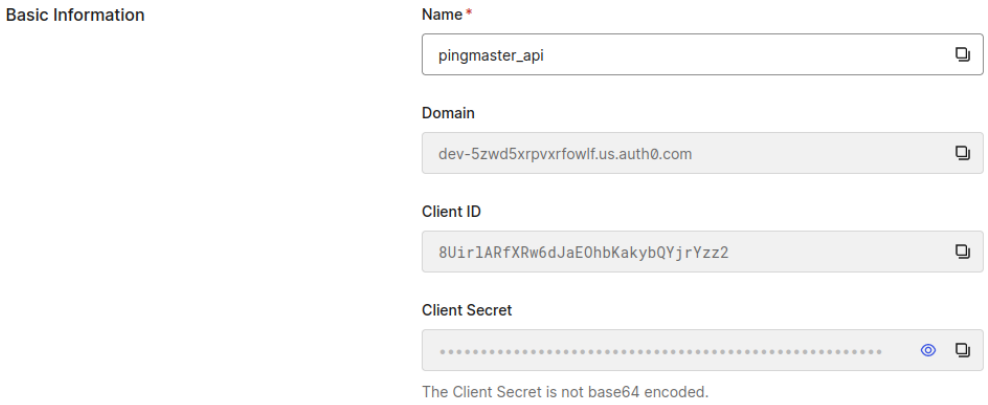
Obrázek 7.5: Povolení obnovovacích tokenů pro `Auth0 API`

Posledním krokem v nastavení `Auth0` je definování `Scopes` pro přístup k jednotlivým zdrojům. Pro nastavení je třeba kliknout na záložku `Permissions`, kde se vytvoří dva různé přístupy (`Scope`), viz obrázek 7.6.



Obrázek 7.6: Nastavení `scopes` pro `Auth0 API`

Po nastavení `Auth0 API` se lze skrze záložku `Applications` > `Applications` dostat do seznamu vytvořených aplikací, kde se nachází i aplikace pro vytvořené `API`. Po kliknutí na aplikaci proběhne přesměrování na nastavení dané aplikace pro `API`. V záložce `Settings` lze najít informace, které budou dále využity, viz obrázek 7.7.



Obrázek 7.7: Ukázka `Domain`, `Client ID` a `Client Secret` hodnot pro `API` aplikaci

Serverová část řešení 8

V této části bude rozebrána implementace serverové části řešení. Postupně budou probrány části jako návrh a realizace databáze, struktura a typ jednotlivých `API` přístupových bodů, bezpečnostní nastavení a konfigurace projektu.

Jak již bylo řečeno v části 5.3.1, jako nejvhodnější architektura pro `API` byla vybrána architektura `REST`. Pro implementaci `REST` architektury byl vybrán `Spring Boot` jako nejvhodnější framework [Boo24c].

`Spring Boot` je framework založený nad jazykem `Java`. Je vysoce konfigurovatelný a škálovatelný pro větší aplikace. Většina nastavení lze provést buď v konfiguračních souborech (`*.xml`, `*.properties`), nebo v samostatných třídách za pomoci anotací (`Spring Boot` před spuštěním analyzuje celý projekt a nastavení použije automaticky).

8.1 Rozvržení projektu

`Spring Boot` vynucuje pro své fungování určitou strukturu projektu, viz obrázek 8.1.

Obrázek 8.1: Struktura projektu `Spring Boot`

Z obrázku 8.1 lze vidět, že v kořenovém adresáři projektu jsou umístěny soubory `build.gradle` a `settings.gradle`. Soubor `build.gradle` slouží k definování využitých knihoven (závislostí), definování verze projektu a ostatních vlastností, viz Gradle dokumentace [Gra].

Kořenový adresář také dále obsahuje složku `Docker`, ve které jsou umístěny soubory pro vytvoření `Docker` kontejnerů z projektu, viz kapitola 8.6.

Poslední důležitou složkou kořenového adresáře je `src`, která obsahuje dvě podsložky - `main` a `test`. Složka `main` slouží k vývoji celé aplikace a jsou zde umístěny všechny zdrojové kódy a konfigurace pro funkcionalitu aplikace. Složka `test` slouží pro testování aplikace a jsou zde umístěny všechny zdrojové kódy a konfigurace pro testovací prostředí. Obě složky (`main` a `test`) mají stejnou strukturu a obsahují podsložky `java` a `resources`. Ve složce `java` jsou umístěny zdrojové kódy a ve složce `resources` se nachází konfigurační soubory, jako je například `application.properties` viz kapitola 8.2.

8.2 Konfigurace projektu

Konfigurace projektu psaném ve `Frameworku Spring Boot` je velice snadná a je provedena pouze pomocí souboru `application.properties`, který se nachází ve adresáři `resources`.

Zdrojový kód 8.1: Konfigurace projektu v souboru `application.properties`

```
1  spring.application.name=Server
2  server.address=127.0.0.1
3  server.port=9090
4  spring.datasource.url=jdbc:mariadb://localhost:3306/pingmaster
5  spring.datasource.username=root
6  spring.datasource.password=root
7  spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
8  spring.jpa.hibernate.ddl-auto=update
9  okta.oauth2.issuer=https://dev-5zwd5xrpvxrfowlf.us.auth0.com/
10 okta.oauth2.audience=https://pingmaster.cz
11 okta.oauth2.client-id=8UirlARfXRw6dJaEOhbKakybQYjrYzz2
12 okta.oauth2.client-secret=REALLY_SECRET
```

Na první řádce nastavení 8.1 lze vidět nastavení názvu celé aplikace. Druhý a třetí řádek slouží pro konfigurace adresy, kde bude aplikace umístěna a portu na kterém bude naslouchat.

Další řádky s prefixem `spring.datasource.*` definují vlastnosti databáze. Nastavením `spring.datasource.driver-class-name=org.mariadb.jdbc.Driver` je například aplikaci řečeno, že jako ovladač databáze bude využit `MariaDB`. Mož-

ností nastavení `spring.datasource.url` je nastavena adresa, na které se nachází `MariaDB` databáze, kterou bude aplikace využívat.

Poslední kategorií nastavení jsou nastavení s předponou `okta.oauth2.*`. Tato kategorie nastavení slouží pro konfiguraci nástroje `Auth0` pro serverovou aplikaci. Konkrétně položka konfigurace `okta.oauth2.issuer` představuje nastavení domény pro `API`, viz obrázek 7.7. Položka nastavení `okta.oauth2.issuer`, představuje `Audience`, viz kapitola 7, které je nastavováno pro `API` při jeho vytváření, viz obrázek 7.4. Poslední dvě položky nastavení představují `ClientID` a `Client Secret`, které jsou po založení `Auth0 API` umístěny v sekci `Applications` > `Applications` v nástroji Auth0, viz obrázek 7.7.

8.3 Databáze

Pro ukládání dat byla vybrána `SQL` databáze. Oproti `NoSQL` databázím jsou výhodnější pro své jednoduché dotazování nezanořených objektů (nezanořený objekt je řádka v tabulce, která obsahuje data pouze z jedné hlavní tabulky) [Tec24]. Další výhodou `SQL` databáze je, že je výrazně efektivnější při tvorbě složitějších dotazů než databáze `NoSQL`.

Pro výběr databáze byl proveden výzkum a analýza databází `Oracle`, `MariaDB`, `MySQL` a `PostgreSQL`. Jako nejvhodnější `SQL` databáze byla vybrána `MariaDB` pro její efektivitu a snadné využití [alt23]. Dále se jedná o `Open Source` projekt a tím pádem využití této databáze je zcela zdarma.

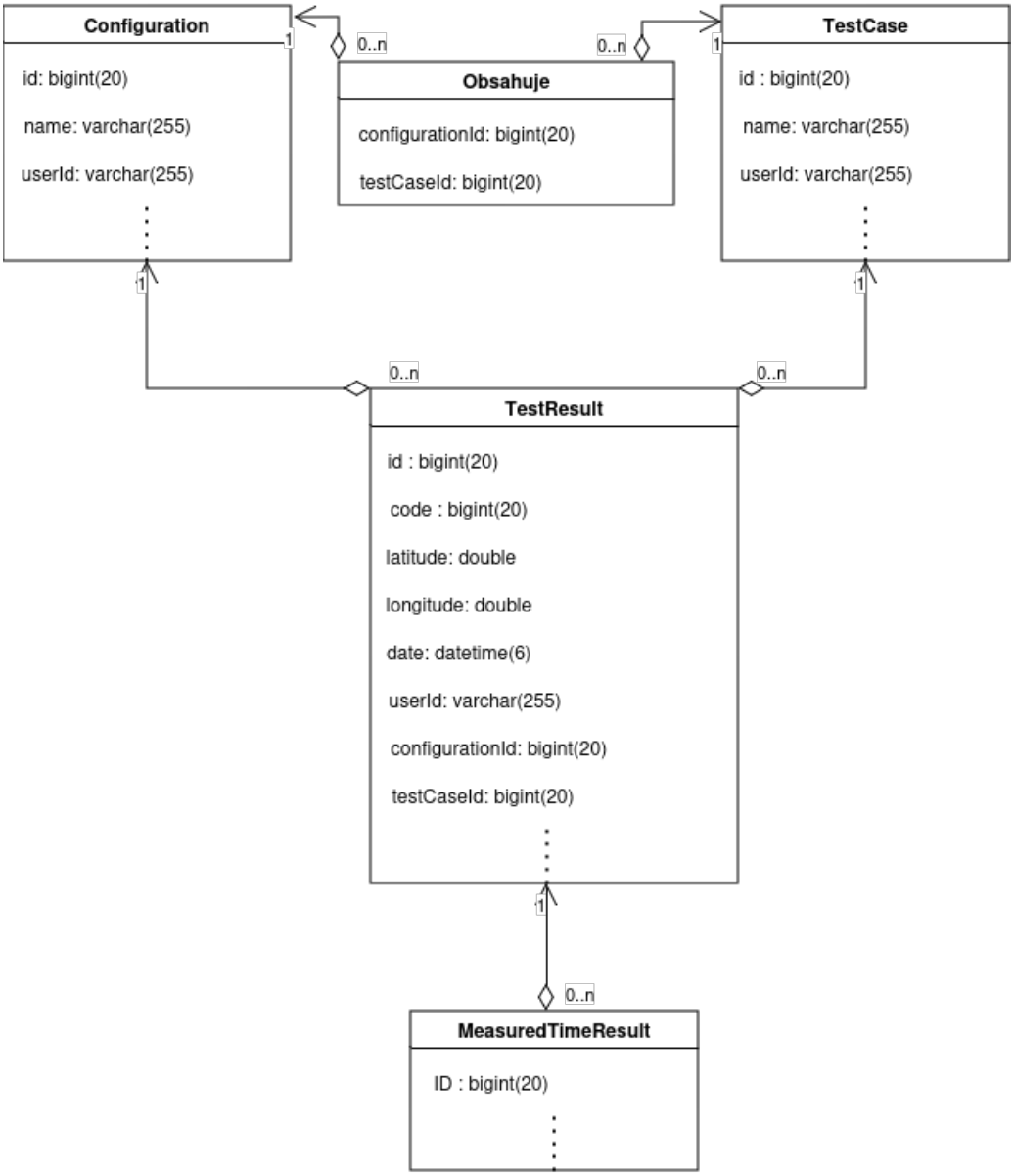
8.3.1 Návrh databázového schématu

Pro návrh databáze je třeba nejdříve definovat data, která budou v databázi uložena a jakým způsobem budou dotazována.

Databáze bude ukládat konfigurace pro jednotlivé protokoly, které budou využity pro měření (`SSH`, `Ping` a `HTTPS`) a k nim náležící výsledky měření. Obecné schéma `SQL` je znázorněno na obrázku 8.2.

Z obrázku 8.2 lze vidět, že většina tabulek obsahuje kromě klasického identifikátoru i ID vlastníka (`userid`). Tato položka je zde z důvodu, že většina záznamů, kromě měřených dat, které jsou přímo závislé na výsledků testu, může být reprezentována samostatně. V praxi to poté znamená, že testovací případ může být editován a je potřeba verifikace, zda patří opravdu uživateli, který se ho snaží upravit. Dalším využitím, které se tato položka poskytuje je, že lze filtrovat všechny záznamy pro daného uživatele a tak provést jednoduchou a efektivní filtraci v `SQL` tabulkách.

Pro návrh byla zvolena mezi tabulkami konfigurace (`Configuration`) a testovacího případu (`TestCase`) relace `M:N`. Tato relace umožní, že při spuštění testu nad jedním testovacím případem lze otestovat více konfigurací. Dále také přináší



Obrázek 8.2: Obecný návrh databáze pro ukládání konfigurace protokolu a výsledků měření

výhodu, že pro přidání konfigurace do testovacího případu není třeba konfiguraci znovu definovat, ale pouze se použije již vytvořená konfigurace.

Tabulka výsledků(`TestResult`) testů reprezentuje výsledek spuštěného testu pro danou konfiguraci. Výsledek obsahuje relaci `1:N` jak na konfiguraci, tak na testovací případ. V praxi to znamená, že výsledek testu lze filtrovat pro každou relaci mezi konfigurací a testovacím případem. Dále to také umožňuje snadnou filtraci výsledků pouze pro určitou konfiguraci, nebo pro testovací případ, např. pro srovnání výsledků jednotlivých konfigurací v testovacím případu. Tabulka dále obsahuje položky pro lokalizaci polohy pomocí zeměpisné šířky a délky, datum a čas kdy byl test uskutečněn a kód vrácený měřením.

Poslední tabulka měřená data (`MeasuredTimeResult`) obsahuje informaci o každé zaznamenané činnosti z měření. Tabulka obsahuje relaci `1:N` na výsledek testu (`TestResult`).

8.3.2 Realizace databáze

Databáze je realizována pomocí knihovny `JPA`. Knihovna `JPA` usnadňuje vytváření aplikací založených na platformě Spring, které využívají technologie přístupu k datům [Boo24a]. Dále umožňuje zjednodušenou tvorbu databázových dotazů pomocí syntaxe `JPQL`, které umožní maximálně využít vlastností jazyka `Java` [Boo24b].

V praxi může databázová entita reprezentovaná `Java` objektem vypadat viz kód 8.2 reprezentující konfiguraci pro `SSH` protokol.

Zdrojový kód 8.2: Reprezentace databázové entity pomocí `Java` třídy

```
1  @EqualsAndHashCode(callSuper = true)
2  @Data
3  @Entity
4  @Table(name = EntityColumnNameRegistry.SSH_CONFIGURATION_TABLE)
5  public class SshServerConfiguration extends MasterTable {
6
7      @Column(name = EntityColumnNameRegistry.NAME_COLUMN)
8      private String name;
9
10     @Column(name = EntityColumnNameRegistry.HOST_COLUMN)
11     private String host;
12
13     @OneToMany(fetch = FetchType.LAZY, mappedBy = "sshServerConfiguration",
       ↪  cascade = CascadeType.ALL)
14     private List<SshTestResult> sshTestResultList = new ArrayList<>();
15
16     @ManyToMany(fetch = FetchType.LAZY, cascade = CascadeType.PERSIST)
17     private List<SshTestCase> sshTestCaseList = new ArrayList<>();
```

```
18
19      @Column(name = EntityColumnNameRegistry.USER_COLUMN)
20      private String user;
21
22      @Column(name = EntityColumnNameRegistry.AUTHENTICATION_METHOD_COLUMN)
23      private String authenticationMethod;
24
25      @Column(name = EntityColumnNameRegistry.PORT_COLUMN)
26      private int port;
27
28      @Column(name = EntityColumnNameRegistry.LOCATION_COLUMN)
29      private String location;
30
31      @Column(name = EntityColumnNameRegistry.SFTP_MODE_COLUMN)
32      private String sftpMode;
33
34      @PreRemove
35      private void removeSshTestCaseList() {
36          this.sshTestCaseList.clear();
37      }
38
39      public void addSshTestResult(SshTestResult sshTestResult) {
40          sshTestResultList.add(sshTestResult);
41          sshTestResult.setSshServerConfiguration(this);
42      }
43  }
```

Z kódu 8.2 lze vidět, že třída musí být anotována jako `@Entity` a `@Table`. Pomocí těchto dvou anotací `Spring Boot` automaticky rozpozná, že se jedná o databázovou entitu a vytvoří příslušnou tabulku se sloupci vyžadovaných typů. Dále se v kódu 8.2 nachází anotace `@OneToMany` a `@ManyToMany`. Jak už název napovídá, tak anotace `@OneToMany` představuje relaci `1:N`, konkrétně obsahující seznam výsledků testů. Anotace `@ManyToMany` reprezentuje relaci `M:N`, která v tomto případě obsahuje seznam testovacích případů pro danou konfiguraci. Anotace `@OneToMany` i `@ManyToOne` definují parametr `fetch` jako `FetchType.LAZY`. Je to z toho důvodu, že `JPA` dokáže načítat data dynamicky až na jejich vyžádání. V praxi to tedy znamená, že po získání záznamu konfigurace z databáze se nenačtou i všechny jeho vnořené záznamy, ale jsou načteny až při jejich explicitním volání. To dokáže výrazně zlepšit efektivitu načítání z databáze.

Pro dotazování nad entitami jsou využity tři způsoby. Pro jednoduché dotazy je využita syntaxe `JPQL`, automatická tvorba dotazů poskytovaná knihovnou `JPA` a pro složitější tvorbu dotazů `Criteria Builder`, který je poskytován také knihov-

nou JPA. Criteria Builder umožňuje dynamickou tvorbu SQL dotazů, což není pomocí předchozích dvou postupů možné.

Automatická tvorba dotazů a využití JPQL je velice snadné. Pro vytvoření prostředí pro tyto dotazy stačí pouze jednoduché rozhraní s anotací @Repository, který je potomkem rozhraní CrudRepository, nebo Repository. Obvykle se využívá spíše CrudRepository, protože poskytuje již předdefinované funkce, jako například nalezení položky podle jejího ID, uložení položky a smazání položky. Deklarace celého řešení může vypadat viz kód 8.3.

Zdrojový kód 8.3: Deklarace třídy pro jednoduché dotazování v SQL

```
1 @Repository
2 public interface SshTestCaseDao extends CrudRepository<SshTestCase, Long> {
3     List<SshTestCase> findByUserId(String userId);
4
5     boolean existsByIdAndUserId(long id, String userId);
6
7     @Query("SELECT new cz.zcu.fav.kiv.bp.networktools_server.dto.ssh
      ↪  .SshTestCaseServerConfigurationRelationSync(tc.id, sc.id) FROM
      ↪  SshTestCase tc JOIN tc.sshServerConfigurationList sc WHERE
      ↪  tc.userId=:userId AND sc.userId =:userId")
8     List<SshTestCaseServerConfigurationRelationSync>
      ↪  findRelationListByUserId(String userId);
9 }
```

Pro definování databázových dotazů a získání dat z databáze není potřeba využití ničeho jiného kromě rozhraní. Jeho využití a volání bude rozebrané dále.

Z kódu 8.3 lze vidět, že rozhraní CrudRepository je generické. Jako první generický typ se uvádí typ záznamu, pro který je databázová struktura vytvářena. Druhým argumentem je typ identifikátoru daného objektu.

Rozhraní 8.3 obsahuje tři metody. První metoda s názvem findByUserId je automaticky zkompilována jako dotaz do databáze, viz kód 8.3.

Zdrojový kód 8.4: Převod názvu metody na SQL

```
1 SELECT * FROM SSH_TEST_CASE WHERE userId=?;
```

Při volání metody findByUserId je pomocí parametru userId předána hodnota do SQL dotazu (8.4) a vrácena databázová entita namapovaná na návratový typ metody.

Metoda findRelationListByUserId ze zdrojového kódu 8.3 slouží pro nalezení všech relací, mezi testovacím případem a konfigurací pro daného uživatele. Metoda je anotována anotací @Query, která jako hodnotu přijímá JPQL dotaz. Syntaxe JPQL spolu s knihovnou JPA dokáže převádět názvy Java tříd a polí na databázové

tabulky a sloupce. Ze syntaxe lze tedy vidět, že se dotazují všechny testovací případy pro protokol SSH společně s všemi jejich relacemi na konfiguraci pro protokol SSH za podmínky, že oba prvky patří uživateli předanému parametrem. Nakonec se výsledek namapuje do objektu SshTestCaseServerConfigurationRelationSync, a vrátí jako seznam instancí tohoto objektu.

Posledním využitým typem tvorby dotazů je takzvané Criteria Query a jím využívaný Criteria Builder. Tato metoda je vhodná ve Spring Bootu spíše pro tvorbu složitějších dotazů, kdy je například potřeba přidávat jednotlivé filtrace dynamicky, viz kód 8.5.

Zdrojový kód 8.5: Tvorba Criteria Query

```
1   private final EntityManager em;
2
3   @Autowired
4   public FilterDao(EntityManager em) {
5       this.em = em;
6   }
7
8   /**
9    * @param filter Input object which method use for filtering.
10   * @param userId Unique identification of current user.
11   * @param clazz  Class extending from Test Result to be filtered. According
     ↪  to this parameter can be called filtering in HttpTestResult,
     ↪  SshTestResult or PingTestResult.
12   * @param <A>    Template parameter extending from Nameable. Expecting
     ↪  configuration.
13   * @param <B>    Template parameter extending from Nameable. Expecting test
     ↪  case.
14   * @param <T>    Template parameter extending from TestResult with template
     ↪  arguments A and B.
15   * @return Filtered results from specific filter.
16   */
17  public <A extends Nameable, B extends Nameable, T extends TestResult<A, B>>
    ↪  List<T> filterBy(FilterDtoIn.TypeFilter filter, String userId, Class<T>
    ↪  clazz) {
18      CriteriaBuilder cb = em.getCriteriaBuilder();
19      CriteriaQuery<T> cq = cb.createQuery(clazz);
20      Root<T> root = cq.from(clazz);
21      List<Predicate> predicates = new ArrayList<>();
22
23      predicates.add(cb.equal(root.get("userId"), userId));
24      if (filter != null) {
```

```
25              if (filter.getLocation() != null) {
26                  predicates.addAll(createLocationFilter(root, cb,
                    ↪  filter.getLocation()));
27              }
28              if (filter.getCode() != null) {
29                  predicates.add(cb.equal(root.get("code"), filter.getCode()));
30              }
31              if (filter.getFrom() != null && filter.getTo() != null) {
32                  predicates.add(cb.between(root.get("date"), filter.getFrom(),
                    ↪  filter.getTo()));
33              } else if (filter.getFrom() != null) {
34                  predicates.add(cb.greaterThanOrEqualTo(root.get("date"),
                    ↪  filter.getFrom()));
35              } else if (filter.getTo() != null) {
36                  predicates.add(cb.lessThanOrEqualTo(root.get("date"),
                    ↪  filter.getTo()));
37              }
38              if (StringUtils.isNotBlank(filter.getConfigurationName())) {
39                  predicates.add(cb.equal(root.join("configuration").get("name"),
                    ↪  filter.getConfigurationName()));
40              }
41              if (StringUtils.isNotBlank(filter.getTestCaseName())) {
42                  predicates.add(cb.equal(root.join("testCase").get("name"),
                    ↪  filter.getTestCaseName()));
43              }
44          }
45          cq.select(root).where(predicates.toArray(Predicate[]::new));
46          return em.createQuery(cq).getResultList();
47      }
```

Z kódu 8.5 lze vidět tvorba komplexnějšího dotazu pomocí `CriteriaBuilder`. Konkrétně se v kódu nachází filtrace položek podle několika možných filtrů, z nichž žádný kromě `ID` uživatele není povinný. Pokud tedy není položka pro filtrování vyplněna, je nutné podle ní nefiltrovat, aby se nestalo, že bude filtrováno podle položky s hodnotou `null`.

8.4 Přístupové body pro REST

Tato kapitola se bude zabývat tvorbou přístupových bodů pro architekturu REST. Dále bude rozebrána funkcionalita využití přístupu k databázovým dotazům definovaných v kapitole 8.3.2 a využitých při tvorbě logiky pro jednotlivé přístupové body.

Z kódu v následujících sekcích se může často vyskytovat anotace `@Autowired`. Tato anotace slouží pro automatickou správu objektů a jejich auto-inicializaci [bae24b].

8.4.1 Controller třídy

Přístupové body ve frameworku `Spring Boot` se vytváří ve speciálních třídách zvaných `Controller`. `Controller` je třída anotovaná jako `@RestController` a slouží pro tvorbu přístupových bodů. `Spring Boot` automaticky vyhledává tyto třídy a dokáže z jejich obsahu vytvářet přístupové body pro `REST API`. Přístupový bod se následně vytvoří jako metoda, která, musí být anotována pomocí jedné z anotaci `@GetMapping`, @PostMapping, @PutMapping, @DeleteMapping nebo @PatchMapping. Tyto metody přímo souvisí s metodou využitou `REST` architekturou (`GET`, `POST`, `PUT`, `DELETE` a `PATCH`). Všechny vyjmenované anotace přijímají jako parametr `value`, která představuje relativní `URL` adresu daného přístupového bodu. Vstupní parametr metody představující výsledný přístupový bod může být libovolný `Java` objekt, který byl předán v těle požadavku, nebo informace o uživateli. Příklad implementace přístupového bodu lze vidět v kódu 8.6.

Zdrojový kód 8.6: Implementace `Controlleru` ve `Spring Boot`

```
1  @RestController
2  @RequestMapping("/filter")
3  public class FilterController {
4
5      private final FilterService filterService;
6
7
8      @Autowired
9      public FilterController(FilterService filterService) {
10         this.filterService = filterService;
11     }
12
13
14     @PostMapping("/")
15     public FilterDtoOut doFilter(@RequestBody FilterDtoIn dtoIn, Authentication
       ↪  authentication) {
16         return filterService.doFilter(dtoIn, authentication.getName());
17     }
18 }
```

Z kódu 8.6 lze vidět, že je třída anotovaná kromě anotace `@RestController` také anotací `@RequestMapping` s parametrem `'/filter'`. Tato hodnota slouží jako prefix pro všechny definované přístupové body. Metoda `doFilter` z kódu 8.6 ano-

tovaná jako `@PostMapping` bude tedy poskytovat přístupový bod metodou `POST` na relativní `URL` adrese `/filter/`.

Dále lze z kódu 8.6 vidět, že metoda `doFilter` přijímá dva argumenty. První argument je tělo požadavku, které `Spring Boot` automaticky konvertuje z `JSON` formátu do `Java` objektu. Druhým argumentem metody `doFilter` je `Authentication` objekt, který obsahuje všechny důležité informace k identifikaci uživatele.

8.4.2 Service třídy

Kód je rozdělen tak, aby `Controller` třídy neobsahovaly přebytečnou logiku programu a byly přehledné. Veškerá logika programu byla tedy přesunuta do tříd zvaných `Service`.

`Service` třída je třída anotovaná pomocí anotace `@Service`. Tato třída slouží pro tvorbu logiky programu, včetně práce s databází. Jednoduchý příklad `Service` třídy tedy může vypadat viz kód 8.7.

Zdrojový kód 8.7: Implementace `Service` ve `Spring Boot`

```
1  @Service
2  public class FilterService {
3
4      private final FilterDao filterDao;
5
6      @Autowired
7      public FilterService(FilterDao filterDao) {
8          this.filterDao = filterDao;
9      }
10
11     /**
12      * Method applying set of filters to database and return achieved data.
13      * Filters should not be in collisions to avoid returned data collisions.
14      * For example if trying to fetch two PINGs test results and set date filter
        ↪  colliding, redundant data can be returned.
15      *
16      * @param dtoIn  Object representing criteria for filtering.
17      * @param userId The user who accesses the data.
18      * @return Filtered results.
19      */
20     public FilterDtoOut doFilter(FilterDtoIn dtoIn, String userId) {
21         if (dtoIn == null) {
22             return new FilterDtoOut();
23         }
24         FilterDtoOut dtoOut = new FilterDtoOut();
```

```
25
26          for (FilterDtoIn.TypeFilter typeFilter : dtoIn) {
27              if (typeFilter.getFilterType() != null) {
28                  if (typeFilter.getFilterType() == FilterType.SFTP) {
29                      if (dtoOut.getSsh() == null) {
30                          dtoOut.setSsh(new ArrayList<>());
31                      }
32
                        ↪  dtoOut.getSsh().addAll(convertSftp(filterDao.filterBy(typeFilter,
                        ↪  userId, SshTestResult.class)));
33                  } else if (typeFilter.getFilterType() == FilterType.PING) {
34                      if (dtoOut.getPing() == null) {
35                          dtoOut.setPing(new ArrayList<>());
36                      }
37
                        ↪  dtoOut.getPing().addAll(convertPing(filterDao.filterBy(typeFilter,
                        ↪  userId, PingTestResult.class)));
38                  } else if (typeFilter.getFilterType() == FilterType.HTTP) {
39                      if (dtoOut.getHttp() == null) {
40                          dtoOut.setHttp(new ArrayList<>());
41                      }
42
                        ↪  dtoOut.getHttp().addAll(convertHttp(filterDao.filterBy(typeFilter,
                        ↪  userId, HttpTestResult.class)));
43                  }
44              }
45          }
46          return dtoOut;
47      }
48  }
```

Z kódu 8.7 lze vidět deklaraci `Service` třídy. Třída obsahuje proměnou `filterDao`, která je pomocí anotace `@Autowired` automaticky doplněna do konstruktoru a slouží pro operace nad databází (8.5).

8.5 Zabezpečení

Zabezpečení jednotlivých přístupových bodů je velice důležitým aspektem hlavně pro citlivá data, jako jsou například `SSH` konfigurace. `Spring Boot` dokáže provést autentizaci uživatele jen za pomoci konfigurace ze sekce 8.2. Pro dodatečnou ochranu dat je důležité zavést k autentizaci i autorizaci, která slouží pro povolení přístupu k jednotlivým přístupovým bodům na základě udělených práv [Sha21]. K tomuto účelu se využívají `Scopes` (7), které jsou alternativou rolí. `Scopes` jsou vyžá-

dány klientem při založení spojení s Auth0 a vyžádání tokenu. Spring Boot při kontrole přístupového tokenu za pomoci nástroje Auth0 pošle i vyžadované Scopes pro daný přístupový bod. Pokud přístupový token neobsahuje dané Scopes, je navrácen návratový stav 403 (Forbidden), který obvykle značí, že uživatel nemá dostatečná práva pro využití daného přístupového bodu. Při chybě autentizace, která může být zapříčiněna například nevalidním nebo chybějícím tokenem je vrácen návratový stav 401 (Unauthorized). V praxi deklarace autorizace pro přístupové body může vypadat viz kód 8.8.

Zdrojový kód 8.8: Implementace autorizace ve Spring Boot

```
1  @Configuration
2  @EnableWebSecurity
3  public class SecurityConfiguration {
4
5      /**
6       * Configure Security filter chain for access limitation to program access
       ↪  points.
7       * @param http Http object to be configured.
8       * @return Configured Security filtered chain.
9       * @throws Exception Exception should not be thrown.
10      */
11     @Bean
12     public SecurityFilterChain configure(HttpSecurity http) throws Exception {
13         return http.authorizeHttpRequests(authorize -> authorize
           ↪  .requestMatchers("/sshAction/**").access(hasScope("pingmaster"))
           ↪  .requestMatchers("/sshSynchronization/**").access(hasScope("pingmaster"))
           ↪  .requestMatchers("/pingSynchronization/**").access(hasScope("pingmaster"))
           ↪  .requestMatchers("/pingAction/**").access(hasScope("pingmaster"))
           ↪  .requestMatchers("/httpAction/**").access(hasScope("pingmaster"))
           ↪  .requestMatchers("/httpSynchronization/**").access(hasScope("pingmaster"))
           ↪  .requestMatchers("/filter/**").access(hasScope("read:data"))
           ↪  .anyRequest().authenticated()).oauth2ResourceServer(oauth2ResourceServer
           ↪  -> oauth2ResourceServer.jwt(withDefaults())).build();
14     }
15
16 }
```

Z kódu 8.8 lze vidět, že většina přístupových bodů vyžaduje Scope zvaný pingmaster. Tento Scope slouží pro komunikaci s Android aplikací a neměl by být udělen nikomu jinému než Android aplikaci.

Dále se v kódu 8.8 vyskytuje Scope zvaný read:data. Tento Scope slouží pouze pro přístup k přístupovému bodu s relativní adresou /filter/. Přístupový bod

slouží pouze pro filtrování naměřených výsledků a proto může být Scope `read:data` udělen i uživatelům třetích stran.

8.6 Kontejnerizace projektu

Posledním krokem implementace byla kontejnerizace projektu pro jeho snadné spuštění. Kontejnerizace aplikace je vytvořena pro nástroj Docker. Docker je otevřená platforma pro vývoj, nasazení a spouštění aplikací [docb].

Výhoda tohoto přístupu je, že není potřeba explicitně nastavovat `MariaDB` databázi pro ukládání dat a poté přepisovat `URL` adresu k této databázi, ale nástroj `Docker` dokáže tyto závislosti vyřešit sám.

Pro konfiguraci `Dockeru` byly vytvořeny v adresáři `docker` dva soubory. První soubor se nazývá `Dockerfile` a slouží pro vytvoření `Docker Image` z aplikace. `Docker Image` slouží jako sada pokynů k sestavení kontejneru `Docker`, podobně jako šablona [tec].

Druhým souborem je soubor s názvem `docker-compose.yml`. Tento soubor slouží pro tvorbu `Docker` kontejneru. Kontejner je standardní softwarová jednotka, která zahrnuje kód a všechny jeho závislosti tak, aby aplikace rychle a spolehlivě fungovala a byla snadno spustitelná [doca]. Struktura souboru `docker-compose.yml` je popsána viz kód 8.9.

Zdrojový kód 8.9: Nastavení kontejneru pro `Docker`

```
1  version: '3.7'
2  services:
3    pingmaster:
4      container_name: pingmaster-app
5      build:
6        dockerfile: Dockerfile
7      depends_on:
8        - maria_db
9      ports:
10       - "5006:5006"
11     environment:
12       SERVER_PORT: 5006
13       SPRING_DATASOURCE_URL: jdbc:mariadb://maria_db:3306/pingmaster
14       SPRING_DATASOURCE_USERNAME: pingmaster
15       SPRING_DATASOURCE_PASSWORD: pingmaster
16   maria_db:
17     container_name: pingmaster-app-database
18     restart: always
19     image: mariadb:latest
20     ports:
```

```
21        - "3308:3006"
22      environment:
23        MARIADB_ROOT_PASSWORD: root
24        MARIADB_DATABASE: pingmaster
25        MARIADB_USER: pingmaster
26        MARIADB_PASSWORD: pingmaster
27      volumes:
28        - ./schema.sql:/docker-entrypoint-initdb.d/1.sql
29        - maria_db:/var/lib/mysql
30  volumes:
31    maria_db:
32
```

Z kódu 8.9 lze vidět, že jsou vytvářeny dva kontejnery. Jeden kontejner s názvem `pingmaster` obsahuje veškerý kód vyvíjené aplikace. Druhý kontejner má název `maria_db` a reprezentuje databázi `MariaDB`.

Důležitými parametry pro oba kontejnery jsou `ports`, které nastavují port na kterém bude aplikace naslouchat. První částí řetězce `ports` je port na kterém bude aplikace naslouchat na hostitelském zařízení. Druhá část potom značí port, na kterém aplikace naslouchá v rámci kontejneru.

Dalším důležitým parametrem je `environment` a jeho položky. Tyto položky nastavují proměnné prostředí a pro vyvíjenou aplikaci jsou přetížením proměnných nastavených v projektové konfiguraci (8.2).

Při sestavení aplikace je tedy z vyvíjené aplikace vytvořen automaticky `Docker Image`. Dále se stáhne databáze `MariaDB` z Docker Hubu a oba kontejnery se nastartují.

Sestavení a nastartování celé aplikace i s nastavením databáze se následně provede pouze příkazy viz výpis 8.10.

Zdrojový kód 8.10: Nastartování `Docker` kontejneru

```
1  cd docker
2  docker compose up
```

8.7 Testování

Za účelem ověření správné funkcionality programu je část aplikace testována pomocí jednotkových testů. Pro testování byla vybrána část aplikace, která zajišťuje složité filtrování v databázi a poskytuje uživateli data o dosažených výsledcích (8.5).

Pro jednotkové testy je důležitou vlastností jejich izolované prostředí, to znamená, že se jednotlivé testy nesmí ovlivňovat a nesmí být ovlivněny ani produkční data.

Pro zachování produkčních dat je využito databáze H2 [bae24a], která je velice vhodná pro testování. Hlavním důvodem jejího využití je, že databáze nepotřebuje pro své správné fungování databázový server jako většina ostatních SQL databází, ale je inicializována pouze v paměti.

Celkově bylo vytvořeno třináct jednotkových testů, kterými byla snaha pokrýt všechny možnosti vstupů filtrování, případně jejich kombinací. Příklad testované metody je popsán viz kód 8.11.

Zdrojový kód 8.11: Implementace metody pro realizaci testu ve Spring Boot

```
1   /**
2    * Test filtering of Ping for specific user according to date filters.
     ↪  Filter is set exactly from 15.4.2024 14:00:00 to 15.4.2024 14:34:00.
3    */
4   @Test
5   @Sql(executionPhase = Sql.ExecutionPhase.BEFORE_TEST_METHOD, scripts =
    ↪  "classpath:init_data.sql")
6   public void testFilterFromTo() {
7       FilterDtoIn.TypeFilter typeFilter = FilterDtoIn.TypeFilter.builder()
        ↪  .from(getLimiterDate(0)).to(getLimiterDate(34))
        ↪  .filterType(FilterType.PING).build();
8       FilterDtoIn dtoIn = new FilterDtoIn(typeFilter);
9       FilterDtoOut out = filterService.doFilter(dtoIn, DEFAULT_USER);
10      assertNotNull(out);
11      assertNotNull(out.getPing());
12      assertNull(out.getSsh());
13      assertEquals(8, out.getPing().size());
14  }
```

Z kódu 8.11 lze vidět, že metoda je kromě anotace `@Test`, která slouží pro deklaraci testu, anotována také pomocí anotace `@Sql`. Anotace `@Sql` definuje, že před spuštěním testu se má provést SQL skript, který se nachází v `resources` a nazývá se `init_data.sql`.

Zbytek serverové aplikace slouží převážně ke komunikaci s Android aplikací a proto je testován až pomocí uživatelského testování v aplikaci Android.

Modul pro měření kvality připojení

9

Tato část se zabývá implementací modulu pro platformu Android, který poskytuje rozhraní pro měření kvality připojení. Modul je kolekce zdrojových souborů a nastavení, které umožňují rozdělit projekt na samostatné jednotky funkcí. Projekt může mít jeden nebo více modulů a jeden modul může používat jiný modul jako závislost. Každý modul lze nezávisle sestavovat, testovat a ladit [dev24e].

Implementovaný modul obsahuje funkcionality potřebné pro měření kvality připojení. Konkrétně obsahuje měření pomocí protokolu `HTTP(s)`, `SFTP` a nástroje `Ping`.

9.1 Sestavení knihoven

Pro měření pomocí protokolu `HTTP(s)` a `SFTP` byly vybrány knihovny psané v jazyce `C/C++`. Knihovny nejsou určené primárně pro platformu Android a proto byla potřeba je zkompilovat pro všechna Android `ABI` (Application Binary Interface). Android `ABI` poskytuje instrukční sadu zařízení, zahrnuje konvenci pro předávání dat mezi aplikacemi a systémem a způsob, jakým systém používá zásobník a registry při volání funkcí [dev24a].

Kromě knihoven `libssh-2` pro protokol `SSH` a `libCurl` pro `HTTP(s)` byla potřeba využít ještě knihovna `WolfSSL`. Tato knihovna je potřebná pro obě výše zmíněné knihovny, protože poskytuje kryptografické funkce, které obě knihovny vyžadují.

Knihovny jsou buď sestaveny již v modulu, nebo pro jejich sestavení byl vytvořen skript `build.sh`. Skript se nachází v adresáři `/libs`. Jeho testování a kompilace knihoven probíhala na prostředí `Ubuntu 23.04` s verzí `NDK` (`Native Development Kit`) `26.2.11394342`.

9.2 Měření času

Pro co nejpřesnější měření času a vyhodnocení výsledků byly zváženy dvě hlavní možnosti. První možností bylo měření času pomocí tiků procesoru. Tato metoda měření je velice přesná, protože poskytnutí tiku procesoru je téměř okamžité. V dnešní době bohužel není jednoduché tuto metodu efektivně využít, protože novější procesory nemusí mít konstantní délku tiku [Plo18]. Druhou možností bylo využití standardních hodin poskytnutých jazykem C++. Konkrétně se jedná o třídu `std::chrono::steady_clock`, nebo `std::chrono::system_clock`. Z těchto dvou možností byla nakonec vybrána jako lepší možnost `std::chrono::steady_clock`, protože poskytuje absolutní čas. To znamená, že výsledky nemohou být ovlivněny například změnou systémových hodin. Pro zachování co nejvyšší přesnosti měření jsou všechny časové výsledky ukládány v nanosekundách.

9.3 Měření pomocí protokolu SFTP

Měření rychlosti stahování a nahrávání byly realizovány pomocí knihovny `libssh-2`. Uplynulý čas a velikost přeneseného bufferu jsou uloženy vždy hned po provedení dané transakce a nejsou nijak redukovány.

Zdrojový kód 9.1: Měření rychlosti stahování/nahrávání s využitím protokolu SFTP

```
1            // MEASUREMENT
2            if (mode == SftpMode::SFTP_DOWNLOAD) {
3                start = std::chrono::high_resolution_clock::now(); //
                 ↪  MEASUREMENT START
4                returnCode = libssh2_sftp_read(sftpHandle, buffer, bufferSize);
5                end = std::chrono::high_resolution_clock::now(); // MEASUREMENT
                 ↪  END
6            } else {
7                for (auto i = 0; i < bufferSize; i++) {
8                    buffer[i] = (rand() % 26) + 'A';
9                }
10               start = std::chrono::high_resolution_clock::now(); //
                 ↪  MEASUREMENT START
11               returnCode = libssh2_sftp_write(sftpHandle, buffer, bufferSize);
12               end = std::chrono::high_resolution_clock::now(); // MEASUREMENT
                 ↪  END
13           }
```

Z kódu 9.1 lze vidět postup při měření rychlosti stahování/nahrávání. Operace zápisu/čtení bufferu je vždy ohraničena proměnnými `start` a end, do kterých je přiřazen

aktuální čas v nanosekundách. Z rozdílu těchto proměnných lze poté vypočítat čas potřebný pro přenesení bufferu. Skutečná velikost přeneseného bufferu je vrácena metodou `libssh2_sftp_write` pro zápis dat, nebo `libssh2_sftp_read` pro čtení dat.

Údaje o velikosti bufferu a rychlosti čtení/zápisu jsou přidány do seznamu, který je bez dalších modifikací vrácen jako výsledek dané funkce.

9.4 Měření pomocí protokolu HTTP(s)

Měření pomocí protokolu HTTP(s) probíhá odlišnou metodou než měření pomocí SFTP. Knihovna `libcurl` poskytuje za běhu informace o přenosu, jako je čas, kdy započala transakce, nebo čas, kdy byl přijat první bit. Z těchto informací lze velice přesně změřit aktuální rychlost stahování, viz kód 9.2.

Zdrojový kód 9.2: Měření rychlosti stahování s využitím protokolu HTTP(s)

```
1  /**
2   * Callback called by libcurl with inner mechanism. Calling intensity of this
    ↪  method is affected by download/upload speed.
3   * @param clientp Pointer to object passed to callback by curl
    ↪  CURLOPT_XFERINFODATA option.
4   * @param dltotal Predicted size to be downloaded (not used).
5   * @param dlnow Total amount of bytes currently downloaded.
6   * @param ultotal Predicted size to be uploaded (not used).
7   * @param ulnow Total amount of bytes currently uploaded (not used).
8   * @return Zero if everything is OK. Non-Zero value will abort connection.
9   */
10 int progressCallback(void *clientp, curl_off_t dltotal, curl_off_t dlnow,
   ↪  curl_off_t ultotal, curl_off_t ulnow) {
11     auto progress = reinterpret_cast<networktools::CallbackStruct *> (clientp);
12     curl_off_t postTransferMicros, postTransferNanos;
13     curl_off_t preTransferMicros, preTransferNanos;
14     curl_off_t iterationElapsedNanos;
15     if (progress->firstIterationNanos > progress->callbackDefinitions.endNanos)
       ↪ { // End if time elapsed
16         return 1;
17     }
18     if (dlnow > 0 && progress->lastIterationBytes != dlnow) {
19         if (curl_easy_getinfo(progress->curl,
           ↪  CURLINFO::CURLINFO_STARTTRANSFER_TIME_T, &postTransferMicros)) {
20             return 1;
21         } else if (curl_easy_getinfo(progress->curl,
           ↪  CURLINFO::CURLINFO_PRETRANSFER_TIME_T, &preTransferMicros)) {
```

```
22              return 1;
23          }
24          postTransferNanos = postTransferMicros * 1000LL;
25          preTransferNanos = preTransferMicros * 1000LL;
26          iterationElapsedNanos = postTransferNanos - preTransferNanos;
27          auto bytes = dlnow - progress->lastIterationBytes;
28          progress->results.emplace_back(iterationElapsedNanos, bytes);
29
30          progress->lastIterationBytes = dlnow;
31      }
32      return 0;
33  }
```

Z kódu 9.2 lze vidět výpočet velikosti aktuálně přenášeného bufferu a času potřebného pro jeho přenos. Konkrétně pro výpočet přenesené velikosti bufferu je potřeba odečíst celkový počet přenesených bytů zaznamenaný předchozí iterací od celkového počtu přenesených bytů v aktuální iteraci.

9.5 Konverze výsledků z jazyka C++ do jazyka Java

Pro výsledky měření bylo nutné převést kód z jazyka C++, kde bylo měření prováděno do jazyka Java. K tomuto účelu bylo využito rozhraní JNI (Java Native Interface). Volání nativní metody v jazyce Java tedy vypadá viz kód 9.3.

Zdrojový kód 9.3: Volání nativní funkce v jazyce Java

```
1       static {
2           System.loadLibrary("networktools");
3       }
4
5       /**
6        * Call of native C++ method for HTTP Download.
7        *
8        * @param dtoIn      Input configuration for testing.
9        * @param maxTime    Maximum time which can be elapsed for test in seconds.
10       * @param bufferSize Recommended buffer size for testing. This value can be
        ↪  ignored by system.
11       * @return Measured data.
12       */
13      private static native JNIHttpDownloadDtoOut httpDownload(HttpDtoIn dtoIn,
        ↪  double maxTime, int bufferSize);
```

Z kódu 9.3 jde vidět, že je potřeba načíst nativní knihovnu pomocí systémové metody `loadLibrary`. Pro deklaraci nativní metody v jazyce `Java` je poté důležité klíčové slovo `native`, které překladači říká, že se jedná o metodu přetíženou z nativního jazyka. Typ návratové hodnoty a vstupní data mohou být libovolné, ale musí se shodovat s deklarací vstupních parametrů a typu návratové hodnoty funkce v jazyce C++, viz kód 9.4.

Zdrojový kód 9.4: Deklarace nativní funkce v jazyce C++

```
1  extern "C"
2  JNIEXPORT jobject JNICALL
3  Java_cz_zcu_fav_kiv_bp_networktools_lib_utils_HttpUtils_httpDownload(JNIEnv
   ↪  *env, jclass thiz, jobject dto, jdouble maxTime, jint bufferSize) {
4      HttpDto inDto = networktools_convertor::fromJHTTPDto(env, dto);
5      try {
6          auto httpResult = networktools::http(inDto, maxTime, bufferSize);
7          auto jHttpResult = networktools_convertor::toJListOf<TimeResult>(env,
           ↪  httpResult, networktools_convertor::toJTimeResult);
8          auto dtoOut = networktools_convertor::toJHttpDownloadDtoOut(env,
           ↪  jHttpResult, HttpError::OK, CURLE_OK);
9          return dtoOut;
10     } catch (HttpException &exception) {
11         auto dtoOut = networktools_convertor::toJHttpDownloadDtoOut(env,
           ↪  nullptr, exception.getError(), exception.getCode());
12         return dtoOut;
13     }
14 }
```

Z kódu 9.4 lze vidět, že název nativní metody je uveden jako `Java_cesta_ke_tride_metoda`. Tato konvence pro název je povinná. Zároveň se shodují i typy předané v kódu 9.3. Zatímco konvertování jednoduchých typů, jako je `jdouble`, nebo `jint` zvládne jazyk C++ udělat implicitně, tak složitější objekty, obvykle předávané jako `jobject` musí být transformovány ručně, viz kód 9.5.

Zdrojový kód 9.5: Konvertování parametru `jobject` do C++ objektu

```
1  HttpDto networktools_convertor::fromJHTTPDto(JNIEnv *env, jobject object) {
2      HttpDto dto;
3      if (object == nullptr) {
4          return {};
5      }
6      jclass clazz = env->GetObjectClass(object);
7      jmethodID getUrl = env->GetMethodID(clazz, "url", "()Ljava/lang/String;");
8
```

```
9       if (getUrl == nullptr) {
10          __android_log_print(ANDROID_LOG_FATAL, "JNI",
11                              "Methods for %s are not available. Application will
                                ↪  be terminated.",
12                              getClassName(env, clazz).c_str());
13          std::terminate();
14      }
15      auto url = (jstring) env->CallObjectMethod(object, getUrl);
16      dto.setUrl(toCppString(env, url));
17      return dto;
18  }
```

Z kódu 9.5 je vidět, že pomocí třídy `JNIEnv` jsou volány metody z jazyka `Java`, pomocí kterých se lze dostat k primitivním typům a ty následně využít v jazyce `C++`.

9.6 Měření pomocí nástroje `Ping`

Pro měření pomocí nástroje `Ping` je využita knihovna `Icmp4a`. Při importování knihovny do modulu není knihovna dostupná aplikacím, které tento modul využívají. Z toho důvodu bylo vytvořeno rozhraní, které slouží k volání metod knihovny `Icmp4a`. Volání knihovní metody pro provedení měření pomocí nástroje `Ping` tedy vypadá viz kód 9.6.

Zdrojový kód 9.6: Měření rychlosti stahování s využitím nástroje `Ping`

```
1               icmp.pingInterval(dtoIn.host, dtoIn.count, dtoIn.timeoutMillis,
                ↪  dtoIn.packetSize, dtoIn.intervalMillis, dtoIn.network).onEach {
                ↪  status ->
2                   if (status.result is Icmp.PingResult.Success) {
3                       val success = status.result as Icmp.PingResult.Success
4                       pingListener.onItemGet(PingResult(success.sequenceNumber,
                        ↪  success.sequenceNumber, success.ms))
5                       if (dtoOut == null) {
6                           dtoOut = PingDtoOut()
7                       }
8                       dtoOut = toDtoOut(status, dtoOut!!)
9                   } else {
10                      throw Icmp.Error.UnknownHost("Cannot access remote
                        ↪  endpoint")
11                  }
12              }.catch {
13                  if (it !is CancellationException) {
14                      pingListener.onError(
15                          when (it) {
```

```
16                          is Icmp.Error.UnknownHost -> PING_UNKNOWN_HOST_ERROR
17                          is Icmp.Error.SocketException -> PING_SOCKET_ERROR
18                          is Icmp.Error.ProtocolException ->
                            ↪  PING_PROTOCOL_ERROR
19                          else -> PING_UNKNOWN_ERROR
20                      }
21                  )
22              }
23          }.onCompletion {
24              if ((it == null || it is CancellationException) && dtoOut !=
                ↪  null) {
25                  pingListener.onComplete(dtoOut!!)
26              }
27          }.launchIn(CoroutineScope(Dispatchers.IO))
```

Z kódu 9.6 lze vidět, že měření je asynchronní. Na specifické události, jako dokončení jedné iterace, dokončení celého procesu, nebo přijetí výjimky jsou volány akce pomocí rozhraní `PingListener`, které je předáno v parametru.

9.7 Testování

Pro každé měření byl připraven set testů, pomocí kterých lze otestovat funkcionalitu jednotlivých měření a případně i jejich chybové návratové stavy. Testovací sady jsou nastaveny ve složce `androidTest`. Obyčejné jednotkové testy nedokáží pracovat s nativním jazykem a proto jsou všechny testy spustitelné pouze pod platformou Android.

Zdrojový kód 9.7: Ukázka testování modulu

```
1   @Test
2   public void runDownloadHappyDay() {
3       HttpDtoIn dtoIn = new HttpDtoIn("http://185.215.166.217:9091/gradlew");
4       HttpDtoOut dtoOut = HttpUtils.curlDownload(dtoIn, 10.0, 64 * 1024);
5       assertNotNull(dtoOut);
6       assertNotNull(dtoOut.getTimeResults());
7       assertEquals(0, dtoOut.getCode());
8       assertEquals(0, dtoOut.getCurlCode());
9       assertFalse(dtoOut.getTimeResults().isEmpty());
10  }
```

Z kódu 9.7 lze vidět jednoduchý příklad testování stahování souboru pro protokol `HTTP(s)`. Kvůli využití reálné sítě není možné kontrolovat výsledky rychlosti stahování. Také se může stát, že některé testy neuspějí z důvodu nepředvídatelnosti

reálné sítě. Při neúspěchu testu je tedy vhodné ho spustit znovu separovaně. Pro ověření funkcionality je tedy kontrolován pouze návratový stav a zda není navrácený seznam výsledků prázdný.

9.8 Využití modulu v Android projektu

Pro využití modulu v jiném Android projektu je nutné zkopírovat celý modul do kořenového adresáře cílového projektu. Závislost na modulu se následně přidá do souboru `build.gradle.kts` nebo `build.gradle` viz kód 9.8.

Zdrojový kód 9.8: Ukázka přidání závislosti na modulu

```
1 dependencies {
2     implementation(project(mapOf("path" to ":NetworkTools")))
3     ...
4     ...
5 }
```

Z kódu 9.8 lze vidět přidání závislosti na modulu pro měření kvality připojení. Po přidání této závislosti lze v cílové aplikaci plně využívat všech funkcionalit modulu.

Implementace Android aplikace

10

Tato kapitola se bude zabývat implementací aplikace pro platformu Android. Postupně budou probrány řešené problémy, jako například ukládání hesel pro SSH konfigurace, využití databáze a architektura jejího návrhu, přihlášení, synchronizace dat a výměna tokenů pro autorizaci na vzdáleném serveru.

10.1 Databáze

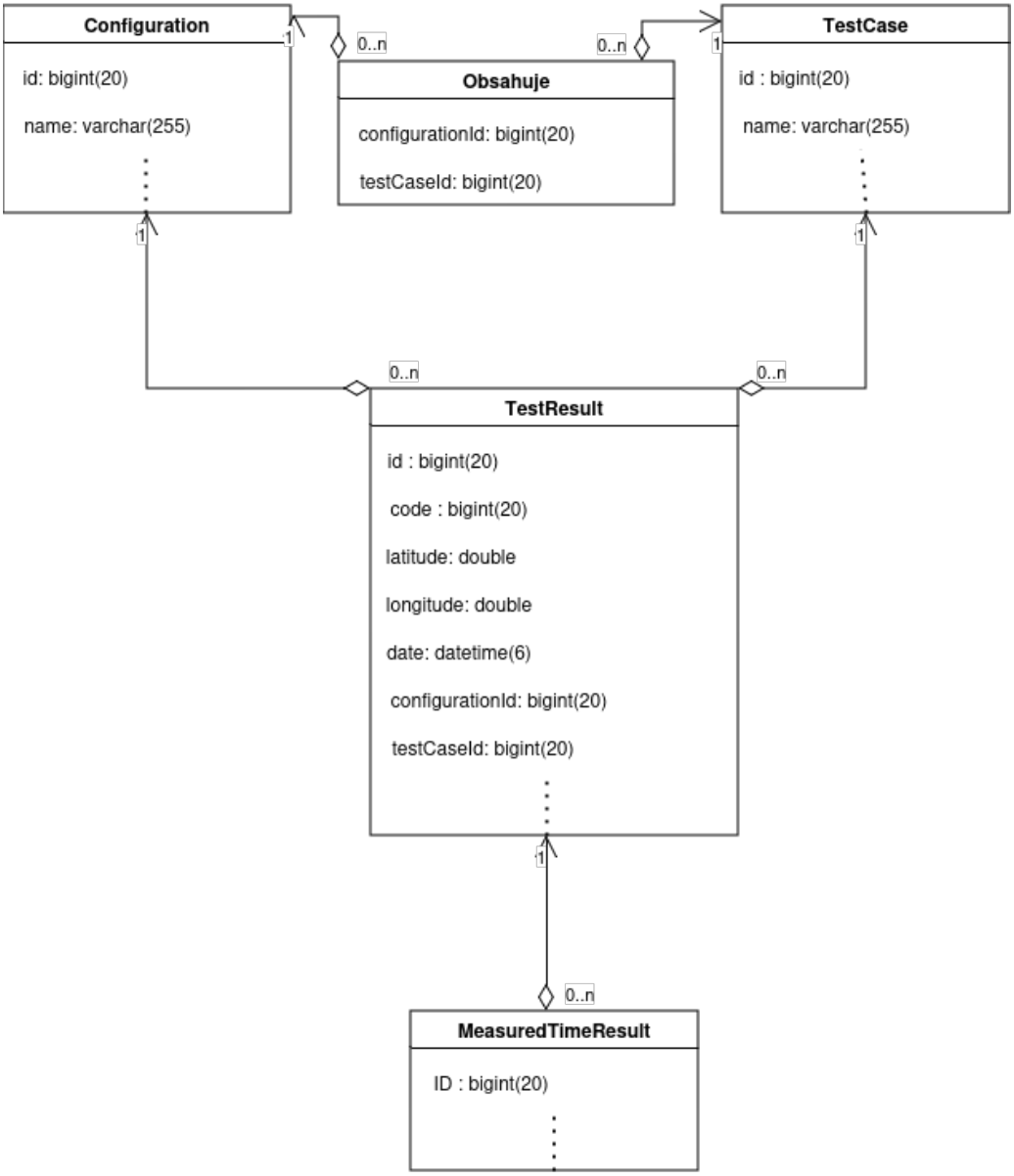
Tato kapitola bude rozebírat vytvoření databáze, její architekturu a využití na platformě Android.

10.1.1 Návrh databáze

Databáze je navržena tak, aby se co nejvíce podobala schématu databáze pro serverové řešení implementované v kapitole 8.3, především z důvodu jednodušší výměny dat. Největšími rozdíly obsahu navržené databáze je to, že databázové tabulky v zařízení Android neobsahují sloupec `userId`, protože jsou realizovány pouze pro jednoho uživatele. Dále jsou v tabulce pro SSH konfigurace přidány sloupce `auth` a `iv`, které slouží pro ukládání zašifrovaného hesla a inicializačního vektoru rozebíraného v kapitole 10.2. Obecný návrh databáze tedy vypadá viz obrázek 10.1.

Z obrázku 10.1 lze vidět obecnou strukturu databázových tabulek a jejich relací. U tabulek jsou uvedeny vždy jen společné vlastnosti, které tabulky obsahují.

Struktura 10.1 je v databázi reprezentována celkem třikrát pro každý typ měřícího protokolu nebo nástroje (SSH, `Ping`, `HTTP(s)`). Tabulky jsou replikovány hlavně z důvodu, že každá z tabulek obsahuje jiné informace. Například tabulka pro konfiguraci SSH obsahuje informace o adrese hostitelského zařízení, uživatele, heslo a port. Naproti tomu pro testování pomocí `HTTP(s)` je uváděna pouze adresa URL, která je pro testování využita.



Obrázek 10.1: Obecný návrh databáze pro ukládání konfigurace protokolu a výsledků měření na platformě Android

10.1.2 Vytvoření databáze

Vytvoření instance databáze je realizováno ve třídě `NetworkToolsApplication`, která představuje kontext celé Android aplikace. Databáze je inicializována viz kód 10.1.

Zdrojový kód 10.1: Vytvoření instance Room databáze

```
class NetworkToolsApplication : Application() {

    /**
     * Lazy initialization of database instance.
     * Providing instance of Room database used for accessing individual
     ↪  repositories.
     */
    val database by lazy { AppDatabase.getDatabase(this) }

    ...
    ...
```

Z kódu 10.1 lze vidět inicializaci databáze Room pro platformu Android. Třída AppDatabase slouží k vytvoření schématu dané databáze a specifikace některých funkcionalit, jako je například verzování databáze, nebo definice inicializačních dat. Dále třída AppDatabase definuje všechny využité tabulky, viz kód 10.2.

Zdrojový kód 10.2: Vytvoření instance Room databáze

```
@Database(
    entities = [SshServerConfiguration::class, SshTestCase::class,
    ↪  SshTestResult::class, SshMeasuredTimeResult::class,
    ↪  SshTestCasesServerConfigurationsCrossRef::class,
    ↪  PingConfiguration::class, PingTestCase::class, PingTestResult::class,
    ↪  PingMeasuredTimeResult::class, PingTestCaseConfigurationCrossRef::class,
    ↪  HttpConfiguration::class, HttpTestCase::class, HttpTestResult::class,
    ↪  HttpMeasuredTimeResult::class,
    ↪  HttpTestCasesWithConfigurationsCrossRef::class],
    version = 1,
    exportSchema = true
)
@TypeConverters(RoomConverters::class)
abstract class AppDatabase : RoomDatabase() {
    /**
     * @return Return Data access object for SSH configurations
     */
    abstract fun sshConfigurationDao(): SshServerConfigurationDao

    ...
}
```

Z kódu 10.2 lze vidět, že je třída AppDatabase anotovaná pomocí dvou anotací.

První anotací je `@Database`. Tato anotace specifikuje, které tabulky budou v aktuální databázi využity pro manipulaci s daty. Při deklaraci nové entity tedy musí být entita přidána i do tohoto seznamu. Anotace `@Database` také obsahuje parametr s názvem `version`. Tento parametr slouží pro verzování databáze a při změně datové struktury by tato verze měla být zvýšena. Druhá anotace `@TypeConverters` slouží pro definici konvertorů pro ukládání dat do databáze. Konvertory jsou podstatné pouze pro typy, které nejsou primitivní a `Room` databáze s nimi nedokáže standardně pracovat. Mezi tyto typy patří například `java.util.Date`, `java.lang.Enum<>`, nebo `java.math.BigDecimal` sloužící pro uložení přesnějších výsledků.

Dále lze z kódu vidět, že třída `AppDatabase` je potomkem třídy `RoomDatabase`. Tato dědičnost je povinná pro každou třídu anotovanou jako `@Database` [dev24c].

Posledním důležitým bodem kódu 10.2 je deklarace metody `sshConfigurationDao`. Tato metoda je uvedena jako příklad definice objektu pro přístup k databázovým datům, viz kapitola 10.1.4. Definice této metody je automaticky vygenerována před startem aplikace, tudíž není potřeba definovat tělo této metody.

10.1.3 Deklarace databázových entit

Databázové entity jsou jsou třídy anotované jako `@Entity` a přidané v seznamu databázových entit, viz kapitola 10.1.2. Každá entita reprezentuje jednu databázovou tabulku. Databázová entita může vypadat například viz kód 10.3.

Zdrojový kód 10.3: Deklarace entity pro databázi `Room`

```
1  /**
2   * Class representing configuration of SSH served for testing.
3   */
4  @Entity(tableName = EntityColumnNameRegistry.SSH_CONFIGURATION_TABLE)
5  open class SshServerConfiguration(
6      @ColumnInfo(name = EntityColumnNameRegistry.NAME_COLUMN) var name: String? =
       ↪  "",
7      @ColumnInfo(name = EntityColumnNameRegistry.HOST_COLUMN) var host: String? =
       ↪  "",
8      @ColumnInfo(name = EntityColumnNameRegistry.USER_COLUMN) var user: String? =
       ↪  "",
9      @ColumnInfo(name = EntityColumnNameRegistry.AUTHENTICATION_METHOD_COLUMN)
       ↪  var authenticationMethod: SshAuthMethod? = SshAuthMethod.PASSWORD,
10     @ColumnInfo(name = EntityColumnNameRegistry.AUTH_COLUMN) var auth: String? =
       ↪  "",
11     @ColumnInfo(name = EntityColumnNameRegistry.IV_COLUMN) var iv: String? = "",
12     @ColumnInfo(name = EntityColumnNameRegistry.PORT_COLUMN) var port: Int? =
       ↪  22,
```

```
13      @ColumnInfo(name = EntityColumnNameRegistry.LOCATION_COLUMN) var location:
        ↪  String? = "",
14      @ColumnInfo(name = EntityColumnNameRegistry.SFTP_MODE_COLUMN) var sftpMode:
        ↪  SftpMode? = SftpMode.SFTP_UPLOAD,
15      @ColumnInfo(name = EntityColumnNameRegistry.ID_COLUMN)
        ↪  @PrimaryKey(autoGenerate = true) var id: Long = 0
16  )
```

Z kódu 10.3 lze vidět deklarace entity pro ukládání SSH konfigurací. Pomocí anotace `@Entity` je entitě přiřazen název příslušné SQL tabulky. Následně jsou uvnitř třídy definovány příslušné atributy, které jsou anotovány jako `@ColumnInfo`. Anotace `@ColumnInfo` zaručí, že bude atribut mapován do databázové tabulky jako sloupec se jménem uvedeným v parametru `name`. Důležitým atributem pro každou entitu je atribut anotovaný jako `@PrimaryKey`. Tato anotace nastaví daný atribut jako primární klíč dané SQL tabulky. Anotace má navíc nastavený parametr `autoGenerate` na `true`. Tato vlastnost zajistí, že pokud není klíč vyplněn, tak jej databáze Room automaticky vygeneruje.

10.1.4 Operace nad databází

Pro dosažení databázových operací bylo využito návrhu `Dao-DbHelper-ViewModel`. Rozhraní definované jako `Dao` slouží pro definice veškerých databázových transakcí. Třída `DbHelper` slouží pro omezení spuštění metod na popředí aplikace. Třída `ViewModel` nakonec spouští databázové transakce na pozadí a jako výsledek může volat `callback` funkci, která je v případě potřeby předána v parametru a může poskytovat výsledek dané transakce.

Pro definici databázových transakcí je možné využít dvou základních přístupů. Prvním přístupem, který slouží pro jednoduché operace, je přístup za použití anotací `@Insert`, `@Update`, nebo `@Delete`. Jak již z názvu anotací vyplývá, tyto anotace slouží pro jednoduchou manipulaci s daty. Konkrétně metoda anotovaná jednou z těchto anotací musí v parametru přijímat entitu, nad kterou je potřeba operaci provést, viz kód 10.4.

Zdrojový kód 10.4: Deklarace metody pro vložení záznamu do databáze Room

```
1   @Insert(onConflict = OnConflictStrategy.REPLACE)
2   suspend fun insert(sshServerConfiguration: SshServerConfiguration): Long
```

Druhým přístupem využitým pro získání dat je využití anotace `@Query` společně s definicí jejího parametru `value`. Parametr je specifikovaný jako řetězec obsahující JPQL dotaz. Syntaxe JPQL dotazovacího jazyka usnadňuje tvorbu nativních SQL dotazů, viz kód 10.5.

Zdrojový kód 10.5: Deklarace metody pro nalezení záznamu z databáze Room

```
1    @Query("SELECT * FROM SSH_CONFIGURATION WHERE ID=:id")
2    suspend fun findById(id: Long): SshServerConfiguration
```

Z kódu 10.5 lze vidět definici metody pro nalezení konfigurace SSH serveru podle jejího identifikátoru.

10.2 Ukládání a získání hesel pro SSH konfigurace

Pro účel testování pomocí protokolu SFTP bylo potřeba řešit problém ukládání klientských hesel a jejich následného získání. Hesla jsou velice citlivými daty a proto bylo využito maximálně spolehlivých postupů pro jejich zabezpečení.

I přes veškerá zabezpečení je vhodné využít pro definici SSH konfigurace testovacího uživatele s omezenými právy, viz příručka A.1, například pouze k cílovým souborům a dále využívat dostatečně silných hesel.

K zabezpečení hesel je využito Android Keystore. Systém Android Keystore umožňuje ukládat kryptografické klíče do kontejneru, aby bylo obtížnější je ze zařízení získat. Jakmile jsou klíče v úložišti klíčů, lze je používat pro kryptografické operace, přičemž materiál klíče zůstává neexportovatelný. Systém úložiště klíčů také umožňuje omezit, kdy a jak lze klíče používat, například vyžadovat ověření uživatele pro použití klíče nebo omezit použití klíčů pouze v určitých kryptografických režimech [dev24b].

Při uložení SSH konfigurace se tedy zavolá funkce k zašifrování daného hesla, viz kód 10.6.

Zdrojový kód 10.6: Šifrování hesel pro SSH konfiguraci

```
1    /**
2     * Method encrypting plaintext password for specific ID.
3     *
4     * @param id ID of object storing password.
5     * @param password Plaintext password to be decrypted.
6     * @return Encrypted material for password. This material can be used later
      ↪  for decryption of password.
7     */
8    fun encrypt(id: Long, password: String): EncryptedMaterial {
9        var secretKey = getOrCreateKey(id)
10       val cipher = Cipher.getInstance(CIPHER_ALGORITHM)
```

```
11          try {
12              cipher.init(Cipher.ENCRYPT_MODE, secretKey)
13          } catch (e: KeyPermanentlyInvalidatedException) {
14              keyStore.deleteEntry(getAlias(id))
15              keyStore.load(null)
16              secretKey = createKey(id)
17              cipher.init(Cipher.ENCRYPT_MODE, secretKey)
18          }
19          val iv = cipher.iv
20          val encoder = Base64.getEncoder()
21          val encryptedPassword =
            ↪  cipher.doFinal(password.toByteArray(Charsets.UTF_8))
22          return EncryptedMaterial(encoder.encodeToString(encryptedPassword),
            ↪  encoder.encodeToString(iv))
23      }
```

Z kódu 10.6 lze vidět, že se na začátku vytvoří, nebo získá kryptografický klíč pro konfiguraci podle jejího identifikátoru. Dále proběhne inicializace klíče pro mód šifrování. Následně je získána hodnota `iv`, která značí inicializační vektor dané šifry a je potřebná pro její následné dešifrování. Poté se provede samotné šifrování hesla. Hodnoty `iv` i zašifrovaná podoba hesla jsou reprezentovány jako pole bytů a pro jejich snadnější interpretaci a ukládání jsou před navrácením konvertovány do formátu `Base64`.

Dešifrování hesla, popřípadě množiny hesel je velice citlivou operací a proto je v aplikaci pro tuto operaci vyžadováno potvrzení biometrikou. Konkrétně je vyžadováno potvrzení pomocí otisku prstu, viz kód 10.7.

Zdrojový kód 10.7: Dešifrování hesel pro SSH konfiguraci

```
1       /**
2        * Method decrypting passed Encrypted material and callbacking result in
         ↪  case of success.
3        *
4        * @param context Current context calling this method.
5        * @param encryptedIdMap Map if ids to be decrypted with its encrypted
         ↪  material used for decryption.
6        * @param onSuccess Callback passing map if id with decrypted passwords.
7        * @param onFail Callback called on decryption fail.
8        */
9       fun decrypt(context: Context, encryptedIdMap: Map<Long, EncryptedMaterial>,
        ↪  onSuccess: (Map<Long, String>) -> Unit, onFail: () -> Unit) {
10              BiometricPrompt.Builder(context)
                ↪  .setTitle(context.getString(R.string.confirm_password_store))
11              .setDescription(
```

```
12      context.getString(R.string.confirm_password_store_description))
        ↪  .setNegativeButton(
13              context.getString(R.string.cancel), context.mainExecutor
14          ) { _, _ -> onFail() }.build().authenticate(CancellationSignal(),
            ↪  context.mainExecutor, object : AuthenticationCallback() {
15              override fun onAuthenticationSucceeded(result:
                ↪  BiometricPrompt.AuthenticationResult?) {
16                  val cipher = Cipher.getInstance(CIPHER_ALGORITHM)
17                  val passwordMap = HashMap<Long, String>()
18                  val decoder = Base64.getDecoder()
19                  for (entry in encryptedIdMap) {
20                      try {
21                          val secretKey = getOrCreateKey(entry.key)
22                          val decodedPass =
                            ↪  decoder.decode(entry.value.password)
23                          val decodedIv = decoder.decode(entry.value.iv)
24                          val ivSpec = IvParameterSpec(decodedIv)
25                          cipher.init(Cipher.DECRYPT_MODE, secretKey, ivSpec)
26                          val decryptedString =
                            ↪  String(cipher.doFinal(decodedPass),
                            ↪  StandardCharsets.UTF_8)
27                          passwordMap[entry.key] = decryptedString
28                      } catch (ex: Exception) {
29                          passwordMap[entry.key] = ""
30                      }
31                  }
32                  onSuccess(passwordMap)
33              }
34
35              override fun onAuthenticationFailed() {
36                  onFail()
37              }
38
39              override fun onAuthenticationError(errorCode: Int, errString:
                ↪  CharSequence?) {
40                  onFail()
41              }
42          })
43      }
```

Z kódu 10.7 lze vidět, že metoda `decrypt` využívá objektu `BiometricPrompt`. `BiometricPrompt` je třída poskytovaná platformou Android a poskytuje funkcionalitu ověření `Biometrických` vlastností uživatele. Metody `onAuthenticationFailed`

nebo `onAuthenticationError` jsou volány při neúspěchu ověření uživatele pomocí biometriky, například při zrušení výzvy k ověření, přepnutí aplikace na pozadí, nebo při chybějící funkcionalitě ověření pomocí biometriky. Metoda `onAuthentication-Succeeded` je volána jako potvrzení úspěšného biometrického ověření. V této metodě proběhne dešifrování všech požadovaných hesel, které jsou metodě `decrypt` předány v parametru `encryptedIdMap`. Dešifrovaná hesla jsou následně uložena do mapy, kde klíčem je identifikátor daného hesla a hodnotou dešifrované heslo. Výsledná mapa je nakonec předána funkci `onSuccess`, která je taktéž předána parametrem metody `decrypt`.

Důležitou informací pro zašifrovaná hesla je, že nikdy neopustí zařízení. To znamená že i při komunikaci a synchronizaci dat je heslo zachováno pouze na zařízení Android.

10.3 Přihlášení

V této kapitole bude popsán postup přihlášení uživatele pomocí nástroje `Auth0`. Následně bude rozebrána správa a využití tokenů poskytnutých nástrojem `Auth0`.

10.3.1 Nastavení důležitých proměnných `Auth0`

Pro přihlášení je potřeba nejdříve deklarovat důležité proměnné využívané k autentizaci a autorizaci pomocí nástroje `Auth0`. Pro nastavení těchto proměnných je důležité znát pojmy definované v kapitole 7. Definice jednotlivých vlastností je umístěna v souboru `/values/auth0.xml` adresáře `/res`, který slouží pro ukládání všech souborů, které jsou vyžadovány pro přístup ze zdrojového kódu aplikace. Konfigurační soubor vypadá viz kód 10.8.

Zdrojový kód 10.8: Konfigurační soubor pro nastavení nástroje `Auth0`

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <string name="com_auth0_scheme">pingmaster</string>
4     <string name="com_auth0_audience">https://pingmaster.cz</string>
5     <string name="com_auth0_domain">dev-5zwd5xrpvxrfowlf.us.auth0.com</string>
6     <string name="com_auth0_client_id">DFnDAKjS3eYNQcTGTkBTX4fIkJvN0mcT</string>
7     <string name="com_auth0_scopes">openid profile email offline_access
    ↪  pingmaster</string>
8 </resources>
```

Z kódu 10.8 lze vidět konfiguraci důležitých proměnných pro nástroj `Auth0`. Význam jednotlivých nastavení je následující:

- **com_auth0_scheme** - Tato hodnota reprezentuje využité schéma pro přihlášení. Hlavní význam této hodnoty je jeho přidání do `Callback` adresy, viz kapitola 7.
- **com_auth0_audience** - Tato hodnota je využita pro vyžádání tokenu platného k volání `API` s `URL` adresou nastavenou na tuto hodnotu, viz kapitola 7.2.
- **com_auth0_domain** - Tato hodnota přestavuje doménu vytvořené Android aplikace v nástroji `Auth0`, viz obrázek 7.1.
- **com_auth0_client_id** - Hodnota představující jedinečný identifikátor Android aplikace vytvořené v nástroji `Auth0`, viz obrázek 7.1.
- **com_auth0_scopes** - Hodnota představující vyžadované `Scopes` udělené pomocí přístupového tokenu. Z kódu 10.8 lze vidět, že je vyžádáno celkově pět `Scopes`. `Scope openid` poskytuje možnost přihlášení uživatele pomocí systémů třetích stran, jako je Google, Facebook a podobné. `Scopes profile` a `email` poté vyžadují přístup k základním uživatelským informacím, jako je jméno, příjmení, `URL` fotografie a e-mailová adresa uživatele. `Scope offline_access` vynucuje využití obnovovacího tokenu při žádání o novou sadu tokenů. Poslední `Scope pingmaster` představuje roli vytvořenou ručně v nástroji `Auth0` a potřebnou pro autorizaci ve volaném `API`. Nastavení `Scope pingmaster` lze vidět na obrázku 7.6 a definici autorizace pro tento `Scope` v kódu 8.8.

10.3.2 Přihlášení pomocí `Auth0`

Pro přihlášení byla vytvořena třída `LoginUtils`, která poskytuje všechny důležité funkcionality potřebné pro přihlášení uživatele.

Při volání jakékoliv metody je volána metoda `initAccount`, která zkontroluje, zda již bylo navázáno spojení s nástrojem `Auth0` a pokud ne, tak ho naváže, viz kód 10.9.

Zdrojový kód 10.9: Inicializování spojení s `Auth0`

```
1   /**
2    * Method performing Auth0 account connection if it wasnt established yet.
3    * @param context Current context used for achieving resources values.
4    */
5   private fun initAccount(context: Context) {
6       if (account == null) {
7           account = Auth0(
8               context.getString(R.string.com_auth0_client_id),
                ↪  context.getString(R.string.com_auth0_domain)
```

```
            )
            authenticationAPIClient = AuthenticationAPIClient(account!!)
        }
        credentialsManager = CredentialsManager(authenticationAPIClient!!,
        ↪ SharedPreferencesStorage(context))
    }
```

Z kódu 10.9 lze vidět, že na začátku metody `initAccount` proběhne kontrola, zda již Auth0 spojení nebylo inicializováno. Pokud je zjištěno že spojení zatím neproběhlo, je vytvořeno s využitím konfigurace z kapitoly 10.3.1 nové spojení. Dalším důležitým krokem je inicializování objektu `CredentialsManager`, který slouží pro ukládání přijaté sady tokenů do specifického úložiště. Jako úložiště tokenů byly zvoleny sdílené preference (`SharedPreferences`) v privátním režimu. To znamená, že tokeny jsou přístupné pouze vlastnící aplikaci.

Prvním krokem při zapnutí aplikace je pokus o přihlášení uživatele pomocí tichého přihlášení. Tiché přihlášení probíhá na pozadí a snaží se zjistit, zda byl uživatel již někdy přihlášen a pokud ano, zda uložená sada tokenů je stále validní, viz kód 10.10.

Zdrojový kód 10.10: Tiché přihlášení uživatele pomocí Auth0

```
    /**
     * Method performing silent login. If user was never logged it, this method
     ↪  should fail (callback.onFailure method will be called).
     *
     * @param context Current context used for account creation.
     * @param callback Callback used for handling of specific events.
     */
    fun silentLogin(context: Context, callback: Callback<Credentials,
    ↪ CredentialsManagerException>) {
        initAccount(context)
        credentialsManager!!.getCredentials(object : Callback<Credentials,
        ↪ CredentialsManagerException> {
            override fun onSuccess(result: Credentials) {
                authenticationSuccess(result)
                callback.onSuccess(result)
            }

            override fun onFailure(error: CredentialsManagerException) {
                callback.onFailure(error)
            }
        })
    }
```

Z kódu 10.10 lze vidět volání metody `getCredentials`. Tato metoda se pokouší vyhledat, zda již byla v minulosti vyžádána sada tokenů a je uložena v úložišti tokenů. Pokud uložené tokeny jsou již expirované z důvodu jejich životnosti, tak se je pokusí automaticky obnovit.

Pokud tiché přihlášení neuspěje uživatel je přesměrován na uvítací obrazovku, odkud může zvolit možnost přihlášení. Přihlášení probíhá pomocí webové stránky standardně poskytované knihovnou `Auth0` pro Android, viz kód 10.11.

Zdrojový kód 10.11: Standardní přihlášení uživatele pomocí `Auth0`

```
1   /**
2    * Method providing Web Authentication with usage of Auth0.
3    *
4    * @param context Current context used for login.
5    * @param callback Callback used for specific events of Login flow.
6    */
7   private fun webAuthLogin(context: Context, callback:
    ↪  AdvancedCallBack<Credentials>) {
8           WebAuthProvider.login(account!!)
9           .withScheme(context.getString(R.string.com_auth0_scheme))
10          .withAudience(
11              context.getString(R.string.com_auth0_audience)
12          ).withScope(
13              context.getString(R.string.com_auth0_scopes)
14          ).start(context, object : Callback<Credentials,
            ↪  AuthenticationException> {
15              override fun onFailure(error: AuthenticationException) {
16                  callback.onFailure(error)
17              }
18
19              override fun onSuccess(result: Credentials) {
20                  authenticationSuccess(result)
21                  callback.onFirstLogin(result)
22              }
23          })
24  }
```

Z kódu 10.11 lze vidět volání metody `login` poskytované třídou `WebAuthProvider`. Po volání metody `login` jsou nastaveny dodatečné informace pro získání relevantní sady tokenů, viz kapitola 10.3.1. Voláním metody `start` se spustí webová stránka, která slouží pro přihlášení uživatele. Při úspěšném přihlášení je volána metoda `onSuccess`, která obsahuje v parametru nově přijatou sadu tokenů.

10.4 Komunikace se serverem

Komunikace se serverem probíhá pomocí `REST API`. K usnadnění komunikace byla vybrána knihovna `retrofit2`. `Retrofit2` je typově bezpečný klient `HTTP` pro Android a Javu [Vas23].

10.4.1 Navázání komunikace

Pro navázání komunikace byla ve třídě `NetworkToolsApplication` vytvořena metoda `createService`, která vytvoří instanci rozhraní potřebného pro komunikaci se vzdáleným serverem, viz kód 10.12.

Zdrojový kód 10.12: Vytvoření rozhraní pro komunikaci se vzdáleným serverem

```
1   /**
2    * Instance interface including retrofit2 calls.
3    *
4    * @param serviceClass Class of interface which will be instantiated.
5    * @return Instance od interface of with retrofit2 calls.
6    */
7   fun <T> createService(serviceClass: Class<T>): T {
8       httpClient.interceptors().clear()
9       httpClient.addInterceptor(AuthenticationInterceptor(this))
10      httpClient.authenticator { _, response ->
11          runBlocking {
12              val request = addAuthentication(response.request.newBuilder(),
                ↪  refreshToken())
13              if (responseCount(response) >= 3) {
14                  null
15              } else request
16          }
17      }
18      retrofitBuilder.baseUrl(getString(R.string.backup_url))
19      retrofitBuilder.client(httpClient.build())
20      return retrofitBuilder.build().create(serviceClass)
21  }
```

Z kódu 10.12 lze vidět vytváření rozhraní pro komunikaci se serverem. Před vytvořením tohoto rozhraní jsou definovány některé důležité vlastnosti. Metodou `addInterceptor` je předána instance objektu, která se stará o to, že v případě neúspěšného volání se pokusí vyměnit aktuální sadu tokenů za novou a požadavek odešle znovu. Metodou `baseUrl` se poté nastaví `URL` adresa serveru, se kterým bude komunikováno. Konkrétně se nastavení adresy serveru nachází v souboru

`/values/network.xml` adresáře `/res`, který slouží pro ukládání všech souborů, které jsou vyžadovány pro přístup ze zdrojového kódu aplikace.

10.4.2 Definice přístupových bodů

Definice a příprava volání přístupového bodu je realizována pouze pomocí rozhraní, viz kód 10.13.

Zdrojový kód 10.13: Definice přístupového bodu

```
1  /**
2   * Interface used for client-server action for SSH protocol specifications.
3   */
4  interface SshActionService {
5
6      /**
7       * Endpoint adding sshTestCase to server. Result of this call is identifier
       ↪  of inserted object in server database.
8       */
9      @POST("/sshAction/addSshTestCase")
10     fun addSshTestCase(@Body sshTestCase: SshTestCase): Call<Long>
11
12     ...
13     ...
14 }
```

Z kódu 10.13 lze vidět definici přístupového bodu. Přístupový bod se nachází na adrese `/sshAction/addSshTestCase` a je poskytnut pro `HTTP` metodu `POST`. Tělo požavku je předáno parametrem `sshTestCase`, který je anotován jako `@Body`. Tato anotace značí, že objekt bude automaticky konvertován do formátu `Json` a vložen do těla požadavku. Návratovou hodnotou této metody je číslo, které reprezentuje identifikátor vloženého objektu v databázi na serveru.

10.4.3 Volání přístupových bodů

Volání přístupových bodů může probíhat v libovolné aktivitě. Jako první je nutné vytvořit instanci požadovaného rozhraní pro komunikaci, viz kód 10.14.

Zdrojový kód 10.14: Vytvoření rozhraní pro volání přístupových bodů

```
1      @SuppressLint("MissingSuperCall")
2      override fun onCreate(savedInstanceState: Bundle?) {
3          super.onCreate(savedInstanceState, R.layout.activity_ssh_dashboard)
4          actionService = (application as
           ↪  NetworkToolsApplication).createService(SshActionService::class.java)
```

```
5           addAddTestCaseButtonClickListener()
6           addAddServerConfigurationButtonClickListener()
7           addLiveData()
8       }
```

Z kódu 10.14 lze vidět metodu `onCreate`, která je je implicitně volána při startu aktivity. V této metodě je na řádce čtyři vytvořena instance rozhraní `SshActionService` pomocí metody `createService` popsané v kapitole 10.4.1.

Posledním krokem ve využití komunikace se serverem je volání samotné metody. Metody jsou volány jako asynchronní, aby nezpomalovaly chod aplikace. Volání metody pro přidání testovacího případu pro SSH tedy může vypadat viz kód 10.15.

Zdrojový kód 10.15: Volání a zpracování výsledků poskytnutých přístupovým bodem

```
1       /**
2        * Add SSH test case on server. If operation is successfully completed, test
         ↪  case will be added locally too.
3        * @param sshTestCase Test case to add.
4        */
5       private fun addSshTestCaseOnline(sshTestCase: SshTestCase) {
6           actionService.addSshTestCase(sshTestCase).enqueue(object :
            ↪  Callback<Long> {
7               override fun onResponse(call: Call<Long>, response: Response<Long>)
                ↪  {
8                   if (response.isSuccessful) {
9                       val id = response.body()
10                      if (id != null) {
11                          sshTestCase.id = id
12                          sshTestCaseDao.insert(sshTestCase) { tcId ->
13                              val sshServerTestCaseDetailActivity =
                                ↪  Intent(this@SshDashboardActivity,
                                ↪  SshServerTestCaseDetailActivity::class.java)
14                              sshServerTestCaseDetailActivity
15                                  .putExtra(IntentExtraIdRegistry.SSH_TEST_CASE_ID,
                                    ↪  tcId)
16                              startActivity(sshServerTestCaseDetailActivity)
17                          }
18                      }
19                  } else {
20                      Log.e("Actions", "Failed to add Test Case to remote server
                        ↪  (ended with status ${response.code()}).")
21                  }
22              }
23
```

```
24              override fun onFailure(call: Call<Long>, t: Throwable) {
25                  Log.e("Actions", "Failed to add Test Case to remote server.", t)
26              }
27          })
28      }
```

Kód 10.15 ukazuje příklad volání přístupového bodu serveru a zpracování vrácených výsledků, popřípadě i ošetření chybových stavů vrácených při chybě v komunikaci nebo chybě vrácené serverem.

10.5 Dodatečné funkcionality

V této kapitole budou probrány dodatečné funkcionality Android aplikace, které jsou podstatné pro prezentaci výsledků měření, nebo samotné fungování aplikace. Konkrétně zde budou rozebrány funkce jako získání lokace uživatele a získání kvality aktuálního připojení za pomoci zařízení Android.

10.5.1 Získání šířky pásma

Důležitou funkcionalitou je získání šířky pásma poskytovaného aktuálním připojením. Šířka pásma reprezentuje maximální teoretickou možnost rychlost přenosu [Sol]. Tato hodnota je ukládána spolu s měřeními rychlosti stahování/nahrávání pro lepší prezentaci výsledků. Šířka pásma je ze zařízení Android získána pomocí metody `getConnectionSpeed`, viz kód 10.16.

Zdrojový kód 10.16: Získání šířky pásma z platformy Android

```
1       /**
2        * Return
3        * @param context Current context used for retrieving of bandwidth
4        * @return Connection speed filled with bandwidth (upload/download)
5        */
6       fun getConnectionSpeed(context: Context): ConnectionSpeed {
7           val cm = context.getSystemService(Context.CONNECTIVITY_SERVICE) as
            ↪  ConnectivityManager
8           val nc = cm.getNetworkCapabilities(cm.activeNetwork)
9           return ConnectionSpeed(nc?.linkUpstreamBandwidthKbps,
            ↪  nc?.linkDownstreamBandwidthKbps)
10      }
```

Z kódu 10.16 lze vidět získání šířky pásma pomocí systémové služby pro správu internetových připojení.

10.5.2 Získání lokace zařízení

Lokace zařízení je získávána pomocí dvou metod. První metodou získání lokace uživatele je GPS, která obvykle poskytuje přesnější informace o poloze než druhá možnost, avšak nemusí být vždy zařízením povolena. Druhou možností je získání aktuální lokace uživatele pomocí internetových služeb, viz kód 10.17.

Zdrojový kód 10.17: Získání lokace zařízení

```
1   val gpsLocation: android.location.Location? =
    ↪ locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER)
2   if (isNetworkLocationEnabled) {
3       networkLocation = locationManager
        ↪ .getLastKnownLocation(LocationManager.NETWORK_PROVIDER)
4   }
5
6   if (gpsLocation != null && networkLocation != null) {
7       if (gpsLocation.accuracy > networkLocation.accuracy) {
8           location.latitude = networkLocation.latitude
9           location.longitude = networkLocation.longitude
10      } else {
11          location.latitude = gpsLocation.latitude
12          location.longitude = gpsLocation.longitude
13      }
14  } else {
15      if (gpsLocation != null) {
16          location.latitude = gpsLocation.latitude
17          location.longitude = gpsLocation.longitude
18      } else if (networkLocation != null) {
19          location.latitude = networkLocation.latitude
20          location.longitude = networkLocation.longitude
21      }
22  }
```

Z kódu 10.17 lze vidět získání aktuální lokace zařízení. Při možnosti využití obou metod pro lokalizaci zařízení je zjištěno, která z metod poskytuje přesnější výsledky a ta je následně uložena do vráceného objektu. Návratovým objektem metody je lokace, která v případě úspěšného získání polohy poskytuje zeměpisnou šířku a zeměpisnou délku polohy zařízení.

10.5.3 Režim hosta

Aplikace poskytuje možnost využití funkcionalit aplikace i bez přihlášení. Rozlišení mezi módem bez přihlášení a přihlášeným uživatelem je zajištěno pomocí třídy

`AppStateUtils`. Tato třída poskytuje funkce, které ukládají stav aplikace pomocí sady metod, viz kód 10.18.

Zdrojový kód 10.18: Nastavení a získání módu aplikace

```
1   /**
2    * Set application state as offline (guest)
3    * @param context Current context used for setting up guest mode
4    */
5   fun setOffline(context: Context) {
6       setState(context, true)
7   }
8
9   /**
10   * Set application state as online (logged in)
11   * @param context Current context used for setting up logged in mode
12   */
13  fun setOnline(context: Context) {
14      setState(context, false)
15  }
16
17  /**
18   * Checking if application is offline (guest mode)
19   */
20  fun isOffline(context: Context): Boolean {
21      val sharedPreferences = context.getSharedPreferences(OFFLINE,
        ↪  Context.MODE_PRIVATE)
22      return sharedPreferences.getBoolean(OFFLINE, true)
23  }
24
25  /**
26   * Checking if application is online (logged in mode)
27   */
28  fun isOnline(context: Context): Boolean {
29      return !isOffline(context)
30  }
31
32  /**
33   * Set state of application. Save it to private shared preferences.
34   */
35  private fun setState(context: Context, value: Boolean) {
36      val sharedPreferences = context.getSharedPreferences(OFFLINE,
        ↪  Context.MODE_PRIVATE)
37      val editor = sharedPreferences.edit()
```

```
38          editor.putBoolean(OFFLINE, value)
39          editor.apply()
40      }
```

Z kódu 10.18 lze vidět metody sloužící pro manipulaci a získání informací o aktuální režimu aplikace. Metody pro nastavení režimu jsou volány pouze na akci úspěšného přihlášení, kdy mód aplikace je nastaven na `online`, nebo při stisku úvodního tlačítka pro pokračování aplikace v `guest` režimu, kdy je režim nastaven jako `offline`. V těle aplikace je poté pomocí metod `isOnline` a `isOffline` zjištěno, zda mají být data ukládány pouze lokálně, nebo synchronizovány se vzdáleným serverem.

10.6 Testování

Pro testování Android aplikace byly z důvodu její velké komplexity vytvořeny pouze testy pro `SSH`. Všechny testy jsou umístěny ve složce `androidTest`, která slouží pro testování za využití emulátoru, nebo fyzického zařízení. K testování je využito knihovny `Espresso`. Knihovna poskytuje široké množství funkcionalit určených pro testování uživatelského rozhraní [dev24d].

Metoda pro testování uživatelského rozhraní může vypadat viz kód 10.19.

Zdrojový kód 10.19: Testování pomocí knihovny `Espresso`

```
1       @Test
2       fun testSSHServerConfigurationItemClick() {
3           ActivityScenario.launch(SshDashboardActivity::class.java)
            ↪ onView(withId(R.id.ssh_server_configuration_recycler_list)).perform(
            ↪ scrollToPosition<SshServerConfigurationPreviewAdapter.ViewHolder>(1),
            ↪ actionOnItemAtPosition<SshServerConfigurationPreviewAdapter.
            ↪ ViewHolder>(1, click()))
4           intended(hasComponent(SshServerConfigurationActivity::class.
            ↪ java.getName())) .onView(withId(R.id.ssh_server_configuration_name))
            ↪ .check(matches(withText(endsWith("bad user"))))
5       }
```

Z kódu 10.19 lze vidět testování uživatelského rozhraní. Konkrétně metoda testuje zda po kliku na jednu z konfigurací je uživatel přenesen do dialogu pro úpravu dané konfigurace a zda obsahuje správné údaje.

Testů bylo napsáno celkem devět, pro ukázku je uveden i výstup testování, viz obrázek 10.2.

Tests	Duration	samsung SM-A346B
✓ Test Results	22 s	9/9
✓ SshActivitiesTest	22 s	9/9
✓ testSSHTestCaseAddRelationWithSSHServerConfiguration	3 s	✓
✓ testSSHTestCasesPresence	1 s	✓
✓ testSSHTestCaseEdit	4 s	✓
✓ testSSHTestCaseDeleteButton	1 s	✓
✓ testTestResultItemsPresence	3 s	✓
✓ testSSHTestCaseDeleteRelationWithSSHServerConfiguration	2 s	✓
✓ testSSHTestCaseItemClick	1 s	✓
✓ testTestResultData	2 s	✓
✓ testSSHServerConfigurationsPresence	1 s	✓

Obrázek 10.2: Výsledek `Espresso` testů pro platformu Android

Závěr 11

V rámci bakalářské práce byly důkladně analyzovány postupy a důležité vlastnosti měření kvality síťového připojení. Dále byly rozebrány nástroje, které k jsou k měření kvality připojení nejvhodnější a s pomocí nich byl vytvořen volně přenositelný modul pro platformu Android, který poskytuje funkce pro měření kvality síťového připojení.

Dále byla vyvinuta aplikace pro platformu Android, která tento modul využívá a dokáže uživateli přívětivě prezentovat jednotlivé konfigurace a zároveň vytvářet testovací sady, ve kterých je možné mít více konfigurací a tím pádem urychlit proces testování. Nakonec aplikace poskytuje snadno srozumitelnou prezentaci jednotlivých výsledků, která je důležitá zejména pro porovnání jednotlivých měření.

V druhé části práce byla vytvořena serverová aplikace, která slouží k synchronizaci jednotlivých konfigurací, testovacích sad, ale i měřených výsledků. Nakonec bylo navrženo a implementováno `API` sloužící pro export naměřených dat volně přes internet.

Z předchozích kapitol a výčtu jednotlivých implementovaných funkcionalit vyplývá, že všechny zadané požadavky byly splněny. Při analýze aplikace avšak bylo navrhnuto ještě několik funkcionalit, které by mohly být zapracovány do budoucna. Jednou z hlavních navržených funkcionalit je vytvoření grafů v Android aplikaci, pomocí kterých by bylo možné srovnávat naměřené výsledky na jedné obrazovce. Dalším navržených rozšířením je webová stránka, která by poskytovala přehledné filtrování v naměřených výsledcích a snadnější manipulaci s daty.

Uživatelská příručka A

A.1 Nastavení uživatele pro SSH

Pro testování pomocí protokolu SFTP je doporučeno využít pro autentizaci vzdálenému serveru uživatele s omezenými právy. Takového uživatele lze v systému Linux nastavit pomocí příkazů, viz kód A.1.

Zdrojový kód A.1: Nastavení uživatele pro testování SFTP

```
1 # Create new user with "home" directory
2 adduser --home /path/to/user/directory user_to_be_replaced
```

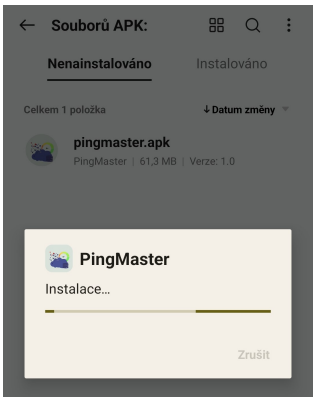
Příkaz A.1 nejdříve vytvoří uživatele, kterému následně vytvoří domovskou složku. Uživatel nemá práva k jiné složce než je vytvořená složka a v ní vložené soubory.

A.2 Rozdělení aplikace

Androidová část aplikace má několik vlastností. Nejprve je třeba aplikaci stáhnout a nainstalovat do vlastního zařízení. Po spuštění je třeba vybrat jednu ze dvou možností přihlášení do aplikace. Lze pokračovat jako host, nebo se přihlásit svým přihlašovacím jménem a heslem. Pokud uživatel není zatím přihlášený, existuje ve formuláři pro přihlášení i možnost registrace uživatele. Po vstupu do aplikace si uživatel může vybrat, zda bude měřit SSH, Ping nebo HTTP(s). V každém nástroji či protokolu je možné vytvořit novou konfiguraci serveru nebo testovací sadu. Po nastavení vlastností nutných k měření může uživatel spustit testy a zobrazit si jejich výsledky.

A.3 Stažení a instalace aplikace

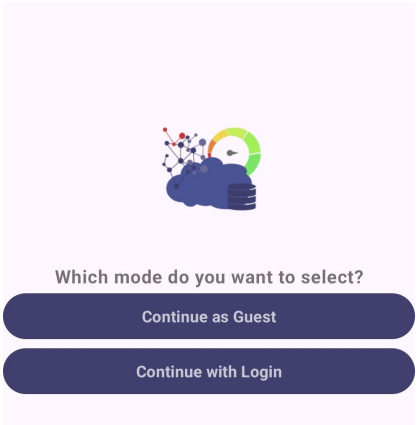
Pro spuštění aplikace Pingmaster je třeba stáhnout soubor pingmaster.apk a nainstalovat jej ve svém Android zařízení.



Obrázek A.1: Instalace Androidové aplikace

A.4 Použití aplikace

Po spuštění aplikace je uživateli zobrazena úvodní obrazovka s možností 2 režimů: pokračování jako host, nebo přihlášení uživatelským jménem a heslem.

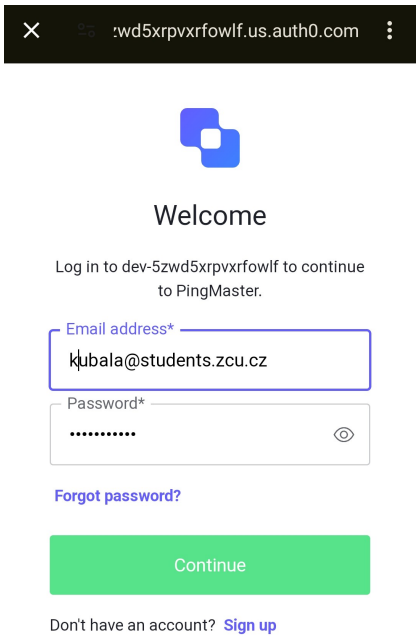


Obrázek A.2: Úvodní obrazovka aplikace

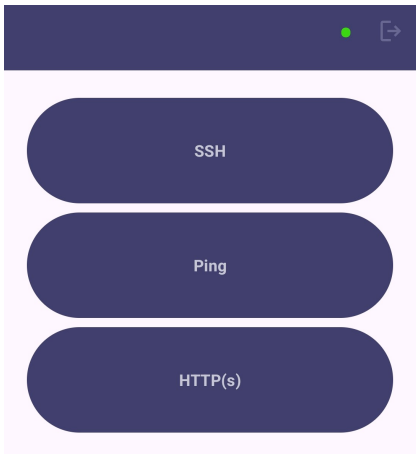
A.4.1 Přihlášení do aplikace

Pokud uživatel zvolí přihlášení pomocí uživatelského jména a hesla, je přenesen na stránku `Auth0` kde zadá své přihlašovací údaje. Pokud se uživatel nepřihlásí a pokračuje jako host, může procházet a využívat aplikaci stejným způsobem, nebude mít však zálohovaná data skrze více zařízení. Po zvolení jednoho z přístupů se uživateli zobrazí možnosti protokolů a nástrojů, které je možné v aplikaci měřit.

Poté co uživatel vybere jednu z možností se mu zobrazí obrazovka daného protokolu/nástroje. Na obrázku A.5 je obrazovka po té co uživatel co vybral protokol `SSH`. Struktura obrazovky je stejná i pro `Ping` a `HTTP(s)`, obsahuje seznam serverových konfigurací a seznam testovacích sad.



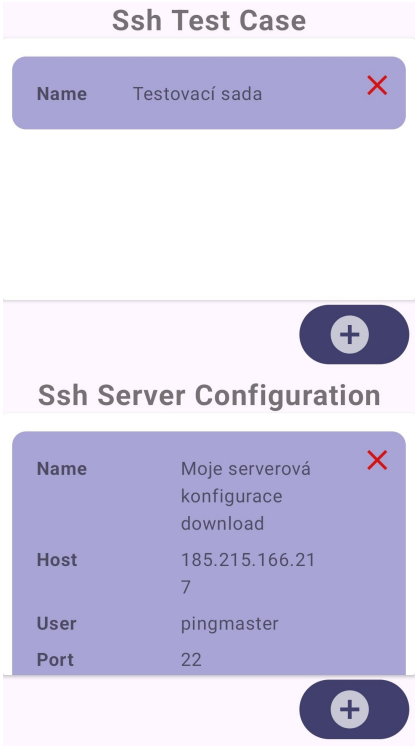
Obrázek A.3: Přihlášení do aplikace



Obrázek A.4: Možnosti protokolů a nástrojů

A.4.2 Nastavení serverových konfigurací

Uživatel může přidat serverových konfigurací několik, pro každou musí vyplnit hodnoty polí a zvolit zda se jedná o nahrávání či stahování dat. Vyplňuje se název serverové konfigurace, adresa serveru (ta může být vyplněna jako `IP` daného serveru nebo doména). Následně je nutné vyplnit přihlašovací údaje, pokud to server vyžaduje. `Location` představuje místo, kde se na serveru uloží testový soubor s naměřenými hodnotami, `Port` je nastaven jako primární identifikátor serverového spojení. Posledním nastavením je volba mezi měřením stahování a nahrávání, ta se



Obrázek A.5: Obrazovka SSH protokolu

volí pomocí ikonek vpravo dole viz obrázek A.6.

Stejně tak lze vytvořit konfigurace pro HTTP(s) a Ping.

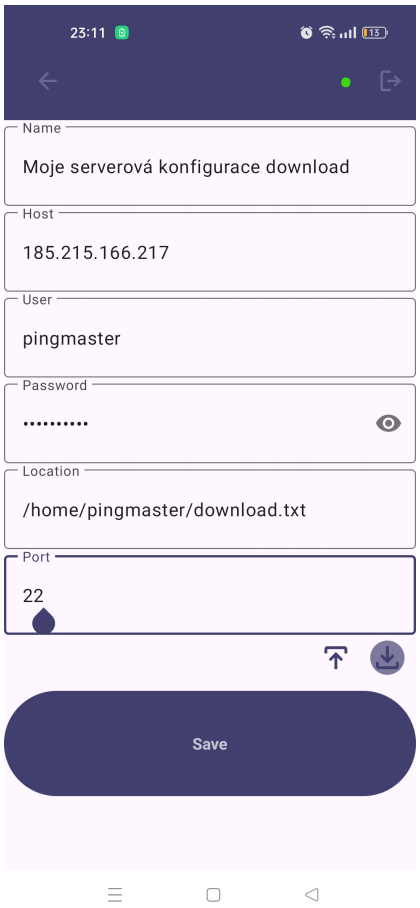
A.4.3 Vytvoření testovacích sad dat

Testovacích dat může být také vyrobeno několik, v nich uživatel nastaví vlastnosti testování a připojí k nim serverové konfigurace, kterými značí, na jakých serverech budou testy probíhat. V SSH uživatel vyplní název testovací sady, maximální čas běhu testů, maximální velikost přenášených dat a jak velký má být přenosový buffer. Zde má možnost uživatel zaškrtnout Smart Buffer Size, který ponechá velikost bufferu na systému. Tlačítkem vpravo dole lze přidat jednotlivé serverové konfigurace, které v této sadě chceme otestovat, viz obrázek A.8.

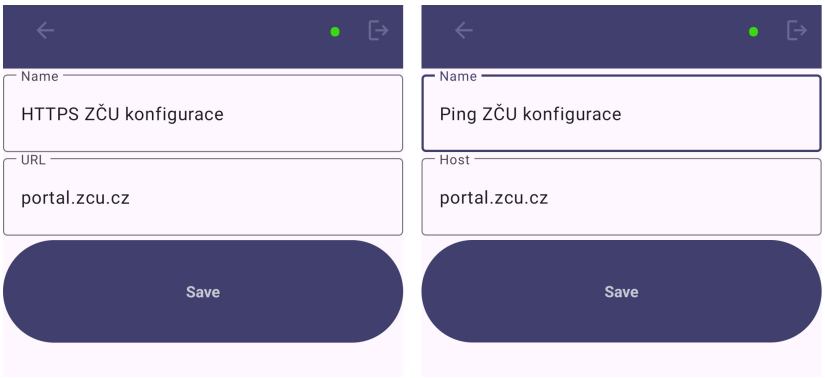
Oproti protokolům SSH a HTTP(s), v testovací sadě nástroje Ping se nastaví pouze počet tiků, které má nástroj vykonat pro dané konfigurace, viz obrázek A.9.

A.4.4 Spuštění testů

Test pak můžeme spustí pomocí tlačítka Run Test v testovací sadě viz A.9, A.8.



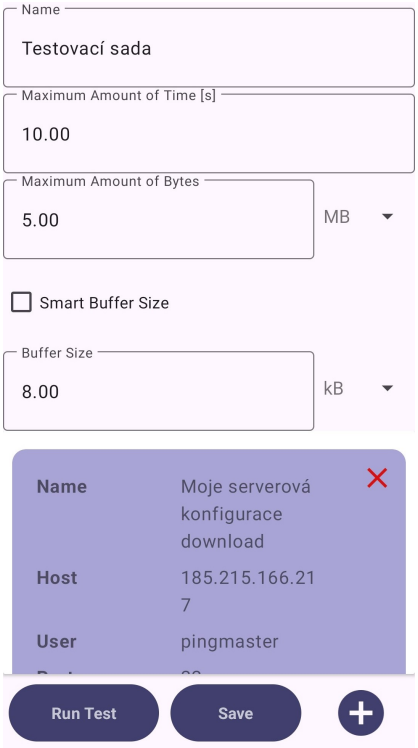
Obrázek A.6: Serverová konfigurace pro protokol `SSH`



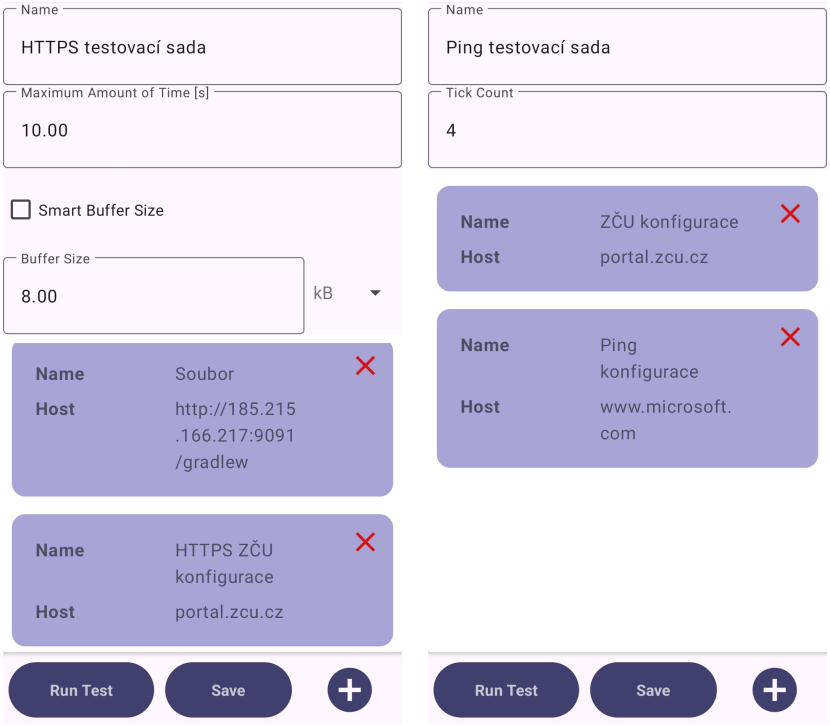
Obrázek A.7: Konfigurace pro protokol `HTTP(s)` a nástroj `Ping`

A.5 Zobrazení výsledků

Výsledky lze zobrazit dvojím způsobem, buď přímo v Android aplikaci stisknutím na serverovou konfiguraci v testovací sadě. Ta nám ukáže hlášku o stavu, výsledky měření a lokalizaci kde bylo měření provedeno, viz obrázek A.10. Pokud bylo více

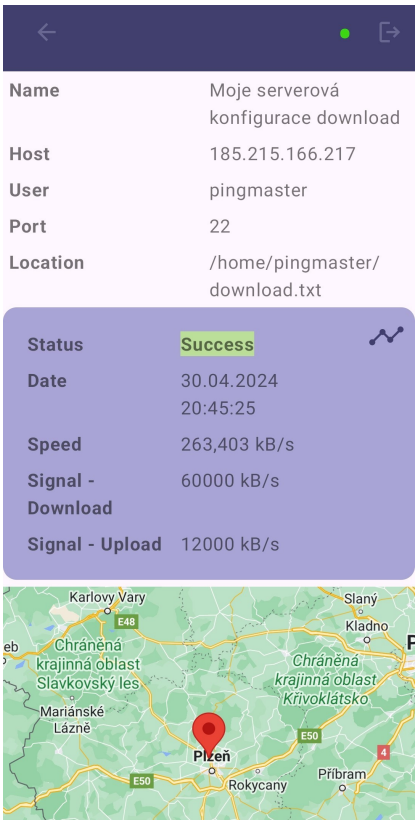


Obrázek A.8: Testovací sada pro protokol SSH



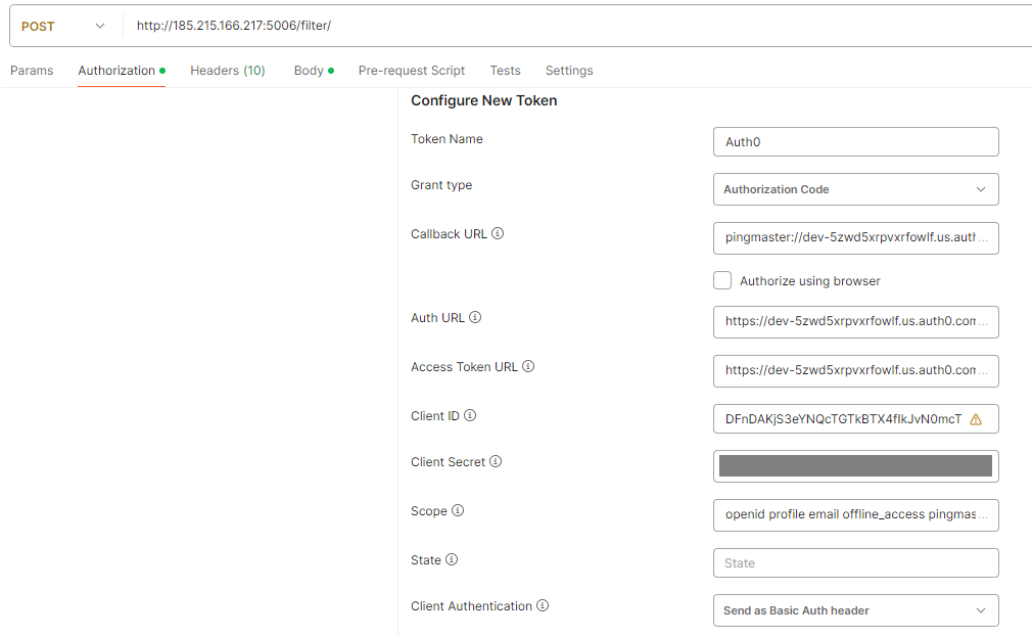
Obrázek A.9: Testovací sada pro protokol HTTP(s) a nástroj Ping

spuštěných testů pro danou konfiguraci, uživatel si je může prohlédnout na jednom místě s pomocí `slider` funkcionality.



Obrázek A.10: Výsledek testu pro protokol `SSH`

Druhou možností zobrazení je využití `API` v aplikaci `Postman`, tím získáme výsledky uložené na serveru. Nejprve je třeba nastavit `token` pomocí `Auth0`, kterým se k serveru lze připojit, viz obrázek A.11. Následně pomocí `POST` lze získat data ze serveru, jak je vidět na obrázku A.12. Je třeba do těla dotazu zadat jaké chceme výsledky měření `SFTP`, `PING`, či `HTTP`.



Obrázek A.11: **Token** pro připojení k serveru


```
{
    "ssh": [
        {
            "id": 3,
            "sshServerConfigurationId": 2,
            "sshTestCaseId": 5,
            "location": {
                "latitude": 49.78935666666666,
                "longitude": 13.972123333333332
            },
            "connectionSpeed": {
                "linkDownstreamBandwidthKbps": 60000,
                "linkUpstreamBandwidthKbps": 12000
            },
            "date": "2024-04-29T20:03:44.000+00:00",
            "code": 0,
            "configurationName": "Moje serverová konfigurace download ",
            "testCaseName": "Testovací sada ",
            "measuredTimeResultList": [
                {
                    "id": 829,
                    "testResultId": 3,
                    "bytes": 8192,
                    "nanoseconds": 53865538
                },
                {
                    "id": 830,
                    "testResultId": 3,
                    "bytes": 8192,
                    "nanoseconds": 1539
                },
                {
                    "id": 831,
                    "testResultId": 3,
                    "bytes": 8192,
                    "nanoseconds": 846
                },
                {
                    "id": 832,
```


Obrázek A.12: Výsledek dat z **API**

Bibliografie

[dev24a] DEVELOPER.ANDROID.COM. *ABI documentation*. developer.android.com, 2024. Dostupné také z: https://developer.android.com/ndk/guides/abis. ABI documentation.

[dev24b] DEVELOPER.ANDROID.COM. *Android Keystore*. developer.android.com, 2024. Dostupné také z: https://developer.android.com/privacy-and-security/keystore. Android Keystore.

[dev24c] DEVELOPER.ANDROID.COM. *Android Room Declaration*. developer.android.com, 2024. Dostupné také z: https://developer.android.com/training/data-storage/room. Android Room Declaration.

[Zin] ZINCIRCIOĞLU, Sena. *Android database compare*. medium.com. Dostupné také z: https://medium.com/huawei-developers/roomdb-vs-sqlite-exploring-database-options-for-android-development-1120151e6737. Android database.

[Big] BIGELOW, Stephen J. *APIs definitions*. techtarget.com. Dostupné také z: https://www.techtarget.com/searchapparchitecture/tip/What-are-the-types-of-APIs-and-their-differences. APIs.

[Cli24] CLIENT, Async HTTP. *Async HTTP Client*. AsyncHttpClient, 2024. Dostupné také z: https://github.com/AsyncHttpClient/async-http-client. Async HTTP Client.

[Aut24a] AUTH0. *Auth0 Access Token*. Auth0, 2024. Dostupné také z: https://auth0.com/docs/secure/tokens/access-tokens. Auth0 Access Token.

[Aut24b] AUTH0. *Auth0 Audience*. Auth0, 2024. Dostupné také z: https://community.auth0.com/t/what-is-the-audience/71414. Auth0 Audience.

[Aut] AUTH0. *Auth0 documentation*. Auth0. Dostupné také z: https://auth0.com/. Auth0 documentation.

[Aut24c] AUTH0. *Auth0 Scopes*. Auth0, 2024. Dostupné také z: https://auth0.com/docs/get-started/apis/scopes. Auth0 Scopes.

[Kum23] KUMAR, Satish. *Buffering in networks*. tutorialspoint.com, 2023. Dostupné také z: https://www.tutorialspoint.com/buffering-in-computer-network. Buffering in networks.

[DL10] DAI, Qin; LEHNERT, Ralf. Impact of Packet Loss on the Perceived Video Quality. In: *2010 2nd International Conference on Evolving Internet*. 2010, s. 206–209. Dostupné z DOI: 10.1109/INTERNET.2010.51.

[dev24d] DEVELOPER.ANDROID.COM. *Espresso: UI Testing Framework for Android*. 2024. Dostupné také z: https://developer.android.com/training/testing/espresso.

[doca] DOCKER.COM. *Docker container*. docker. Dostupné také z: https://www.docker.com/resources/what-container/. Docker container.

[tec] TECHTARGET.COM. *Docker image*. techtarget. Dostupné také z: https://www.techtarget.com/searchitoperations/definition/Docker-image. Docker image.

[docb] DOCKER.COM. *Docker overview*. docker. Dostupné také z: https://docs.docker.com/get-started/overview/. Docker overview.

[FP12] FERDOUS, Md. Sadek; POET, Ron. A comparative analysis of Identity Management Systems. 2012, s. 454–461. Dostupné z DOI: 10.1109/HPCSim.2012.6266958.

[Ker24] KERNER, Sean Michael. *FTP protocol definition*. techtarget.com, 2024. Dostupné také z: https://www.techtarget.com/searchnetworking/definition/File-Transfer-Protocol-FTP. FTP protocol definition.

[Ide] IDENTITY, Google Cloud. *Google Cloud Identity documentation*. Google. Dostupné také z: https://cloud.google.com/identity/. Google Cloud Identity documentation.

[Gou+02] GOURLEY, David; TOTTY, Brian; SAYER, Marjorie; AGGARWAL, Anshu; REDDY, Sailu. *HTTP: The Definitive Guide*. O'Reilly Media, Inc., 2002. Definitive Guides.

[Gra] GRADLE. *Gradle documentation*. Gradle. Dostupné také z: https://gradle.org/. Gradle documentation.

[bae24a] BAELDUNG. *H2 usage*. baeldung.com, 2024. Dostupné také z: https://www.baeldung.com/spring-boot-h2-database. H2 usage.

[Rao] RAO, Rohit. *IAM comparsion*. zluri.com. Dostupné také z: https://www.zluri.com/blog/identity-management-vs-access-management/. IAM comparsion.

[24] *ICMP protocol*. cloudflare, 2024. Dostupné také z: `https://www.cloudflare.com/learning/ddos/glossary/internet-control-message-protocol-icmp/`. ICMP protocol.

[Icm] ICMP4A. *Icmp4a documentation*. Icmp4a. Dostupné také z: `https://github.com/marsounjan/icmp4a`. Icmp4a documentation.

[Dug] DUGAN, Jon. *iPerf*. iperf. Dostupné také z: `https://iperf.fr/`. iperf.

[JSC] JSCH. *JSCH documentation*. www.libssh.org. Dostupné také z: `http://www.jcraft.com/jsch/`. JSCH documentation.

[Kan04] KANTAWALA, Anshul. Selecting the Buffer Size for an IP Network Link. *researchgate.net*. 2004.

[Key] KEYCLOAK. *Keycloak documentation*. Keycloak. Dostupné také z: `https://www.keycloak.org/`. Keycloak documentation.

[KR16] KUROSE, James F.; ROSS, Keith W. *Computer Networking: A Top-Down Approach*. 7. vyd. Pearson, 2016.

[Lap19] LAPUKHOV, Petr. Buffer Sizing Experiments at Facebook. *https://buffer-workshop.stanford.edu/papers/paper30.pdf*. 2019.

[Gid] GIDNEY, Craig. *Measurement of One Way Latency* [`https://cs.stackexchange.com/questions/602/measuring-one-way-network-latency`]. Accessed: 2015-11-15.

[cur24] CURL.SE. *LibCurl documentation*. curl.se, 2024. Dostupné také z: `https://curl.se/libcurl/`. LibCurl documentation.

[Liba] LIBSSH. *Libssh documentation*. www.libssh.org. Dostupné také z: `https://www.libssh.org/`. Libssh documentation.

[Libb] LIBSSH2. *Libssh2 documentation*. www.libssh2.org. Dostupné také z: `https://libssh2.org/`. Libssh2 documentation.

[Man23] MANASEER, Saher. Measuring Network Performance: Best Practices and Tools. *researchgate.net*. 2023.

[Vas23] VASAVA, Kaushal. *Android Retrofit*. medium.com, 2023. Dostupné také z: `https://medium.com/@KaushalVasava/retrofit-in-android-5a28c8e988ce`. Android Retrofit.

[dev24e] DEVELOPER.ANDROID.COM. *Module description*. developer.android.com, 2024. Dostupné také z: `https://developer.android.com/studio/projects`. Module description.

[Kri21] KRISHNA, Vamsi. *Network Tools*. techwiser.com, 2021. Dostupné také z: `https://techwiser.com/network-monitoring-apps-for-android/`. Network Tools.

[aut23] AUTH0.COM. *OAuth 2.0 Workflows*. auth0.com, 2023. Dostupné také z: https://auth0.com/docs/get-started/authentication-and-authorization-flow/which-oauth-2-0-flow-should-i-use. OAuth 2.0 Workflows.

[kon23] KONGHQ.COM. *OpenID vs OAuth 2.0*. konghq.com, 2023. Dostupné také z: https://konghq.com/blog/engineering/openid-vs-oauth-what-is-the-difference. OpenID vs OAuth 2.0.

[Pet24] PETERKA, Jiří. *Počítačové sítě*. 2024. Dostupné také z: http://jiri.peterka.cz/pocitacovesite.php3.

[Zol24] ZOLA, Andrew. *Ping*. TechTarget, 2024. Dostupné také z: https://www.techtarget.com/searchnetworking/definition/ping. Ping tool.

[Vil] VILLANUEVA, John Carl. *SCP/SFTP comparsion*. Dostupné také z: https://www.jscape.com/blog/scp-vs-sftp. SCP/SFTP comparsion.

[ssh] SSH.COM. *SFTP protocol documentation*. Dostupné také z: https://docs.ssh.com/manuals/server-admin/44/Using_Secure_File_Transfer__sftp2_.html. SFTP documentation.

[Sha21] SHARMA, Sourabh. *Modern API Development with Spring and Spring Boot*. Packt Publishing Limited, 2021.

[Sol] SOLARWINDS. *What Is Bandwidth in Networking?* SolarWinds, IT Glossary, [b.r.]. Dostupné také z: https://www.solarwinds.com/resources/it-glossary/network-bandwidth.

[gsm20] GSMA. *Speed TestBy Ookla*. gsma.com, 2020. Dostupné také z: https://www.gsma.com/get-involved/gsma-membership/gsma_resources/how-ookla-ensures-accurate-reliable-data-a-guide-to-our-metrics-and-methodology-updated-for-2020/. Network Tools.

[bae24b] BAELDUNG.COM. *OpenID vs OAuth 2.0*. baeldung, 2024. Dostupné také z: https://www.baeldung.com/spring-autowire. Spring Autowire.

[Boo24a] BOOT, Spring. *Spring Boot JPA Documentation*. spring.io, 2024. Dostupné také z: https://spring.io/projects/spring-data-jpa. Spring Boot JPA Documentation.

[Boo24b] BOOT, Spring. *Spring Boot Query Methods Documentation*. spring.io, 2024. Dostupné také z: https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html. Spring Boot JPA Documentation.

[Boo24c] BOOT, Spring. *Spring Boot Rest Documentation*. spring.io, 2024. Dostupné také z: `https://spring.io/projects/spring-restdocs`. Spring Boot Rest Documentation.

[alt23] ALTEXSOFT.COM. *OpenID vs OAuth 2.0*. altexsoft.com, 2023. Dostupné také z: `https://www.altexsoft.com/blog/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/`. SQL database comparsion.

[Tec24] TECHIE, Java. *SQL vs NoSQL*. medium.com, 2024. Dostupné také z: `https://medium.com/@javatechie/sql-vs-nosql-cbf7e147539b`. SQL vs NoSQL.

[SSH] SSHJ. *SSHJ documentation*. SSHJ. Dostupné také z: `https://github.com/hierynomus/sshj`. SSHJ documentation.

[SF11] STEVENS, W. Richard; FALL, Kevin R. *TCP/IP Illustrated, Volume 1: The Protocols*. 2. vyd. Addison-Wesley Professional Computing Series, 2011.

[Plo18] PLONUS, Uwe. *Time measurement using CPU*. stackoverflow.com, 2018. Dostupné také z: `https://stackoverflow.com/questions/48697024/time-measurements-cpu-ticks-and-adjustable-cpu-frequency`. Time measurement using CPU.

[She24] SHELDON, Robert. *Traceroute definition*. techtarget.com, 2024. Dostupné také z: `https://www.techtarget.com/whatis/definition/traceroute`. Traceroute definition.

Seznam obrázků

Seznam tabulek

Seznam výpisů