**Univerzita Hradec Králové**
**Fakulta informatiky a managementu**
**Katedra informatiky a kvantitativních metod**

# Rekonfigurace aplikace na základě kontextu uživatele

Bakalářská práce

Autor: Přemysl Pazderka
Studijní obor: Informační management

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové květen 2019

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 16.4.2019 Přemysl Pazderka

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Ing. Filipu Malému, Ph.D. za metodické vedení práce a vstřícné rady při jejím vypracování.

# Anotace

Bakalářská práce se věnuje možnostem rekonfigurace mobilní aplikace na platformě Android. Hlavním cílem práce je prozkoumat a popsat metody, které na základě uživatelského kontextu umožňují provádět změny v konfiguraci mobilní aplikace na této platformě. Souvisejícím cílem je popsané postupy rekonfigurace demonstrovat na ukázkové aplikaci a navrhnout rámec, který definované postupy zastřeší a umožní jejich znovupoužití v jiných oblastech a oborech. Vlastní řešení aplikace bude navrženo s ohledem na možné hrozby, které z užívání mobilního zařízení plynou, proto bude v práci věnována značná pozornost prozkoumání možností zabezpečení aplikací na platformě Android.

Metodika zpracování spočívá v prostudování odborné, renomované literatury, která se věnuje vždy konkrétnímu řešení problému v oblasti, která je relevantní k problematice rekonfigurace, a na základě poznatků, které z tohoto zkoumání vzejdou, je vyvinuta finální aplikace.

# Annotation

## Title: Reconfiguration of application based on user's context

Bachelor thesis aims to study options of reconfiguration of mobile application running on Android platform. Main goal of the work is to analyze and describe methods, which based on user's context, enable changes in configuration on mobile application. Next goal is to demonstrate analyzed methods of reconfiguration on sample application and propose a framework, which covers all the procedures of reconfiguration and allows their reuse in other scientific disciplines. Development of the application is being done with the respect of possible threats, that come up with using the mobile application. That's why the Bachelor thesis aims to analysis of options of the security on Android platform.

The methodology in this Bachelor thesis is formed with studying professional, renowned literature, which always solves the exact problem in the field, which is relevant to the reconfiguration. With the knowledge, that comes from this studying, is developed the final application.

# Obsah

# Seznam obrázků

# Seznam tabulek



# Seznam kódu

# 1 Úvod

Problém rekonfigurace mobilních aplikací spočívá v nedostatečně zvládnuté optimalizaci výkonu těchto zařízení. Pokud je metoda rekonfigurace zvolená nevhodným způsobem, může načítání nových dat, které je iniciováno uživatelem, významně vytěžovat mobilní zařízení. Zásadní otázkou pro tuto práci je proto zjištění, jaká z metod rekonfigurace, které mohou být aplikovány na mobilním zařízení s platformou Android, je nejvhodnější z hlediska optimalizace výkonu. Související otázkou je i management rizik mobilních zařízení, protože platforma Android je snadno napadnutelná při neobezřetnosti koncového uživatele nebo při nevhodné konstrukci aplikací.

Hlavním cílem této práce je popis metod rekonfigurace, které je možné realizovat na mobilním zařízení s platformou Android. S tímto cílem souvisí i implementace mobilní aplikace na platformě Android, která realizuje stanovenou metodu rekonfigurace. Aplikace je navržena s ohledem na maximální znovupoužití, kdy ve svém jádru bude využívat API (application programming interface) třetí strany, které obstarává ekonomická data. S ohledem na rozsah a časové možnosti této práce nebude implementován vlastní server, který je nahrazen zmiňovaným API. API cizí strany se dá kdykoliv nahradit vlastním serverem, případně jiným, vhodnějším API, tudíž aplikace bude maximálně znovupoužitelná.

Základní hypotézou při počátečním stadiu vypracovávání práce je existence právě jednoho optimálního řešení pro rekonfiguraci na daném mobilním zařízení. Pro posun v práci je klíčové zjistit, jakým způsobem se aktuálně řeší problematika rekonfigurace mobilních aplikací v zahraničí. Na základě tohoto zjištění bude aplikován vlastní teoretický postup pro vývoj výsledné aplikace. Tato metodika zpracování je zvolena z důvodu vysoké míry odbornosti zahraničních renomovaných zdrojů, proto je možné na základě teoretických poznatků, získaných z této literatury, vyvinout aplikaci, která bude maximální způsobem reflektovat požadavky na rekonfiguraci mobilních aplikací na platformě Android. Realizovaná metodika má i svá omezení, například v tom, že výčet problémů, které odborná literatura řeší, není vyčerpávající a může existovat i způsob, který nebude v rámci literatury prostudován, a který by mohl být vhodnější než autorem realizovaný

postup řešení. Na druhou stranu je vhodnější uvádět jen takové množství teorie, které je nezbytné k pochopení dané problematiky, aby mohla být nosná část práce věnována praktickému vývoji aplikace. Výsledná aplikace by měla usnadňovat práci ekonomickým poradcům a analytikům korporátních firem, kteří potřebují v reálném čase znát hodnoty ekonomických ukazatelů, aby tyto hodnoty mohli využívat při komunikaci s klienty, případně tvorbě firemní strategie.

Důvod výběru tématu rekonfigurace vychází z autorova zájmu o vývoj mobilních aplikací pro platformu Android a o programování jako celek, kdy autor se primárně zaměřuje na vývoj fullstack aplikací (tj. aplikací, které implementují server (backend), uživatelské rozhraní (frontend) a databázi)) v programovacích jazycích Java a Javascript.

# 2 Metody rekonfigurace

Rekonfigurací aplikace na mobilním zařízení je v kontextu práce myšlena aktivní změna dat mobilní aplikace na platformě Android, s ohledem na uživatelský kontext, kdy požadavek na tuto změnu se dle zvolené metody rekonfigurace aplikuje a aktualizovaná data se zobrazí uživateli.

## 2.1 Uživatelský kontext

Uživatelský kontext je pro potřeby práce rozdělen na aktivní a pasivní, kdy aktivní uživatelský kontext znamená záměrně iniciovanou změnu uživatelem dané mobilní aplikace. Uživatel si sám zvolí, která data chce zobrazovat, a která nikoliv.

Pasivním uživatelským kontextem jsou myšleny okolnosti, které uživatel nemůže přímo ovlivnit, například jeho umístění. Podle umístění uživatele je možné v mobilní aplikaci provádět lokalizaci, což znamená jazykové přizpůsobení s ohledem na předpokládaný mateřský jazyk uživatele. Za pasivní uživatelský kontext se dají označit i informace, které bude potřebovat výsledná aplikace pro práci pozadí (např. identifikátor přihlášeného uživatele).

## 2.2 Java a Android

Java je všestranný programovací jazyk, který řeší celou řadu oblastí vývoje software, přičemž jednou z těchto oblastí je i vývoj mobilních aplikací.

Ačkoliv to není podmínkou, většina aplikací pro platformu Android je v současné době vyvíjena v jazyce Java (Allen, 2013). Proto je vhodné i aplikaci, realizující metodu rekonfigurace, vyvinout v jazyce Java, aby byla zajištěna maximální podpora a využitelnost.

Android je platforma, která je v současné době dominantní v použití na chytrých zařízeních, jakou jsou mobilní telefony a tablety. Platforma Android byla původně navržena a vyvíjena společností Android Inc., později byla převzata a nadále budována společností Google. (Altuwaijri, Ghouzali, 2018). Aby mohl být program, napsaný v programovacím jazyce Java, spuštěn na mobilním zařízení s platformou Android, musí mobilní zařízení vědět, jak daný program přeložit. K tomuto účelu používá mobilní zařízení překladač (compiler).

Překladač říká systému, jakým způsobem má přeložit instrukce od uživatele (zdrojový kód), aby je mohl následně zpracovat procesor výpočetního zařízení. (Burd, 2017). Poté, co jsou data předána překladači, jsou přeložena do byte kódu, kterému chytré zařízení rozumí a instrukce vykoná prostřednictvím virtuálního počítače, označovaného jako DalvikVM. (Sawada, Arai, Ootsu, Yokota, Ohkawa, 2017). Uvedeným postupem je zakončen cyklus inicializace a průběhu operace aplikace na platformě Android.

### 2.2.1 Kotlin

Kotlin je jazykem, který s vývojem aplikací na platformě Android úzce souvisí, protože vznikl jako určitá nástavba jazyka Java, a je s tímto jazykem plně kompatibilní. Kotlin byl vyvinut společností JetBrains, která vyvíjí i vlastní sadu nástrojů, do kterých patří vývojové prostředí IntelliJ IDEA nebo textový editor Webstorm. Na platformě IntelliJ IDEA je postaveno i Android studio, které oficiálně vlastí společnost Google. Kotlin lze i přes odlišnou syntaxi chápat jako modernější jazyk Java, který je přizpůsobený dnešním požadavkům vývoje software a nezatěžují ho problémy s vývojem v minulosti. Pro vývoj aplikace na platformě Android v rámci této práce bude využito základní konstrukce jazyka Java, protože pro potřeby vyvíjené aplikace je tento jazyk dostačující a není nutné integrovat nástavby, jako je Kotlin, ačkoliv při tvorbě rozsáhlejších projektů bývá použití Kotlinu na místě.

### 2.2.2 Vývojové prostředí

Zásadním předpokladem pro realizaci vybrané metody rekonfigurace je volba vývojového prostředí. Pro tuto práci je zvoleno vývojové prostředí Android studio, ve kterém se vyvíjejí mobilní aplikace pro platformu Android. Existují i alternativy v podobě jiných vývojových prostředích, jako je například Eclipse, Netbeans nebo zmiňované IntelliJ IDEA, nicméně tyto jsou primárně určeny pro vývoj aplikací pro počítače nebo web a pro vývoj Android aplikací je nutné instalovat sadu pluginů, a také emulátor. Pluginy jsou balíčky, které určitým způsobem rozšiřují funkčnost vývojového prostředí, aby bylo možné jej přizpůsobit platformě Android. Emulátor je v rámci vývoje aplikací na platformě Android více než nutný,

protože simuluje mobilní zařízení, je tedy možné napsaný kód ihned spustit a zkontrolovat, zda funguje správně na zvolené konfiguraci mobilního zařízení. Android studio má tyto vlastnosti zabudované a zaměřuje se primárně na tvorbu mobilních aplikací na této platformě, proto výsledná aplikace bude v případě tohoto vývojového prostředí disponovat největším funkčním potenciálem.

Android studio má i značné nevýhody, jako jsou vysoké nároky na RAM zařízení, na kterém je spuštěno, proto je problematické pomocí Android studia vyvíjet mobilní aplikace na zařízeních s menší kapacitou paměti RAM, než jsou 4 GB.

#### Gradle

Postupným vývojem požadavků na kvalitu programování došlo k nutnosti automatizace při sestavení kódu a jeho následného spuštění. Jeden z nástrojů, který je možno k tomuto využít, je Gradle, který bude figurovat i v rámci vyvíjené aplikace. Funkčnost nástroje Gradle není pro potřeby vývoje aplikace v této práci příliš významná, proto je zde zmíněn pouze pro úplnost.

### 2.2.3 Application programming interface (API)

Application programming interface (API) obecně představuje ucelený soubor definovaných metod, rozhraní, protokolů nebo frameworků, které může vývojář používat za účelem zjednodušení a zpřehlednění svojí práce. V případě této práce bude API představovat rozhraní, jako zdroj pro aktuální ekonomické ukazatele, které budou v rámci vyvíjené aplikace zobrazovány uživateli mobilního zařízení, využívající platformu Android. API slouží jako vhodný nástroj pro vytvoření znovupoužití, protože data z něho získaná se dají použít v jakémkoliv kontextu, který je v aplikaci vytvořen.

#### Přístupy k API

API obvykle zasílá data dle požadavku ze strany klienta. V některých případech je přístup k API limitován a je nutná určitá míra ověření, například je nutné se zaregistrovat na domovské stránce, která API poskytuje a vygenerovat si přístupový API klíč, který umožní data získat. V některých případech je API veřejné, protože neobsahuje žádná citlivá ani jinak zneužitelná data, proto je možné se k nim dostat bez API klíče.

### Formát dat API

V praxi existuje celá řada formátů, ve kterých lze data z API vracet, při vývoji a potažmo využívání API je vhodné dbát na to, aby API podporovalo základní a nejvyužívanější formáty, kterými jsou dle (Sandoval, 2016) XML, JSON a YAML. XML zkratka znamená Extensible Markup Language a poskytuje flexibilitu při výměně a integraci dat napříč aplikacemi. Na druhou stranu lze XML vytknout sémantiku dat, která jsou tímto formátem přenášena (Hartmann, Link, 2010).

XML se dá označit za přehledný a specifický formát, ale při vývoji rozsáhlejší aplikace se může jevit jako hůře udržovatelný, právě z důvodu své vyšší specifičnosti a sdílnosti informací. Ačkoliv se XML dá označit za starší formát, je stále využíván, například při vývoji aplikace pro platformu Android ho lze využít pro konfiguraci aplikace. Příklad XML kódu je uveden v kódu 1 níže.

```
<xs:element name="user" type="lettertype"></xs:element>
<xs:complextype name="lettertype" mixed="true">
 <xs:sequence>
  <xs:element name="usertype" type="xs:string"></xs:element>
  <xs:element name="userage" type="xs:positiveInteger"></xs:element>
  <xs:element name="regdate" type="xs:date"></xs:element>
 </xs:sequence>
</xs:complextype>
```

**Kód 1 Příklad XML formátu**
Zdroj: Převzato z (Sandoval, 2016)

JSON zkratka znamená Javascript Object Notation, přičemž se jedná o moderní formát dat, který je využíván celou řadou programovacích nástrojů. Oproti XML se jeví JSON jako méně specifický, protože poskytuje menší množství informací o přenášených datech. JSON je zároveň méně přehledný při vyšším množství dat, jak je patrné z poskytnutého příkladu v kódu č. 2. JSON se hodí pro vývoj aplikací, které jsou napsané v programovacím jazyce Javascript, protože na rozdíl od XML se mnohem lépe analyzuje (parsuje), jelikož JSON již sám o sobě vrací připravený objekt bez nutnosti parsování, což je jeho hlavní předností. U JSON formátu je důležitá jeho validita, je nutné, aby všechna data byla umístěna v kořenových složených závorkách. Je rovněž nutné používat dvojité uvozovky. Uvozovky jednoduché nejsou validním formátem JSON. Jednotlivá data musejí být oddělena čárkami. Příklad JSON kódu je uveden v kódu 2 níže.

```
{
 "title": "Age Gate",
 "type": "object",
 "properties": {
   "firstName": {
     "type": "string"
   },
   "knownValue": {
     "type": "boolean"
   },
   "age": {
     "description": "Age in years",
   "type": "integer",
   "minimum": 18
   }
 },
 "required": ["firstName", "lastName"]
}
```

**Kód 2 Příklad JSON formátu**
Zdroj: Převzato z (Sandoval, 2016)

Zkratka YAML znamená YAML ain't markup language, přičemž se jedná o kompromis mezi výše popsanými formáty XML a JSON. YAML se snaží o maximální přívětivost vůči člověku, přičemž je čitelný, lehce udržovatelný a nezabírá příliš mnoho místa.

Příklad formátu YAML je uveden v následujícím kódu 3.

```
# failover url
url_403: /
#url_403: http://example.org/underage

# snippet definition
snippet_enter: /templates/verified.html.twig
snippet_exit:  /templates/underageexit.html.twig
snippet_403: /templates/validate.html.twig

# minimium age config
min_age: 18

# container variable
width: 300
height: 300

# container bg
overlay: '#ffffff'
```

**Kód 3 Příklad YAML formátu**
Zdroj: Převzato z (Sandoval, 2016)

## 2.3 Dynamická rekonfigurace

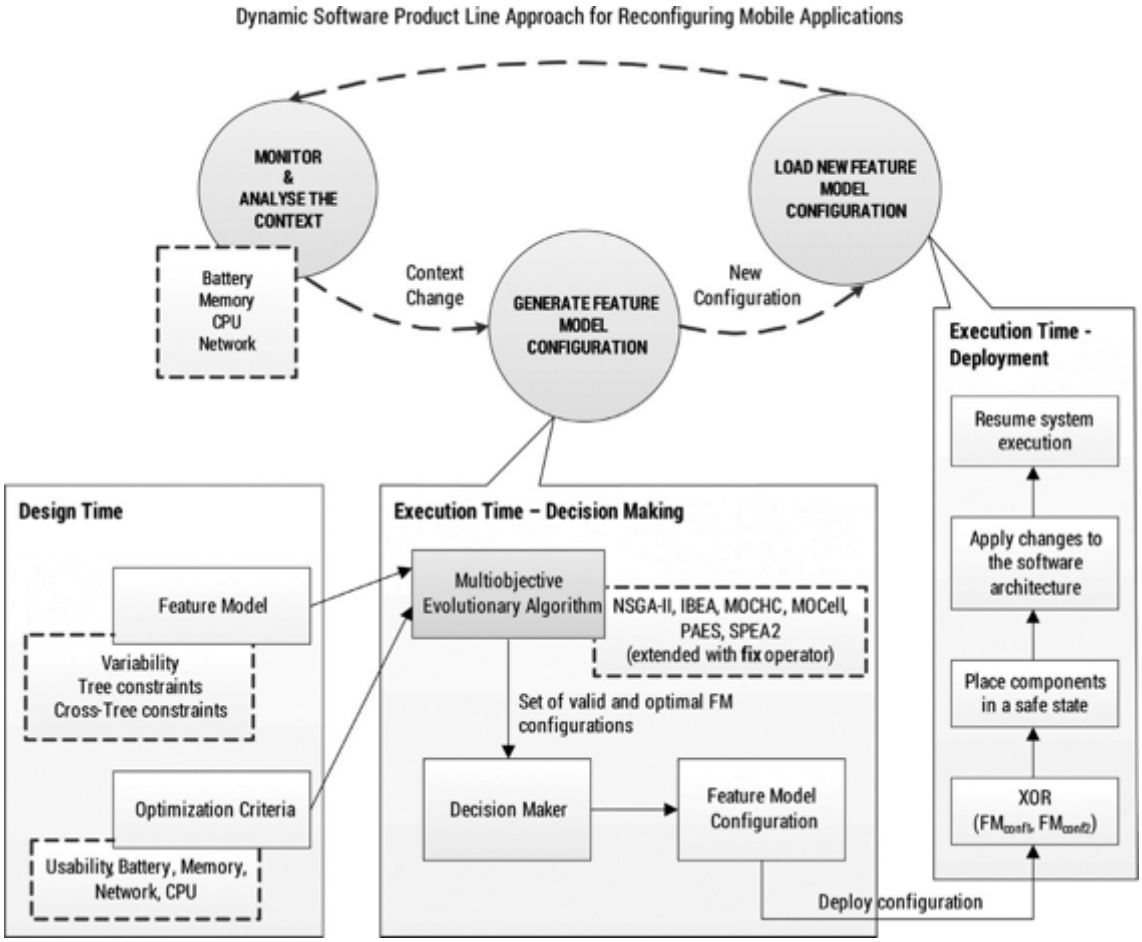
Aby mohla být rekonfigurace na mobilním zařízení s platformou Android prováděna efektivně, je nutné, aby aplikace uměla při svém běhu aktivně měnit zobrazovaná data. K tomu je nejvhodnější zvolit dynamickou rekonfiguraci.

Aplikace je následně schopna rekonfigurace i v případě, že jsou spuštěna paralelní vlákna. (Sawada, Arai, Ootsu, Yokota, Ohkawa, 2017). Dynamickou rekonfigurací je zajištěno znovupoužití a škálovatelnost. (Schmitt, Orfgen, Zühlke, 2015). Tyto vlastnosti vyplývají z dynamického chování aplikace. Je možné ji libovolně rozšiřovat, protože prostředí aplikace se vždy přizpůsobí změněným podmínkám. Pro znovupoužití je nejvhodnějším postupem navrhnout rámec, který bude celou aplikaci zastřešovat a umožňovat tím znovupoužití při minimální práci s přeprogramováním klíčových komponent.

Řada autorů nestaví všechny metody rekonfigurace na stejnou úroveň důležitosti, ale naopak tvrdí, že dynamická rekonfigurace je nutná u každé aplikace. (Pascual, Lopez-Herrejon, Pinto, Fuentes, Egyed, 2015). Při této koncepci se vždy

vyvine aplikace s ohledem na dynamickou rekonfiguraci, a teprve poté je implementována další metoda rekonfigurace.

Například s ohledem na aktuální výdrž baterie mobilního zařízení nebo kvalitu služeb při daném připojení. (Pascual, Lopez-Herrejon, Pinto, Fuentes, Egyed, 2015). Dynamická rekonfigurace je v současné době využívána ve všech druzích aplikací co do své velikosti, proto vznikly i schématické architektury, které jsou na dynamické rekonfiguraci postavené, jako je architektura autorů (Pascual, Lopez-Herrejon, Pinto, Fuentes, Egyed, 2015), která obsahuje veškeré kroky, které je nutné realizovat při dynamické rekonfiguraci. Na schématu jsou přehledně zobrazeny vrstvy, které spolu musejí při dynamické rekonfiguraci komunikovat, aby ji bylo možné efektivně realizovat a předešlo se problémům s výkonem, kterými aplikace, realizující dynamickou rekonfiguraci trpí v důsledku nutnosti neustálého načítání a kontroly nových a zobrazovaných dat.




**Obr. 1 Schéma dynamické rekonfigurace**
Zdroj: Pascual, Lopez-Herrejon, Pinto, Fuentes, Egyed, 2015

### 2.3.1 RESTful služby

Při dynamické rekonfiguraci aplikací na mobilních zařízeních je možné v kombinaci s API využít RESTful služeb. V rámci této problematiky je nutné rozlišovat mezi pojmy REST a RESTful. Zkratka REST (representational state transfer) představuje architekturu, využívající služeb webu. Pojmem RESTful se rozumí služby webu, které danou architekturu reprezentují. Tyto služby jsou reprezentovány HTTP požadavkem, který může mít podobu GET, POST, DELETE, PUT nebo PATCH. Každé API nemusí podporovat všechny metody, záleží vždy na tom, jak je API konstruované.

Zmíněné metody jsou vhodné zejména pro mobilní zařízení, které mají menší výkon, protože vyžadují menší využití zdrojů chytrého zařízení (Schmitt, Orfgen, Zühlke, 2015). Požadavek na některou ze služeb je typicky vyvolaný uživatelskou událostí, jako je kliknutí na tlačítko v uživatelském rozhraní nebo odeslání formuláře. Při tomto přístupu je značnou nevýhodou situace, kdy aplikace nebude schopna získat internetové připojení kvůli chybějícímu WI-FI signálu nebo chybě při připojení. V takovém případě nedojde k načtení dat a aplikace zůstane prázdná. Tento problém by byl řešitelný uchováváním většího množství dat v paměti chytrého zařízení, kdy v případě výpadku by se načetla poslední známá data, ale před samotnou realizací tohoto procesu je vhodné zvážit jeho nutnost, protože mobilní zařízení často disponují značně limitovanou pamětí.

#### GET požadavek

Vrací stav zdroje, identifikovaného URL adresou. (Kobusińska, Hsu, 2018) Pokud požadavek proběhne v pořádku, vrátí spolu s daty status 200, což v překladu znamená úspěšně vykonaný požadavek. Existuje celá řada serverových odpovědí, které mohou nastat, z nichž nejpodstatnější jsou odpovědi 200 (požadavek proběhl v pořádku), 400 (interní chyba serveru) a 404 (adresa požadavku nebyla nalezena).

#### POST požadavek

Po tomto požadavku dojde k vytvoření nového zdroje. (Kobusińska, Hsu, 2018). Typicky jsou tímto požadavkem zasílána nová data od uživatele do uživatelského rozhraní aplikace (frontendu), kde dojde k jejich validaci, případně

uložení do databáze. Při jeho realizaci je nutná implementace bezpečnostních prvků, viz následující podkapitola DELETE požadavek.

#### DELETE požadavek

Delete požadavek funguje obdobě jako požadavek POST, kdy dojde k vymazání požadovaných dat s ohledem na URL. (Kobusińska, Hsu, 2018) K jeho provedení, stejně tak jako při požadavku POST, je nutné implementovat bezpečnostní opatření, protože pokud by požadavek provedla neautorizovaná aktivita, mohla by zásadně poškodit aplikací uchovávaná data. Typicky se totiž po DELETE požadavku provede nějaká operace nad databází, která je s aplikací spojena.

#### PUT požadavek

Tento požadavek je podobný požadavku POST, protože umí rovněž vytvářet nová data. Rozdíl oproti POST požadavku spočívá v tom, že PUT požadavek nejprve ověří, zda data, která chce uživatel vytvořit, již neexistují. Pokud ano, poté pouze upraví jejich hodnoty. Pokud ne, vytvoří zcela nová data.

#### PATCH požadavek

Požadavek PATCH slouží výhradně k upravování již existujících dat. Pokud neexistují již vytvořená data, která má tento požadavek upravit, pak operace selže. Aby bylo možné data upravit, musí mít požadavek přesně specifikované vzory, podle nichž se úprava provede.

### 2.3.2 Problém s výkonem

Dynamická rekonfigurace je vhodná pro separátní mobilní zařízení, pokud ale dojde k požadavku na propojení více mobilních zařízení za účelem komunikace, nastane problém s exponenciálně rostoucím zpomalením systému daného zařízení.

Tento problém je řešitelný použitím Wi-fi sítě jako komunikačního kanálu (Sawada, Arai, Ootsu, Yokota, Ohkawa, 2017). Wi-fi je dostupná na téměř jakémkoliv místě, a zároveň poskytuje vhodný komunikační standard v kombinaci s nejlepším výkonnostním poměrem v oblasti metod bezdrátové komunikace. Problém s výkonem nastane i v situaci, kdy aplikace generuje velké množství požadavků a

mechanismus rekonfigurace je v krátkých časových intervalech pravidelně opakován. Pro takový případ je možné využít některý z přístupů distribuované rekonfigurace, která tento problém řeší, jak uvádí (Vasconcelos, Vasconcelos, Endler, 2016)

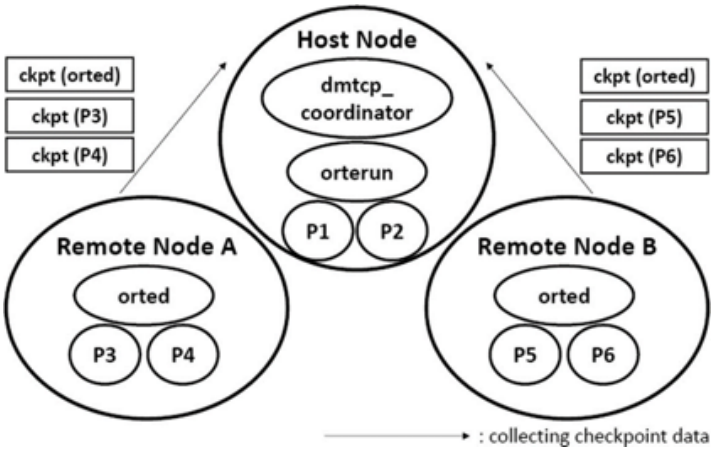
### 2.3.3 Postup dynamické rekonfigurace

Postup při dynamické rekonfiguraci je možné rozčlenit celkem do tří sekvenčně následujících kroků, které se svým obsahem zásadně liší s ohledem na kontext aplikace a na problematiku, kterou aplikace řeší. Prvním krokem je analyzování kontextu konkrétního problému, který je prostřednictvím rekonfigurace řešen. Následuje vytvoření algoritmu, který se spustí pokaždé, když je potřeba realizovat rekonfiguraci. Posledním krokem je zobrazení dat, které byly v předchozím stadiu pomocí algoritmu načteny. Ve studii, zabývající se vývojem algoritmů pro rekonfiguraci mobilních zařízení (Pascual, Lopez-Herrejon, Pinto, Fuentes, Egyed, 2015) jsou jako výchozí data brána v potaz výdrž baterie mobilního zařízení, dostupná paměť a stav sítě. V kontextu této práce bude postup rekonfigurace obdobný, ale data, ke kterým bude přihlíženo, budou podléhat finální podobě programované aplikace, což bude zobrazování ekonomických ukazatelů. Proto v tomto případě budou klíčovými navíc i zdroje dat, přesnost zobrazovaných dat a perioda aktualizace dat.

## 2.4 Statická rekonfigurace

Statickou rekonfiguraci je možné využívat v případech, kdy je nutné monitorovat aktuálně dostupná zařízení v síti, aby bylo možné navazovat komunikaci. Statická rekonfigurace zajistí přesun dat, které zpracovávalo mobilní zařízení, které za určitých okolností bylo odpojeno ze sítě, a není tak možné navázat s ním kontakt, jak uvádí (Sawada, Arai, Ootsu, Yokota, Ohkawa, 2017). Ostatní zařízení tak nemusejí čekat, až dané zařízení dokončí svoji úlohu, protože tato je přenesena na jiné zařízení, které je v síti aktuálně dostupné. Statická rekonfigurace svojí povahou odpovídá aplikacím velké komplexity a složitosti, protože právě tyto aplikace budou provázány přes řadu síťových uzlů, které jsou náchylné k výpadkům. Síťové uzly si je možné představit jako jednotlivá mobilní zařízení, která mají

přístup do sítě a je na nich spuštěna daná mobilní aplikace, kdy schematicky odpovídá rozvržení uzlů následujícímu obrázku ze studie autorů (Sawada, Arai, Ootsu, Yokota, Ohkawa, 2017), kde je sice primárně řešena dynamická rekonfigurace, ale práce obsahuje i schéma popisující obecné fungování síťových uzlů.




**Obr. 2 Schéma síťových uzlů**
Zdroj: Sawada, Arai, Ootsu, Yokota, Ohkawa, 2017

Na obrázku je možné vidět celkem tři síťové uzly, které jsou reprezentovány koncovými zařízeními uživatelů. Pokud je jeden z uzlů odpojen, musejí ostatní uzly převzít jeho funkci, popřípadě musí aplikace nouzově dodat maximálně přesná data, která má uložená.

### 2.4.1 Nedostupné připojení

Dle autorů (Sawada, Arai, Ootsu, Yokota, Ohkawa, 2017) je možné pomocí statické rekonfigurace zajistit přesun dat ze zařízení, které bylo odpojeno ze sítě. Tohoto mechanismu by se dalo využít i v případě, že mobilní zařízení nemá přístup k internetu a nebude moci pomocí API načíst nová data. V takovém případě lze pomocí statické rekonfigurace naprogramovat obdobný mechanismus, který je implementován ve studii autorů (Thanigaivelan, Nigussie, Hakkala, Virtaen, Isoaho, 2018), kdy tento mechanismus umožní statickou rekonfiguraci, aby si aplikace průběžně uchovávala data z API a při nemožnosti připojení k síti (internetu) načetla naposledy uchovaná data.

### 2.4.2 Postup statické rekonfigurace

Při realizaci statické rekonfigurace je zásadní si uvědomit, zda vyvíjená aplikace skutečně možnost této rekonfigurace potřebuje. Pokud se aplikace vyvíjí tzv. jako „standalone“, tedy tak, že je sama sobě soběstačná, bez potřeby implementace vnějších zdrojů, není třeba statickou rekonfiguraci implementovat, protože aplikace nevyžaduje získávání dílčích dat ze sítě, ale všechna potřebná data má lokálně umístěné na zařízení, na kterém je instalována. Druhým krokem je implementace mechanismu, který umožní získat statická data, která jsou potřebná pro vyvíjenou aplikaci, například prostřednictvím API, databáze, vlastního serveru nebo jiných zdrojů. Třetím krokem je implementace získaných dat do aplikační logiky vyvíjené aplikace, aby se tato data zobrazovala vždy, když není dostupné připojení k internetu nebo toto připojení selže.

## 2.5 Distribuovaná rekonfigurace

Distribuovaná rekonfigurace je využívána při řešení problémů, které vznikají při rekonfiguraci dynamické. Aplikace, založené na dynamické rekonfiguraci, běží v dlouhých časových úsecích, a proto je obtížné provádět v nich změny v reálném čase, aby nedošlo k poškození koncových uživatelů aplikace. Řešení spočívá v distribuované rekonfiguraci, což je z vývojářského hlediska konkurentní (paralelní) programování, kdy aplikace dokáže současně spouštět více vláken.

### 2.5.1 Konkurentní vývoj

V případě potřeby změnit rekonfiguraci dojde ke spuštění nového (paralelního) vlákna, které obstarává aktuální a plynulý chod původní aplikace (Vasconcelos, Vasconcelos, Endler). Zatímco nová aplikace se vyvíjí a testuje v rámci nově vytvořeného vlákna, stávajícím uživatelům aplikace nic nebrání v jejím používání, protože ta běží v nezměněné podobě ve druhém vlákně. Tento přístup je náročnější na zdroje, protože vyžaduje vyšší porozumění problematice rekonfigurace a vyšší počet spolupracujících prostředků v rámci aplikace. Je vhodnější pro velké firmy, které si nemohou dovolit zastavit provoz dané aplikace a potřebují v ní změny provádět za běhu. Distribuovanou rekonfigurací se rovněž

rozumí například aktivita, kdy uživateli je zobrazena obrazovka s daty, ale aplikace provádí ještě další změny na pozadí.

### 2.5.2 Postup distribuované rekonfigurace

Distribuovaná rekonfigurace se realizuje v případech, kdy je finální aplikace hotová a je nasazená v běžném provozu. Pokud jde o případ, kdy je nutné spouštět více paralelních vláken pro funkčnost aplikace, realizuje se při samotném vývoji aplikace. Dále je nutné vytvořit paralelní vlákno k této aplikaci, kde probíhají plánované změny a mohou být zároveň otestovány bez ovlivnění aplikace, která je dostupná koncovým uživatelům. Až jsou změny, prováděné paralelně, dokončeny a otestovány, je nutné aplikaci na krátký časový úsek zastavit, aby mohla být smazána, pokud je nutné nahrazení nosných částí této aplikace.

## 2.6 SWOT analýza metod rekonfigurací

Pro vyhodnocení metod rekonfigurace je nutné využít hodnotícího nástroje, který umožní efektivně a přehledně zachytit výhody a nevýhody zmíněných metod rekonfigurace. Protože SWOT analýza je primárně určena jako nástroj pro strategické plánování (Phadermrod, Crowder, Wills, 2017), je vhodné ji využít i pro vyhodnocení metod rekonfigurace, protože ty mají strategický význam pro budoucí podobu vyvíjené aplikace. SWOT analýza vychází primárně ze čtyř kritérií, pomocí kterých budou výsledné metody rekonfigurace hodnoceny, jimiž jsou silné stránky, slabé stránky, příležitosti a hrozby.

Silnými stránkami se rozumí přednosti metody rekonfigurace, které ji zvýhodňují oproti ostatním metodám. Slabými stránkami jsou míněny oblasti, kde daná metoda rekonfigurace oproti ostatním metodám zaostává nebo má určité nedostatky, které mají vliv na výslednou podobu vyvíjené aplikace. Příležitostmi se v tomto případě rozumí možnosti rekonfigurace pro zlepšení, díky kterým by se metoda stala použitelnější v dané oblasti a poskytovala by větší pohodlí koncovým uživatelům. Hrozbami metod rekonfigurace je myšleno nebezpečí, které vzniká při jejich užívání z pohledu uživatele i vývojáře. Nebudou pokryty veškeré bezpečnostní hrozby, které se mohou v rámci dílčích metod rekonfigurací objevit, protože ty detailně řeší kapitola management rizik na platformě Android.

### 2.6.1 Vyhodnocení dynamické rekonfigurace

Pro vyhodnocení dynamické rekonfigurace byly zjištěny následující charakteristiky:

**Tabulka 1 Charakteristiky dynamické rekonfigurace**

| SILNÉ STRÁNKY | SLABÉ STRÁNKY | PŘÍLEŽITOSTI | HROZBY |
|---|---|---|---|
| Škálovatelnost | Výkon | Optimalizace | Pád aplikace |
| Zabezpečení | Připojení k síti | off-line režim | Bezpečnostní útok |
| Podpora | Redundance | Stabilita | Neudržovatelnost |
| Znovupoužití | Přetížení | Interoperabilita | Složitost |

**Zdroj**: vlastní zpracování

První silnou stránkou dynamické rekonfigurace je její škálovatelnost, což znamená možnost rozšiřitelnosti vyvíjené aplikace o nové komponenty, které mohou být jednoduše přidány do uživatelského rozhraní aplikace a není nutné složité přeprogramování funkčnosti, pouze napojení komponent na aplikační logiku mobilní aplikace. Zabezpečení souvisí s využíváním RESTful služeb, které v základu neakceptují požadavky z jiných uzlů než z těch ověřených. Podpora vychází z převažujícího množství aplikací, které jsou pomocí dynamické rekonfigurace vyvinuté, protože téměř každá mobilní aplikace využívá určitou metodu dynamické rekonfigurace. Znovupoužití vychází z možnosti přidávání nových, znovupoužitelných komponent, jejichž množství neustále roste z důvodu zvyšující se tendence využití a zlepšování metody dynamické rekonfigurace.

Slabou stránkou je výkon, který může kolísat při častých požadavcích na rekonfiguraci aplikace a jejích zobrazovaných dat. Připojení k síti (internetu) je při dynamické rekonfiguraci vždy vyžadováno, pokud se tedy není možné připojit k síti, nebude ani rekonfigurace fungovat. Redundance a přetížení mohou nastat při neuváženém používání RESTful požadavků, které aplikaci zahltí.

Z nedostatků dynamické rekonfigurace vyplývají její příležitosti, z nichž nejzásadnější je optimalizace zmiňovaného výkonu a funkčnost off-line režimu při nemožnosti připojení k síti. Při vývoji rozsáhlejších aplikací je nutné dbát na co nejmenší vytížení, aby z důvodu nedostatečného výkonu nedocházelo k jejímu pádu. Interoperabilitou se rozumí schopnost aplikace komunikovat s ostatními systémy a zařízeními, což není možné při off-line režimu.

Ze slabých stránek vyplývají hrozby ve formě neočekávaného chování aplikace, jako je pád aplikace, který se pojí s nedostatečným výkonem. Při realizaci RESTful požadavků na server je zásadní implementovat autorizační mechanismus, aby bylo možné předejít útokům a data ze serveru nebylo možné zaslat neověřenému zdroji. Neudržovatelnost aplikace může nastat při neuváženém redundantním použití nadbytečných komponent a požadavků na server, stejně tak jako složitost.

### 2.6.2 Vyhodnocení statické rekonfigurace

Pro vyhodnocení statické rekonfigurace byly zjištěny následující charakteristiky:

**Tabulka 2 Charakteristiky statické rekonfigurace**

| SILNÉ STRÁNKY | SLABÉ STRÁNKY | PŘÍLEŽITOSTI | HROZBY |
|---|---|---|---|
| Off-line funkčnost | Pružnost | Spolupráce | Nekonzistence |
| Škálovatelnost | Náročnost | Velké projekty | Prostředky |
| Přenesení výkonu | Definice pravidel | Efektivnost | Komplikovanost |
| Uchování dat | Persistence | Zabezpečení | Ztráta dat |

**Zdroj**: Vlastní zpracování

Off-line funkčnost je hlavní předností oproti dynamické rekonfiguraci, protože umožňuje provádění změn i v případě, že zařízení není aktuálně připojené k síti. Škálovatelnost souvisí s přenášením výkonu na ostatní uzly, které propojují aplikaci a zajišťují její funkčnost, což snižuje požadavky na výkon, které jsou kladeny na jednotlivá výpočetní zařízení. Uchovávání dat je pro zajištění statické rekonfigurace nutné realizovat lokálně na daném zařízení, protože jinak by nebylo možné při off-line režimu data zpracovávat.

Mezi slabé stránky statické rekonfigurace patří její nedostatečná pružnost, protože mechanismus statické rekonfigurace se aktivuje vždy za daných podmínek a neumožňuje takové přizpůsobení, jako princip dynamické rekonfigurace. Náročností je myšlená náročnost na výpočetní techniku a infrastrukturu, protože statická rekonfigurace vyžaduje množství propojených a spolupracujících uzlů, které jsou schopny předávat si informace při off-line režimu. Větší množství uzlů je nutné i z důvodu přenášení výkonu a využívání výhod rozděleného výpočetního

systému. Při větším množství interně spolupracujících uzlů je nutné implementovat pravidla, kterými se uzly řídí při předávání, zpracování a zobrazování získaných dat, což zvyšuje složitost celé aplikace. Vzhledem k tomu, že při statické rekonfiguraci se předpokládá uchovávání dat lokálně na daných uzlech, je nutné realizovat rozsáhlou strukturu pro persistenci dat, která může být potenciálně více zranitelná než u dynamické rekonfigurace, kde se data uchovávají v zabezpečené databázi.

Hlavní příležitostí statické rekonfigurace je jedinečná možnost spolupráce napříč propojenými uzly, a to i v případě off-line režimu. To zajišťuje vhodnou škálovatelnost v případě velkých projektů. Efektivnost aplikace je zajištěna vysokým potenciálem na provedení náročných výpočetních úkonů, prostřednictvím přenesení výpočtu na více uzlů. Statická rekonfigurace musí nutně zajišťovat efektivní mechanismus pro zabezpečení uchovaných dat, proto je vhodnou příležitostí vyvinout znovupoužitelné komponenty, které zabezpečení aplikace zastřeší.

Hlavní hrozbou statické rekonfigurace je nekonzistence, která může být způsobena nesouladem dat při off-line a on-line režimu, kdy aplikace bude zpracovávat data dle zadaných kritérií, které již nejsou aktuální, protože došlo ke změnám, které ještě nebyly promítnuty do mechanismu daného uzlu z důvodu jeho dočasné nedostupnosti v síti. Další hrozbou je neúměrné vynakládání výpočetních prostředků na rozšiřování sítě uzlů. Je nutné realizovat pouze takovou síť, která je nezbytně nutná pro funkčnost vyvíjené aplikace, aby nedocházelo k vytváření nadbytečných spojení mezi uzly, které nejsou žádným způsobem reflektovány ve výsledné aplikaci, což souvisí i s komplikovaností aplikace. Při nedůsledném mechanismu persistence může dojít ke ztrátě uchovaných dat.

### 2.6.3 Vyhodnocení distribuované rekonfigurace

Pro vyhodnocení distribuované rekonfigurace byly zjištěny následující charakteristiky:

**Tabulka 3 Charakteristiky distribuované rekonfigurace**

| SILNÉ STRÁNKY | SLABÉ STRÁNKY | PŘÍLEŽITOSTI | HROZBY |
|---|---|---|---|
| Kontinuálnost | Zdroje | Automatizace | Neefektivita |
| Paralelní vlákna | Výkon | Zefektivnění | Nízký přínos |
| Úspora | Složitost | Zjednodušení | Nadbytečnost |
| Okamžité změny | Koordinace | Odstranění | Možnost použití |

**Zdroj**: Vlastní zpracování

Zásadní výhodou distribuované rekonfigurace je možnost kontinuálního chodu aktuálně nasazené aplikace a upravované aplikace, tudíž není nutné původně nasazenou aplikaci omezovat na chodu. S tím se pojí využití paralelních vláken, které mají výhodu i pro samotný proces vývoje, kdy mohou být různé části aplikace vyvíjené současně, čímž lze realizovat přírůstkový způsob vyvíjení projektu a není nutné vodopádově dokončit celou aplikaci a až následně provádět testování. S tím souvisí úspora času a zdrojů, které je nutné na aplikaci vynaložit, protože více komponent je vyvíjeno současně a následně dojde k jejich optimalizaci a propojení. Změny v aplikaci se projeví okamžitě, protože paralelně upravovaná aplikace je okamžitě po dokončení nasazena, tudíž nenásleduje žádná prodleva v průběhu vývoje a nasazení.

Distribuovaná rekonfigurace je náročná na výpočetní kapacity a zdroje, protože obvykle se pomocí ní řeší již hotová aplikace, která realizuje dynamickou rekonfiguraci, do které je distribuovaná rekonfigurace následně zakomponována. S tím souvisí slabší výkon, který může být způsoben buď neefektivním propojením dynamické a distribuované rekonfigurace nebo přílišným využitím paralelních vláken, které celou aplikaci zpomalují. Propojování dvou a více metod rekonfigurace je potenciálně značně složité a může způsobit nepřehledný kód v případě větších projektů, stejně tak jako koordinace vývoje dílčích komponent v rámci jednotlivých metod rekonfigurace.

Jednou ze značných příležitostí, která může vývoj pomocí distribuované rekonfigurace významně zjednodušit, je automatizace, která ulehčí implementaci pomocí automatických mechanismů. Při použití distribuované rekonfigurace dochází ke zefektivňování chodu celé aplikace, protože práce na aplikaci neustále pokračuje a její běh se nikdy nezastavuje z důvodu neustálého nasazení v paralelním

vlákně. Potenciálem při distribuované rekonfiguraci je možnost jejího úplného odstranění, pokud budou postupy, realizované v původní metodě rekonfigurace korespondovat s postupy a principy distribuované rekonfigurace.

Hrozbou při distribuované rekonfiguraci je neefektivnost jejího použití, kdy optimalizace dané aplikace nebude dostatečná, tudíž přínos pro projekt bude minimální a dojde ke zbytečné ztrátě zdrojů. Distribuovaná rekonfigurace se tím stane nadbytečnou a nebude možné její efektivní využití. Je nutné si uvědomit, že distribuovaná rekonfigurace se realizuje zpravidla v projektech, které jsou již dokončeny a je nutná jejich úprava nebo optimalizace. Tyto projekty mohou být postaveny na zastaralých technologiích, které nebudou možnost distribuované rekonfigurace podporovat.

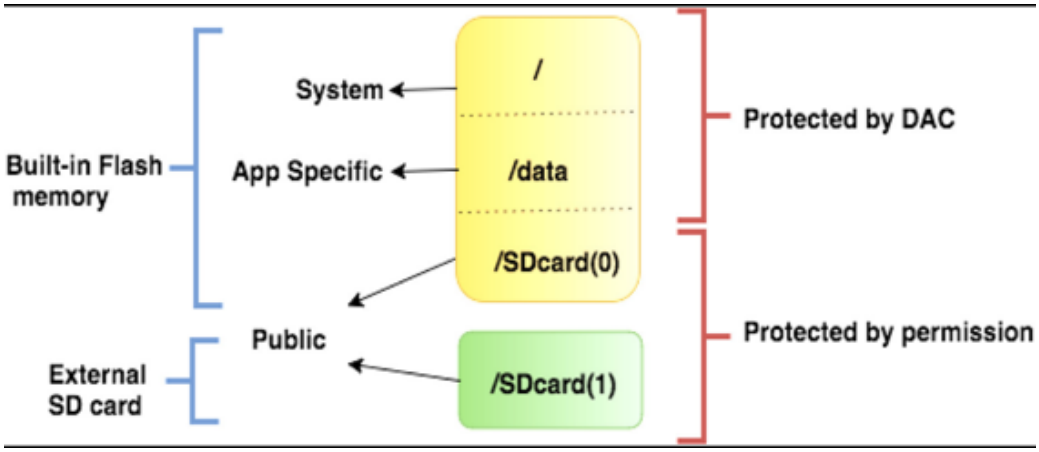
# 3 Management rizik na platformě Android

Z analýzy metod rekonfigurací na platformě Android vyplynulo, že nutným principem rekonfigurace je realizace zabezpečení uchovávaných dat. Aby bylo možné navrhovanou aplikaci v praktické částí této práce řádně zabezpečit, je nutné prozkoumat možnosti zabezpečení a jejich řízení (management) na platformě Android. Android zařízení obvykle obsahuje celou řadu potenciálně citlivých dat, jako jsou obrázky, videa, hesla, pracovní údaje a další, jak uvádí (Altuwaijri, Ghouzali, 2018). Tato data mohou být potenciálně zneužita při běhu aplikací, proto je nutné jejich zabezpečení.

## 3.1 Struktura platformy Android

Pro lepší orientaci v bezpečnosti platformy Android je vhodné zmínit její strukturu. Jak již bylo v této práci zmíněno, aplikace, spuštěné na platformě Android, jsou napsané v programovacím jazyce Java, případně Kotlin, nicméně samotná platforma je postavená na Linux Kernel, což je jádro, které používá operační systém Linux. I když aplikace na platformě Android jsou psané v jazyce Java, samotné jádro platformy je naprogramováno v jazyce C a C++, stejně tak jako potřebné sdílené knihovny, jak uvádí (Altuwaijri, Ghouzali, 2018).

Podle autorů (Altuwaijri, Ghouzali, 2018), je na platformě Android možné najít celkem tři třídy úrovní uchovávání dat, které se liší s ohledem na oprávnění přístupu k datům. Mezi tyto úrovně se řadí systémová třída, aplikačně specifická třída a veřejná třída. Systémovou třídu lze vnímat jako úložiště pro celý operační systém Android. Aplikačně specifická třída se stará o realizaci čtení a zápisu dat, které iniciují aplikace na daném mobilním zařízení. Veřejná třída je reprezentována ve formě SD karty mobilního zařízení, která se stará o sdílení dat mezi jednotlivými aplikacemi. Výše popsaná struktura je přehledně zobrazena na obrázku níže. Souhrnně lze popsanou strukturu označit za úložiště dat systému Android (Android data storage).




**Obr. 3 Struktura OS Android**
Zdroj: Altuwaijri, Ghouzali, 2018

## 3.2 Konkrétní bezpečnostní rizika

V předchozí kapitole byla popsána hierarchická struktura operačního systému Android. Data jsou ohrožena řadou faktorů, kterým se bude věnovat tato kapitola.

### 3.2.1 Hrozby fyzického charakteru

Hrozby fyzického charakteru vznikají z důvodu možné ztráty mobilního zařízení, případně z jeho krádeže. Data, která jsou uložená na mobilním zařízení a na SD kartách, se tímto stávají velmi zranitelná a zneužitelná. Do této kategorie se dle (Altuwaijri, Ghouzali, 2018) řadí i nebezpečí z odcizení dat aplikacemi, které svým počínáním mohou v krátké době získat data z úložiště mobilního zařízení. Tyto aplikace jsou obvykle do zařízení nahrány po jeho odcizení, popř. ztrátě cizí osobou, která má tak k zařízení fyzický přístup.

#### Útok při načtení zařízení

Jedním ze způsobů, kterým mohou být získány data z úložiště mobilního zařízení, je útok při načtení zařízení (cold boot attack). Tento útok je možné provést za pomoci RAM (random access memory), protože tato paměť v sobě uchovává data po určitou dobu poté, co je zařízení vypnuto. Po dobu, po kterou paměť data uchovává, může být ze zařízení vyjmuta a připojena k zařízení jinému, které umožní data z paměti RAM přečíst, a to včetně šifrovacích klíčů. Pomocí těchto šifrovacích

klíčů je následně možné prolomit ochranu celého mobilního zařízení a dostat se ke všem citlivým datům uživatele.

#### Útok na nechráněné oblasti

Útok na nechráněné oblasti vyplývá ze samotné funkčnosti platformy Android, protože každé softwarově orientované šifrování potřebuje určité části systému nezašifrované, jak uvádí (Altuwaijri, Ghouzali, 2018). To je problém zejména u platformy Android, protože ta ponechává jako jednu z nezašifrovaných částí samotné jádro operačního systému. Útočník, který má fyzický přístup k mobilnímu zařízení, do něho tak může nahrát vlastní, modifikované jádro systému, které mu bude umožňovat veškerá oprávnění, a tím pádem i přístup k citlivým datům uživatele.

#### Útok na slabiny paměti

Toto bezpečnostní riziko spočívá v možnosti opakovaného přistupování k částem paměti, což po určité době způsobí narušení ochrany částí paměti, které mohou obsahovat citlivá data uživatele.

### 3.2.2 Hrozby softwarového charakteru

Stejně jako hrozby fyzického charakteru, existují i hrozby softwarového charakteru, které mohou ohrozit bezpečnost mobilního zařízení a platformy Android. Na rozdíl od fyzických hrozeb nemusí v tomto případě dojít ke ztrátě nebo odcizení mobilního zařízení útočníkem, ale do zařízení mohou být propašované potenciálně nechtěné aplikace, které způsobí ztrátu citlivých dat uživatele. Dle (Altuwaijri, Ghouzali, 2018) se do hrozeb softwarového charakteru řadí následující rizika:

#### Malware útok

Malware je specifický druh software, který se dostane do operačního systému Android a jeho cílem je poškození mobilního zařízení, případně odcizení citlivých informací uživatele. Toto je problém zejména na platformě Android, protože dle (Altuwaijri, Ghouzali, 2018) většina aplikací na platformě Android uchovává citlivá data pouze v čistém textu. To znamená, že tyto informaci jsou

snadno zneužitelné, jakmile se potenciálně nechtěná aplikace dostane určitým způsobem do mobilního zařízení uživatele.

### Zneužití špatného návrhu aplikace

Za řadu potenciálních rizik, které vznikají při používání platformy Android, mohou vývojáři software, kteří nedůsledně realizují bezpečnostní politiku samotné platformy. Konkrétně vznikají v aplikaci bezpečnostní rizika z důvodu špatného návrhu aplikace, opomenutí realizace přístupu a oprávnění určitých částí systému nebo ukládání citlivých dat uživatele do veřejných, lehce napadnutelných úložišť systému nebo externího zdroje.

### Přílišná oprávnění aplikací

Aplikace na platformě Android potřebují ke svému správnému chodu určitá oprávnění v systému, aby mohly manipulovat s daty. Jednoduchým příkladem je aplikace fotoaparát, která potřebuje oprávnění ke galerii a obrázkům v ní, aby do ní mohla přidávat obrázky nové, případně aplikovat nějaké filtry, použité při focení. Problémem je, že vývojář dané aplikace počítá vždy s povoleným přístupem, který bude realizovat pouze daná aplikace. Po nainstalování na operační systém ale neexistuje žádná záruka ani kontrola toho, že daný přístup skutečně využívá pouze k tomu určená aplikace.

Tímto způsobem tak může dojít k propašování malware do mobilního zařízení, který následně naboural jinou aplikaci s vysokými oprávněními a zneužije tato oprávnění k tomu, aby získal citlivá data uživatele. Problém vysokých oprávnění aplikací by měl být řešen již při jejich vývoji, aplikace by neměla ke svému chodu potřebovat vyšší oprávnění, než které naplňují smysl její existence.

### Špatné použití šifrování

V dnešní době již platforma Android disponuje šifrovacím API, které má za cíl pomoci vývojářům při zabezpečení Android platformy a zejména aplikací, která na této platformě fungují. Nicméně ve studii, zmíněné v (Altuwaijri, Ghouzali, 2018), více než 88 % testovaných aplikací šifrovací API využívá špatně, což je alarmující číslo, protože tyto aplikace mohou být potenciální hrozbou pro mobilní zařízení a jejich uživatele, kteří mají tyto aplikace nainstalované a aktivně je používají.

### Využívání veřejných úložišť

V případě aplikací na platformě Android je dle (Altuwaijri, Ghouzali, 2018) možné klasifikovat uživatelská data do tří skupin, kterými jsou citlivá data, veřejná data a data, která nepodléhají zvláštní nutnosti zabezpečení. Pro první dvě skupiny dat poskytuje platforma Android dostačující zabezpečení, což může být považováno za problém, pokud vývojář nebude schopen identifikovat, která data je nutné zabezpečit, a která nikoliv. Pokud data, která jsou pro uživatele citlivá, zůstanou uložená ve veřejných úložištích, jsou vystavena enormnímu riziku.

### Root mobilního zařízení

Root mobilního zařízení znamená nahrazení továrního nastavení výrobce svým nastavením, které odstraňuje veškeré předepsané limity a oprávnění aplikace a poskytuje vyšší možnosti jeho nastavení. Sám o sobě je root zařízení neškodný a může být užitečný, pokud je potřeba odstranit aplikace, které ve výchozím nastavení odstranit nelze a nejsou uživateli žádným způsobem užitečné, případně zpomalují chod mobilního zařízení. Riziko spočívá ve zcela nechráněném přístupu do zařízení, protože root obvykle vyžaduje přepnutí ochrany zařízení na neaktivní, čímž se stává zranitelným vůči veškerým externím útokům.

## 3.3 Konkrétní řešení bezpečnostních rizik

V předchozí části této práce byly popsány obvyklé hrozby a bezpečnostní rizika, která se nejčastěji vyskytují napříč aplikacemi, které jsou spuštěny na platformě Android. Pro úplnost budou zahrnuty i běžná řešení, která pomáhají předcházet únikům potenciálně citlivých informací a ztrátou osobních údajů, popř. jejich zneužití třetí osobou. Autoři (Altuwaijri, Ghouzali, 2018) prezentují ve své studii celkem devět řešení pro hrozby fyzického charakteru a celkem dvě řešení pro hrozby softwarového charakteru.

Pro účely této práce jsou relevantní pouze řešení softwarového charakteru, protože ta budou muset být nutně promítnuta do výsledné navrhované aplikace, aby byla zajištěna maximální bezpečnost jejích dat. Hrozby fyzického charakteru není možné v navrhované aplikaci ovlivnit, protože ty jsou součástí návrhu samotné platformy Android.

### 3.3.1 Obecné principy šifrování

Šifrováním se rozumí zakódování potenciálně citlivých dat, která by mohla být zneužita v případě, že dojde k jejich úniku. Pro přečtení zašifrovaných dat je zpravidla nutný klíč, který umožní jejich opětovné přečtení. K tomu, aby bylo možné data převést z běžného textu do zašifrované podoby, je třeba šifrovací algoritmus, který tento převod uskuteční. Když jsou data zašifrovaná, je možné je zasílat koncovým adresátům skrze síť s vyšší ochranou. Když data dorazí k adresátovi, je nutné je rozkódovat ze zašifrované podoby, aby bylo možné jejich čtení nebo úprava. K tomu je opět nutný klíč. V praxi existují dva hlavní typy klíčů, kterými je možné šifrovanou komunikaci realizovat.

Prvním je symetrický klíč, který je zcela unikátní a privátní jak pro odesílatele, tak pro adresáta, přičemž pouze tito dva účastníci komunikace klíčem disponují a mohou si navzájem zasílat a překládat zašifrovaná data. Druhým typem je veřejný klíč, který je navenek k dispozici všem uživatelům v síti, ale zásadní rozdíl oproti symetrickému klíči je v tom, že pouze adresát dat disponuje heslem, které je schopno zasílaná data dešifrovat.

#### Doporučení pro implementaci šifrování

Existuje celá řada kryptografických (šifrovacích) mechanismů a principů, které jsou v různé míře využívány všemi aplikacemi, které pracují s citlivými daty. Pro Českou republiku se touto problematikou zabývá primárně Národní úřad pro kybernetickou a informační bezpečnost (dále jen NÚKIB).

Kryptografické algoritmy lze rozdělit na dvě skupiny, a to na dlouhodobě bezpečné a krátkodobě bezpečné. Podobné označení je použito i v (NÚKIB, 2018), kde jsou tyto dvě skupiny označeny jako schválené a dosluhující. U dlouhodobě bezpečných šifrovacích algoritmů neexistuje v současné době žádná výpočetní technologie, která by je dokázala prolomit, a předpokládá se, že tato technologie nebude dostupná v řádech budoucích let, proto se nazývají dlouhodobě bezpečné. Krátkodobě bezpečné algoritmy jsou potenciálním rizikem, protože v současné době neexistují technologie, které by dokázaly těmito algoritmy vytvořené šifrování prolomit, ale v budoucnu tyto technologie dostupné být mohou (případně již jsou, jen jsou ve fázi testování), například dojde k vývoji v oblasti kvantových počítačů.

Kvůli tomuto riziku se doporučuje odstranit krátkodobě bezpečné algoritmy nejpozději v roce 2023 (NÚKIB, 2018).

### 3.3.2 Šifrovací mechanismy platformy Android

Z hlediska šifrování na platformě Android můžeme rozlišovat celkem 2 hlavní oblasti, kterými je plné šifrování disku (Full disk encryption) a tzv. keyChain.

#### Plné šifrování disku

Plné šifrování disku má za primární cíl ochránit uživatele mobilního zařízení před vystavením citlivých dat nežádoucím externím entitám. Tento mechanismus využívá buď pinu uživatele nebo jeho hesla k zamykací obrazovce pro zašifrování daného zařízení pomocí derivační funkce. Problém tohoto mechanismu spočívá v tom, že je jen tolik silný, jak silné heslo zvolí uživatel mobilního zařízení. Tento mechanismus zároveň snižuje výkon daného zařízení a pokud uživatel zapomene heslo, není možné se k zašifrovaným datům žádným způsobem dostat. Mechanismus nemůže být navíc použit pro každé mobilní zařízení, protože dle (Altuwaijri, Ghouzali, 2018) je mechanismus použitelný až pro zařízení s platformou Android 4 a vyšší, přičemž ne na všech zařízeních s verzí 4 je mechanismus dostupný.

#### KeyChain

KeyChain je API, které bylo navrženo, aby umožnilo vývojářům ukládat uživatelské údaje. KeyChain je aplikovatelný v případě, že není aktivované plné šifrování disku, protože keyChain neposkytuje vyšší ochranu, než je plné zašifrování. Hlavní rozdíl plného šifrování a keyChain spočívá v tom, že keyChain ukládá data mimo samotný zašifrovaný prostor, tudíž může být a je vhodné jej využít ve chvíli, kdy plné zašifrování není aktivní, protože méně zatěžuje dané mobilní zařízení na jeho výkonu.

#### Šifrování hesel napříč mobilní aplikací

Pro účely této práce bude zapotřebí podrobněji přiblížit jednu z metod šifrování, která se používá pro ukládání hesel do databází. Jedná se o hašovací funkci, která bude zaručovat integritu hesla. Ve vyvíjené aplikaci v rámci této práce

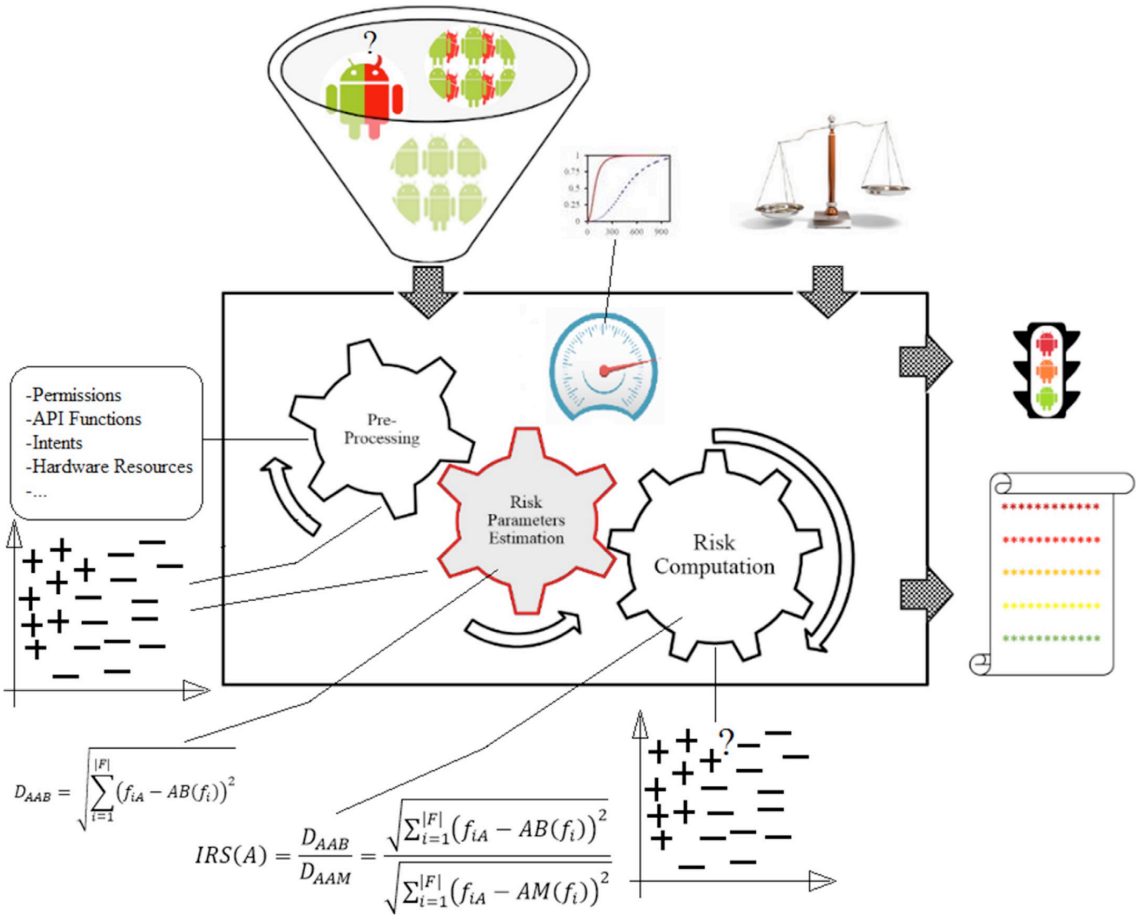
bude implementován přihlašovací a registrační mechanismus, pomocí něhož bude možné, aby si uživatel uložil data, která bude potřebovat pro svou práci a později se k nim bez ztráty mohl vrátit. Tento mechanismus s sebou nese riziko zneužití hesla uživatele v případě, že by bylo uloženo do databáze jako prostý text, protože kdokoliv z uživatelů, kdo by měl přístup k jádru databáze, by si mohl přečíst hesla jakéhokoliv uživatele.

Aby bylo tomuto riziku možné předejít, je nutné, aby heslo dorazilo do databáze již zašifrované, což bude zajištěno spuštěním hašovací funkce v programovém kódu mobilní aplikace ihned poté, co se uživatel zaregistruje, případně bude databáze implementovat vlastní princip šifrování hesel. Pro pozdější ověření přihlášení uživatele bude kódem z databáze získáno heslo příslušného uživatele, které bude teprve v kódu zpět dekódováno a porovnáno s heslem, které zadává uživatel přes přihlašovací formulář. Tímto způsobem je zajištěno, že administrátor, případně jiný uživatel databáze aplikace nikdy nebude mít neautorizovaný přístup k heslům registrovaných uživatelů.

### 3.3.3 Globální bezpečnostní mechanismus

Zmíněné potenciální hrozby a jejich řešení jsou vodítkem k tomu, aby bylo možné vytvořit globální metodiku k rozpoznávání nebezpečných aplikací, které mohou poškodit bezpečí koncových uživatelů, protože právě aplikace jsou zásadním bezpečnostním problémem a nejčastějším zdrojem úniku citlivých informací. Touto myšlenkou se inspirovali autoři (Deypir, Horri, 2018), kteří vytvořili návrh metriky, která bude tyto potenciálně nechtěné aplikace vyhodnocovat. Schéma funkčnosti metriky je znázorněno na obrázku č. 4.

Jedná se o obecný princip, který bude pomocí algoritmu již při nahrávání aplikace do online obchodu vyhodnocovat její rizikovost. Výpočet rizikovosti zvolili autoři (Deypir, Horri, 2018) na základě porovnávání dosud známých hrozeb v rámci aplikací a vlastností testované aplikace. Jakmile dojde k vyhodnocení, bude aplikace označena mírou hrozby, kterou představuje. Schéma na obrázku č. 4 reprezentuje grafické zobrazení celého procesu zjištění potenciální hrozby aplikace.




**Obr. 4 Metrika rozpoznávání hrozeb**
Zdroj: Deypir, Horri, 2018

Aby bylo možné udržet koncept výše zmíněného mechanismu, je vhodné na platformě Android vyvíjet takové aplikace, které budou maximálně zabezpečené. Takové aplikace budou chráněné proti vnějším rizikům, ale budou zároveň získávat lepší hodnocení v případě analýzy metrikou, vyvinutou autory (Deypir, Horri, 2018), viz obrázek č. 4.

### Zabezpečení aplikace v programovém kódu

Platforma Android zahrnuje množství komponent, které implementují různé druhy zabezpečení. Zde budou uvedeny pouze ty komponenty, které jsou významné s ohledem na vyvíjenou aplikaci. Zásadní komponentou pro implementaci bezpečnosti na platformě Android je balíček s názvem „java.security“. Tento balíček v sobě zahrnuje třídy a rozhraní, které jsou snadno přizpůsobitelné aktuálně vyvíjené aplikaci. Balíček „java.security“ rovněž zahrnuje šifrovací mechanismy, což je také důvod, proč je více než vhodný pro implementaci bezpečnosti aplikace na

platformě Android. V oblasti bezpečnosti existují tzv. „best practices“, tedy metody, které se doporučují pro implementaci bezpečnosti na platformě Android. Jak uvádí (Android developers, 2019), pro prvotní inicializaci aplikace je vhodné zajistit, aby uživatel mohl bezpečně zasílat data do aplikace, které důvěřuje. Tento princip je implementován pomocí níže uvedeného kódu v úpravách dle potřeb vyvíjené aplikace.

```
Intent intent = new Intent(Intent.ACTION_SEND);
List<ResolveInfo> activityList =
queryIntentActivities(intent, PackageManager.MATCH_ALL);

if (activityList.size() > 1) {

  String title = getResources().getString(R.string.chooser_title);
  Intent chooser = Intent.createChooser(intent, title);
  startActivity(chooser);

} else if (intent.resolveActivity(getPackageManager()) != null) {
  startActivity(intent);
}
```

**Kód 4 Příklad bezpečnosti - Intent**
Zdroj: Převzato z (Android developers, 2019)

Tento postup se na platformě Android realizuje přes třídu Intent. Kód zkontroluje, zda mohou být spuštěny alespoň dvě aplikace na mobilním zařízení uživatele. Pokud ano, ihned se spustí výběr aplikací. Pokud tyto dvě aplikace neexistují, dojde ke spuštění pouze tehdy, pokud je mobilní zařízení uživatele schopno operaci zpracovávat. Výše uvedeným způsobem je zajištěna zabezpečené zasílání dat mezi uživatelem a aplikací, které uživatel důvěřuje, a které chce zasílat určitá data.

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.myapp">
  <application ... >
    <provider
      android:name="android.support.v4.content.FileProvider"
      android:authorities="com.example.myapp.fileprovider"
      ...
      android:exported="false">
      <!-- Place child elements of <provider> here. -->
    </provider>
    ...
  </application>
</manifest>
```

**Kód 5 Příklad bezpečnosti – Zamítnutí přístupu**
Zdroj: Převzato z (Android developers, 2019)

Způsob uvedený v kódu 5 se používá v případě, že dochází ke sdílení dat mezi více aplikacemi, z nichž ne všechny aplikace jsou v režii stejného vývojáře. Tímto způsobem dojde k zamítnutí neautorizovaných přístupů k jádru vlastní aplikace.

```
URL url = new URL("https://www.google.com");
HttpsURLConnection urlConnection = (HttpsURLConnection)
url.openConnection();
urlConnection.connect();
InputStream in = urlConnection.getInputStream();
```

**Kód 6 Příklad bezpečnosti – SSL komunikace**
Zdroj: Převzato z (Android developers, 2019)

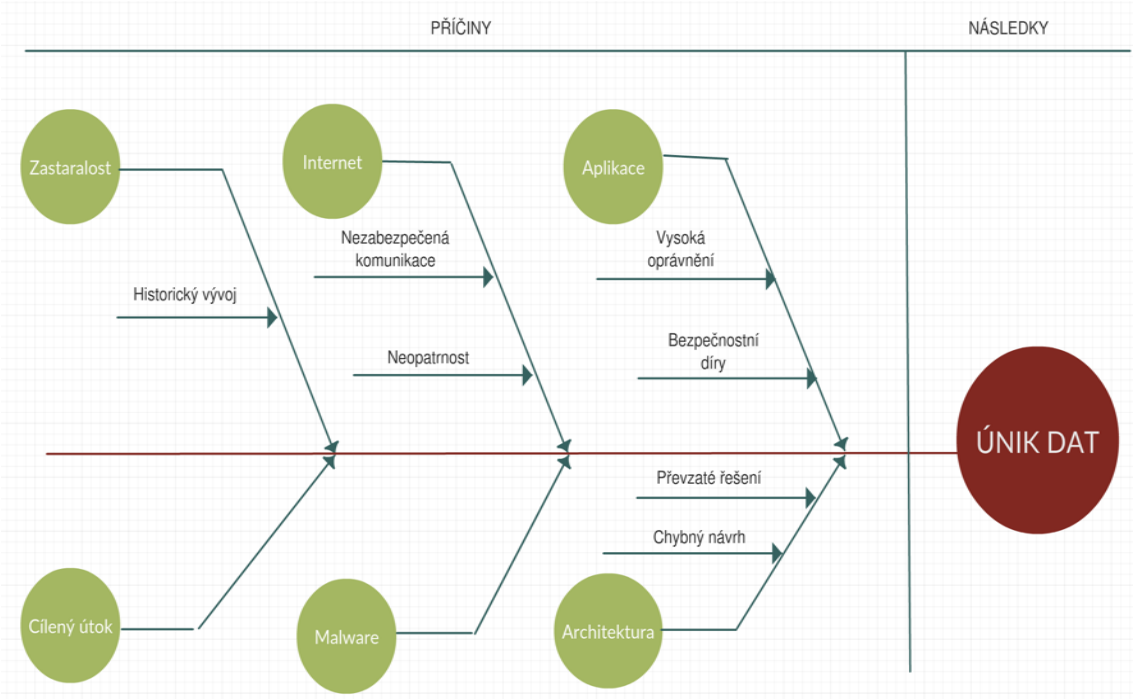
Výše uvedeným kódem dojde k inicializaci důvěrné komunikace, která je realizována pouze přes renomovaný certifikát, čímž se zvyšuje celkové zabezpečení zasílaných dat.

### 3.3.4 Zhodnocení bezpečnostních rizik

Vzhledem k rozsáhlosti a neuspořádanosti veškerých bezpečnostních rizik, která mohou nastat na platformě Android, je vhodné k jejich utřídění využít obdobné manažerské techniky, jako je SWOT analýza u metod rekonfigurace, aby bylo možné vytvořit si jasný a ucelený obrázek o rizicích na platformě Android.

### Ishikawův diagram

Ishikawův diagram nebo také diagram rybí kosti či diagram příčin a následků je vhodnou manažerskou technikou, která nachází uplatnění v širokém spektru oblastí, do kterých spadá i oblast bezpečnosti. (Management mania, 2015). V rámci této práce poslouží Ishikawův diagram jako metoda managementu rizik pro zachycení jednotlivých příčin vzniku zranitelností mobilních aplikací na platformě Android, kdy jako vstupní data budou sloužit zjištěné teoretické poznatky z předchozích kapitol ohledně bezpečnostních rizik platformy Android. Výsledek zkoumání je zachycen na obrázku 5 níže:




**Obr. 5 Ishikawův diagram příčin zranitelností platformy Android**
Zdroj: Vlastní zpracování

### Vyhodnocení bezpečnostních rizik

Jak je patrné z obrázku č. 5, nejčastějšími příčinami úniku dat z platformy Android jsou zastaralost kódu aplikace, což může být způsobeno historickým vývojem aplikace v rámci desítek let, kdy nejsou dostupné časové nebo finanční prostředky k modernizaci a aktualizaci kódu aplikace. Další příčinou úniku dat může být internet. Na internetu dochází k úniku nejčastěji z důvodu neopatrnosti uživatelů, kteří např. nevědomky stáhnou malware, který poškodí chod aplikace

nebo jinak zneužije její data. Dalším problémem je nezabezpečené komunikace po síti, kdy je vždy při provádění rizikových operací, jako je např. platba přes internet, vhodné komunikovat přes zabezpečený kanál, který realizuje vhodný protokol.

Největším problémem a příčinou vzniku bezpečnostních rizik jsou přílišná oprávnění aplikací, která následně v kombinaci s nevhodným chováním uživatele mohou způsobit, že skrze aplikaci je možné se dostat ke všem datům uživatele. Vývojáři aplikací často nevědomky zanechají v aplikacích bezpečnostní díry, kterými může být například používání zastaralých (deprecated) balíčků, které během krátké časové periody ztratí podporu nebo jsou prolomeny. Cílený útok na aplikaci je rovněž příčinou bezpečnostního rizika, nicméně není na vývojáři, aby tomuto zabránil, je pouze nutné držet se doporučených bezpečnostních principů a realizovat maximální prevenci před podobným útokem.

Jak již bylo zmíněno, malware se typicky dostane do mobilního zařízení kvůli neopatrnosti uživatele, což rovněž není bezpečnostní riziko, které lze zcela eliminovat při návrhu a vývoji aplikace. Posledním problémem aplikace je její architektura, kdy z důvodu chybného návrhu může dojít k prolomení bezpečnostních mechanismů. Převzaté řešení je rovněž značným problémem, protože ačkoliv je možné toto řešení částečně upravit a přizpůsobit, u větších projektů nebude nikdy možné naprosto přesně určit, zda se v aplikaci nenachází kód, který by ohrožoval její data.

# 4 Implementace vlastního řešení rekonfigurace

Po teoretickém zmapování metod rekonfigurací na platformě Android bude vhodné demonstrovat tyto metody na konkrétní aplikaci. Jak již bylo v práci zmíněno, bude se jednat o aplikaci sloužící především ekonomickým poradcům a analytikům pro usnadňování jejich každodenní práce. Aplikace je navrhována tak, aby byla maximálně znovupoužitelná pro jakoukoliv firmu. Není ani nutné, aby se firma zaměřovala výhradně na ekonomickou oblast, protože API je možné nakonfigurovat takovým způsobem, aby získávalo jakákoliv data, která budou pocházet od třetí strany nebo z vlastního serveru. Je podstatné zmínit, že aplikace nemá za cíl poskytovat enormní množství funkcionality. Výsledná aplikace bude pouze natolik komplexní, aby dokázala pokrýt potřeby demonstrace metod rekonfigurací.

## 4.1 Obecné požadavky na aplikaci

Je důležité předem vymezit, jaká data budou v aplikaci zapotřebí. Jelikož se aplikace zaměřuje na ekonomickou oblast, budou zvoleny základní ekonomické ukazatele České republiky. V České republice neexistuje bohužel mnoho veřejně dostupných API, které by tyto údaje poskytovaly. Jednou z mála stránek, která disponuje touto funkcí, je Český statistický úřad (dále jen ČSÚ). ČSÚ na svém webovém portálu nabízí následující ukazatele:

- Národní účty (HDP)
- Index průmyslové revoluce
- Zaměstnanost
- Nezaměstnanost
- Mzdy / platy
- Spotřebitelské ceny
- Ceny výrobců (Index cen průmyslových výrobců)
- Zahraniční obchod
- Obyvatelstvo

Všechna tato data jsou poskytována přes veřejně dostupné webové adresy (dále jen URL). Každá URL má svůj specifický identifikátor, který pokaždé zobrazuje jiný výsledek. Data jsou poskytována ve formátu SDMX, což je formát určený ke statistickému zpracování a výměně těchto dat. Formát SDMX je kompatibilní s formátem XML, tudíž v aplikaci bude nutné vytvořit XML Parser. Parser bude reprezentován třídou, která se bude starat o překlad formátu XML, aby z něho bylo možné získat konkrétní hodnoty.

### 4.1.1 Požadavky na funkcionalitu

Pokud má aplikace být užitečná, bude se muset lišit od API, které už samo o sobě poskytuje uživatelům ekonomická data. Kdokoliv může navštívit stránky ČSÚ a data si přečíst. Problém nastane ve chvíli, kdy je nutné data v krátkém čase zjistit, vyhodnotit a učinit na jejich základě závěry. Rovněž opakované „proklikávání se" k datům na webu ČSÚ může být značně zdlouhavé a frustrující. Tento problém bude řešen implementací mechanismu registrace a přihlašování. Tímto způsobem bude mít každý uživatel možnost zvolená data uložit a kdykoliv v krátkém čase se k nim ve svém mobilním zařízení vrátit.

Dalším požadavkem na aplikaci bude ukládání zvolených dat pro pozdější nahlédnutí. Aplikace totiž bude vždy po přihlášení získávat aktuální ekonomická data za poslední dostupný rok, nicméně pokud by uživatel chtěl zobrazovat data za minulý rok, nebylo by to možné. Proto bude možné zobrazené ekonomické ukazatele ukládat a k těmto uloženým hodnotám se po přihlášení kdykoliv vrátit. Každý uživatel tak díky mechanismu přihlašování uvidí pouze svá data, která si někdy v minulosti uložil.

### 4.1.2 Zvolená metoda rekonfigurace

V prvotní fázi vývoje je nutné rozhodnout o tom, jakou ze zmíněných metod rekonfigurací z předešlé části práce bude aplikace využívat. S ohledem na dosavadní znalost oblasti rekonfigurace, která vyplývá z teoretického prozkoumání odborné literatury, bude ve výsledné aplikaci využita dynamická a distribuovaná rekonfigurace, protože aplikace nebude enormně velká co do své velikosti, tudíž nebude nutné řešit problémy spojené s jejím výkonem. Problémem dynamické

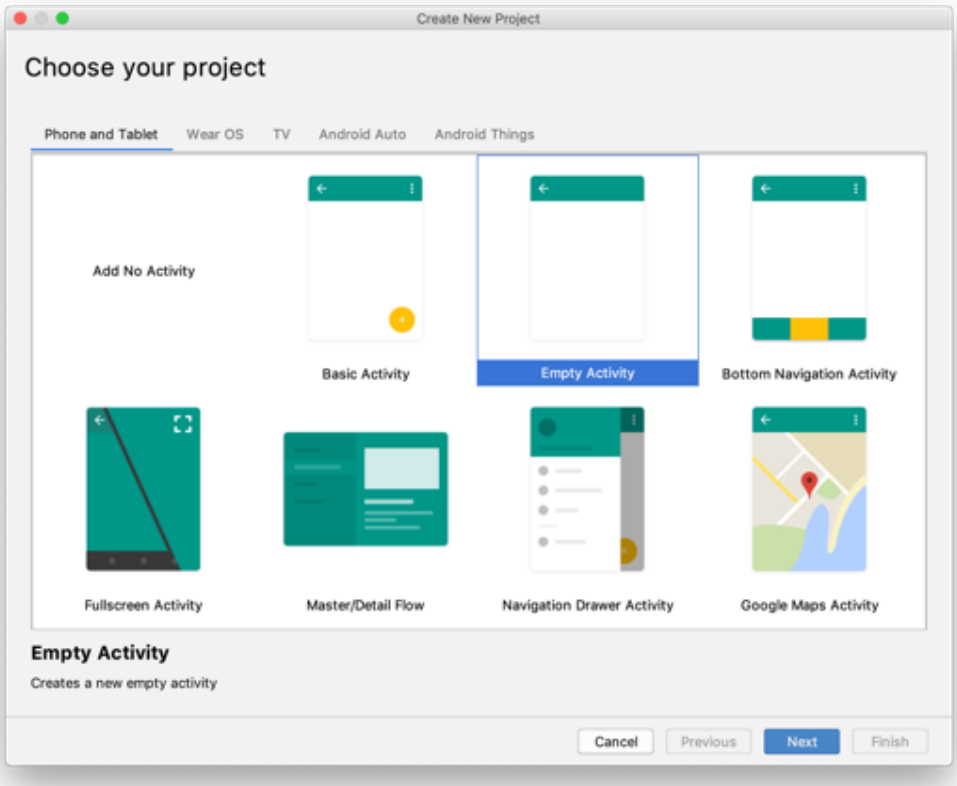
rekonfigurace bude off-line připojení, kdy nebude možné ekonomická data načítat, nicméně předpokládá se, že uživatel bude vždy připojen k síti, pokud bude chtít aplikaci využívat. Bude možné využití mobilních dat, protože aplikace bude mít kvůli své velikosti na tato data minimální požadavky. Dynamická rekonfigurace přinese řadu benefitů, jako je např. okamžitá úprava ekonomických ukazatelů podle hodnot API, tudíž bude zaručena maximální aktuálnost zobrazovaných dat. Distribuovaná rekonfigurace se bude projevovat ve chvíli, kdy bude spouštěno více paralelních aktivit uživatele najednou.

## 4.2 Představení Android studia

V této fázi je již jasně vymezena funckionalita i parametry, které musí daná aplikace splňovat, proto je možné začít s jejím vývojem. Je nutné zvolit si vývojové prostředí. Jak již bylo uvedeno a zdůvodněno v úvodu této práce, aplikace bude naprogramována v programovacím jazyce Java za pomoci vývojového prostředí Android studio. Pro zahájení vývoje dané aplikace je nutné si v Android studio vytvořit nový projekt s vhodným pojmenováním. V případě aplikace v této práci se jedná o ekonomickou aplikaci, proto jméno aplikace bude „EconomicApp“. Veškerý programový kód aplikace bude psán v angličtině z důvodu vhodnější konzistence s jazykem Java a jeho pojmenovávacími konvencemi, které jsou psané v anglickém jazyce. Android studio nabízí v základu dva vzhledy, první vzhled je klasický bílý a druhý vzhled se jmenuje darcula. Darcula je opakem klasického vzhledu, jedná se o černý vzhled. Pro potřeby této práce je zvolen klasický vzhled pro lepší čitelnost programového kódu ve vývojovém prostředí.

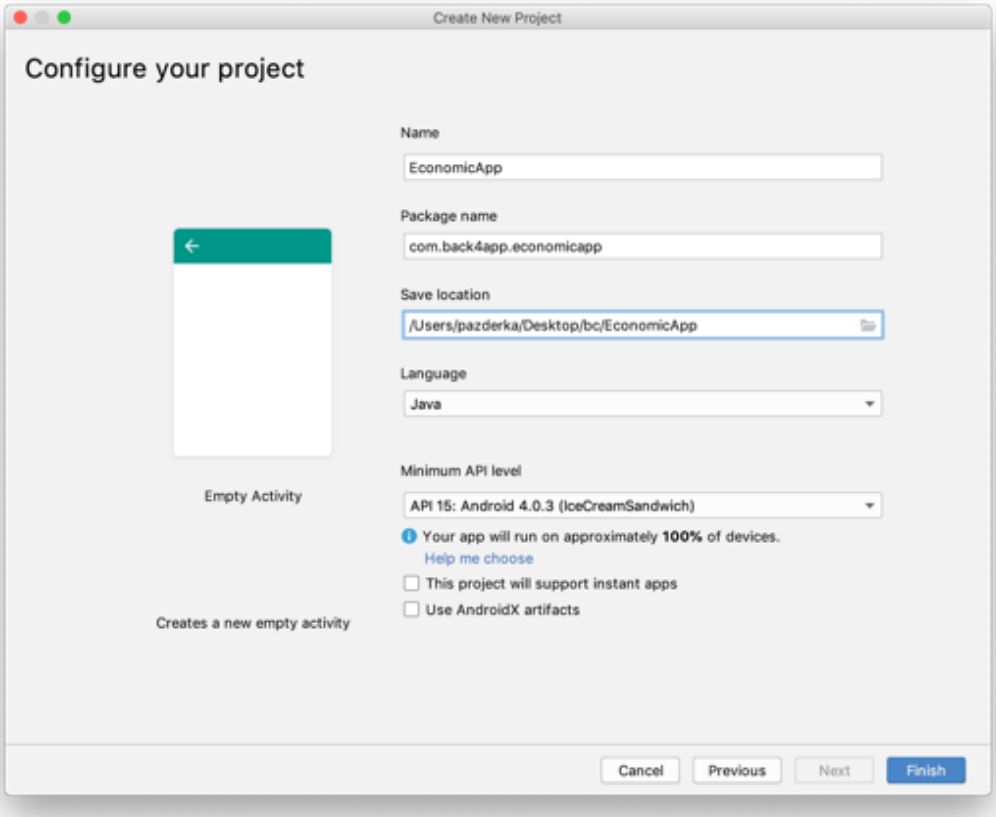

**Obr. 6 Vytvoření nového projektu**
Zdroj: Vlastní zpracování

Android studio umožňuje hned několik možností, viz obrázek 6 výše, například importovat existující projekt nebo část kódu. V případě této práce neexistuje žádná rozpracovaná varianta navrhované aplikace, proto je vytvořen zcela nový projekt. Zajímavou vlastností při vývoji na platformě Android jsou aktivity, viz obrázek 7 níže.




**Obr. 7 Aktivity platformy Android**
Zdroj: Vlastní zpracování

Aktivitou na platformě Android se rozumí okno, které právě uživatel vidí. Typicky disponuje daná aplikace hned několika aktivitami, které se přepínají v závislosti na akcích, které uživatel provede. Aktivitu lze připodobnit například k libovolnému oknu v operačním systému Windows. Pro vyvíjenou aplikaci je vhodné mít připravenou základní šablonu, která ale nebude příliš specifická, aby nebylo nutné odstraňovat přebytečné prvky. Proto je zvolena prázdná aktivita (Empty Activity). Poslední věcí, kterou potřebuje Android studio znát, je konkrétní konfigurace projektu.




**Obr. 8 Konfigurace projektu**
Zdroj: Vlastní zpracování

V konfiguraci projektu je možné zvolit jméno aplikace, jméno balíčku, ve kterém bude umístěna (obvykle se užívá notace teček) a umístění aplikace na disku počítače spolu s vývojovým jazykem. Zajímavou položkou je Minimum API level. V tomto případě API znamená verzi operačního systému Android. Je možné vybrat z celé řady starších verzí. Vždy je vhodné volit tu verzi, kterou bude podporovat nejvíce mobilních zařízení, protože pak bude aplikace maximálně využitelná.

V současné době budou téměř všechny mobilní zařízení aplikaci spustit, pokud bude vyvinuta pro verzi operačního systému Android 4.0.3 s názvem IceCreamSandwich, proto je i tato verze zvolená pro výslednou aplikaci této práce. Je možné, že v průběhu vývoje aplikace se minimální API ještě změní, protože například bude nutné implementovat funkcionalitu, které není možné dosáhnout metodami staršího API a je nutné jej aktualizovat na novější verzi.

Je nutné rozlišovat mezi ekonomickým API, které je zmiňované v rámci této práce a API Android studia, protože se jedná o dvě zcela odlišné věci. Ekonomické API je tzv. interface, který umožňuje získávat aktuální ekonomické ukazatele od třetí strany, v tomto případě ČSÚ. API Android studia pak představuje spíše ucelený soubor tříd, jejich metod a nástrojů, které jsou využívané při práci s Android studiem. Platí, že čím vyšší API je, tím více pokročilejších metod je možné používat, na druhou stranu se tímto rovněž zvyšují nároky na operační systém Android mobilních zařízení, které budou danou aplikaci spouštět. Na obrázku 8 výše je možné si všimnout jména balíčku, které obsahuje „back4app“. Obsah tohoto slova bude vysvětlen dále v této práci.

### 4.2.1 Struktura projektu

Android studio udělá po iniciování nového projektu řadu práce na pozadí, jako je založení základních a pomocných souborů, stažení aktualizací a pluginů.

Jak je vidět v příloze 2, Android studio vygenerovalo řadu složek a souborů. Pro začátek vývoje aplikace jsou nejdůležitější tři soubory, prvním souborem je MainActivity.java, což je soubor, který je zodpovědný za chování konkrétní aktivity.

Druhým důležitým souborem je soubor aktivity_main.xml, který je zodpovědný především za vzhled dané aktivity. Vzhled aktivity je možné měnit skrze okno, kdy pouze pomocí uživatelského rozhraní vývojář nastaví, které elementy chce v aktivitě mít a jaké mají pojmenování. Grafické uživatelské rozhraní pro vývojáře na pozadí rovněž edituje XML soubor, tudíž výsledek je stejný a je na vývojáři, aby rozhodl, která z metod je pro něho přijatelnější. Soubor AndroidManifest.xml obsahuje nutné konfigurační soubory, které Android studio potřebuje ke svému chodu. Na spodní liště Android studia je možné si všimnout oznámení „Gradle build finished in 1s 245ms“. Tímto Android studio říká, že kompilace aplikace byla úspěšně dokončena v čase jedné sekundy a 245 milisekund, přičemž o kompilaci se stará nástroj Gradle, který byl popsán v teoretické části této práce.

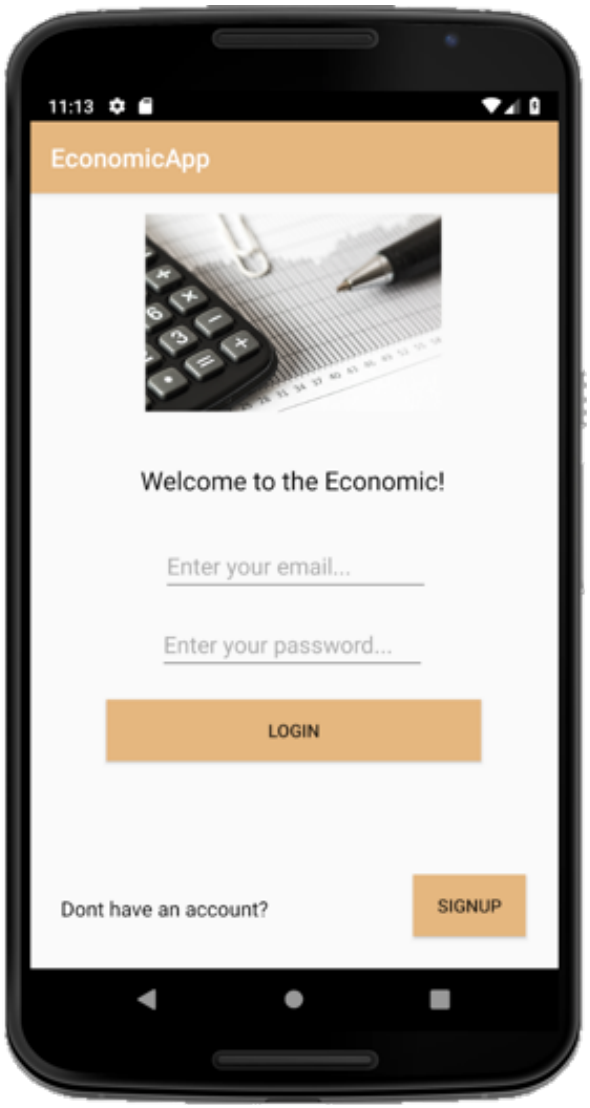
### 4.2.2 Spouštění aplikace

Nabízí se otázka, jakým způsobem je možné aplikace otestovat, lépe řečeno spustit na odlišných laptopech s odlišnými operačními systémy, když vyvíjená aplikace potřebuje ke svému chodu právě operační systém Android. Řešením tohoto problému je tzv. emulátor, který spustí nové okno, ve kterém nasimuluje podmínky operačního systému Android. Je rovněž možné zvolit typ chytrého zařízení, na kterém bude aplikace spuštěna. Na výběr jsou i tablety a televize. V případě této aplikace je vybráno mobilní zařízení Nexus 6.


**Obr. 9 Spouštění emulátoru**
Zdroj: Vlastní zpracování

Původní spuštění vygenerované aplikace vypadá tak, jak ukazuje obrázek č. 9. Pro potřeby aplikace vyvíjené v této práci je jako jedna z priorit vyvinout přihlašovací systém. Za tímto účelem je nejprve vhodné navrhnout uživatelské rozhraní, které se bude skládat ze dvou políček pro zadání emailu a hesla. Pod nimi

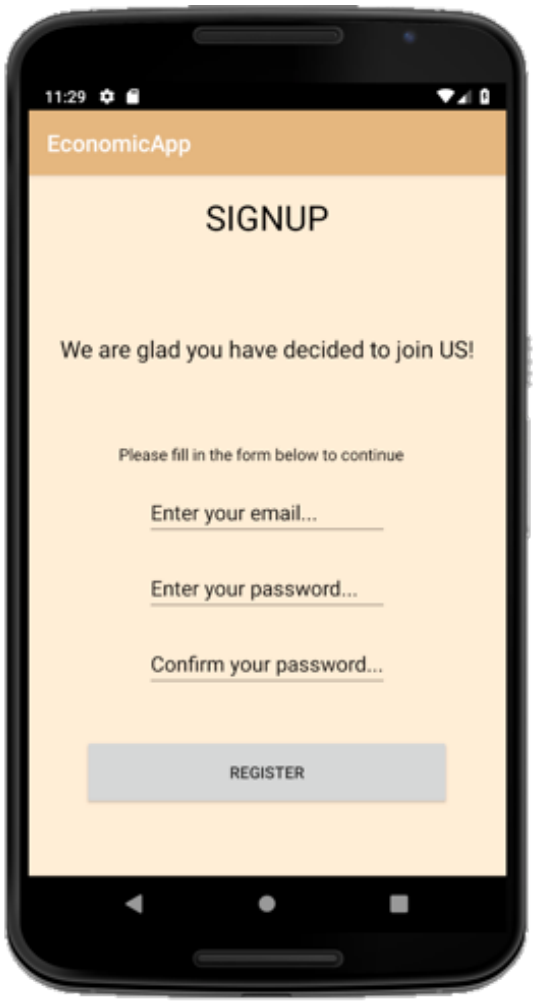
se bude nacházet tlačítko, které iniciuje přihlášení uživatele. Alternativně bude dostupné i další tlačítko pro registraci, ačkoliv tato funkcionalita nemusí být uživatelem aplikace vyžadována, protože firma může například interně vytvářet účty svým zaměstnancům, kteří se budou následně pouze přihlašovat, tudíž nikdo jiný nebude moci v aplikaci vytvořit účet. Tlačítko lze ale jednoduše odstranit, proto bude do výsledné aplikace zahrnuto s tím, že je kdykoliv jednoduše možné ho odstranit. Dále je vhodné umístit reprezentativní obrázek, který bude nejlépe vystihovat podstatu vyvíjené aplikace. Výsledná uvítací aktivita vypadá následovně:




**Obr. 10 Přihlašovací aktivita**
Zdroj: Vlastní zpracování

Je vhodné si zvolit primární barvu, aby aplikace byla kontrastní. V tomto případě byl zvolen odstín hnědé, kterému se v odvětví designu říká „burlywood“.

Barva má výhodu v tom, že je na ní poměrně dobře čitelné jak černý, tak bílý text, což je rovněž patrné z obrázku č. 10. Při vytváření aktivit je vhodné držet se stanovených pojmenovávacích konvencí. Aktivity jsou v této třídě pojmenovávány stejně jako třídy v odvětví objektového modelování, tudíž první písmeno velké a písmena dalšího začínajícího slova rovněž velké, vše bez mezer. Na konec je umístěno slovo „Activity“, aby nedošlo k záměně s jinými třídami. Přihlašovací aktivita tak nese jméno LoginActivity. V dolní části obrazovky se vyskytuje tlačítko pro registraci, které bude odkazovat na novou aktivitu, která bude uživatele registrovat. Podoba této aktivity je patrná z obrázku č. 11 níže. Tato aktivita se jmenuje SignUpActivity.




**Obr. 11 Registrační aktivita**
Zdroj: Vlastní zpracování

Jak přibývá aktivit a tříd ve vývojovém prostředí, začne za nějakou dobu projekt být nepřehledný a hůře udržovatelný. Za tímto účelem je vhodné rozčlenit

třídy a aktivity do jednotlivých balíčků, které budou reprezentovat a zastřešovat jejich funkcionalitu. Základními balíčky pro tuto práci budou:

- authActivities
- core
- dataActivities
- parser

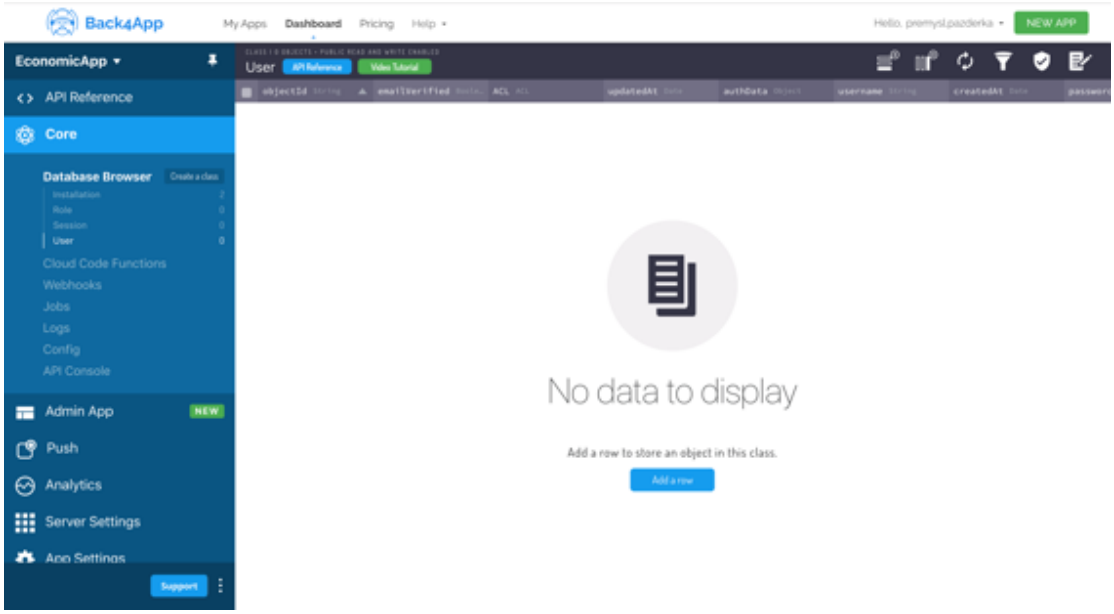
Nyní vytvořené aktivity jsou přesunuty do balíčku authActivities, který zahrnuje všechny akce, které souvisejí s ověřováním uživatelů při přihlašování a registraci. Ostatní balíčky zůstávají prozatím prázdné a budou postupně doplněny v průběhu vývoje aplikace.

### 4.2.3 Databáze

Nyní, když jsou hotové dvě základní aktivity, přihlašovací a registrační aktivita, je nutné implementovat jejich funkčnost. K ověřování uživatelů a jejich registraci bude zapotřebí databáze, protože je nutné ukládat uživatelská hesla a jména, která budou sloužit k přihlašování. Při vývoji Android je možné vybírat z celé řady alternativních databází, jako je např. SQLite, což je odlehčená varianta SQL databáze nebo Firebase, což je databáze umístěná v cloudu. Cloudem se rozumí online uložiště na internetu, které je většinou spravováno nějakou větší korporací a umožňuje uživatelům využití celé řady služeb, jako je přenášení výpočetního výkonu, management projektů nebo právě databázové úložiště. Výhodou cloudového řešení databáze je, že vývojář se zaregistruje na danou stránku, vytvoří si databázi a následně ji pouze přes privátní klíč integruje do své aplikace. Nemusí se starat o její funkčnost ani rozsáhlost, protože tyto aspekty zajišťuje společnost, která danou aplikaci zastřešuje. Příkladem hojně využívaných databázových řešení v cloudu může být společnost Amazon, která zastřešuje služby Amazon Web Services (dále jen AWS), které rovněž tuto funkcionalitu umožňují.

Pro tuto práci bude zvolen modernější trend ve vývoji databází, kterými jsou nerelační, tzv. NoSQL databáze. Rovněž je ale vhodné zachovat výhody cloudového řešení, proto bude využita databáze MongoDB v cloudu, což je NoSQL databáze. K tomuto účelu by bylo možné použít zmiňované AWS služby, nicméně pro potřeby

této aplikace by toto řešení bylo zbytečně komplexní a komplikovalo by celý vývoj svým nastavováním. Namísto toho bude využita služba stránky back4app, která extrahuje služby AWS do jediného komponentu, které je mnohem přehlednější a snadnější na použití. (Toto je důvod přítomnosti názvu back4app v programovém kódu). Tato služba se jmenuje Parse server, který zastřešuje právě funkcionalitu MongoDB databázového systému v cloudu. Pro využívání Parse serveru z webu back4app je nutné se na tomto webu registrovat a založit databázi pro aplikaci. Po vytvoření účtu a přihlášení do aplikace vypadá uživatelské rozhraní následovně.




**Obr. 12 Uživatelské rozhraní Parse serveru**
Zdroj: back4app.com

Pro potřeby této aplikace bude zapotřebí sloupec po levé straně na obrázku č. 12. Jak je vidět, aplikace již předem vygeneruje kolekce Instalation, Role, Session a User. Kolekcemi se rozumí ucelený souborové celky, které mají společné vlastnosti, v případě nerelačních databází jsou kolekce alternativou ke klasickým SQL tabulkám. Kolekce instalation není pro funkčnost aplikace příliš podstatná, jedná se pouze o lokální instalace dané aplikace. Kolekce Role rovněž nemá pro funkčnost aplikace žádný význam. Kolekce Session zachycuje přihlášené instance uživatelů a zaznamenává je pro potřeby pozdějšího použití.

Nejdůležitější kolekcí je kolekce User, která bude obsahovat registrované uživatele. U uživatelů se bude zaznamenávat email jako přihlašovací jméno do aplikace a heslo, kterým se uživatel ověří. Bude také nutné vytvořit sloupec, který

bude zaznamenávat uživatelem uložená data, aby bylo možné ke konkrétním uživatelům přiřadit jimi uložená data. Toto řešení bude implementováno pomocí vytvoření sloupce, který bude typu JSON a bude v sobě zahrnovat celou sadu uložených informací uživatelem.

### 4.2.4 Registrace uživatelů

Nyní je již možné naprogramovat samotnou funkcionalitu registračního mechanismu v Android studiu. Za registraci bude zodpovědný následující kód:

```
package com.back4app.EconomicApp.authActivities;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import com.back4app.EconomicApp.R;
import com.google.gson.Gson;
import com.google.gson.JsonObject;
import com.parse.ParseException;
import com.parse.ParseUser;
import com.parse.SignUpCallback;

public class SignUpActivity extends AppCompatActivity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_sign_up);
                Button btnSignUp = (Button)findViewById(R.id.btnRegister);

                btnSignUp.setOnClickListener(new View.OnClickListener() {

                        @Override
                        public void onClick(View view) {
                        registerClicked();
                        }
                });
}
```

**Kód 7 Implementace registrace**
Zdroj: Vlastní zpracování

Nejprve je nutné vytvořit třídu SignUpActivity, která rozšiřuje třídu AppCompatActivity, což je třída platformy Android, která umožňuje interakci mezi uživatelským rozhraním a třídou. Veškerý kód je vkládán do metody onCreate, což je metoda vygenerovaná Android studiem, která je spuštěna vždy poté, co je aktivita iniciována uživatelem. Nejprve je nutné získat instanci tlačítka registrace, aby bylo možné na jeho kliknutí navázat specifickou akci. To lze provést metodou setOnClickListener, která na tlačítko naváže specifickou akci. Do této akce je vložena vlastní funkce, aby byl kód více separovaný a přehlednější. Funkce se bude jmenovat

registerClicked, aby bylo zřejmé, co reprezentuje. Samotný kód této funkce je vidět v kódu níže.

```
public void registerClicked() {

EditText txtEmail = findViewById(R.id.txtSignUpEmail);
EditText txtPassword = findViewById(R.id.txtSignUpPassword);
EditText txtPasswordConfirm = findViewById(R.id.txtCheckSignUpPassword);

if (txtEmail.getText().toString().matches("") || txtPassword.getText().toString().matches("") ||
txtPasswordConfirm.getText().toString().matches("")) {

		Toast.makeText(SignUpActivity.this, "Sorry, all fields are required",
		Toast.LENGTH_SHORT).show();
		}
else if (!txtPassword.getText().toString().matches(txtPasswordConfirm.getText().toString())) {
		Toast.makeText(SignUpActivity.this, "Passwords must match",
		Toast.LENGTH_SHORT).show();
		}
else {
		ParseUser user = new ParseUser();
		user.setUsername(txtEmail.getText().toString());
		user.setPassword(txtPassword.getText().toString());
		JsonObject jobj = new JsonObject();
		Gson gson = new Gson();
		jobj.addProperty("Consumer", "0");
		jobj.addProperty("Producer", "0");
		jobj.addProperty("HDP", "0");
		jobj.addProperty("Industrial", "0");
		jobj.addProperty("Employment", "0");
		jobj.addProperty("Population", "0");
		jobj.addProperty("Wages", "0");
		jobj.addProperty("Unemployment", "0");
		jobj.addProperty("International", "0");
		user.put("savedData", gson.fromJson(jobj, Object.class));

		user.signUpInBackground(new SignUpCallback() {
		@Override
		public void done(ParseException e) {
			if(e == null) {
				Toast.makeText(SignUpActivity.this, "You have been
				signed up!", Toast.LENGTH_SHORT).show();
				finish();
				}
			else {
				Toast.makeText(SignUpActivity.this, e.getMessage(),
				Toast.LENGTH_SHORT).show();
				}
		}
});
}
}
}
```

**Kód 8 Registrační funkce**
Zdroj: Vlastní zpracování

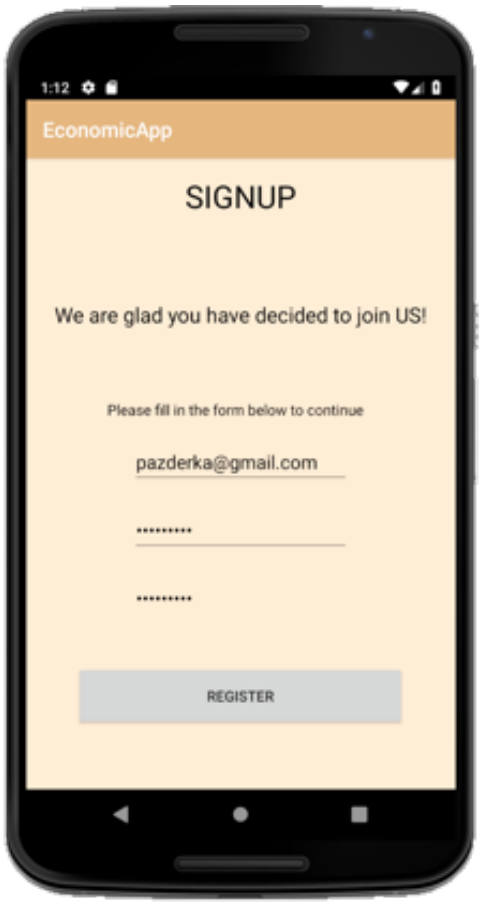
Nejprve je nutné extrahovat veškerý obsah, který uživatel do formulářů zadal. Jedná se o uživatelské jméno, heslo a opakované zadání hesla pro kontrolu. To lze provést metodami findViewById, do kterých se zadá název textového pole, který byl zvolen při návrhu, v těchto případech postupně txtSignUpEmail, txtSignUpPassword a txtCheckSignUpPassword. Poté, co jsou hodnoty extrahované v proměnných, je nutné provést drobnou validaci. Pomocí metody matches, volané na jednotlivých textových polích je zjištěno, zda jsou pole prázdná. Pokud ano, pak je uživateli zobrazena informace o tom, že je nutné vyplnit všechna zobrazená pole. Dále je validováno heslo, rovněž pomocí metody matches, ale tentokrát se neporovnává s prázdným textem, ale porovnávají se pole hesla a hesla pro kontrolu. Pokud se pole neshodují, uživateli je o tomto podána informace. Pokud celá validace proběhne v pořádku, je nutné uživatele uložit do databáze. Dojde k vytvoření nové instance třídy ParseUser, což je třída, která se stará o ukládání záznamů, v tomto případě uživatelů, do MongoDB databáze Parse serveru. Dále pomocí metod setUserName a setUserPassword, do kterých jsou dosazeny hodnoty z polí formuláře, dojde ke spárování vytvořeného uživatele se zadanými údaji.

Dále je nutné myslet na fakt, že uživatelé budou mít možnost ukládat si aktuální ekonomické ukazatele. Za tímto účelem je v databázi dostupný sloupec savedData, který je formátu JSON. Při registraci uživatelů je ale nutné tento sloupec naplnit výchozími hodnotami, aby bylo následně možné sloupec upravovat, jinak by server při pozdějším pokusu o uložení dat hlásil chybu. Všechny výchozí hodnoty budou nastaveny na nulu. Hodnoty jsou pojmenovány zkratkami adekvátně podle dat, která jsou získávána z ČSÚ, konkrétně:

- Consumer
- Producer
- HDP
- Industrial
- Employment
- Population
- Wages

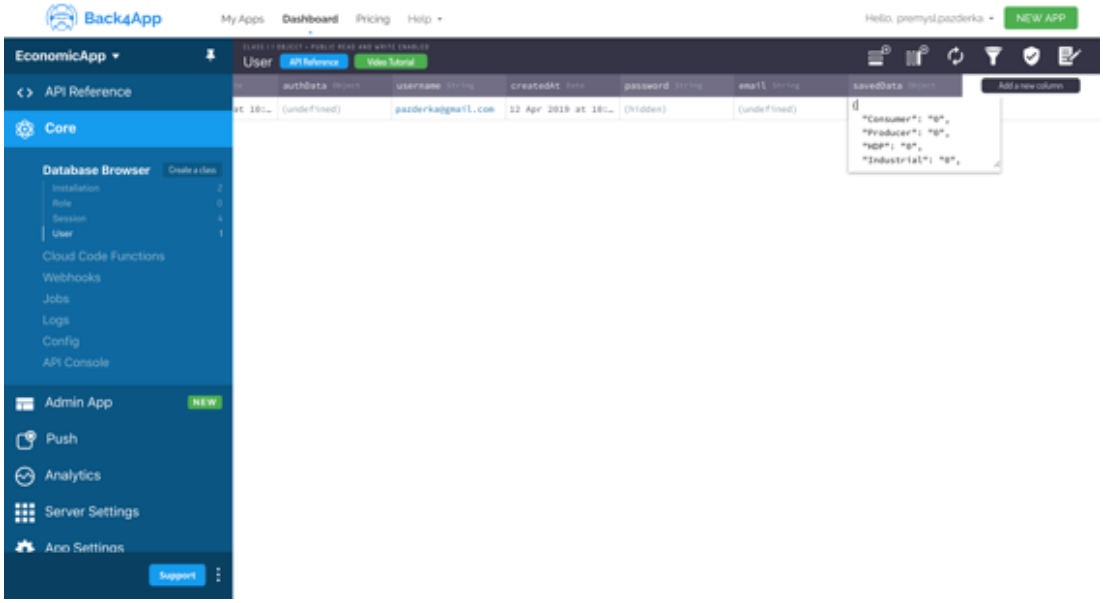
- Unemployment
- International

Všechny tyto hodnoty jsou pouze interní pro potřeby aplikace, uživatel uvidí vždy celé názvy proměnných a jejich hodnoty. Následně je pomocí metody put vložen tento vygenerovaný sloupec do instance konkrétního uživatele. Na závěr je nutné na konkrétní instanci uživatele zavolat metodu saveInBackground, která se postará o import nastavených dat do databáze. V případě chyby vrací metoda výjimky ParseException. Pokud je výjimka hodnoty null, znamená to, že uživatel byl úspěšně uložen, protože aplikace do této proměnné neuložila žádnou hodnotu. V takovém případě je uživateli zobrazena zpráva o úspěšné registraci a je přesměrován na přihlašovací aktivitu. Funkcionalita je zřejmá z obrázků 13 a 14 níže.




**Obr. 13 Vytvoření testovacích dat**
Zdroj: Vlastní zpracování

Je vytvořen email pazderka@gmail.com spolu s heslem. Poté, co je kliknuto na tlačítko register, je zobrazena zpráva o úspěšné registraci a zobrazí se přihlašovací aktivita. Rovněž je nyní nutné zkontrolovat, že záznam byl skutečně uložen do databáze. To lze zjistit pomocí akce refresh na webové stránce back4app.




**Obr. 14 Ověření uživatele v databázi**
Zdroj: back4app.com

Jak je vidět, mechanismus registrace je nyní funkční, protože v databázi se objevil jeden záznam, který odpovídá zadaným hodnotám. Je rovněž patrné, že je funkční mechanismus, který napojuje výchozí ekonomická data na registrovaného uživatele. Ve sloupci savedData jsou vidět všechny výchozí hodnoty nastavené na nulu, které jsou formátu JSON.

### 4.2.5 Získávání dat z ČSÚ

Nyní je hotový mechanismus registrace uživatelů. Ještě předtím, než bude implementován mechanismus přihlašování, je logické zajistit získávání dat z ČSÚ, protože po přihlášení budou tato data ihned zobrazena. Na tuto funkčnost je nutné vytvořit vlastní parser, který se o získávání dat bude starat. V ideálním případě by stačila jedna třída, která bude data parsovat, nicméně v případě ČSÚ je problémem rozdílná interpretace dat, některá data jsou zobrazena pouze za daný rok, jinde jsou zase data za několik let zpětně. Dalším problémem jsou jednotky. Každý údaj je v jiných jednotkách, některé údaje jsou dokonce spočteny jako průměry již předem.

Proto je nutné pro každý záznam vytvořit vlastní parser, který se bude starat o správnou interpretaci číselných údajů. Parser je tzv. analyzátor, který bude data z ČSÚ číst.

### Národní účty

Tento údaj reprezentuje Hrubý domácí produkt (dále jen HDP) státu. Běžně čitelný formát dat ČSÚ pro HDP vypadá následovně.

Zobrazeno 1-30 záznamů z celkového počtu 36 na 2 stránkách
Velikost stránky 30 < 1 2 >

| Období | Hrubý domácí produkt v kupních cenách - běžné ceny - sezónně očištěno NGDP_SA_XDC | Výdaje na konečnou spotřebu domácností a neziskových institucí- běžné ceny - sezónně očištěno NCP_SA_XDC | Výdaje na konečnou spotřebu vládních institucí - běžné ceny - sezónně očištěno NCG_SA_XDC | Tvorba hrubého fixního kapitálu - běžné ceny - sezónně očištěno NFI_SA_XDC | Změna zásob - běžné ceny - sezónně očištěno NINV_SA_XDC |
|---|---|---|---|---|---|
| 2018-Q4 | 1 355 330 | 643 367 | 267 613 | 358 252 | -1 298 |
| 2018-Q3 | 1 334 709 | 638 831 | 269 518 | 362 921 | -4 120 |
| 2018-Q2 | 1 314 633 | 629 543 | 259 703 | 344 977 | -8 873 |
| 2018-Q1 | 1 305 626 | 622 669 | 257 586 | 329 015 | 8 670 |
| 2017-Q4 | 1 293 928 | 613 448 | 248 455 | 317 214 | 18 659 |
| 2017-Q3 | 1 278 087 | 604 225 | 242 985 | 321 989 | 15 586 |
| 2017-Q2 | 1 256 814 | 594 657 | 240 163 | 315 287 | 9 787 |
| 2017-Q1 | 1 224 999 | 583 302 | 236 835 | 299 741 | 9 747 |
| 2016-Q4 | 1 203 410 | 573 028 | 233 516 | 299 206 | 5 458 |
| 2016-Q3 | 1 195 161 | 563 459 | 231 555 | 300 707 | 7 026 |
| 2016-Q2 | 1 185 584 | 555 351 | 228 658 | 290 508 | 18 478 |
| 2016-Q1 | 1 181 683 | 550 231 | 224 949 | 297 427 | 19 076 |
| 2015-Q4 | 1 168 477 | 546 766 | 224 692 | 309 809 | 6 981 |

**Obr. 15 Běžný formát dat HDP**
Zdroj: převzato z (Český statistický úřad, 2019)

Pro potřeby této aplikace bude důležitý především první sloupec z obr. 16 výše, tedy HDP v kupních cenách s identifikátorem NGDP_SA_XDC. Dále bude důležitý vždy pouze poslední aktuální rok, tedy v tomto případě rok 2018. Jak je patrné z obrázku, data jsou rozdělena podle jednotlivých čtvrtletí na čtyři části. Výsledný aktuální HDP bude reprezentován součtem těchto čtyř čtvrtletí za poslední rok (nyní 2018).

### Index průmyslové produkce

Běžně čitelný formát dat ČSÚ pro tento údaj vypadá následovně:

Zobrazeno 1-30 záznamů z celkového počtu 110 na 4 stránkách
Velikost stránky 30 < 1 2 3 4 >

| Období | Index průmyslové produkce (bazický index, průměrný měsíc roku 2015 = 100) AIP_IX |
|---|---|
| 2019-02 | 109,5 |
| 2019-01 | 110,3 |
| 2018-12 | 100,8 |
| 2018-11 | 128,8 |
| 2018-10 | 126,4 |
| 2018-09 | 112,6 |
| 2018-08 | 108,3 |
| 2018-07 | 99,8 |
| 2018-06 | 119,9 |
| 2018-05 | 117,2 |
| 2018-04 | 111,0 |
| 2018-03 | 119,4 |
| 2018-02 | 107,9 |
| 2018-01 | 111,5 |
| 2017-12 | 102,4 |
| 2017-11 | 122,8 |
| 2017-10 | 118,9 |

**Obr. 16 Běžný formát dat indexu průmyslové produkce**
Zdroj: převzato z (Český statistický úřad, 2019)

Index průmyslové produkce nemá žádnou jednotku, jedná se pouze o koeficient. Navíc není počítán za čtvrtletí, jako HDP, ale je počítán za jednotlivý měsíc. Pro potřeby aplikace bude zobrazován jako průměr za poslední rok. Bude tedy potřeba získat součet indexů za jednotlivé měsíce, který bude následně vydělen 12 (počet měsíců). Zároveň je nutné, aby byl již dostupný veškerý počet dat, proto bude index získáván za poslední dostupný rok, v tomto případě 2018, aby měl maximální vypovídající hodnotu.

### Zaměstnanost

U zaměstnanosti ČSÚ eviduje dva základní údaje:

- Průměrný evidenční počet zaměstnanců přepočtený na plně zaměstnané
- Průměrný evidenční počet zaměstnanců v podnicích s 50 a více zaměstnanci v průmyslu)

Pro potřeby aplikace je důležitý první ze dvou údajů, jehož čitelný formát dat, který ČSÚ poskytuje, vypadá následovně:

Zobrazeno 1-30 záznamů z celkového počtu 36 na 2 stránkách
Velikost stránky 30 < 1 2 >

| Období | Průměrný evidenční počet zaměstnanců přepočtený na plně zaměstnané<br>LE_PE_NUM |
|---|---|
| 2018-Q4 | 4 094 |
| 2018-Q3 | 4 070 |
| 2018-Q2 | 4 066 |
| 2018-Q1 | 4 034 |
| 2017-Q4 | 4 057 |
| 2017-Q3 | 4 017 |
| 2017-Q2 | 3 994 |
| 2017-Q1 | 3 951 |
| 2016-Q4 | 3 971 |
| 2016-Q3 | 3 933 |
| 2016-Q2 | 3 926 |
| 2016-Q1 | 3 874 |
| 2015-Q4 | 3 910 |
| 2015-Q3 | 3 873 |

**Obr. 17 Běžný formát dat zaměstnanosti**
Zdroj: převzato z (Český statistický úřad, 2019)

Zaměstnanost je uvedena v tisících osob, a zároveň je rovněž počítána za jednotlivá čtvrtletí. V aplikaci bude zobrazena jako průměr za poslední dostupný rok, tedy součet jednotlivých čtvrtletí vydělený 4.

### Nezaměstnanost

Nezaměstnanost, na rozdíl od zaměstnanosti, je v datech ČSÚ zobrazena jako míra, tudíž nemá žádné jednotky. Obecná míra nezaměstnanosti se počítá u osob v

rozmezí věku 15 – 64 let. Běžně čitelný formát dat ČSÚ pro tento údaj vypadá následovně:

Zobrazeno 1-30 záznamů z celkového počtu 110 na 4 stránkách
Velikost stránky 30 < 1 2 3 4 >

| Období | Obecná míra nezaměstnanosti (15 až 64letých osob) LUR_PE_NUM |
|---|---|
| 2019-02 | 2,0 |
| 2019-01 | 2,1 |
| 2018-12 | 2,2 |
| 2018-11 | 2,0 |
| 2018-10 | 2,1 |
| 2018-09 | 2,2 |
| 2018-08 | 2,4 |
| 2018-07 | 2,4 |
| 2018-06 | 2,3 |
| 2018-05 | 2,3 |
| 2018-04 | 2,3 |
| 2018-03 | 2,3 |
| 2018-02 | 2,4 |
| 2018-01 | 2,4 |
| 2017-12 | 2,4 |
| 2017-11 | 2,5 |
| 2017-10 | 2,6 |

**Obr. 18 Běžný formát dat nezaměstnanosti**
Zdroj: převzato z (Český statistický úřad, 2019)

Údaj je evidován vždy za jednotlivý měsíc. Opět bude zobrazena průměrná míra za poslední dostupný rok, tedy součet jednotlivých měsíců za poslední dostupný rok vydělený 12.

### Mzdy / platy

U mezd a platů ČSÚ eviduje dvě základní skupiny:

- Průměrná hrubá mzda v Kč na osobu za měsíc
- Průměrná hrubá měsíční nominální mzda v Kč na osobu

Pro potřeby aplikace je důležitý první ze dvou údajů, jehož čitelný formát dat, který ČSÚ poskytuje, vypadá následovně:

Zobrazeno 1-30 záznamů z celkového počtu 36 na 2 stránkách
Velikost stránky 30 < 1 2 >

| Období | Průměrná hrubá mzda v Kč na osobu za měsíc (na přepočtené počty zaměstnanců) LCEA_XDC |
|---|---|
| 2018-Q4 | 33 840 |
| 2018-Q3 | 31 516 |
| 2018-Q2 | 31 876 |
| 2018-Q1 | 30 284 |
| 2017-Q4 | 31 661 |
| 2017-Q3 | 29 058 |
| 2017-Q2 | 29 335 |
| 2017-Q1 | 27 880 |
| 2016-Q4 | 29 491 |
| 2016-Q3 | 27 396 |
| 2016-Q2 | 27 452 |
| 2016-Q1 | 26 683 |

**Obr. 19 Běžný formát průměrné hrubé mzdy**
Zdroj: převzato z (Český statistický úřad, 2019)

Jak je patrné, průměrná hrubá mzda je evidována za jednotlivé roky v jednotlivých čtvrtletích. Výsledek bude zobrazen jako součet těchto dat vydělený 4.

### Spotřebitelské ceny

Index spotřebitelských cen je opět koeficientem, tudíž nemá žádné jednotky. Běžně čitelný formát dat ČSÚ pro tento údaj vypadá následovně:

Zobrazeno 1-30 záznamů z celkového počtu 111 na 4 stránkách
Velikost stránky 30 < 1 2 3 4 >

| Období | Index spotřebitelských cen (průměr roku 2015 = 100) PCPI_IX |
|---|---|
| 2019-03 | 107,5 |
| 2019-02 | 107,3 |
| 2019-01 | 107,1 |
| 2018-12 | 106,0 |
| 2018-11 | 105,9 |
| 2018-10 | 106,0 |
| 2018-09 | 105,6 |
| 2018-08 | 105,9 |
| 2018-07 | 105,8 |
| 2018-06 | 105,6 |
| 2018-05 | 105,2 |
| 2018-04 | 104,7 |
| 2018-03 | 104,4 |
| 2018-02 | 104,5 |
| 2018-01 | 104,5 |
| 2017-12 | 103,9 |

**Obr. 20 Běžný formát indexu spotřebitelských cen**
Zdroj: převzato z (Český statistický úřad, 2019)

Index spotřebitelských cen se zobrazuje za každý měsíc separátně, proto bude zobrazen jako součet za jednotlivé měsíce z posledního roku vydělený 12.

### Ceny výrobců (Index cen průmyslových výrobců)

Běžně čitelný formát dat ČSÚ pro tento údaj vypadá následovně:

Zobrazeno 1-30 záznamů z celkového počtu 110 na 4 stránkách
Velikost stránky 30 < 1 2 3 4 >

| Období | Index cen výrobců (průměr roku 2015 = 100) PPPI_IX |
|---|---|
| 2019-02 | 102,3 |
| 2019-01 | 102,0 |
| 2018-12 | 101,0 |
| 2018-11 | 102,1 |
| 2018-10 | 102,2 |
| 2018-09 | 101,5 |
| 2018-08 | 101,2 |
| 2018-07 | 101,1 |
| 2018-06 | 100,8 |
| 2018-05 | 100,2 |
| 2018-04 | 99,2 |
| 2018-03 | 99,0 |
| 2018-02 | 98,7 |
| 2018-01 | 99,1 |
| 2017-12 | 98,6 |
| 2017-11 | 98,3 |
| 2017-10 | 98,4 |
| 2017-09 | 98,4 |
| 2017-08 | 98,0 |

**Obr. 21 Běžný formát indexu cen výrobců**
Zdroj: převzato z (Český statistický úřad, 2019)

Ceny výrobců jsou zobrazeny jako koeficient dle jednotlivých měsíců, tudíž opět bude jejich průměr vypočítán jako součet za poslední dostupný rok, vydělený 12.

### Zahraniční obchod

U zahraničního obchodu rozlišuje ČSÚ 2 základní skupiny:

- Vývoz zboží v přeshraničním pojetí
- Dovoz zboží v přeshraničním pojetí

Pro potřeby aplikace bude důležitější vývoz zboží, protože ten představuje zisk pro stát. Běžně čitelný formát dat ČSÚ pro tento údaj vypadá následovně:

Zobrazeno 1-30 záznamů z celkového počtu 110 na 4 stránkách
Velikost stránky 30 < 1 2 3 4 >

| Období | Vývoz zboží v přeshraničním pojetí (v mil. Kč) TXG_FOB_XDC | Dovoz zboží v přeshraničním pojetí (v mil. Kč) TMG_CIF_XDC |
|---|---|---|
| 2019-02 | 359 092 | 318 370 |
| 2019-01 | 382 946 | 339 811 |
| 2018-12 | 317 099 | 297 280 |
| 2018-11 | 432 608 | 383 655 |
| 2018-10 | 434 400 | 398 746 |
| 2018-09 | 371 202 | 331 774 |
| 2018-08 | 343 198 | 329 050 |
| 2018-07 | 331 365 | 319 993 |
| 2018-06 | 377 211 | 341 161 |
| 2018-05 | 363 493 | 336 153 |
| 2018-04 | 354 831 | 314 256 |
| 2018-03 | 371 293 | 326 857 |
| 2018-02 | 337 624 | 298 398 |
| 2018-01 | 368 242 | 331 827 |
| 2017-12 | 309 974 | 288 757 |
| 2017-11 | 387 161 | 346 648 |

**Obr. 22 Běžný formát zahraničního obchodu**
Zdroj: převzato z (Český statistický úřad, 2019)

### Obyvatelstvo

U obyvatelstva ČSÚ eviduje tzv. střední stav obyvatelstva, což je počet obyvatel daného území v okamžiku, který byl zvolen za střed sledovaného období

(Český statistický úřad, 2019). Běžně čitelný formát dat ČSÚ pro tento údaj vypadá následovně:

Zobrazeno 1-30 záznamů z celkového počtu 36 na 2 stránkách
Velikost stránky 30 < 1 2 >

| Období | Střední stav obyvatelstva (v tis. osob)<br>LP_PE_NUM |
|---|---|
| 2018-Q4 | 10 646 |
| 2018-Q3 | 10 633 |
| 2018-Q2 | 10 619 |
| 2018-Q1 | 10 612 |
| 2017-Q4 | 10 606 |
| 2017-Q3 | 10 595 |
| 2017-Q2 | 10 584 |
| 2017-Q1 | 10 578 |
| 2016-Q4 | 10 577 |
| 2016-Q3 | 10 569 |
| 2016-Q2 | 10 561 |
| 2016-Q1 | 10 556 |

**Obr. 23 Běžný formát obyvatelstva**
Zdroj: převzato z (Český statistický úřad, 2019)

Střední stav se opět počítá jako průměr za čtvrtletí, proto výsledek bude interpretován jako součet za čtvrtletí, vydělený 4.

### 4.2.6 Parsování dat z ČSÚ

Nyní, když je jasně vymezená podoba a zdroj dat, je nutné data tzv. naparsovat, což znamená přeložit je z API do číselné podoby, se kterou se dá manipulovat v dané aplikaci. Za tímto účelem je nutné naprogramovat třídu, která se o tuto funkčnost bude starat, tzv. parser. Jak již bylo zmíněno, pro každý ekonomický údaj bude Parser separátní. Může se to jevit jako redundantní kód, na druhou stranu ekonomické údaje nejsou všechny stejné a nejsou ve stejných jednotkách. Navíc v průběhu času se mohou změnit a v takovém případě by se musela změnit celá podoba parseru. Když bude parser pro každý údaj, vždy se změní pouze určitá část kódu v případě nutnosti.

Příklad parseru HDP vypadá následovně:

```
package com.back4app.EconomicApp.parser;
import android.os.AsyncTask;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;
import org.xmlpull.v1.XmlPullParserFactory;
import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Calendar;

public class NationalAccountsParser extends AsyncTask {

        URL urlNationalAccounts;
        ArrayList<String> values = new ArrayList<>();
        long sum;

        @Override
        protected Object doInBackground(Object[] objects) {
        try {
        urlNationalAccounts = new
        URL("https://vdb.czso.cz/pll/eweb/sdmx.getXML?d=NAG");
        XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
        factory.setNamespaceAware(false);
        XmlPullParser xpp = factory.newPullParser();
        xpp.setInput(getInputStream(urlNationalAccounts),"UTF_8");
        boolean insideItem = false;
        int eventType = xpp.getEventType();
        while (eventType != XmlPullParser.END_DOCUMENT) {
          if (eventType == XmlPullParser.START_TAG) {
            if (xpp.getName().equalsIgnoreCase("eco:Series")) {
              if(xpp.getAttributeValue(null, "INDICATOR").matches("NGDP_SA_XDC")) {
                    insideItem = true;
              } else {
                    insideItem = false;
              }
            } else if (xpp.getName().equalsIgnoreCase("eco:Obs")) {
                int year = Calendar.getInstance().get(Calendar.YEAR);
                int rightYear = year - 1;
                if (insideItem) {
                  if (xpp.getAttributeValue(null, "TIME_PERIOD").matches(".*" + rightYear +
                  ".*")) {
                    values.add(xpp.getAttributeValue(null, "OBS_VALUE"));
                    System.out.println(xpp.getAttributeValue(null, "OBS_VALUE"));
                  }
                }
              }
          }
              else if (eventType == XmlPullParser.END_TAG &&
              xpp.getName().equalsIgnoreCase("item")) {
              insideItem = false;
          }
          eventType = xpp.next();
        }
```

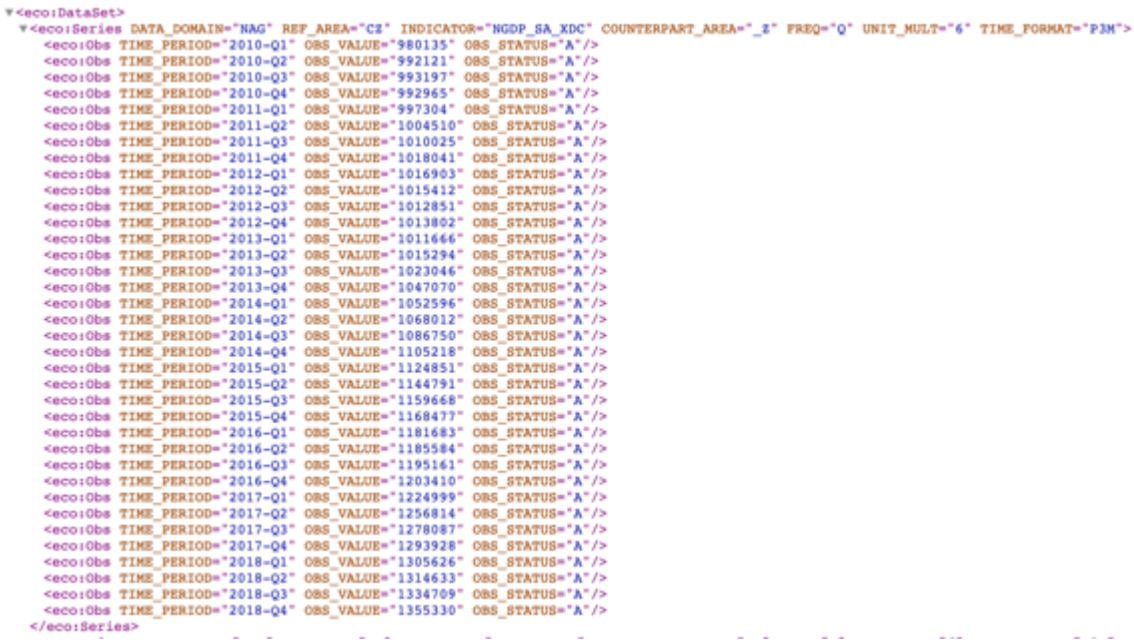
```
            } catch (MalformedURLException e) {
                    e.printStackTrace();
            } catch (XmlPullParserException e) {
                    e.printStackTrace();
            } catch (IOException e) {
                    e.printStackTrace();
            }
            for(int i = 0; i<values.size(); i++){
                    sum = sum + Long.parseLong(values.get(i));
            }
            values.clear();
            values.add("Actual HDP = "+String.valueOf(sum) + " CZK");

            return values;
    }

            public InputStream getInputStream(URL url) {
                    try {
                            return url.openConnection().getInputStream();
                    } catch (IOException e) {
                            return null;
                    }
                    }
            public ArrayList<String> heads() {
                    return values;
            }
}
```

**Kód 9 HDP parser**
Zdroj: Vlastní zpracování

Je nutné, aby Parser implementoval třídu AsyncTask, která se stará o asynchronní volání operací a dat, protože tento parser je vyvolán vždy při přihlášení uživatele, aby vždy po přihlášení byla dostupná poslední aktualizovaná data z ČSÚ. Dále je vhodné si předpřipravit URL, ze které se budou data získávat, seznam hodnot, které budou z URL získány, v tomto případě typ ArrayList a jednoduchou proměnnou typu long, do které se zaznamená výsledný součet polí.

Dále se již o parsování výsledků postará třída XMLPullParserFactory, ze které se získá instance xmlPullParser. Do tohoto parseru se poté musí dosadit tzv. InputStream, což je url adresa, ze které je potřeba data získávat. Parser pomocí while cyklu prochází dokument, dokud nedojde na jeho konec. Vždy je pomocí metody získán aktuální rok, od kterého je odečtena jednička (aby bylo zajištěno, že data budou vždy získaná za poslední dostupný rok v celku). Parser prochází dokument a hledá schodu se symbolem „NGDP_SA_XDC“, dále pak ověřuje, zda dokument obsahuje symboly „eco:Obs“, pokud ano, získá z nich hodnotu

„OBS_VALUE“, kterou uloží do připraveného ArrayListu. Význam těchto jednotlivých pojmenování je zřejmý z XML dokumenty níže, který se nachází na dané adrese parsování:




**Obr. 24 Ukázka XML parsovaného formátu**
Zdroj: převzato z (Český statistický úřad, 2019)

„eco:Obs“ je vždy název řádku a „OBS_VALUE“ reprezentuje hodnotou tohoto řádku. Tímto způsobem jsou parsovány všechny ostatní ekonomické ukazatele. Autor považuje za nadbytečné zahlcovat práci dalšími údaji o parsování, které jsou téměř totožné, proto zde nejsou uvedeny další příklady parserů, které pracují na obdobném principu, jen je například změněná vstupní podmínka podle toho, jestli se údaj počítá za čtvrtletí, za rok atd. Veškeré parsery spolu s ostatním kódem jsou pro úplnost k nahlédnutí v příloze č. 1, kde se nachází celá vyvíjená aplikace. Parsery jsou pro větší přehlednost umístěny do balíčku parser, který se nachází na kořenové úrovni celého projektu.

### 4.2.7 Zobrazování dat (dynamická rekonfigurace)

Nyní je hotový parser i registrace uživatelů. Poté, co se uživatel zaregistruje, je vhodné, aby se také mohl přihlašovat, proto bude potřeba implementovat i přihlašovací mechanismus. Tento mechanismus imeplementuje dynamickou rekonfiguraci, protože pokaždé, když je uživatel přihlašován, aplikace získá a naparsuje data z ČSÚ, která následně zobrazí. V případě větší aplikace by zde téměř

jistě nastával problém s výkonem, který byl popsán u dynamické rekonfigurace v teoretické části této práce. Vyvíjená aplikace ale neparsuje příliš mnoho dat, tudíž netrvá příliš dlouho, než data zobrazí, proto problém s výkonem není nutné řešit. Kód, který zprostředkovává samotné přihlášení uživatele je uložen v aktivitě LoginActivity a vypadá následovně:

```
package com.back4app.EconomicApp.authActivities;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import com.back4app.EconomicApp.R;
import com.back4app.EconomicApp.dataActivities.ActualDataDisplayActivity;
import com.parse.LogInCallback;
import com.parse.ParseException;
import com.parse.ParseInstallation;
import com.parse.ParseUser;

public class LoginActivity extends AppCompatActivity {
        private Button btnSignUp;
        private Button btnLogin;
        private EditText txtEmail;
        private EditText txtPassword;
        @Override
        protected void onCreate(Bundle savedInstanceState) {

                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_login);

                btnSignUp = (Button) findViewById(R.id.btnSignUp);
                TextView loginView = findViewById(R.id.loginView);

                btnSignUp.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View view) {
                        launchActivity();
                        }
                });
                btnLogin = (Button) findViewById(R.id.btnLogin);
                btnLogin.setOnClickListener(new View.OnClickListener() {

                        @Override
                        public void onClick(View view) {
                        txtEmail = findViewById(R.id.txtEmail);
                        txtPassword = findViewById(R.id.txtPassword);
                        ParseUser.logInInBackground(txtEmail.getText().toString(),
                        txtPassword.getText().toString(), new LogInCallback() {
```

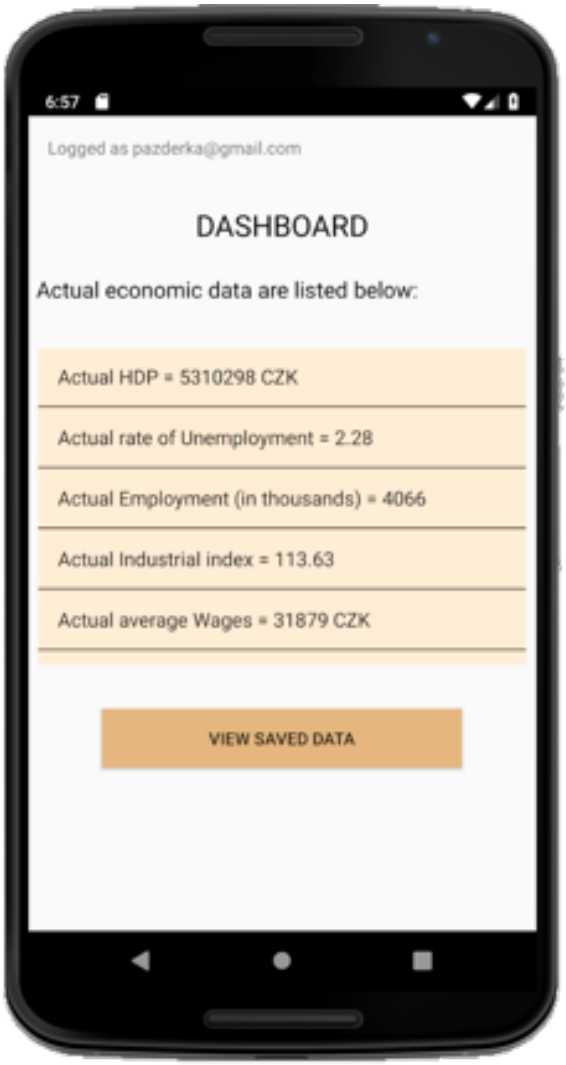
```
                                    @Override
                                    public void done(ParseUser user, ParseException e) {
                                      if(user != null) {
                                        Intent intent = new Intent(getApplicationContext(),
                                        ActualDataDisplayActivity.class);
                                        startActivity(intent);
                                    } else {
                                            Toast.makeText(LoginActivity.this, e.getMessage(),
                                            Toast.LENGTH_SHORT).show();
                                      }
                                    }
            });
        }
    });

            // Save the current Installation to Back4App
            ParseInstallation.getCurrentInstallation().saveInBackground();
    }

            private void launchActivity() {
                        Intent intent = new Intent(this, SignUpActivity.class);
                        startActivity(intent);
            }
}
```

**Kód 10 Přihlašovací aktivita**
Zdroj: Vlastní zpracování

Nejprve jsou předpřipraveny proměnné, do kterých budou při vytvoření aktivity vloženy instance z layoutu. Tlačítko signup přesměrovává na aktivitu registrace, která byla popsána v předchozí části práce. Nejprve je opět tlačítku Login přidělen listener kliknutí. Ten nejdříve získá zadaná data z polí uživatelského jména a hesla. Následně je zavolána metoda LogInBackground, která uživatele přihlásí do systému. Pokud uživatel zadá neplatné nebo žádné údaje, bude na to aplikací

upozorněn. Dále se již spustí aktivita ActualDataDisplayActivity, která je zodpovědná právě za realizaci daného parsování.

Uživatel po přihlášení uvidí následující obrazovku:




**Obr. 25 Obrazovka s ekonomickými daty**
Zdroj: Vlastní zpracování

Data, která jsou vidět v seznamu aplikace, reprezentují právě zmíněnou dynamickou rekonfiguraci, protože jsou při každém přihlášení uživatele znovu získávána z API ČSÚ, které bylo popsáno v předchozí části práce. Pokud uživatel na některou z položek klikne, zobrazí se mu vyskakovací okénko, zda si přeje vybraný ekonomický ukazatel uložit. Pokud uživatel zvolí možnost uložení, aplikace podá zprávu o úspěšném uložení (pokud úspěšné bylo). Kód aktivity zodpovědný za toto chování, je k dispozici níže:

```
package com.back4app.EconomicApp.dataActivities;
import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.DialogInterface;
import android.content.Intent;
import android.graphics.Color;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import com.back4app.EconomicApp.R;
import com.back4app.EconomicApp.parser.ConsumerPricesParser;
import com.back4app.EconomicApp.parser.EmploymentParser;
import com.back4app.EconomicApp.parser.IndustrialProductionIndexParser;
import com.back4app.EconomicApp.parser.InternationalTradeParser;
import com.back4app.EconomicApp.parser.NationalAccountsParser;
import com.back4app.EconomicApp.parser.PopulationParser;
import com.back4app.EconomicApp.parser.ProducerPricesParser;
import com.back4app.EconomicApp.parser.UnemploymentParser;
import com.back4app.EconomicApp.parser.WagesParser;
import com.google.gson.Gson;
import com.google.gson.JsonObject;
import com.parse.GetCallback;
import com.parse.ParseException;
import com.parse.ParseUser;
import com.parse.SaveCallback;
import java.util.List;
import java.util.concurrent.ExecutionException;

public class ActualDataDisplayActivity extends ListActivity {
        private TextView txtLoggedAs;
        ListView lv;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.actual_data_display);
                Button btnRedirect = findViewById(R.id.btnRedirect);

                btnRedirect.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View view) {
                        Intent intent = new Intent(ActualDataDisplayActivity.this,
                        SavedDataActivity.class);
                        startActivity(intent);
                        finish();
                }
        });
        List<String> headlines;
        NationalAccountsParser getXML = new NationalAccountsParser();
        UnemploymentParser unempl = new UnemploymentParser();
        IndustrialProductionIndexParser ind = new IndustrialProductionIndexParser();
```

```
EmploymentParser emp = new EmploymentParser();
WagesParser wp = new WagesParser();
ConsumerPricesParser cpp = new ConsumerPricesParser();
ProducerPricesParser ppp = new ProducerPricesParser();
InternationalTradeParser itp = new InternationalTradeParser();
PopulationParser pp = new PopulationParser();
try {
        getXML.execute().get();
        unempl.execute().get();
        ind.execute().get();
        emp.execute().get();
        wp.execute().get();
        cpp.execute().get();
        ppp.execute().get();
        itp.execute().get();
        pp.execute().get();
} catch (InterruptedException e) {
        e.printStackTrace();
} catch (ExecutionException e) {
        e.printStackTrace();
}
headlines = getXML.heads();
ArrayAdapter adapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1,
headlines);
setListAdapter(adapter);
String currentUser = ParseUser.getCurrentUser().getUsername();
txtLoggedAs = (TextView) findViewById(R.id.txtLoggedAs);
txtLoggedAs.setText("Logged as "+currentUser);
adapter.add("Actual rate of Unemployment = "
+Math.round((Float.parseFloat(unempl.heads().get(0)) / 12) * 100.0) / 100.0);
adapter.add("Actual Employment (in thousands) = "
+Integer.parseInt(emp.heads().get(0)) / 4);
adapter.add("Actual Industrial index = "
+Math.round((Float.parseFloat(ind.heads().get(0)) / 12) * 100.0) / 100.0);
adapter.add("Actual average Wages = "+Long.parseLong(wp.heads().get(0)) / 4 + "
CZK");
adapter.add("Actual index of Consumer prices = "
+Math.round((Float.parseFloat(cpp.heads().get(0)) / 12) * 100.0) / 100.0);
adapter.add("Actual index of Producer prices = "
+Math.round((Float.parseFloat(ppp.heads().get(0)) / 12) * 100.0) / 100.0);
adapter.add("Actual International trade (export) (millions CZK) = "
+Long.parseLong(itp.heads().get(0)) / 12 + " CZK");
adapter.add("Actual number of Population (thousands) = "
+Long.parseLong(pp.heads().get(0)) / 4);
adapter.notifyDataSetChanged();
saveChosenRecord();
}

private void saveChosenRecord() {
        lv = getListView();
        lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
```

```
                @Override
                public void onItemClick(final AdapterView<?> adapterView, View view,
                final int position, final long test2) {
                        AlertDialog.Builder builder = new
                        AlertDialog.Builder(ActualDataDisplayActivity.this)
                        .setTitle("Item action")
                        .setMessage("Do you wish to save this record?")
                        .setNeutralButton("Yes", new DialogInterface.OnClickListener() {
                          @Override
                          public void onClick(DialogInterface dialogInterface, int i) {
                            String selectedValue = (String)
                            adapterView.getItemAtPosition(position);
                            selectedValue.replaceAll("\\D", "");
                            final String selectedValue2 = selectedValue.replaceAll("[^0-
                            9.]", "");

                            final Object savedData =
                            ParseUser.getCurrentUser().get("savedData");
                            final Gson gson = new Gson();
                            String json = gson.toJson(savedData);

                            final JsonObject jobj = new Gson().fromJson(json,
                            JsonObject.class);
                              for(String keyStr: jobj.keySet()) {
                                if(selectedValue.contains(keyStr)) {
                                  jobj.addProperty(keyStr, selectedValue2);
                                  ParseUser
                                  .getQuery()
                                  .getInBackground(ParseUser.getCurrentUser()
                                  .getObjectId(), new GetCallback<ParseUser>() {
                            @Override
                            public void done(ParseUser object, ParseException e) {
                            object.put("savedData", gson.fromJson(jobj, Object.class));
                            object.saveInBackground(new SaveCallback() {
                              @Override
                              public void done(ParseException e) {
                                if (e == null) {
                                  Toast.makeText(getApplicationContext(), "Chosen
                                   record successfully saved",
                                  Toast.LENGTH_LONG).show();
                                 } else {
                                      Toast.makeText(getApplicationContext(),
                                      e.getMessage(), Toast.LENGTH_LONG).show();
                                 }
                                }
                            });
                          }
                        });
                    }
                }
            }
        }
    })
```

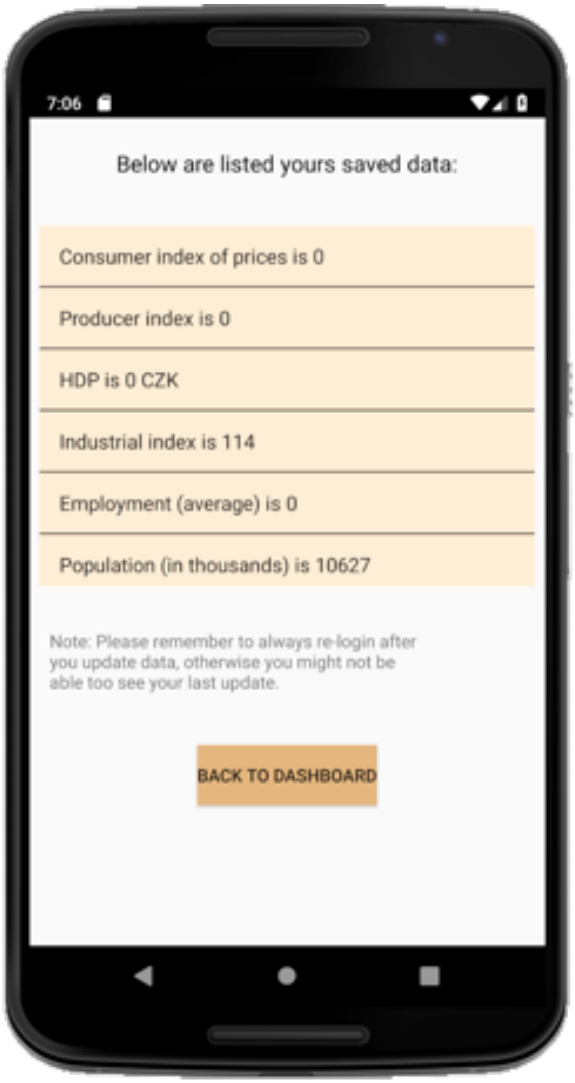
```
                                .setNegativeButton("No", new DialogInterface.OnClickListener() {
                                    @Override
                                    public void onClick(DialogInterface dialog, int which) {
                                            dialog.cancel();
                                    }
                            });
                    AlertDialog ok = builder.create();
                    ok.show();
                    ok.getButton(AlertDialog.BUTTON_NEUTRAL).setTextColor(Color.BLACK);
                    ok.getButton(AlertDialog.BUTTON_NEGATIVE).setTextColor(Color.BLACK);
                }
            });
        }
    }
```

**Kód 11 Zobrazování aktuálních ekonomických ukazatelů**
Zdroj: Vlastní zpracování

### 4.2.8 Ukládání dat (distribuovaná rekonfigurace)

Nyní, když aplikace již umí registrovat uživatele, přihlašovat uživatele, parsovat data a zobrazovat tato data uživatelům, je vhodné implementovat mechanismus, který tato data umožní uložit pro pozdější použití pro případ, že ČSÚ změní jejich hodnotu. Poté by totiž aplikace ukazovala jiné hodnoty a starší

parametry by nebylo možné vyhledat. K tomuto účelu bude sloužit aktivita SavedDataActivity, která vypadá následovně:




**Obr. 26 Uložená data uživatelů**
Zdroj: Vlastní zpracování

Na první pohled se může zdát, že aktivita vypadá téměř totožně, jako aktivita, která parsuje aktuální data. To je způsobeno tím, že jsou použity stejné designové komponenty, jako jsou tlačítka a seznam hodnot. Na pozadí však tato aktivita funguje zcela odlišně. Kód této aktivity je k dispozici níže:

```
package com.back4app.EconomicApp.dataActivities;
import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import com.back4app.EconomicApp.R;
import com.google.gson.Gson;
import com.google.gson.JsonObject;
import com.parse.ParseUser;
import java.util.ArrayList;
import java.util.List;

public class SavedDataActivity extends ListActivity {
        List<String> displayed = new ArrayList<>();
        ArrayAdapter adapter;
        Button btnGoBack;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_data_saved);
                btnGoBack = findViewById(R.id.btnGoBack);

                btnGoBack.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View view) {
                        Intent intent = new Intent(SavedDataActivity.this,
                        ActualDataDisplayActivity.class);
                        startActivity(intent);
                        finish();
                }
        });
findObjects();
}
        protected void findObjects() {
                Object savedData = ParseUser.getCurrentUser().get("savedData");
                Gson gson = new Gson();
                String json = gson.toJson(savedData);
                JsonObject jobj = new Gson().fromJson(json, JsonObject.class);
                for(String keyStr: jobj.keySet()) {
                        Object keyvalue = jobj.get(keyStr);
                        if(keyStr.contains("HDP")) {
                                keyStr = keyStr + " is ";
                                keyvalue = keyvalue + " CZK";
                        }

                        if(keyStr.contains("Consumer")) {
                                keyStr = keyStr + " index of prices is ";
                        }

                        if(keyStr.contains("Employment")) {
                                keyStr = keyStr + " (average) is ";
                        }
```

```
                                    if(keyStr.contains("Population")) {
                                            keyStr = keyStr + " (in thousands) is ";
                                    }

                                    if(keyStr.contains("Producer")) {
                                            keyStr = keyStr + " index is ";
                                    }

                                    if(keyStr.contains("Wages")) {
                                            keyStr = keyStr + " (average) are ";
                                    }

                                    if(keyStr.contains("Unemployment")) {
                                            keyStr = keyStr + " rate is ";
                                    }

                                    if(keyStr.contains("International")) {
                                            keyStr = keyStr + " trade (export) is ";
                                    }

                                    if(keyStr.contains("Industrial")) {
                                            keyStr = keyStr + " index is ";
                                            keyvalue =
                                            String
                                            .valueOf
                                            (Math.round((Float.parseFloat(keyvalue.toString()
                                            .replace("\"", "")) * 100.0) / 100.0));
                                    }

                            displayed.add((keyStr + keyvalue).replace("\"", ""));
                            }
                    adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1,
                    displayed);
                    setListAdapter(adapter);
                    adapter.notifyDataSetChanged();
            }
    }
```

**Kód 12 Uložená data uživatelů**
Zdroj: Vlastní zpracování

Aktivita rozšiřuje třídu ListActivity, stejně jako aktivita naparsovaných dat, protože jinak nelze data ze seznamu (ArrayListu) efektivně dostat do seznamu, který je zobrazen uživateli. Nejprve jsou opět předpřipraveny proměnné, jedna je typu ArrayList, která v sobě integruje veškerá uživatelem uložená data, která se čtou z databáze, ze sloupce savedData. Rovněž je zde přítomno tlačítko, které vrací uživatele na úvodní obrazovku se zobrazenými daty. Metodou getCurrentUser se následně vyhledá aktuálně přihlášený uživatel a načtou se uživatelova data z jeho sloupci v databázi, který se jmenuje savedData. Sloupec je, jak již bylo zmíněno, typu JSON, proto je nutné data obdobně jako v jiných aktivitách, které s tímto sloupcem

pracují, konvertovat pomocí Tříd Gson a JsonObject. Následně jsou pomocí if podmínek k jednotlivým ukazatelům přiřazeny drobné vysvětlivky, aby uživatel věděl, o jaká data se jedná. Všchna tato data jsou uložena do ArrayListu, který je pomocí adapteru připnut seznamu, který uživatel vidí v této aktivitě.

# 5 Shrnutí výsledků

V rámci této práce byly definovány a popsány metody rekonfigurace na platformě Android, které jsou v současné době nejvíce využívané. Byla provedena SWOT analýza, která definuje základní výhody, nevýhody, silné a slabé stránky metod rekonfigurací, na jejichž základě je posouzena vhodnost dílčích metod. Tato analýza je podpořena Ishikawovým diagramem, který reprezentuje hrozby v rámci managementu bezpečnostních rizik aplikace. Vzhledem k nedostatku české odborné literatury, která by se této problematice věnovala, vychází většina literatury ze zahraničních zdrojů.

Výsledná aplikace v praktické části této práce implementuje celkem dvě ze tří popsaných metod rekonfigurací, a to dynamickou a distribuovanou rekonfiguraci. Žádná aplikace nemůže totiž implementovat veškeré metody rekonfigurace, které byly v této práci popsány, protože je velmi obtížné najít takovou funkcionalitu, která by dávala v kontextu dané aplikace smysl, aby se do ní veškeré zmíněné metody vešly. Aplikace dle aktuálního uživatelského kontextu aktivně vyhodnocuje, jaký uživatel byl přihlášen, jaká data má uživateli zobrazovat a jaká data chce uživatel uložit do databáze. Aplikaci lze označit za tzv. fullstack produkt, který neimplementuje vlastní server, protože tento je nahrazen API, které poskytuje ČSÚ. Z tohoto API jsou čerpány základní ekonomické ukazatele, které jsou následně v aplikaci předkládány koncovému uživateli.

Toto API, které poskytuje ČSÚ, slouží zároveň jako element, který celou aplikaci zastřešuje a velmi usnadňuje její znovupoužití, protože pokud bude potřeba implementovat vlastní server, případně použít jiné API, stačí toto API odstranit a použít jiné. Samozřejmě je poté nutné udělat menší úpravy v kódu, aby dával smysl u hlediska pojmenovávacích konvencí, formátů atp. Celá aplikace je vyvinuta s ohledem na maximální znovupoužití, jsou oddělovány dílčí struktury kódu, které spolu významově souvisí, aby byla dodržena systémová logičnost výsledné aplikace. Jako databáze je využita MongoDB v cloudu, což zásadním způsobem přispívá ke znovupoužití, protože koncový uživatel nebude nucený aplikační databázovou

vrstvu předělávat. Bude mu stačit vlastnit účet na back4app a pouze nahradit klíč k aplikaci.

Při vývoji aplikaci byl kladen důraz na dodržování maximálních bezpečnostních principů, například ukládání hesel do databáze, kdy uživatel, který je v databázi přihlášený, nemá možnost si hesla zobrazovat. Lze konstatovat, že neexistuje žádná univerzální a nejvhodnější metoda rekonfigurace, která by se dala uplatnit maximálně efektivně pro jakoukoliv aplikaci. Z hlediska výkonu je nejvhodnější implementovat distribuovanou rekonfiguraci, z hlediska efektivnosti změny zobrazovaných dat zase dynamickou rekonfiguraci. Z hlediska výkonu se tak dá označit za nejefektivnější metoda distribuované a statické rekonfigurace.

# 6 Závěry a doporučení

Závěrem lze konstatovat, že byly naplněny základní cíle práce, tedy popis metod rekonfigurací na platformě Android spolu s managementem jejich rizik a vyhodnocením jejich vhodnosti použití. Při vývoji samotné aplikace byl kladen důraz na nejlepší praktiky v oblasti bezpečnosti vývoje mobilních aplikací, nicméně aplikace nedisponuje příliš velkou paletou pro tento mechanismus, jelikož je již v základu poměrně dobře zabezpečená. Sama o sobě nemanipuluje s žádnými daty, která by byla citlivá. Bezpečnost aplikace byla podpořena registračním a přihlašovacím mechanismem, tudíž se ani do dat aplikací nikdo neautorizovaný nedostane, na druhou stranu je to příležitost, jak zneužít citlivá uživatelská data, jako jsou emaily a hesla.

Do cloudu aplikace se ale je rovněž možné dostat jen pod přihlášením, kterým disponuje pouze autor práce. Navíc ani v cloudu není možné zobrazovat hesla registrovaných uživatelů, tudíž nehrozí riziko ani v případě, že by se do cloudu přihlašoval někdo jiný. Funkcionalita aplikace je dostatečná k tomu, aby bylo možné demonstrovat v cíli práce nutné požadavky, nicméně v případě dalšího vývoje aplikace by bylo vhodné implementovat například vykreslování dat z ekonomických ukazatelů, které jsou získávány z API ČSÚ. Zároveň by bylo vhodné implementovat mechanismus, který by uživatelům umožňoval zobrazovat uložená data ihned poté, co je uloží, bez nutnosti opětovného přihlášení.

Na zvážení je zavedení principu lokalizace uživatelů. Jedná se sice o pasivní uživatelských kontext, tudíž by se nabízelo implementovat tento mechanismus i do této aplikace, na druhou stranu ale nedává z hlediska použití smysl, protože aplikace využívá API ČSÚ, které obsahuje výhradně informace o České republice, proto je nepravděpodobné, že by aplikaci chtěl používat i někdo ze zahraničí. V případě rozšíření aplikace na mezinárodní úroveň není příliš velkým problémem tento mechanismus přidat. Dále by bylo vhodné zahrnout do výsledné aplikace i unity testování. Do této aplikace nebyly testy zahrnuty, protože nemají žádnou bližší souvislost s cílem práce a tím pádem by zde byly nadbytečné. Pro tuto velikost

aplikace je možné veškerou její funkčnost otestovat ručně. Kromě toho testování aplikací je kapitola, o které by se dala napsat samostatná práce.

# 7 Seznam použité literatury


[1] ALLEN, Grant. Android 4: Průvodce programováním mobilních aplikací. 1. Brno: Computer Press, 2013. ISBN 978-80-251-3782-6.

[2] ALTUWAIJRI, Haya a Sanaa GHOUZALI. Android data storage security: A review. Journal of King Saud University – Computer and Information Sciences [online]. 2018, 2018, 1-10 [cit. 2019-01-13]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1319157818301046?via%3Dihub

[3] Android developers. Android developers [online]. Google, 2019 [cit. 2019-03-01]. Dostupné z: https://developer.android.com/topic/security/best-practices#j

[4] BURD, Barry. Java Programming for Android Developers For Dummies, 2nd Edition. Druhé. New York: John Wiley, 2016. ISBN 978-1-119-30109-7.

[5] Český statistický úřad [online]. Česká republika, 2019 [cit. 2019-04-12]. Dostupné z: https://www.czso.cz/

[6] DEYPIR, Mahmood a Abbas HORRI. Instance based security risk value estimation for Android applications. Journal of information security and applications [online]. 2018, 2018, 20-30 [cit. 2019-03-01]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2214212616300941

[7] HARTMANN, Sven a Sebastian LINK. Numerical constraints on XML data. Elsevier [online]. 2010, 2010, 521-544 [cit. 2019-03-01]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0890540109002399

[8] KOBUSIŃSKA, Anna a Ching-Hsien HSU. Towards increasing reliability of clouds environments with RESTful web services. Future Generation Computer Systems [online]. 2018, 2018(87), 502-513 [cit. 2019-01-13]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0167739X17316709?via%3Dihub

[9] Management mania. Management mania [online]. Česká republika, 2015 [cit. 2019-03-02]. Dostupné z: https://managementmania.com/cs/ishikawuv-diagram

[10] Minimální požadavky na kryptografické algoritmy. NÚKIB [online]. Praha, 2018, 28.11. 2018 [cit. 2019-03-01]. Dostupné z: https://www.govcert.cz/download/uredni-deska/Kryprografick%C3%A9%20prost%C5%99edky/Kryptograficke_prostredky_doporuceni_v1.0.pdf

[11] PASCUAL, Gustavo G., Roberto E. LOPEZ-HERREJON, Mónica PINTO, Lidia FUENTES a Alexander EGYED. Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications. The Journal of Systems and Software [online]. 2015, 2015(103), 392-411 [cit. 2019-01-13]. Dostupné z:

https://www.sciencedirect.com/science/article/pii/S016412121400291X?via%3Dihub

[12] PHADERMROD, Boonyara, Richard CROWDER a Gary WILLS. Importance-performance analysis based SWOT analysis. International journal of information management [online]. 2019, 2019, 194-203 [cit. 2019-03-01]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0268401216301694

[13] SANDOVAL, Kristopher. Nordic Apis. Nordic Apis [online]. 2016, únor 2016 [cit. 2019-03-01]. Dostupné z: https://nordicapis.com/what-data-formats-should-my-api-support/

[14] SAWADA, Yuki, Yusuke ARAI, Kanemitsu OOTSU, Takashi YOKOTA a Takeshi OHKAWA. Performance of android cluster system allowing dynamic node refonfiguration. Springer US [online]. 2017, únor 2017, 2017, 1067-1087 [cit. 2019-03-01]. DOI: 10.1007/s11277-017-3978-9. Dostupné z: https://link.springer.com/article/10.1007/s11277-017-3978-9

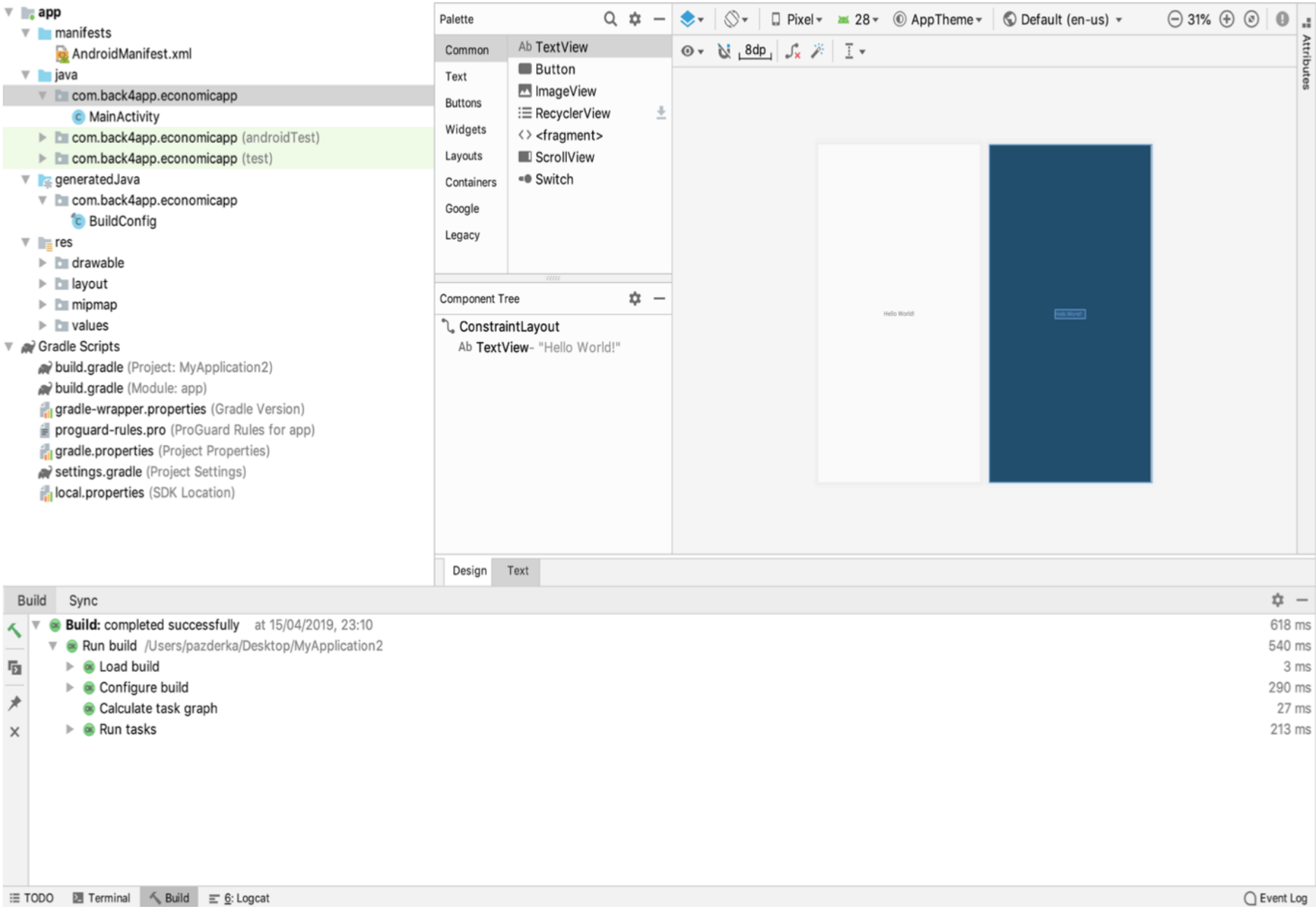
[15] SCHMITT, Mathias, Marius ORFGEN a Detlef ZÜHLKE. Dynamic reconfiguration of intelligent field devices by using modular software applications. Elsevier [online]. 2015, 2015, 561-566 [cit. 2019-03-01]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2405896315003791

[16] THANIGAIVELAN, Nanda Kumar, Ethiopia NIGUSSIE, Antti HAKKALA, Seppo VIRTANEN a Jouni ISOAHO. CoDRA: Context-based dynamically reconfigurable access control system for android. Journal of Network and Computer Applications [online]. 2018, 2018(101), 1-17 [cit. 2019-01-13]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S108480451730351X?via%3Dihub

[17] VASCONCELOS, Rafael, Igor VASCONCELOS a Markus ENDLER. Dynamic and coordinated software reconfiguration in distributed data stream systems. : Journal of Internet Services and Applications [online]. 2016, 2016, 1-21 [cit. 2019-03-01]. DOI: 10.1186/s13174-016-0050-z. Dostupné z: https://link.springer.com/article/10.1186/s13174-016-0050-z

# 8 Přílohy

1) Struktura projektu Android studia
2) Naprogramovaná aplikace (EconomicApp)


app
manifests
AndroidManifest.xml
java
com.back4app.economicapp
MainActivity
com.back4app.economicapp (androidTest)
com.back4app.economicapp (test)
generatedJava
com.back4app.economicapp
BuildConfig
res
drawable
layout
mipmap
values
Gradle Scripts
build.gradle (Project: MyApplication2)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Google
Legacy
TextView
Button
ImageView
RecyclerView
<fragment>
ScrollView
Switch
Component Tree
ConstraintLayout
TextView- "Hello World!"
Pixel
28
AppTheme
Default (en-us)
31%
8dp
Attributes
Design
Text
Build
Sync
Build: completed successfully at 15/04/2019, 23:10
618 ms
Run build /Users/pazderka/Desktop/MyApplication2
540 ms
Load build
3 ms
Configure build
290 ms
Calculate task graph
27 ms
Run tasks
213 ms
TODO
Terminal
Build
6: Logcat
Event Log

Příloha č. 1

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2018/2019

Studijní program: Systémové inženýrství a informatika
Forma: Prezenční
Obor/komb.: Informační management (im2-p)

**Podklad pro zadání BAKALÁŘSKÉ práce studenta**

| PŘEDKLÁDÁ: | ADRESA | OSOBNÍ ČÍSLO |
|---|---|---|
| Pazderka Přemysl | Jateční 802, Kolín - Kolín IV | 11600753 |

TÉMA ČESKY:

Rekonfigurace aplikace na základě kontextu uživatele

TÉMA ANGLICKY:

Reconfiguration of application based on user's context

VEDOUCÍ PRÁCE:

doc. Ing. Filip Maly, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl:
Navrhnout postupy, jak na základě aktuálního kontextu uživatele mobilního přístroje (aplikace) provádět změny v konfiguraci aplikace spuštěné na daném mobilním zařízení. Tyto postupy řádně popsat a demonstrovat na ukázkové aplikaci, navrhnout rámec, který tyto postupy zastřeší a umožní jejich znovupoužití.

Osnova:
1. Úvod
2. Metody rekonfigurace
3. Management rizik na platformě android
4. Implementace vlastního řešení rekonfigurace
5. Výsledky, závěr
6. Literatura

SEZNAM DOPORUČENÉ LITERATURY:

ALLEN, Grant. Android 4: průvodce programováním mobilních aplikací. ISBN 978-80-251-3782-6.
BURD, Barry. Java programming for android developers. 2nd edition. ISBN 978-1-119-30109-7.
OLIVEIRA, Juliana. An Exploratory Study of Exception Handling Behavior in Evolving Android and Java Applications ISBN: 978-1-4503-4201-8
NIGUSSIE, Ethiopia, Antti HAKKALA, Seppo VIRTANEN a Jouni ISOAHO. CoDRA: Context-based dynamically reconfigurable access control system for android. DOI: https://doi.org/10.1016/j.jnca.2017.10.015

Podpis studenta: ..... Datum: 9.10.2018

Podpis vedoucího práce: ..... Datum: 9.10.2018