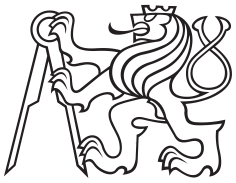
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Adaptace UI založena na emočních vzorech chování uživatele

Marek Klement

Vedoucí: Ing. Jiří Šebek
Obor: Software
Studijní program: Otevřená informatika
Květen 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Klement** Jméno: **Marek** Osobní číslo: **465928**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Adaptace UI založena na emočních vzorech chování uživatele

Název bakalářské práce anglicky:

UI adaptation based on emotional patterns of user behavior

Pokyny pro vypracování:

Prostudujte možnosti adaptivních uživatelských rozhraní [2] a nástrojů pro zachycení emocí uživatele. Rozšiřte framework [3] pro Android zařízení s možností adaptací UI dle emocí pro ukládání jednotlivých emocí do stromů celé aplikace. Vytvořte seznam vzorů podle kterých se aplikace bude měnit a zlepší se tím její usability, zrychlý se čas strávený nad danými úkoly. Důležitá část práce je správně navrhnout model ukládání kontextových informací a model změn adaptací. Proveďte testy funkčnosti a uživatelské testy podle HCI standard. Demonstrujte použití na modelovém příkladě. Výsledná implementace musí být snadno rozšiřitelná. Zhodnoťte výhody a možná omezení řešení. Prostudovat možnosti vývoje adaptivní uživatelských rozhraní v mobilních aplikacích a zachycení emocí uživatele. Testovat framework na známých strukturách v určitém okruhu uživatelů

Seznam doporučené literatury:

1. ŠEBEK, J. and K. RICHTA. Aspect-oriented User Interface Design for Android Applications [online]. In: DATESO 2015. Databases, Texts, Specifications, and Objects 2015, Nepřívěc u Sobotky, Jičín, 2015-04-14/2015-04-16. Praha: MATFYZPRESS, vydavatelství Matematicko-fyzikální fakulty UK, 2015, pp. 121-130. CEUR Workshop Proceedings. vol. 1343. ISSN 1613-0073. ISBN 9788073782856. Available from: http://www.cs.vsb.cz/dateso/2015/
2. Šebek, J. - Richta, K.: Usage of Aspect-Oriented programming in Adaptive Application Structure. New Trends in Databases and Information Systems: ADBIS 2016 Short Papers and Workshops, BigDap, DCSA, DC, Prague, Czech Republic, August 28-31, 2016, Proceedings
3. Lunova , A. – Šebek, J.:Adaptace UI založena na emocích uživatele , bakalářská práce, České vysoké učení technické v Praze, 2017-05-20

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jiří Šebek, kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2019** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Jiří Šebek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Rád bych poděkoval mému školiteli Ing. Jiřímu Šebkovi za skvělé vedení práce a cenné rady, které mi pomohly při psaní této práce. Rovněž děkuji své rodině a své přítelkyni za to, že mi byli oporou po celou dobu studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 18. května 2020

Abstrakt

Tato bakalářská práce se zaměřuje na vývoj frameworku pro adaptaci uživatelského rozhraní na základě emocí uživatele. Práce klade vysoký důraz na prozkoumání dostupných studií o zachycení emocí uživatelů a jejich použití pro odhalení nesrovnalostí v uživatelských rozhraních. Tyto studie pak práce využívá pro návrh několika možných navázání emocí na struktury aplikace a stanovení intervalů pro změnu právě těchto struktur. Hlavním cílem této práce je již zmíněný framework, který rozeznává emoce uživatele a adaptuje strukturu aplikace dle jeho potřeb. Na závěr je aplikace otestována na uživatelích pro porovnání výsledků několika druhů adaptivních struktur.

Klíčová slova: Adaptivní uživatelské rozhraní, Adaptivní struktura aplikace, OS Android, Emoce

Vedoucí: Ing. Jiří Šebek

Abstract

This bachelor thesis focuses on the development of a framework for adapting the user interface based on user emotions. The base of this work is exploring the possibilities of studying the recognition of user emotions and their use to detect inconsistencies in user interfaces. These studies are used for emotion-based structures of applications and determine the interval for changing these structures. The main aspect of this work is the abovementioned framework for recognizing the emotions of users and adapting the structure of the application according to its requirements. Finally, the application is tested on users to compare a couple of different adaptive structures.

Keywords: Adaptive user interface, Adaptive structure, Android OS, Emotions

Title translation: UI adaptation based on emotional patterns of user behavior

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

1.1 Motivace

V dnešní době je člověk obklíčen moderními technologiemi. Kolem nás můžeme vidět nejen mobilní telefony, notebooky, ale také chytré televize a dokonce i chytré toalety [3]. Přibližně polovina času [4] při práci na softwaru je investována do tvorby uživatelského rozhraní. Uživatelské rozhraní poskytuje uživateli moc nad daným zařízením a umožňuje mu jej používat. Každý uživatel je jiný, jeden vyznává rychlost a snadno se může ukliknout, jiný pečlivě a logicky proklikává aplikaci a snaží se najít svůj cíl. Pro záměr zjednodušit práci každého uživatele aplikace, existuje několik možností: [5]

1. Nedělat nic - Spousta softwarů v těchto dnech neaktualizuje své verze a pouze uživatele odkazuje na stránky s dokumentací, pokud jsou dostupné.
2. Podpora - Zajistit podporu uživatelů a pomoc při zotavení z negativních emočních stavů způsobených aplikací.
3. Nastavení - Umožnit uživateli dynamicky měnit software a přizpůsobit si ho vlastním požadavkům.
4. Open source - Umožnit uživateli být distributorem projektu a rozvinout si ho dle vlastního uvážení.

Prvním a čtvrtým bodem se v této práci nebudeme zabývat. Třetí bod představuje adaptabilní [6] aplikace, kde si uživatel sám nastaví vše potřebné

pro snadné používání. Toto řešení není špatné, ale pro optimální funkci předpokládá zapojení samotného uživatele. Problémem je, že uživatel sám většinou neví, co přesně chce a je těžké pro něj tyto preference nastavit. Dalším úskalím tohoto řešení je pak samotný uživatel, který tyto preference nechce, nebo neumí nastavit.

Proto se dostáváme k bodu dva. Jedná se o přizpůsobení aplikace každému uživateli na míru. Toto řešení nezní vůbec špatně a je v něm vidět potenciál. Bohužel nemůžeme chtít po vývojářích, aby vytvořili ručně tisíce ne-li miliony různých druhů struktur přesně na míru každému uživateli.

Existuje však možný kompromis. Aplikace, která bude na základě emocí uživatele poskytovat přizpůsobení každému zvlášť, aniž by musel uživatel sám cokoliv nastavovat a přizpůsobovat. Každý uživatel tak získá svou vlastní přizpůsobenou strukturu, která naprosto přesně vyhovuje jeho požadavkům. Toto řešení se nazývá adaptace [6] uživatelského rozhraní.

1.2 Cíle

Cílem této práce je definovat možnosti a problémy pro úpravu různých druhů stromových struktur aplikace. Dále pak vyčíst jednotlivé problémy s jejich klady a zápory, řešením a také s ukázkou výsledku adaptace. Součástí těchto úkonů bude také výčet různých druhů emocí, stanovení intervalů emocí a jejich využití pro adaptaci struktury aplikace. Následně má práce za cíl navrhnout model frameworku, který adaptuje strukturu aplikace právě na základě emocí uživatele.

Implementace samotná je dalším cílem této práce a jejím výsledkem bude rozšíření dostupného frameworku [7] o uzpůsobení pro adaptaci struktury uživatelského rozhraní.

Posledním cílem je testování vytvořeného frameworku na modelové aplikaci. Pro tuto aplikaci bude potřeba vybrat některou z existujících a velmi používaných struktur. Testování bude provedeno na okruhu uživatelů a výsledky budou v této práci zobrazeny a zapracovány.

Kapitola 2

Rešerše

2.1 Definice pojmů

Tato část definuje všechny frekventovaně používané nebo užitečné pojmy a poskytuje tak čtenáři větší vhled do problematiky.

2.1.1 Uživatelské rozhraní

Uživatelské rozhraní, zkráceně UI (z anglického user interface), zajišťuje uživatelům kontrolu nad hardwarem nebo softwarem. Poskytuje uživateli sérii úkonů či příkazů, na základě kterých je uživatel schopen dané zařízení jednoduše ovládat. Dobré uživatelské rozhraní [8] je charakteristické tím, že má vysoký stupeň použitelnosti a je intuitivní. Existuje mnoho typů uživatelského rozhraní od myši až po obrazovku. V této práci se zaměříme hlavně na obrazovku chytrého telefonu, kde budeme toto uživatelské rozhraní dynamicky upravovat.

2.1.2 Kontext

Kontextem [9] v prostředí vývoje mobilních aplikací a frameworků je myšleno zachycení a využití kontextuálních informací, jako například lokace, čas nebo informace o uživateli.

Součástí kontextu mohou být i méně prozkoumané odvětví vývoje mobilních aplikací a to eye-tracking (sledování zorniček) nebo sledování emocí. Právě sledování emocí je součástí zaměření této práce. Uživatel vykazuje nemalé množství emocí při procházení příslušné mobilní aplikace. Tyto emoce se dají získat několika způsoby: [10]

- Detekovat určité druhy emoce lze například zachycením souvisejících signálů, jako je tep srdce, pomocí externích zařízení tomu uzpůsobených. Tyto signály jsou ale často vyvolány i mnoha dalšími událostmi v lidském těle. [11]
- Údaje z GPS, záznamu rychlosti či zvukových periférií mohou být také ukazatelem emoce uživatele. [10]
- Kombinace hlasu a pohybu ústy může poskytnout vhled do emocí uživatele, ale vyžaduje to použití mikrofonu a kamery, což je velmi nákladné na spotřebu baterie. [12]
- Snímání pohybů prsty na obrazovce [13]
- Záznam výrazu obličeje [14]

2.1.3 Druhy uživatelského rozhraní

Statické uživatelské rozhraní

[15] Statické uživatelské rozhraní zůstává stále stejné v průběhu práce s ním. Uživatel si ho nemůže nijak upravit a zároveň nenastává automatická změna. Jedná se o základní druh uživatelského rozhraní.

Dynamické uživatelské rozhraní

[15] Uživatelské rozhraní pro tento přístup se mění při uživatelském používání. Existují dva hlavní směry dynamického přístupu: adaptabilní a adaptivní

Adaptabilní uživatelské rozhraní

[6] Adaptabilní uživatelské rozhraní je definováno jako systém, ve kterém je uživatelská změna (přizpůsobivost) provedena koncovým uživatelem. Uživatel si tak může libovolně sám nastavit preference programu. Adaptabilita je založena na znalostech uživatele a instinktu k tomu, jak by měla jeho aplikace fungovat. [2] Hlavní výhodou adaptabilních uživatelských rozhraní je fakt, že uživatel má absolutní kontrolu nad individuálním vzhledem.

Adaptivní uživatelské rozhraní

[6] Toto uživatelské rozhraní je definováno jako systém, který umí přizpůsobit aspekty jeho struktury nebo funkcionality pro potřeby jednotlivých uživatelů v daném čase. Adaptivní systémy jsou založeny na tom principu, že systém má být zodpovědný za identifikaci okolností, které vyžadují adaptaci. [2] Výhodou adaptivního uživatelského rozhraní je, že pomáhá zlepšit interakci s aplikací a snižuje potřebu uživatele požádat o pomoc s komplexními průchody.

V tabulce 2.1 najdete výčet některých kladů a záporů adaptabilních resp. adaptivních uživatelských rozhraní.

Typ UI	Výhody	Nevýhody
Adaptabilní	Jednoduchost	Vyžaduje znalost konkrétního uživatelského rozhraní
	Jednoduché na naučení	Není dostupné pro komplexní systémy
	Podpora uživatele	Může poškodit soukromí
Adaptivní	Nevyrušuje	Nejvíce invazivní
	Předpovídá chování jedince	Rušivé a obtěžující
	Snižuje kognitivní zatížení uživatele	Může poškodit soukromí

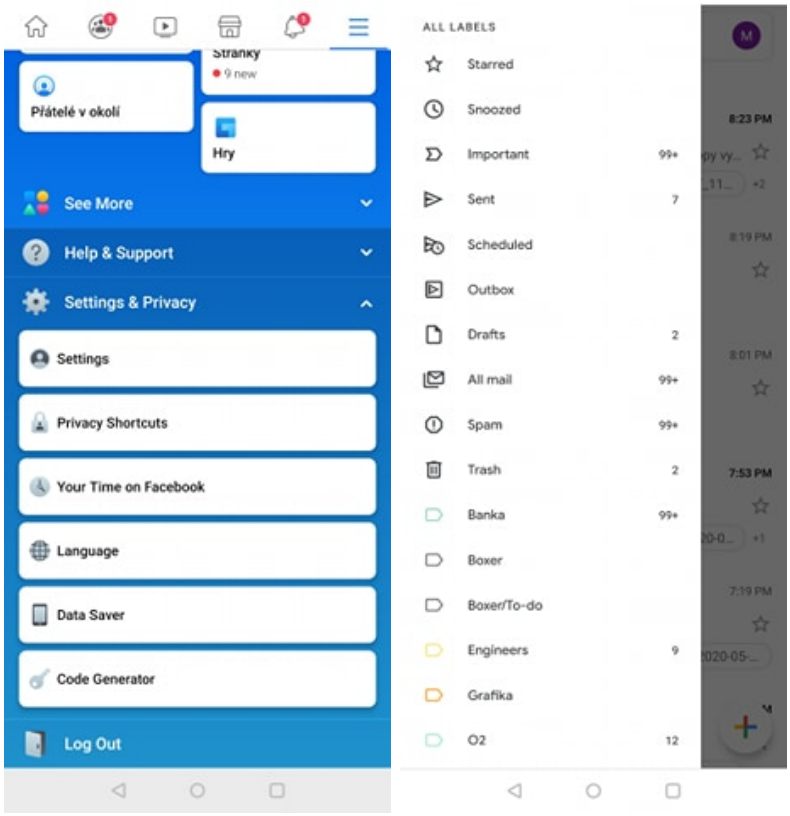
Tabulka 2.1: Výhody a nevýhody dvou dynamických přístupů. [2]

2.2 Definice a analýza problémů

2.2.1 Problém: Výběr struktury aplikace

Na začátek bude dobré si vymezit, jaké struktury aplikací je třeba využít pro adaptaci. Vezmeme si na zkoumání několik různých aplikací z chytrého telefonu - **Gmail, Air Bank, Facebook a Zprávy**. Výběr těchto aplikací je odůvodněn hlavně tím, že jsou globálně velmi používané. V následujících bodech jsou krátké statistiky o vybraných aplikacích:

- **Facebooku** má až 2.45 miliard uživatelů. [16]
- **Gmail** má až 1.5 miliardy uživatelů a 75% z nich navštěvuje Gmail z mobilu. [17]
- Mobilní telefon byl původně hlavně na psaní zpráv a telefonování. Na nových chytrých telefonech nechybí ani velmi využívaná aplikace **zpráv**.
- Aplikace **Air Bank** je z Českého prostředí a za rok 2018 se počet uživatelů zvýšil o 36% na 300 tisíc. [18]



Obrázek 2.1: Ukázka menu aplikací Facebook (vlevo) a Gmail (vpravo)

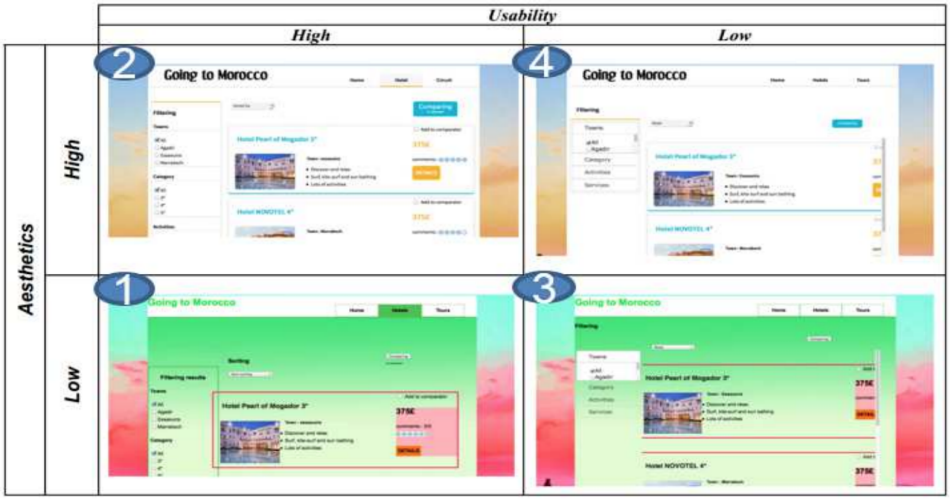
Nyní si definujeme, co znamená hloubka struktury. Strukturu si můžeme představit jako strom, každé další patro stromu představuje o jednu vyšší hloubku. Například cesta v Air Bank ke kontaktům představuje hloubku 3 = hlavní stránka - menu - kontakty. [19]

Pokud projdeme námi vybrané aplikace, uvidíme, že vyšší hloubku má právě Air Bank a také Facebook. U aplikace Facebook je to způsobené hlavně uživatelským nastavením, které je velmi zanořené a není tak přímo přístupné. Proto bude dobré nastavení zanedbat, protože uživatel do něj nebude tak často chodit. Gmail a Zprávy nemají hloubku vyšší než 3, protože již na úvodní stránce jsou potřebné informace, tedy koncová funkce.

Zbývá nám aplikace Air Bank, která má velmi hluboký strom struktury. Není se čemu divit, bankovní aplikace se snaží lidem nabídnout co nejvíce možností. Při bližším zkoumání dalších bankovních aplikací bylo zjištěno, že všechny, až na výjimky, mají alespoň hloubku stupně 4. Pro náš účel si tedy v testovací aplikaci vypůjčíme část struktury aplikace Air Bank.

2.2.2 Problém: Druhy emocí a jejich vliv na uzly stromové struktury aplikace

V roce 2018 probíhal průzkum [1], který sledoval emoce uživatele při testování webových stránek. Uživatelé byli rozděleni do čtyř skupin - nejprve podle pohlaví a pak podle věkové kategorie. Podle věkové kategorie byli rozděleni na mladší 27 let a starší 27 let. Tato věková hranice byla stanovena výzkumníky na 27 let, aby ve všech 4 skupinách bylo stejné množství účastníků. Každý dostal za úkol projít čtyři různé webové stránky (viz. obrázek 2.2). Tyto stránky reprezentovaly dva hlavní problémy - problém se vzhledem a problém s použitelností. Problém se vzhledem se týkal kontrastu barev a zavádějících nadpisů, naopak problém s použitelností se zaobíral špatným umístěním důležitých položek.



Obrázek 2.2: Webové stránky k testování (1. Problém se vzhledem, 2. Bez problémů, 3. Problém se vzhledem a použitelnosti, 4. Problém s použitelnosti). Převzato z [1]

Hlavním záměrem této práce bude právě bod čtvrtý - použitelnost. Proto budeme věnovat pozornost výsledkům třetího a čtvrtého problému, které se týkají špatného umístění důležitých uzlů struktury aplikace. V tabulce 2.2 jsou zachyceny pouze výsledky obrázků 3 a 4 a jsou v ní využité následující logické operátory:

- nebo - **or**
- a zároveň - **and**

Pro demonstraci si rozebereme význam těchto výroků:

- zlost **or** strach = převažuje zlost nebo strach
- strach **and** štěstí = převažuje strach a zároveň štěstí

Skupina	**Problém se vzhledem a použitelnosti**	**Problém s použitelnosti**
Ženy < 27	strach **or** znechucení **or** neutrální **or** opovržení **or** smutek	znechucení **or** zlost **or** štěstí **or** strach **or** smutek **or** opovržení
Ženy > 27	zlost	strach **and** štěstí
Muži < 27	štěstí **and** neutrální **or** strach **or** štěstí **or** neutrální	zlost **or** smutek **or** opovržení
Muži > 27	zlost	-

Tabulka 2.2: Výsledky průzkumu. Převzato a upraveno z [1]

Jak je vidět z tabulky 2.2, u žen mladších 27 let bylo zaznamenáno mnoho emocí. Z důvodu zjednodušení následujícího šetření se budeme muset omezit pouze na některé emoce. Dáme proto přednost emocím, které jsou obsažené v obou sloupcích - znechucení, opovržení, strach a smutek. Dále k nim přidáme emoce pouze z druhého sloupce - zlost a štěstí.

U žen starších 27 let nejsou v obou sloupcích žádné společné typy emocí. Můžeme usuzovat, že zlost z prvního sloupce je způsobena právě špatným vzhledem a zaměřit se primárně na strach a štěstí.

U mužů mladších 27 let se opět setkáváme s problémem, že v obou sloupcích jsou jiné hodnoty emocí. Řešení bude stejné jako u žen starších 27 let, takže budeme vnímat pouze emoce v druhém sloupci, který indikuje problém s použitelnosti aplikace, a to zlost, smutek a opovržení. Pro kategorii mužů nad 27 let je evidentně těžké rozpoznat emoce, a proto je v tabulce pouze zlost. Výsledky jsou k nahlédnutí v tabulce 2.3.

Pohlaví	Věk	Emoce
Ženy	< 27	znechucení, opovržení, strach, smutek, zlost, štěstí
	> 27	strach a štěstí
Muži	< 27	zlost, smutek, opovržení
	> 27	zlost

Tabulka 2.3: Emoce ukazující na problém se strukturou aplikace

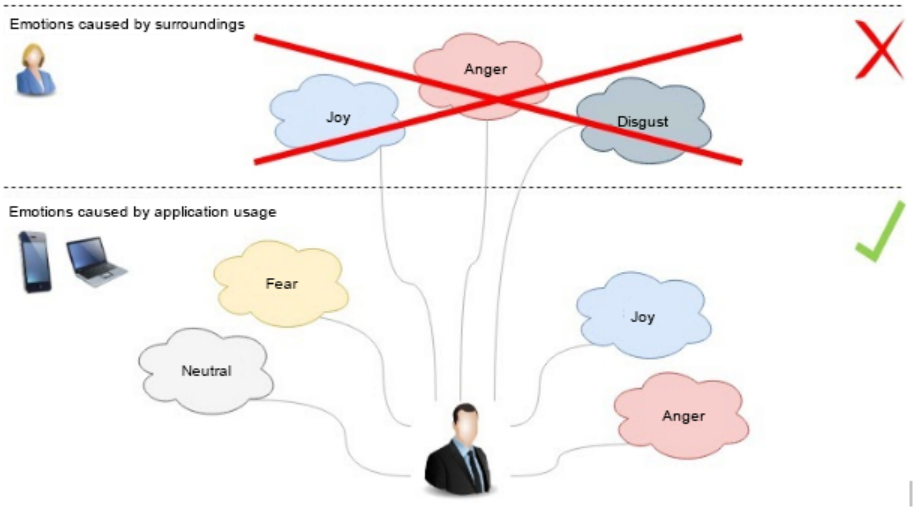
Z tabulky 2.3 je patrné, že obecně můžeme rozeznávat mnoho negativních emocí. Částečně dokonce vidíme, že i emoce štěstí může být považována někdy za negativní emoci.

Na chvíli se ještě pozastavíme nad typy emocí. Aktuálně rozeznáváme 7 typů emocí. Některé emoce pro nás nejsou zajímavé, jelikož nejsou zapříčiněné naší aplikací, ale vlivem okolí. Nyní se proto budeme zabývat otázkou jak poznat emoce, které jsou pro nás relevantní a které naopak zanedbat.

Poznamenejme, že každá emoce má 4 stádia [20]:

- **Rozpoznání** - Emoce je způsobena výzvou k akci. Předchozím stavem je stav bez emocí.
- **Přetrvání** - Subjekt projevuje stejnou emoci a i po výzvě k další akci.
- **Regulace** - Subjekt projevuje odlišnou emoci nebo se změní hladina dané emoce při výzvě k další akci.
- **Zaniknutí** - Při další akci subjekt neprojevuje žádné emoce.

Emoce způsobené okolním prostředím se dají snadno detekovat odložením aplikace, například uzamčením obrazovky u chytrého telefonu a nebo tím, že nějakou dobu nedetekujeme žádné emoce, jelikož uživatel nekouká na své zařízení. Emoce odečtené po návratu do aplikace můžeme považovat za emoce způsobené okolním prostředím a můžeme je tedy zanedbat po celou dobu jejich přetrvání. Přepnutím do jiné emoce začneme zase data snímat, analyzovat a adaptovat strukturu aplikace.



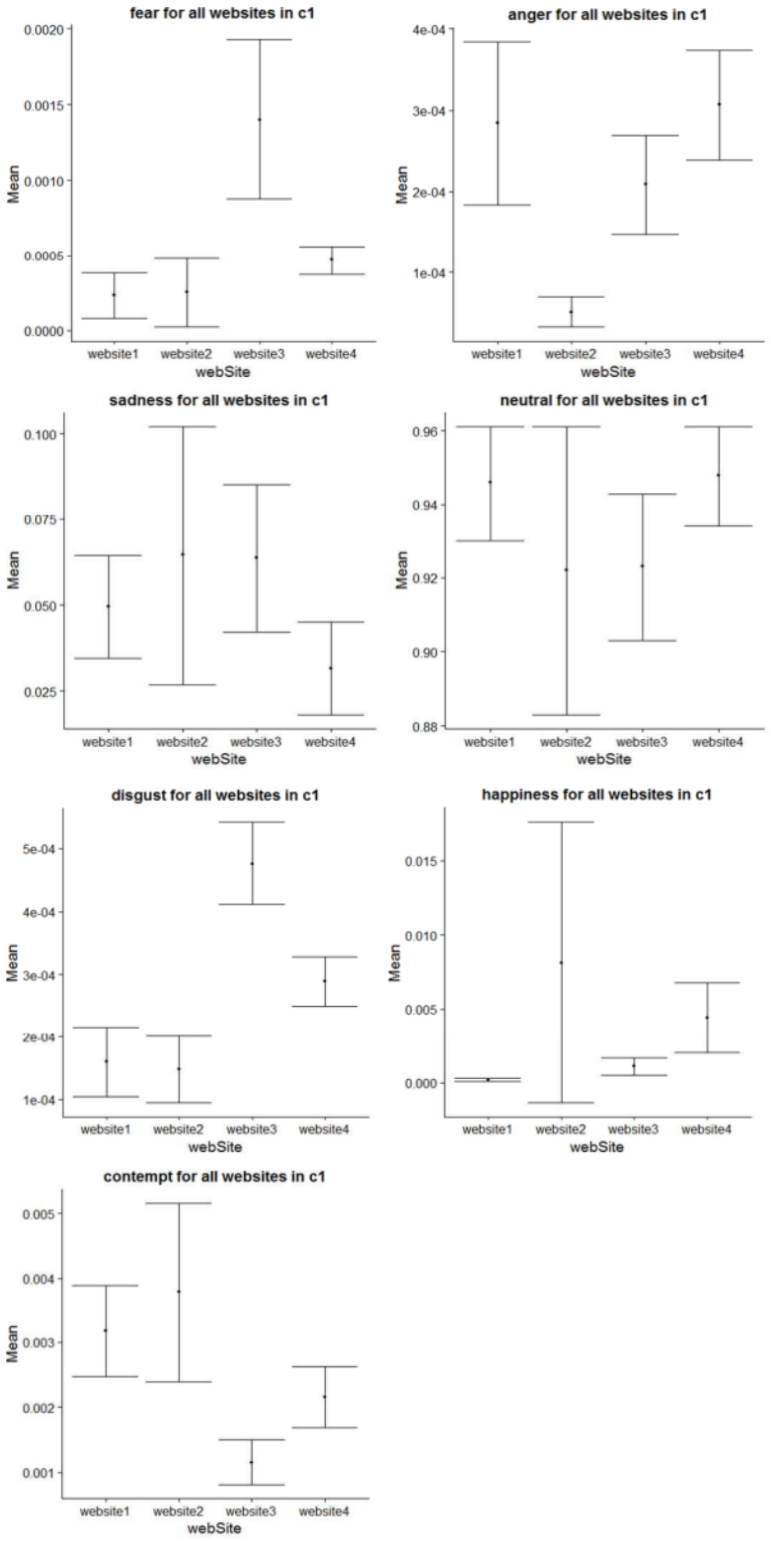
Obrázek 2.3: Filtrování uživatelských emocí

2.2.3 Problém: Poměr emocí určitého uzlu stromu určující adaptaci

Nyní, když se nám podařilo vymezit několik základních typů emocí, potřebujeme ještě prozkoumat hranici toho, při jak vysoké procentuální hodnotě špatných emocí je třeba strukturu aktualizovat.

Jinými slovy, potřebujeme pro každou emoci vymezit intervaly, při kterých nastane adaptace aplikace. Jedná se o vymezení pojmu špatná emoce. Studie [1] sleduje uživatelské emoce zapříčiněné jedním ze 2 problémů, které jsou uvedeny na obrázku 2.2.

Pro další výzkum si studie rozdělila testované subjekty do tří skupin na základě podobnosti vykázaných emocí pomocí PCA (Principal Component Analysis) a HCA (Hierarchical Cluster Analysis). Jednotlivé skupiny byly pak předmětem dalšího zkoumání.



Obrázek 2.4: Emoce zachycené pro jednotlivé stránky z obrázku 2 pro mladé ženy. Převzato z [1]

Z výzkumu, zobrazeného obrázkem 2.4 pro skupinu č. 1, je patrné, že například nízká použitelnost nebo nízká estetičnost může zapříčinit určité hladiny strachu, znechucení nebo opovržení pro mladé ženy.

Tato práce se však zabývá hlavně adaptací uživatelského rozhraní pro lepší použitelnost. Z toho důvodu se musíme zaměřit hlavně na výsledky webové stránky č. 4, která znázorňovala problém s použitelností. Tabulka 2.4 zobrazuje pozorování právě pro tuto stránku.

Kategorie	**Skupina**	**Emoce**	**Intervaly**
Ženy mladší 27 let	c1	znechucení	$2.5 \cdot 10^{-2}$ a $3.28 \cdot 10^{-2}$
	c2	zlost	$2.49 \cdot 10^{-1}$ a $4.47 \cdot 10^{-1}$
		štěstí	5.12 a 8.22
	c3	strach	4.07 a 5.20
		zlost	$2.96 \cdot 10^{-2}$ a $3.76 \cdot 10^{-2}$
		smutek	11.4 a 15
		opovržení	$5.09 \cdot 10^{-1}$ a $8.6 \cdot 10^{-1}$
Ženy starší 27 let	c2	strach	$2.46 \cdot 10^{-2}$ a $5.78 \cdot 10^{-2}$
		& štěstí	$1.70 \cdot 10^{-2}$ a $7.04 \cdot 10^{-2}$

Tabulka 2.4: Výsledky pozorovaných intervalů emocí pro ženy podle [1]

Na základě pozorování v tabulce 2.4 můžeme stanovit procentuální interval emocí pro znechucení na $2.5 \cdot 10^{-2}$ a $3.28 \cdot 10^{-2}$ pro mladé ženy. Výsledky stejné studie pro muže naleznete v tabulce 2.5.

Kategorie	**Skupina**	**Emoce**	**Intervaly**
Muži mladší 27 let	c1	zlost	$4.45 \cdot 10^{-2}$ a $6.17 \cdot 10^{-1}$
		smutek	12 a 15.5
	c2	opovržení	$1.31 \cdot 10^{-2}$ a $4.18 \cdot 10^{-1}$
	c3	smutek	40.6 a 40.6
Muži starší 27 let	-	zlost	-

Tabulka 2.5: Výsledky pozorovaných intervalů emocí pro muže podle [1]

Hodnoty z obou tabulek pak bereme jako výchozí intervaly pro adaptaci uživatelského rozhraní.

Kapitola 3

Návrh

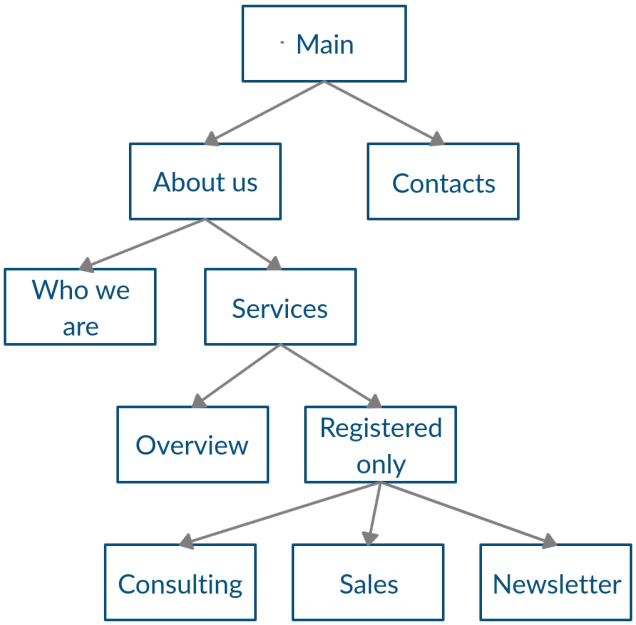
3.1 Obecné vymezení zachycených emocí navázaných na průchod strukturou aplikace

Pro účely této práce je potřeba, aby čtenář vnímal strukturu aplikace jako určitý strom. Po zanedbání možných cyklických prvků můžeme z každé aplikace vytvořit strom, který obsahuje kořen, uzly a také listy. [9]

- Kořen aplikace je obrazovka, která je zobrazena ihned po startu aplikace.
- Uzel je každý jeden prvek struktury, odpovídá tedy jedné obrazovce aplikace.
- List je koncový prvek struktury, který dále nepokračuje a neodkazuje na další obrazovky / uzly.

Jen pro představu můžeme zobrazit podobnou strukturu jako aplikaci, a to menu aplikace nebo webové stránky na obrázku 3.1 do stromu na obrázku 3.2.

Obrázek 3.1: Ukázka struktury menu



Obrázek 3.2: Příklad stromové struktury z obrázku 3.1

Z obrázků 3.1 a 3.2 můžeme vidět, že každá úroveň menu odpovídá dalšímu uzlu stromu. V našem případě ovšem nemůžeme používat strukturu menu, protože náš použitý framework [7] nedokáže zaznamenávat emoce na jednotlivé položky menu, ale pouze na celou obrazovku aplikace. Toto chování je logické, jelikož framework nesleduje zorničky a nemůže tak rozeznat, na které prvky na obrazovce upírá uživatel svou pozornost. Nemůžeme tak přesně určit, která položka menu danou emoci způsobila. Dále budeme používat pouze stromy z mobilní aplikace.

Naše navrhované řešení si bude muset uchovávat a ukládat minimálně dvě datové struktury:

- Uzel stromu - Zde bude třeba zachytit emoce, návštěvy a další informace.
- Informace o aplikaci - Jedná se o informace o uživatelově věku, pohlaví.

```
class Node {
    Node parent;
    List<Node> buttons;
    List<Emotions> emotions;
}

class AppInfo {
    int age;
    String gender;
}
```

Ukázka 3.1: Pseudokód návrhu entit pro uzly a informace pro aplikaci

Je nutno poznamenat, že třídy jsou pouze návrhem budoucího vývoje a s vysokou pravděpodobností se mohou měnit a následně mohou držet mnohem více informací, které budou pro implementaci potřebné.

3.1.1 Určení t-hranice

Pro následující odstavec si vymezíme následující pojem.

- **Špatná emoce** - emoce zaznamenána v souladu s tabulkami 2.4 a 2.5

Představme si situaci, kdy se uživatel proklikává stromem aby našel nějakou akci. Tuto akci nemůže dlouho najít a začne vykazovat špatné emoce. Akci nakonec najde a my máme v každém listu stromu zaznamenané informace o špatných emocích. Následně musíme odlišit, jak v tuto chvíli poznáme, jaký uzel stromu uživatel hledá, jaké prvky jsou zdrojem špatné emoce a které uživatel "potkal" jen po cestě.

Stejný případ může být ukliknutí na hlavní stránce. Tlačítko je špatně umístěno a uživatel se konstantně uklikne a prožívá špatné emoce. Jak následně rozpoznat, že emoce nejsou způsobené pouze výsledkem nějaké akce na dané straně? Takový uživatel, kterému se toto stane většinou opustí obrazovku velmi rychle. Pro toto odlišení nám pomůže stanovení t-hranice v jednotkách času. Pokud doba strávená uživatelem v jednom uzlu nepřekročí danou hranici, považujeme tento uzel za navštívený po cestě, nebo omylem.

Podle výsledků výzkumu [21] je průměrná doba reakce člověka na obsah 284ms. Vzhledem k tomu, že použité technologie mohou prodloužit dobu reakce o dobu odezvy aplikace, stanovíme t-hranici o něco vyšší než průměrnou dobu, tedy na hodnotu 300ms. Pro budoucí rozvoj frameworku bude dobré tuto položku zavést do jeho nastavení. Každý implementátor frameworku si pak může sám tuto dobu nastavit.

Náš model uzlu se tedy nyní rozšíří o dvě položky. První z nich bude průměrný čas strávený na stránce. Pokud bude stránka nepoužívaná a člověk se na ni často překlikne, pak můžeme soudit, že je třeba tuto položku přemístit. Další položkou, která bude pouze pomocná, je začátek návštěvy dané stránky. Z této položky lze pak vypočítat čas strávený na obrazovce. Výpočet času bude následující:

- newAverageTime = $\frac{(now - startVisit) + averageTimeSpent}{visits}$

Model je upraven v ukázce 3.2.

```
class Node {
    Node parent;
    List<Node> buttons;
    List<Emotions> emotions;
    int averageTimeSpent;
    DateTime startVisit;
}

class AppInfo {
    int age;
    String gender;
}
```

Ukázka 3.2: Pseudokód návrhu entit pro framework s přidanými informacemi na výpočet t-hranice

3.1.2 Druhy průchodů aplikací s navázanými emocemi

Pro následující kapitolu zavedeme slovníček rozšiřující původní pojmy: [19]

- Podstrom - je tvořen uzlem a všemi jeho potomky
- Cesta ve stromě - posloupnost navštívených uzlů

- Sousední uzel - uzel stromu, který se nachází v potomcích rodičovského uzlu
- Problematický uzel - uzel, v němž uživatel prožíval špatnou emoci
- Ukončení cesty nastane když:
  - uživatel odloží aplikaci
  - uživatel se vrátí do startovacího uzlu

Nyní se podíváme na několik různých druhů průchodu aplikací v návaznosti na emoce uživatele. V každém bodě pak rozebereme, na co poukazují a jaké může být jejich řešení. Nyní si zavedeme značení uzlů stromu tak, aby bylo názorně vidět, o co se jedná. V tabulce 3.1 je uvedena legenda značení průchodů strukturou aplikace.

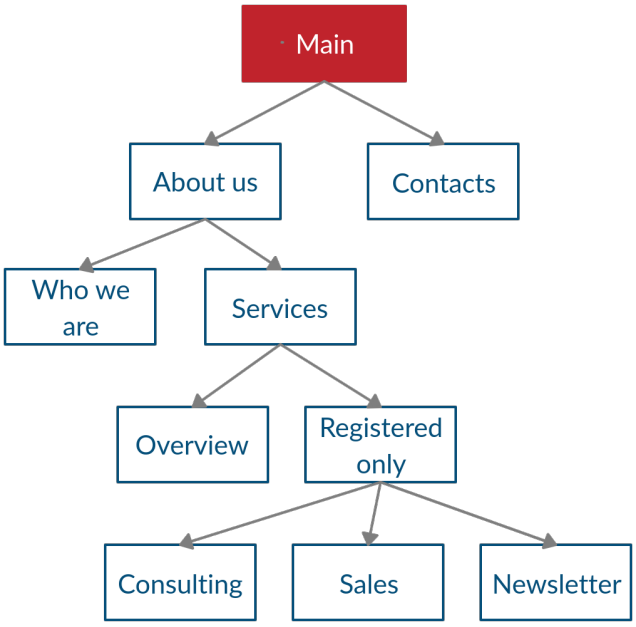
Main	Uzel je problematický
Contacts	Uzel není problematický
Newsletter	Změněný uzel
t < threshold	Doba strávená uživatelem v uzlu nepřekročila hodnotu t-hranice
t > threshold	Doba strávená uživatelem v uzlu překročila hodnotu t-hranice

Tabulka 3.1: Legenda značení v grafickém znázornění

Emoce způsobené okolním vlivem

Uživatel prožívá špatné emoce již v startovacím uzlu.

- **Strom emocí**



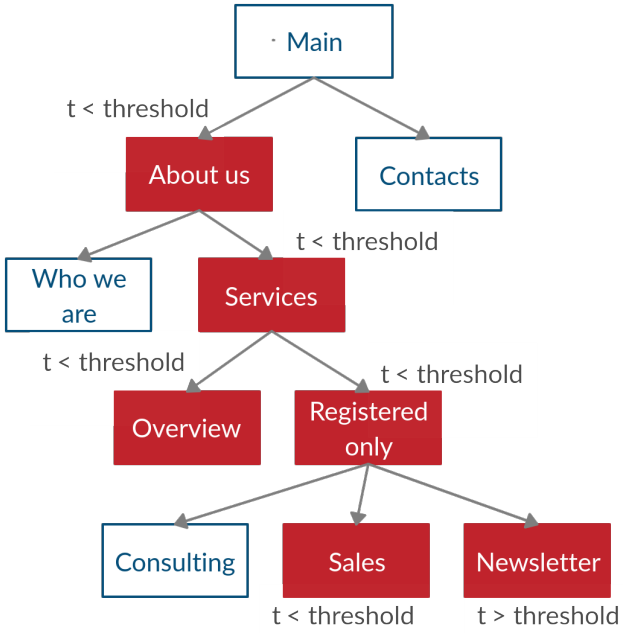
Obrázek 3.3: Strom emocí způsobený okolním vlivem

- **Průchod stromem** - Main
- **Detekovaný druh problému** - Emoce jsou způsobeny okolním vlivem.
- **Řešení** - Zanedbáváme emoce dokud se uživatel nepřepne do jiných uzlů.
- **Výsledek** - Zkoumáme pouze emoce způsobené špatnou strukturou aplikace. Vše zůstává stejné.

Špatně umístěné položky

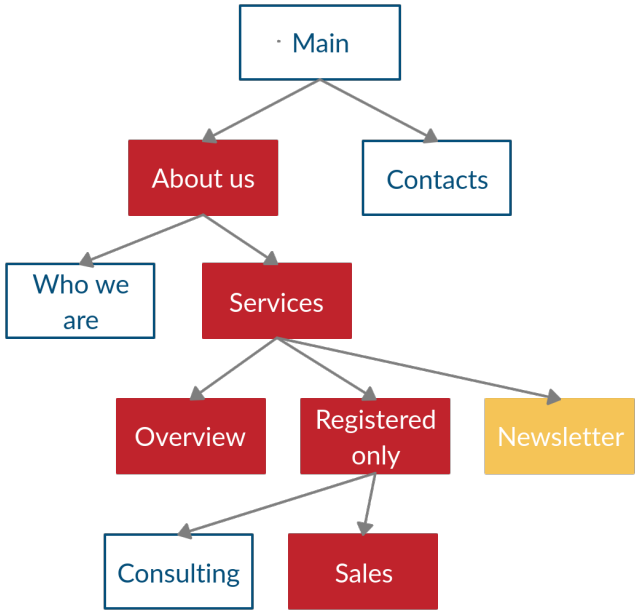
Uživatel prožívá špatnou emoci v uzlech jedné větve. V každém tomto uzlu je doba zde strávená nižší než stanovená t-hranice. Uzel, ve kterém doba strávená v uzlu překročí stanovenou hranici, považujeme za konec cesty.

- **Strom emocí**



Obrázek 3.4: Strom emocí pro špatné umístění položky

- **Průchod stromem** - Main - About us - Services - Overview - Services - Registered only - Sales - Registered only - Newsletter
- **Detekovaný druh problému** - Uživatel dlouho hledá položku "Newsletter", tudíž je umístěna ve stromě moc hluboko.
- **Řešení** - Přesunout položku "Newsletter" výš ve stromě do rodiče jeho rodiče.



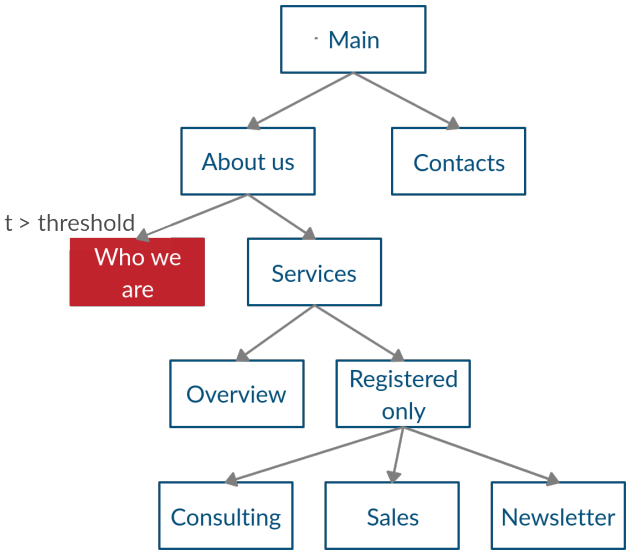
Obrázek 3.5: Strom emocí s novým umístěním položky na základě obrázku 3.4

- **Výsledek** - Při příštím průchodu stromu najde uživatel hledaný prvek rychleji, vzhledem k tomu, že na něj ve stromě narazí dřív.

Zavádějící vzhled položky nebo nečekané chování akce na položce

Uživatel prožívá špatnou emoci v listu stromu. Doba strávena v uzlu je vyšší než stanovená t-hranice. Uživatel v tomto listu cestu ukončí. I když se uživatel dostal k položce rychle, její obsah v uživateli vyvolal negativní emoce.

- **Strom emocí**



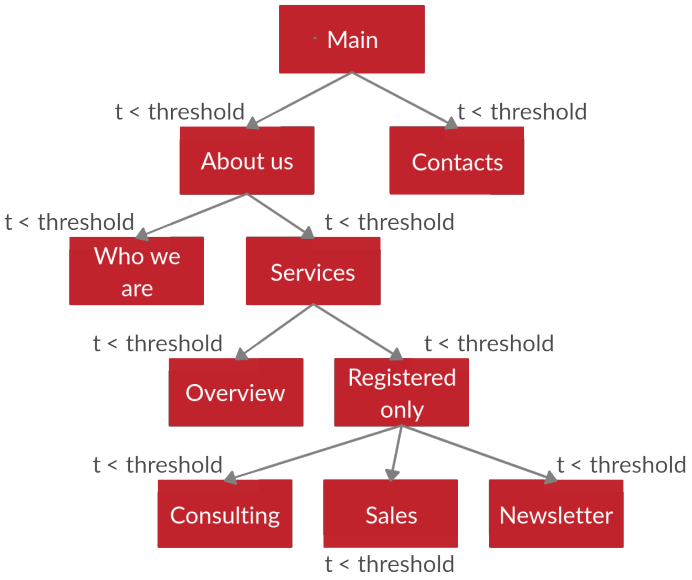
Obrázek 3.6: Strom emocí se zavádějícím vzhledem

- **Průchod stromem** - Main - About us - Who we are
- **Detekovaný druh problému** - Položka "Who we are" se zachovala nečekaně, tudíž má špatný vzhled nebo pojmenování, což vyvolalo v uživateli negativní emoce.
- **Řešení** - Upozornit vývojáře na problém a při dalším nasazení upravit vzhled položky nebo akci na dané položce.
- **Výsledek** - Položka bude upravena v následující aktualizaci.

Hledaná položka neexistuje

Uživatel prochází uzly stromu. V každém uzlu je doba zde strávená nižší než stanovená t-hranice. Cesta se ukončí odložením aplikace.

- **Strom emocí**



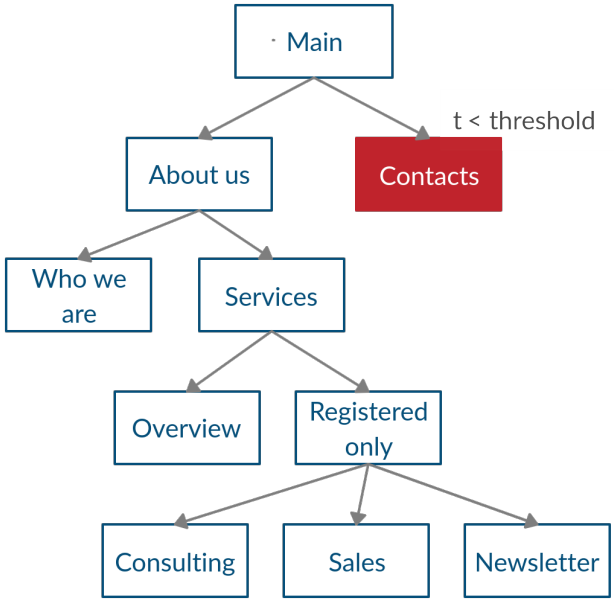
Obrázek 3.7: Strom emocí pro neexistující položku

- **Průchod stromem** - Všechny uzly
- **Detekovaný druh problému** - Hledaná položka neexistuje.
- **Řešení** - Při dalších přihlášení uživatele do aplikace mu nabídnout možnost zpětné vazby.
- **Výsledek** - Vývojář dodá chybějící položku pro další verzi aplikace.

Položka je umístěna nevhodně

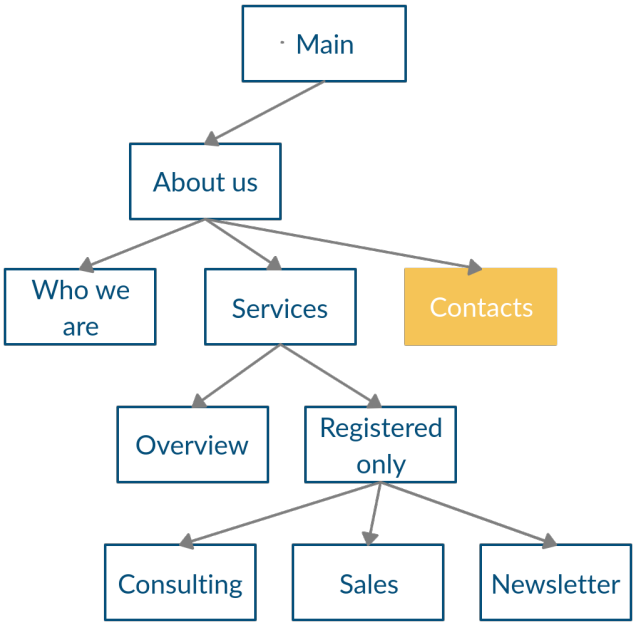
Uživatel prochází uzly stromu. V nějakém z uzlů hodnota t překročí stanovenou hranici threshold. Pravděpodobně se jedná pouze o častý překlik na méně využívanou položku.

- **Strom emocí**



Obrázek 3.8: Strom emocí pro nevhodné umístění položky

- **Průchod stromem** - Main - Contacts - Main - About us
- **Detekovaný druh problému** - Položka "Contacts" je umístěna nevhodně. Uživatel na ní ve většině případů klikne omylem a rychle reaguje tak, že se překlikne zpět.
- **Řešení** - Přesunout položku do některého jejího souseda.



Obrázek 3.9: Strom emocí se zanořenou položkou na základě obrázku 3.8

- **Výsledek** - Ukryjeme tuto položku do jednoho z jejich sousedů, aby nedocházelo k překliku.

3.2 Vytvoření adaptivní uživatelské struktury pro Android

Dalším bodem této práce je vytvoření takové struktury pro Android, která se dokáže za chodu měnit. Existuje podobná studie [22], která na základě prostředí, ve kterém se uživatel nachází, emocí a dalších okolností mění uživatelské rozhraní telefonu. Tato studie zpracuje data, analyzuje je a odešle na server. Na serveru se vše vyhodnotí, vybere se předpřipravené nastavení pro danou okolnost a server následně vytvoří novou verzi aplikace, připravenou pro sestavení a nahrání na zařízení. Tato funkčnost se zakládá na principu dynamického sestavení a nasazení aplikace (z angl. “dynamic deployment”) [23]. Naše práce nebude posílat data na žádný server a je potřeba adaptaci provádět při startu aplikace. Android bohužel zatím není pro tuto schopnost dostatečně vybaven, proto bude potřeba zajistit jiný způsob, díky kterému by to bylo možné.

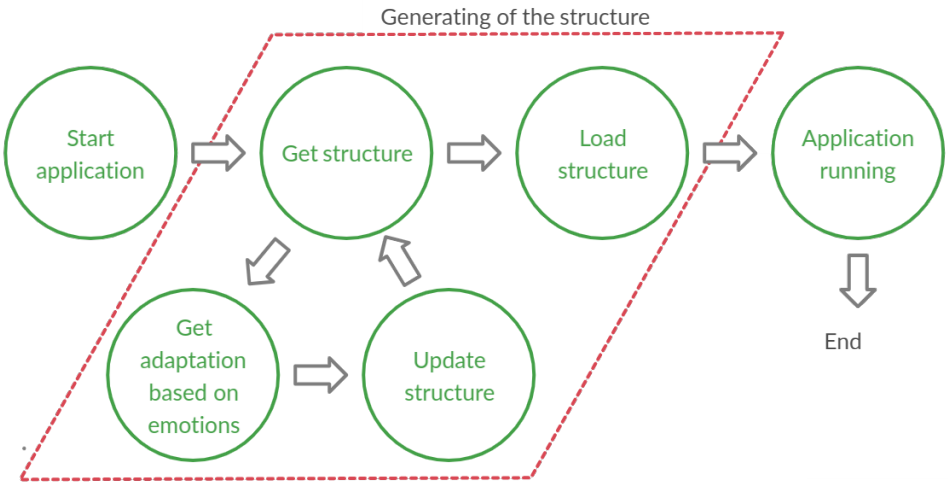
Nejprve uveďme, jak bude aplikace fungovat. Rozdělme její fungování na několik bodů:

A První zapnutí

B Fungování

C Každé další zapnutí

Nyní si rozebereme každou jednu z výše uvedených činností aplikace a její fungování.



Obrázek 3.10: Schéma načtení struktury

U bodu A je třeba inicializovat strukturu aplikace. Tedy zobrazit průchod přesně tak, jak ho uživatel vyžaduje v první verzi. Následuje bod B, který znázorňuje chod aplikace. Při chodu samotném je nutné vždy najít správný uzel ve stromu a správně ho zobrazit. Uzel stromu bude tedy potřebovat uchovávat jednotlivá tlačítka a svého rodiče. Struktura potřebuje zatím jen ke každému uzlu přiřadit všechna tlačítka na dané stránce.

Nejzajímavějším je ale bod C, ve kterém se odehrává změna struktury. Nyní již nemůžeme načíst pouze statická XML (typ datového souboru) s definovanou strukturou, ale musíme sáhnout do databáze a získat naši strukturu. Poté se spustí přepočet adaptace na základě emocí a v případě nové verze struktury se zajistí změna. Struktura se změní a uloží. Pro případ zachování historie a následných zpětných úprav si definujeme u struktury pomocný atribut, který bude držet verzi struktury. Takto se snadno dostaneme k předchozím verzím struktury.

```
class Structure {
    int version;
    Map<Node, List<Button>> path;
}
```

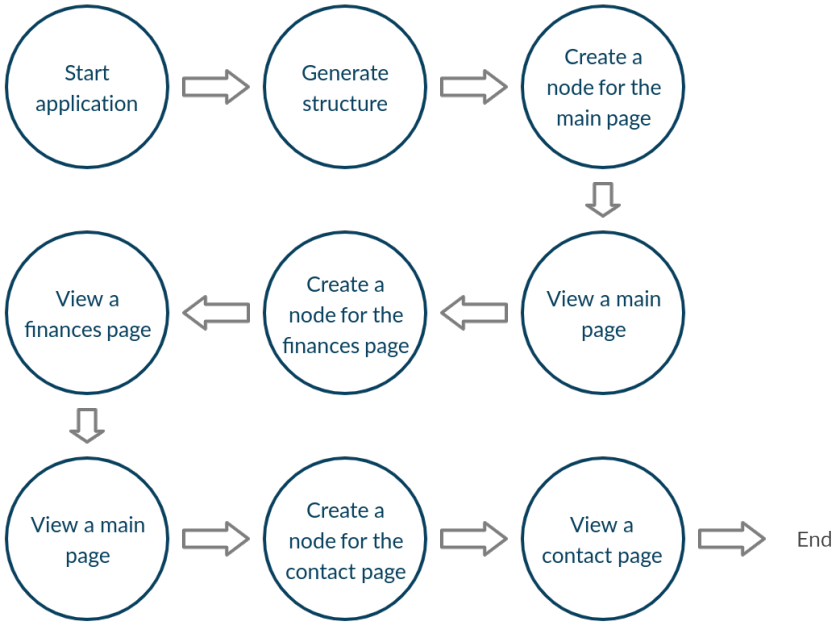
Ukázka 3.3: Struktura aplikace

Otázka ukládání struktury a uzlů je již vyřešena, ale ještě je potřeba najít způsob, jakým budeme strukturu načítat. Android má na načítání definovaných obrazovek nástroj zvaný LayoutInflater [24]. Ten dokáže na základě daného XML načíst celou stránku aplikace. Bohužel, tento nástroj zatím není uzpůsoben dynamické změně a adaptaci aplikace. Neumí načítat zdrojové soubory [25], tedy XML, z jiné lokace než z resources (repozitář v projektu implementace Android aplikace) a zároveň aplikace nemůže do tohoto souboru nic zapisovat. Proto je nutné se zamyslet nad tím, jak vygenerovat vždy každou stránku a kam ji uložit.

Protože potřebujeme vytvořit nové načítání jednotlivých obrazovek, musíme si nejprve vytvořit návrh fungování aplikace. Rozebereme si životní cyklus aplikace o třech stránkách:

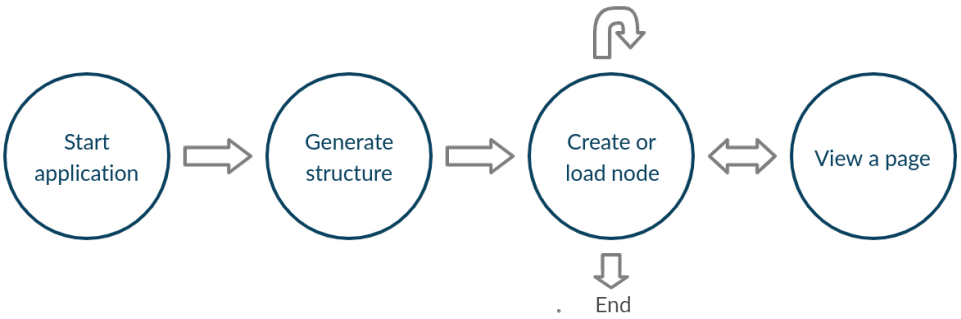
- hlavní stránka - obsahuje dvě tlačítka a odkazuje na další dvě stránky
- finance - list stromu, obsahuje cílovou funkčnost
- kontakt - list stromu, obsahuje cílovou funkčnost

Naším cílem bude zprostředkovat následující postup: zapnutí aplikace - main page - finances page - main page - contacts page - konec aplikace



Obrázek 3.11: Schéma průchodu scénářem

Start aplikace a vygenerování struktury z obrázku 3.11 je zanesen v obrázku 3.10 (červeně vyznačená oblast). Zde tedy bereme celý cyklus generování struktury jako jednu část. Pokud jde o vytvoření uzlu pro hlavní stránku, tak zde se jedná o inicializaci emocí, uložení rodiče a tlačítek do databáze. Načtení stránky je právě ono načtení XML z nějakého zdroje. Dále z nákresu vidíme stále se opakující uzly, takže to můžeme obecně rozkreslit snadněji.



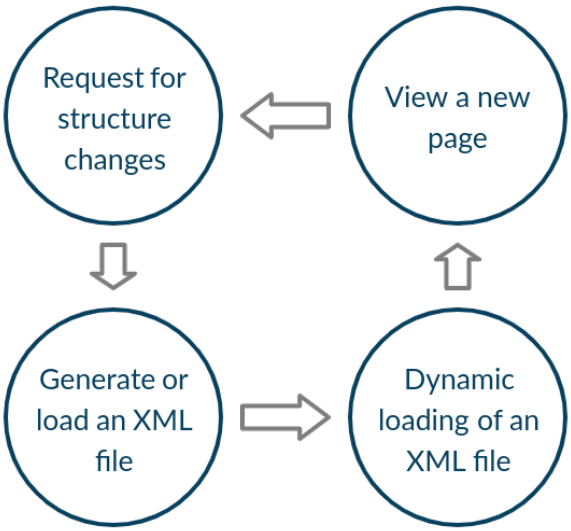
Obrázek 3.12: Obecný cyklus adaptivní aplikace

Zatímco vytvoření nebo načtení uzlu z databáze je jednoduché, načtení stránky je třeba dodělat. Nutno podotknout, že k načtení stránky v našem případě bude docházet i při stisknutí tlačítka zpět. Některé uzly budou zanikat a naopak budou vznikat nové, takže není možné vytvořit všechny možnosti.

Bude třeba vše generovat dynamicky.

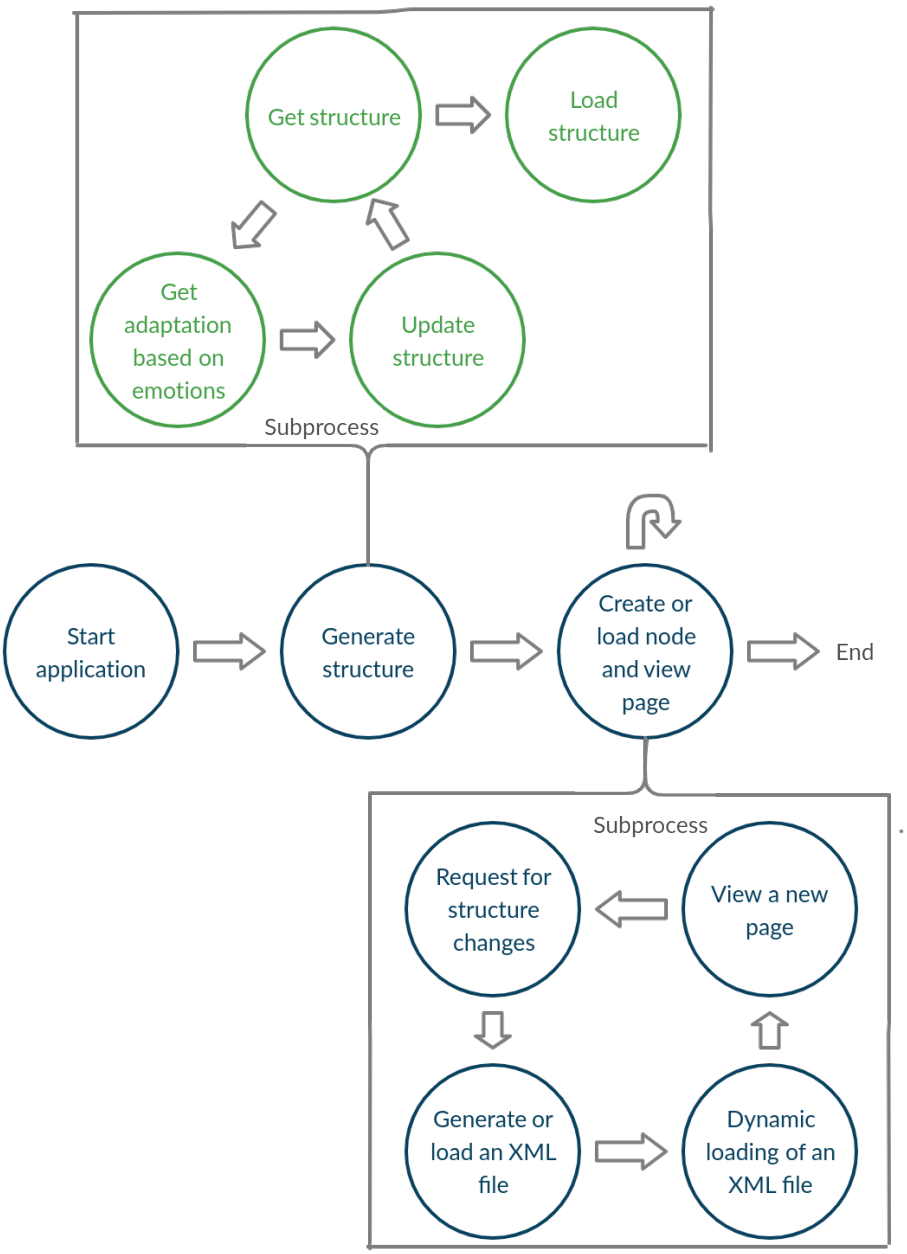
Nejlepší možností se zdá být ukládání XML a načítání při změně stránky. Pro tento účel byla použita implementace [26] dynamického inflateru (nástroj pro převedení XML souborů do objektů), který dokáže načítat XML mimo složku resources. Inflater dostane cestu k souboru a načte novou stránku. Toto je přesně to chování, které potřebujeme.

K zajištění vytvoření předlohy XML pro stránku potřebujeme vytvořit generátor XML. Tento generátor uloží XML se všemi stránkami do stejného úložiště, kde ho dynamický inflater bude hledat. Prozatím bude generátor generovat pouze tlačítka, ale bude snadno rozšiřitelný i o jiné pohledy. Generátor připraví stejné XML, jako kdyby ho načítal běžný ne-dynamický inflater. Dokázali jsme nakonec spojit dynamické generování stránek a jeho dynamické načítání. Velkou výhodou tohoto řešení je to, že do budoucna může být snadno modifikováno pro více typů pohledů a zároveň uzpůsobeno na dynamickou změnu přímo za běhu aplikace (ne jen při každém zapnutí).



Obrázek 3.13: Životní cyklus stránky

Celý proces načítání struktury a stránek je přehledně zanesen do obrázku č. 18. Jedná se o provázání obrázků 3.10, 3.11, 3.12 a 3.13.



Obrázek 3.14: Celý proces generování struktury a načítání stránek

Kapitola 4

Implementace

Implementace frameworku je vytvořena v objektově orientovaném jazyce Java na JDK 1.8 a vyvinuta v prostředí Android Studio. Framework je sestavován pomocí Gradle [27] a jako předlohu využívá primárně implementaci frameworku [7] na Adaptaci UI založenou na emocích uživatele. Použitý framework pracuje s AffDex SDK ve verzi 3.2 a je tak zprostředkovatelem zachycení emocí uživatele. V následujících bodech si rozebereme základní stavební strukturu frameworku.

4.1 Struktura projektu

Build gradle

Build gradle je klíčový sestavovací soubor [27] pro framework. V tomto souboru lze přidávat jednotlivé závislosti potřebné pro náš projekt. Mimo jiné zde najdete klíčovou knihovnu pro zachycení emocí.

```
1 dependencies {
2   implementation 'com.affectiva.android:affdexsdk:3.2'
3 }
```

Ukázka 4.1: Přidání závislosti na AffDex [14] frameworku ve verzi 3.2

config.properties

Tento soubor je potřeba zejména pro vývojářské nastavení frameworku. Obsahuje několik konfigurovatelných položek. Jednotlivé položky nastavení jsou následně načteny ve třídě PropertyUtil.

- Minimal visits for change - minimální počet návštěv uzlu, po kterém se má kontrolovat změna struktury
- Main page name - jméno hlavní stránky
- Detector rate - rychlost detektoru
- Directory of pages - jméno složky, kam se ukládají vygenerované soubory XML
- Change after each version update - při každé změně se vynulují předchozí hodnoty emocí v daném uzlu
- Threshold - časové omezení strávení v jednom uzlu pro změnu

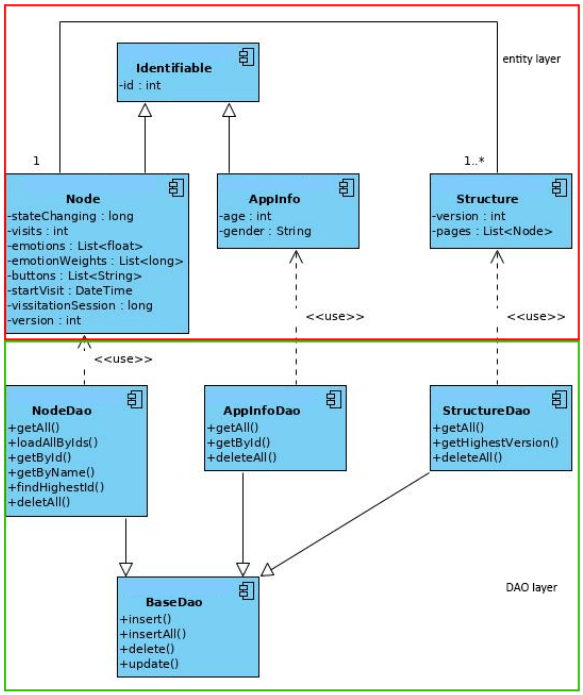
```
minimal_visits_for_change=50
main_page_name=mainPage
detector_rate=10
directory_of_pages=adaptive-structure-layout
change_after_each_version_update=true
threshold=3
```

Ukázka 4.2: Nastavení vlastností

Databázové operace

Jeden z důvodů, proč nebylo v této práci pouze doplněno řešení frameworku [7], je databáze. Původní řešení využívalo zastaralou verzi SQLite. Pro jednodušší orientaci a lepší použitelnost byla databáze migrována na Room Database, což je obdobný framework jako JPA využívající právě SQLite pro přístup do databáze. Výhodou je přehlednost uložených entit v datových objektech a předpřipravené databázové operace. Room využívá anotaci u entit a DAO (z angl. "Data access object") třídy pro CRUD (operace pro vytvoření, čtení, aktualizaci a smazání) operace nad entitami. Díky tomu vznikly v projektu tři directory: database, DAO a entity. Adresář "database"je využíván hlavně pro načtení databáze. Nakonec bylo také potřeba vytvořit několik konvertorů, které převádí do databáze nekonvertovatelné datové typy.

V této implementaci bylo využito návrhu DAO, ale nabízela se i možnost Repository (návrhový vzor pro práci s databází). Tyto dva návrhové vzory [28] jsou velmi podobné a oba poskytují správu operací CRUD nad entitami. Je mezi nimi jen několik málo rozdílů, ten největší je, že DAO je blíže datovému modelu, zatímco Repository je blíže doménovému modelu (z angl. "Domain model"). Datový model reprezentuje data tak, jak jsou ukládány do databáze, naopak doménový model představuje objekty v objektově orientované programování [29].



Obrázek 4.1: Domain model diagram

Původní řešení

Na základě úpravy původního frameworku [7] vzniklo v projektu několik adresářů - states a utils, které obsahují pomocné třídy pro načítání emocí.

Inflater

Jak již bylo zmíněno, řešení používá dynamický inflater [26], který načítá dynamicky generované XML soubory. Inflater byl upraven na základě potřeb

vyvíjeného frameworku o novou cestu k adresáři se složkami. Dále byla přidána možnost generování SurfaceView, které nakonec nebylo použito v generátoru, ale mimo něj. Na ukázce 4.3 je zachyceno použití dynamického inflatoru.

```
View view = DynamicLayoutInflator.inflateName(context,
    currentViewName);
```

Ukázka 4.3: Načtení nového view

Creation

Postupně se dostáváme ke klíčovým částem této práce. Jednou z nich je právě část vytvoření, která dělá první inicializaci struktury testovací aplikace a následně načítá strukturu z databáze.

```
List<String> incomeANSIPOButtons = new LinkedList<>();
incomeANSIPOButtons.add("Income");
incomeANSIPOButtons.add("SIPO");
pairs.put("Income and SIPO", incomeANSIPOButtons);
List<String> incomeButtons = new LinkedList<>();
incomeButtons.add("New Income");
pairs.put("Income", incomeButtons);
List<String> SIPOButtons = new LinkedList<>();
SIPOButtons.add("New SIPO");
pairs.put("SIPO", SIPOButtons);
pairs.put("New Income", new LinkedList<>());
pairs.put("New SIPO", new LinkedList<>());
```

Ukázka 4.4: První inicializace struktury aplikace

Obrázek 4.2: Inicializovaná struktura z ukázky 4.4

Adaptation

Hlavní adresář celé práce. Obsahuje přípravu struktury, tedy vkládání uzlů do databáze, které je upravené a převzaté z frameworku A. Lunové [7] s migrací na Room [30] databázi. Dále se zde nachází třída, která umožňuje adaptaci struktury při startu testovací aplikace a následně také třída, která provádí přesun z jedné stránky na druhou.

XML generator

V této části se nachází generátor, který vytváří XML soubory z načtené struktury.

4.2 Klíčové metody frameworku

V této kapitole si projdeme klíčové metody posledních tří zmíněných částí projektu. Jedná se o důležitou funkčnost celého frameworku a jeho chování.

4.2.1 Úprava struktury

Úprava struktury je prováděna vždy při startu testovací aplikace. Tedy lépe řečeno je vyvolána kontrola, při které se analyzují nasbírané emoce jednotlivých uzlů a na základě nich je následně načtena buď nová struktura nebo ta stávající. Prohledávání je realizováno pomocí DFS (z angl. "depth-first search"- prohledávání do hloubky [31]), jelikož každý uživatel cílí na koncovou funkčnost aplikace, tedy list struktury a ne jen její uzel. Metoda *depthFirstChange(pair)* je volána rekurzivně, čímž docílíme právě zmíněného DFS prohledávání.

```
1  void changeStructure() throws IOException {
2      if (structure == null) {
3          return;
4      }
5      String mainPageName = PropertyUtil.getMainPageName(context);
6      List<String> mainPage = structure.getPages().get(
       mainPageName);
7      if (mainPage == null) {
8          throw new IllegalArgumentException("Main page should not
       be null!");
9      }
10     Pair<String, List<String>> pair = new Pair<>(mainPageName,
       mainPage);
11     databaseCopy = db.nodeDao().getAll();
12     depthFirstChange(pair);
13     exportDatabase();
14     if (changeVersion) {
15         initAllNodes();
16         saveToDatabase();
17         updateNewStructureVersion();
18         db.structureDao().insert(newStructure);
```

```
19      }
20  }
```

Ukázka 4.5: Ukázka funkce *changeStructure();*

Jak je vidět z ukázky 4.5, pokud je zjištěna nová verze struktury, do databáze je uložena pomocná struktura, která je vytvářena při kontrole. Zde je na místě strukturu neupravovat rovnou, protože bychom si ji akorát měnili pod rukama, takže by mohly nastat nekonzistentní stavy.

Velmi důležitou metodou je také *shouldMove(Node node)*, která zkontroluje, zda je potřeba uzel přesunout. Kam, to už záleží na kontextu testovací aplikace a aktuálním stromu průchodu. Tato metoda využívá omezení na základě věku, pohlaví a emočních intervalů navržených v části 2.2.

```
1  private boolean shouldMove(Node node) {
2     long visits = node.getVisits();
3     int version = node.getVersion();
4     int numberOfVisits = PropertyUtil.getNumberOfVisits(context);
5     int changeValue = version * numberOfVisits;
6     String mainPageName = PropertyUtil.getMainPageName(context);
7     if (visits >= changeValue && !node.getName().equals(
      mainPageName)) {
8         updateNodeToNextVersion(node);
9         return getChangeOnGenderAndAge(node);
10    }
11    return false;
12 }
```

Ukázka 4.6: Metoda *shouldMove()* rozhodující o přesunutí uzlu

4.2.2 Přesun mezi stránkami

Jak již bylo řečeno v návrhové části, tento framework nedokáže využít jednoduchý přesun mezi stránkami, který je součástí většiny Android aplikací. Důvodem je fakt, že nedokážeme předem říct, jak se bude struktura v čase modifikovat. Pro přechod z jedné stránky na druhou vznikla v projektu třída zajišťující změnu XML, která nastavuje tlačítkům načteným z XML onClickListener. Ten rekurzivně volá opět stejnou metodu. Tato metoda se nazývá *move()* a je zachycena na ukázce 4.7.

```
1  private View move(Activity context, List<String> buttons, String
       className, String viewName) {
2     db = DatabaseInit.getASDatabase(context);
3     if (currentViewName != null) {
4        List<Node> parents = db.nodeDao().getByName(
      currentViewName);
```

```
5          if (parents.size() != 1) {
6             throw new IllegalArgumentException("Have to be exactly
       one result!");
7          }
8          setTimeSession(parents.get(0));
9       }
10      incrementId();
11      if (viewName != null) {
12         currentViewName = viewName;
13      } else {
14         currentViewName = PropertyUtil.getMainPageName(context);
15      }
16      List<Node> byIds = db.nodeDao().getByName(currentViewName);
17      Node byId;
18      if (byIds.size() > 1) {
19         throw new IllegalArgumentException("Must be just one Node
       by name!");
20      } else if (byIds.size() == 1) {
21         byId = byIds.get(0);
22      } else {
23         byId = null;
24      }
25      View view;
26      try {
27         XMLMaker.generateXML(currentViewName, className, buttons,
       context);
28          view = DynamicLayoutInflator.inflateName(context,
       currentViewName);
29      } catch (IOException e) {
30          throw new IllegalArgumentException(e);
31      }
32      setId(view, byId);
33      AdaptationPrepare.getAdaptationMaker().
       createApplicationStructure(view, currentViewName, buttons);
34      nextView = view;
35      MoveMaker.getInstance().setOnClickListeners(view, context);
36      return view;
37  }
```

Ukázka 4.7: Metoda *move()* na přesun mezi obrazovkami

Metoda se stará o správné přiřazení ID obrazovce, o načtení správného pohledu a také o zapsání předchozího pohledu pro návrat zpět. Výsledná aplikace rozšiřující tento framework je založena na principu jedné aktivity (z angl. "Single activity"), což znamená, že se obsah vykresluje tak, jak v něm uživatel prochází.

4.2.3 Úpravy původního frameworku

Tato práce se částečně zabývá rozšířením frameworku [7] na adaptaci uživatelského rozhraní na základě emocí. Původní práce ukládala pomocí SQLite ke každému uzlu emoce a zároveň tyto emoce využívala k úpravě barev jednotlivých komponent. Jelikož bylo pro tuto práci nutné migrovat na Room databázi a zároveň doplnit ukládané záznamy do databáze, bylo rozhodnuto odstranit také původní barevnou adaptaci jednotlivých komponent. Velkou výhodou tohoto rozhodnutí je zrychlení frameworku a lepší orientace v kódu při implementaci.

Základním stavebním kamenem této funkčnosti je metoda *analyzeEmotions()*, která detekuje aktuální stav emocí a ukládá jej. Tato metoda je vyvolána vždy detektorem [14], který sleduje změny emocí.

Ukládání uzlu při návštěvě další stránky, které se velmi obměnilo, je prováděno metodou *createNode()*. Tato metoda buď aktualizuje počet návštěv, či začátek návštěvy, nebo vytvoří nový uzel, pokud ještě není v databázi.

```
1  Node createNode(int uid, String name, int parent, List<String>
       buttons, Context context) {
2      ASDatabase db = DatabaseInit.getASDatabase(context);
3      Node node = db.nodeDao().getById(uid);
4      if (node == null) {
5          node = new Node();
6          node.setId(uid);
7          node.setName(name);
8          node.setParent(parent);
9          node.setButtons(buttons);
10         node.setVersion(1);
11         node.setStartVisit(LocalDateTime.now());
12         db.nodeDao().insert(node);
13     } else {
14         long visits = node.getVisits() + 1;
15         node.setVisits(visits);
16         node.setStartVisit(LocalDateTime.now());
17         db.nodeDao().update(node);
18     }
19     return node;
20 }
```

Ukázka 4.8: Vytvoření uzlu

4.2.4 Využití frameworku v demo aplikaci

Implementace tohoto frameworku byla prováděna účelně tak, aby byl snadno a jednoduše rozšiřitelný a použitelný. Tento framework je například výhodný pro analýzu již hotových struktur za účelem zlepšení.

Pokud by vývojář chtěl framework použít, pak musí splnit několik základních kroků.

- Aplikace bude potřebovat přístup k fotoaparátu a k souborovému systému na telefonu. Pokud chce vývojář uživatelům zjednodušit tento úkol, pak je dobré se zeptat na povolení k výše zmíněným zdrojům. K tomu se dá použít následující model. Do třídy onCreate() v naší jediné aktivitě přidáme následující řádky kódu. Dále přidáme také pomocné funkce.

```
protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     checkPermission(
             new String[]{Manifest.permission.
    WRITE_EXTERNAL_STORAGE, Manifest.permission.CAMERA},
             PERMISSION_CODE);
     waitAnswered();
}


public void checkPermission(String[] permissions, int
    requestCode) {
     List<String> permissionsToGet = new LinkedList<>();
     for (String permission : permissions) {
         if (ContextCompat.checkSelfPermission(
                 MainActivity.this,
                 permission)
                 == PackageManager.PERMISSION_DENIED) {
             permissionsToGet.add(permission);
         }
    }
    if (permissionsToGet.size() != 0) {
         ActivityCompat
                 .requestPermissions(
                         MainActivity.this,
                         permissionsToGet.toArray(new String[
    permissionsToGet.size()]),
                         requestCode);
    }

}

public void waitAnswered() {
    boolean isAnswered = false;
    while (!isAnswered) {
```

```
33          if (ActivityCompat.checkSelfPermission(getApplication(),
    Manifest.permission.WRITE_EXTERNAL_STORAGE)
            == PackageManager.PERMISSION_GRANTED &&
34                      ActivityCompat.checkSelfPermission(
    getApplication(), Manifest.permission.CAMERA) ==
    PackageManager.PERMISSION_GRANTED) {
35              isAnswered = true;
36          }
37      }
38  }
```

Ukázka 4.9: Povolení přístupu k úložišti a fotoaparátu

- Na ukázce 4.9 je uvedena možná implementace, která byla použita v testovací demo aplikaci. Existuje mnoho jiných možností, jak získat povolení k potřebným zdrojům. Důvodem, proč je potřeba explicitně od uživatele vyžádat povolení k některým funkcím, je změna politiky přístupů [32] ve verzi Android 6.0. Přístupy k funkcím jsou rozděleny do dvou skupin - normální a nebezpečné. Kamera a úložiště se nacházejí ve skupině nebezpečných.
- Využít v aktivitě další potřebné metody na adaptaci, ukládání a přesouvání mezi stránkami. Metody onCreate, onResume a onBackPressed vybavíme následujícím způsobem.

```
1  protected void onCreate(Bundle savedInstanceState) {
2      super.onCreate(savedInstanceState);
3      surfaceView = new SurfaceView(this);
4      surfaceView.setLayoutParams(new ViewGroup.LayoutParams(1, 1)
       );
5      setContentView(R.layout.activity_main);
6      adaptationPrepare = AdaptationPrepare.getAdaptationMaker();
7      MoveMaker.makeMove(this, surfaceView);
8  }
9
10 protected void onResume() {
11     super.onResume();
12     adaptationPrepare.adapt(this, surfaceView, MoveMaker.
       getInstance().currentView, -1, MoveMaker.getInstance().
       currentViewName);
13 }
14
15
16 public void onBackPressed() {
17     MoveMaker.getInstance().setBackClickListeners(this);
18 }
```

Ukázka 4.10: Základní nastavení v hlavní aktivitě pro použití frameworku

- Nejprve je třeba vytvořit a frameworku předat SurfaceView, pomocí kterého jsou sledovány emoce uživatele. Tento požadavek přetrvává již od

původního frameworku [7]. Následně přibylo několik dalších funkcí pro fungování adaptace struktury a přesunu mezi stránkami. Zde můžeme zmínit životní cyklus Android aplikace. Aktivita je vytvořena metodou onCreate a metoda onResume zajišťuje hladký návrat k aktivitě. Z tohoto důvodu nejprve nainicializujeme v metodě onCreate instanci AdaptationPrepare, zobrazíme hlavní stránku a následně v metodě onResume provádíme přesun na další stránky.

Kapitola 5

Testování

Pro testování frameworku byly vytvořeny 3 testovací aplikace se strukturou mobilní aplikace Air Bank. Výběr této struktury je odůvodněn hlavně její hloubkou (> 5 pater). Každá z aplikací je pak přiřazena pro jinou skupinu uživatelů. K testování je k dispozici celkem 14 subjektů rozdělených do tří skupin.

- Skupina 1 - Aplikace se adaptuje na základě emocí a používá vytvořený framework. Pro tuto skupinu je nastavena kontrola změny struktury po třech návštěvách obrazovky. Skupina má 5 členů.
- Skupina 2 - Aplikace je statická, nemá žádnou adaptaci. Tato skupina dostala v půlce testování stejnou aplikaci s upravenou strukturou na základě analýzy emocí. Skupina má 5 členů.
- Skupina 3 - Aplikace je statická po celou dobu testování. Skupina má 4 členy.

Studie [33] uvádí, že pro relevantní výsledky testování je potřeba 5 uživatelů. Jelikož chceme porovnávat výsledky 3 různých typů adaptace, pak potřebujeme uživatelů více. Pro pozorování prožívaných emocí a získání dat o uživateli byly testovací aplikace vybaveny reportem, který tyto informace zahrnuje. Tento report se vytváří vždy při startu aplikace a uživatelé měli za úkol každý den tento report posílat. Díky tomu byla možnost nahlédnou do uživatelských emocí u distančního testování. Pro zajímavost v tabulce 5.1 můžete vidět informace o uživatelích.

Skupina	Průměrný věk mužů	Průměrný věk žen	Průměrný věk celkem
Skupina 1	24.6	23	24
Skupina 2	24	23	23.8
Skupina 3	22.5	-	22.5

Tabulka 5.1: Informace o testovaných uživatelích

5.1 Průběh testování

Každý z testovaných uživatelů nejprve obdržel návod na přípravu aplikace, její testování včetně scénářů a instrukce k odesílání dat. Pro testování bylo vytvořeno 5 scénářů, které znázorňují hledání uzlů v dané struktuře. Cílem scénářů je co nejpřesněji replikovat situaci, kdy uživatel hledá určitou funkčnost v aplikaci a prožívá při tom různé emoce.

- Scénář 1 - Uživatel si vytvoří novou šablonu. Hledá tedy stránku "Vytvořit novou šablonu". Strom má při inicializaci hloubku 4. Scénář byl vybrán primárně pro jeho hlubší zanoření.
- Scénář 2 - Uživatel prozkoumá možnosti E-Shopu. V tomto scénáři je hledána stránka "E-Shop". Strom má při inicializaci hloubku 3. Scénář byl vybrán díky jinému umístění, než předchozí scénář a hlavně protože E-Shopy jsou obecně často vyhledávaná položka.
- Scénář 3 - Uživatel si vytvoří nové pravidelné spoření. Je proto vyhledávána stránka "Nové pravidelné spoření". Strom má při inicializaci hloubku 5. Scénář byl vybrán na základě jeho složitosti a zanoření.
- Scénář 4 - Uživatel prozkoumá možnosti hypoték. Cílem je stránka "Hypotéky" a strom má při inicializaci hloubku 3. Scénář reprezentuje jednoduchý průchod aplikací.
- Scénář 5 - Uživatel si vytvoří nové SIPO. Tento scénář hledá stránku "Nové SIPO" a hloubka stromu má při inicializaci hloubku 5. Scénář reprezentuje jasně popsanou cestu ke koncové položce.

Každý uživatel prováděl scénáře opakovaně v časovém rozmezí 12 hodin po dobu 5 dní. První testování je prováděno večer a poslední také večer. Máme tedy celkem 9 měření.

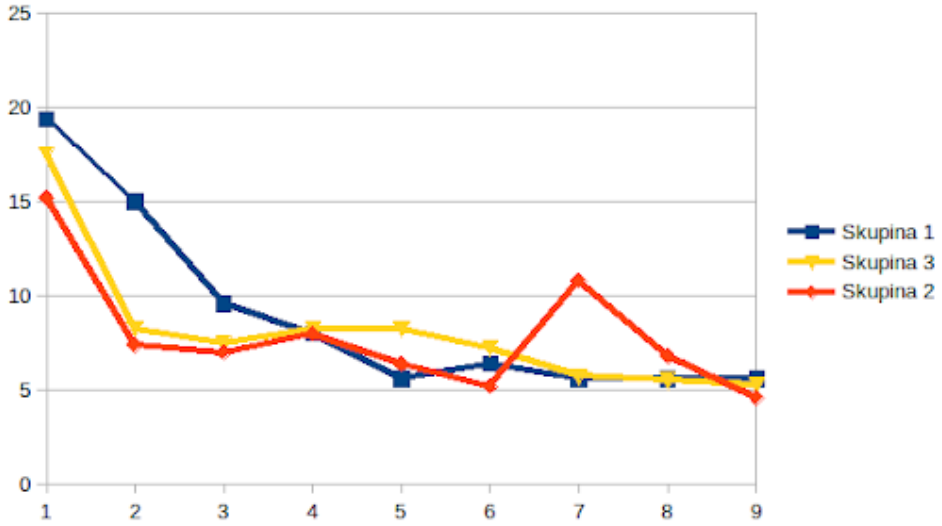
5.2 Výsledky testování

Již v první den testování byla odhalena chyba, kdy se struktura automaticky adaptovala špatně pro jednoho uživatele. Touto adaptací je myšleno zanoření uzlu do nejbližšího souseda. Bohužel, předpokladem pro zanoření bylo, aby uzel měl za rodiče hlavní stránku a zároveň, aby nebyl překročen threshold. Tento předpoklad byl vyvrácen na příkladu stránky menu, která má za rodiče právě hlavní stránku. Menu uživatel prošel vždy velmi rychle a tak byly oba požadavky splněny. Navíc při průchodu menu vyjadřoval několik emocí z předchozích uzlů. Menu se následně zanořilo do nejbližšího souseda a uživatel ho dlouho nemohl najít. Navíc tím utrpěly i ostatní scénáře, takže výsledky prvního dne byly anulovány.

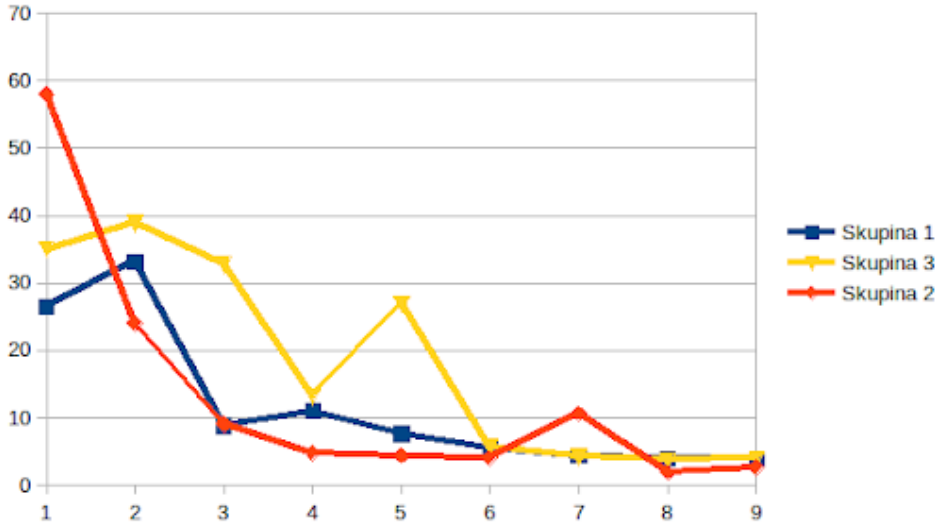
Nápravou této situace byla změna pravidel, kdy uzel již nemusí být pouze potomkem hlavní stránky, ale zároveň nesmí mít žádné potomky. Pravidlo na threshold bylo zachováno a testování i s opravenou demo aplikací bylo obnoveno.

- Při testování první skupiny nastalo hned několik adaptací na základě emocí. Pro každého uživatele to ovšem nastalo v jiný okamžik.
- Pro druhou skupiny byly dvě vlny adaptace provedené na základě reportů v 5. měření a v 7. měření.

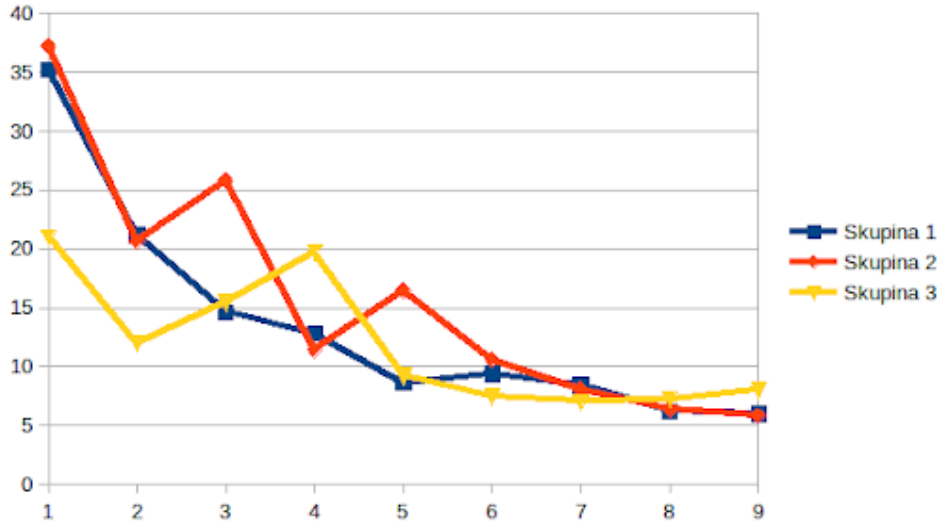
Postupně pro každý scénář a skupinu byl vytvořen průměr v každém zaznamenání měření. Tento průměr byl pak zanesen do 5 grafů dle scénářů.



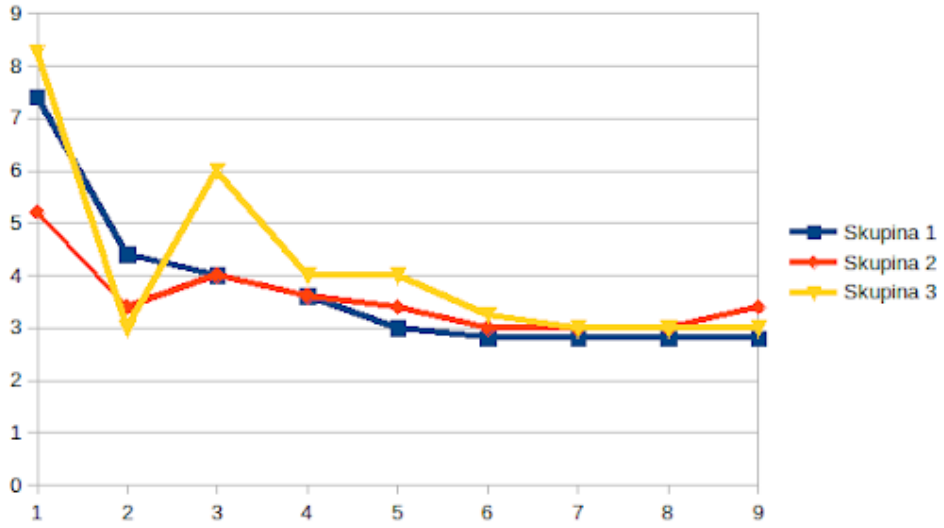
Obrázek 5.1: Průměrný čas pro první scénář. U skupiny č. 2 lze v 7. měření pozorovat výchylku.



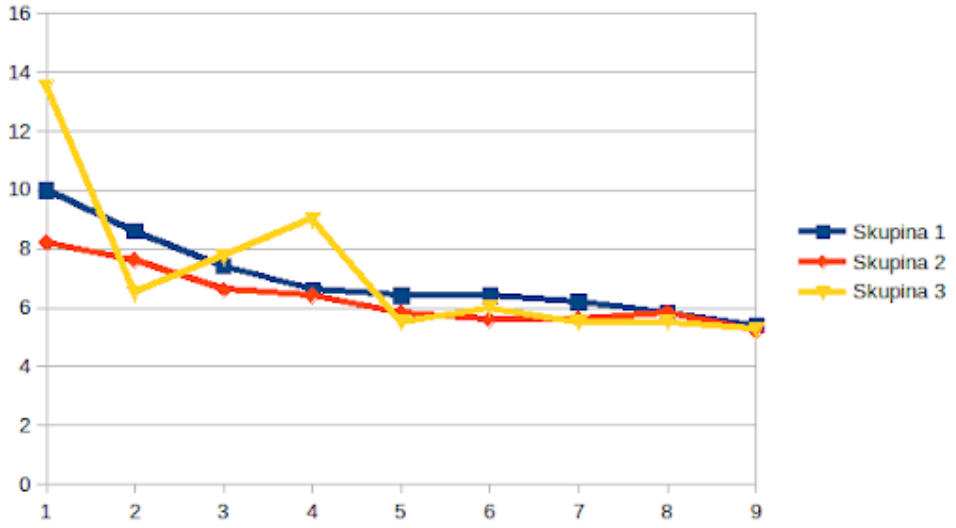
Obrázek 5.2: Průměrný čas pro druhý scénář. U skupiny č. 2 lze pozorovat výchylku v 7. měření a u skupiny č. 1 pak ve 4. měření.



Obrázek 5.3: Průměrný čas pro třetí scénář. U skupiny č. 2 lze v 5. měření pozorovat výchylku. Pro skupiny je vidět mírná výchylka v 6. měření.

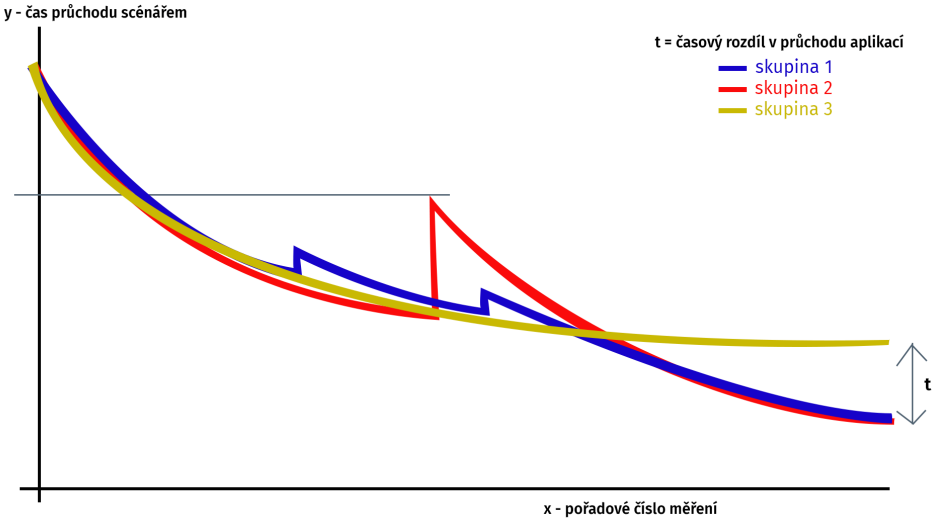


Obrázek 5.4: Průměrný čas pro čtvrtý scénář.



Obrázek 5.5: Průměrný čas pro pátý scénář.

Nejprve si rozebereme výsledky pro jednotlivé skupiny. Pro první skupinu jsou křivky vždy bez větších výchylek. To je způsobeno tím, že pro některé uživatele se průchod změnil a pro ostatní ne. Díky tomu je výsledný průměr jen lehce vyšší než předchozí. Chování připomíná třetí skupinu s tím rozdílem, že první skupina byla schopna posunout výsledné časy ještě více do nižších hodnot. Druhá skupina má viditelný bod - měření (viz. obrázky 5.1, 5.2 a 5.3), kdy byla struktura změněna. U scénářů č. 4 a č. 5 nebyla struktura pro druhou skupinu změněna vůbec. Příčinou je fakt, že v těchto scénářích uživatelé nevykazovali tolik emocí jako v předchozích.



Obrázek 5.6: Odvození obecného grafu pro všechny 3 způsoby adaptace

Nyní se zaměříme na celkové výsledky a jejich porovnání. Pokud porovnáme výsledné časy zobrazené na obecném obrázku 5.6, je vidět že finální měření adaptivních přístupů byla nejefektivnější (rozdíl vyznačen šipkami). Statické uživatelské rozhraní svou křivku srovnává a nemůže průchod již více zrychlit. Naopak adaptivní přístup zajišťuje změnu struktury a pomáhá tak uživatelům zrychlit práci.

Nakonec porovnejme automatickou adaptivní strukturu a tu vyvolanou administrátorem. Výsledné časy jsou stejné, jediný rozdíl je ve výchylce, kdy pro první skupinu nenastává adaptace pro všechny uživatele. Navíc změna byla provedena bez nutnosti úprav administrátorem. Můžeme tak považovat automatickou adaptaci za způsob úpravy uživatelského rozhraní s velkým potenciálem.

5.2.1 Ukliknutí uživatelů

Při testování bylo jedním z pozorovaných prvků také kliknutí na jinou položku, než uživatel vyžadoval. Tento prvek měl často za následek výkyvy v jednotlivých grafech, jelikož průměrné ukliknutí trvalo přibližně dvě sekundy. K ukliknutí docházelo častěji hlavně na začátku testování a při změnách struktury. Například v obrázku 5.4 došlo k ukliknutí u jednoho z uživatelů druhé skupiny v poslední den měření a má to za následek určitý vyšší výsledný průměrný čas.

Nás, jak už bylo naznačeno, nejvíce zajímá součet těchto překliků. Součty pro jednotlivé skupiny jsou zaznamenány v tabulce 5.2.

Skupina	Součet ukliknutí
Skupina 1	20
Skupina 2	17
Skupina 3	12

Tabulka 5.2: Počet překliknutí pro jednotlivé skupiny

Jak je vidět z tabulky 5.2, k překliknutí docházelo nejčastěji u prvních dvou skupin, pro které se struktura měnila. Uživatelé nebyli na nové umístění položek zvyklí, a tak se často stávalo, že se uklikli. Pro třetí skupinu jsme pak pozorovali snížení počtu překliknutí se zvyšujícím se počtem měření. Můžeme

tedy vyvodit závěr, že ačkoliv adaptace struktury zefektivní rychlost průchodu, může naopak narušit zaběhlé zvyklosti uživatelů a vyvolat frustrovanost.

5.3 Dotazník k testované struktuře

Po testování samotném dostal každý uživatel krátký dotazník na průchod strukturou demo aplikace. Pro každý scénář mohl vybrat jednu ze tří možností.

- Nebylo těžké po delší době testování šablonu nalézt.
- Po přesunu položek se hledání usnadnilo.
- Velmi špatné umístění, položka nebyla nikdy k nalezení.

Pro drtivou většinu scénářů byli uživatelé spokojeni s průchodem aplikací nebo po její adaptaci. Pouze u scénáře č. 3 až 15% uživatelů uvedlo, že byla cesta zmatečná. Na konci formuláře byly ještě následující dvě otázky:

1. Zdá se vám automatická adaptace průchodu strukturou aplikace na základě emocí jako správný způsob aktualizace struktury?
2. Zdá se vám ručně korigovaná adaptace průchodu strukturou aplikace na základě emocí jako správný způsob aktualizace struktury?

Na obě otázky odpovědělo 15-23% negativně, přičemž shodně uvedli, že jsou radši, když nemusí měnit své zvyklosti. Jeden uživatel dokonce navrhnul zajímavé řešení. Změna struktury není vůbec špatná, ačkoliv by bylo lepší, kdyby po změně zůstala komponenta i na původním místě pro přechodnou dobu. Uživatelé by si tak zapamatovali nové umístění a přechod by byl snadnější. Tento návrh však patří spíše do budoucího vývoje.

Kapitola 6

Instalace

Vývoj tohoto frameworku a testovací demo aplikace probíhal na operačním systému Linux Mint 19.1 v prostředí Android Studio, verze 3.6.1. Framework je kompatibilní s ostatními operačními systémy a je tak možné jej rozvíjet na všech platformách podporujících Java 8. Pro instalaci využijte následující postup.

Kroky instalace:

1. Vložte CD do zařízení
2. Otevřte komprimovaný soubor ve svém zařízení
3. V Android Studiu nebo jiném IDE naimportujte demo aplikaci
4. Připojte své Android OS zařízení
5. Spusťte build aplikace a následně aplikaci samotnou na svém Android OS zařízení
6. Hotovo, aplikace nyní běží na vašem telefonu.

Kroky pro rozšíření:

1. Vložte CD do zařízení

2. Otevřte komprimovaný soubor ve svém zařízení
3. Přidejte závislost na frameworku do vaší aplikace
4. Dále postupujte dle kapitoly 4.2.4
5. Implementujte

Kapitola 7

Závěr

Cílem práce bylo navrhnout rozšíření stávajícího frameworku na rozpoznávání emocí o adaptaci uživatelského rozhraní. Analýza stanovila, které struktury aplikací bude dobré adaptovat a které naopak ne. Vhledem do emocí uživatele na základě věku a pohlaví byly rozděleny subjekty na čtyři skupiny a za pomoci studie [1] o vyčlenění emočních intervalů se následně podařilo připravit podklady pro rozpoznání špatných emocí.

Návrh nejprve připravuje všechny objekty používané frameworkem a dále vytvoří seznam několika druhů průchodu stromem se stanovením druhu problému, ukázkami a adaptací. Součástí výčtu bylo také stanovení časové hranice pro setrvání v uzlu stromu. V posledním bodu návrhu byl připraven postup implementace a návrh jednotlivých částí frameworku.

V implementaci se podařilo rozšířit původní framework o adaptaci uživatelského rozhraní. Načtení jednotlivých stránek je dle návrhu prováděno dynamicky a využívá předem definované emoční intervaly. Výhodou tohoto frameworku je, že se zabývá málo prozkoumanou oblastí adaptace uživatelského rozhraní na základě emocí, která je velmi perspektivní pro budoucí rozvoj.

Pro testování byla využita struktura aplikace Air Bank. Testované subjekty byly rozděleny do tří skupin a každá z nich měla jiný typ adaptace. Z výsledků je vidět, že právě adaptivní struktura aplikace má potenciál na zlepšení efektivity práce uživatelů s uživatelským rozhraním. Všechny body zadání tak byly splněny.

Kapitola 8

Budoucí práce

Vytvořený framework byl navržen a vytvořen pro jednoduchou úpravu a rozšíření. Díky tomu má velký potenciál pro další rozvoj. Jedna z možností může být kombinace s dalšími periferiemi na rozpoznávání emocí. Jako první se nabízí zvuk [12], následuje záznam o tepu [11] díky chytrým hodinkám a náramkům. Neméně zajímavé se zdá být sledování informací o poloze [10] uživatele. To vše nám může zpřesnit záznam dané emoce a vytvořit velmi dobrý podklad pro adaptaci.

Z hlediska adaptace uživatelského rozhraní máme možnost rozšíření tohoto frameworku o jiné druhy adaptace. Jednou z variant je adaptace vzhledu pozadí nebo velikosti komponent pro zpříjemnění uživatelské zkušenosti. Optimální pro to by bylo současně rozšířit XML generátor o další typy komponent.

Poslední možnost, která se nabízí, je vytvoření konkrétní aplikace se všemi důležitými funkcemi a kompletním grafickým uživatelským rozhraním. Tato možnost může být založena na vytvořené testovací aplikaci.

Příloha A

Seznam zkratek

- XML - Extensible Markup Language
- JDK - Java development kit
- DFS - Depth first search
- JPA - Java Persistence API
- CRUD - Operace create, read, update a delete
- UI - User Interface
- PCA - Principal Component Analysis
- HCA - Hierarchical Cluster Analysis

Příloha B

Ukázky kódu

Příloha C

Obsah přiloženého média

- code - adresář s implementační částí
  - demo-app - testovací demo aplikace na adaptaci
  - framework - vytvořený framework na adaptaci
  - readme.md - instrukce k použití
- form - ukázka testovacího formuláře
- img - soubor s obrázkami
- pdf - soubor obsahující práci a zadání
- tex - zdrojové soubory práce pro LaTeX

Příloha D

Ukázka testovacího formuláře

Spokojenost při hledání scénářů

Prosím, vyplňte krátký dotazník ke spokojenosti testování struktury aplikace.
*Povinné pole

1. Jméno a příjmení *

_____

2. Scénář 1: Nová šabona *

Označte jen jednu elipsu.

- ◯ Nebylo těžké po delší době testování šablonu nalézt
- ◯ Po přesunu to bylo lepší
- ◯ Velmi špatné umístění, položka nebyla nikdy k nalezení

3. Scénář 2: E-shopy *

Označte jen jednu elipsu.

- ◯ Nebylo těžké po delší době testování položku nalézt
- ◯ Po přesunu to bylo lepší
- ◯ Velmi špatné umístění, položka nebyla nikdy k nalezení

4. Scénář 3: Nové pravidelné spoření *

Označte jen jednu elipsu.

- ◯ Nebylo těžké po delší době testování položku nalézt
- ◯ Po přesunu to bylo lepší
- ◯ Velmi špatné umístění, položka nebyla nikdy k nalezení

Příloha E

Literatura

[1] J. A. Galindo, S. Dupuy-Chessa, N. Mandran, and E. Céret, "Using user emotions to trigger ui adaptation," in *2018 12th International Conference on Research Challenges in Information Science (RCIS)*. IEEE, 2018, pp. 1–11.

[2] F. Gullà, S. Ceccacci, M. Germani, and L. Cavalieri, "Design adaptable and adaptive user interfaces: A method to manage the information," in *Ambient Assisted Living*. Springer, 2015, pp. 47–58.

[3] harmitag@stanford.edu, "'Smart toilet' monitors for signs of disease." [Online]. Available: http://med.stanford.edu/news/all-news/2020/04/smart-toilet-monitors-for-signs-of-disease.html

[4] B. A. Myers and D. R. Olsen Jr, "User interface tools," in *Conference companion on Human factors in computing systems*, 1994, pp. 421–422.

[5] D. M. Nichols, D. McKay, and M. B. Twidale, "Participatory usability: supporting proactive users," in *Proceedings of the 4th Annual Conference of the ACM Special Interest Group on Computer-Human Interaction*, 2003, pp. 63–68.

[6] "What are Adaptive User Interfaces?" dec 2017. [Online]. Available: https://ds.gpii.net/content/what-are-adaptive-user-interfaces

[7] L. Anastasiia, "Adaptace UI založena na emocích uživatele," May 2017. [Online]. Available: https://dspace.cvut.cz/handle/10467/68521

[8] "What is the User Interface?" [Online]. Available: https://www.workana.com/i/glossary/what-is-the-user-interface/

[9] G. W. Musumba and H. O. Nyongesa, "Context awareness in mobile computing: A review," *International Journal of Machine Learning and Applications*, vol. 2, no. 1, p. 5, 2013.

[10] G. C.-L. Hung, P.-C. Yang, C.-C. Chang, J.-H. Chiang, and Y.-Y. Chen, "Predicting negative emotions based on mobile phone usage patterns: an exploratory study," *JMIR research protocols*, vol. 5, no. 3, p. e160, 2016.

[11] J. A. Chalmers, J. A. Heathers, M. J. Abbott, A. H. Kemp, and D. S. Quintana, "Worry is associated with robust reductions in heart rate variability: a transdiagnostic study of anxiety psychopathology," *BMC psychology*, vol. 4, no. 1, p. 32, 2016.

[12] A. Grünerbl, A. Muaremi, V. Osmani, G. Bahle, S. Öhler, G. Tröster, O. Mayora, C. Haring, and P. Lukowicz, "Smartphone-Based Recognition of States and State Changes in Bipolar Disorder Patients," *IEEE Journal of Biomedical and Health Informatics*, vol. 19, no. 1, pp. 140–148, Jan. 2015.

[13] M. Kim, H.-J. Kim, S.-J. Lee, and Y. S. Choi, "A touch based affective user interface for smartphone," in *2013 IEEE International Conference on Consumer Electronics (ICCE)*, Jan. 2013, pp. 606–607, iSSN: 2158-4001.

[14] D. McDuff, A. Mahmoud, M. Mavadati, M. Amr, J. Turcot, and R. e. Kaliouby, "Affdex sdk: a cross-platform real-time multi-face expression recognition toolkit," in *Proceedings of the 2016 CHI conference extended abstracts on human factors in computing systems*, 2016, pp. 3723–3726.

[15] L. Findlater and J. McGrenere, "A comparison of static, adaptive, and adaptable menus," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2004, pp. 89–96.

[16] "Facebook Revenue and Usage Statistics (2020)," Aug. 2017. [Online]. Available: https://www.businessofapps.com/data/facebook-statistics/

[17] "Gmail Statistics And Trends: What You Need To Know In 2020," Mar. 2019. [Online]. Available: https://techjury.net/stats-about/gmail-statistics/

[18] "Air Bank roste v mobilním bankovnictví." [Online]. Available: https://www.czechbanking.cz/air-bank-dal-roste-v-mobilnim-bankovnictvi/

[19] J. Nešetřil and P. O. De Mendez, "Tree-depth, subgraph coloring and homomorphism bounds," *European Journal of Combinatorics*, vol. 27, no. 6, pp. 1022–1041, 2006.

[20] S. Miettinen *et al.*, "Emotion differentiation," 2011.

[21] "Human Benchmark." [Online]. Available: https://humanbenchmark.com/tests/reactiontime/statistics

[22] H. Lee, Y. S. Choi, and Y.-J. Kim, "An adaptive user interface based on spatiotemporal structure learning," *IEEE Communications Magazine*, vol. 49, no. 6, pp. 118–124, 2011.

[23] N. Haderer, R. Rouvoy, and L. Seinturier, "Dynamic deployment of sensing experiments in the wild using smartphones," in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2013, pp. 43–56.

[24] "LayoutInflater." [Online]. Available: https://developer.android.com/reference/android/view/LayoutInflater?hl=cs

[25] S. Komatineni and D. MacLean, *Pro Android 2*, ser. Books for professionals by professionals. Apress, 2010. [Online]. Available: https://books.google.gp/books?id=XLAJXwcCRz0C

[26] N. White, "nickwah/DynamicLayoutInflator," may 2020, original-date: 2015-08-07T23:37:56Z. [Online]. Available: https://github.com/nickwah/DynamicLayoutInflator

[27] K. Pelgrims, *Gradle for Android*. Packt Publishing Ltd, 2015.

[28] V. Vernon, *Implementing domain-driven design*. Addison-Wesley, 2013.

[29] "Domain Model vs. Data Model - Video & Lesson Transcript." [Online]. Available: https://study.com/academy/lesson/domain-model-vs-data-model.html

[30] "Save data in a local database using Room." [Online]. Available: https://developer.android.com/training/data-storage/room?hl=cs

[31] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[32] P. Mori, S. Furnell, and O. Camp, *Information Systems Security and Privacy: 4th International Conference, ICISSP 2018, Funchal-Madeira, Portugal, January 22-24, 2018, Revised Selected Papers*. Springer, 2019, vol. 977.

[33] N. N. Group, "How Many Test Users in a Usability Study?" [Online]. Available: https://www.nngroup.com/articles/how-many-test-users/