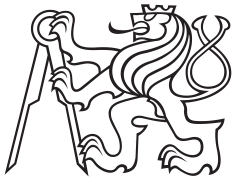
Bakalárska práca



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

# Využití stromových operací při návrhu pravidel v adaptivních strukturách

Michal Matija

Studijní program: Otevřená informatika

Studijní obor: Informatika a počítačové vědy

Máj 2019

Vedúci práce: Ing. Jiří Šebek

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Matija** Jméno: **Michal** Osobní číslo: **465938**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Studijní obor: **Informatika a počítačové vědy**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Využití stromových operací při návrhu pravidel v adaptivních strukturách**

Název bakalářské práce anglicky:

**Using Tree Trends to Design Rules in Adaptive Structures**

Pokyny pro vypracování:

Uživatelské rozhraní je klíčovou součástí softwarových aplikací pro uživatelské zkušenosti a použitelnost systému. Nové systémy musí čelit stále rostoucím očekáváním uživatelů a snažit se, aby plně využily svůj potenciál přilákat uživatele. K tomuto účelu se využívají kontextové informace aplikace (ze senzorů, kamery, zadaných informací atd.). Jednou z nejsložitějších struktur v aplikacích je menu. Většina struktur v aplikacích, jako je menu, seznamy, widgety a další, se dají reprezentovat pomocí modelu. Tento návrh se nazývá Adaptivní aplikační struktura (AAS)[1,2,3].
Návrh adaptivních struktur má výhodu, že se umí přizpůsobovat, ale oproti statickým strukturám může působit pro uživatele mírně chaoticky. [1] Ale při vhodném návrhu pravidel adaptivní struktura zlepšuje a zefektivňuje užívání aplikace uživatelům.
Cíle práce:
1. Zjistit a prozkoumat jednotlivé problematiky spojené s adaptivní strukturou aplikace (AAS) [1].
2. Navrhnout další adaptační pravidla dle [1], které by řešily rozvržení prvků v stromu AAS, aby zlepšily čas strávený uživatelem na daném úkolu, eliminovaly by množství negativních aktualizací (operace, které zhorší použitelnost aplikace) a další zjištěné problémy. Dále je potřeba navrhnout kooperaci vice adaptivních menu.
3. Najít propojení mezi jednotlivými pravidly [1].
4. Integrovat nová pravidla do systému [1].
5. Všechny nové funkcionality otestovat jak uživatelskými testy tak heuristikami, a vyhodnotit je.

Seznam doporučené literatury:

[1] 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.
[2] ŠEBEK, J., M. TRNKA, and T. ČERNÝ. On Aspect-Oriented Programming in Adaptive User Interfaces. In: Proceedings of the 2nd International Conference on Information Science and Security. The 2nd International Conference on Information Science and Security, Seoul, 2015-12-14/2015-12-16. Piscatway: IEEE, 2015, pp. 147-151.
[3] MISHCHENKO, Nikita. Aspektově orientovaný vývoj adaptivní struktury aplikace pro Java EE aplikace. Praha, 2016. Bakalářská práce.ČVUT v Praze, Fakulta elektrotechnická.

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: **15.02.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **30.09.2020**

| Ing. Jiří Šebek | doc. Ing. Tomáš Svoboda, Ph.D. | prof. Ing. Pavel Ripka, CSc. |
|---|---|---|
| podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | 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

# Poďakovanie / Prehlásenie

Chcel by som poďakovať Ing. Jiřímu Šebekovi za vedenie a cenné rady pri vytváraní bakalárskej práce.

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

In Prague 22. 5. 2019

.....

# Abstrakt / Abstract

Bakalárska práca sa zaoberá hľadaním nových pravidiel pre adaptívne štruktúry. Prácu možno rozdeliť do troch celkov, kde prvým je rozlišovanie užívateľských rolí. Bol navrhnutý algoritmus, ktorý bude detekovať navigovanie po menu od prezerania stránky. Súčasne sa bude dynamicky prispôsobovať užívateľovi. Detekcia využíva viacero informácií pre rozhodovanie, a to čas strávený na prvku, počet slov na stránke a počet navštívení. Druhým celkom je rozvrhnutie prvkov v uzle, v ktorom sa hľadá optimálne poradie prvkov, pričom sa zohľadňuje rýchlosť preklikávania a priamočiarosť k nájdeniu prvku. Využívajú sa informácie od viacerých užívateľov a aplikujú zmenu na konkrétneho. V poslednom celku ide o kooperáciu viacerých menu, kde algoritmus detekuje prechody medzi viacerými a následne upraví model prehodením prvku alebo zoradením druhého menu na základe pozícii v prvom. V pravidlách sa snažíme zlepšiť čas strávený užívateľom na danom prvku a eliminovať negatívne aktualizácie. Všetky pravidlá sú implementované do Java EE frameworku, ktorý umožňuje jednoduchým spôsobom jeho využívanie. Pri riešení bol použitý programovací jazyk Java s použitím Spring frameworku a databázovým serverom od Oracle 12c.

**Kľúčové slová:** Java, Spring framework, adaptívna štruktúra, menu, adaptívne pravidlá

The bachelor thesis deals with finding new rules for adaptive structures. The thesis can be divided into three parts, where the first is to distinguish user roles. An algorithm has been designed to detect menu navigation from page browsing. Concurrently, it will dynamically adapt to the user. Detection uses multiple decision-making information, namely the time spent on an item, the number of words per page, and the number of visits. The second part is the sorting of the items in the node in which the optimal order is searched for, taking into account the speed and directness click through to find the item. It uses information from multiple users and applies the change to a specific one. The last part is about the cooperation of several menus where the algorithm detects the transitions between multiple menus and subsequently adjusts the model move the item or by sorting a second menu based on the position in the main menu. In the rules, we try to improve the time spent on a given item and eliminate negative updates. All rules are implemented into the Java EE framework, which makes it easy to use. The Java programming language was used in the solution using Spring framework and database server from Oracle 12c.

**Keywords:** Java, Spring framework, adaptive structure, menu, adaptive rules

# Obsah /

# Tabuľky / Obrázky

# Kapitola 1
# Úvod

## 1.1 Motivácia

U väčšiny aplikácií či už webových, mobilných alebo desktopových sa môžeme stretnúť s tým, že poskytujú užívateľské rozhranie, ktoré ponúka ovládania jednotlivých častí aplikácii. Zároveň by malo byť pre užívateľa intuitívne, zrozumiteľné a ľahké na používanie.

Užívatelia s danými aplikáciami prichádzajú každodenne do styku a využívajú ich. Rad radom sa od seba líšia, ale častokrát majú spoločnú jednu vec a to tú, že pre lepšiu manipulovateľnosť používajú menu. Typy menu môžu byť rôzne ako drop-down, pie, mega atď. Vhodnosťou správneho typu sa snažia programátori docieliť užívateľské rozhranie viac pochopiteľnejším. Ďalším spôsobom, ktorým ho chcú vylepšiť, je vhodne rozvrhnutie prvkov v štruktúre. Pridávajú ďalšie menu na svoje stránky alebo do aplikácii, aby znížili komplexnosť tým, že rozdelia úlohu na podúlohy. Tieto užitočné veci využívajú väčšinou statické štruktúry a programátori manuálne nastavujú rozvrhnutie alebo pridanie či odstránenie prvku.

Informácií je dnes strašne mnoho. Preto ich ľudia chcú hľadať čo najrýchlejšie a čo najúčinnejšie. Tým pádom navigovanie po menu by malo byť priamočiare a trvať krátko. Mnoho aplikácii ani nedisponuje štatistikou navigovania po menu pri vyhľadávaní informácií. Väčšinou je to testované na istej vzorke ľudí a až tak aplikované do produkcie. Tie, ktoré týmto disponujú, používajú vlastné riešenia pre tento účel.

Všetky vyššie spomenúte problémy vedú na vytvorenie niečoho univerzálnejšieho a prepoužiteľného. Tým niečim je adptívna štruktúra, ktorá sa bude prispôsobovať v určitom čase podľa správania a daného kontextu k užívateľovi.

## 1.2 Cieľ práce

Cieľom práce je nájsť nové pravidlá pre adaptívne štruktúry, s ktorými sa bežne môžeme stretnúť pri používaní menu. Pre nové pravidlá navrhnúť algoritmy, aby sa vyriešili problematiky, ktoré popisujú konkrétne pravidlá. Pri návrhu dbať na

negatívne aktualizácie, frustráciu užívateľa a aby výsledný model nepôsobil mätúco. Ďalej zanalyzovať výhody a nevýhody konkrétneho problému a navrhnúť možné ďalšie vylepšenia a rozšírenia.

Po rozbore a navrhnutí algoritmov skúsiť nájsť prepojenia medzi konkrétnymi riešeniami tak, aby mohli fungovať kolektívne a poskytovať informácie medzi sebou.

Následne implementovať tieto algoritmy do už existujúceho Java Enterprise Edition frameworku, ktorý umožňuje generovať meta model užívateľského rozhrania v závislosti na kontexte. Pri implementovaní využívať design patterny a paradigma, ktoré sa bežne používajú pri návrhu webových aplikácii.

Na záver práce sa zamerať na otestovanie pravidiel na užívateľoch za použitia rozšíreného frameworku a vyhodnotiť zistené poznatky.

# Kapitola 2
# Rešerše

## 2.1 Typy rozhraní

Pri používaní aplikácii sa môžeme stretnút s tromi typmi rozhraní:

- Statické
- Adaptívne
- Adaptateľné

**Statické rozhrania** majú na začiatku nastavené rozvrhnutie a počas behu aplikácie sa nemení. Mení ho manuálne programátor, ak pridáva novú funkcionalitu, alebo odoberá prvok. Väčšinou návrh spočíva najskôr testovaním na užívateľoch a podľa výsledkov vytvoria užívateľské rozhranie.

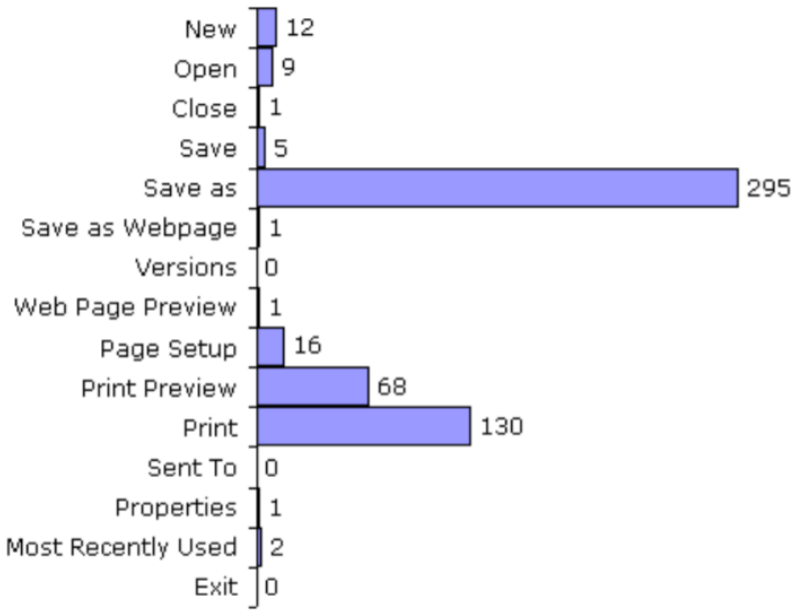
- Rovnaké rozhranie pre každého užívateľa
- Sú nemenné počas behu aplikácie

**Adaptívne rozhrania** zisťujú chovanie užívateľa počas behu aplikácie a na základe toho kvalitatívne vyhodnocujú. Vzápäti sa dané rozhranie prispôsobuje. Informácie, ktoré získava sú napr. frekvencia používania alebo čas trávenia na prvku.

- Prispôsobujú sa k užívateľovi
- Menia svoju štruktúru
- Dlhšie učenie
- Hlavne na začiatku môžu pôsobiť mätúco
- Využíva navyše informácie pre adaptáciu.

**Adaptateľné rozhrania** má v réžii užívateľ, ktorý si rozvrhnutie prvkov môže sám prispôsobiť podľa seba.

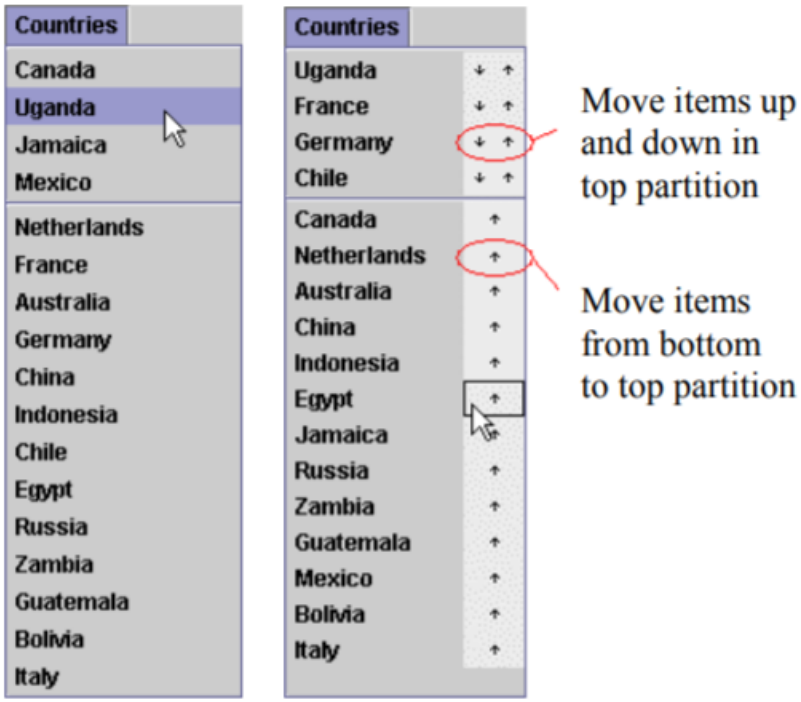
- Prispôsobuje si sám užívateľ
- Bez zásahu užívateľa fungujú rovnako ako statické




(a) `File` (15 items, 541 selections)

**Obrázok 2.1.** Príklad frekvencií klikania pri adaptivních štruktúrach[26]

Obrázok 2.1 zobrazuje schému klikania na prvky v menu. Na základe týchto klikaní sa môže štruktúra adaptovať. Kde poradie prvkov bude zostupné vzhľadom na početnosť kliknutí. Na obrázku 2.2 môžeme vidieť príklad ďalších dvoch štruktúr. Na pravom menu vidíme možnosť presúvania prvkov v menu.




**Obrázok 2.2.** Príklad statického a adabtateľného split menu[16]

### 2.1.1 Porovnanie statického, adaptívneho a adaptateľného rozhrania

Tabuľka 2.1 poukazuje na vlastnosti a problémy pri jednotlivých rozhraniach.

Statické rozhrania trpia jednotnou štruktúrov pre všetkých užívateľov. Každý užívateľ je iný a má iné požiadavky, takže jedná štruktúra nemusí byť vhodná pre všetkých. Rozdielnosť ľudí spôsobuje problém aj pri nelogickom rozvrhnutí. Čo môže spôsobiť frustráciu[28].

Adaptívne rozhrania bývajú nepredvidateľné, pretože nietoré akcie vykonané systémom sa nemusia stretnúť s očakavaním užívateľa. S tým súvisí aj neprehľadnosť, kde užívateľ nechápe fungovaniu adaptácie. Ďalším problémom môže byť, že dochádza k narúšaniu súkromia, tým že adaptívne rozhrania si zberajú informácie o interakcii užívateľa so systémom. Viera v systém je flutujúca. Pri zlých nápovedách môže veľmi znížiť dôveru. Môže dochádzať k pocitu straty kontroly nad systémom[26].

Adabtateľné rozhrania prinašajú výhodu v predvidateľnosti, pretože sú upravené samotným užívateľom. To má následok k zrozumiteľnosti daného rozhrania a nespôsobujú úzkosť. Naproti adaptívnym rozhraniam môže mať užívateľ pocit, že ovladaná daný systém a tak ho ma pod kontrolou[29]. Problém pri adabtateľných rozhraniach prichádza vtedy, ak užívatelia nemajú záujem meniť danú štruktúru. Taktiež zručnosť užívateľov má dôležitý dopad k využívaniu tohto typu rozhrania. Napríklad programátory budú skôr chcieť prispôsobiť dané rozhranie, ako bežný človek[30]. Znechutenie adaptateľného rozhrania môže spôsobiť častá aktualizácia systému čo nutí užívateľa stále si prispôsobiť dané rozhranie. Môže to mať za následok, že prestane si ho prispôsobovať[26].

| Statické | Adaptívne | Adabtateľné |
|---|---|---|
| Jednotnosť | Nepredvidateľné | Predvidateľné |
| Rozvrhnutie | Viera | Zrozumiteľné |
| | Nepriehladné | Ovladateľné |
| | Súkromie | Záujem |
| | Strata kontroly | Zručnosť |
| | | Časté aktualizácie |

**Tabuľka 2.1.** Vlastnstí a problémy jednotlivých rozhraní

Štúdia[16] porovnávala tri typy menu statické, adaptívne a adaptateľné. Testovali sa 4 hypotézy:

1. *„adaptívne je pomalšie ako statické a adaptateľné“*
2. *„adaptateľné nie je pomalšie ako statické“*
3. *„adaptateľné je preferované pred adaptívnym a statickým“*
4. *„statické je preferované pred adaptívnym“*

Dospelo sa k nasledujúcim výsledkom:

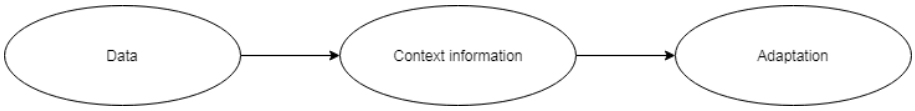
1. *„Adaptívne menu bolo pomalšie ako statické. Adaptívne menu bolo tiež pomalšie ako adaptateľné, okrem prípadov, keď subjekty používali najprv adaptateľné menu.“*
2. *„Adaptateľné menu nebolo pomalšie ako statické, okrem prípadov, keď subjekty používali najprv adaptateľné menu.“*
3. *„Adaptateľné menu bolo preferované pred statickým, ale nie pred adaptívnym.“*
4. *„Statické nebolo preferované pred adaptívnym.“*

Z výsledkov môžeme vidieť, že ani jedno menu statické, či adaptateľné nie je preferované pred adaptívnim. Horšie to je už pri rýchlosti adaptívneho menu. Dôvodom prečo adaptívne menu dopadlo ako najpomalšie je, že vykonávalo príliš veľa zmien. Bola použitá frekvencia preklíkavania medzi jednotlivými prvkami bez obmedzenia fluktuácie. To býva pri adaptívných štruktúrach veľkým negatívom, ale pri správnom návrhu a nevytvaraním príliš častých zmien sa dá znížiť frustrácia užívateľa.

## 2.2 Návrh adptívneho rozhrania

Adaptívne rozhrania pozostávajú pri behu aplikácie z troch hlavných časti:

- Získávania dát
- Kontextových informácií
- Adaptácii




**Obrázok 2.3.** Hlavné časti adaptívneho rozhrania

V prvom rade sa získávajú dáta od užívateľa. Príkladom môže byť už spominaná frekvencia navštívenia. V druhej časti dochádza k hľadaniu kontextových informácií. Kontextové informácie charakterizujú objekt, osobu alebo miesto, ktoré sú

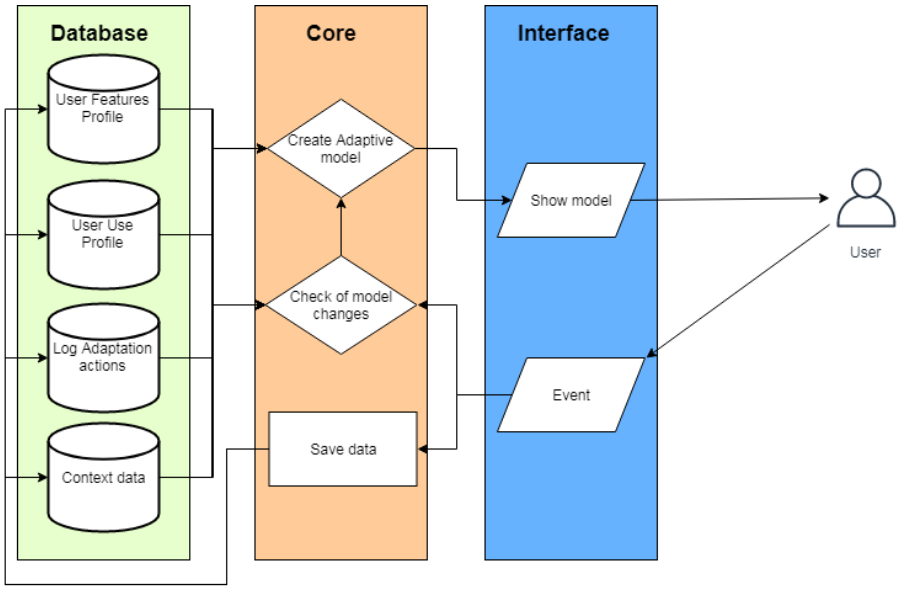
v interakcii medzi užívateľom a aplikáciou. Kontext taktiež zahŕňa stavy alebo nastavenia prostredia, ktoré určujú chovanie aplikácie[17]. Príkladom kontextovej informácie môže byť rýchlosť navigácie, ktorá je určená na základe času potrebného na výkonanie úlohy. Poslednou časťou je adaptácia, kedy dôjde k zmene UI. Pri adaptácii môžeme napríklad zmeniť poradie prvkov.

Architektúra na základe vyššie spomenutých informácií pozostáva z troch hlavných jednotiek. Tieto jednotky medzi sebou neustále komunikujú. Ide o databázu, jadro systému a UI.

Databáza udržiava niekoľko dát, ktoré môžeme rozdeliť na:

- Profil užívateľa (User Features Profile)
- Použitie užívateľského profilu (User Use Profile)
- Záznamy adaptačných akcií (Log Adaptation Actions)
- Kontextové dáta (Context Data)

Profil užívateľa pozostáva z osobných informácii. Použitie užívateľského profilu obsahuje informácie o kontexte modelu. Záznamy adaptačných akcií obsahujú všetky úkony, ktoré vykonáva rozhranie. Inými slovami ide o zaznamenávanie všetkých zmien, ktoré vznikli pri adaptácii. Kontextové dáta sú všetký údaje, hodnoty, ktoré boli pozorované, odmerané alebo zistené pri používaní aplikácie užívateľom. Jadro systému má za úlohu zisťovanie zmien, kedy má dôjsť k adaptácii a samotné vytváranie adaptívneho modelu. K tomuto cieľu využívajú informácie uložené v databáze. Poslednou časťou je užívateľské rozhranie, ktoré ma za úlohu zobrazovať vykonané zmeny a preposielanie informácii pri vzniknutí udalosti. Udalosť môže byť kliknutie na prvok v menu[15].



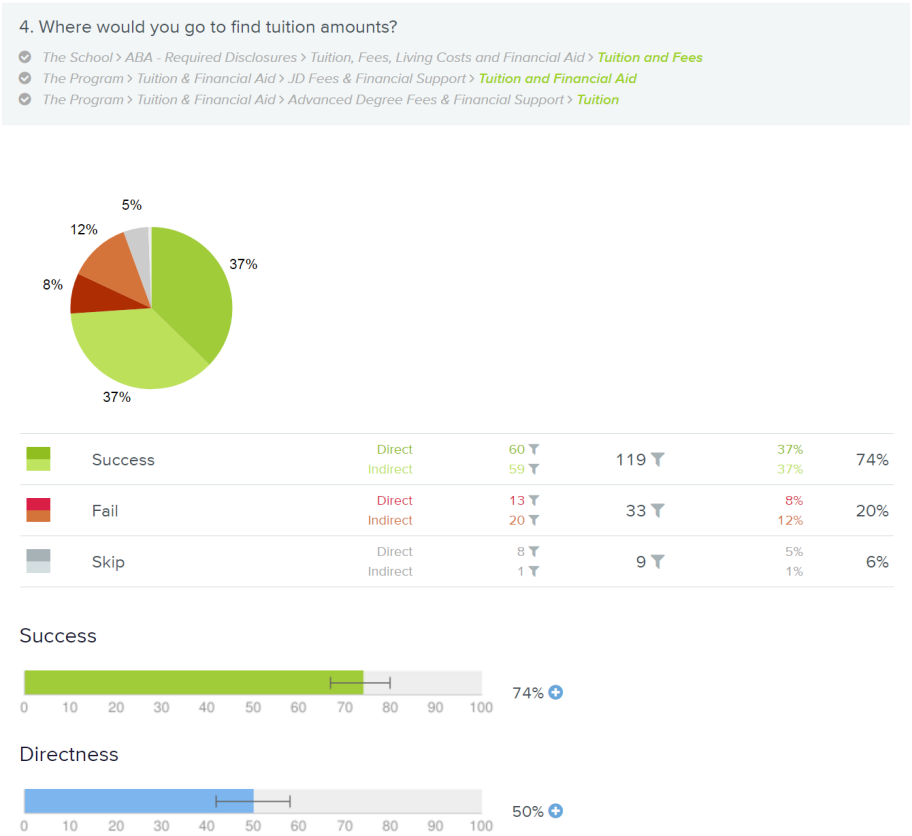

**Obrázok 2.4.** Architektúra pre adaptívne systémy

## 2.3 IA metódy pre rozvrhnutie prvkov

Spôsob ako sú usporiadané prvky v menu je pre užívateľoch taktiež dôležité. Rôzne typy radení môžu mať iný následok na užívateľa, ako napríklad užívateľ nájde to čo hľadá rýchlo, alebo preklikáva sa celou štruktúrou a nemôže sa dopátrať prvku, ktorý hľadal. To ho môže odradiť a nabudúce už stránku nenavštívi. Stačí, len správne roztriediť jednotlivé prvky, aby boli čo najintuitívnejšie resp. najvhodnejšie pre daného užívateľa. Vieme tým zvýšiť rýchlosť dosiahnutia informácie, čitateľnosť, nemätenie užívateľa a v konečnom dôsledku aj znížiť jeho frustráciu v hľadaní.

Pri statických štruktúrach sú rôzne spôsoby, ako dosiahnuť vhodné radenie pre užívateľov. V IA sú to metódy ako:

- **Cart Sorting** – čo je metóda, kde zisťujeme ako vhodne usporiadať jednotlivé prvky do kategórii, ktoré užívatelia vymyslia alebo roztriedia. Táto metóda ma výhodu, že ide o kvalitatívne vyhodnocovanie, kde máme spätnú väzbu od užívateľov a tým pádom nám stačí menej testovacích objektov, ako u kvantitatívnom vyhodnocovaní. Bežné typy card sortingu sú[10]:
  - **Open**: nepoznáme kategórie, ale iba prvky a zisťujeme aké kategórie užívatelia vytvárajú, a aké prvky k ním priradzujú.
  - **Closed**: poznáme kategórie i prvky, ale chceme zistiť, ktoré prvky kde umiestniť, do ktorej kategórie
  - **Hybrid**: spojenie vyššie spomenutých spôsobov, účastník postupuje, tak ako v closed sortingu, ale má dovolené vytvárať i nové kategórie.
- **Tree testing** – Tento spôsob testovania je vhodný, ak už vieme ako máme radené jednotlivé prvky v menu. Chceme zistiť, ako sú ľahko naleziteľné. Pri tomto teste sa účastníkovi položí otázka a užívateľ má za úlohu nájsť prvok, kde si myslí, že táto informácia sa bude nachádzať. Na rozdiel od cart sortingu toto testovanie je kvantitatívne a potrebuje viac užívateľov na otestovanie nejakého spôsobu radenia. Pri tejto metóde sa zisťuje niekoľko parametrov ako napríklad[5],[6]:
  - **Direct success**: našiel priamu cestu k hľadanému prvku
  - **Indirect success**: našiel cestu k hľadanému prvku, ale musel sa preklikávať vetvami, ktorými nemusel prechádzať
  - **Time spent**: čas potrebný na vykonanie úlohy
  - **First click**: čo bolo jeho prvým kliknutím pri hľadaní daného prvku
  - **Path**: sekvencie jednotlivých prvkov, ktoré boli navštívené počas úlohy



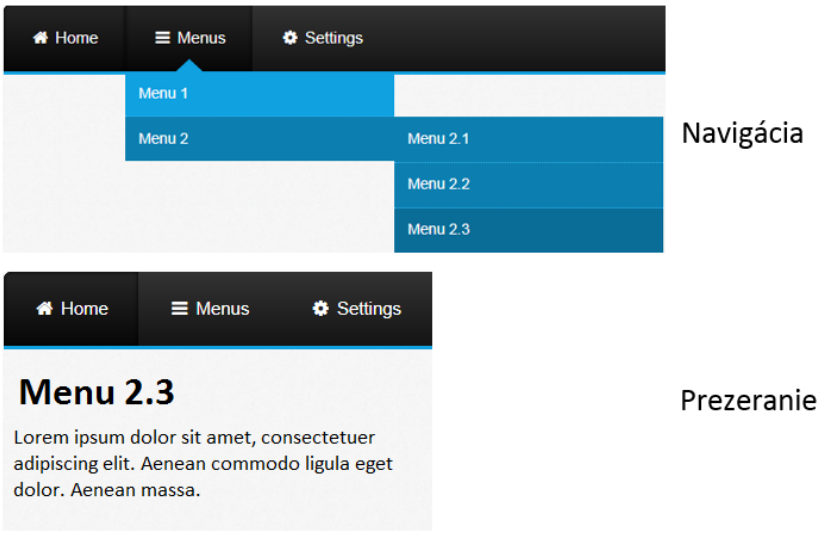

**Obrázok 2.5.** Ukážka vyhodnotenia Tree testingu[6]

# Kapitola 3
# Pravidla

## 3.1 Rozlišovanie užívateľských úloh pri navigácii menu

Ako samotný názov hovorí, že ide o zisťovanie užívateľských úloh pri navigácii menu, kde potrebujeme rozlišovať jednotlivé úkony užívateľa k následnému vyhodnocovaniu. Konkrétne ide o úlohy:

- **Navigácia** – pri tejto úlohe užívateľ prechádza menu a hľadá konkrétny prvok resp. stránku, na ktorú sa chce dostať.
- **Prezeranie** – užívateľ sa nachádza na nejakom prvku v menu a prezerá si jeho obsah




**Obrázok 3.1.** Užívateľské role

Úloha prezeranie by sa dala rozčleniť do niekoľkých bodov. Ako sledovanie obsahu stránky, ktorý užívateľ chcel. Kde náhodou prišiel a zaujala ho. Toto rozčlenenie by mohlo byť určite vhodné a dalo by sa zistiť viac o správaní užívateľa pri navigácii a samotnom prezeraní stránky. Pri adaptívnych štruktúrach mi nemáme žiadnu spätnú väzbu od užívateľov, aby sme vedeli kvalitatívne vyhodnotiť aktuálny model.

Pri metódach správneho návrhu rozvrhnutia menu (napr. Tree Testing) vieme zistiť, kedy daná úloha skončila a na ktorom prvku, pretože je tam spätná väzba. Vie sa teda posúdiť, kedy je úloha navigovaním, prezeraním a ešte ju aj rozčleniť. Túto metódu si však môžeme dovoliť pri statických štruktúrach, kde sa nám prvky nemenia počas behu aplikácie.

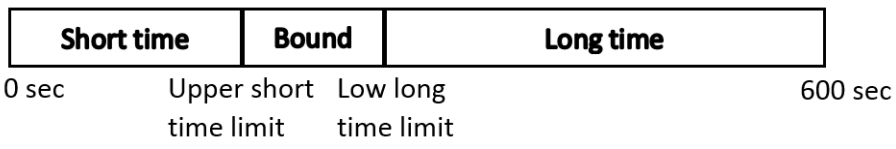
Ako bolo spomenuté vyššie, nemôžeme použiť žiadnu z bežne dostupných metód IA, kde by sa dala otestovať na nejakej vzorke ľudí a následne ju vyhodnotiť. Pri tomto probléme použijeme iné informácie, ktoré máme k dispozícii. Na základe nich budeme kvantitatívne testovať a následne vyhodnocovať. Budeme sa upriamovať na čas stravený na danom prvku menu, koľko slov obsahuje daný prvok a počet navštívení. Čas a počet slov nám pomôže rozlíšiť či ide o navigovanie alebo prezeranie a počet navštívení nám bude slúžiť na korekciu rozlíšenia.

Užívateľov si ešte rozdelíme do 3 skupín na základe veku. Dôvod tohto rozdelenia je, že každá skupina ľudí potrebuje iný čas pri navigácii. Napríklad mladší užívatelia potrebujú menej času, ako starší[19]. Užívateľské kategórie:

- Mladý(do 30 rokov)
- Dospelý (31 – 54 rokov)
- Starý (55 a viac rokov)

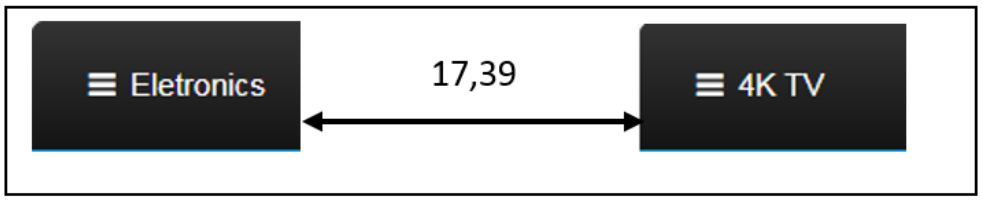
Pre rozlíšenie, v akej úlohe sa nachádza užívateľ, rozdelíme čas strávení na prvku do troch kategórii:

- **Krátky čas (Short time, ST)** Predstavuje navigovanie po menu. V niektorých prípadoch bude krátky čas označený dlhým. To sa bude používať pri skúsených užívateľoch, kedy potrebujú vykonať len jeden rýchly úkon.
- **Neznámi (Unknown, UK)** Nejednoznačnosť, nevieme sa rozhodnúť či ide o dlhý čas alebo krátky. Aktívne prvky budú označené ako dlhé, ale podobne ako výnimka v krátkom čase, bude podliehať zložitejšiemu rozhodovaniu. Zvyšné neklasifikované údaje budú ohodnotené pomocou **K-NN algoritmu**.
- **Dlhý čas (Long time, LT)** Označuje prvok, pri ktorom si užívateľ prezerá stránku a nenaviguje sa po menu. Tento čas bude mať obmedzenú hornú hranicu na 10min(600sec) a to z dôvodu, že chceme eliminovať vplyv extrémnych hodnôt, kedy by si užívateľ mohol niekde odbehnúť.




**Obrázok 3.2.** Znázornenie jednotlivých časov a hraničných hodnôt

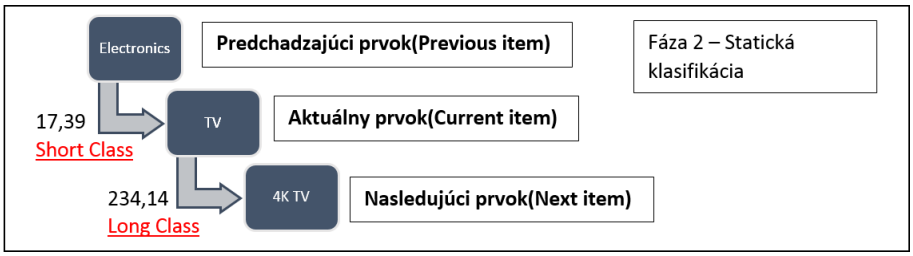
### 3.1.1 Fáza 1: Meranie časov




**Obrázok 3.3.** Príklad merania času

Na začiatku sa budú zberať časy od užívateľov, koľko času strávili, na ktorom prvku. Toto meranie bude prebiehať stále u každého užívateľa, v každom stave aplikácie.

### 3.1.2 Fáza 2: Statická klasifikácia




**Obrázok 3.4.** Príklad statickéj klasifikácie

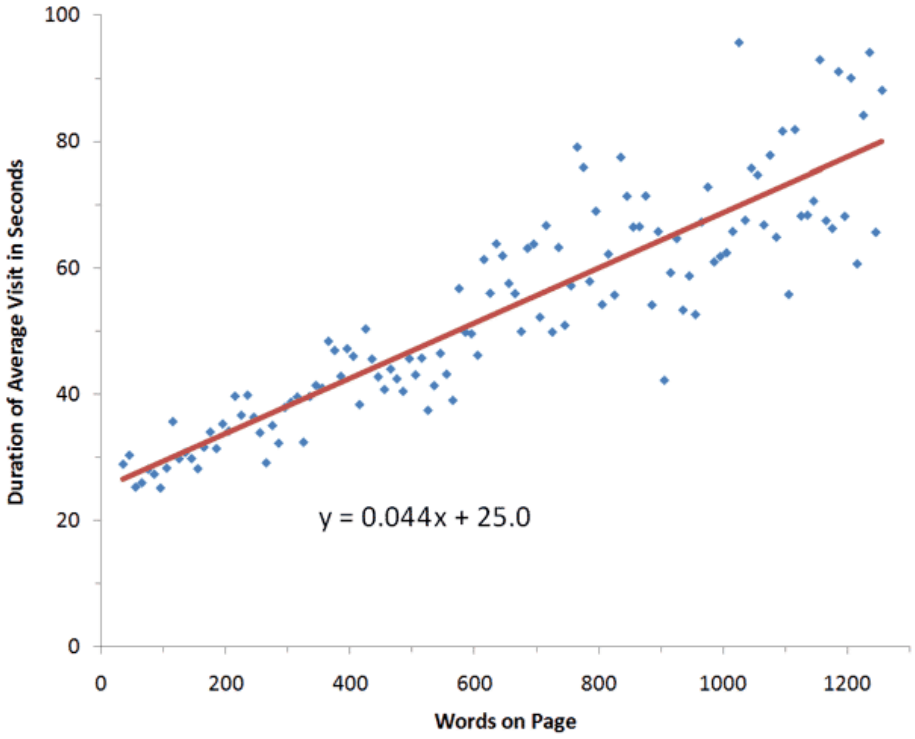
V druhej fáze dôjde k statickému klasifikovaniu časov do tried. Máme dve triedy a to **krátky** a **dlhý čas**. Pri vyhodnocovaní budeme využívať informácie, ako odmeraný čas a počet slov na stránke. Ďalej sa bude zohľadňovať, či je daný prvok v menu aktívnym. Rozhodovanie bude prebiehať nasledujúco ak ešte neprebehla **fáza 4**:

- Časy, ktoré budú dlhšie ako 3min. budú hneď klasifikované ako dlhé časy.
- Časy kratšie ako 10s budú klasifikované ako krátke časy. S výnimkou aktívneho prvku, kde sa bude prihliadať na nasledujúci prvok. Ak pôjde o prvok v stromovej štruktúre nachádzajúci sa hlbšie a čas bude dlhší bude sa klasifikovať ako krátky čas. V opačnom prípade ide o dlhý čas.
- Všetky ostatné prvky sa budú klasifikovať podľa nižšie zobrazeného grafu s prihliadnutím na predchádzajúce i nasledujúce neklasifikované časy.

Ak podľa grafu na obrázku 3.5 budú časy rovnaké alebo väčšie, ako priemerný čas pri danom počte slov na stránke(červená čiara v grafe), ide pravdepodobne o dlhý čas. Potom sa zohľadňujú predchádzajúce časy, kde aktuálny sa musí líšiť od predchádzajúcich a to výraznejšie. Na výraznejšie odlíšenie sa použije vzorec (3.1).

Ak to bude spĺňať, zohľadní sa nasledujúci prvok a ak bude taktiež výraznejšie kratší, klasifikuje sa ako dlhý čas. V opačnom prípade neklasifikujeme.

$$firstMinInSequence + secondMinInSequence < currentTime \tag{3.1}$$




**Obrázok 3.5.** Priemerný čas strávený na stránke pri danom počte slov[9]

Ak časy budú menšie ako je znázornená priamka v grafe na obrázku 3.5, potom ide pravdepodobne o krátky čas. Čakám na najbližší nasledujúci prvok, ktorého dĺžka je väčšia alebo rovná priamke na grafe. Ak sa nasledujúci prvok označí ako dlhý, potom predchádzajúce budú krátke. Ak sa neklasifikuje, potom ani predchádzajúce nebudú.

Vzorec pre zistenie či je čas menší alebo väčší, ako znázorňuje daná priamka, vyzerá nasledujúco:

$$4.4 * \frac{countWords}{100} + 25 \tag{3.2}$$

V tabuľke nižšie sú zobrazené jednotlivé sekvencie, ktoré môžu nastať a k ním dané ohodnotenia. Rozlišujeme klasifikovanie pred fázou 4 a po fáze 4.

| Odmeraný čas prvku x | Klasifikovanie |
|---|---|
| x <= 10sec | Krátky čas(ST) |
| x > 180sec | Dlhý čas(LT) |
| x > expectedReadTime | ExptectedShortTime(E_ST) |
| x >= expectedReadTime | ExpectedLongTime(E_LT) |

**Tabuľka 3.1.** Klasifikovanie odmeraného prvku x

V Tabuľke 3.1 môžeme vidieť počiatočnú klasifikáciu prvku po odmeraní času. Hodnota **expectedReadTime** je získana zo vzorca (3.2). Prvky označené ako **E_ST** alebo **E_LT** sa ďalej spracúvajú, kým nebude vyhodnotené, že sú ST, LT alebo UK. Trieda UK bude klasifikovaná pomocou **K-NN algoritmu** vo fáze 3.

| Sekvencia | Zistené x | Klasifikovanie |
|---|---|---|
| E_ST, E_ST, ..., E_ST | ST | Všetko ST |
| E_ST, ..., E_LT, ..., E_LT | ST | x $\rightarrow ST, Zvyšok(UK)$ |
| E_ST, ..., E_ST, E_LT | ST | x $\rightarrow ST, E\_LT \rightarrow LT,$ všetky E_ST $\rightarrow ST$ |
| E_ST, ..., E_ST | LT | x $\rightarrow LT, E\_ST \rightarrow ST$ |
| E_ST, ..., E_LT, ..., E_ST | LT | x $\rightarrow LT,$ do najbližšieho E_LT E_ST $\rightarrow ST,$ zvyšok UK |
| ..., E_LT | LT | x $\rightarrow LT, zvyšokUK$ |
| E_ST, E_LT | E_LT/Different | x $\rightarrow LT, zvyšokUK$ |
| E_LT, E_ST, E_LT | E_ST | E_LT $\rightarrow UK, E\_ST \rightarrow ST,$ E_LT $\rightarrow LT, x \rightarrow čakaj$ |
| E_LT, E_ST | E_LT/Different | E_LT $\rightarrow UK, E\_ST \rightarrow ST,$ x $\rightarrow LT$ |
| E_ST, ..., E_ST | E_LT/Different | všetky E_ST $\rightarrow STax \rightarrow LT$ |
| E_ST, ..., E_ST | E_LT | Čakaj |
| E_ST, E_LT | E_LT | Čakaj |
| E_LT, E_LT | E_LT/Different | x $\rightarrow LT, zvyšok(UK)$ |
| E_LT, ..., E_LT | E_LT | Čakaj |
| E_LT, E_ST | E_ST | Čakaj |
| E_ST, ..., E_ST | E_ST | Čakaj |

**Tabuľka 3.2.** Klasifikovanie odmeraných časov pred fázov 4.

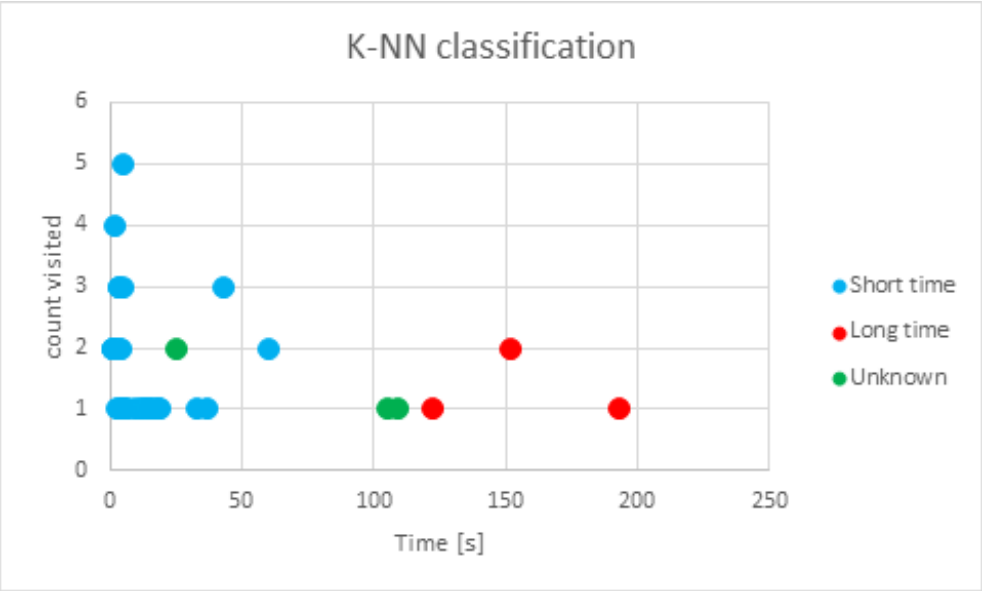
V tabuľke 3.2 môžeme vidieť ohodnocovanie sekvencií. Klasifikovanie ako čakaj znamená, že sekvencia pokračuje ďalej, kým algoritmus nezistí, do akých tried patria prvky, alebo, kým sa užívateľ **neodhlási** a tým sa všetko klasifikuje ako UK. E_LT/Different znamená, že podľa vzorca (3.1) sa číslo výraznejšie odlišuje od ostatných.

### 3.1.3 Fáza 3: K-NN klasifikácia

Pri tejto fáze dochádza k ohodnocovaniu zvyšných neklasifikovaných časov. V tomto prípade sa použije jednoduchý algoritmus K-NN, ktorý bude klasifikovať data na základe najbližších susedov[11]. K tomuto klasifikovaniu dôjde až pri vyvolaní zmeny meta-modelu. Algoritmus bude využívať euklidovskú metriku. Obsahujúc počet navštívení a stravený čas na prvku.

Pri **K-NN** algoritme ostáva otázkou, ako správne nastaviť hodnotu **k**, aby výsledky, ktoré boli klasifikované mali čo najmenšiu chybu. Tu použijeme hodnotu **k = 3**. Dôvodom je, že tento algoritmus je viac striktnejší k ohodnoteniu dlhého času a viac benevolentnejší ku krátkemu. Znamená to, že pri vyšších časov, ale ešte neklasifikovaných ako dlhý, budeme dostávať ohodnotenie Unknown. Bude počet dlhých časov nižší. Pri väčšom **k** by algoritmus mohol začať väčšinu pravdepodobných dlhých časov ohodnocovať ako krátkych a tým by sa zvýšila chybovosť algoritmu výrazne.

Tento algoritmus sa bude využívať vždy na doklasifikovanie časov, ktoré neboli klasifikované buď prvotným statickým rozpoznávaním alebo sa časy nachádzali v medzi viď. **Fáza 4**.




**Obrázok 3.6.** K-NN klasifikácia

### 3.1.4 Fáza 4: Počítanie limitov

V poslednej fáze dôjde k vypočítaniu horného limitu krátkeho času(Upper short time limit) a k dolnému limitu dlhého (Low long time limit). Horný limit značí, do kedy budeme klasifikovať, že užívateľ sa naviguje po menu. A dolný limit značí kedy si užívateľ už prezerá stránku. Podľa týchto limitov sa budú ďalej klasifikovať časy. Okrem toho v tomto prípade môže pribudnúť medzera medzi týmito limitmi, ktorú nazývame medz(bound), kde za istých podmienok môžeme označiť čas ako dlhý. Prvok musí byť aktívnym a sekvencia nasledujúca musí mať kratší čas(výraznejšie).

Podobne ako u fázy 3 sa využije K-NN algoritmus k ohodnoteniu neklasifikovaných časov. K výpočtom dôjde až pri iniciovanej zmene meta-modelu alebo dostatku dát popisané nižšie v sekcii zmeny časov 3.1.6.

Po prebehnutí tejto fázy sa časy budú klasifikovať na základe vypočítaných limitov, ako ukazuje tabuľka 3.3.

| Odmeraný čas prvku x | Klasifikovanie |
|---|---|
| x <= Upper limit | Krátky čas(ST) |
| x >= Low limit | Dlhý čas(LT) |
| Upper limit < x < Low limit | Unknown(UK) |

**Tabuľka 3.3.** Klasifikovanie odmeraných časov po fáze 4

### 3.1.5 Skúsený užívateľ a aktívny prvok(Obľúbený)

Pri bežných webových aplikáciách sa môžeme stretnúť s dvomi typmi užívateľov a to začiatočníkom(novice) a skúseným(expert). Pri návrhu je stále potrebné brať ohľad na začiatočníkov, pretože každý užívateľ nim raz bol, ale z dôvodu aby údaje zistené boli bližšie k pravde sú v niektorých bodoch dané výnimky, ktoré počítajú so skúsenými užívateľmi[8]. Podľa zistenia, že ide o aktívny prvok posudzujeme či sú začiatočníkmi alebo skúsenými. Aktívne prvky sú zvlášť pre užívateľa dôležité, pretože na nich trávi viac času, ako na zvyšných, alebo sa častokrát k ním vracia. Za aktívny prvok budeme považovať:

- Prvok, ktorý užívateľ navštívi za 2 týždne aspoň 4-krát(1 deň = 1 navštívenie)
- Prvok je aktívnym po dobu dvoch mesiacov, ak ho navštívi v tom čase znovu, doba začína plynúť od posledného navštívenia
- Prvok, ktorý bol v minulosti aktívnym stačí znova aktivovať 3 návštevami za deň(myslí sa, že bude prvok klasifikovaný ako dlhý čas) alebo 2-krát za posledný týždeň.

Za skúseného užívateľa budeme považovať:

- Užívateľa, ktorý má aspoň jeden prvok aktívny
- Jeho navigovanie po menu je častokrát rýchlejšie, ako hodnoty nastavené pre novoregistrovaných (tieto hodnoty sa vypočítavajú cez všetkých užívateľov patriacich do rovnakej vekovej kategórie). Tento bod vychádza z paradoxu aktívnych užívateľov[3].

### 3.1.6 Zmeny časov

V aplikácii bude dochádzať k dynamickej zmene limitov ako vyššie popisuje **fáza 4**. Teraz ostáva otázkou, ako a kedy meniť tieto časy. Predtým, ako popíšeme zmenu časov, je potrené zistiť, o aké pravdepodobnostné rozdelenie pri našich datach ide. Pri krátkom čase predpokladáme, že ide o **exponenciálne rozdelenie**, pretože:

- Máme **nezáporné hodnoty**, pretože čas odmeraný na prvku nemôže byť záporný,
- Vyjadruje čas medzi náhodne sa vyskytujúcimi udalosťami,
- Máme **spojitú náhodnú veličinu**,
- Početnosť dát bude vyššia pri nižších časoch ako pri väčších, pretože máme dáta rozdelené do dvoch skupín a limity sa dynamicky menia,
- Pre rozhodovanie sme si pomohli **histogramom** vytvoreným z testovacích dát,
- Z testovacích údajov sme vykreslili **q-q plot**, čo je metóda na posúdenie, či dáta pochádzajú z nejakého známeho rozdelenia

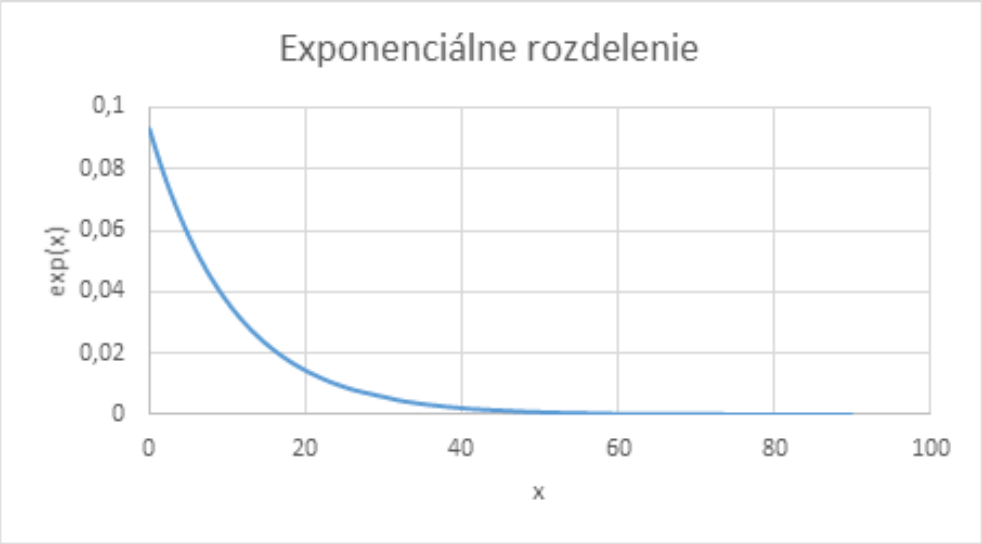
Exponenciálne rozdelenie má nasledujúcu hustotu pravdepodobnosti[1],

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases} \tag{3.3}$$

s parametrom $\lambda > 0$, ktorý bude potrebné odhadnúť pomocou **MLE**, z ktorej dostáneme, že

$$\lambda = \frac{1}{mean}$$

Po zistení parametra, budeme vypočítavať upper limit z dát, a to pomocou smerodajnej odchýlky, kde k strednej hodnote pripočítame **2 sigma**, čo je približne **95%**. Pri exponenciálnom rozdelení to vychádza na rozsah $[0; 3/\lambda]$[7].




**Obrázok 3.7.** Príklad exponenciálneho rozdelenia z testovacích dát

Pri dlhom čase predpokladáme **pravdepodobnostné rozdelenie gamma**:

- Máme **nezáporné hodnoty**, pretože čas odmeraný na prvku nemôže byť záporný,
- Máme **spojitú náhodnú veličinu**,
- Podobne ako pri krátkom čase sme si pomohli **histogramom**
- Rovnako ako pri krátkom čase sme z testovacích údajov vykreslili **q-q plot**

Gamma rozdelenie má hustotu pravdepodobnosti[2]

$$f_X(x) = \begin{cases} \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} & x > 0, \\ 0 & x \leq 0 \end{cases} \tag{3.4}$$

Kde,
$\alpha, \beta$ - sú parametre,
$\Gamma(\alpha)$ - označuje gamma funkciu,
$x$ - je náhodná veličina.

Ďalej vypočítame parametre:

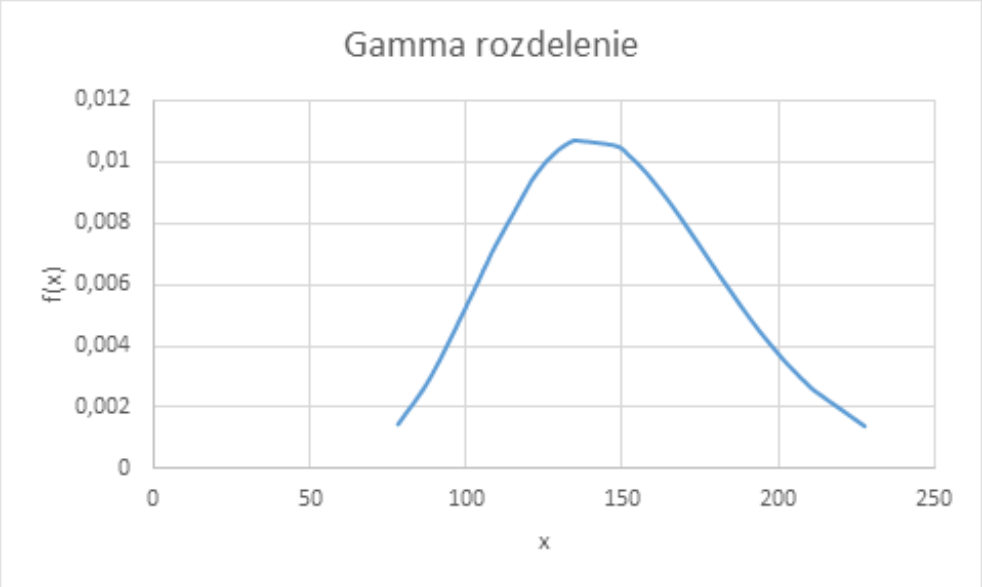
$$\alpha = \frac{E^2(X)}{D(X)} \tag{3.5}$$

$$\beta = \frac{E(X)}{D(X)} \tag{3.6}$$

Kde,
$E(X)$ je stredná hodnota,
$D(X)$ je rozptyl.

Nakoniec vypočítame dolní limit(short long time limit).

$$L = \frac{\alpha}{\beta} - \sqrt{\frac{\alpha}{\beta^2}} \tag{3.7}$$




**Obrázok 3.8.** Príklad gamma rozdelenia z testovacích dát

Histogramy a q-q plots, ktoré boli vykreslené z testovacích dát sú zobrazené v kapitole 5.2 Pravdepodobnostné rozdelenia.

Poslednou nezodpovedanou otázkou zostáva, kedy vykonávať zmeny limitov. Tie sa budú vykonávať **pri zmene štruktúry/modelu**, ktorý bude iniciovaný ďalšími pravidlami. Okrem toho môže dôjsť k prepočítaniu limitov vtedy, ak máme:

- aspoň 3 krátke a dlhé časy(dôvodom je hodnoa $k$ v K-NN algoritme)
- aspoň 30 nových dát. Dôvodom je, žeby limity boli pravidelne aktuálne.

### 3.1.7 Výhody a nevýhody

#### Výhody

- Zohľadňuje skúsených a menej skúsených užívateľov.
- Dynamické prispôsobovanie časov ku konkrétnym užívateľom a tým napomáha k lepšej účinnosti pri klasifikácii dlhého a krátkeho času.
- Pomáha nám zistiť, kedy užívateľ skončil danú úlohu(rola užívateľa je prezeranie/dlhý čas).
- Prihliada na rôzne vekové kategórie a vyhodnocuje v rámci nich.

#### Nevýhody

- Nepresné klasifikovanie skončenia úlohy(rola užívateľa je prezeranie/dlhý čas).
- Nemáme spätnú väzbu od užívateľa a preto nevieme či mal užívateľ o daný prvok záujem.
- Nevhodne nastavená hodnota **k** parametru pri K-NN algoritme a tým vyššia chybovosť.
- Statické klasifikovanie vykonávané len na základe slov. Nemožné použiť na všetky stránky napríklad galérie(resp. veľké množstvo hodnôt bude klasifikované pomocou K-NN).

## 3.2 Radenie prvkov v uzle

V kapitole 2.3 sú rozobrané IA metódy, ktoré riešia spôsob usporiadania prvkov, navigovateľnosť a pod. Tieto metódy sa snažia radiť prvky **podľa očakávania** užívateľa. Okrem nich sa môžeme stretnúť s ďalšími typmi radení:

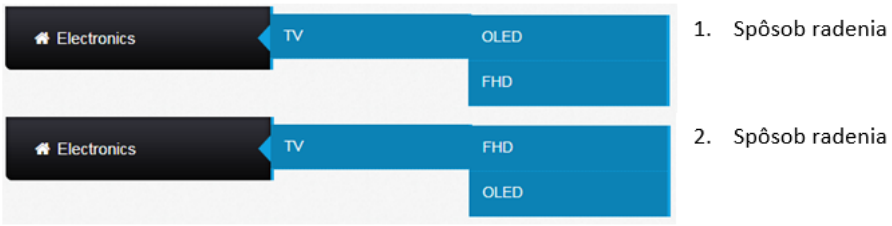
- **Konvenčné:** radenie čísel, veľkostí, dátumy atď.,
- **Frekvencií používania:** podľa použitia jednotlivých prvkov na základe klikania
- **Kategorický:** podľa sémantických vlastnosti daných prvkov(podobnosti)
- **Abecedne**

Konvenčné a abecedné radenia sú bežné a dajú sa ľahko zrealizovať. Pri kategorickom radení je to zložitejšie, pretože potrebujeme porovnávať významy slov. To budeme uskutočňovať hľadaním synonym, hyponym a hyperonym. Frekvenciu vieme zisťovať z počtu navštívenia daného uzla.

Pri adaptívnych štruktúrach vieme vyššie spomínané spôsoby radenia zrealizovať relatívne jednoducho, ale problém nastáva s radením podľa očakávania. Ako vieme zistiť, čo je pre užívateľov najvhodnejšie, ak nám to sami nepovedia? V tomto prípade nám nepomôže Card sorting ani Tree testing, pretože vyžadujú spôsob statickej štruktúry a následne tu štruktúru otestovať na nejakej vzorke. Preto je potrebné navrhnúť spôsob, ako zisťovať, čo je pre užívateľov najvhodnejšie.

Metóda, ktorá bude použitá pre adaptívne štruktúry vychádza z vyššie spomenutých informácii. Na to, aby sa zistilo, ako mať zoradené prvky v danom uzle, sa bude náhodne vybraným užívateľom zobrazovať rôzne usporiadania prvkov(slabo pripomínajúci reverzní closed sorting). Budú sa vyhodnocovať niektoré parametre z Tree testing, viď. popis algoritmu nižšie. Takýto spôsob pripomína **A/B testovanie**, kde na istej vzorke užívateľov aplikujeme jednotlivé typy poradí a pri

zvyšných použijeme defaultne poradie. Príklad viac spôsobov radení zobrazuje obrázok 3.9. Po otestovaní vyhodnotíme výsledky a aplikujeme iniciované zmeny.

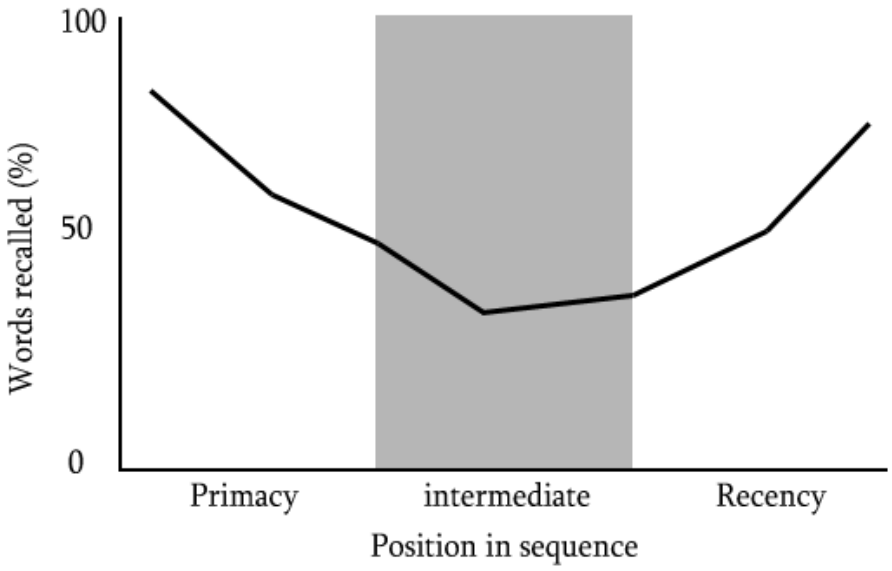



**Obrázok 3.9.** Príklad dvoch možnosti rozvrhnutia.

### 3.2.1 Popis algoritmu

Na začiatku sa prvky zoradia podľa vyššie spomenutých 4 radení, pričom priorita jednotlivých prvkov typov je nasledujúca:

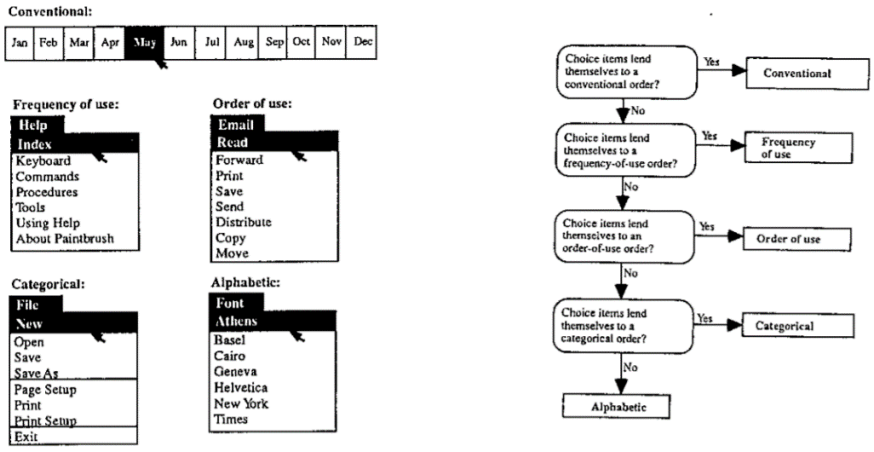
1. Konvenčné(má najvyššiu prioritu),
2. Frekvencia používania (podľa počtu navštívení aktívnych prvkov),
3. Podľa kategórii(synonyma, hyperonym, hyponym),
4. Abecedne




**Obrázok 3.10.** Krivka zobrazujúca serial position effect[14].

Pri radení sa využíva **serial position effect**, čo je ľudská tendencia, kde si človek najprv všíma prvky na začiatku a na konci sekvencie a prvky v strede si veľmi nevšíma[14]. Tento spôsob radenia je trochu poznemnení, pretože je potrebné držať prvky istých radení spolu. Kategoricky zoradené prvky sa neoddelia, budú stále pospolu. Taktiež ani konvenčné radenie nerozdelí svoje prvky, iba radenia podľa frekvencií a abecedne budú využívať daného efektu. Ak by kategorické a

konvenčné radenie využívali efektu, stratil by sa zmysel daného radenia. Tento spôsob radenia je určený pre bežných užívateľov. Pri nich si len zaznamenávame údaje a informácie, ktoré neskôr použijeme pre porovnávanie.




**Obrázok 3.11.** Príklad jednotlivých typov radení a ich postupne ohodnocovanie[13].

Pre n-vybraných užívateľov sa prvky v uzloch roztriedia náhodne s príslušnými obmedzeniami:

- Prvky často používané sa budú v uzle zobrazovať čo najvyššie(s dôrazom na serial position effect),
- Najviac podobné prvky sa budú držať pri sebe.

Zvyšne prvky sa roztriedia náhodne. Náhodnosť sa myslí postupnosť pri permutácii daného uzla. Z dôvodu permutácie, ale aj šírky daného uzla v stromovej štruktúre je počet v jednej úrovni stromu obmedzení na **5 prvkov**. V najhoršom prípade pri permutácii by sme mali dostať 5!, čo je 120 možností.

Od jednotlivých užívateľov sa budú zaznamenávať informácie:

- Čas potrebný na vykonanie úlohy
- Našiel daný prvok priamočiaro?
- Zaznamenávať cestu
- Počet navštívení prvku

Na základe týchto údajov bude prebiehať rozhodovanie, ktorý spôsob radenia je vhodnejší. Pravidla, ktoré budeme využívať pre rozhodovanie sú:

- **Rýchlosť vykonania úlohy**
- **Priamočiarosť(Directness)**

Prvý spôsob popisuje, ako rýchlo sa vie užívateľ dostať k informácii, ktorá sa nachádza na konkrétnom prvku v menu. Toto rozhodovanie je výhodnejšie až pre

skúsenejších užívateľov, pretože už poznajú dané menu/štruktúru a vedia sa v ňom pohybovať.

Priamočiarosťou sa myslí, že sa užívateľ z koreňa stromu dostal do uzla, kde skončil svoju úlohu, pričom pri ceste nenavštívil viac uzlov ako je hĺbka uzla, v ktorom skončil. Táto metóda je viac vhodnejšia pre nováčikov, ktorý netušia, kde čo môžu nájsť a len sa chaoticky preklikávajú.

Tabuľka 3.4 slúži na porovnanie, ktorý spôsob je lepší od druhého. Zachytáva obe typy rozhodovania rýchlosť a priamočiarosť.

**Poznámka:** $1 > 2$ znamená, že prvý spôsob rozvrhnutia je lepší než druhý.

| Návštevnosť | Čas[sec]/Hodnota priamočiarosti | Rozhodnutie |
|---|---|---|
| $1 > 2$ | $1 > 2$ | 1 |
| $2 > 1$ | $1 > 2$ | 1 |
| $1 > 2$ | $2 > 1$ | 2 |
| $2 > 1$ | $2 > 1$ | 2 |
| -(tolerancia) | $1 > 2$ | 1 |
| -(tolerancia) | $2 > 1$ | 2 |
| $1 > 2$ | -(tolerancia) | Nevie sa |
| $2 > 1$ | -(tolerancia) | Nevie sa |
| -(tolerancia) | -(tolerancia) | Nevie sa |

**Tabuľka 3.4.** Rozhodovanie, ktoré poradie je lepšie podľa času/priamočiarosti a návštevnosti

### 3.2.2 Rýchlosť vykonania úlohy

Rýchlosť vykonania úlohy sa bude vyhodnocovať nasledujúco:

1. Zistíme priemer časov užívateľa a užívateľov pre jeden spôsob radenia.

$$\frac{1}{K}\sum_{i=1}^{K}\frac{1}{N_i}\left(\sum_{j=1}^{N_i}t_{ij}\right) \tag{3.8}$$

   Kde,
   $K$ - je počet užívateľov pre daný spôsob radenia,
   $N_i, i = 1, .., \infty$ - je návštevnosť jedného užívateľa,
   $t_{ij}, i = j = 1, .., \infty$ - je čas potrebný na vykonanie úlohy
2. Ďalej zistíme priemernú návštevnosť daného prvku za užívateľov.

$$\frac{1}{K}\sum_{i=1}^{K}N_i \tag{3.9}$$

3. Porovnáme jednotlivé prvky medzi sebou podľa vyššie spomínanej tabuľky 3.4
4. Porovnáme radenia medzi sebou, kde pre každý spôsob priradíme -1, 0, 1 k porovnaniam a urobíme súčet. Najlepšie rozvrhnutie prvkov bude mať najväčšie číslo. Ak by bolo viac takých spôsobov s rovnakým najväčším číslom, potom porovnáme hodnoty jednotlivých radení s hodnotami užívateľa a zistíme, ktoré mu bude najviac vyhovovať, inak sa vyberie prvé.

### 3.2.3 Priamočiarosť (Directness)

Pri priamočiarosti potrebuje zachytávať celú cestu z posledného ohodnoteného uzla ako dlhý až po ďalší najbližší čas ohodnotený ako dlhý. Následne zistiť, aký je najmenší počet prekliknutí potrebný a porovnať so súčasným zisteným počtom.

$$\sum_{i=0}^{N} \frac{expectedCountClicks}{observedCountClicks} \tag{3.10}$$

Kde,
$N$ - je počet všetkých odpozorovaných sekvencií

Pomocou vzorca (3.10) zisťujeme **hodnotu priamočiarosti**. Tú využijeme pri porovnávani obdobne ako rýchlosť. Taktiež namiesto času budeme porovnávať priamočiarosť. Rozhodovanie je znázornené v tabuľke 3.4

### 3.2.4 Kedy vykonať zmeny?

Aplikovanie zmien na užívateľské rozhranie je veľmi krehká situácia a mala by sa tomu venovať dostatočná pozornosť, pretože nejaké chyby alebo zmeny môžu rozhodnúť, že sa užívateľ viackrát nevráti na stránku. Tým nastane opačný efekt, o ktorý sa snažíme. Preto je vhodné si premyslieť rôzne prípady, kedy aplikovať zmeny, ktoré iniciuje algoritmus. Po rozbore ohľadom aktualizácie užívateľského modelu sa môžeme stretnúť s nasledujúcimi prípadmi, ktoré popisujú zároveň vykonanie zmeny:

1. **Neaktívny užívateľ** - zmenu môžeme vykonať hneď
2. **Aktívny užívateľ** a uzol, ktorý **nenavštívil** za poslednú dobu/vôbec - zmenu môžeme vykonať hneď
3. **Aktívny užívateľ** a uzol, ktorý **navštevuje pravidelne** - porovnáme hodnoty užívateľa s hodnotami, ktoré podnecuje algoritmus a ak sa hodnoty nebudú nachádzať v tolerancii, vykonáme zmenu.

Bod prvý opisuje prípad, kedy užívateľ sa už dlhšie neprihlásil a tak všetky zmeny, ktoré chceme aplikovať, môžeme vykonať, pretože užívateľ si dané menu veľmi nebude pamätať.

V druhom bode popisujeme problém, kedy užívateľ prichádza na stránku pravidelne, ale iniciované zmeny nemajú takmer žiadny dopad a nebudú spôsobovať zmätok alebo frustráciu.

Posledný bod je najkomplikovanejším spôsobom. Chceme vykonávať zmeny v štruktúre, ktorá by veľmi ovplyvnila používateľnosť a mohla by pôsobiť mätúco kvôli navyknutosti. Preto v tomto prípade budeme porovnávať hodnoty uzla, ktoré chceme meniť z hodnotami, ktoré sme získali testovaním. Ak zistené zmeny sú v tolerancii, tak nedôjde k ich aplikovaniu. V opačnom prípade zmeníme štruktúru.

### 3.2.5 Aká veľká má byť množina testovacích subjektov?

5? 10? alebo 100? Koľko užívateľov použiť na kvantitatívne otestovanie daného uzla, aby sme mali dostatočnú vzorku a pritom i dostatočný počet dát? Je to ďalšia otázka, ktorá nemá jednoznačnú odpoveď. Podľa Nielsena je dobrým počtom na otestovanie aspoň **20 užívateľov**[12]. Túto hodnotu využijeme aj pre naše testovanie. Užívatelia sa rozvrhujú do skupín **pomerom 1:10**. To znamená, že jeden z 10 užívateľov bude testovacím subjektom. Ak nastane situácia, že všetky uzly boli otestované, potom už žiadny užívateľ nebude testovacím, a počká sa na najbližší retest.

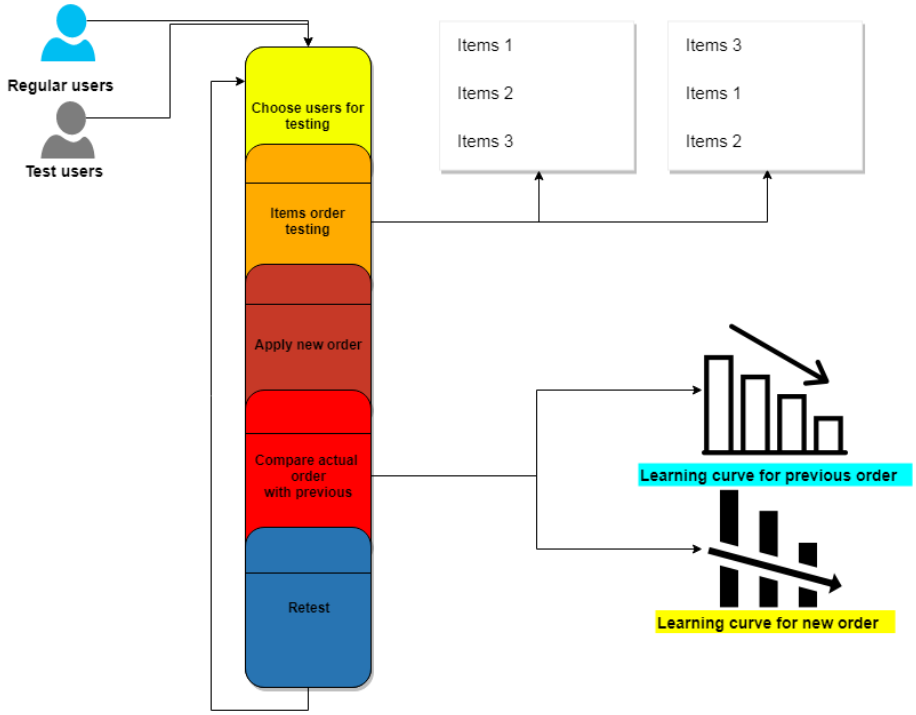
### 3.2.6 Ako dlho nechať užívateľov učiť sa?

Príznačnou otázkou ostáva, kedy ukončiť testovanie na množine testovacích subjektov, a vzápätí začať s výpočtami, porovnávaním a aktualizáciami. Problémom je, že nevieme presne posúdiť, či máme dostatok dát, z ktorých vieme vyzistiť správanie. Kvôli rozhodnutiu používať hodnotu $k = 3$ v K-NN algoritme sa vyžaduje pre uskutočnenie zmeny aspoň:

- 3 zistené krátke a dlhé časy
- 25 dát odmeraných na užívateľovi

Drubý bod 25 odmeraných dát odpovedá piatim úloham viď. kapitola 5 testovanie. Tieto požiadavky sa budú vyžadovať pri prvom vyhodnocovaní. Ďalšie vyhodnocovania budú závisieť na dĺžke predchádzajúcej zmeny. Napr. ak došlo k prvej zmene od registrácie po dvoch týždňoch, ďalšie porovnávanie bude až za 2 týždne. Tento proces zobrazuje obrázok 3.12, kde na začiatku dôjde k rozdeleniu užívateľov. Následne tým, ktorí sú označený ako testovací vygenereje

poradie. Ďalej dôjde k aplikovaniu novej zmeny. Po aplikovaní novej zmeny sa počká rovnakú dobu ako bola aplikovaná predchadzajúca zmena a tak porovná. Ak vyhodnotí, že daná zmena nebola vhodná, dôjde k retestu, a cyklus sa opakuje.




**Obrázok 3.12.** Proces radenia prvkov

### 3.2.7 Výhody a nevýhody

#### Výhody

- Zisťuje poradie prvkov na užívateľov podľa ich očakávania
- Vyhodnocuje novú zmenu s predošlou a rozhoduje či bola vhodná
- Prihliada na nováčikov a skusených užívateľov

#### Nevýhody

- Málo dát pre vykonanie testovania
- Dlhšie učenie
- Zložitejší algoritmus
- Náhodna vzorka užívateľov pre otestovanie poradia.

## 3.3 Kooperácia viacerých menu

Pri prechádzaní rôznymi stránkami si môžeme všimnúť, že uživateľské rozhranie obsahuje viaceré menu. Napríklad pri stránkach škôl môžeme vidieť, že v ho-

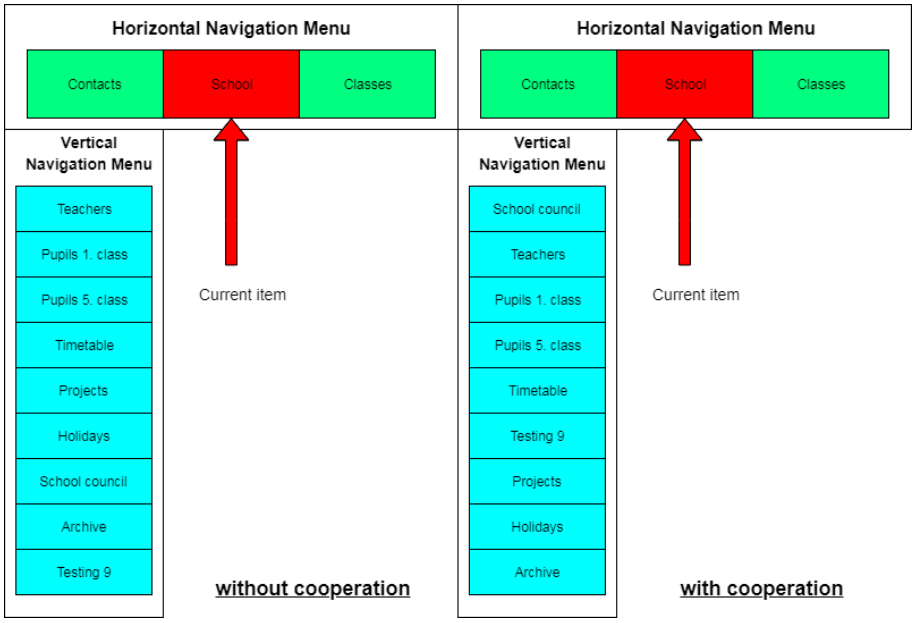
rizontálnom menu obsahujú informácie o škole, kontaktoch, tried atď. Vo vertikálnom menu sa môžu nachádzať rôzne položky, ktoré nespadajú priamo do kategórie z vyššie spomínaných. Menu by mohlo byť príliš hlboké, ak by sme prvok chceli včleniť do jednej z kategórii ako napríklad archív. Na tomto príklade môžeme vidieť vhodnosť použitia ďalšieho menu.

Tieto menu si však žijú svojím vlastným životom a nevedia vzájomne o sebe. Pričom, Ak korelujú spolu, mohli by spolupracovať a vymieňať si informácie. Tento problém by malo riešiť pravidlo kooperácie. Kooperácia bude riešiť dva problémy:

- **Poradie prvkov** - závisi na pozíci druhého menu
- **Presun prvkov** - presúvanie prvkov z jedného menu do druhého

### 3.3.1 Aktualizácia poradia jedného menu podľa druhého

Na obrázku 3.13 môžeme vidieť stav pred a po kooperácii. V stave s jej použitím sa prvky zoradili podľa aktuálnej pozície v horizontálnom menu. Takou zmenou chceme vylepšiť navigovanie po vertikálnom menu, a to tak, že poradie prvkov bude významovo bližšie ku kontextu aktuálneho prvku.




**Obrázok 3.13.** Znázornenie poradia prvkov pred a po kooperácii

Algoritmus, ktorý rieši poradie vychádza z nasledujúcich informácii:

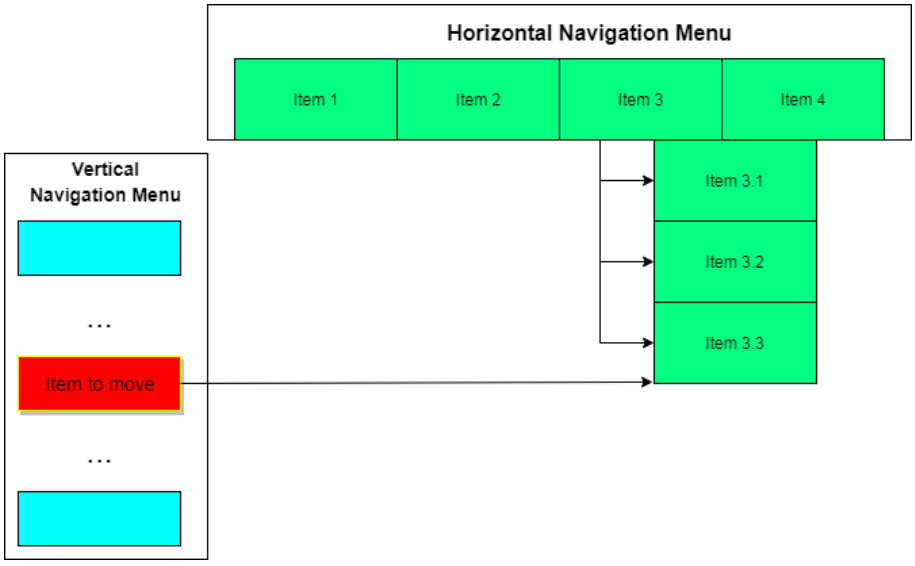
- Synonama
- Hyponyma a hyperonyma
- Frekvencii prechodov

**Synonyma** využijeme pre podobnosť daných výrazov. Ak ide o nejaké súvetie porovnajú, sa jednotlivé slová medzi sebou. Zhoda v porvnaní znamená významový súvis prvkov. Hľadanie podoby s **hyponymami a hyperonymami** nám pomôže zoradiť zvyšné prvky po porvnaní synonym. Ďalej sa prvky zoradia podľa **frekvencii** klikania na vertikálnom menu a zvyšné prvky budú mať poradie ako na vstupe(nie pozíciu), teda algoritmus bude stabilný.

Vertikálne, resp. druhé menu môže obsahovať viac prvkov v uzle ako hlavné menu, ktoré bolo obmedzené na 5. Dôvodom je, že ďalšie menu nebýva tak sofistikované ako hlavné a len ho rozširuje resp. zjednodušuje. Poradie prvkov bude brať do úvahy **serial position effect**[14].

### 3.3.2 Presúvanie prvkov medzi viacerými menu

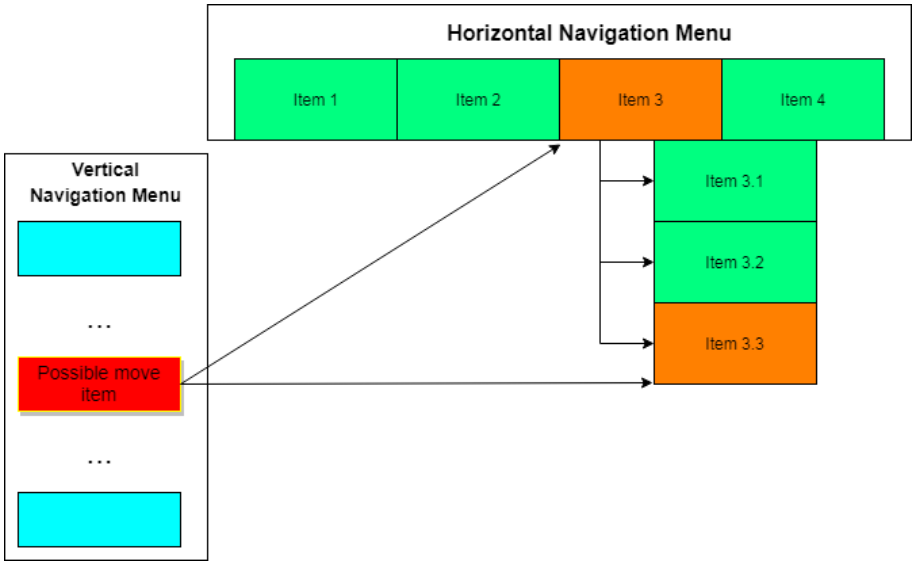
Pri tomto probléme sa snažíme zisťovať súvis prvkov medzi dvomi menu. Zisťujeme to pomocou frekvencii prechodov. V predošlej sekcii sme zisťovali prechod z hlavného na prvok do vedľajšieho menu. To budeme využívať aj v tomto algoritme, ale pridáme aj prechod opačný z vedľajšieho do hlavného. Plus pridáme porovnávanie významov, ktorý je obdobný taktiež predošlej sekcii. Budeme zisťovať synoma, hyponyma a hyperonyma. Z týchto všetkých informácii rozhodneme o presune prvku.




**Obrázok 3.14.** Presun prvku z jedného menu do druhého

Obrázok 3.14 znároňuje zmenu pozície prvku. Zmenu bude iniciovať frekvencia prechodov. Bude brať do úvahy prechod dvoch konktrétnych prvkov a ešte jeho parenta. To zachytáva obrázok 3.15. Ak dôjde k zhode, prvok sa presunie, v opačnom prípade nezmení pozíciu. Prvok bude presunutý pod prvok, s ktorým inicioval zmenu. Ak počet prvkov jeho následovníkov bude aspoň 5

alebo hĺbka stromu bude aspoň 3, skúsi prvok vložiť do rovnákej úrovne. Ak aj tu je počet prvkov aspoň 5, tak poruší pravidlo a pridá ďalší prvok pod prvok, s ktorým inicioval zmenu. Teda nie do rovnákej úrovňe. Počet prvkov, ale nesmie byť viac ako 7. Ak by nastal takýto stav. Potom nedôjde k zmene.




**Obrázok 3.15.** Porovnánie potenciálneho prvka pre presun

### 3.3.3 Výhody a nevýhody

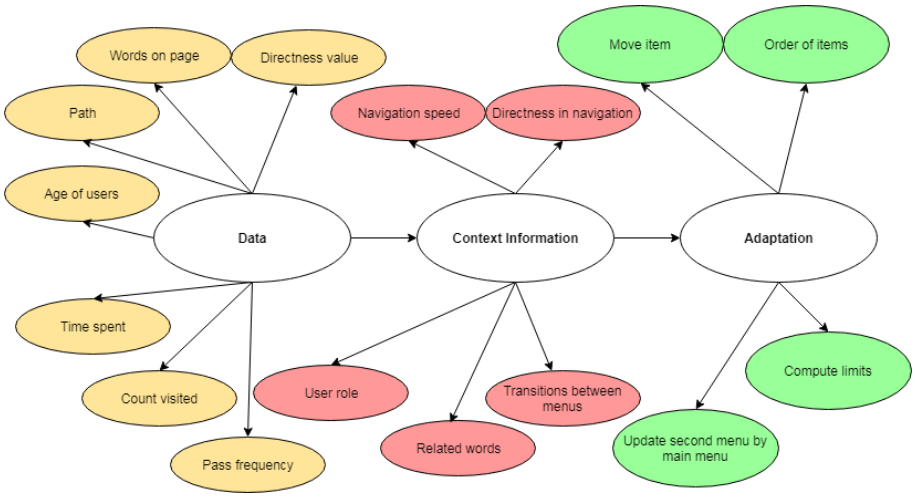
#### Výhody

- Zoradením prvkov sa snažíme pomôcť k rýchlejšiemu navigovaniu.
- Prehodením prvku dosahujeme jeho lepšiu naleziteľnosť v menu, pretože zo získaných informácii posudíme ich podobnosť.
- Vykonané zmeny nespôsobujú veľkú zmenu na GUI. Buď je zmena v rámci uzlu alebo dôjde k prehodeniu jedného prvku. Nemeníme teda celý model, alebo väčšinu, ako to býva v adaptívných štruktúrach na začiatku, kedy prvky fluktujú v stromovej štruktúre[17].

#### Nevýhody

- V prípade prehodenia prvku dlhšie učenie.
- Vyžadovaná korelácia pri aktualizácii poradia.

## 3.4 Zhrnutie




**Obrázok 3.16.** Metodológia(Data, kontextové informácie, adaptácia)

Na obrázku 3.16 môžeme vidieť tri časti, z ktorých pozostáva adaptácia. Podobný obrázok môžeme vidieť v rešerši pri návrhu adaptívneho rozhrania 2.2. Tento obrázok bol rozšírený o informácie, ktoré využívajú algoritmy alebo k akej adaptácii dochádza. V prvej časti vidíme dáta. Pri nich získavame radu hodnôt, ktoré sú následne potrebné pri kontextových informáciach a nakoniec dochádza k niekoľkým adaptáciam. Obrázok predstavuje zhrnutie všetkých informácii, s ktorými pracujú algoritmi.

# Kapitola 4
# Implementácia

## 4.1 Definície, frameworky a technológie

**Spring boot** je java framework, ktorý umožňuje ľahko vytvoriť stand-alone aplikáciu. Konfigurácia je jednoduchá, kde stačí použiť súbor `.properties` alebo `.yaml` pre nastavenie.

**ORM** ide o spôsob zaistenia konverzie dát medzi objektom v objektovom orientovanom programovacom jazyku a relačnou databázou.

**JPA** predstavuje rozhranie pre ORM. Zjednodušuje ukládanie entít do databázy a ich vyťahovanie.

**Hibernate** ide o konkretnú implementáciu rozhrania JPA[24].

**Spring Data JPA** je súčasťou rodiny Spring Data. Umožňuje ľahšiu implementáciu JPA založených na repositories. Pri jej využití nám už ponúka základnú sadu metod pre prácu s vrstvou repository. Napríklad save, findById atď. Umóžňuje ľahko vytvoriť query pomocou názvu metódy. Pre zložiteľšie query ponúka anotáciu @query. Taktiež má typovo bezpečné JPA query[23].

**Gradle** nástroj pre zostavovanie programu.

**Oracle database** databazový systém s pokročilými možnosťami spracovania dát. Je vysoko výkonný a ľahko škálovateľný.

**AJAX** (Asynchronous JavaScript and XML) označenie pre technológie, kroré menia obsah svojich stránok bez načítania celej stránky. To vykonáva pomocou asynchronného spracovania stránok[20].

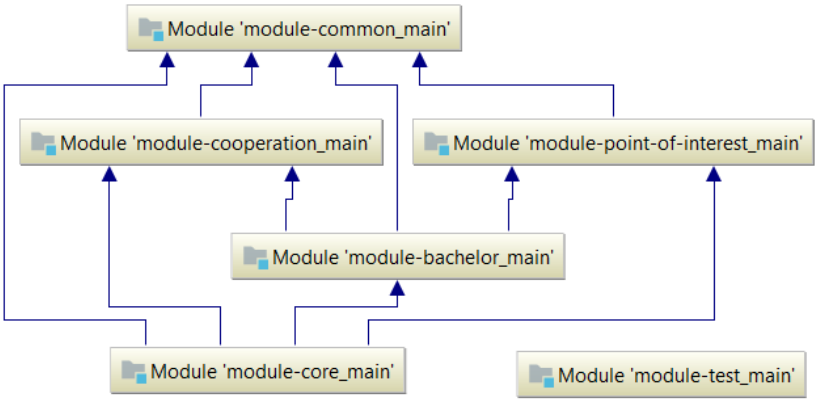
**REST** je architektúra rozhrania, určená pre distribuované prostredie orientované na data.

**Groovy** je objektovo orientovaný programovací jazyk, určený pre platformu Java. Umožňuje spúšťať Java súbory. Má jednoduchšiu syntax ako Java. V našej aplikácii ho využívame na unit testy.

**Spock Framework** ide o testovací framework, ktorý je inšpirovaný ďalšími testovacími frameworkmi, ako JUnit alebo jMock. Jeho veľkou prednosťou je, že testy písane v ňom sú ľahko čitateľné. Oddeľuje časť prípravy dát, vykonávania a s následným assertovaným[22].

**Liquibase** je databázová nezávislá knižnica pre správu, sledovanie a uplatňovanie zmien schém databázy[21].
**extjwnl** je knižnica pre čítanie a editovanie slovníkov vo WordNet formáte[25].

## 4.2 Popis frameworku




**Obrázok 4.1.** Moduly frameworku

Do už existujúceho frameworku[17] boli pridané nové moduly:

- module-common
- module-cooperation
- module-point-of-interest

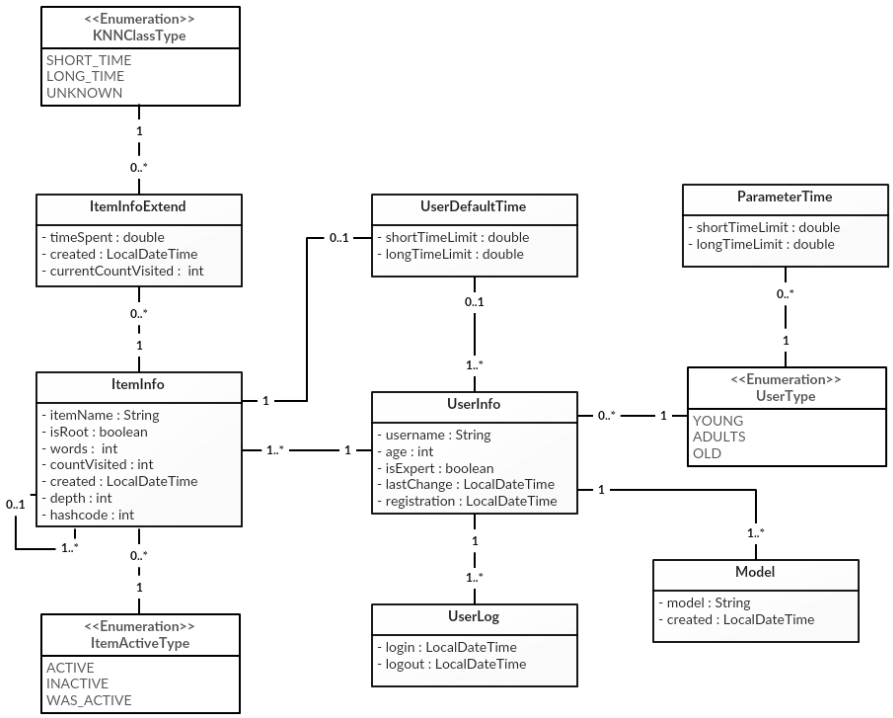
V module **common** sa nachádzjú triedy, ktoré sú potrebné vo viacerých moduloch. Ide o triedy tykajúce sa ukládania informácií o užívateľovi. Ďalej o `SerializationService`, ktorá poskytuje metódy pre serializáciu a deserializáciu meta modelu. Presunul som do tohto modulu stromovú štruktúru `NavigationNode` a všetky triedy, ktoré z nej dedia(`UserNavigationNode`, `UserNavigationNodeExtend` a novo vzniknutá `BaseItemInfo`). Nakoniec sa tu nachádza trieda **Test**, ktorá obsahuje stromové štruktúry určené pre testovanie, ako pre hlavné, tak i pre vedľajšie menu.

Modul **cooperation** zahňuje implementované funkcie popísané v kapitole 3.3.

Modul **point-of-interest** je najzložitejší vzhľadom k ostatným. Obsahuje implementované pravidlá pre detekciu užívateľských rolí z kapitoly 3.1 a radenie prvkov v uzle z kapitoly 3.2. Packages s názvom **sorting** sa tykajú konkrétne pravidla pre rozvrhnutie prvkov. Zvyšne packages prisluchajú pre detetekciu.

## 4.3 Rozlišovanie užívateľských rolí

### 4.3.1 Class diagram




**Obrázok 4.2.** Class diagram pre rozlišovanie užívateľských roli

Trieda `UserInfo` slúži pre uloženie informácií o užívateľovi.

Trieda `Model` uchováva aktuálnu štruktúru(model) užívateľa.

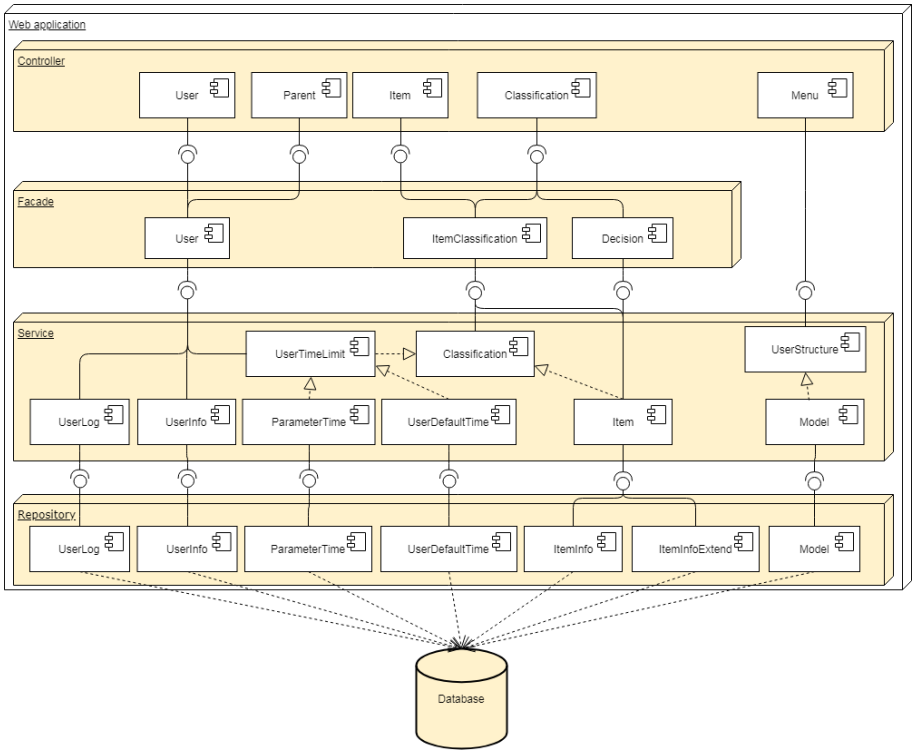
Trieda `ItemInfo` slúži pre ukladanie základných informácií o prvku v menu. Každý užívateľ má iný kľúč pre rovnaký prvok. Pre zistenie, či dané prvky sú rovnaké sa využíva **hashcode** počítaný z mena prvku. Spolu s číslom uzla tvorí jedinečný kľúč prvku, pričom sa predpokladá, že prvkov s rovnakým menom a v rovnakom uzle nebude existovať. Každý prvok si drží referenciu na svojho rodiča. Prvok, ktorý nemá rodiča (je koreňom) má priradenú hodnotu null.

Trieda `ItemInfoExtend` ukladá data o čase strávenom na prvku. Ide o uloženie informácie z prvej fázy algoritmu. Trieda `UserDefaultTime` slúži pre uchovanie konkrétnych časov, ktoré boli vypočítané vo fáze 4. Uchováva limity, ktoré boli vypočítane pre konkrétneho užívateľa a pre konkrétny prvok, ak je aktívnym.

Trieda `ParameterTime` slúži podobne ako `UserDefaultTime` pre uchovanie limitných časov. Rozdiel spočíva v tom, že udržiava limitné hodnoty počí-

tané cez všetkých užívateľov, ktorí patria do rovnakej vekovej skupiny. Ide o globálne hodnoty používané pre novoregistrovaných užívateľov.

### 4.3.2 Popis štruktúry




**Obrázok 4.3.** Komponent diagram pre rozlišovanie užívateľských rolí

**Poznámka:** pre zložitosť počtu komponent sú zobrazené len tie, ktoré majú súvis s rozlišovaním užívateľských rolí.

Na obrázku 4.3 vidíme komponent diagram, ktorý zobrazuje jednotlivé vrstvenie aplikácie a komponenty, s ktorými pracujeme.

U väčšiny Services ide o vrstvu, ktorá komunikuje s repositores a predáva iba informácie pre uloženie alebo získanie. Ide o services `ParameterTimeService` a `UserDefaultTimeService`. `UserTimeLimitService` ide o službu, ktorá pristupuje k dvom vyššie spomenutým services. `UserInfoService` slúži pre uloženie a získanie informácii o užívateľovi. `ItemService` zahŕňa prácu nad dvomi entitami a to `ItemInfo` a `ItemInfoExtend`. `UserMeasureTimeSpentService` je služba, ktorá vykonáva úlohu **fázy 1** algoritmu, kde meria čas strávený na konkrétnom prvku.

Najzložitejšou service je `ClassificationService`, ktorej úlohou je klasifikovať prvok a táto service pokrýva **fázy 2-4**. Umožňuje statické ohodnotenie prvku, pričom zisťuje, či už neexistujú časové limity pre triedy. Ak neexistujú, je volaná metóda `isPointOfInterest` triedy `itemClassification`. Pri

tejto klasifikácii sa používajú **Strategy** a **Temple method patterns**. Strategy pattern je použitý, pretože dochádza k dynamickej zmene chovania pri vyhodnocovaní. Máme 4 stratégie, a to `ShortTimeStrategy`, `ExpectedShortTimeStrategy`, `ExpectedLongTimeStrategy` a `LongTimeStrategy`. Temple method sa používa spoločne so Strategy pattern-om, kde predpisuje abstraktné chovanie metódy pre variantne časti chovania. Tým máme na mysli to, že je potrebné vykonať klasifikovanie predtým, ako sa začne ohodnocovať daný prvok alebo potrebuje klasifikovať prvky po ohodnotení aktuálneho prvku a preto bol použitý temple method pattern. Ďalej umožňuje K-NN klasifikáciu, pre ktorú je volaná trieda `KNNClassification`, ktorá obsahuje implementáciu K-NN algoritmu a má statickú premennú $k$ nastavenú na hodnotu, ktorá bola vysvetlená v sekcii 3.1. Túto hodnotu je ale možné zmeniť programátorom, ak by pre nejakú konkrétnu aplikáciu dávala zlé výsledky. Poslednou operáciou, ktorú umožňuje, je vypočítanie časových limitov, pri ktorých využíva triedu `ItemInfoStatistics`, ktorá má za úlohu vypočítať upper alebo low limits. Využíva triedy `ExponentialDistribution` a `GammaDistribution`. Service umožňuje vypočítanie globálnych, ale i lokálnych limitov.

Pri implementácii tohto algoritmu sa okrem services používajú 4 **facades**. `UserInfoFacade`, ktorá umožňuje zisťovanie či je užívateľ expert alebo stále nováčik, ostatné operácie sú len pre ukladanie alebo získanie informácii od nižšej vrstvy. `ItemFacade` zahrňuje manipuláciu s prvkami v menu. `ClassificationFacade` poskytuje služby pre **fázy 2-4**. `DecisionFacade` sa využíva pre vykonanie rozhodnutia, či nastala zmena.

V aplikácii pre manipulovanie z vonku je potrebný iba `ParentController`, ktorý rozširuje framework o ukladanie informácií o rodičoch(parents). Metóda `addParent` slúži na uloženie časov a update počtu slov na stránke. Okrem parentController-a sa v aplikácii nachádzajú controlleri, ktoré nevyužíva modul core, ale ponúkajú programátorom manipuláciu nad výpočtami, uloženiami alebo iba získaní informácií. Dôsledkom čoho boli vytvorené `UserInfoController`, `ItemInfoController` a `ClassificationController`.

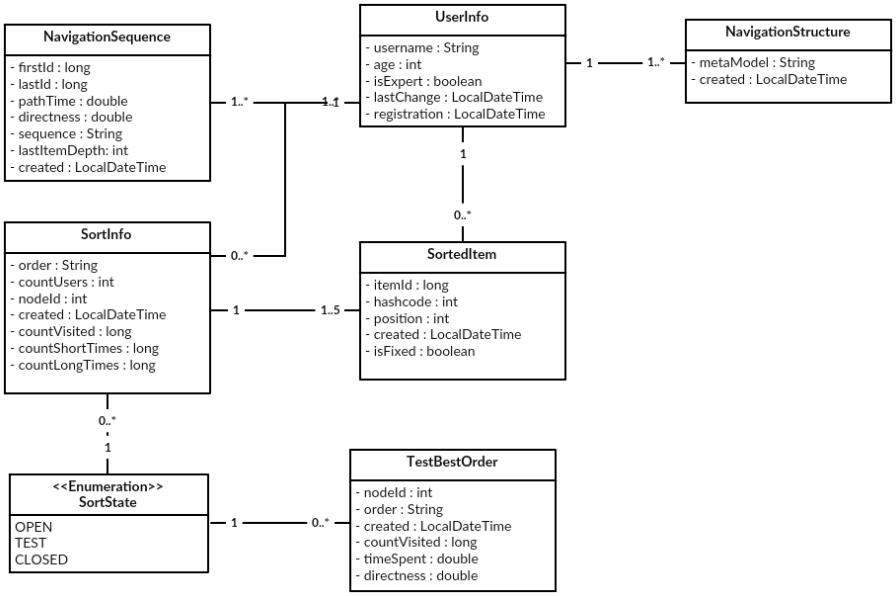
Pre optimalizáciu počtu dotazovaní do databázi sa využíva cache pri zisťovaní `UserInfo` podľa mena. Ide o veľmi časté volanie z rôznych modulov a úrovni aplikácie. Používa sa priamo zabudovaná cache v Spring frameworku.

```
@Cacheable(value = "users")
public UserInfo getUserByUserName(String username) {
return userInfoRepository.getUserInfoByName(username);
}
```

**Listing 1:** Ukládanie užívateľov do cache

## 4.4 Radenie prvkov v uzle

### 4.4.1 Class diagram




**Obrázok 4.4.** Class diagram pre radenie prvkov v uzle

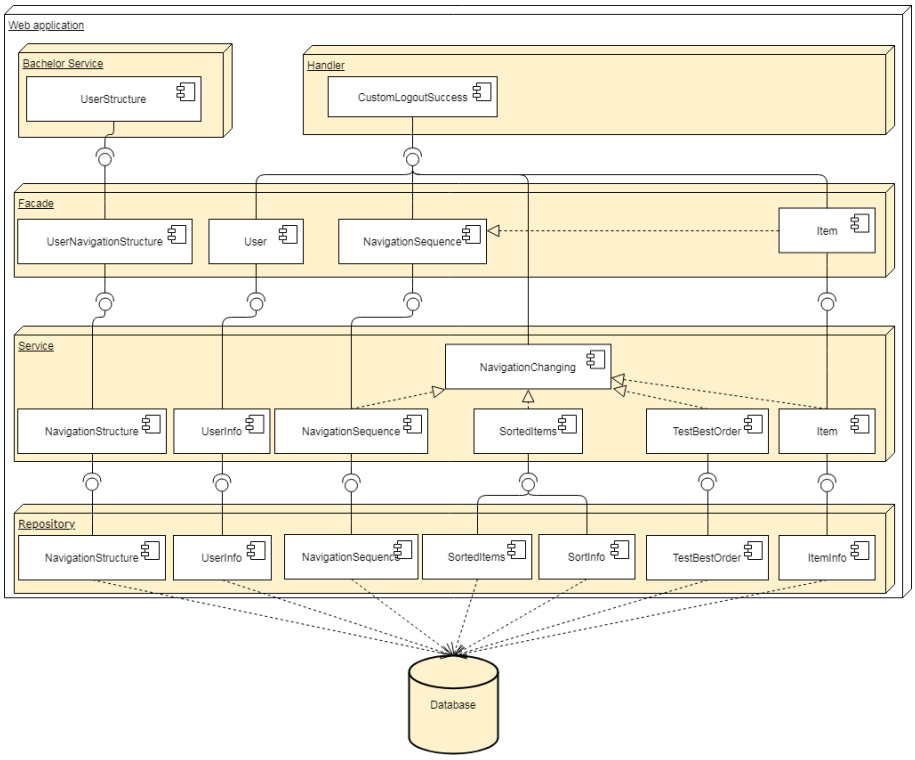
Na Obrázku 4.4 vidíme triedy, ktoré sú potrebné pri radení prvku v uzle. `NavigationSequence` uchováva informácie o sekvencii medzi prvým prvkom klasifikovaným ako krátky a prvým zisteným dlhým časom. Zároveň obsahuje informácie o čase, ktorý bol potrebný pre vykonanie úlohy, teda sekvencii a hodnote priamočiarosti.

Trieda `SortInfo` popisuje poradie prvkov v jednotlivých uzloch a užívateľov, ktorým prislúcha dané poradie. Ide iba o testovacích užívateľov. `SortInfo` má informáciu o poradí prvkov v uzle, ale nemá žiadnu spojitosť s konktrétnym prvkom. Je teda je potrebné ešte pridať informáciu, ktorý prvok má akú pozíciu. To rieši trieda `SortedItem`.

Trieda `NavigationStructure` má rovnaký účel, ako `Model` pri rozlišovaní užívateľských rolí.

Predchadzajúce triedy riešili získávanie informácií od testovacích užívateľov. Poslednou nespomenutou triedou je `TestBestOrder`, ktorá obsahuje vyhodnotenie najlepších poradí v uzloch. Slúži pre porovnanie aktuálnych poradí ostatných užívateľov s iniciovanou zmenou modelu.

### 4.4.2 Popis štruktúry




**Obrázok 4.5.** Komponent diagram pre radenie prvkov v uzle

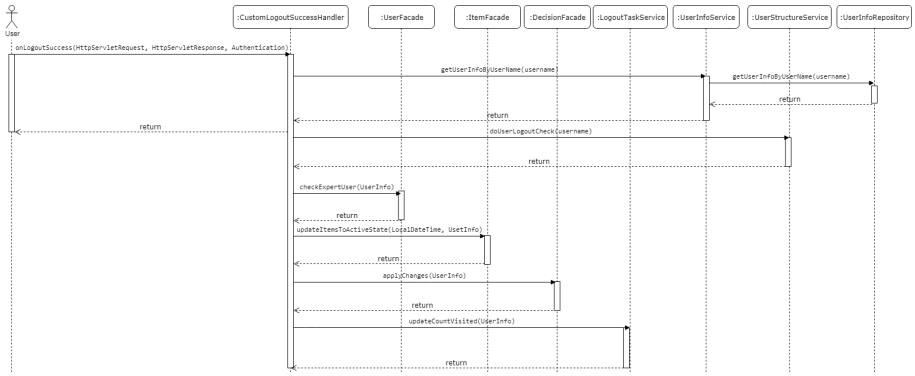
Komponent diagram zobrazený na obrázku 4.5 predstavuje, podobne ako pri predchadzajúcej sekcii o rozlišovaní užívateľských rolí, vrstvenie programu. Môžeme si všimnúť, že sa opakujú niektoré komponenty z predchadzajúceho komponent diagramu. Ide o dôležité komponenty, ktoré sa využívajú naprieč celým programom. Tým máme na mysli `UserInfo` a `ItemInfo`.

Pribudla nám aj nová vrstva, a to `Handler`. V tejto vrstve nájdeme dve triedy, ktoré vykonávajú úlohy pri prihlásení a po odhlásení. Tou dôležitejšiou je `CustomLogoutSuccessHandler`. Pri odhlásení vykonávame nieľko podstatných úkonov:

- Doklasifikujeme neohodnotené prvky.
- Zisťujeme, či je užívateľ skúseným.
- Zisťujeme aktívne prvky
- Zisťujeme prepočítavanie limitov.
- Ak ide o testovacieho užívateľa, tak aktualizujeme počty navštívení prvkov v testovacích uzloch.

Komponenta `NavigationChanging` má za úlohu porovnať viaceré spôsoby radení a najlepšie z nich uložiť do databázy, aby sa pri prihlásení užívateľa

mohol aktualizovať uzol. Zvyšné komponenty majú za úlohy funkcie, ktoré odpovedajú triedam v class diagrame.



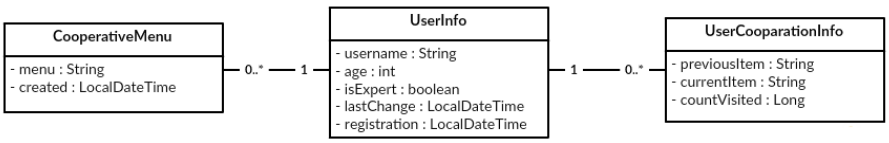

**Obrázok 4.6.** Sekvenčný diagram zobrazujúci odhlásenie užívateľa

Na obrázku 4.6 je zobrazený sekvenčný diagram, ktorý ukazuje vyššie spomínane úlohy vykonávané po odhlásení užívateľa. Môžeme vidieť, že po získaní entity `UserInfo` je užívateľ odhlásený a zvyšok programu pokračuje **asynchrónne**. Kvôli množstvu volaní medzi jednotlivými vrstvami je na sekvenčnom diagrame zobrazené len prvé volanie v najvyššej vrstve.

Ďalej si môžeme všimnúť, že metóda `doUserLogoutCheck` sa pôvodne volala `doUserLoginCheck`[17]. Ide o ďalšiu zmenu vo frameworku. Dôvodom je, aby sa pri prihlasovaní užívateľa nevykonávali žiadne operácie resp. čo najmenej, aby užívateľ dlho nečakal na načítanie stránky.

## 4.5 Kooperácia viacerých menu

### 4.5.1 Class diagram

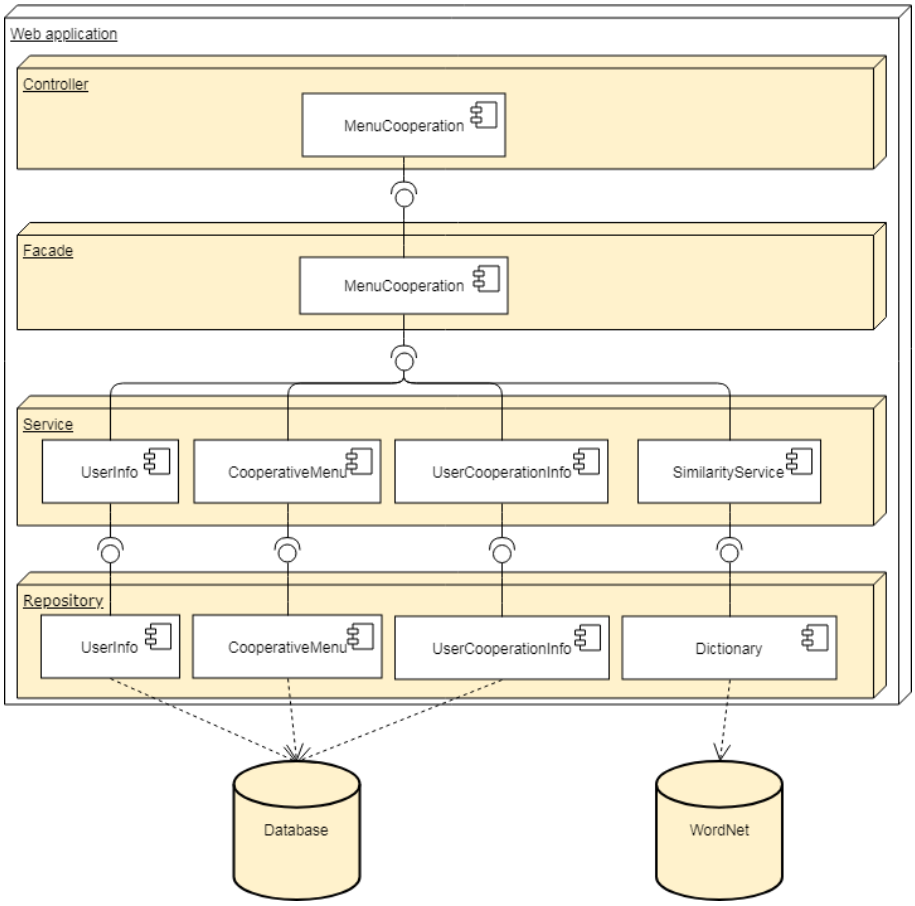



**Obrázok 4.7.** Class diagram pre kooperáciu viacerých menu

Trieda `CooperativeMenu` slúži na uloženie kooperatívneho menu. Podobnú implementáciu môžeme vidieť pri rozlišovaní užívateľských rolí a radení prvkov v uzle. Dôvodom vytvorenia novej triedy je odstránenie závislosti na moduloch medzi sebou, aby vedeli fungovať samostatne.

Trieda `UserCooperationInfo` zahŕňa informácie o frekvencii prechodov medzi jednotlivými prvkami rôznych menu.

### 4.5.2 Popis štruktúry

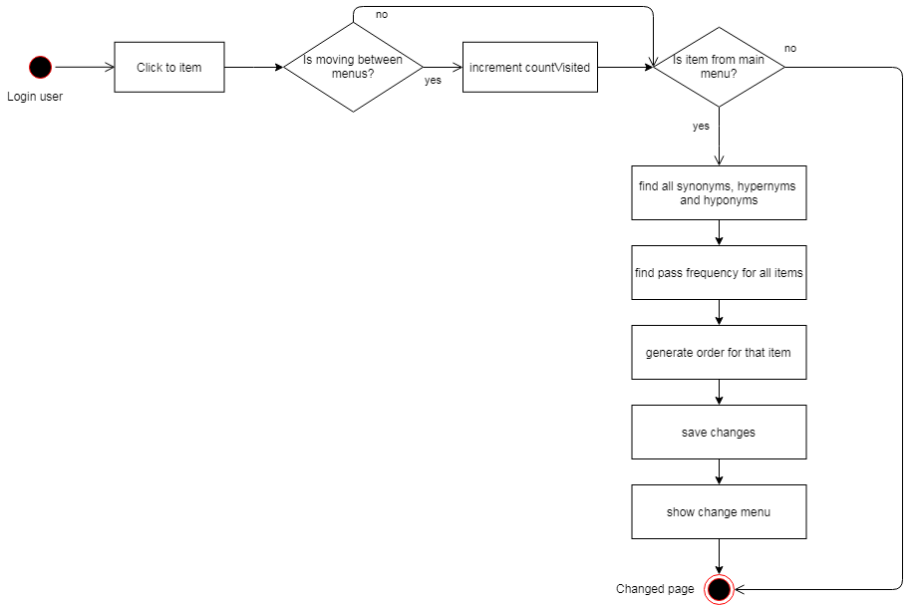



**Obrázok 4.8.** Komponent diagram pre kooperáciu menu

Na obrázku 4.8 môžeme vidieť komponent diagram, ktorý ma podobnú štruktúru ako predošlé dve časti programu. Jediným rozdielom je, že využíva dve databázy, jednu pre ukladanie dát a kontextových dát a druhú pre **WordNet**. WordNet je lexikálna databáza pre anglický jazyk[27]. Pre prístup do databázy využívame framework extjwnl. O prácu s týmto frameworkom sa stará `SimilarityService`. Táto service najprv zistí vzťah dvoch slov na základe synonym. Následne porovná hyperonyma prvku z vedľajšieho menu s aktuálnym prvkom v hlavnom menu. Nakoniec ešte porovná hyponyma. Po porovnaní všetkých slov z vedľajšieho menu so slovom aktuálneho prvku dôjde k zoradeniu slov. O túto funkcionalitu sa stará `MenuCooperationFacade`. Kontrolér poskytuje 4 operácie, ktoré môže programátor využívať. Najdôležítejšie sú:

- `addMenu` - pridá menu, ktoré bude kooperovať s hlavným menu. Je potrebné mať nastavené hlavné menu.
- `getSortedCooperativeMenu` - získanie roztriedeného menu

Na obrázku 4.9 môžeme vidieť fungovaie celého procesu kooperácie.




**Obrázok 4.9.** Fungovanie aktualizácii poradia prvkov v menu na základe druhého

# Kapitola 5
# Testovanie

## 5.1 Testovanie rozlišovania užívateľských úloh pri navigácii menu s užívateľmi

Na základe tohto testovania si overíme úspešnosť statického klasifikovania. Porovnáme rôzne hodnoty k v K-NN algoritme.

Pre testovacie účely bola vytvorená stránka s ponukaním tovarom, ktorá obsahuje menu. Užívatelia si môžu prezerať tovar a čítať jeho popis. Inšpiráciou na túto stránkou boli bežné e-shopy, teda názvy prvkov sú realistické a taktiež ponúkaný tovar. Data boli ziskané zo stránok `www.alza.sk` a `www.datart.sk`. Každý tovar obsahuje obrázok, názov, cenu a popis. Informácie o tovare nám poslúžia na zisťovanie počtu slov a následne pre zistenie `expectedReadTime`(očakávaného času pre "prečítanie" stránky). Ako bolo vyššie spomenuté v kapitole implementácia v triede Test obsahuje testovacie dáta, pre tento test. Užívatelia dostali 3. úlohy pri ktorých mali nájsť prvok v menu, ktorý by mal byť následne klasifikovaný ako dlhý čas. Pri tomto teste sa budú vyhodnocovať všetky časy, ktoré boli získané meraním. Jednotlivé zadania úloh sa snažia byt realistické a vyhýbajú sa pomenovaniu jednotlivých krokov a názvov prvkov[4].

Požiadavky pre užívateľov na testovanie:

- Preklikávanie sa každou úrovňou stromovej štruktúry(modela)
- Na konci bude získaná spätná väzba od užívateľa, kde povie či sa navigoval(klasifikovaný čas ako krátky) alebo prezeral(klasifikovaný čas ako dlhý)

Zadanie úloh

1. Máte záujem o drony, skúste nájsť položku v menu, kde by ste ich hľadali? Okrem toho chcete si vychutnať zábery čo zachytíte pomocou dronu, doma máte pokazenú obrazovku k PC. Skúste si nájsť novú?
2. Predstavte si, že často šoférujete a máte pravidelné telefonáty pri vedení vozidla. Preto si chcete zaobstarať lepší handsfree, aby ste mohli volať

pohodlnejšie a bez odvrátenia pozornosti od vozovky. Kde by ste hľadali potrebný handsfree? Okrem toho máte silnú záľubu v gramofónových platní a chcete si kúpiť novú od vášho obľúbeného speváka. Kde by ste ju hľadali?

3. Predstavte si, že ste si so svojou polovičkou kúpili nový byt a zariaďujete si kuchyňu. Momentálne nemáte kam ukladať potraviny, aby sa vám neskazili. Skúste si vybrať potrebné zariadenie. Ďalej máte radi kofeínové nápoje, ale nemáte mnoho peňazí, ale chcete si namleté zrnka filtrovať. Kde by ste hľadali potrebnú pomôcku? Ešte máte jeden problém o dva týždne máte oslavu a chcete piecť koláče, ale nemáte v čom. Kde by ste hľadali potrebné vybavenie s prípojkou na elektrinu?

| **Užívateľ** | N | Poťet ST | Počet LT | Počet UK | LT $\rightarrow$ *ST* | ST $\rightarrow$ *LT* |
|---|---|---|---|---|---|---|
| 1 | 36 | 30 | 3 | 2 | 6% | 0% |
| 2 | 46 | 35 | 6 | 1 | 2.4% | 0% |
| 3 | 50 | 40 | 5 | 0 | 0% | 0% |
| 4 | 39 | 31 | 6 | 0 | 0% | 0% |
| 5 | 41 | 34 | 6 | 2 | 5% | 0% |
| **Celkovo** | 212 | 170 | 26 | 5 | 2.6% | 0% |

**Tabuľka 5.1.** Výsledok tesotvania pri detekcii užívateľských rolí

Tabuľka 5.2 zobrazuje súhrné informácie o testovacej vzorke. **N** označuje počet odmeraných časov. Počet ST, počet LT a počet UK predstavujú jednotlivé triedy odhadnuté pri klasifikácií. $LT \rightarrow ST$ predstavuje chybu, kedy prvok longTime bol klasifikovaný ako shortTime. $ST \rightarrow LT$ ukazuje chybu, kedy prvok shortTime bol klasifikovaný ako longTime. Môžeme vidieť, že na testovacej vzorke nedošlo ani raz k zlému označeniu shortTime za longTime. Táto informácia je veľmi dôležitá, pretože následne využitie tejto detekcie závisi hlavne na označení longTime. Ďalej si môžeme všimnúť celkovú chybu vypočítanu pre triedy LT a ST. Ide o veľmi nízke číslo. Čo dokazuje účinnosť algoritmu.

V Tabľke 5.2 máme porovnanie jednotlivých hodnôt **k**, aký majú vplyv na celkovú chybu. Pole $Z$ je počet hodnotených hodnôt v K-NN algoritme. Tabuľka demonštruje, že najlepšie hodnoty sú pri hodnote $\mathbf{k = 3}$. Postupným zvyšovaním sa chyba zväčšuje. Čo môžeme vidieť na užívateľoch 1 a 3.

Na obrázku 5.1 je tabuľka, ktorá zobrazuje koľko bolo potrebných preklíknutí pre dokončenie jednotlivých úloh. Priemerne bolo potrebných niečo

| Užívateľ | Z | k = 3 | k = 5 | k = 7 | k = 9 |
|---|---|---|---|---|---|
| 1 | 3 | 5.6% | 11% | 11% | 11% |
| 2 | 6 | 2.17% | 2.17% | 2.17% | 2.17% |
| 3 | 5 | 2% | 4% | 4% | 6% |
| 4 | 2 | 0% | 0% | 0% | 0% |
| 5 | 1 | 5% | 5% | 5% | 5% |
| **Celkovo** | 17 | 2.4% | 2.4% | 2.4% | 2.4% |

**Tabuľka 5.2.** Chybovosť algoritmu pri použití rôzneho k v K-NN algoritme

vyše 5 preklíknutí, pričom najčastejšie bolo potrebných práve 3 navštívení. Tabuľka zobrazuje odpozorované hodnoty, ale aj mierne upravené. Úprava spočívala odstranením príliš krátych hodnôt pri sebe.

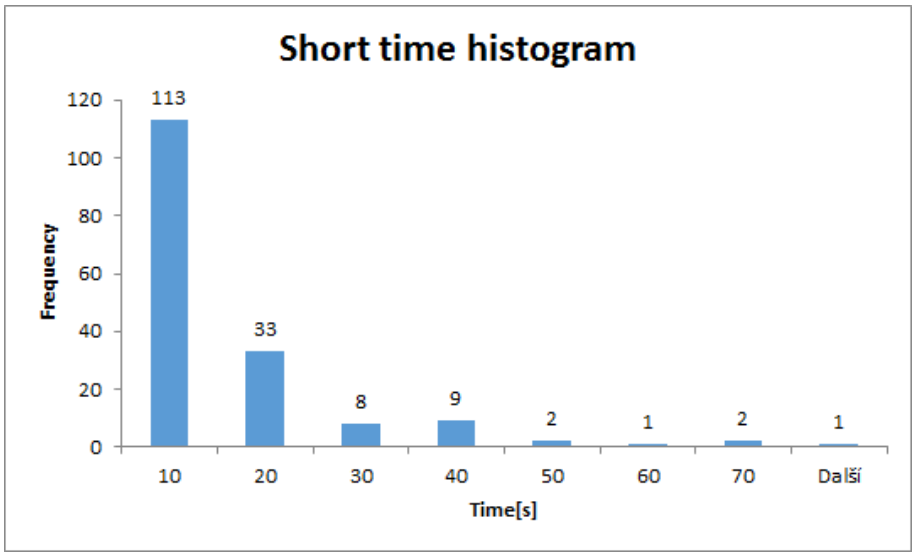
| Užívateľ | | Hľadaní prvok | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Monitory | dron | vinyl | hands-free | chladnicka | french press | pec | SUMA | AVG |
| 1 | Počet prvkov pre splnenie úlohy | 3 | 6 | 7 | 5 | 3 | 2 | 9 | 35 | 5 |
| | Počet prvkov pre splenie úlohy bez šumu | 3 | 6 | 6 | 3 | 3 | 2 | 9 | 32 | 4,571428571 |
| | Splnená úloha | ano | nie | ano | ano | ano | ano | ano | | |
| 2 | Počet prvkov pre splnenie úlohy | 5 | 9 | 8 | 3 | 4 | 7 | 6 | 42 | 6 |
| | Počet prvkov pre splenie úlohy bez šumu | 3 | 8 | 4 | 3 | 3 | 4 | 5 | 30 | 4,285714286 |
| | Splnená úloha | ano | ano | ano | nie | ano | ano | ano | | |
| 3 | Počet prvkov pre splnenie úlohy | 16 | 7 | 3 | 15 | 3 | 3 | 3 | 50 | 7,142857143 |
| | Počet prvkov pre splenie úlohy bez šumu | 14 | 7 | 3 | 14 | 3 | 3 | 3 | 47 | 6,714285714 |
| | Splnená úloha | ano | ano | ano | ano | ano | nie | ano | | |
| 4 | Počet prvkov pre splnenie úlohy | 3 | 6 | 3 | 3 | 3 | 17 | 6 | 41 | 5,857142857 |
| | Počet prvkov pre splenie úlohy bez šumu | 3 | 6 | 3 | 3 | 3 | 10 | 6 | 34 | 4,857142857 |
| | Splnená úloha | ano | ano | ano | ano | ano | ano | nie | | |
| 5 | Počet prvkov pre splnenie úlohy | 4 | 5 | 16 | 4 | 3 | 3 | 5 | 40 | 5,714285714 |
| | Počet prvkov pre splenie úlohy bez šumu | 3 | 5 | 16 | 4 | 3 | 3 | 5 | 39 | 5,571428571 |
| | Splnená úloha | ano | nie | nie | ano | ano | nie | ano | | 0 |
| | | | | | | | | | 208 | 5,942857143 Odpozerané |
| | | | | | | | | | 182 | 5,2 Reálne |
| | | | | | | | | | | |
| | | | | | | | | | 3 | 3 Najčastejšia hodnota |

**Obrázok 5.1.** Počet prekliknutí pre dokončenie úloh

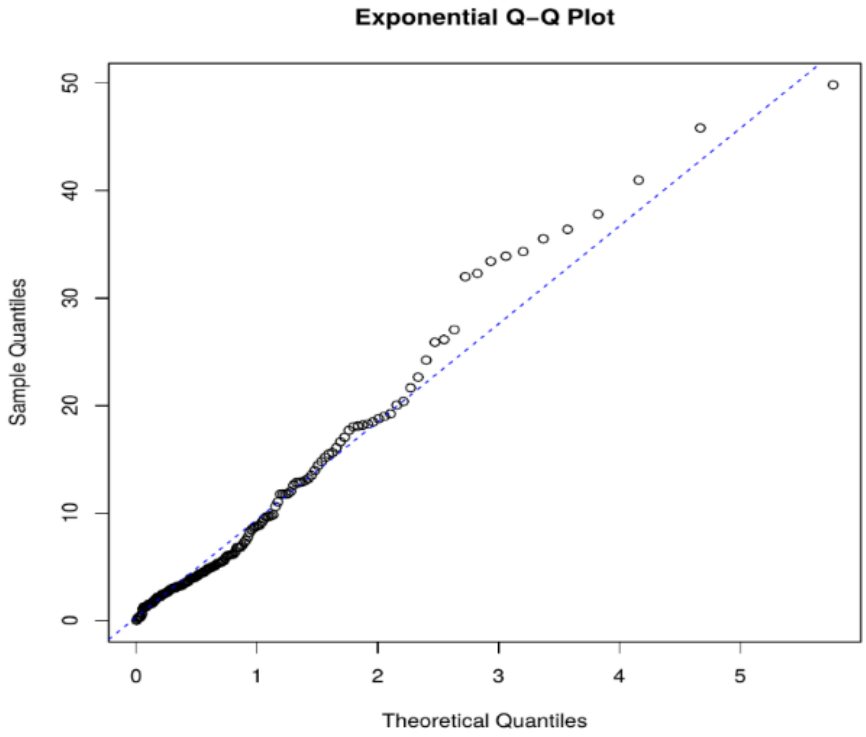
## 5.2 Pravdepodobnostné rozdelenia

V kapitole 3.1 boli popisované pravdepodobnostné rozdelenia pre krátky a dlhý čas. Taktiež sa tam spomínajú jednotlivé testy vykonané na testovacích datách, ktoré sú tu zobrazené. Pre prehľadnosť a množstvo získaných dát sú uvedené len grafy. Všetky testovacie hodnoty sú uvedené na priloženom CD v excel súbore. V jednotlivých listoch sú tabuľky a grafy, z ktorých boli robené vyhodnocovania. Prvé dva obrázky zobrazujú histogram a q-q plot pre krátky čas. Na druhom obrázku môžeme vidieť, že testovacie datá odpovedajú exponeciálnemu rozdeleniu. Zvyšne dva obrázky zobrazujú histogram a q-q plot pre dlhý čas. V tomto prípade je množina testovacích dát menšia ako pri krátkom čase. Z histogramu vidíme, že nejde ľahko odhadnúť o aké ide pravdepodobnostné rozdelenie. Preto som si pomohol q-q plotom, ktorý

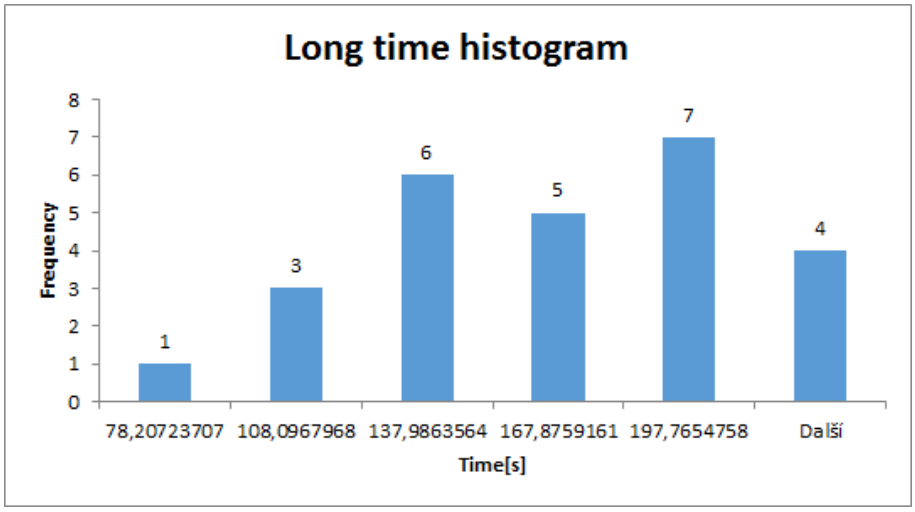
ukázal, že testovacia sada údajov vyhovuje gamma distribúcii. Pri q-q plote menší počet dát nespôsobil výraznú odlišnosť od kvantilovej priamky.



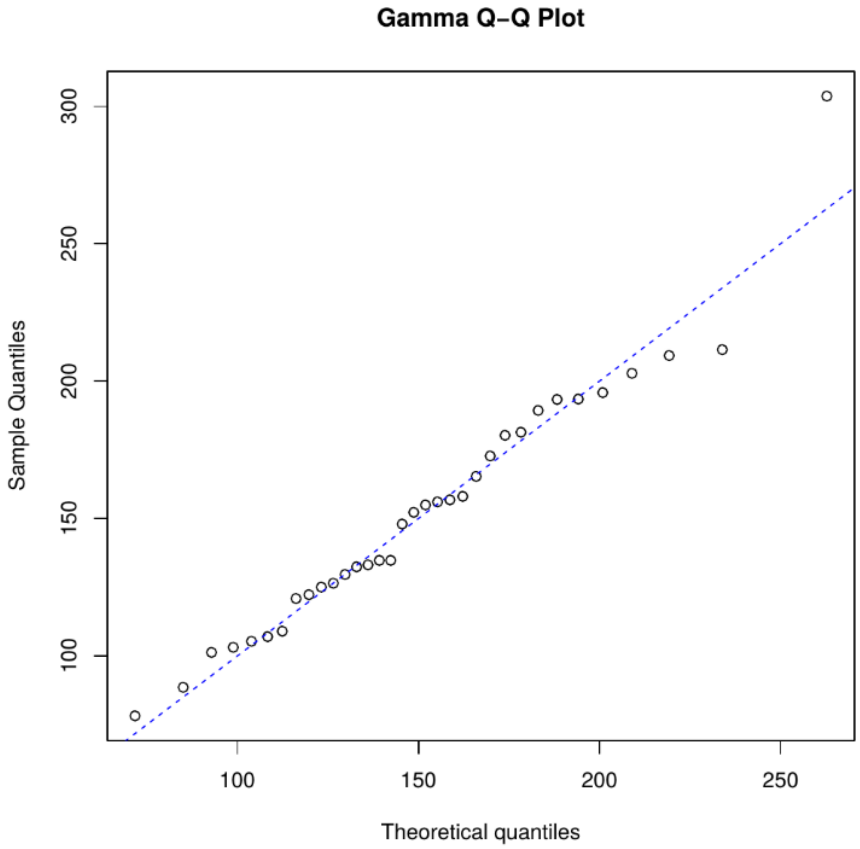

**Obrázok 5.2.** Histogram pre krátky čas




**Obrázok 5.3.** Q-Q plot pre exponenciálne rozdelenie(krátky čas)




**Obrázok 5.4.** Histogram pre dlhý čas




**Obrázok 5.5.** Q-Q plot pre gamma rozdelenie(dlhý čas)

## 5.3 Testovanie radenia prvkov v menu

Testovanie radenia prvkov v menu bolo vykonané na 3 rôznych uzlov v menu. Najprv boli užívatelia testovaní pri prvom spôsobe radenia a neskôr na

druhom spôsobe. Tabuľka zobrazuje priemerné časy pre nájdenie hľadaného prvku. Každý stĺpec obsahuje vľavo prvý spôsobom radenia, ktorý bol lepší na základe času. Prvok sa vždy nachádzal na vrchu alebo na konci v submenu. Vpravo od zvislej čiary ide o druhý predpokladaný horší spôsob radenia. Ako môžeme vidieť celkový priemer je vždy lepší v prvom spôsobe radenia. V niektorých prípadoch u užívateľoch sa vyskytol druhý spôsob ako lepším. Teda môžeme vidieť, že na spôsobe rozvrhnutia prvkov záleží.

| **Užívateľ** | 1. uzol | 2. uzol | 3. uzol |
|---|---|---|---|
| 1 | 9,56s \| 17,75s | 3,39s \| 10,11s | 6,34s \| 10,75s |
| 2 | 13,40s \| 15,17s | 14,40s \| 7,74s | 15,27s \| 15,89s |
| 3 | 7,78s \| 9,50s | 6,13s \| 8,03s | 2,29s \| 13,25s |
| 4 | 6,88s \| 6,50s | 7,45s \| 11,65s | 4,12s \| 6,47s |
| 5 | 10,87s \| 9,88s | 4,77s \| 10,71s | 10,82s \| 10,79s |
| **Priemer** | 9,70s \| 11,81s | 7,23s \| 9,65s | 7,77s \| 11,43s |

**Tabuľka 5.3.** Porovnanie dvoch spôsobov rozvrhnutia menu

## 5.4 Kooperácia viacerých menu

Testovanie kooperácie prebiehalo meraním času pri prechode prvku z hlavného menu do vedľajšieho. Užívateľia najprv boli testovaní bez kooperácie a následne vykonali rovnaké úlohy s kooperáciou. Výsledky vykonáneho testu znázorňuje tabuľka 5.4. Kde môžeme vidieť, že u všetkých užívateľoch došlo k zlepšeniu. Dokonca pri dvoch užívateľoch išlo o zlepšenie rýchlosti navigovania až o 3sekundy. Po aplikovaní kooperácie došlo celkovo k zlepšeniu o 2,88s. Teda môžeme vidieť, že kooperácia pri danej vzorke užívateľov mala za následok zlepšenie rýchlosti navigovania.

| **Užívateľ** | Pred kooperáciou | Po kooperácii |
|---|---|---|
| 1 | 23,35s | 19,81s |
| 2 | 17,72s | 16,5s |
| 3 | 12,67s | 10,23s |
| 4 | 14,18s | 11,06s |
| 5 | 19,44s | 15,37s |
| **Priemer** | 17,47s | 14,59s |

**Tabuľka 5.4.** Porovnanie rýchlosti navigovania pred a po kooperácií

# Kapitola 6
# Inštalácia

1. Vložte CD do mechaniky.
2. Rozbaľte archív Application.zip na svoj disk.
3. Pomocou vyvojového prostredia napr. Intellij IDEA naimportujte rozbalenú aplikáciu.
4. Spustíte Gradle build.
5. Aplikácia sa spúšťa pomocou main metódy, ktorá sa nachádza v `module-core` a v triede `BachelorThesisApplication`.

**Poznámka:** Aplikácia používa databázový server H2, ktorý nie je potrebné pre spustenie aplikácie ďalej konfigurovať. Pri vývojí som používal databazový server Oracle 12c. Pre jeho využitie je potrebné nastaviť parametre nachadzajúce sa v súbore `application.yaml`.

# Kapitola 7
# Záver

Cieľom bakalárskej práce bolo nájsť nové pravidlá resp. problémy pri adaptívnych štruktúrach a navrhnúť pre ne riešenia. Následne tieto pravidlá implementovať a otestovať.

Pri analýze som dospel k skutočnosti, že bude vhodné danú problematiku rozdeliť do troch častí. Každá časť obsahuje podpravidlá. Pre jednotlivé problematiky boli navrhnuté algoritmy. Ku každej problematike sú vysvetlené dôvody použitia riešení a na konci každej časti sa rozoberajú výhody a nevýhody.

Prvou časťou je rozlišovanie užívateľských rolí, ktorá zisťuje, či sa užívateľ naviguje alebo si prezerá stránku. Alogritmus, ktorý je delený do fáz, bol otestovaný na užívateľoch. Výsledkom bolo zistenie jeho vysokej učinnosti detekovania rolí.

Druhá časť spočíva v radení prvkov v menu. Zisťujem, ktoré poradie je vhodnejšie na základe testovania na užívateľoch. Na demonštráciu potreby tohto pravidla bol vykonaný test, ktorý dokazuje, že odlišné poradia majú rôzny vplyv na užívateľov.

Posledná časť prináša rozšírenie adaptácie medzi viaceré menu. Napomáha k rýchlejšimu navigovaniu pomocou aktualizácie vedľajšieho na základe pozície v hlavnom menu. Presúvaním prvkov medzi oboma menu dochádza k zlepšeniu naleziteľnosti.

Navrhnuté pravidlá vylepšujú rýchlosť navigovania. Zmeny aplikované po každej adaptácii nepôsobia na užívateľov mätúco, pretože každa zmena mení len nejakú časť štruktúry, ale nikdy nie celú.

Následne boli pravidlá implementované už do existujúceho Java EE frameworku. Pri implementácii som dbal na správny návrh, použitie vrstvenia aplikácii a design patternov. Taktiež sa využíva cache a asynchrónne volania metód pre optimalizovanie vykonávania úloh, čoho výsledkom je funkčné rozšírenie frameworku. Framework disponuje radov podporných operácii pre vlastné customizovanie a získavanie dát.

## 7.1 Budúca práca

Budúca práca umožňuje širokú škálu rozšírení a vylepšení navrhnutých pravidiel. Pri rozlišovaní uživateľských úloh ide o vylepšenia ako:

- Využitie **elbow metódy** pre zistenie správnej hodnoty $k$ v K-NN algoritme. Momentálne je hodnota $k$ nastavená na hodnotu 3. To nemusí vyhovovať vo všetkých aplikáciach alebo pri hranici medzi krátkou a dlhou triedou.
- Rozšírenie kontextových informácií pri detekcii rolí, použitie **eyetracking** pre zisťovanie polohy očí, kde sa užívateľ pozerá na stránke.
- Otestovať priemerný čas na stránke s odlišným počtom slov na mobilných a desktopových aplikáciách. Štúdia[9] popisuje iba priemrný čas pre webové stránky.

Ďalšími zmenami pri radení prvkov v uzle by mohli byť:

- Odstránenie permutácie. Nahradenie generovania poradia pomocou obmedzení, napríklad netestovať všetky poradia, ale iba každé druhé ak sa nejedná o čísla. To by znamenalo, že pri vyššom počte prvkov v uzle zmenšilo počet testovaní a rýchlejšie by sa zistilo najlepšie poradie.
- Podobne ako pri detekcii rošírenie kontextových informácií. Išlo by o aplikovanie **first-click**[18]. Ide o súvis prvého kliknutého prvku v koreni stromovej štruktúre s hľadaním prvkom v jej podstrome.

U kooperácii viacerých menu by mohlo dôjsť k zlepšeniu u nasledujúcich problémov:

- Zisťovanie korelácie medzi menu, napr. pomocou významového porovnania prvkov medzi sebou.
- Využitie **parent naming** pri presune prvkov.

Poslednou časťou, kde môžeme hľadať vylepšenia je samotná implementácia. Prvým vylepšením by malo byť redukovanie počtu dotazovaní do databázy, využitím cache vo viacerých častiach programu, ako je momentálne použitá pri dotazovaní na užívateľoch. Zmenšiť počet zisťovaní, či nastala zmena a jej počítanie a vylepšiť to nastavením konkretného času, kedy dôjde k prepočítaniu, napríklad raz za deň v noci.

# Príloha A
## Symboly

| | |
|---|---|
| AUI | Adaptive User interface |
| GUI | Graphical User Interface |
| IA | Information Architecture |
| Java EE | Java Enterprise Edition |
| JPA | Java Persistence API |
| MLE | Maximal likelihood estimation |
| ORM | Object-relational mapping |
| UI | User interface |

# Príloha B
# Priklady kodov

Listing 1: Ukládanie užívateľov do cache

# Príloha C
# Literatúra

[1] Balakrishnan, K. (2018). *Exponential distribution: theory, methods and applications.* Routledge.

[2] Abdi, A., & Kaveh, M. (1999, July). *On the utility of gamma PDF in modeling shadow fading (slow fading).* In 1999 IEEE 49th Vehicular Technology Conference (Cat. No. 99CH36363) (Vol. 3, pp. 2308-2312). IEEE.

[3] Nielsen, J. (1998, October 4). *Paradox of the Active User.* Retrieved from `https://www.nngroup.com/articles/paradox-of-the-active-user`

[4] *Task Scenarios for Usability Testing.* (2014, July 12). Retrieved from `https://www.nngroup.com/articles/task-scenarios-usability-testing`

[5] Whitenton, K. (2017, May 7). *Tree Testing: Fast, Iterative Evaluation of Menu Labels and Categories.* Retrieved from `https://www.nngroup.com/articles/tree-testing`

[6] Whitenton, K. (2017, July 9). *Tree Testing Part 2: Interpreting the Results.* Retrieved from `https://www.nngroup.com/articles/interpreting-tree-test-results`

[7] *Control Charts for Exponential Distributions.* (2011, October 02). Retrieved from `https://brunochassagne.wordpress.com/2011/10/02/control-charts-for-exponential-distributions`

[8] Nielsen, J. (2000, February 6). *Novice vs. Expert Users.* Retrieved from `https://www.nngroup.com/articles/novice-vs-expert-users/`

[9] Nielsen, J. (2008, May 6). *How Little Do Users Read?* Retrieved from `https://www.nngroup.com/articles/how-little-do-users-read/`

[10] Hinkle, V. (2008). *Card-sorting: What you need to know about analyzing and interpreting card sorting results.* Usability News, 10(2), 1-6.

[11] Cover, T. M., & Hart, P. E. (1967). *Nearest neighbor pattern classification.* IEEE transactions on information theory, 13(1), 21-27.

[12] Nielsen, J. (2006, June 26). *Quantitative Studies: How Many Users to Test?* Retrieved May 6, 2019, from `https://www.nngroup.com/articles/quantitative-studies-how-many-users`

[13] Department of Electronic Engineering City University of Hong Kong (Semester B 2018-2019). *Interface Type and Screen Design* [PDF file]. Retrieved from `http://www.ee.cityu.edu.hk/~hcso/ee4213_8.pdf`

[14] Crestodina, A. (2019, February 15). *Are You Making These Common Website Navigation Mistakes?* Retrieved from `https://neilpatel.com/blog/common-website-navigation-mistakes/`

[15] Gulla, F., Cavalieri, L., Ceccacci, S., Germani, M., & Bevilacqua, R. (2015, August). *Method to design adaptable and adaptive user interfaces.* In International Conference on Human-Computer Interaction (pp. 19-24). Springer, Cham.

[16] Findlater, L., & McGrenere, J. (2004, April). *A comparison of static, adaptive, and adaptable menus.* In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 89-96). ACM.

[17] Nikita, M. (2017, February 09). *Aspektově orientovaný vývoj adaptivní struktury aplikace pro Java EE aplikace.* Retrieved from `https://dspace.cvut.cz/handle/10467/75784`

[18] Bailey, B. (2013, October 08). *FirstClick Usability Testing.* Retrieved from `http://webusability.com/firstclick-usability-testing/`

[19] Loos, E. (2011, July). *In search of information on websites: a question of age?.* In International Conference on Universal Access in Human-Computer

Interaction (pp. 196-204). Springer, Berlin, Heidelberg.

[20] *AJAX Introduction.* (n.d.). Retrieved from `https://www.w3schools.com/xml/ajax_intro.asp`

[21] *Liquibase.* (n.d.). Retrieved from `https://www.liquibase.org/`

[22] *Spock.* (n.d.). Retrieved from`http://spockframework.org/`

[23] *Spring Data JPA.* (n.d.). Retrieved from `https://spring.io/projects/spring-data-jpa`

[24] *Hibernate.* (n.d.). Retrieved from `http://hibernate.org/`

[25] Autayeu, A., & Fedoseeva, O. (n.d.). *ExtJWNL (Extended Java WordNet Library).* Retrieved from `http://extjwnl.sourceforge.net/`

[26] Findlater, L. (2004). *Comparing Static, Adaptable, and Adaptive Menus.*

[27] *WordNet.* (n.d.). Retrieved from `https://wordnet.princeton.edu/`

[28] Mitchell, J., & Shneiderman, B. (1989). *Dynamic versus static menus: an exploratory comparison.* ACM SigCHI Bulletin, 20(4), 33-37.

[29] Shneiderman, B. (1997, January). Direct manipulation for comprehensible, predictable and controllable user interfaces. In IUI (Vol. 97, pp. 33-39).

[30] MacLean, A., Carter, K., Lövstrand, L., & Moran, T. (1990, March). *User-tailorable systems: pressing the issues with buttons.* In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 175-182). ACM.

# Príloha D
# Obsah CD

Obsah CD je organizované do nasledujúcich súborov a priečinkov:

- bakalarska_praca – priečinok obsahuje bakalársku prácu v pdf formáte, taktiež zdrojové tex súbory, obrázky, excel súbor obsahujúce štatistiku na tetstovaných datách
  - bakalarska_praca.pdf - bakalárska práca v pdf formáte
  - tex - priečinok z tex súbormi
  - tex\img - obrázky použíte v bakalárskej práci
  - bp_statistics.xlsx - štatistika na testovacích datách
- Application.zip – implementovaný framework