VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

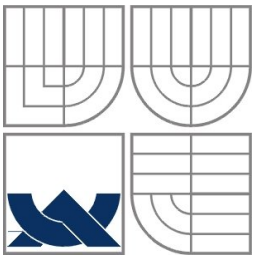
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DYNAMICKÝ DEFINOVATELNÝ DASHBOARD

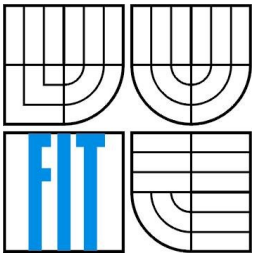
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. BORIS POČATKO
AUTHOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DYNAMICKÝ DEFINOVATELNÝ DASHBOARD

DYNAMIC DEFINABLE DASHBOARD

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. BORIS POČATKO

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL ŠEBEK

BRNO 2012

Zadání diplomové práce/13243/2011/xpocat00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2011/2012

Zadání diplomové práce

Řešitel: **Počatko Boris, Bc.**
Obor: Informační systémy
Téma: **Dynamický definovatelný dashboard**
Dynamic Definable Dashboard
Kategorie: Databáze

Pokyny:

1. Seznamte se s přístupy tvorby dashboardů a problematikou proudů dat.
2. Analyzujte problematiku tvorby dashboardů dynamicky s ohledem na možnost změny sledovaných měr.
3. Navrhněte metodu tvorby uživatelem dynamicky definovatelného dashboardu. Zaměřte se na možné optimalizace zpracování pro rozsáhlá proudová data.
4. Navržené řešení implementujte.
5. Zhodnoťte efektivitu implementace a diskutujte další možný vývoj systému.

Literatura:

- Few, S.: *Information Dashboard Design: The Effective Visual Communication of Data*. O'Reilly Media, 2006, 224 p. ISBN 978-0596100162
- Hotek, M.: *Microsoft SQL Server 2008 :krok za krokem*. Vyd. 1. Brno : Computer Press, 2009. 488 s. : il. ISBN 978-80-251-2466-6
- Harts, D.: *Microsoft SQL Server 2008 R2 analytics & data visualization*. New York : McGraw-Hill, c2011. xxix, 545 s. : il. ISBN 978-0-07-160143-6

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Šebek Michal, Ing.**, UIFS FIT VUT
Datum zadání: 19. září 2011
Datum odevzdání: 23. května 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato diplomová práce pojednává o návrhu a implementaci dynamicky definovatelného dashboardu. Uživatel si bude moci definovat podmínky, které budou filtrovat a zaznamenávat pouze informace pro něj důležité. Zároveň bude aplikace podporovat změnu podmínek a vzhledu jednotlivých grafů za běhu systému. Nastudované implementace, které jsou dostupné na webu, jsou určené jen pro jeden účel a nejsou navrhnuté tak, aby respektovaly všeobecně platné postupy pro tvorbu dashboardu. Dashboard řešený v rámci diplomové práce je navržen tak, aby zvládal databáze s vysokou zátěží a nezpomaloval chod celé aplikace.

Abstract

This thesis deals with the design and implementation of a dynamic user-definable dashboard. The user will be able to define conditions dynamically, which will filter out and save only the data he needs. The application will support the changing of the condition definitions and the display of the graphs after they were created. The current implementations available on the internet are usually solutions designed to fit only one type of project and are not designed to meet general guidelines for a dashboard. The dashboard is designed for a smooth cooperation with high load databases and therefore not to slow down the whole solution.

Klíčová slova

Dynamický definovatelný dashboard, proud dat, databázový trigger, graf, Microsoft SQL server.

Keywords

Dynamic definable dashboard, data stream, database trigger, graph, Microsoft SQL database.

Citace

Počatko Boris: Dynamický Definovatelný Dashboard, diplomová práce, Brno, FIT VUT v Brně, 2012

Dynamický definovatelný dashboard

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Michala Šebka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Boris Počatko
15.5.2012

Poděkování

Děkuji panu Ing. Michalu Šebkovi za cenné rady poskytnuté v průběhu celé práce.

Obsah

Obsah 1
Úvod 3
1 Dashboard 5
1.1 Definícia dashboardu 5
1.2 História a vznik dashboardov 6
1.3 Uplatnené princípy 7
1.4 Príklad dashboardu s nedostatkami 10
1.5 Grafy vhodné na prezentáciu informácií 11
1.6 Postup vytvárania vzhľadu dashboardu 13
1.6.1 Funkcionalita 15
1.6.2 Správne použité grafy v dashboardoch 15
2 Prúdové spracovanie dát 17
2.1 Dátové techniky spracovania prúdu dát 17
2.1.1 Vzorkovanie 18
2.1.2 Load shedding 18
2.1.3 Sketching 18
2.1.4 Dátové štruktúry - synopse 19
2.1.5 Agregačné techniky 19
2.1.6 Jednoduché náhodné vzorkovanie 19
3 Architektúra riešenia 20
3.1 Požiadavky na systém 20
3.2 Stavba aplikácie 21
3.3 Klient/server model 22
3.3.1 Definícia modelu klient/server 22
3.3.2 Dynamický dashboard na architektúre klient/server 23
3.3.3 Tok dát na architektúre klient/server 24
3.3.4 Trigger 25
3.4 Dynamická tvorba podmienok 26
3.4.1 Definovateľné parametre podmienok 26
4 Návrh 28
4.1 Návrh dátovej vrstvy 28
4.2 Technológie užívateľského rozhrania 29
4.2.1 HTML 29
4.2.2 JavaScript 30

4.2.3 CSS 30
4.2.4 Výber implementačnej techniky 30
4.3 Návrh implementácie logickej vrstvy 31
4.3.1 PHP (Jazyk) 32
4.3.2 ASP.NET 32
4.3.3 Python (Jazyk) 32
4.3.4 Perl (Jazyk) 33
4.3.5 Výber platformy 33
4.4 Formálny návrh 33
4.4.1 Pomocné databázové tabuľky 33
4.4.2 Diagram balíkov 34
4.4.3 Diagram nasadenia 34
5 Implementácia 36
5.1 Implementačné zásady 36
5.2 Fyzická štruktúra aplikácie 36
5.2.1 Súbory v adresári App_Code 36
5.2.2 Súbory v adresári App_Data 37
5.2.3 Súbory v adresári App_Themes 37
5.2.4 Ostatné súbory 38
5.3 Postup pri implementácii 38
5.3.1 Overenie možností navrhovaného riešenia 38
5.3.2 Implementácia správy databázových spojení 39
5.3.3 Implementácia správy grafov 39
5.3.4 Implementácia a správa triggerov 41
5.3.1 Implementácia nastavení a administračného rozhrania 42
5.3.2 Implementácia dashboardu 44
5.3.1 Postup pridávania a odoberania objektov 48
5.3.2 Zbieranie štatistických údajov 49
5.3.3 Vzorkovanie 49
5.3.4 Optimalizácie 50
6 Testovanie a zhodnotenie výsledkov 51
6.1 Spôsob testovania 51
6.2 Namerané výsledky 53
6.3 Vyhodnotenie 55
7 Možné rozšírenia systému 57
8 Záver 58
Príloha 1 – Obsah priloženého CD 62

Úvod

V dnešnej dobe sme na každom kroku zahlcovaný veľkým množstvom informácií. Sociálne siete, spravodajské servery či informácie v práci musíme každý deň triediť na informácie, ktoré vyžadujú od nás okamžitú reakciu, dôležité informácie a nepodstatné. Čím je dát viac, tým sa tieto triedia ťažšie a rozpoznanie rizikového stavu sledovaných dát sa stáva prakticky nemožným. Najproblematickejší je tento stav v pracovnom prostredí, kde sú správne rozhodnutia nesmierne dôležité. Najmä v tomto rozhodovaní pri kritických situáciách záleží na kvalitnom zdroji a spracovaní dát. Tento zdroj dát stále častejšie predstavuje dashboard.

Dashboardy nás sprevádzajú v dnešnej dobe na mnohých miestach. Nahrádzajú postupne zložité prehľady a umožňujú užívateľom prijať správne rozhodnutie v krátkom okamihu nad veľkým množstvom dát. Na zostavenie dashboardu je zvyčajne nutné prefiltrovať tisíce záznamov a vybrať iba tie najpodstatnejšie. Už pri tomto krátkom opise je zrejmé, aké užitočné dashboardy môžu byť, ak sa dodržuje pri ich tvorbe pár základných princípov.

Medzi príklady dashboardov možno uviesť rozličné internetové stránky zobrazujúce informácie v kumulovanej forme, ako napríklad stránky o počasí[1], o vývoji na burzách[2], o ekonomických ukazovateľoch[3], medzi ktoré sa počítajú prirodzene aj banky. Banky môžu dashboardy, alebo ich zjednodušené verzie používať na zobrazovanie aktivity na účtoch alebo v samotnom internetovom bankovníctve. Dashboardy sa zároveň používajú na zobrazovanie a sledovanie dopravnej situácii v leteckom priemysle alebo v automobilovej doprave v jednotlivých mestách. Všetky uvedené príklady sú dashboardy vytvorené pre danú aplikáciu, ktoré neumožňujú ich použitie v inom prostredí. Zároveň ani jeden z uvedených príkladov nespĺňa princípy tvorby dashboardov, ktoré sú spomenuté v jednej z nasledujúcich kapitol. Tieto dashboardy boli vytvárané pre jedno použitie a riadia sa skôr heslom, čo sa páči užívateľom je správne. Dashboard by pritom mal byť čo najjednoduchší a bez zbytočných grafických prvkov.

Ako vidno na uvedených príkladoch, dashboard môže byť použitý pre rôzne účely a v rôznych formách. V tejto práci sa obmedzíme na spracovanie textových a číselných dát, keďže práve tieto dáta sú základným stavebným kameňom každého dashboardu. Aplikácia bude rozšíriteľná aj o ďalšie zobrazovacie typy, keďže dátová a zobrazovacia vrstva programu bude oddelená a rozhranie na vytváranie pravidiel pre sledovanie dát bude dostatočne všeobecné na vytváranie pravidiel použiteľných na získanie akýchkoľvek dát, napríklad booleovských hodnôt.

V tejto práci sa zaoberám tvorbou dashboardu pre prúdové dáta nad obecnou Microsoft SQL Server databázou, kde databázová štruktúra nie je vopred definovaná. Práca nad prúdovými dátami

[1] <http://www.weather.com/weather/monthly/98168>

[2] <http://www.stockcharts.com>

[3] <http://www.tradingeconomics.com>

síce neovplyvní výzor dashboadu, ale vnútorná architektúra sa od bežného dashboardu značne líši, keďže v tomto prípade pracujeme nad oveľa väčším objemom dát ako pri bežnom dashboarde. Zároveň musí byť daný dashboard dynamicky zostaviteľný užívateľom, aby si ho mohol prispôsobiť k svojim potrebám.

V úvode čitateľa oboznamujem s problematikou v kontexte reálnych riešení spolu s príkladmi použitia dashboardu. Týmto si čitateľ v ďalších kapitolách jednotlivé aspekty riešenia zadania predstaví v kontexte reálnych príkladov. Druhá kapitola približuje pojem dashboardu, jeho definíciu a všeobecne platné postupy, ktoré sa odporúčajú dodržiavať pri jeho tvorbe z grafického aj z obsahového hľadiska. Čiastočne sa kapitola zaoberá spracovaním vizuálnej informácie pozorovateľom pre lepšie pochopenie problematiky. Princípy správnej tvorby dashboardu aplikujeme na jeden z nájdených dashboardov a prevedieme ho na variantu rešpektujúcu tieto pravidlá. Tretia kapitola jedná o prúdovom spracovaní dát a prístupoch zaznamenávania charakteristík daného prúdu. Štvrtá kapitola opisuje základný prístup k riešeniu problému a zvolenej základnej architektúry. V piatej kapitole je priblížený návrh aplikácie z logického a zo štrukturálneho hľadiska. Zároveň sa v tejto kapitole porovnávajú možné platformy na implementáciu riešenia a implementačné prístupy. V šiestej kapitole sa opisuje implementácia jednotlivých častí riešenia, od užívateľského rozhrania po vnútornú štruktúru objektov použitých na spravovanie dát v aplikácii. Čiastočne sa venujem aj použitým optimalizáciám. V siedmej kapitole opisujem, akým spôsobom sa aplikácia testovala a v akom pripade je riešenie možné použiť. V ôsmej kapitole sa venujem oblastiam, v ktorých je aplikáciu možné rozšíriť, prípadne, ktoré jej časti modifikovať s ohľadom na optimalizáciu.

1 Dashboard

V nasledujúcich kapitolách si priblížime, čo sa skrýva pod pojmom dashboard a uvedieme si v skratke príklady na dobre navrhnutý dashboard a na dashboardy, ktoré nie sú z informatívnej ani z grafickej stránky navrhnuté optimálne. Uvedené príklady pomôžu pri návrhu vlastného dashboardu, respektíve jeho stavebných blokov, keďže užívateľ musí byť schopný si zostaviť vlastný dashboard, ktorý bude spĺňať kritériá pre jednotlivé úlohy čo najlepšie. V nasledujúcich podkapitolách je čerpné primárne zo zdroja [1].

1.1 Definícia dashboardu

Definícia dashboardu nie je všeobecne stanovená. Väčšina definícií sa uchyľuje k opisom konkrétnych implementácií, s ktorými sa autor definície oboznámil a prišiel do kontaktu. Zároveň sa často skloňuje, už z názvu vyplývajúca spojitosť (dashboard = palubná doska), že by sme si mali pri návrhu dashboardu za príklad položiť palubnú dosku vozidla. Neskôr si zdôvodníme, prečo zobrazovacie prostriedky pužité v automobile nie sú vhodným nástrojom na zobrazovanie dát v informačnom prostredí.

Z dôvodu chýbajúcej odbornej definície sa autor Stephen Few pokúsil o zostavenie definície zo svojich poznatkov získaných pri práci a implementácii dashboardov v rozlyčných prostrediach. Dôležitá je jeho poznámka, že dashboard je forma prezentácie a nie samotná technológia. Túto skutočnosť si neuvedomuje väčšina spoločností a autorov, ktoré sa snažia vytvoriť definíciu pre pojem dashboardu. Preklad definície z [1] znie nasledovne:

“Dashboard je vizuálne zobrazenie najdôležitejších informácií, ktoré sú potrebné na dosiahnutie jedného alebo viacerých cieľov. Aby sme boli schopný získať potrebné informácie na prvý pohľad, tak musí dashboard byť konsolidovaný a rozmiestnený tak, aby bol celý zobrazený na jedinej obrazovke.”

Z tejto definície plynú viaceré dôležité poznatky. Dashboard musí spĺňať hlavne nasledujúce body, aby spĺňal svoj účel čo najlepšie:

- dashboard musí byť zobrazený na jednej obrazovke,
- musíme vyňať iba najdôležitejšie informácie,
- jedná sa o vizuálne zobrazenie informácií, takže treba použiť čo najvhodnejšie prostriedky a útvary.

Z týchto bodov plynú nasledujúce závery pre túto prácu. Treba si uvedomiť, že navrhovaný dashboard má byť dynamický, konfigurovateľný, ale zároveň má spĺňať vyššie uvedené princípy. Takže úlohou tohto dashboardu bude umožniť užívateľovi zostaviť čo najdynamickejší dashboard a zároveň ho obmedziť tak, aby výsledná konfigurácia spĺňala dané princípy v najväčšej možnej miere. Niektoré princípy sa môžu riešiť aj slovným doporučením, keďže pri veľkom množstve informácií a obmedzenom zobrazovacom priestore môže nastať situácia, keď dashboard prekročí medze jednej obrazovky a užívateľ si na daných dátach bude trvať naďalej. Preto by nebolo žiaduce obmedziť zobrazovaciu plochu na jednu obrazovku. Dodržovanie tohto princípu musí z tohto dôvodu byť ponechané na samotnom užívateľovi, respektíve na osobe zostavujúcej dashboard.

Ďalší princíp, zobrazenie iba dôležitých informácií, je v kontexte tejto práce taktiež ťažko dodržateľným pravidlom. Daný dashboard je pripojiteľný na ktorúkoľvek databázu, a preto sa nedá vopred určiť aké informácie sú dôležité. Z toho vyplýva, že aj tento princíp musí byť ponechaný čiastočne na užívateľovi. Je vhodné podotknúť, aké princípy sa musia použiť pri výbere dát a vytvoriť doporučenie aj pre tento princíp vytvárania užitočného dashboardu. Zobrazované dáta musia byť pretriedené do takej miery, aby neprestali slúžiť danému účelu v kontexte použitia dashboardu pre daného užívateľa. Ak to je možné, je vhodné použiť zosumarizované dáta a výnimky.

Posledný z menovaných princípov je ale už v našej plnej réžii. Prostriedky na zobrazovanie dát ponúkneme užívateľovi v takej forme, aby spĺňali princíp najvhodnejšie použitých prostriedkov a útvarov. Pri tomto bode sa treba riadiť minimalizmom vo všetkých smeroch. Vypúšťanie prvkov je základnym nástrojom tvorby funkčného a príťažlivého prvku na dashboarde a tým aj dashboardu samotného.

1.2 História a vznik dashboardov

História tejto grafickej pomôcky nie je obzvlášť rozsiahla. Úlohu zobrazenia a získavania informácií spĺňa väčšina spomenutých technológií podobne ako dashboard a dané technológie sa vzájomne nevylučujú a tak sa používajú zároveň s dashboardami.

Podľa [2] bol predchodcom dashboardu balanced scorecard. Balanced scorecard predstavuje kolekciu kľúčových indikátorov podľa ktorých sa hodnotia a sledujú napríklad podniky. Balanced scorecard zaviedol do zobrazovania informácií zhustenie, sprehľadnenie a výber iba dôležitých ukazovateľov potrebných pre sledovanie subjektu. Tieto ukazovatele sú posudzované metrikami, ktoré upresňujú v akých jednotkách sa daná veličina meria, a aké sú obmedzujúce hodnoty, ktoré určujú, že daný stav je priaznivý.

Ďalším krokom k vytvoreniu dashboardu bol medzistupeň EIP (Executive Information System), čo je systém poskytujúci vedeniu firmy všeobecný prehľad o dianí vo firme na základe získaných dát z databáz [3]. Tento systém je prevažne určený vedúcim spoločností a nekladie dôraz na systém zobrazenia dát. Dáta sú dostupné cez navigáciu a zobrazené prevažne v tabuľkovej či

textovej forme. V niektorých prípadoch sa definícia dashboardu a EIP zlieva ako je možno pozorovať pri produkte spoločnosti Kbase.com[4]. Produkt tejto spoločnosti už používa názov dashboard pre rôzne grafické prvky ako napríklad budíky, ale zároveň používa navigáciu čo v prípade dashboardu podľa definície nie je prípustné, keďže by mali byť všetky informácie dostupné na jednej stránke, respektíve obrazovke.

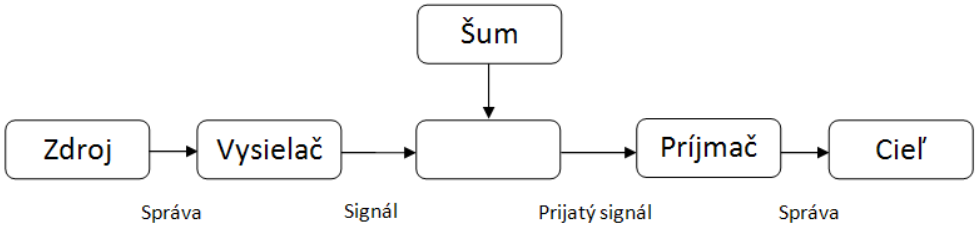
Ďalšou technológiou, ktorá mala vplyv a vývoj dashboardu, je Online Analytical Processing (OLAP) [4]. OLAP predstavuje koncept, v ktorom sa databáza alebo iný dátový sklad monitoruje ďalším serverom, ktorý má za úlohu vybrať najdôležitejšie dáta a následne ich vyhodnotiť. Zmienky o tomto koncepte možno nájsť už v roku 1963, ale formálne spracovaný bol až v roku 1993. Pre dashboardy je koncept OLAP nástrojom na získavanie potrebných informácií, ktorý zastrešuje aj data mining.

Tieto a iné technológie a koncepty, ako Business Process Management (BPM), slúžili ako podklady a východiskové technológie pre dashboardy. Na rozdiel od nich sa dashboard zaoberá intenzívne aj grafickou stránkou zobrazovaných informácií a spôsobom ich podania. Najväčším prínosom dashboardov je to, že zmenili doposiaľ nevyužité dátové sklady na použiteľné zdroje informácií, a preto je spôsob efektívneho sprostredkovania dát pre mnoho aplikácií neoceniteľný.

1.3 Uplatnené princípy

V tejto podkapitole si priblížime, aké grafické princípy by sa mali dodržiavať pri tvorbe dashboardov. Priemerný človek prijíma väčšinu informácií zo svojho okolia prostredníctvom zraku [5]. Z toho dôvodu budú oči hlavným cieľom každej informácie, ktorú sa nám niekto bude snažiť sprostredkovať. Preto je dôležité, aby grafická stránka dashboardu bola čo najjednoduchšia a pritom čo najpríjemnejšia na pohľad. Dashboard nesmie obsahovať farby a ich kombináciu, ktoré sú príliš výstredné. Musíme si uvedomiť, že užívateľ bude daný dashboard používať aj niekoľko hodín denne, takže dizajn komponent a dizajn samotnej stránky musí byť čo najprívetivejší. Podľa komunikačného modelu Shannon a Weaver z roku 1949 je možné každé sprostredkovanie informácií rozložiť na vysielač a prijímač [6]. Informácie sú prenášané médiom, napríklad médium je v prípade orálnej komunikácie vzduch, ktorý prenáša zvuk. Pri každej výmene informácie môže dôjsť k rušeniu prenosu šumom, napríklad pri orálnej komunikácii môže byť šumom hluk okolia. V našom prípade je tento šum prebytočná grafika použitá v dashboardoch. Odhliadnuc od toho, že tento komunikačný model je už dávno prekonaný, je uvedený princíp rušenia komunikácie stále faktom. Schéma tohto modelu je uvedená na obrázku 1.

[4] <http://www.kbase.com/dashboard.htm>

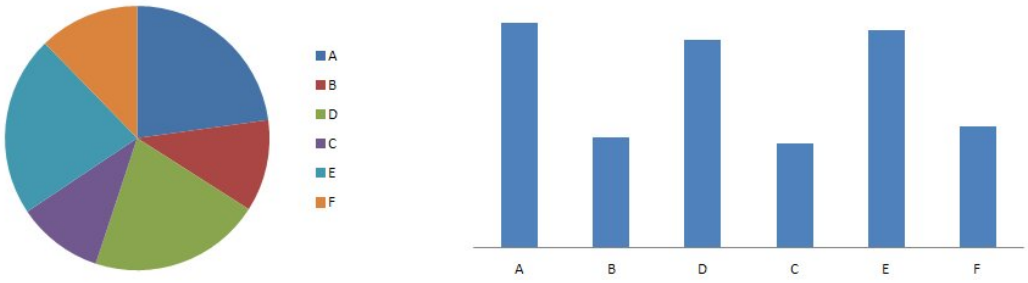


Obrázok 1: Shannon – Weaver komunikačný model (vlastné spracovanie podľa [6])

Niektoré spoločnosti, vytvárajúce dashboardy, sa snažia čo najviac priblížiť skutočnej palubnej doske. Priblíženie sa k realite v tomto prípade ale nie je žiaduce, keďže nám vnáša do dashboardu prebytočné grafické prvky a tým zvyšuje šum v prenášanej informácii. Ako príklad si môžeme uviesť použitie budíkov, podobným budíkom v aute, na zobrazenie splnenia cieľa, napríklad dosiahnutie určitého objemu v predajoch. Takéto budíky sú zobrazené na obrázku 2.

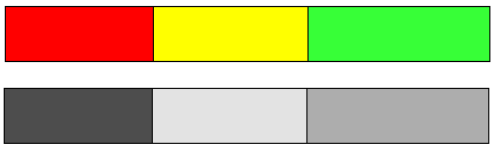
Obrázok 2: Budíky zobrazujúce stav ukazovateľov vo firme, prevzaté z [7]

Tieto budíky sú väčšinou problematicky čitateľné a zároveň je pre pozorovateľa ťažšie porovnať jednotlivé hodnoty medzi sebou, ak to je potrebné. Každý pozorovateľ má zároveň ťažkosti s plošným čítaním informácie, ako napríklad vyčítanie percentuálneho podielu z koláčového grafu alebo porovnanie veľkosti dvoch plôch. Na obrázku 3 je vidno, že pri stĺpcovom grafe nám nerobí veľké problémy zoradiť jednotlivé hodnoty podľa veľkosti. Naopak pri koláčovom grafe by sme s mali s touto úlohou problém (poradie od najmenšieho: C, B, F, D, E, A).



Obrázok 3: Ukážka lepšej čitateľnosti stĺpcového grafu oproti plošnému typu grafu

Často je potrebné v dashboardoch zobraziť stav nejakého ukazovateľa, napríklad stav dokončenia projektu, stav plnenia mesačného plánu alebo stav nejakej fyzikálnej veličiny. Pritom si musíme uvedomiť, že príliš veľká granularita stavov môže spôsobiť neprehľadnosť a zmätok. Zároveň treba myslieť aj na ľudí so zdravotnými problémami. Zvlášť na ľudí, ktorí trpia farbosleposťou. Preto treba obzvlášť dbať na výber farieb. Pre týchto ľudí sa najlepšie rozlišujú farby s rozdielnym stupňom intenzity. Ako príklad sa dajú uviesť stupne šedi. Keď prevedieme bežne používané signalizačné farby zelenú, červenú a žltú na stupne šedi, ktoré takto vnímajú farbu farboslepé osoby, tak je vidno, že aj človek s týmto zdravotným problémom nebude mať problém rozlíšiť jednotlivé stavy pomocou týchto farieb. Tento princíp je zobrazený na obrázku 4.



Obrázok 4: Prevedenie farieb na stupne šedi a tým simulácia videnia farboslepého človeka

Z toho dôvodu je použitie daných farieb na určenie stavu vhodné. Zároveň dokáže človek spoľahlivo rozoznať iba 5 stupňov šedi, preto je rozumné použiť maximálne päť farieb na rozlišovanie dát v grafoch.

Úroveň detailu je ďalším dôležitým princípom. Pre medzi denného obchodníka na burze nebude potrebné vidieť vývoj akcií komodity za desať rokov, ale za daný deň či týždeň. Splnenie tohto princípu musí byť ale v rámci tejto práce znovu ponechané na užívateľovi, keďže si dáta pre dashboard určuje sám.

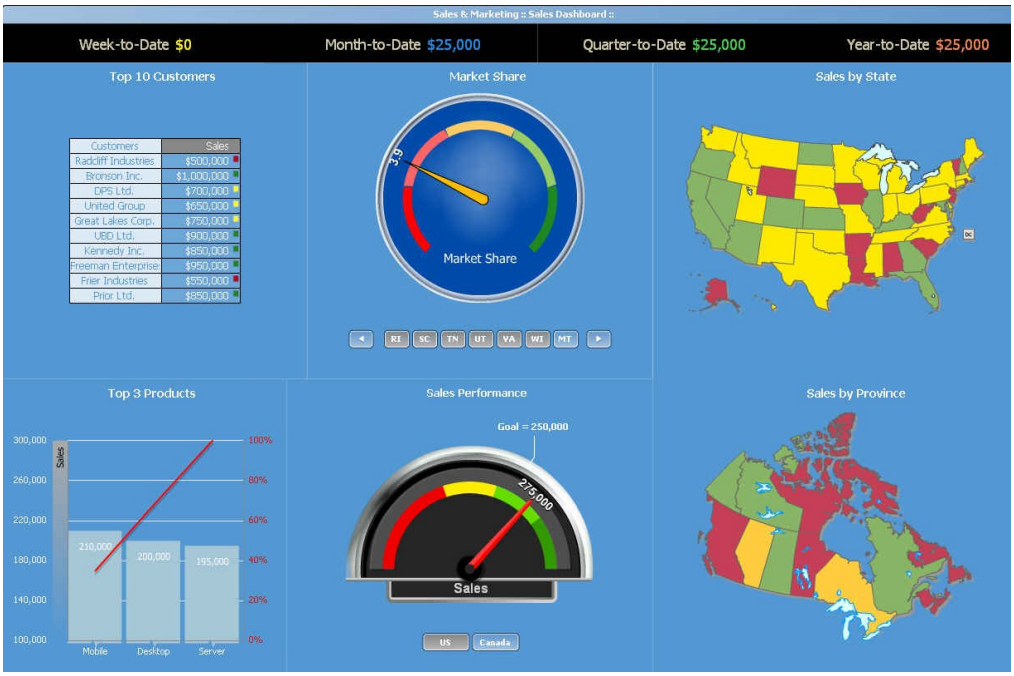
Väčšina dashboardov sa podľa [1] dopúšťa aspoň niekoľkých chýb z nasledujúceho zoznamu. Pri vytváraní dashboardu treba preto stále sledovať, či sme nejaké z daných pravidiel neporušili:

- prekročenie plochy jednej obrazovky,

- poskytnutie nedostatočného kontextu pre dáta,
- zobrazovanie dát do nepotrebnej hĺbky,
- použitie nedostatočnej mierky,
- použitie nesprávneho zobrazovacieho média,
- vkladanie nepotrebnej rozmanitosti,
- používanie zle navrhnutých zobrazovacích prostriedkov,
- nepresné zakódovanie kvantitatívnych dát,
- nepremyslene zvolené poradie dát,
- neefektívne zvýraznenie dôležitých dát (nezvýraznenie dôležitých dát) ,
- využívanie prebytočnej grafiky,
- zneužitie či nadbytočné použitie farieb,
- navrhnutie neatraktívneho zobrazenia dashboardu.

1.4 Príklad dashboardu s nedostatkami

V tejto kapitole si uvedieme príklad dashboardu, ktorý nespĺňa naše kritériá stanovené v predchádzajúcej kapitole.



Obrázok 5: Príklad dashboardu s nedostatkami, prevzaté z [8]

Dashboard uvedený na obrázku 5 má viacero nedostatkov. Nedostatky zahŕňajú:

- farebné pozadie,
- nedostatočné označenie údajov na budíkoch,
- použitie budíkov,
- nadbytočné formátovanie tabuľky,
- nadbytočné formátovanie grafu,
- mapy možno nahradiť tabuľkami.

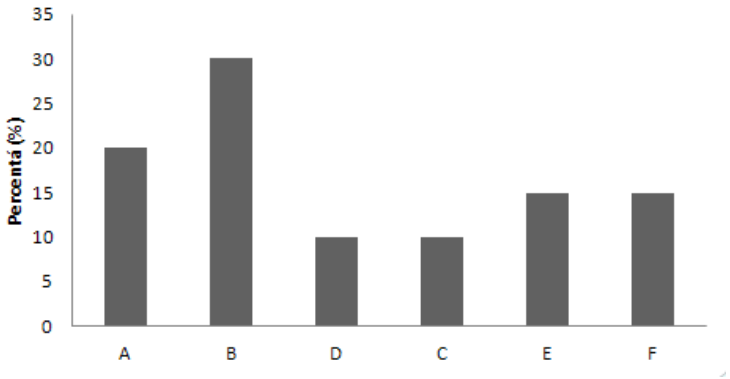
1.5 Grafy vhodné na prezentáciu informácií

Pri výbere typov grafov, ktoré použijeme v dashboarde, musíme dbať najmä na čitateľnosť a prípadnú možnosť čo najpresnejšieho porovnania dvoch alebo viacerých hodnôt. V našej aplikácii sa zameriame na zobrazenie celkového postupu (napríklad v projekte), klasického spojitého a nespojitého grafu a na zobrazenie jednotlivých hodnôt napríklad v tabuľkovej forme.

Zaužívaným typov grafu pre zobrazenie nejakého postupu je graf predstavujúci tachometer vo vozidle.

Obrázok 6: Rôzne typy budíkových grafov, prevzaté z [9, 10, 11, 12]

Tento typ grafu nie je vhodný z viacerých dôvodov. Čitateľnosť nie je vo väčšine prípadov ideálna, na niektorých chýba aj označenie jednotlivých úsekov na stupnici, ako vidno na obrázku 6. Pri zobrazení dvoch takýchto ciferníkov vedľa seba sa ťažšie určí, ktorá hodnota je väčšia. Na rozdiel od úsečok alebo stĺpcov, pri ktorých ľahko určíme, ktorá hodnota je väčšia, keď ich umiestnime pod seba alebo vedľa seba. Príklad takéhoto grafu môžeme vidieť na obrázku 7.

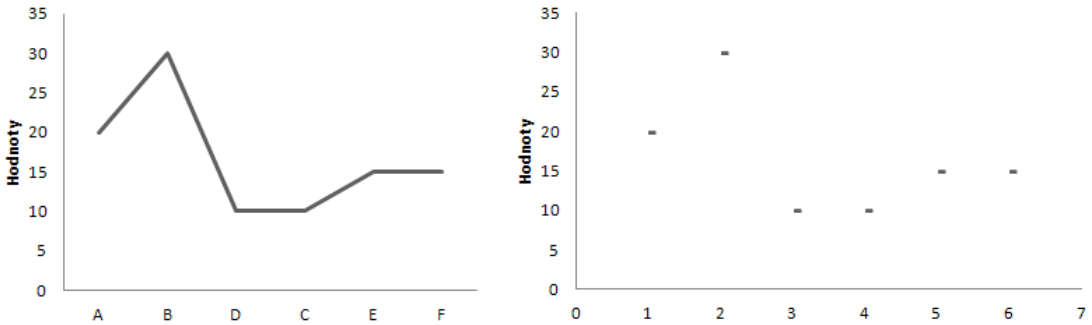


Obrázok 7: Zobrazenie porovnateľných hodnôt

Základným princípom je stále čitateľnosť, ktorú musíme brať stále v úvahu pri výbere typu grafu. Z uvedených dôvodov vyplýva, že danú informáciu, zobrazenie postupu, je najlepšie zobraziť ako úsečku. Na nej treba zobraziť minimálne hodnotu, ktorú keď dosiahneme tak je daná úloha splnená. Tento typ úsečky môže aj „pretiecť“, a to vtedy, keď daný cieľ prevýšime, napríklad predáme zákazky za väčšiu hodnotu, ako sa plánovalo. Tento graf bude čitateľný, ľahko porovnateľný a zároveň zaberá čo najmenej miesta, čo je taktiež dôležité, keďže miesto na obrazovke je obmedzené. Takýto typ grafu sa nazýva „bullet“ graf a príklad takéhoto grafu je na obrázku 8.

Obrázok 8: Bullet graf [1]

Na zobrazenie spojitého a nespojitého grafu použijeme jednoduché grafy bez zbytočných grafických prvkov. Príklady takých grafov vidno na obrázku 9.



Obrázok 9: Príklad spojitého a nespojitého grafu

Na zobrazenie informácií, napríklad na zoznam osôb a ich príjmov spolu s rodným číslom, je najlepšie použiť jednoduchú tabuľku. Ako vidno na obrázku 10, tabuľka je jedným z najprehľadnejších spôsobov ako zobraziť výpočet hodnôt bez nadbytočnej grafiky. Jedinými grafickými prvkami sú medzery a jedna úsečka oddeľujúca názvy stĺpcov od hodnôt.

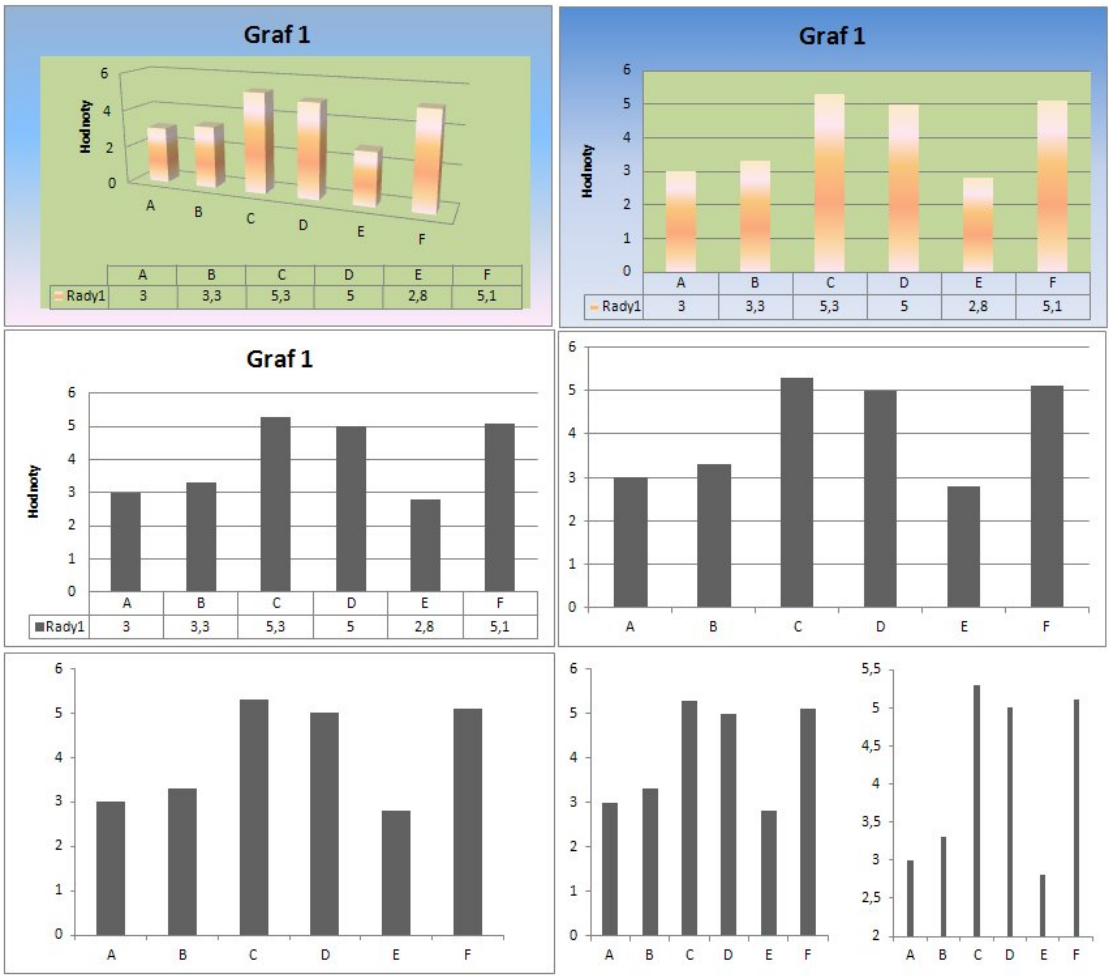
Meno	Priezvisko	Vek	Príjem (CZK)
Ján	Novák	35	32 500
Monika	Novotná	28	25 700
Ondrej	Stodola	40	41 200

Obrázok 10: Jednoduchá tabuľka bez zbytočných štýlov

Týmto sme pokryli hlavné grafy potrebné na zobrazenie údajov v dashboardoch. Dané grafy môžu vo výsledku obsahovať prídavné informácie, ak to je potrebné, ako dodatočný popis pre doplneniu kontextu.

1.6 Postup vytvárania vzhľadu dashboardu

Pri vytváraní celkového dashboardu treba dbať na to, aby komponenty boli navrhnuté čo najefektívnejšie, ako priestorovo, tak aj s použitím čo najmenšieho počtu dizajnových prostriedkov. Z toho sa dá podľa [1] odvodiť jednoduchý, ale účinný spôsob vytvárania dashboardu ako aj jednotlivých prvkov. Vyberieme postupne východiskové komponenty, ktoré sú vhodné pre zobrazenie našich dát, a postupne budeme uberať nadbytočné prvky, až kým nezostaneme s objektom, ktorý reprezentuje dané dáta čo najjednoduchšie. Takúto transformáciu možno vidieť na obrázku 11. Z trojdimenzionálneho grafu postupne urobíme v prvom kroku dvojdimenzionálny graf, v druhom odoberieme nadbytočné farby, v treťom odstránime duplicitné informácie. V tomto príklade sa zameriame na porovnanie hodnôt a nie na odčítanie presnej hodnoty. Keď potrebujeme odčítať presnú hodnotu nejakého ukazovateľa, tak je vo väčšine prípadov vhodné použiť jednoduchú tabuľku hodnôt. V ďalšom kroku preto odstránime nadbytočnú mriežku, zmenšíme zobrazovaciu plochu grafu a odstránime ohraničenie plochy grafu, keďže prázdne miesto je dostatočným oddeľovacím prvkom. V poslednom kroku posunieme počiatok y-osi pre zvýraznenie rozdielov medzi sledovanými ukazovateľmi.



Obrázok 11: Postupné odoberanie nadbytočných grafických prvkov z grafu

Pri vytváraní dashboardu treba dbať na výber informácií rovnako, ako na dizajn. Ako sme už spomenuli, používanie výnimiek a sumárov je vhodným prostriedkom na obmedzenie zobrazovaných dát. Tieto nefiltrujú iba dáta, ale aj zmenšujú počet potrebných ukazovateľov, čím šetria vzácne miesto na obrazovke.

Podľa Edward R. Tufteho [13] sa môžeme riadiť aj takým jednoduchým princípom, akým je „data-ink ratio“. To znamená, aký je pomer atramentu (pixelov) použitého na zobrazenie dát na určitej ploche vzhľadom na atrament použitý na dizajnové prostriedky ako rozdeľovacie čiary, nadpisy, logá. Čím je tento pomer väčší v prospech dát, tým lepšie. Kroky na dosiahnutie dobrého data-ink ratio sú podľa S. Fewa [1], ktorý transformoval Tufteho tvrdenie, nasledujúce:

1) Redukuj pixely nereprezentujúce dáta:
   a) vypusť nepotrebné ne-dátové pixely úplne,
   b) znevýrazni a reguluj ne-dátové pixely, ktoré zostali.

2) Zvýrazni dátové pixely:

a) eliminuj všetky nepotrebné dátové pixely,

b) zvýrazni najdôležitejšie pixely, ktoré zostali.

V príklade, zobrazenom na obrázku 11 sme splnili body 1a, 1b a 2a. Podľa použitia grafu by sa v kroku 2b pridali popisy a názov grafu, ak z kontextu dashboardu nebude na prvý pohľad zrejmé, o aké dáta sa jedná.

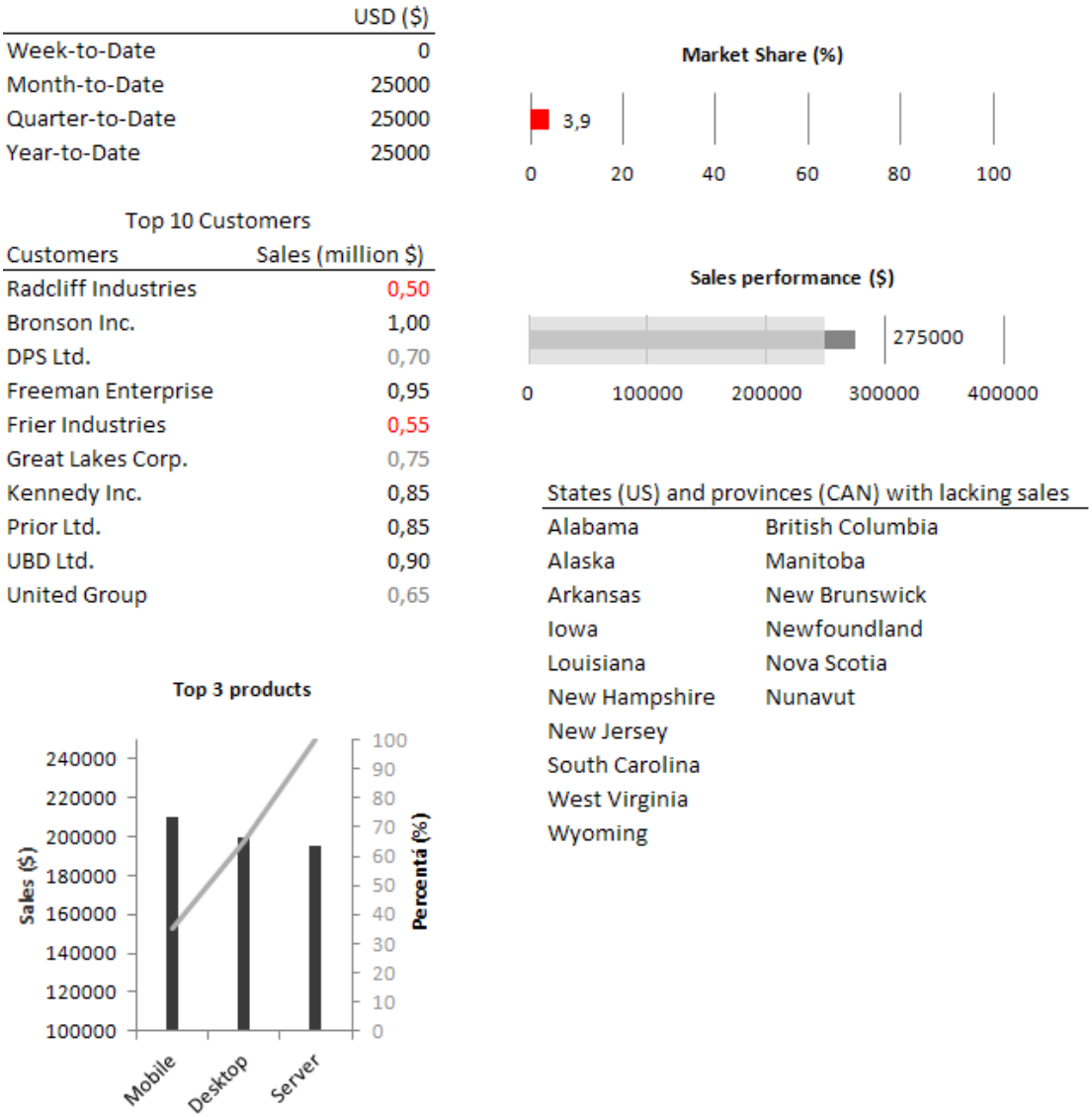
1.6.1 Funkcionalita

Základnou funkcionalitou dashboardu je schopnosť si daný dashboard prispôsobiť. Užívateľ potrebuje mať voľnú ruku nad rozmiestnení prvkov, poradím a zobrazovanými dátami. Túto flexibilitu ponúkajú dynamické užívateľské prvky na stránke. Filtrovanie je zabezpečené nastavením týchto komponent zo strany užívateľa. Týmto má užívateľ kontrolu nad dizajnom, poradím a dátami, ktoré budú zobrazené jednotlivými komponentmi.

Pri dizajne dashboardu si treba uvedomiť, pre aké dáta sa daný komponent navrhuje. Keď sa jedná o prevažne statické dáta, tak je interval na obnovu údajov pomerne dlhý a z pohľadu výkonnosti aplikácie sa touto stránkou veci netreba zaoberať do veľkej hĺbky. Ak sa jedná ale o prúdové dáta, ako v tomto prípade, tak je potrebné sa zamyslieť nad výkonom aplikácie.

1.6.2 Správne použité grafy v dashboardoch

V tejto kapitole pretransformujeme spomenutými postupmi graf z kapitoly 2.1.5 do takého formátu, aby čo najlepšie vyhovoval našim stanoveným kritériám. Postup je totožný s postupom uvedeným v kapitole 2.2. Postupne sa odoberajú všetky nadbytočné prvky, až kým nezostane iba dátová kostra. V tejto kostre postupne zvýrazňujeme najpodstatnejšie dáta. Výsledok tejto transformácie je uvedený na obrázku 12.



Obrázok 12: Príklad modifikovaného dashboardu z obrázku 5 so správne aplikovanými grafickými princípmi

Do dashboardu z príkladu na obrázku 12 by bolo vhodné doplniť vysvetľujúce nadpisy k jednotlivým grafom. Keďže tieto chýbajú, alebo sú príliš stručné na to, aby sme pochopili o aký typ údajov ide, už v pôvodnom spracovaní zobrazenom na obrázku 5, tak tento nedostatok nie je možné skorigovať ani v obrázku 12. Preto má tento dashboard ešte vždy jeden nedostatok spomenutý v kapitole 2.1.14, a to ten, že sa neposkytuje dostatočný kontext pre zobrazené dáta.

2 Prúdové spracovanie dát

V predchádzajúcich kapitolách sme naznačili, že nami spracované dáta nie sú statické, ale jedná sa o kontinuálny prúd dát. Andrew McGregor charakterizuje v [14] prúd dát ako *m* elementov z univerza o veľkosti *n,* napríklad:

$<x_1, x_2, x_3, \ldots, x_n> = 3, 5, 4, 9, 8, 3$

K prúdovým dátam je možné pristúpiť iba raz, a to v dobe, keď sa spracujú systémom. Z toho vyplýva, že dáta je potrebné zaznamenať, ak ich systém má neskôr vyhodnotiť. McGregor zdôrazňuje, že z toho plynie jeden dôležitý poznatok a to ten, že tvoríme záznamy teoreticky nekonečného prúdu dát v konečnom priestore pamäti počítača. Taktiež uvádza, že každý element sa musí spracovať dostatočne rýchlo na to, aby pri vstupe ďalšieho elementu bol systém pripravený na jeho spracovanie. V pokračovaní tejto kapitoly primárne čerpám zo zdroja [15].

Existujú dva prístupy spracovania prúdových dát:

- techniky založené na dátach (data-based techniques),
- techniky založené na úlohách (task-based techniques).

Techniky založené na dátach sa zameriavajú na sumarizovanie celého vzorku dát alebo na výber podmnožiny z prijímaného prúdu dát, ktorá sa spracuje. Prístup, zaoberajúci sa sumarizovaním celej vzorky dát, používa techniky:

- vzorkovanie (angl. sampling),
- load shedding,
- sketching.

Druhý prístup, vyberajúci a spracúvavajúci podmnožinu z prúdu dát, používa na rozdiel od prvého prístupu techniky:

- dátové štruktúry - synopsie,
- agregačné techniky.

2.1 Dátové techniky spracovania prúdu dát

V tejto kapitole si priblížime jednotlivé možné prístupy spracovania prúdu dát. Spomenuté techniky sú aplikované nad podmnožinou vstupných dát, väčšinou obmedzených technikou posuvné okno

(sliding window). Posuvné okno možno podľa [14] definovať nasledovne, pričom w je veľkosť okna, x_i jednotlivé hodnoty z prúdu dát: $W = <x_{j-w+1}, ..., x_j>$.

2.1.1 Vzorkovanie

Pri vzorkovaní sa určí pravdepodobnosť, či sa dátová položka spracuje, alebo sa spracovanie vynechá. Vzorkovanie je jedna z najstarších metód výberu podmnožiny dát z dátového prúdu. Využívajúca rôzne algoritmy určujúce, ktorá položka sa má spracovať a ktorá nie. Najjednoduchší prístup je vynechanie každej n-tej položky, spracovanie každej n-tej položky, alebo časové obmedzenie spracovania, ktoré nehľadí na počet dát, ktoré pretečie daným prúdom dát. Podľa [14] sa vzorkovanie dá taktiež uskutočniť náhodným výberom položky na spracovanie. Postup je nasledovný:

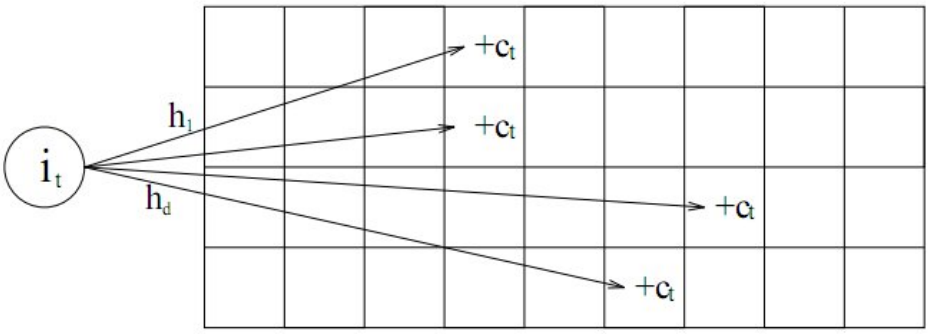
1. Pre každé x_i patriace do prúdu dát, priradíme náhodne hodnotu $v_i \in <0,1>$.
2. V okne $<x_{j-w+1}, ..., x_j>$ navráť hodnotu x_i s najmenšou hodnotou v_i.
3. Udržuj množinu prvkov z posuvného okna s najmenšou v hodnotou.

2.1.2 Load shedding

Load shedding pracuje na podobnom princípe ako vzorkovanie. Podľa Nesime Tatbul [16] sa load shedding používa najmä v prípade, ak je počítač zaisťujúci spracovanie dát preťažený a nedokáže spracovať ďalšie prichádzajúce dáta. Preto sa na určitú dobu zastaví spracovanie dát, kým sa nedokončí spracovanie predchádzajúceho bloku dát. Historicky pochádza tento prístup z odpájania častí elektrickej siete, keď sa v sieti nenachádza dostatok elektrickej energie na pokrytie dopytu a hrozil by kolaps elektrickej siete. Rovnaký prístup sa uplatňuje v niektorých aplikáciách, či serveroch, ktoré majú obmedzenú veľkosť bufferu použitého na spracovanie dát. Ďalšie dáta sa prestanú prijímať, kým sa v bufferi neuvoľní miesto. Z toho pohľadu sa táto technika používa v prípade, keď máme obmedzené prostriedky na spracovanie dát, či sú to obmedzenia pamäťové, výkonnostné alebo obmedzenia týkajúce sa obmedzenej prenosovej rýchlosti pri použití siete pri procese spracovania dát.

2.1.3 Sketching

V [17] sa dočítame, že doposiaľ spomenuté algoritmy vynechávali časti prúdových dát a tým sa výsledok spracovania skresľuje. Sketching je algoritmus pracujúci so všetkými dátami z prúdu, jednotlivé spracované položky sa ale neukladajú. Sketch sa dá docieliť lineárnou transformáciou prúdu dát uskutočnenou multiplikáciou toku dát s maticou. Ako príklad sketchingu možno uviesť Count Min Sketch. Systém podľa [18] vytvorí obraz streamu dát pomocou hashovacích funkcií. Tento proces je načrtnutý na obrázku 13.



Obrázok 13: Grafické znázornenie aktualizácie viacrozmerného poľa slúžiaceho na záznam hodnôt z prúdu dát pri postupe CM Sketch [18]

2.1.4 Dátové štruktúry - synopse

Synopse sú podľa [18] dátové štruktúry používané ako náhrada za konkrétne dáta. Cez dátové prúdy môže v dnešnej dobe prechádzať set dát o veľkosti až niekoľko gigabajtov. Kopírovať tieto dáta nie je vo väčšine prípadov možné a vykonávanie operácií nad takým množstvom dát je v čase zápisu dát problematické. Preto sa v mnohých prípadoch používajú synopsis data structures ako náhrada týchto dát. Šetria miesto a sú často reprezentované sumárom vlastností dátového prúdu. Takéto dátové štruktúry možno definovať nasledovne z [19]:

„Dátová, $f(n)$ synoptická štruktúra pre triedu dotazov Q, je dátová štruktúra umožňujúca poskytnúť (presné alebo približné) odpovede na dotazy z triedy Q. Táto dátová štruktúra zaberá $O(f(n))$ priestoru pre dáta o veľkosti n, kde $f(n) = o(n^{\epsilon})$, pre nejakú konštantu $\epsilon < 1$.“

2.1.5 Agregačné techniky

Aggregácia je výpočet štatistických vlastností prichádzajúceho prúdu dát ako napríklad medián spracovaných dát. Týmito prístupmi sa charakterizuje tok dát. Táto technika sa neosvedčila pri dátach s veľkými výchylkami.

2.1.6 Jednoduché náhodné vzorkovanie

Jednoduché náhodné vzorkovanie (Simple Random Sampling) je najjednoduchšou vzorkovacou metódou. Pre každú vzorku sa generuje náhodné číslo, ktoré určí, či sa daná vzorka spracuje, alebo zahodí. Jedná z možných implementácií je generovať čísla v určitom rozmedzí a ak daná hodnota presiahne určitú hranicu, tak sa dáta spracujú [24]. Táto hranica môže byť definovateľná užívateľom, čím mu umožníme ovplyvniť aké množstvo dát sa spracuje a aké množstvo sa zahodí.

3 Architektúra riešenia

V tejto kapitole si postupne rozoberieme požiadavky na aplikáciu, jej hlavné časti, a prístupy, ktoré môžu byť použité pri implementácii nášho riešenia.

3.1 Požiadavky na systém

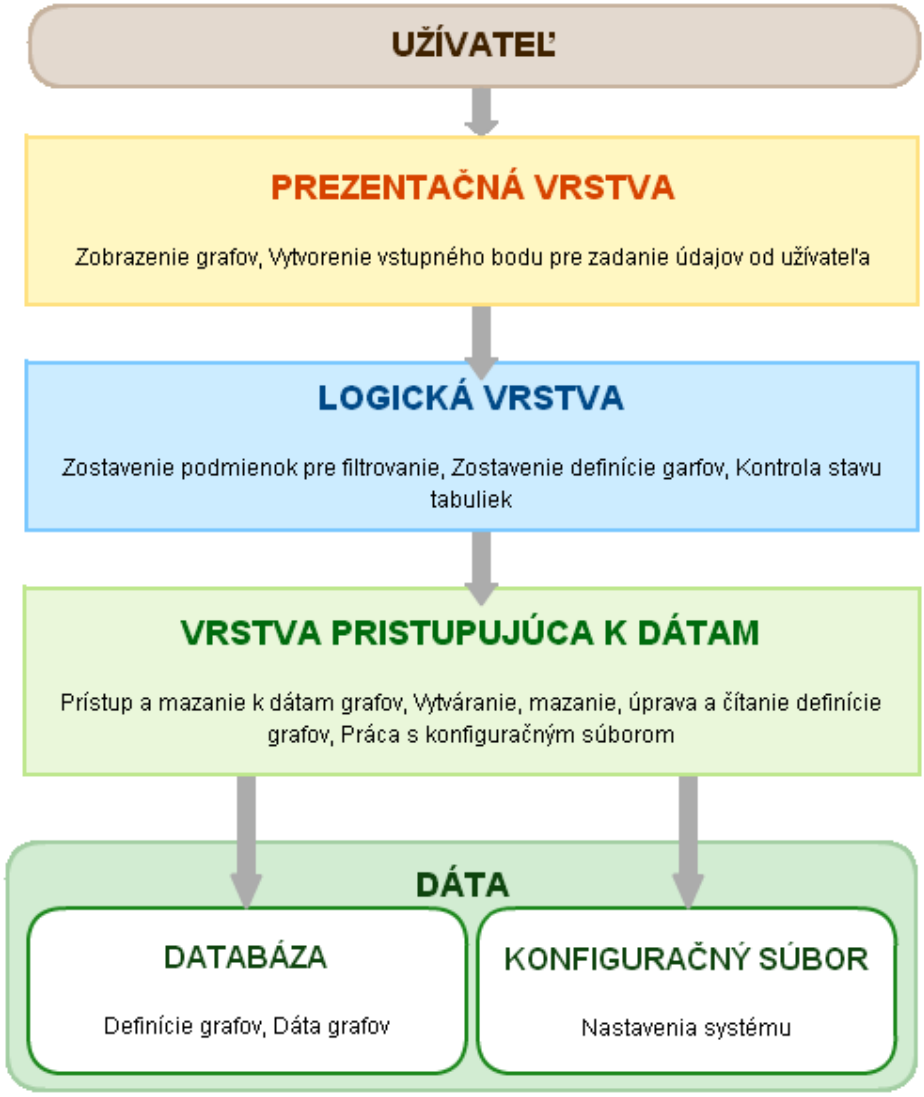
Systém by mal dokázať zvládať sledovať dáta v reálnom čase a zároveň je zo zadania zrejmé, že náš systém musí byť schopný sledovať všeobecnú databázu, takže musí byť dynamicky prispôsobiteľný k akejkoľvek databáze a nesmie byť závislý od jej štruktúry. Systém musí taktiež pracovať nad databázou typu Microsoft SQL. Užívateľ si bude môcť vybrať, nad ktorými dátami sa vytvoria grafy, ktoré budú tvoriť základné kamene dashboardu. Sledované dáta sú obmedzené dátovými typmi na tie, ktoré má zmysel sledovať, ako napríklad reťazce a čísla. Podmienky, vytvárané užívateľom na sledovanie dát musia byť jednoducho definovateľné a nesmú užívateľovi dovoliť vytvorenie neplatnej podmienky. Dashboard je pre užívateľa najprínosnejší, keď ho bude môcť sledovať na rôznych zariadeniach a nebude musieť inštalovať dodatočný softvér. Zároveň je nutné, aby aplikácia bola prístupná viacerým užívateľom v ten istý okamih, keď bude napríklad potrebné dáta konzultovať v tímovom prostredí. Aplikácia musí byť dostatočne rýchla a nenáročná na to, aby nespomaľovala systém a neovplyvňovala bežné spracovanie dát. Z toho vyplýva, že hlavné požiadavky na systém sú nasledujúce:

- práca s Microsoft SQL databázou,
- práca nad neznámou databázovou štruktúrou,
- práca nad prúdom dát,
- zobrazenie dashboardu v prehliadači,
- tvorba podmienok cez užívateľské rozhranie,
- vytvorené podmienky musia byť správne,
- systém nesmie neúnosne zaťažiť hostiteľský systém.

V nasledujúcich podkapitolách budeme riešiť požiadavky na finálny systém, ktoré boli definované v tejto kapitole.

3.2 Stavba aplikácie

Prvým krokom je určenie všeobecnej stavby aplikácie. Aplikácia bude využívať vrstvovú architektúru, ktorá je pre našu aplikáciu zjednodušene znázornená na obrázku 14. Tento prístup je vhodný najmä z pohľadu možného rozšírenia riešenia, alebo jeho adaptácie na iné zdroje dát. Pri vrstvovej aplikácii je nutné rozšíriť, alebo zameniť iba vrstvu pristupujúcu k dátam a nemusia sa robiť zásahy do ostatných vrstiev, ktoré zabezpečujú zobrazovanie údajov a ich spracovanie, keď sa bude napríklad meniť typ obsluhovanej databázy. Vrstvová architektúra má výhodu aj v tom, že pri vhodnom návrhu je možné dané komponenty znovu použiť v iných projektoch. Pri dodržaní zásady správneho návrhu a implementácie, je zároveň kód prehľadný a ľahko porozumiteľný aj pre iných vývojárov.



Obrázok 14: Zjednodušená architektúra aplikácie

Úloha spracovania prúdu dát je pamäťovo a výkonnostne náročná, a preto je dôležité sa zamyslieť nad architektúrou riešenia už v počiatočných fázach projektu. Naše riešenie musí byť zároveň dostatočne flexibilné, aby bolo možné riešiť problémy s výkonom relatívne jednoducho na jednom počítači. V ďalšej kapitole si rozoberieme, prečo nám bude práve zo spomenutých dôvodov vyhovovať model klient/server.

3.3 Klient/server model

Model klient/server je v dnešnej dobe veľmi rozšírenou konfiguráciou aplikácie. Má to viacero dôvodov. Výkon aplikácie možno jednoducho ovplyvňovať zmenou hardvéru jedného servera, pričom nárast výkonu pocíti každý klient bez toho, aby sa zmenila konfigurácia klientskych systémov. Taktiež sa väčšina úloh vykonáva priamo na serveri a nezaťažujú sa koncové zariadenia. Týmto modelom možno poskytnúť rovnaký komfort interaktívneho spracovania obsahu užívateľom na veľkej škále zariadení. Zároveň je ideálne pre užívateľa mať stály prístup k dashboardu.

Tieto a iné dôvody podporujú použitie modelu klient/server. Užívateľ tak musí mať iba prístup ku klientovi, ktorý má prístup k serveru.

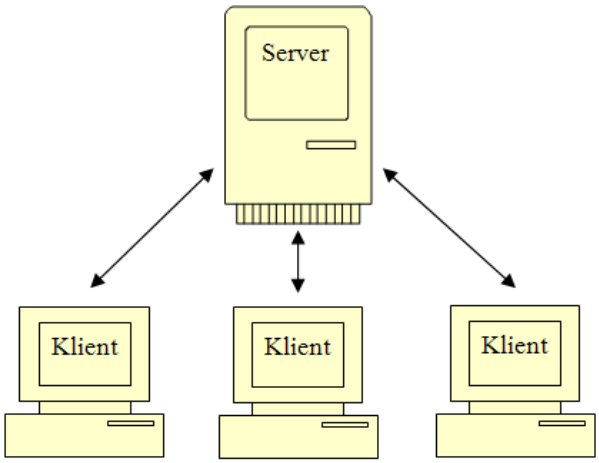
3.3.1 Definícia modelu klient/server

Pre lepšie pochopenie chodu aplikácie si definujme tento model podrobnejšie. Podľa definície, ktorú nájdeme v [20], je klient/server model distribučný počítačový model, v ktorom si klientske aplikácie vyžiadajú služby od serverových procesov. Klientske a serverové procesy bežia zvyčajne na rozdielnych počítačoch, ktoré sú spojené počítačovou sieťou. Klientska aplikácia je proces, ktorý vysiela správy cez počítačovú sieť. Tieto správy žiadajú od servera vykonanie niektorých akcií, ako napríklad vyhľadanie záznamu o zákazníkovi v databáze, alebo o navrátenie súboru, či jeho časti, ktorá je uložená na disku servera. Klient spracúva navrátené údaje, zobrazí ich, prípadne ich uloží na lokálne úložisko. Druhý koniec tohto modelu, proces servera, čaká na požiadavky od klientskych procesov, posielaných cez počítačovú sieť. Server po doručení správy danú úlohu vykoná. Typické úlohy prevádzané serverovým procesom je spustenie dotazu nad databázou a navrátenie výsledku, alebo prečítanie a navrátenie súboru z diskového priestoru servera. Procesy servera bežia zvyčajne na výkonných počítačoch. Príklad klient/server aplikácie môže byť informačný systém banky, ktorý umožňuje úradníkovi pristúpiť k údajom o zákazníkovi v centrálnom dátovom sklade banky. Pri tomto príklade je celý prístup vykonaný z klientskeho počítača, ktorý okrem iného môže vykonať kontrolu vstupných údajov, zadaných úradníkom. Klientsky počítač po získaní údajov od serveru dané údaje zobrazí. Jednoduchá klient/server konfigurácia je zobrazená na obrázku 15.

V [20] sa ďalej hovorí o tomto modelu ako o rozšírení objektového respektíve modulárneho programovania, v ktorom sa rozsiahle programy rozkladajú do menších štruktúr s presne

definovaným rozhraním. Výhodou tohto modelu je obsluha viacerých klientov, čo nám pri tvorbe dashboardu pomôže obslúžiť viacero koncových zariadení. Jednou z ďalších výhod použitia modelu klient/server je možnosť jednoduchého rozšírenia výpočtového výkonu serverovej časti modelu a tým zaručenie dlhodobej včasnej obsluhy klientskych požiadaviek aj pri náraste počtu klientov.

Problémom tohto prístupu môže byť výpadok serveru, počas ktorého žiaden z klientov nie je schopný získať dáta. Krátkodobý výpadok je možné premostiť použitím starších dát, ktoré sú uložené na strane klienta, ak sa nejedná o dáta aktualizované v krátkych intervaloch. Druhým problémom môže byť prenosová rýchlosť počítačovej siete. Tento problém je potrebné odstrániť zvýšením šírky prenosového pásma, alebo obmedzením prenášaných dát. V našom prípade použijeme predovšetkým postup zmenšenia prenášaných dát, keďže je väčšinou problematické rozšíriť dostatočne prenosové pásmo.



Obrázok 15: Model klient/server

3.3.2 Dynamický dashboard na architektúre klient/server

V našom prípade znamená voľba modelu klient/server pre dynamický dashboard to, že sa na serveri bude nachádzať databáza aj logika na predspracovanie grafov. Týmto spôsobom sa všetky náročné procesy budú prevádzať na serveri a klient bude iba zobrazovať spracované dáta. Aplikácia bude dashboard prezentovať formou HTML dokumentu, ktorý bude zobrazený klientskym prehliadačom. Tento model implikuje použitie webových technológií.

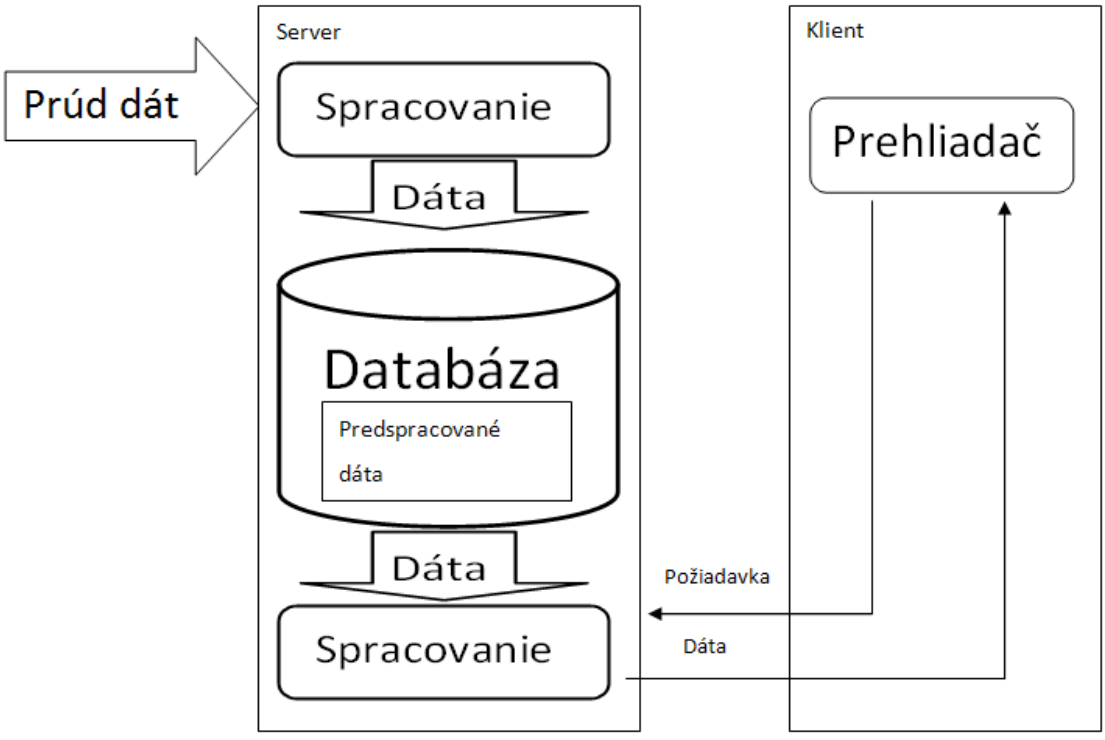
Pre dashboard je aplikácia postavená na webových technológiách najlepšou voľbou pretože užívateľ môže pristúpiť po vhodnej konfigurácii serveru k dashboardu z ktoréhokoľvek miesta na svete a zároveň je webový prehliadač dostupný na všetkých platformách, či sa rozprávame o operačnom systéme Linux, Windows, alebo o mobilných platformách akou je Android alebo iOS.

Takto je užívateľ schopný pristúpiť k daným údajom kedykoľvek, čo predstavuje určite veľkú výhodu ak sa jedná o dáta, ktoré je treba monitorovať s vysokou frekvenciou.

3.3.3 Tok dát na architektúre klient/server

Keďže už máme zvolený model, tak si musíme pripomenúť jeden z cieľov našej implementácie. V našom prípade treba dbať na rýchlosť spracovania dát. Spomalenie aplikácie môže nastať vo viacerých miestach. Jedno z najkritickejších miest je prenos dát. V aplikácii musí byť prenos dát obmedzený na minimum, keďže sa vopred nevie, aké široké dátové pásmo bude aplikácia monitorovať. Aby sme túto slabosť v systéme odstránili, tak je potrebné spracovať a vyfiltrovať dáta čo najbližšie k zdroju. Ak sa dáta z dátového prúdu predspracujú už pri čítaní, tak odpadá nutnosť duplikovania dát a ich ďalší prenos do iných vrstiev aplikácie. Prenášať sa bude tým pádom už len zlomok dát potrebných na vytvorenie dashboardu a zobrazenie iba tých dát, ktoré užívateľ na svoju prácu potrebuje. Preto je dobrou voľbou spracovať dáta už priamo v databáze, keďže sa pôvodné dáta ukladajú priamo do SQL tabuliek. Odpadá tak presun a analýza celého dátového toku mimo databázu.

Pre najlepší návrh treba zistiť, ako budú prúdiť dáta v systéme. Obrázok 16 zobrazuje hlavný prúd dát v našej aplikácii.



Obrázok 16: Jednoduchý náčrt toku dát v architektúre

Pri dátach, ukladaných do SQL tabuliek, je jedným zo spôsobov filtrácie a získavania dát, použitie triggerov. Výhodou použitia triggerov je najmä v ich rýchlosti. Alternatívou k triggerom by bolo použitie uložených procedúr. Problém s týmto prístupom by mohla predstavovať údržba týchto procedúr pri zmenách štruktúry v databáze. Keďže je trigger priamo zviazaný s tabuľkou, tak prípadné odstránenie tabuľky odstráni aj trigger. Takto nebudú na serveri vznikať prebytočné dáta, ktoré v budúcnosti už nevyužijeme. Ďalšou možnosťou by bolo sledovať zmeny v tabuľkách kódom z vrstvy aplikácie. Táto možnosť v našom prípade nie je vhodnou voľbou, pretože spracovanie dát by trvalo dlhšie a pri prúdových dátach je rýchlosť spracovania jedným z najdôležitejších faktorov, ktoré treba vziať v úvahu. Do tohto modelu možno zaviesť optimalizácie na urýchlenie spracovania dát vo forme vzorkovania. Prístupy k vzorkovaniu sú uvedené v kapitole 3.1.

3.3.4 Trigger

Trigger je podľa definície z [21] podobným prvkom ako uložená procedúra. Rozdiel je v uložení triggera, keďže je trigger priamo spätý z databázovou tabuľkou. Trigger sa spustí podľa konfigurácie zakaždým, keď sa prevedie napríklad insert, update alebo delete príkaz nad danou tabuľkou. Triggery sa používajú predovšetkým na zaistenie integrity databázy. Pravidlá integrity pre danú databázu sú dané aplikáciu, pre ktoré sú vytvorené.

Trigger je zjednodušene povedané kód, ktorý je spustený zakaždým keď je splnená podmienka spustenia. Napríklad nasledovný trigger sa spúšťa pri príkazoch update a insert:

```
AFTER INSERT, UPDATE
AS
DECLARE @Test1 nvarchar(MAX), @Test2 nvarchar(MAX)

 IF (SELECT Test1 FROM inserted) like '%1%'
       IF (SELECT Test1 FROM inserted) like '%2%'
        BEGIN

               --set local variables
               SET @Test1 = (SELECT Test1 FROM inserted)
               SET @Test2 = (SELECT Test2 FROM inserted)

               -- write to archive
               INSERT TestTrigger ( TestId, TestData, TestQuery) VALUES( '1', @Test1,
@Test2)
        END
```

Pri tomto jednoduchom príklade vidno, že podmienky na spustenie triggera možno zanoriť (testovanie na výskyt znakov 1 a 2 v reťazci) a trigger možno nastaviť tak, aby sa spúšťal nielen pri jednej podmienke, ale pri viacerých (update a delete v tomto prípade). Keď sú dané podmienky splnené, tak je možné spustiť kód obsluhujúci daný prípad (v tomto prípade sa celý testovaný reťazec uloží do tabuľky TestTrigger).

Ako vidno triggre sú dostatočne flexibilné pre náš účel. Z uvedeného príkladu vyplýva, že základom každého triggera je správne definovaná podmienka. Toto je hlavná časť triggera

definovateľná užívateľom. Ako bude daná podmienka zostaviteľná užívateľom, je opísané v ďalšej kapitole.

3.4 Dynamická tvorba podmienok

Pred konečným návrhom si treba určiť, čo daný systém umožní užívateľovi z pohľadu vytvárania podmienok, pretože vytvorená podmienka je zdrojom dát pre grafy, ktoré tvoria pre zmenu základ dashboardu. Vytvorením podmienky vytvárame dynamicky štruktúru triggera. Podmienka bude poskladaná dialógmi obmedzujúcimi možnosti užívateľa iba do tej miery, aby vytvorená podmienka bola platná. Po vytvorení grafov pomocou podmienok, bude užívateľ môcť sledovať údaje na dashboarde, ak spracované údaje v dátovom toku budú tejto podmienke vyhovovať.

3.4.1 Definovateľné parametre podmienok

V každej podmienke je nutné definovať jej typ. Tento závisí od dátového typu sledovaných dát. Typ dát je v tomto prípade obmedzený na reťazce a čísla, keďže pracujeme nad databázou. Ostatné typy, ako je dátum a čas, či booleovská hodnota, nie sú pre náš účel dôležité. Prípadné sledovanie booleovskej hodnoty je jednoducho implementovateľné dodatočne alebo sa použije dátový typ string.

Typ dát

Typ dát sa určí podľa vybraného atribútu z databázovej tabuľky. Tento bude možné vybrať z dialógu potom, ako sa určí databáza a tabuľka, s ktorou bude daná podmienka previazaná.

Typ podmienky

Keď sa jedná o číselnú hodnotu, tak je možné určiť či sa táto rovná alebo nerovná nami zadanej hodnote, či je menšia (menšia a rovná) alebo väčšia (väčšia a rovná). Ak sa jedná o sledovanie reťazca, tak užívateľ bude mať nasledujúce možnosti. Daný reťazec sa bude môcť porovnávať na rovnosť, nerovnosť, či je daný reťazec prefixom, sufixom porovnávaného reťazca z prúdu dát, alebo či sa jedná o pod-reťazec. Po poskladaní podmienky bude môcť užívateľ zadať ďalšiu podmienku, vytváranú rovnakým spôsobom.

Spôsob sumarizovania nameraných hodnôt

Po vytvorení podmienky, ktorá slúži na získavanie a filtrovanie údajov, bude môcť užívateľ nastaviť ako sa dané dáta budú zaznamenávať. Bude si môcť zvoliť interval, po ktorom sa vytvorí suma výskytov. Tento bude špecifikovaný časovým intervalom (napríklad každú hodinu) alebo počtom

položiek prechádzajúcich cez prúd dát. Prvé údaje budú dostupné po uplynutí tejto doby alebo po spracovaní určeného počtu položiek.

Typ grafu

Ďalej bude môcť užívateľ zadať typ grafu, ktorý má byť použitý na zobrazovanie informácie. K dispozícii bude mať bullet graf, určujúci typ grafu zobrazujúci počet splnení určitej podmienky. Ďalším typom grafu je stĺpcový graf. Tento môže byť použitý na zobrazenie na porovnanie splnení podmienky v určitých intervaloch. Tretím typom grafu je bodový graf. Predposledným typom je spojitý typ grafu. Posledný zobrazovací typ je tabuľka. Po definovaní grafu sa tento zobrazí na dashboarde. Užívateľ si bude môcť vytvoriť ďalší graf rovnakým postupom a tento opakovať kým nemá pokryté sledovanie všetkých potrebných atribútov.

4 Návrh

Nadchádzajúce kapitoly sa budú venovať návrhu aplikácie z rôznych smerov. Jednotlivé časti postupne aplikáciu priblížia na úrovni prezentačnej vrstvy a na úrovni logickej vrstvy pomocou diagramu balíkov a diagramu nasadenia.

4.1 Návrh dátovej vrstvy

Naša aplikácia vyžaduje pomocné databázové tabuľky pre korektné a rýchle fungovanie. Pomocné databázové tabuľky sú vytvorené priamo v monitorovanej databáze. Tento prístup nám umožňuje spracovať dáta rýchlejšie, pretože využitie vlastnej databázy by znamenalo prenášať dáta medzi databázami cez aplikačnú vrstvu, čo by spôsobilo ďalšie zbytočné spomalenie aplikácie. Databázová štruktúra riešenia je navrhnutá tak, aby nezasahovali objekty do databázy, ktorú dashboard monitoruje. Užívateľ si preto bude môcť definovať unikátny prefix, ktorý bude pridaný ku každému databázovému objektu, ktorý bude vytvorený aplikáciou. Takto budú cudzie objekty v databáze jednoznačne odlíšené a vyriešia sa možné konflikty v názvoch objektov.

Pri vytvorení prepojenia na databázu, v ktorej sa bude prevádzať monitorovanie veličín sa v tejto vytvorí tabuľka združujúca objekty vytvorené nad touto databázou. Táto tabuľka bude obsahovať informácie o názvoch monitorovacích tabuliek, názvu sledovanej tabuľky, na ktorej je vytvorený príslušný trigger, názov triggera a prípadne ďalšie informácie (napríklad čas vytvorenia, čas poslednej zmeny).

Pri vytvorení triggera nad databázou sa vytvorí tabuľka, ktorá bude zhromažďovať údaje získané týmto triggerom. Zároveň sa vytvorí záznam vo väzobnej tabuľke a vytvorí sa trigger nad sledovanou tabuľkou. Tabuľka na zhromažďovanie údajov bude obsahovať unikátny identifikátor, zhromažďované dáta, čas vytvorenia a zmeny záznamu. Medzi zhromažďované dáta patrí počet odfiltrovaných dát, počet zhôd, sledovaný interval (časový a kvantitatívny) a ďalšie voliteľné dáta (suma kvadrátov kde to je aplikovateľné). Každý trigger má vlastnú tabuľku, čo umožňuje jednoduchšie zaobchádzanie s dátami, na rozdiel pri riešení, kde by všetky triggery zapisovali dáta do spoločnej tabuľky. Napríklad pri mazaní triggera sa zmaže databázová tabuľka a nemusia sa jednotlivé záznamy vyhľadávať v spoločnej tabuľke. Taktiež vkladanie nového záznamu triggerom do spoločnej tabuľky by znamenalo použitie prídavných dotazov do tabuľky, aby sa daný záznam vložil na správnu pozíciu. Naším cieľom je vytvoriť aplikáciu, ktorá je schopná vytvoriť záznam v čo najkratšom čase, preto implementujeme riešenie so samostatnými tabuľkami, aj keď to má za následok zvýšenie pomocných objektov v databáze.

4.2 Technológie užívateľského rozhrania

Jednotlivé aplikačné vrstvy budú rozdelené do knižníc. Z predošlého návrhu architektúry vyplýva, že riešenie bude obsahovať aspoň dve knižnice okrem prezentačnej vrstvy. Prvá knižnica má za úlohu komunikáciu s databázou a prevádzanie manipulácie so súborovým systémom, ako je vytváranie, mazanie, zmena konfiguračných súborov. Druhá knižnica predstavuje vrstvu implementujúcu objekty použité na manipuláciu dát v prezentačnej vrstve, ako sú grafy, triggery, či nastavenia. Prídavnou knižnicou bude pomocná knižnica zabezpečujúca manipuláciu s objektmi, ktoré neprislúchajú žiadnej vrstve, ako napríklad pokročilé funkcie na manipuláciu s reťazcami, či metódy využité pri manipulácii s nastaveniami aplikácie. Užívateľské rozhranie nie je prezentované uzavretou knižnicou, tak ako manipulačná a dátová vrstva. Prezentačná vrstva je reprezentovaná vlastnou dynamickou webovou stránkou. Táto pomocou webových technológií bude dáta vyfiltrované v databáze prezentovať užívateľovi pomocou štandardných webových technológií ako je HTML, JavaScript a CSS.

4.2.1 HTML

Skratka HTML je skrátenie názvu Hyper Text Markup Language. Z názvu vyplýva, že sa jedná o značkovací jazyk. Používa sa prevažne na vytváranie webových stránok ale aj na vytváranie ostatného obsahu, ktorý je zobraziteľný v prehliadačoch. Ako príklad možno uviesť dokumentácie k programom. Jazyk HTML možno použiť na zobrazenie rôznych prvkov, ako tabuliek, obrázkov, textov. Pomocou objektov možno zobraziť aj videá, prehrať hudbu a zobraziť interaktívne aplikácie.

Predchodcom jazyka HTML je jazyk SGML (Standard Generalized Markup Language). Podľa [21] bola prvá špecifikácia jazyka HTML zverejnená v roku 1991, ale až v roku 1995 bol HTML štandard prijatý IEEE po viacerých úpravách vo verzii HTML 2.0. V súčasnej dobe sa pripravuje špecifikácia verzie HTML 5.0. Ilustrácia HTML kódu je uvedená na obrázku 17.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<HEAD>
     <TITLE>HTML Overview</TITLE>
     <link rel="stylesheet" type="text/css" href="../ss/2.css" id="thecss">
     <script type="text/javascript" src="../scripts/csschange.js"></script>
</HEAD>

<BODY onload="readSS()">

<br>
<h1 align="center"><span class="pagetitle">HTML Overview</span><br>
```

Obrázok 17: Príklad zdrojového kódu HTML dokumentu [21]

4.2.2 JavaScript

JavaScript vnáša dynamiku do statických webových prezentácií. Jedná sa o objektovo orientovaný jazyk slúžiaci prevažne na dynamické vytváranie obsahu pre návštevníka webových stránok. JavaScript je spúšťaný na klientovi, čo znamená, že je možné preniesť výpočtové úlohy zo servera na klienta, čo pri veľkom množstve klientov s rôznymi požiadavkami na obsah môže znamenať značnú úsporu výpočtového výkonu servera. Problém pri tomto postupe môže nastať, keď sa klient zahltí výpočtami a zobrazenie obsahu môže trvať aj niekoľko minút, čo v prípade dashboardu s aktuálnymi, často sa meniacimi údajmi, nie je prípustné. Príklad JavaScript kódu je uvedený na obrázku 18.

Historicky JavaScript podľa [22] vznikol vydaním prehliadača Netscape Navigator 2 v roku 1996. Tento prehliadač umožnil programátorom prvýkrát od vzniku HTML použiť v stránkach programovací jazyk JavaScript. Otcom jazyka je Brendan Eich, ktorý pracoval na vývoji spomenutého prehliadača.

```
1  // JavaScript Document
2  var el; //element to hide/show
3
4  var index = "not_running";
5
6  function showMenu(el)
7  {
8    document.getElementById(el).style.display="block";
9
10   for(var i = 1; i < 6; i++)
11   {
12       if(el != "menu_"+i)
13       {
14         hideMenu("menu_" + i);
15       }
16   }
17 }
```

Obrázok 18: Príklad zdrojového textu JavaScript dokumentu

4.2.3 CSS

Skratka CSS znamená Cascading Style Sheets. CSS je špeciálny štýlovací jazyk, ktorý sa používa na grafickú úpravu zobrazovaného obsahu HTML dokumentu. Štýlovacie jazyky sa podľa [23] používali už v 70tych rokoch spolu so SGML jazykmi. Neskôr sa CSS začalo používať na oddelenie grafickej reprezentácie obsahu od obsahovej. Definícia štýlov sa väčšinou ukladá do externých súborov, ktoré sú neskôr prepojené referenciou umiestnenou v zdrojovom texte HTML dokumentu.

4.2.4 Výber implementačnej techniky

Implementácia užívateľského rozhrania ako dynamickej webovej stránky iba s použitím vyššie menovaných technológií je možné, ale časovo náročné. Pri implementovaní nášho riešenia je preto žiaduce použiť implementačné techniky a platformy urýchľujúce vývoj. V tejto kapitole si

rozoberieme najčastejšie používané, v krátkosti ich porovnáme a zvolíme si jeden prístup pre našu aplikáciu. Platformy sú diskutované v samostatnej kapitole.

Prvým prístupom, ktorý je možno použiť, je technológia Asynchrónneho JavaScriptu a XML (AJAX). Tento prístup možno kombinovať aj s MVC modelom vývoja aplikácie. Takáto kombinácia by nás ale mohla v neskorších fázach vývoja obmedziť, keďže MVC má presne dané pravidlá implementácie. Zároveň použitie AJAX technológie nevylučuje použitie modelu MVC. AJAX umožňuje obnovu zobrazenej informácie užívateľovi bez nutnosti obnoviť celú stránku. Z tohto pohľadu sa chovaním webové aplikácie približujú aplikáciám, ktoré boli navrhnuté pre operačné systémy. Zároveň môže vhodné použitie tejto technológie odbremeniť databázu aplikácie a webový server.

Model View Controler (MVC) prístup jednoznačne oddeľuje dátový model aplikácie, užívateľské rozhranie a riadiacu logiku systému. Obsah generovaný užívateľovi závisí prevažne na ceste, ktorá je použitá na zobrazenie stránky. Často sa v nej nachádza konkrétny identifikátor ukazujúci na určitý typ obsahu, ako je napríklad identifikátor pre článok na spravodajskom portáli. Používa sa väčšinou pri implementácii jednoduchších internetových stránok. Daný model sa jednoducho implementuje a na internete je k dispozícii dostatočné množstvo príkladov pre túto implementačnú techniku. Daná technika sa uprednostňuje pred AJAX technológiou prevažne v tých prípadoch, ak stránka nesmie stratiť funkčnosť pri vypnutí JavaScriptu na klientskom počítači.

Ďalšou možnosťou je využitie redakčného systému. Redakčný systém umožňuje rýchle vytvorenie webovej stránky bez hlbších znalostí programovacích jazykov. Naša aplikácia by potrebovala pri využití ktoréhokoľvek redakčného systému vytvorenie vlastného modulu na prístup k databáze a na správu našich objektov ako napríklad na správu triggerov a grafov. Z tohto pohľadu nám redakčný systém neprinesie zrýchlenie práce, a zároveň by naša aplikácia zbytočne vyžadovala dodatočné prvky redakčného systému, ktoré by neboli využité.

Z vymenovaných implementačných techník použijeme AJAX technológiu v spojení s vlastným kódom. Aplikácia bude webovou stránkou, čo nám umožní jednoducho poskytnúť prístup k zobrazeným dátam iným užívateľom. Zároveň môže odbremeniť webový server a databázu, čo je v našom projekte výhodou.

4.3 Návrh implementácie logickej vrstvy

Existujú viaceré platformy na vytvorenie dynamickej webovej aplikácie. V skratke porovnáme najrozšírenejšie z nich a vyberieme najvhodnejšiu na implementáciu nášho riešenia. Platforma musí podporovať jednoduchú prácu s databázou MS SQL, pretože naše riešenie musí pracovať s touto databázou. Zároveň je vhodné vybrať platformu s pokročilým vývojovým prostredím na uľahčenie implementácie, keďže riešenie môže nadobudnúť značný rozsah. Rýchlosť platformy je ďalším faktorom, ktorý je v našom prípade dôležitý i keď rýchlosť prevažne závisí na spôsobe

implementácie. Ďalšími podmienkami je vytvorenie objektovo orientovaného riešenia pre zachovanie prehľadnosti implementácie z dôvodu možného budúceho rozšírenia, či spätnej kompatibility v prípade inštalácie na nových verziách operačného systému. Tieto sú hlavné vlastnosti platformy a jazyka, ktoré budeme sledovať.

4.3.1 PHP (Jazyk)

PHP (Hypertext Preprocessor) nie je platforma sama o sebe. Je to programovací jazyk využívaný na vytvorenie dynamických webových stránok, takže v riešení je nutné pri tejto voľbe použiť PHP jazyk. Silné stránky jazyka PHP je v rozšírenej komunite, keďže sa jedná o jednoduchý programovací jazyk používaný na veľkom počte webových stránkach. Oficiálna podpora pre PHP ako jazyk nie je dostupná. Z našich definovaných kritérií podporuje MS SQL databázu a objektovo orientovaný prístup k implementácii. Platforma nie je spätne kompatibilná a vyžaduje úpravy v kóde pri nasadení riešenia na novú verziu, čo môže spôsobovať značné problémy v budúcnosti pri rozsiahlych projektoch. Rýchlosť závisí od spôsobu implementácie. Ako vývojové prostredie je možne použiť ktorýkoľvek editor podporujúci PHP syntax (napríklad NetBeans, Eclipse).

4.3.2 ASP.NET

Platforma ASP.NET podporuje viacero programovacích jazykov, ako napríklad Visual Basic, C#, J#. Platforma má širokú podporu v komunitách na webe a zároveň je podporovaná výrobcom (Microsoft). ASP.NET je objektovo orientovaný a oficiálne vývojové prostredie je Microsoft Visual Studio s Intelisense funkcionalitou, napomáhajúcou k rýchlejšej implementácii. Dané vývojové prostredie disponuje so silným debugovacím nástrojom. Rýchlosť riešenia závisí od implementácie, zároveň platforma obsahuje viacero možností ako riešenie optimalizovať (podporuje napríklad združovanie pripojení k databázam[5]). Platforma podporuje MS SQL databázu, keďže sa jedná o produkty tej istej spoločnosti.

4.3.3 Python (Jazyk)

Python je programovací jazyk, takže pri voľbe tohto riešenia sa musí použiť tento a nemáme alternatívnu voľbu. Podporuje MS SQL databázu po pridaní modulu. Existujú viaceré vývojové prostredia podporujúce vývoj v tomto jazyku (Emacs, Eclipse). Nepodporuje združovanie pripojení k databázam. Rýchlosť riešenia závisí od implementácie. Jazyk má široké zázemie v komunite. Spätná kompatibilita je zaistená nástrojmi na konvertovanie.

[5] <http://msdn.microsoft.com/en-us/library/8xx3tyca%28v=vs.71%29.aspx>

4.3.4 Perl (Jazyk)

Aj v tomto prípade sa priamo jedná o programovací jazyk, takže nie je možné použiť alternatívny jazyk. Podporuje MS SQL databázu. Perl jazyk obsahuje množstvo modulov, ktorými sa rozširuje jeho funkčnosť. Rýchlosť riešenia závisí od implementácie. Podpora jazyka komunitou je dobrá. Vývoj v jazyku Perl podporuje prostredie Komodo IDE.

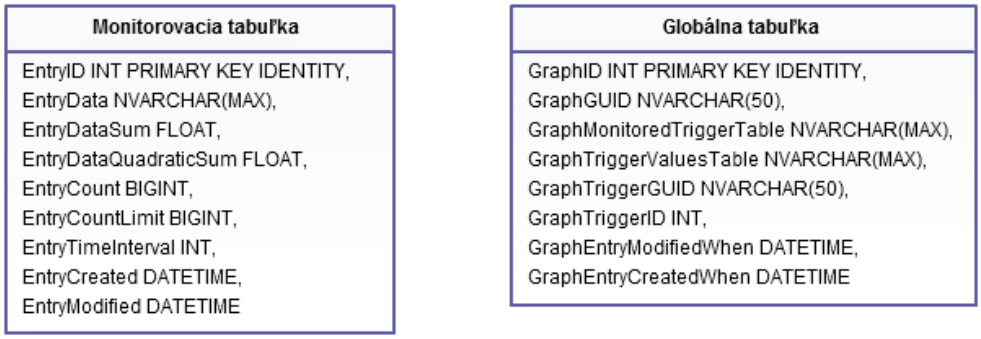
4.3.5 Výber platformy

Pre riešenie daného problému môže byť použitá každá zo spomenutých platforiem. Každá je špecifická v určitom ohľade. Najväčším prínosom v tomto projekte bude čistý a prehľadný kód, rýchla implementácia a riešenie, ktoré bude spracovávať požiadavky v rozumnom čase. Z týchto dôvodov bude riešenie implementované pomocou platformy ASP.NET. Jazyk implementácie bude C#. Táto platforma ponúka jeden z najpokročilejších vývojových nástrojov, široké možnosti optimalizácie a C# kód je dobre čitateľný, keď sa dodržujú určité zásady programovania. Pokročilé vývojové prostredie nám umožní vytvoriť prototyp riešenia v skorých fázach projektu a daný prototyp otestovať, či spĺňa nami zadané kritériá.

4.4 Formálny návrh

4.4.1 Pomocné databázové tabuľky

Posledným zo série štrukturálnych diagramov je schéma pomocných databázových tabuliek, ktoré systém vytvára v sledovanej databáze. Obe typy tabuliek je možné vidieť na obrázku 19.

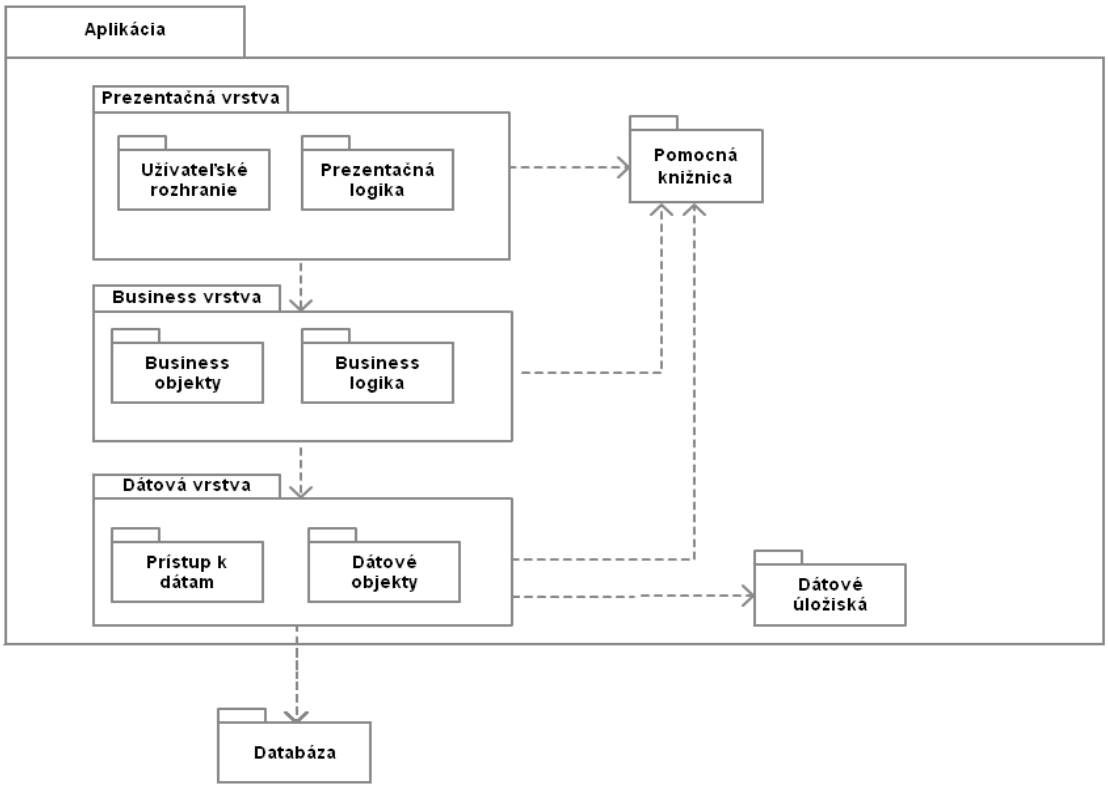


Obrázok 19: Štruktúra pomocných databázových tabuliek

Tabuľka umiestnená na obrázku vľavo je určená na zber konkrétnych informácií, ktoré zachytil trigger. Tabuľka umiestnená na obrázku vpravo predstavuje globálnu tabuľku združujúcu všetky pomocné objekty, ktoré boli vytvorené nad databázou.

4.4.2 Diagram balíkov

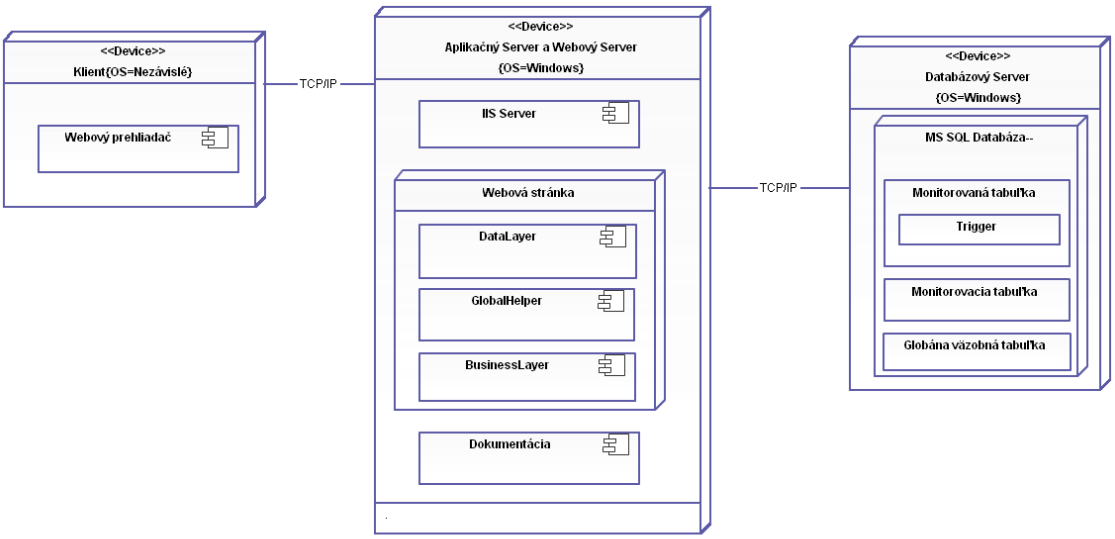
Na obrázku 20 je zobrazený diagram balíkov, ktorý zachytáva celú štruktúru aplikácie. Aplikácia bude členená do troch hlavných balíkov, do prezentačnej vrstvy, business vrstvy a dátovej vrstvy. V týchto sú implementované objekty a metódy zaručujúce potrebnú funkcionalitu.



Obrázok 20: Diagram balíkov

4.4.3 Diagram nasadenia

V diagrame nasadenia, uvedenom na obrázku 21, je znázornená fyzická štruktúra aplikácie. Aplikácia môže byť umiestnená na jednom, alebo na dvoch serveroch. Tretím hardvérovým prvkom, znázornenom na diagrame, je klientske zariadenie pristupujúce cez sieť k webovej stránke, na ktorej je vygenerovaný dashboard. Z výkonnostného hľadiska sa pri databázovej aplikácii odporúča oddeliť databázový server a aplikačný a webový server. Z diagramu možno vyčítať, že aplikácia vytvára pomocné objekty na databázovom serveri, ktoré slúžia na odchytávanie (trigger) a zhromažďovanie (monitorovacia tabuľka) údajov.



Obrázok 20: Diagram nasadenia aplikácie

5 Implementácia

V tejto kapitole sa priblížia implementačné detaily aplikácie, ako napríklad fyzickú štruktúru prezentačnej vrstvy, implementáciu častí užívateľského rozhrania, objekty používané aplikáciou a v krátkosti si popíšeme implementačné zásady.

5.1 Implementačné zásady

Pri implementácii sa dodržiavali základné postupy správneho programovania, menovite dôkladné využívanie komentárov a popisov metód a objektov spolu s ich vlastnosťami. Premenné sa spravidla pomenúvali malým začiatočným písmenom, vlastnosti objektov veľkým začiatočným písmenom a slová sa oddeľovali technikou CamelCase[6]. Počas celého projektu sa používali rovnaké zásady pri riadkovaní. Triedy zapuzdrujú metódy a vlastnosti objektov pre ľahšiu manipuláciu a objekty sú implementované obdobným spôsobom. Napríklad objekty majú rovnaké metódy pre uloženie objektu a jeho odstránenie zo systému. Dodržiavanie týchto jednoduchých zásad celý kód sprehľadňuje a robí ho dobre čitateľným pre vývojárov, keď sú potrebné prípadné úpravy na rozšírenie funkčnosti aplikácie.

5.2 Fyzická štruktúra aplikácie

Aplikácia má štruktúru štandardnej webovej stránky implementovanej pomocou platformy ASP.NET. Na obsluhu prichádzajúcich požiadaviek sa používa Internetová informačná služba (IIS), ktorá je dostupná vo väčšine distribúcii operačného systému Windows. Ako náhradu je možné použiť vstavaný server dostupný vo vývojom prostredí Microsoft Visual Studio.

5.2.1 Súbory v adresári App_Code

V adresári App_Code sa nachádzajú vo všeobecnosti súbory, ktoré ASP.NET kompiluje automaticky do spoločnej odkazovanej knižnice (angl. Dynamic–link–library, DLL) za behu aplikácie. To znamená, že zmeny v týchto súboroch nevyžadujú prekompilovanie aplikácie, ako v prípade zmien v samostatných knižniciach. V našom projekte je v tomto adresári umiestnený súbor obsahujúci testovacie kódy pre aplikáciu, ktoré vytvárajú demonštračný dátový tok a spolu s týmto vytvoria grafy a triggery nad zvolenou databázou. Tieto testovacie kódy obsahujú aj metódy na vyčistenie databázy, nad ktorou bola spustená demonštrácia, po ukončení testu. Pri koncovom nasadení riešenia je možné

[6] <http://en.wikipedia.org/wiki/CamelCase>

tieto odstrániť spolu s modifikovaním administračného rozhrania, ktoré umožňuje tieto testy spustiť a zastaviť.

5.2.2 Súbory v adresári App_Data

Do tohto umiestnenia sa vkladajú pri každej webovej stránke súbory modifikované samotnou aplikáciou. Systém pri zmene akéhokoľvek súboru, okrem súborov v tomto umiestnení a v zložke App_Themes, spustí reštart aplikácie. Preto je nutné každý konfiguračný súbor umiestniť práve do tejto časti systému.

V našom prípade sa v tomto adresári nachádzajú viaceré konfiguračné súbory, ktoré sú umiestnené do zložiek podľa typu obsahu. V koreni tohto adresára nájdeme konfiguračný súbor s nastavením aplikácie (`Settings.txt`) a súbor predstavujúci jednoduchý záznam udalostí v systéme (`EventLog.txt`). Všetky konfiguračné súbory, okrem súboru na záznam udalostí v aplikácii, využívajú formát XML pre jednoduchšiu manipuláciu s jednotlivými hodnotami v systéme.

V koreni tohto adresára sa nachádzajú už spomenuté zložky združujúce ostatné konfiguračné súbory. Adresár `ConnectionString` obsahuje konfiguračný súbor, ktorého obsah predstavuje súbor databázových pripojení, ktoré môžu byť použité v aplikácii. Zložka `Snapshots` obsahuje prípadné uložené snímky grafov a ich dáta v CSV formáte. Tieto môže užívateľ zhotoviť z kontextového menu pre každý graf zobrazený na dashboarde. Zložka `Graphs` obsahuje konfiguračný súbor združujúci definície grafov vytvorených aplikáciou nad databázami, ktoré aplikácia obsluhuje. Ďalej sa v nej nachádzajú testovacie dáta v zložke `TestingData`. Podobnú štruktúru má aj adresár `Triggers`, obsahujúci konfiguračný súbor združujúci definované triggery a testovacie dáta v zložke `TestingData`. Tento adresár obsahuje zároveň zložku `Templates`, ktorá obsahuje predlohy na vytvorenie jednotlivých druhov triggerov. Tento prístup umožňuje užívateľovi vytvárané triggery jednoducho ovplyvniť zmenami práve v týchto súboroch. Skladanie triggerov priamo v aplikácii by neprispelo k prehľadnému kódu a zmeny triggerov by museli byť prevedené v knižniciach, čo by vyžadovalo následnú kompiláciu a reštart aplikácie.

Posledným adresárom, ktorý sa nachádza v tejto zložke, je adresár s názvom `XmlTemplates`. Tento obsahuje predlohy objektov typu trigger, graf a connection, ktoré sa použijú pri vytváraní nového objektu.

Všetky konfiguračné súbory sú obsluhované grafickým užívateľským rozhraním, preto vo všeobecnosti nie je potrebný priamy zásah do týchto súborov.

5.2.3 Súbory v adresári App_Themes

V adresári App_Themes sa nachádzajú súbory použité pri štýlovaní aplikácie. V našom prípade sa v tomto adresári nachádzajú súbory s kaskádovými štýlmi a obrázky používané v aplikácii. Ďalej sa

do tohto adresára generujú všetky grafy, ktoré má užívateľ možnosť vidieť na dashboarde. Dané grafy sú vlastne štýlovým prvkom stránky, preto boli umiestnené do tejto sekcie.

5.2.4 Ostatné súbory

Ďalšie súbory aplikácie sú umiestnené do rôznych adresárov, ktoré nie sú spravidla typické pre webovú stránku vytvorenú nad platformou ASP.NET. Jediným ďalším spoločným adresárom je bin adresár, združujúci kompilované knižnice z jednotlivých projektov. Ostatné zložky obsahujú JavaScript súbory, jednotlivé `UserControl` prvky a súbory vyvárajúce užívateľské rozhranie, ako napríklad sekciu pre nastavenia (Settings) alebo sekciu pre vytváranie grafov (Graphs).

`UserControl` je prvok stránky obsahujúci vlastný kód a vlastnú reprezentáciu na web stránke. Pomocou kódu v pozadí tohto prvku sa generuje dynamicky HTML kód, ktorý sa zobrazí na stránke užívateľovi. Tieto prvky, ktoré možno vytvárať, väčšinou kombinujú viaceré `Control` prvky, čím rozširujú ich funkčnosť. .NET knižnice obsahujú preddefinovanú sadu `Control`[7] prvkov. Ako príklad si možno uviesť `TextBox` prvok a prvok typu `Button`. Skombinovaním dvoch `TextBox` prvkov a jedného `Button` prvku možno vytvoriť `UserControl`, ktorý obsahuje dva vstupné polia na vloženie hesla, užívateľského mena a potvrdzujúce tlačidlo, ktoré užívateľa overí a prihlási. Takto sme vytvorili jednoduchý, znovu použiteľný `UserControl`.

5.3 Postup pri implementácii

V tejto podkapitole si postupne prejdeme jednotlivé časti aplikácie a priblížime si spôsob a postup pri ich implementácii.

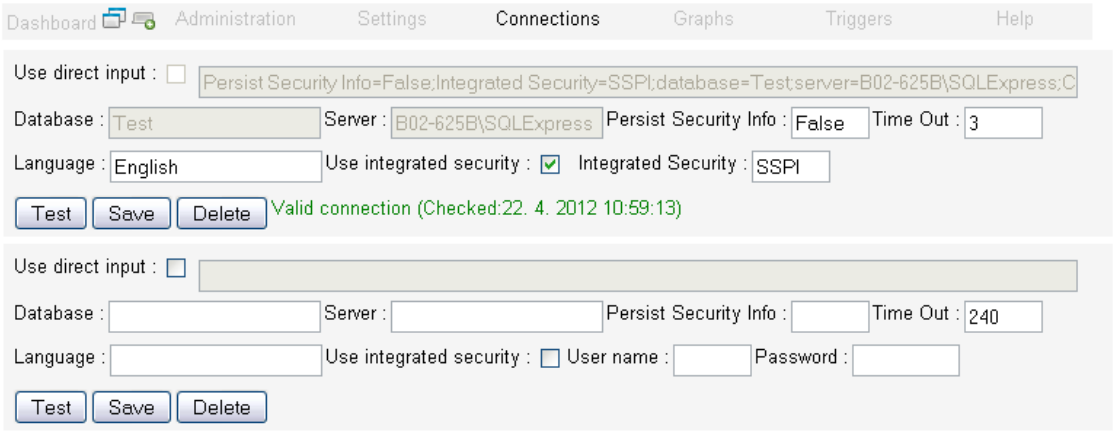
5.3.1 Overenie možností navrhovaného riešenia

Po prvotnom návrhu riešenia sa otestovala použiteľnosť triggerov na danú úlohu, ktoré časti riešenia musia byť implementované softvérovo, a na ktoré je možné použiť definíciu triggera. Triggery museli umožňovať jednoduchú správu, možnosť spustenia záznamu na základe určitej podmienky definovanej užívateľom, vzorkovanie, jednoduché vytváranie záznamov na základe počtu spracovaných dát a na základe časového rámca. Všetky tieto body triggery spĺňali, a preto sa vytvorila základná kostra triggera, ktorá určovala, aké časti musia byť definovateľné užívateľom a tým aj určovala aké prvky musí užívateľské rozhranie obsahovať.

[7] <http://msdn.microsoft.com/en-us/library/system.windows.forms.control.aspx>

5.3.2 Implementácia správy databázových spojení

Prvým krokom z pohľadu implementácie bolo vytvoriť užívateľské rozhranie a logiku na tvorbu databázového spojenia, keďže je toto nevyhnutné na akúkoľvek manipuláciu s databázovými objektmi. Užívateľské rozhranie umožňuje vložiť databázové spojenie priamo, alebo definovať jednotlivé jeho súčasti. Na obrázku 22 vidno užívateľské rozhranie na tvorbu databázového pripojenia s vloženými testovacími dátami.

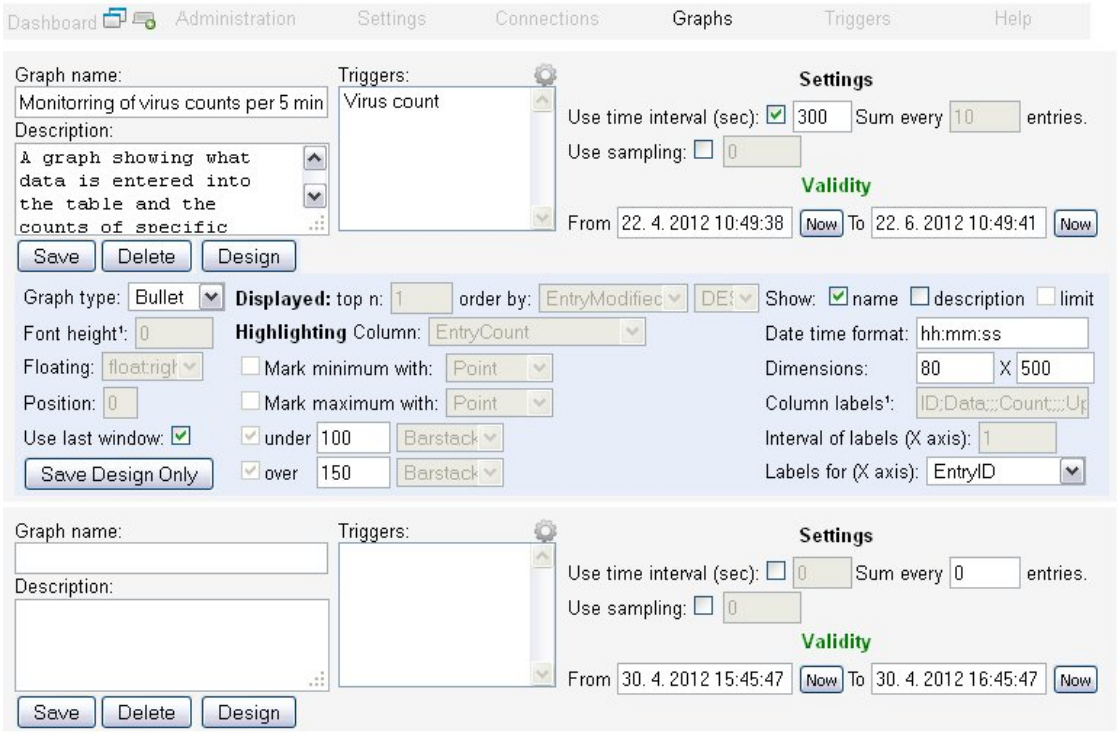


Obrázok 22: Užívateľské rozhranie na tvorbu databázových pripojení

Objekty reprezentujúce databázové spojenia sa ukladajú kliknutím na tlačidlo `Save` do súboru `ConnectionStrings.txt`. Pri vytvorení platného databázového pripojenia sa v databáze vytvorí globálna tabuľka, ktorá bude obsahovať mená objektov vytvorených nad danou databázou. Spodná časť užívateľského rozhrania slúži na vloženie nového spojenia. V aplikácii je definovaný vlastný objekt (`Connection`), ktorý slúži na manipuláciu s daným spojením. Tento objekt je definovaný v dátovej vrstve (knižnica `DataLayer`). `Connection` objekt obsahuje všetky potrebné dáta na zahájenie databázového spojenia. Ako každý objekt má unikátny identifikátor, ktorý sa používa na prepojenie jednotlivých spojení s objektom typu `Trigger`. Taktiež obsahuje príznak určujúci, či je spojenie platné a je možné ho použiť na získavanie informácií z databázy.

5.3.3 Implementácia správy grafov

Ďalším krokom v implementácii bolo vytvorenie logiky na správu grafov a užívateľské rozhranie k tejto časti systému. Inštancie objektov, reprezentujúce v systéme grafy, sa ukladajú v XML formáte do súboru `GraphListing.txt`. Vytvorenie grafu nevytvára žiadne objekty v databáze, aby sa predišlo zbytočnému zahlteniu systému. Na obrázku 23 možno vidieť užívateľské rozhranie na správu a tvorbu grafov.



Obrázok 23: Užívateľské rozhranie na tvorbu grafov

Pre dynamickú manipuláciu s grafmi je na správu grafu využitý vlastný `UserControl`. Tento sa dynamicky vytvára pri načítaní stránky. Na stránke sa vytvorí toľko objektov typu `UserControl`, koľko ich je uložených v systéme a jeden navyše, slúžiaci na vytvorenie nového grafu. Tento princíp sa používa pri databázových spojeniach, grafoch a triggeroch. V hornej časti obrázku sa nachádza príklad grafu typu `Bullet` s testovacími dátami. Každý graf má dve konfiguračné sekcie. Prvá ovplyvňuje naviazané triggery a preto sa pri zmenách v tejto časti musia naviazané triggery pregenerovať a znovu uložiť. Táto časť sa nachádza v sekcii `Settings`. Ostatné nastavenia ovplyvňujú iba definíciu grafu, ktorá je uložená v konfiguračnom súbore, preto je možné tieto časti uložiť nezávisle na definícii triggerov pomocou tlačidla `Save Design Only`. Väčšina týchto nastavení je združená v sekcii `Design` (modrá sekcia) a je možné ju skryť alebo zobraziť kliknutím na tlačidlo `Design`. Graf má danú platnosť. Na pozadí beží proces, ktorý kontroluje v určitých intervaloch platnosť grafov a pri zmene aktualizuje definície triggerov, ktoré sú priradené ku grafu. Každý graf podporuje vzorkovanie a umožňuje užívateľovi určiť pravdepodobnosť s akou sa položka spracuje. Vzorkovanie je podrobnejšie popísané v jednej z nasledujúcich kapitol.

V dolnej časti konfigurácie grafu sa nachádza formulár na vytvorenie nového grafu. Jednotlivé položky sa zobrazujú a odomykajú podľa toho, aký typ grafu je zvolený.

V aplikácii je graf reprezentovaný objektom `Graph`, ktorý je definovaný vo vrstve `BusinessLayer`. Objekt grafu má definované metódy na manipuláciu s týmto objektom. Medzi tieto

patrí metóda `Save`, ktorá objekt uloží do konfiguračného súboru a aktualizuje triggery, ktoré sú asociované s daným grafom. Ďalšia metóda slúžiaca na správu objektu je metóda `Destroy`, ktorá objekt odstráni z konfiguračného súboru, ak graf neobsahuje žiadne triggery. V opačnom prípade musia najprv byť odstránené tieto triggery. Objekt, predstavujúci graf, má dva konštruktory. Jeden slúži na vytvorenie prázdneho objektu a druhý prijíma identifikátor už existujúceho grafu ako parameter. Týmto konštruktorom sa inicializuje graf, ktorý už je v systéme uložený.

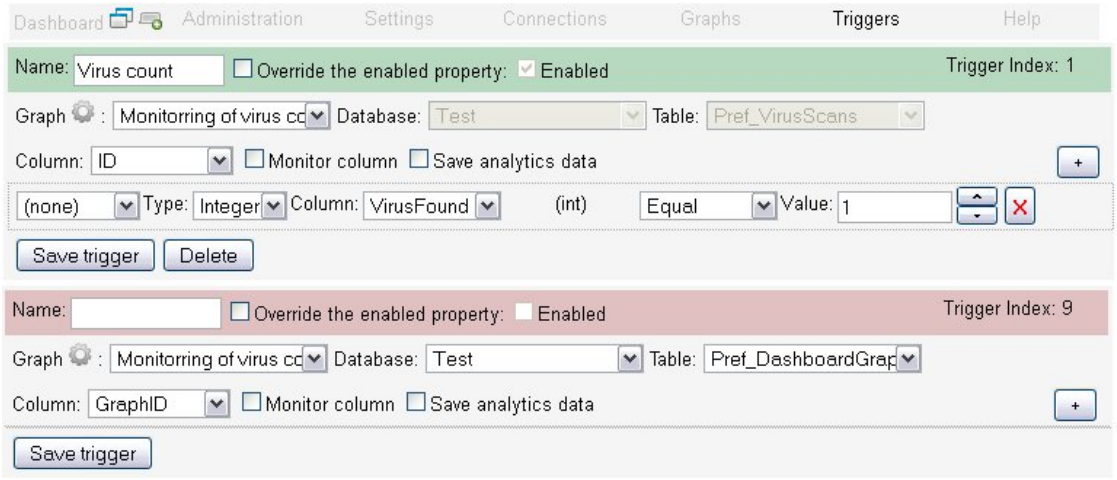
5.3.4 Implementácia a správa triggerov

Po implementovaní objektu reprezentujúci graf nasledovala implementácia objektu reprezentujúci trigger a užívateľského rozhrania na správu týchto objektov. Triggery sú v aplikácii reprezentované objektom `Trigger`, ktorý je rovnako ako objekt `Graph` implementovaný vo vrstve `BusinessLayer`. `Trigger` má podobné rozhranie ako `Graph`. Na manipuláciu s týmto objektom slúžia metódy `Save`, `Destroy` spolu s dvoma konštruktormi, kde jeden slúži na vytvorenie prázdneho objektu a druhý na inicializáciu objektu, ktorý sa už v systéme nachádza. Každý typ triggeru je reprezentovaný nejakou šablónou uloženou v adresári `Templates`. Spomenuté predlohy k triggerom obsahujú špeciálne výrazy, ktoré sa v aplikácii nahrádzajú konkrétnymi hodnotami. Výrazy je možno opakovane použiť v súboroch a aplikácia ich zakaždým nahradí za správnu hodnotu. Ako príklad možno uviesť špeciálny výraz ##TABLENAME##, ktorý sa nahradí názvom tabuľky, nad ktorou sa vytvára daný trigger. Príklad takého triggera možno vidieť nižšie:

```
--trigger for getting the entries if a limit was set
##CREATEORUPDATE## TRIGGER ##TRIGGERNAME## ON ##TABLENAME## AFTER INSERT, UPDATE AS
-- the trigger is enabled or disabled
IF ##ENABLED##
BEGIN
       -- sampling condition
        IF ##SAMPLING##
               BEGIN
               --declare variables to execute minum queries
               DECLARE @matchedCondition INT;
               SET @matchedCondition = 0;
               DECLARE @entryIdToUpdate  BIGINT;

               SELECT @entryIdToUpdate = max( EntryID ) FROM ##TRIGGERVALUETABLE##;

                -- check, if the input matched our condition
               IF ##CONDITION##

                      BEGIN
                            SET @matchedCondition = 1;
                      END

                      --check if we don't need to create a new entry
                      IF ((SELECT EntryCountLimit FROM ##TRIGGERVALUETABLE## WHERE EntryID =
                     ( @entryIdToUpdate)) >= ##LIMIT##)

                     BEGIN
                            -- insert also
                            INSERT ##TRIGGERVALUETABLE## (EntryData, EntryCount,
                            EntryCountLimit, EntryCreated, EntryModified) VALUES
                            ('',@matchedCondition,1, GETDATE(), GETDATE())
                     END
               ELSE
                      BEGIN
```

```
                        -- update only
                        UPDATE ##TRIGGERVALUETABLE## SET EntryCountLimit  =
                        EntryCountLimit+1, EntryCount = EntryCount + @matchedCondition,
                        EntryModified = GETDATE()  WHERE EntryID = @entryIdToUpdate
            END
    END
END
```

Pri vytvorení objektu `Trigger`, pomocou jednej zo šablón, je vytvorená aj dodatočná databázová štruktúra a to tabuľka, do ktorej sa vkladajú zbierané záznamy a samotný trigger, ktorý sa vytvorí nad sledovanou tabuľkou. Pri vytvorení triggera sa taktiež vytvorí záznam vo väzobnej tabuľke, ktorá bola vytvorená pri definovaní pripojenia na túto databázu. Systém dokáže vytvoriť a spravovať desať rôznych typov triggerov. Použitie určitého typu závisí na vlastnostiach triggera a grafu. Tieto vlastnosti triggera možno spravovať obdobným spôsobom ako vlastnosti grafu cez dynamicky generovanú stránku. Príklad takejto stránky s testovacími dátami je uvedený na obrázku 24.



Obrázok 24: Užívateľské rozhranie na správu triggerov

Užívateľské rozhranie je postavené na rovnakom princípe ako užívateľské rozhrania grafov a databázových spojení, kde sa použili prvky `UserControl`. V tomto prípade obsahuje hlavný `UserControl` prvok ďalšie vnorené dynamicky pridávané `UserControl` prvky reprezentujúce jednotlivé časti podmienky, ktoré môže užívateľ ľubovoľne rozšíriť. V dolnej časti obrázku sa znovu nachádza formulár na vytvorenie nového objektu, tentoraz typu trigger.

5.3.1 Implementácia nastavení a administračného rozhrania

Po dokončení implementácie triggerov nasledovalo vytvorenie sekcie s nastaveniami a administračnej sekcie. Sekcia nastavení je v aplikácii nazvaná `Settings` a rovnaký názov má aj objekt

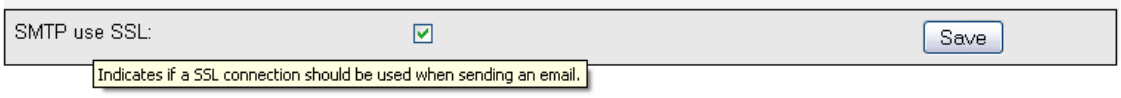
reprezentujúci jedno nastavenie v aplikácii. Každé nastavenie má svoj ekvivalent uložený v XML súbore `Settings.txt`. Aplikácia rozlišuje štyri dátové typy nastavení:

- `String`
- `Integer`
- `Boolean`
- `Password`

Ak pridáme nové nastavenie do spomenutého konfiguračného súboru, tak aplikácia podľa zvoleného dátového typu nastavenia použije správny `Control` na manipuláciu a zobrazenie v sekcii `Settings`. Ako príklad nastavenia možno uviesť nastavenie `EmailUseSSL`:

```
<?xml version="1.0"?>
<file>
    <EmailUseSSL Type="Boolean" Tooltip="Indicates if a SSL connection should be used when sending
    an email." DisplayName="SMTP use SSL:">True</EmailUseSSL>
</file>
```

Túto XML reprezentáciu systém načíta a interpretuje ako zaškrtávací `Control` ako možno vidieť na obrázku 25.



Obrázok 25: Ukážka vygenerovaného nastavenia

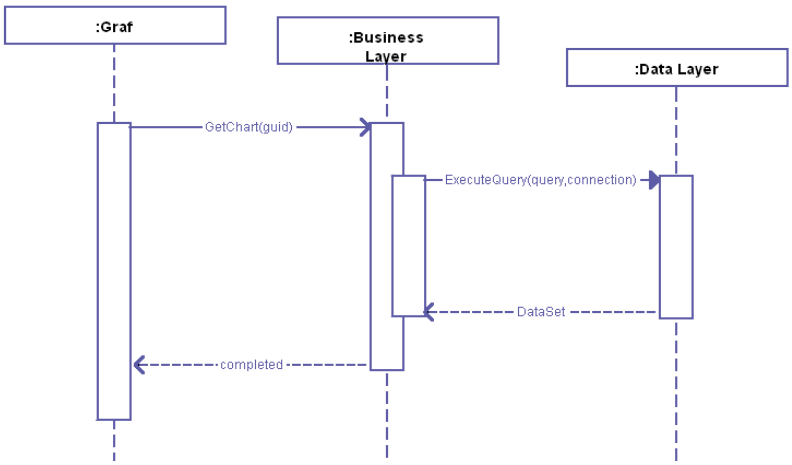
Dátové typy `String`, `Password` a `Integer` sú reprezentované podobným spôsobom. Všetky využívajú textové pole. Tieto textové polia sa líšia iba v nastavených vlastnostiach a validácii. `Integer` sa validuje ako číslo, `String` a `Password` ako text. `Password` na rozdiel od dátového typu `String` nie je zobrazené užívateľovi. V tomto prípade je textové pole využité iba na vloženie nového hesla.

Administrátorské rozhranie obsahuje tlačidlo na testovanie SMTP údajov, ktoré po stlačení pošle testovací e-mail na e-mailovú adresu zadanú v nastaveniach, tlačidlo na vyčistenie systému od vygenerovaných obrázkov grafov a dve tlačidlá slúžiace na testovanie systému. Jedno vytvorí testovacie tabuľky, grafy a triggery a spustí simuláciu a druhé simuláciu zastaví s vymaže testovacie dáta.

5.3.2 Implementácia dashboardu

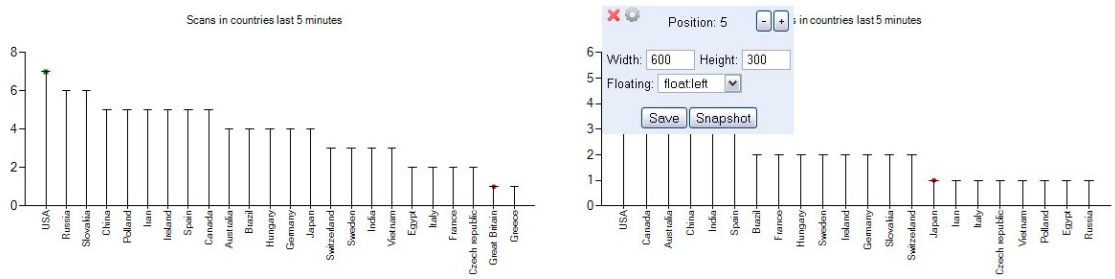
Po vytvorení všetkých potrebných objektov na zaistenie zbierania údajov a užívateľského rozhrania na ich správu, sa pristúpilo k implementácii samotného dashboardu. Každý graf je reprezentovaný jedným objektom `UserControl`. Tieto prvky iba zobrazujú daný graf, ktorý je vygenerovaným obrázkom. Každý graf je uložený jednoznačne s menom jeho unikátneho identifikátora v zložke App_Themes v adresári Graphs. Preto môže byť na správu každého typu grafu použitý ten istý `UserControl`. Tento implementuje obnovovanie grafu v daných intervaloch, ktoré je možné špecifikovať v nastaveniach. Graf sa obnovuje pomocou JavaScript kódu. Tento postup bol zvolený pretože JavaScript umožňuje obrázok načítať rýchlejšie, bez rušivého bliknutia, na rozdiel od implementácie obnovy z kódu pomocou obnovenia časti stránky, ktorá je podporovaná .NET architektúrou.

Na obrázku 26 je zobrazený sekvenčný diagram znázorňujúci proces obnovy grafu. Ako vidno, na obnovu grafu je potrebné prechádzať všetkými vrstvami aplikácie, keďže graf ako zobrazovací prvok v prezentačnej vrstve zobrazuje už predspracované dáta z business vrstvy. Táto získava dáta z databáze cez dátovú vrstvu pomocou metód spravujúcich dotazy do databáze.



Obrázok 26: Sekvenčný diagram obnovenia grafu

Užívateľský prvok grafu ďalej implementuje kontextové menu pomocou ktorého je možné upraviť umiestnenie a dimenzie daného grafu. Taktiež je možné z tohto grafu vytvoriť snímok grafu spolu s uložením dát do CSV súboru. Na obrázku 27 je ukážka grafu s a bez kontextového menu, ktoré je možné vyvolať kliknutím na plochu grafu.



Obrázok 27: Ukážka grafov na dashboarde s a bez kontextového menu

Každý graf sa generuje tým istým procesom na pozadí aplikácie v intervaloch, ktoré sú určené užívateľom v sekcii s nastaveniami aplikácie. Užívateľ si môže vybrať z troch základných reprezentácií dát. Každá z týchto reprezentácií sa implementuje odlišne.

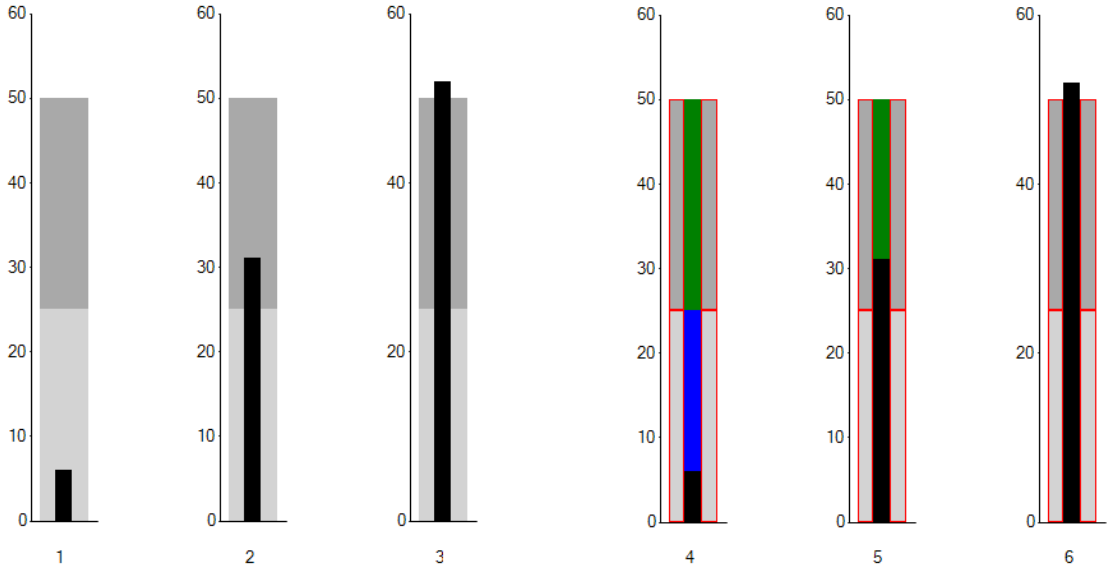
Prvá, tabuľka (`Table`) je implementovaná bez pomoci modulu MSCharts[8], pretože tento zobrazenie dáta v tabuľkovej forme nepodporuje. Preto sa v kóde musia dopočítavať podľa rozmeru grafu veľkosť písma, šírka jednotlivých stĺpcov, prípadne počet tabulátorov využitých na zarovnanie každého riadku. Taktiež sa dynamicky vypočítava poloha a veľkosť značiek označujúcich napríklad maximum v danej tabuľke. Tieto značky sú podporované hlavne preto, aby uľahčili identifikovanie dôležitých údajov osobám, ktoré trpia napríklad farbosleposťou. Príklad tabuľky možno vidieť na obrázku 28.

ID	Data	Count	Created	Updated
24	Greece	2	19:39:52	19:40:23
23	Sweden	2	19:39:23	19:42:06
22	Iran	1	19:39:09	19:39:09
21	Switzerland	1	19:38:43	19:38:43
20	Vietnam	2	19:38:13	19:42:02
19	Slovakia	2	19:36:58	19:42:10
18	USA	3	19:36:48	19:40:32
17	Brazil	4	19:36:31	19:41:27
16	Russia	5	19:36:25	19:39:46
15	Czech republic	1	19:36:08	19:36:08
14	Great Britain	2	19:35:46	19:37:20
13	Japan	2	19:35:37	19:39:56
12	Spain	5	19:35:31	19:41:45
11	Australia	4	19:35:25	19:39:32
10	Egypt	3	19:35:18	19:41:02
9	Italy	4	19:35:08	19:41:50
8	Germany	3	19:34:59	19:39:01
7	India	2	19:34:56	19:37:04
6	France	5	19:34:52	19:40:44
5	Polland	6	19:34:46	19:41:08
4	Ireland	6	19:34:43	19:41:23
3	Canada	6	19:34:37	19:42:24
2	China	3	19:34:27	19:41:40

Obrázok 28: Príklad tabuľky s vyznačeným maximom, minimom a hodnotami, ktoré spadajú pod určitú hranicu a nad určitú hranicu

[8] <http://www.microsoft.com/en-us/download/details.aspx?id=14422>

Druhý typ grafu s názvom `Bullet` je implementovaný s čiastočnou podporou modulu MSCharts. Takýto typ grafu nie je dostupný v danom module[9], preto je graf poskladaný z rôznych grafov, ktoré tento modul podporuje. Na obrázku 29 je znázornený spôsob stavby tohto typu grafu v troch hlavných fázach, ktorými postupne prechádza.



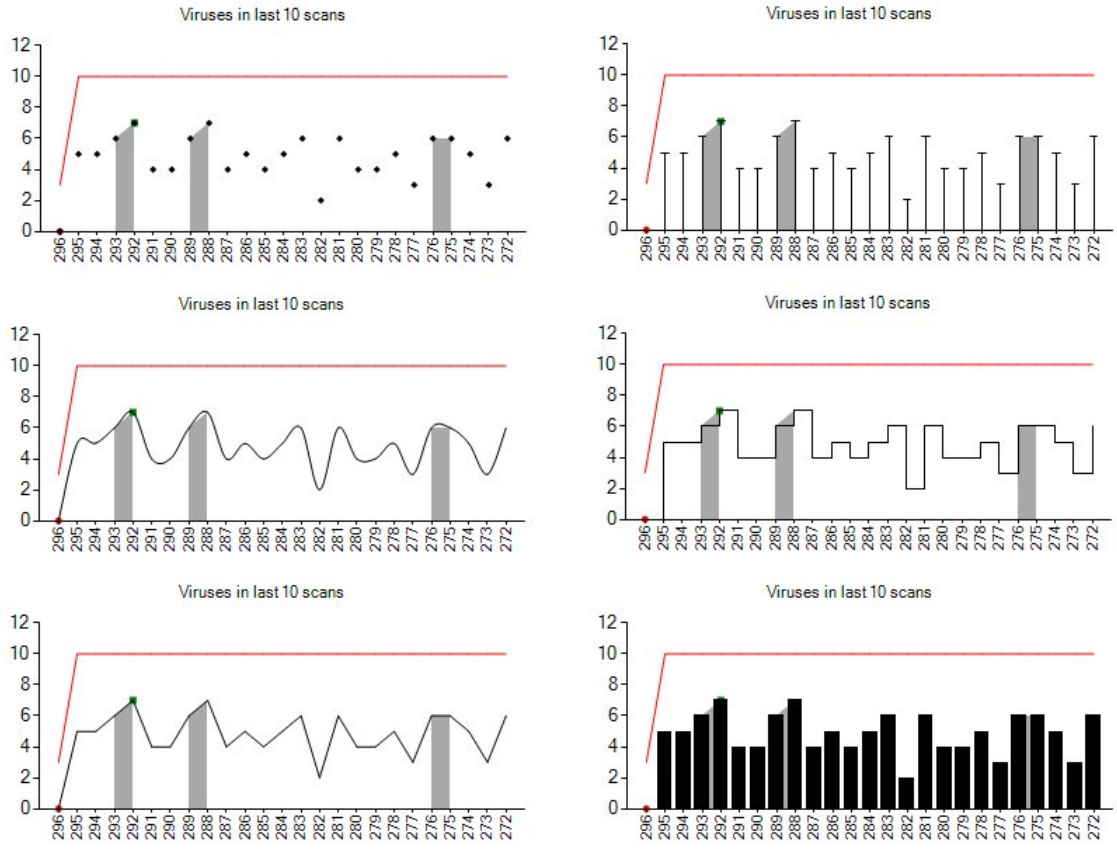
Obrázok 29: Tri fázy grafu typu bullet a ich vnútorná stavba

Graf typu bullet prechádza fázami, keď je hodnota sledovanej veličiny pod dolnou hranicou, medzi hornou a dolnom hranicou a nad hornou hranicou. Tieto fázy sú reprezentované na obrázku 29 grafmi 1, 2 a 3. V grafoch označenými číslami 4, 5 a 6 sú zvýraznené jednotlivé časti grafu, ako sú reprezentované v systéme. Jeden graf je reprezentovaný tromi stĺpcami, ktoré sú vedľa seba usporiadané bez medzier. Tento prístup vytvára ilúziu jednoliateho celku. Každá z označených častí je reprezentovaná grafom typu `BarStacked`, ktorý zobrazuje jednotlivé hodnoty nad sebou, takže najvyšší bod leží v mieste súčtu jednotlivých jeho častí. Preto je nutné zadané hodnoty od užívateľa použiť na výpočet jednotlivých častí jedného stĺpca. V grafe označenom číslom 4 sa postupne zmenšuje hodnota označená modrou a zväčšuje hodnota sledovanej veličiny, ktorá je znázornená čiernou farbou. V grafe s číslom 5 sa zmenšuje zelená časť grafu a zväčšuje sledovaná veličina, až kým tá v poslednom grafe nepresiahne hornú hranicu a v strednej časti grafu zostane zobrazená iba sledovaná veličina. Hodnoty ľavého a pravého stĺpca ostávajú nemenné. Tento typ grafu podporuje zaslanie notifikačnej správy na zadanú e-mailovú adresu, pri prekročení hornej a dolnej hranici. E-mailovú adresu možno vyplniť v sekcii `Settings`.

Posledným typom grafu je graf reprezentujúci údaje v čase alebo počty záznamov splňujúce podmienky definované pomocou triggera v spojením s nastaveniami grafu. Tento môže mať rôzne

[9] <http://msdn.microsoft.com/en-us/library/dd489233.aspx>

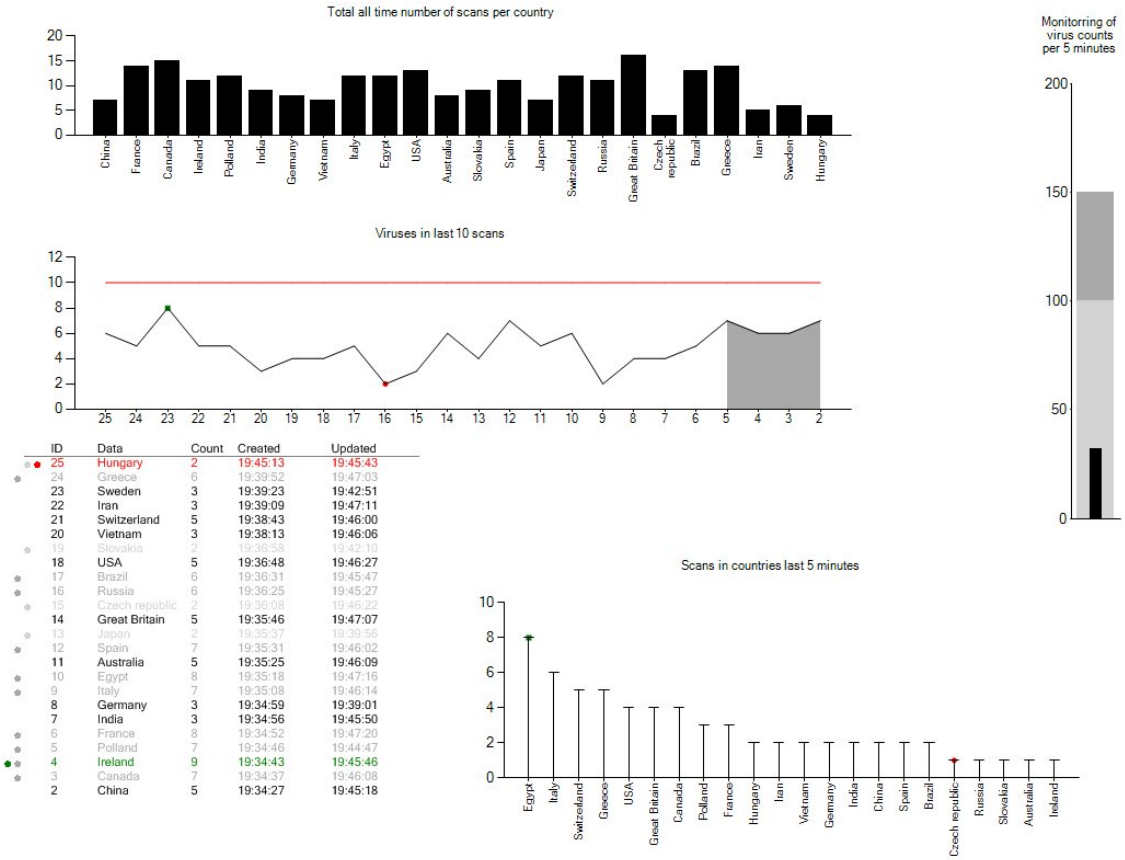
reprezentácie, napríklad jednotlivé hodnoty môžu byť reprezentované bodmi (`Point`), stĺpcami (`Bar`), líniou (`Line`), líniou typu spline (`Spline`), skokovou líniou (`StepLine`) alebo tenkými stĺpcami s hlavičkou (`BoxPlot`). Tieto grafy sú znázornené na obrázku 30. Užívateľ si môže typ grafu zvoliť nielen pre reprezentovanie údajov, ale aj na vyznačenie maxima, minima, hodnôt presahujúce určitú spodnú a vrchnú hranicu. Tento typ zobrazenia dát je plne podporovaný modulom MSCharts a vyžaduje iba správne nastavenie objektu `Chart`, ktorý predstavuje jeden graf. Všetky nastavenia, ktoré sú možné ovplyvniť cez užívateľské rozhranie, musia byť uložené v konfiguračnom súbore `GraphListing.txt`.



Obrázok 30: Rôzne druhy spojitých a diskrétnych grafov zobrazujúce rovnaké dáta

Pomocou spomenutých typov grafov je možné pokryť väčšinu sledovaných údajov, ktoré možno zaznamenávať v databázových systémoch. Všetky nastavenia grafu, ktoré sú umiestnené pod sekciou `Design` v užívateľskom rozhraní pre správu grafov, je možné meniť bez zásahu do databázy. Medzi tieto patria aj všetky typy grafov, ktoré môže užívateľ ovplyvniť. Výsledný dashboard môže vyzerať ako na obrázku 31. Tento príklad dashboardu je skoro totožný z hľadiska dizajnu s naším ideálnym dashboardom, ktorý sme vytvorili v kapitole 2.7.1. Dashboard, ktorý bol použitý ako

predloha nasleduje správne princípy reprezentovania informácií v dashboarde, ktoré sú popísané v zdroji [1] a ktoré sme podrobnejšie rozobrali v prvej polovici tejto práci.

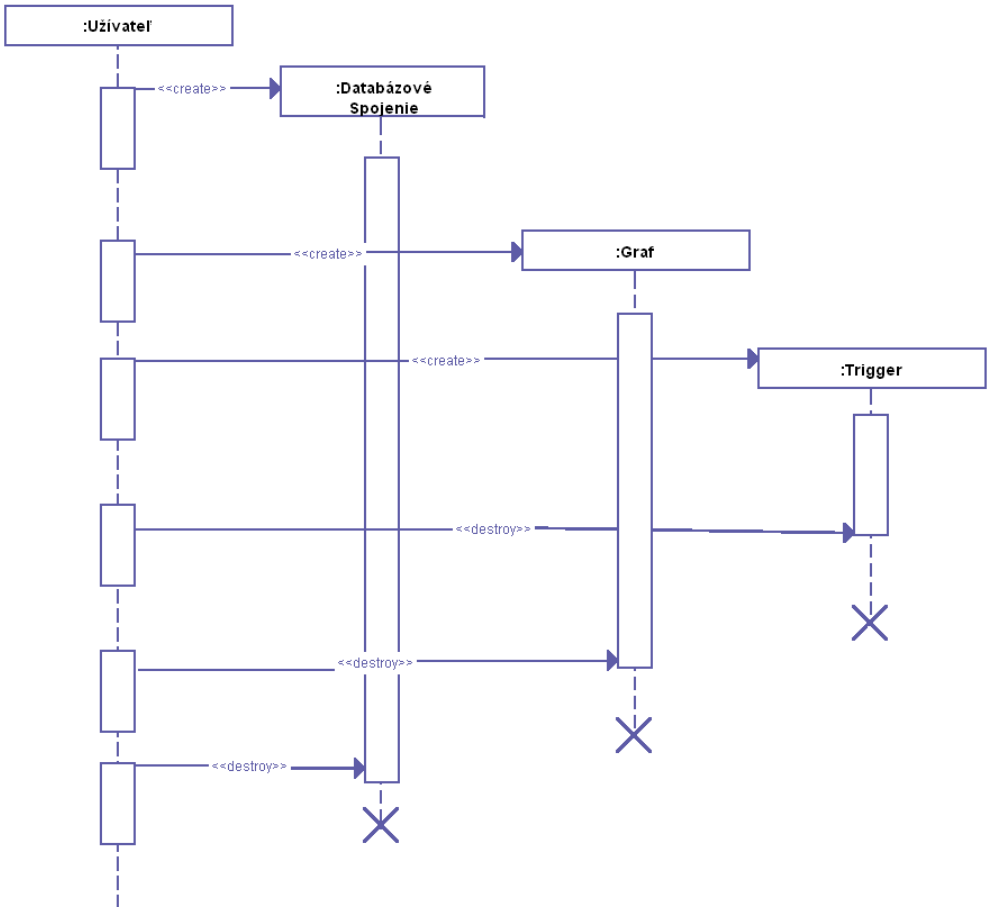


ID	Data	Count	Created	Updated
25	Hungary	2	19:45:13	19:45:43
24	Greece	6	19:39:52	19:47:03
23	Sweden	3	19:39:23	19:42:51
22	Iran	3	19:39:09	19:47:11
21	Switzerland	5	19:38:43	19:46:00
20	Vietnam	3	19:38:13	19:46:06
19	Slovakia	2	19:36:58	19:47:10
18	USA	5	19:36:48	19:46:27
17	Brazil	6	19:36:31	19:45:47
16	Russia	6	19:36:25	19:45:27
15	Czech republic	7	19:36:08	19:46:22
14	Great Britain	5	19:35:46	19:47:07
13	Japan	2	19:35:37	19:36:55
12	Spain	7	19:35:31	19:46:02
11	Australia	5	19:35:25	19:46:09
10	Egypt	8	19:35:18	19:47:16
9	Italy	7	19:35:08	19:46:14
8	Germany	3	19:34:59	19:39:01
7	India	3	19:34:56	19:45:50
6	France	8	19:34:52	19:47:20
5	Polland	7	19:34:46	19:44:47
4	Ireland	9	19:34:43	19:45:46
3	Canada	7	19:34:37	19:46:08
2	China	5	19:34:27	19:45:18

Obrázok 31: Jeden z možných spôsobov reprezentácie dát touto aplikáciou

5.3.1 Postup pridávania a odoberania objektov

Na obrázku 32 je znázornený proces, v ktorom užívateľ postupne vytvára jednotlivé objekty potrebné na zobrazenie dát v dashboarde. Prvým krokom je vytvorenie databázového spojenia. V druhom kroku užívateľ vytvára graf. Posledným krokom je vytvorenie triggera. V opačnom poradí sa tieto objekty zo systému odstraňujú.



Obrázok 32: Sekvenčný diagram pridania a odoberania objektov v systéme

5.3.2 Zbieranie štatistických údajov

Niektoré typy triggerov podporujú zbieranie základných štatistických údajov. Ak daný trigger nie je označený ako monitorovací a je vybraný stĺpec, na ktorý možno použiť zbieranie štatistických dát, tak je možné zaškrtnúť voľbu `Save analytics data`. Medzi stĺpce podporujúce zbieranie štatistických dát patria tie, ktoré sú typu Integer a Float. Pri uložení triggera s týmito nastaveniami sa vygeneruje trigger, ktorý zbiera sumu a kvadratickú sumu daného stĺpca. Z týchto údajov možno vypočítať napríklad smerodajnú odchýlku alebo priemer, keďže poznáme aj počet prvkov, ktorý je zbieraný s každým triggerom.

5.3.3 Vzorkovanie

Aplikácia podporuje vzorkovanie. Zo spomenutých vzorkovacích techník sa v tomto prípade použila technika jednoduchého náhodného vzorkovania. Každá z týchto techník má kladné a záporné stránky. Metóda jednoduchého náhodného vzorkovania je jednoducho implementovateľná, má minimálnu pamäťovú náročnosť a jej implementácia je dostatočne rýchla na to, aby bola použitá v tomto riešení.

Hlavným dôvodom uprednostnenia tejto techniky bol fakt, že na prijatie rozhodnutia, či danú vzorku spracovať alebo vynechať nie sú potrebné dáta z predchádzajúcich dávok, ktoré sa vložili do databázy. Ostatné vzorkovacie techniky práve takéto dáta potrebovali alebo by boli výpočtovo náročnejšie. Zároveň použitie prístupu „vynechaj každý n-tý prvok“ nie je dobrou praktikou, keďže je viac pravdepodobné, že taký prístup skryje periódu, ak sa táto v sledovaných dátach nachádza, pritom perióda je jedným z dôležitých znakov sledovaných dát.

Prijatím tejto vzorkovacej techniky sa zároveň užívateľovi umožní ovplyvnenie množstva spracovaných dát tak, že si nastaví hranicu, ktorá určí či sa vzorka spracuje alebo zahodí.

5.3.4 Optimalizácie

Aby sa predišlo zbytočným čítaniam z disku bola implementovaná cache pre konfiguračné súbory. Táto cache je implementovaná objektom `Dictionary`[10] v triede `MySession`. Ako kľúč pre jednotlivé položky je použitá cesta k súboru. Pri uložení nového záznamu alebo pri aktualizácii sa položka v cache obnoví, aby obsahovala najnovšie dáta. Týmto prístupom sa obmedzili operácie na disku z pohľadu konfiguračných súborov na zápis nových údajov a na načítanie v prípade, keď sa dané dáta potrebujú zapísať do cache po prvý krát. Prvotné načítanie sa prevedie pri prvom požiadavku pre dané dáta.

Podobným spôsobom ako cache pre konfiguračné súbory, je implementovaná cache pre časté databázové dotazy. Týmto spôsobom sa uchováva zoznam všetkých databázových tabuliek pre databázu a zoznam stĺpcov pre databázové tabuľky. V prvom prípade sa ako kľúč využíva názov databázy a v druhom kombinácia názvu databázy a názvu databázovej tabuľky. Tak ako v predchádzajúcom príklade sa uchovávajú iba tie záznamy, ktoré si aplikácia vyžiada. Cache pre tabuľky a stĺpce sa vyprázdňujú pri spravovaní triggerov, keďže pri ich správe sú potrebné vždy aktuálne dáta.

Užívateľ si môže nastaviť horný limit počtu údajov v monitorovacích tabuľkách. Ideálna horná hranica maximálneho počtu údajov je daná počtom potrebných údajov, ktoré sú zobrazené v grafe vynásobených konštantou „1,112“ (presne 100/90), keďže čistiaci mechanizmus odstraňuje 10% záznamov, keď sa daná hranica presiahne. Tento mechanizmus je implementovaný týmto spôsobom preto, aby sa mazanie nespúšťalo pri každom kontrolovaní tabuľky. Ideálna hodnota pre interval spúšťania kontrolného mechanizmu je ovplyvnený množstvom dát, ktoré sú vkladané do tabuľky. Pri modelovom prípade vloženia 1200 záznamov do tabuľky za 60 sekúnd a zobrazovania 500 záznamov v grafe by bola ideálna horná hranica maximálneho počtu záznamov v tabuľke $500 \times 1{,}112 = 556$. Interval kontroly tabuľky je možné vypočítať tak, že sa zistí čas, za ktorý sa dosiahne horná hranica maximálneho počtu záznamov. V našom prípade by to bolo $556/(1200/60) = 27{,}8$ sekúnd. Túto hodnotu je možné zaokrúhliť napríklad na 30 sekúnd.

[10] <http://msdn.microsoft.com/en-us/library/xfhwa508.aspx>

6 Testovanie a zhodnotenie výsledkov

Po dokončení implementácie bola aplikácia testovaná na náhodne generovaných dátach. Toto testovanie a typy jednotlivých testov je detailne opísané v nasledujúcich kapitolách.

6.1 Spôsob testovania

Pri testovaní aplikácie sme sa zamerali na testovanie vplyvu triggerov na vkladanie záznamov do databázy. Tento postup sa zvolil z viacerých dôvodov. Komplexné testovanie aplikácie, ako napríklad meranie vyťaženia počas celého dňa a jednotlivých dní by neprinieslo použiteľné údaje, pretože pri tomto druhu aplikácie je dôležité vyťaženie databázy, a tá sa líši pre každú aplikáciu. Preto by napríklad výsledky testovania na náhodných dátach generovanými časťou aplikácie pre testovacie a predvádzacie účely neboli smerodajné. Aplikácia môže byť nasadená na akúkoľvek databázu a každá databáza použitá s touto aplikáciou má jeden spoločný rys a to, že budú na jej tabuľkách použité triggery. Preto je dôležité vedieť ako triggery danú databázu zaťažia. Generovanie grafov má taktiež dopad na rýchlosť aplikácie. Túto časť výpočtu je ale možné presunúť na odlišný server a tým zaťaženie databázového servera znížiť na minimum. Pri takejto architektúre by aplikácia pristupovala na server iba na získanie údajov zobrazených v grafe a z dôvodu údržby (kontrola počtu záznamov v monitorovacích tabuľkách a ich premazávanie ak presiahnu určitý limit zadaný užívateľom). Aplikácia umožňuje užívateľovi nastaviť interval generovania a znovu načítania grafov v nastaveniach, takže užívateľ môže regulovať týmto nastavením čiastočne aj vyťaženie aplikácie.

Prúd dát je simulovaný trvalým vkladaním záznamov do databáze. Generované boli také dáta, aby sa vykonal pri vložení jedného údaju každý z definovaných triggerov. Vkladané dáta vyhovovali každej z definovaných podmienok, ktoré sa počas testovania nachádzali na triggeroch. Týmto prístupom sa predišlo skresľovaniu údajov, ak by náhodne generované dáta pokryli stále odlišný počet triggerov. Pri prvom teste sa postupne vkladalo 10000 záznamov do sledovanej tabuľky. Tabuľka mala postupne naviazaných 0 až 8 triggerov, pričom hodnoty namerané pri tabuľke bez triggera slúžili ako kontrolné hodnoty. Testovala sa doba vykonania vloženia údajov (INSERT príkaz) a to tak, že sa zaznamenal čas pred a po operácii a tieto hodnoty sa od seba odčítali. Hodnoty sa uložili do textového súboru. Testovací kód vyzeral nasledovne:

```
/********************* testing start ******************/
DateTime queryStart = DateTime.Now;
DateTime queryEnd = DateTime.Now;
/********************* testing end ********************/

try
{
    /********************* testing start ******************/
```

```
        queryStart = DateTime.Now;
        /********************** testing end ********************/

        da.Fill(ds);

        /********************** testing start ******************/
        queryEnd = DateTime.Now;

        TimeSpan difference = queryStart - queryEnd;

        MySession.PerformanceLogs.Add(queryStart.ToString() + "|" + queryEnd.ToString() + "|" +
        difference.TotalMilliseconds + "|" + queryText);

        string output = "";

        foreach (string s in MySession.PerformanceLogs)
        {
            output += s + "\n";

        }

        FileHandling.SaveDataToFile(General.ApplicationLocation + @"App_Data\PerformanceLog.txt",
        output, false);

        /********************** testing end ********************/
    }
    catch (Exception ex)
    {
        EventLog e = new EventLog();
        e.LogEvent("ERROR", "Couldn't execute query. " + ex.Message);
    }
```

Ďalší test bol navrhnutý tak, aby bolo možné porovnať efektivitu implementácie jednotlivých triggerov, aby sa zistilo, aký typ systém zaťažuje najviac. Bola použitá tabuľka s jedným triggerom. Do tejto sa vkladali údaje, ktoré vyhovovali každému druhu triggera, takže sa opäť testovalo najväčšie možné zaťaženie systému. Test bol prevedený s 10000 údajmi.

Posledný test bol zameraný na testovanie účinnosti aplikácie oproti priamym dotazom do databázy. Vytvorená bola testovacia tabuľka s 1000000 záznamami s náhodnými hodnotami. Počas vytvárania testovacích dát boli k tabuľke naviazané triggery, takže sa zároveň zbierali údaje potrebné na tvorbu dashboardu. Tento test prebiehal tak, že sa sledovala doba dotazu a získania údajov z monitorovacích tabuliek a následne sa tie isté dáta získali priamo z tabuľky, kde boli vložené ekvivalentnými dotazmi.

Pri testoch, kde boli naviazaných na tabuľku až 8 triggerov súčasne, boli použité triggery rôzneho typu. Jeden trigger mal zloženú podmienku, štyri triggery mali jednoduchú podmienku a z týchto boli dva monitorovacie. Zvyšné tri triggery boli iba monitorovacie. Monitorovacie triggery sú výpočtovo náročnejšie, keďže sa pri vkladaní prehľadáva tabuľka, či nájdené dáta už neboli zaznamenané. Ak sa dané dáta nájdu, tak sa iba inkrementuje počítadlo výskytov daných dát, v opačnom prípade sa vloží nový záznam. Z toho dôvodu boli na testovanie použité práve prevažne monitorovacie triggery. Pri tomto type triggerov je dôležité, koľko záznamov je v monitorovacej tabuľke už zaznamenaných. Vyhľadávanie v menšom počte záznamov je rýchlejšie ako vyhľadávanie vo viacerých. Hornú hranicu počtu záznamov v monitorovacích tabuľkách je možno určiť v nastaveniach. Zároveň môže užívateľ sám určiť, v akom intervale sa bude táto hodnota kontrolovať. Dlhý čas medzi kontrolami môže zahltiť monitorovacie tabuľky zbytočnými dátami, ktoré budú

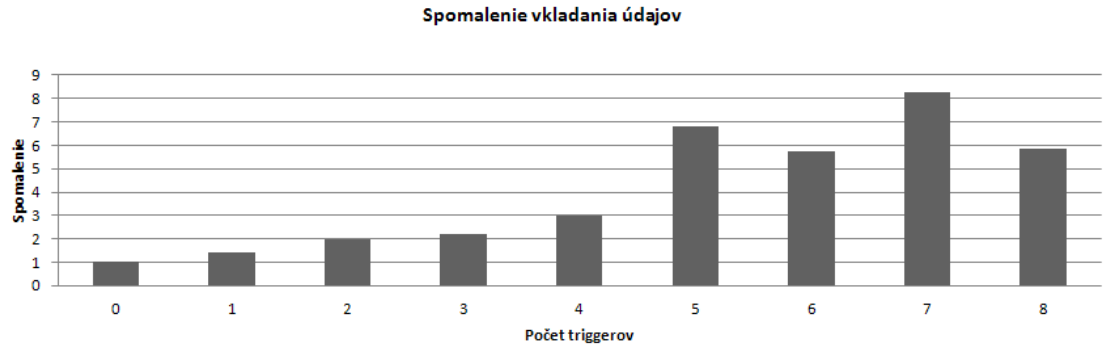
vkladanie iba spomaľovať, a zároveň dané záznamy nebudú použité v žiadnom grafe na dashboarde, takže ich ukladanie bude zbytočné. Krátke intervaly v kontrolovaní môže aplikáciu tiež zahltiť, takže tieto hodnoty je potrebné voliť v rozumných rozmedziach, ktoré záležia na počte zobrazovaných hodnôt v danom grafe a počte vkladaných údajov do monitorovanej tabuľky.

Optimalizácia, vo forme premazávanie údajov v monitorovacích tabuľkách, sa pri testovaní nepoužívala aby sa zamedzilo ovplyvneniu výsledkov merania doby dotazu nad databázou.

6.2 Namerané výsledky

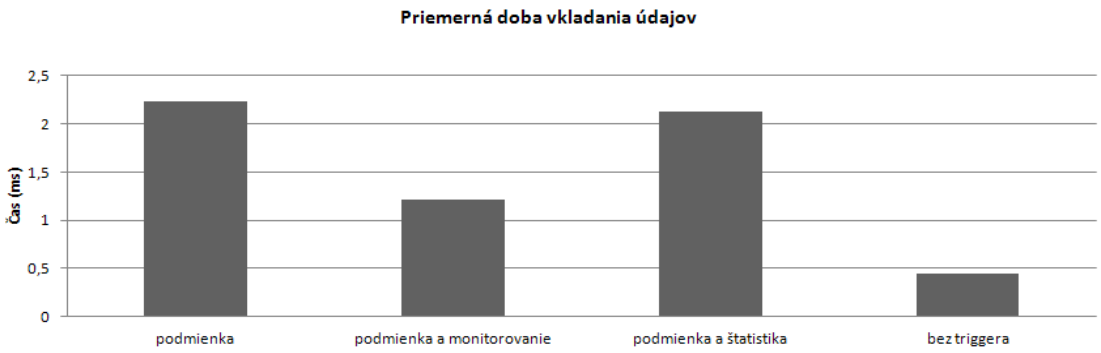
V nasledujúcich grafoch sú znázornené výsledky testov, opísaných v predchádzajúcej kapitole. Na obrázku 33 sú znázornené časové hodnoty vkladania údajov do tabuľky v závislosti na počte triggerov naviazaných k tejto tabuľke. V grafe 34 je znázornené spomalenie vzhľadom k hodnote nameranej pri vložení dát do tabuľky bez naviazaného triggera.

Obrázok 33: Časy vloženia údajov do tabuľky v závislosti na počte triggerov

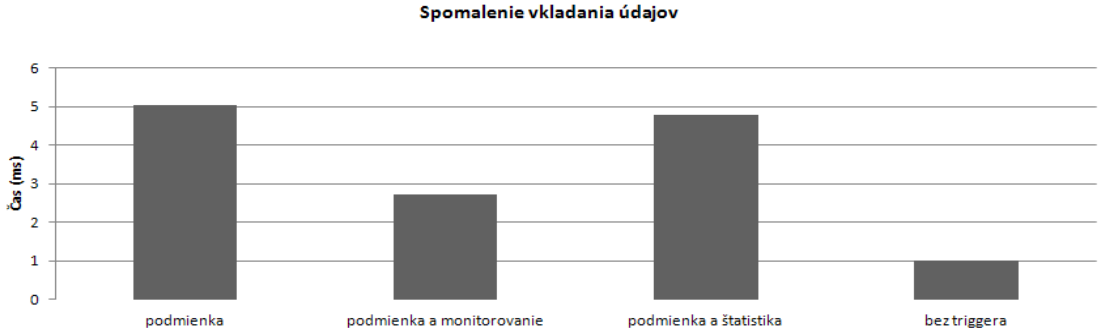


Obrázok 34: Spomalenie vloženia údajov do tabuľky v závislosti na počte triggerov

V ďalšom teste sme porovnávali priemernú dobu vykonávania vkladania údajov pri použití odlišných typov triggerov. Namerané hodnoty sú znázornené v grafoch na obrázku 35 a 36. V prvom je znázornená priemerná doba trvania vykonania príkazu a v druhom spomalenie oproti vloženia dát bez dodatočného triggera.



Obrázok 35: Časy vloženia údajov do tabuľky v závislosti na type triggera



Obrázok 36: Spomalenie vloženia údajov do tabuľky v závislosti na type triggera

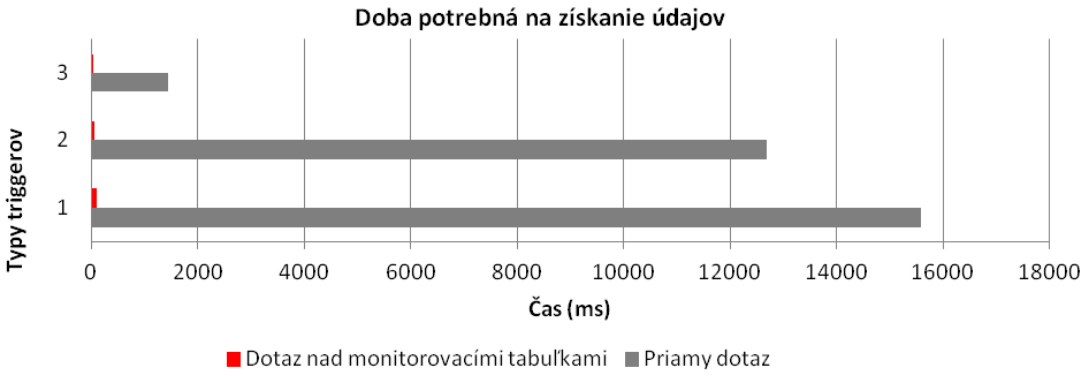
Posledný zo série testov bol zameraný na porovnanie získania údajov priamo z tabuľky, do ktorej sa vkladajú údaje a získania údajov z monitorovacích tabuliek. Namerané údaje sú uvedené v tabuľkách 1 a 2. V prvej sú uvedené časy pre získanie údajov z monitorovacích tabuliek spravovaných našou aplikáciou a v druhej časy na získanie tých istých dát priamo z tabuľky, do ktorej sa dáta z dátového prúdu vkladajú. Tieto údaje sú graficky znázornené v grafoch na obrázku 37 a 38.

Výber dát z monitorovacích tabuliek			
Dotaz	Doba dotazu (ms)	Počet vložených dát	Spomalenie
A (ekvivalent)	110,1584	1000000	1
B (ekvivalent)	70,1008	1000000	1
C (ekvivalent)	20,0288	1000000	1

Tabuľka 1: Časy získania údajov z monitorovacích tabuliek

Výber dát pomocou priamych dotazov			
Dotaz	Doba dotazu (ms)	Počet vložených dát	Spomalenie
A	15582,4064	1000000	141,4545
B	12697,2308	1000000	181,1281
C	1453,5078	1000000	72,57088

Tabuľka 2: Časy získania údajov priamo z tabuľky použitej na ukladanie dát



Obrázok 37: Časy získania údajov z monitorovacích tabuliek a z tabuliek zhromažďujúcich dáta (priamy dotaz)

Obrázok 38: Znázornenie zrýchlenia získania údajov z monitorovacích tabuliek v porovnaní so získavaním dát pomocou priamych dotazov

6.3 Vyhodnotenie

Z testov prevedených v predchádzajúcej kapitole môžeme usúdiť, že každým pridaním triggera sa vkladanie údajov do tabuľky spomalí. V závislosti na počte triggerov a ich type môže byť spomalenie až niekoľko násobné. V našom prípade bolo najväčšie, až 8 násobné spomalenie namerané v teste znázornenom na obrázku 34, v ktorom sa vkladalo 10000 položiek do sledovanej tabuľky s naviazanými siedmymi triggermi. Naviazanie 4 triggerov na tabuľku spomalí vkladanie údajov 2 až 3 násobne, čo môže byť v určitých prípadoch prijateľné v závislosti na rýchlosti príchodu dát. Druhým testom sme zistili, že najjednoduchší trigger (podmienka) sa nevykoná najrýchlejšie, ako by sa dalo predpokladať. Väčšie spomalenie na rozdiel od monitorovacieho triggera je dané množstvom údajov, ktoré sa postupne nahromadí v monitorovacích tabuľkách pre tento typ triggera. Preto je potrebné pri každom dashboarde pouvažovať na vhodných limitoch pre tieto tabuľky v závislosti na počte zobrazovaných dát.

Silná stránka aplikácie leží v získavaní údajov z monitorovacích tabuliek pre zostavenie grafov, testované tretím testom. Ako je možné vidieť v grafe na obrázku 37, sú časy potrebné na získanie údajov niekoľko násobne menšie pre získanie dát z monitorovacích tabuliek na rozdiel od získania dát

pomocou priameho dotazu. Preto je možné aplikáciu použiť na generovanie grafov v krátkych intervaloch a tým sledovať dynamicky sa meniace grafy. Zároveň boli testy prevedené na tabuľke s 1000000 záznamami, pritom databázy zaznamenávajúce tok dát túto métu hravo prekonávajú. So zvyšovaním počtu záznamov sa priame dotazy ešte spomaľujú, na rozdiel od našej aplikácie, kde doba trvanie dotazu je relatívne konštantná, keď sa aplikuje čistenie monitorovacích tabuliek.

7 Možné rozšírenia systému

Systém možno rozšíriť viacerými smermi. Užívateľovi je možné poskytnúť rozšírené nastavenia grafov, ako napríklad výber farieb, ktoré budú pri vykresľovaní použité. Ďalším možným rozšírením je umožniť zobrazenie dát zozbieraných viacerými triggerami v jednom grafe. Spolu s týmto rozšírením je možno implementovať synchronizáciu zbierania údajov jednotlivých triggerov, keďže dáta musia byť zosynchronizované, ak sa majú použiť v spoločnom grafe.

Zobrazenie grafov je možné prepracovať do odlišnej formy, napríklad by bolo možné použiť interaktívne grafy postavené na technológii JavaScript, ktoré by umožnili zobraziť výsek daného grafu a ponoriť sa hlbšie do analýzy úseku grafu.

Systém je taktiež možno dodatočne optimalizovať, aby sa skrátil čas načítania jednotlivých konfiguračných sekcií a obmedzila sa veľkosť generovaných stránok. Ďalším možným rozšírením v oblasti optimalizácie by bolo zavedenie vlastných indexov nad monitorovacími tabuľkami. Tieto by mali hlavne urýchliť vkladanie hlavne záznamov monitorovacích triggerov vytvorením indexu pre stĺpec `EntryData`, ktorý uchováva hodnotu, podľa ktorej sa vyhľadávajú duplicitné záznamy v tabuľke. Optimalizovať aplikáciu by pomohlo aj premiestnenie nastavenia maximálneho počtu záznamov k danému objektu triggera, keďže doterajšie nastavenia umožňujú túto hodnotu nastaviť iba globálne, alebo vypočítavať túto hodnotu dynamicky z počtu zobrazených údajov na grafe, kde je trigger použitý. Hlavnú optimalizáciu v užívateľskom rozhraní, ktorú by som odporúčal zaviesť, je implementácia podpory stránkovania pre grafy a triggery, keďže tieto objekty sú na stránke dynamicky zostavované a s pribúdajúcim množstvom triggerov sa načítanie stránky značne spomaľuje.

Aplikáciu možno rozšíriť aj pridaním ďalších typov triggerov a prehodnotiť spôsob ich implementácie, či nie je možné dodatočne obmedziť zaťaženie databázy týmito objektmi.

8 Záver

Úlohou tejto práce bolo vytvoriť aplikáciu umožňujúcu monitorovať akúkoľvek tabuľku vytvorenú v databázovom sklade MS SQL pomocou dynamicky meniaceho sa dashboardu a zároveň umožniť užívateľovi jednotlivé prvky dashboardu vytvárať a upravovať podľa jeho predstáv. Práca nadväzuje na semestrálny projekt, v ktorom boli priblížené teoretické predpoklady na vypracovanie práce a diskutoval sa návrh riešenia.

Bola prezentovaná tvorba dashboardov a ich návrh z grafického hľadiska, najmä ako zobraziť určité typy informácií a čomu sa pri tvorbe dashboardov vyvarovať. Tento proces bol znázornený na dashboarde z reálneho prostredia a samostatne na jednom vybranom type grafu. Bolo opísané, aké techniky je vhodné použiť pri spracovaní prúdu dát, aby bolo tieto možno jednoducho zobraziť dynamicky definovaným dashboardom a zároveň, aby získanie potrebných dát bolo uskutočniteľné v čo najkratšom čase. Počas celej práce bol kladený dôraz na dynamické vytváranie grafov, takže každý zvolený postup zohľadňoval túto skutočnosť. Detailne bol tento bod spracovaný v kapitole 4.3, v ktorej sa rozoberá aké možnosti má užívateľ pri definovaní grafov a aké hranice mu systém stanovuje. V kapitole, ktorá všeobecne hovorí o prúdových dátach, boli spomenuté možné optimalizácie pre prúdové dáta.

Následne sa v druhej časti práci venujem štruktúre aplikácie z fyzického a z logického hľadiska. V časti návrhu sa venujem stavbe aplikácie, takže je priblížená súborová štruktúra aplikácie a databázové objekty vytvárané touto aplikáciou. Na modelovanie aplikácie boli využité sekvenčné diagramy popisujúce dva hlavné procesy v aplikácii, a to vytváranie objektov a obnovenie grafu. Zároveň je stavba aplikácie ilustrovaná balíčkovým diagramom a diagramom nasadenia. V časti návrhu sa diskutujú aj možné implementačné prístupy k riešeniu problému a výberu jedného z nich.

Nasledovala implementácia riešenia v prostredí .NET jazykom C#. Na zber informácií boli využité triggery nasadené na sledované databázové tabuľky, ktoré zbierajú informácie do dedikovaných tabuliek. Užívateľ túto štruktúru vytvára cez užívateľské rozhranie zobrazené v akomkoľvek prehliadači. Na zobrazenie a generovanie grafov bol čiastočne použitý modul MSCharts dostupný na platforme .NET, čiastočne bolo implementované vlastné riešenie pre tie typy grafov, ktoré tento modul nepodporuje.

Pri meraní efektivity implementácie som sa zameral na testovanie dopadu použitia triggerov na rýchlosť vkladania informácií do databázových tabuliek a závislosť tejto od počtu triggerov. Porovnali sa aj rýchlosti vkladania údajov pre hlavné druhy triggerov. Posledný test bol zameraný na porovnanie rýchlosti výberu potrebných údajov na zostavenie grafov z monitorovacích tabuliek, zostavených aplikáciou, a priamych dotazov do tabuľky s miliónom záznamov. V práci som následne priblížil, akými smermi je aplikácia rozšíriteľná a aké časti sa môžu optimalizovať do budúcnosti.

Literatúra

[1] FEW, S.: *Information Dashboard Design: The Effective Visual Communication of Data*. O'Reilly Media, 2006, ISBN 978-0596100162.

[2] BALANCED SCORECARD: *What is the Balanced Scorecard?*. [online], 2011 [cit. 2011-12-04].
URL <http://www.balancedscorecard.org/BSCResources/AbouttheBalancedScorecard/tabid/55/Default.aspx>

[3] OLAP: *OLAP History*. [online], 2011 [cit. 2011-12-04].
URL <http://www.olap.com/w/index.php/Category:OLAP_History>

[4] BUSINESS DICTIONARY: *executive information system (EIS)*. [online], 2011 [cit. 2011-12-04].
URL <http://www.businessdictionary.com/definition/executive-information-system-EIS.html>

[5] PETTERSSON, R.: *Visual Information*. Educational Technology Publications, Incorporated, 1993, ISBN 0877782628.

[6] SHANNON, C., E., WEAVER, W.: *The mathematical theory of communication*. University of Illinois Press (1998), 1949, ISBN 0252725484.

[7] INETSOFT: *Dashboard examples | Product features*. [online], 2011 [cit. 2011-12-04].
URL <http://www.inetsoft.com/products/dashboard_examples/>

[8] LEWIS WIRE: *iDashboards News Desk*. [online], 2011 [cit. 2011-12-04].
URL <http://www.lewiswire.com/us/lewiswire/iDashboards/Dashboards-in-Vertical-Markets-Lessons-Every-Industry-Should-Heed/c/188/n/128>

[9] DASHBOARDZONE: *Why do you hate Gauges, Dials and Speedometers*. [online], 2011 [cit. 2011-12-04].
URL <http://www.dashboardzone.com/why-do-you-hate-gauges-dials-and-speedometers>

[10] JUICEANALYTICS: *Dashboard Gauges: Indulgence with Restraint*. [online], 2011 [cit. 2011-12-04].
URL <http://www.juiceanalytics.com/writing/dashboard-gauges-indulgence-with-restraint/>

[11] MAGIC DASHES: *Gauges are the devil – Magic Dashboards*. [online], 2011 [cit. 2011-12-04].
URL <http://magicdashes.wordpress.com/2011/08/14/gauges-are-the-devil/>

[12] TARGETDASHBOARD: *Target Dashboard the Online KPI Dashboard*. [online], 2011 [cit. 2011-12-04].
URL <http://www.targetdashboard.com/site/features.aspx>

[13] TUFTE, E., R.: *The Visual Display of Quantitative Information*. Graphics Pr, 2001, ISBN 978-0961392147.

[14] MCGREGOR, A.: *Crash Course on Data Stream Algorithms*. 2010. Podklady k prednáške.
URL <http://www.cs.umass.edu/~mcgregor/slides/10-jhu1.pdf>

[15] GABER, M., M., ZASLAVSKY, A., KRISHNASWAMY, S.: Mining Data Streams: A Review. In *ACM SIGMOD Special Interest Group on Management of Data*, New York, NY, USA, 2005, s. 18-26. ISSN 0163-5808.
URL <http://doi.acm.org/10.1145/1083784.1083789>

[16] TATBUL, N.: Load Shedding. In *ACM SIGMOD ACM Special Interest Group on Management of Data ACM Association for Computing Machinery*, New York, NY, USA. 2009, ISBN 978-1-60558-551-2.
URL <http://www.inf.ethz.ch/personal/tatbul/publications/load_sheddin g_EDBS.pdf>

[17] CORMODE, G.: *Fundamentals of Analyzing and Mining Data Streams*. 2007. Podklady k prednáške.
URL <http://dimacs.rutgers.edu/~graham/pubs/papers/cormodewdsa.pdf>

[18] CORMODE, G., MUTHUKRISHNAN, S.: *An Improved Data Stream Summary: The Count-Min Sketch and its Applications.* 2004, ISBN 978-3-540-21258-4.
URL <http://www.eecs.harvard.edu/~michaelm/CS222/countmin.pdf>

[19] GIBBONS, P., B., YOSSI, M.: Synopsis Data Structures for Massive Data Sets. In *ACM SIGACT Special Interest Group on Algorithms and Computation Theory*
SIAM Society for Industrial and Applied Mathematics. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1999. ISBN 0-89871-434-6.
URL <http://theory.stanford.edu/~matias/papers/synopsis.pdf>

[20] HEMMENDINGER, D., RALSTON, A., REILLY, E., D.: *Encyclopedia of Computer Science*. Grove's Dictionaries, 2000, ISBN 156159248X.
URL <http://www.maffeis.com/articles/research/client_server.pdf>

[21] GOULD, L., ZANEVSKY, A., KLINE, K.: *Transact-SQL Programming*. O'Reilly Media, 1999, ISBN 1565924010.

[21] BLOOBERRY: *HTML Overview*. [online], 1996 – 2005 [cit. 2011-12-11].
URL <http://blooberry.com/indexdot/history/html.htm>

[22] WILTON–JONES, M.: *HOW TO CREATE – JavaScript History*. [online], 2005 [cit. 2011-12-11].
URL <http://www.howtocreate.co.uk/jshistory.html>

[23] XML UK: *CSS History & Information*. [online], 2008 [cit. 2011-12-11].
URL <http://www.xmluk.org/css-history-information.htm>

[24] OLKEN, F., ROTEM, D.: *Simple Random Sampling From Relational Databases*.,
Computer Science Research Dept. Lawrence Berkeley Laboratory, 1986, ISBN 0-934613-18-4.
URL <http://www.vldb.org/conf/1986/P160.PDF>

Príloha 1 – Obsah priloženého CD

Priložené CD obsahuje súbory so zdrojovými kódmi pre aplikáciu a technický text, dokumentáciu na inštalovanie systému, programovou dokumentáciou vo formáte CHM a dokumentáciu vo formáte HTML, predstavujúcu jednoduchý návod na použitie. Súborová štruktúra CD je nasledujúca (uvedené najpodstatnejšie súbory a adresáre):

\

- Sources – zdrojové súbory aplikácie
  - BusinessLayer
  - DataLayer
  - GlobalHelper
  - WebSite
  - README.txt
  - WebSite.sln
- Documentation – zdrojové súbory dokumentácie a generované výstupy dokumentácie
  - Thesis text
  - HTML help
    - Help.aspx_files
    - Help.aspx.htm
  - APIReference.chm
  - README.txt