Valsts reģionālās attīstības aģentūra



|  |  |
| --- | --- |
|  |  |

Par Valsts informācijas sistēmu savietotāja, Latvijas valsts portāla www.latvija.lv un elektronisko pakalpojumu izstrāde un uzturēšana

3.daļa "VISS un Portāla jaunu un esošo moduļu papildinājumu izstrāde, ieviešana, garantijas apkalpošana un uzturēšana saskaņā ar tehnisko specifikāciju"

VISS izstrāde

Vadlīnijas

VRAA-6\_15\_11\_58-VISS\_2010-VISS-VDL

15.10.2013. versija 2.19

Rīgā 2013

|  |  |  |
| --- | --- | --- |
|  | **IEGULDĪJUMS TAVĀ NĀKOTNĒ** |  |

Dokumenta identifikācija

|  |  |
| --- | --- |
| Dokumenta ID: | VRAA-6\_15\_11\_58-VISS\_2010-VISS-VDL-V2.19-15.10.2013. |
| Dokumenta nosaukums: | Par Valsts informācijas sistēmu savietotāja, Latvijas valsts portāla www.latvija.lv un elektronisko pakalpojumu izstrāde un uzturēšana.  3.daļa "VISS un Portāla jaunu un esošo moduļu papildinājumu izstrāde, ieviešana, garantijas apkalpošana un uzturēšana saskaņā ar tehnisko specifikāciju".  VISS izstrāde.  Vadlīnijas. |
| Dokumenta kods: | VRAA-6\_15\_11\_58-VISS\_2010-VISS-VDL |
| Versija: | Versija 2.19, Laidiens 15.10.2013. (saīsināti V2.19 15.10.2013.) |

Saskaņojumi

|  |  |  |  |
| --- | --- | --- | --- |
| Organizācija | Vārds, uzvārds, amats | Datums | Paraksts |
| Valsts reģionālās attīstības aģentūra | Atbildīgā persona no Pasūtītāja puses |  |  |
| SIA "ABC software" | J.Korņijenko, projekta vadītājs par tehniskiem jautājumiem no Izpildītāja puses | 15.10.2013. |  |
| SIA "ABC software" | M.Pētersons, projekta vadītājs par administratīviem jautājumiem no Izpildītāja puses | 15.10.2013. |  |
| SIA "ABC software" | E.Blumberga, projekta kvalitātes kontroles vadītāja | 15.10.2013. |  |

Izmaiņu vēsture

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Versija | Datums | Apraksts | Organizācija | Autors |
| 0.09 | 11.07.2006. | Izveidota dokumenta sākotnējā versija. | SIA „ABC software” | J.Korņijenko |
| 1.00 | 10.11.2006. | Pievienots IVIS Framework (IVIS portāla master page) apraksts (2.8.nodaļa).  Papildināta 2.6.7. sadaļa. | SIA „ABC software” | A.Vorobjovs,  J.Korņijenko |
| 2.00 | 20.10.2008. | IVIS bibliotēku lietošana ir izdalīta cita dokumenta – Koplietojamo bibliotēku apraksts, pievienotas sadaļas Produktu versiju piešķiršana un Projekta informācijas datne. | SIA „ABC software” | J.Gekišs |
| 2.10 | 14.04.2010. | Iestrādātas redakcionālas izmaiņas. | SIA „ABC software” | J.Gekišs,  J.Korņijenko |
| 2.11 | 06.05.2010. | Iestrādātas redakcionālas izmaiņas. | SIA „ABC software” | I.Tuleiko |
| 2.12 | 22.03.2012. | Papildināta informācija par kodu vārdtelpām, pievienots 6.nodalījums – VISS vizuālā dizaina vadlīnijas. | SIA „ABC software | J.Korņijenko,  M.Gasparoviča |
| 2.13 | 20.11.2012. | Pievienots 7.nodalījums | SIA „ABC software” | J.Korņijenko |
| 2.14 | 04.12.2012. | Iekļauti papildinājumi Projekta attīstības gaitā:   1. Koplietojamas Meklētāja XML shēmas spatialCoverage elementam varētu būt vairākas vērtības. 2. Iekļauti labojumi saskaņā ar NI Nr.62 (VRAA-6\_15\_11\_58-VISS\_2010-NI-62-VDL). | SIA „ABC software” | J.Korņijenko,  M.Gasparoviča |
| 2.15 | 10.01.2013. | Papildināta sadaļa „VISS Meklētāja koplietojamās datu shēmas apraksts”. | SIA „ABC software” | J.Korņijenko |
| 2.16 | 24.05.2013. | Pievienota sadaļa „Tīkla servisu izstrāde”, precizēta sadaļa „Komponenšu piegāde”. | SIA „ABC software” | J.Korņijenko |
| 2.17 | 22.08.2013. | Pievienota 7.2.nodaļa. | SIA „ABC software” | V.Nazaruks |
| 2.18 | 15.10.2013. | Papildināta 7.nodalījums saskaņā ar IP Nr.15 realizāciju. | SIA „ABC software” | J.Korņijenko |
| 2.19 | 14.02.2014 | Pievienota 7.3 nodaļa. | SIA „ABC software” | S.Siksna |

**Satura rādītājs**

[Attēlu saraksts 8](#_Toc380139423)

[1. Ievads 10](#_Toc380139424)

[1.1. Dokumenta nolūks 10](#_Toc380139425)

[1.2. Definīcijas un saīsinājumi 10](#_Toc380139426)

[1.3. Saistītie dokumenti 10](#_Toc380139427)

[1.4. Dokumenta pārskats 11](#_Toc380139428)

[2. Kopējas programmatūras izstrādes vadlīnijas 12](#_Toc380139429)

[2.1. Kopējas prasības 12](#_Toc380139430)

[2.1.1. Programmēšanas vide 12](#_Toc380139431)

[2.1.2. Komponentes atkārtota izmantošana 12](#_Toc380139432)

[2.1.3. Komponenšu piegāde 13](#_Toc380139433)

[2.2. Konfigurācijas vadība 13](#_Toc380139434)

[2.2.1. Konfigurācijas krātuve 13](#_Toc380139435)

[2.2.2. Konfigurācijas informācijas raksturs 13](#_Toc380139436)

[2.3. Izpildes trasēšana 13](#_Toc380139437)

[2.3.1. Izpildes skaitītāja izvietojums 14](#_Toc380139438)

[2.3.2. Izpildes skaitītāju nosaukšana 14](#_Toc380139439)

[2.3.3. Izpildes skaitītāja datu analizēšana 14](#_Toc380139440)

[2.3.4. Tipiskie eksponētie izpildes skaitītāji 14](#_Toc380139441)

[2.4. Datu bāzes piekļuve 14](#_Toc380139442)

[2.4.1. Uzticamais lietotājs 14](#_Toc380139443)

[2.4.2. Noklusētais datu bāzes izolācijas līmenis 15](#_Toc380139444)

[2.4.3. Glabājamās procedūras 15](#_Toc380139445)

[2.4.4. Datu bāzes resursi 15](#_Toc380139446)

[2.4.5. Transakcijas 15](#_Toc380139447)

[2.5. Tīkla servisu izstrāde 15](#_Toc380139448)

[2.5.1. Kopējas prasības 15](#_Toc380139449)

[2.5.2. Drošības profils (security profile) 16](#_Toc380139450)

[2.5.3. Kļūdu un izņēmuma situāciju apstrāde 16](#_Toc380139451)

[3. .NET programmēšanas standarti 17](#_Toc380139452)

[3.1. C# projektu izstrādes vadlīnijas 17](#_Toc380139453)

[3.1.1. Projektu nosaukumi 17](#_Toc380139454)

[3.1.2. Produktu versiju piešķiršana 17](#_Toc380139455)

[3.1.2.1. Produkta versija 17](#_Toc380139456)

[3.1.2.2. Projekta versija 17](#_Toc380139457)

[3.1.2.3. Laidiena versija 17](#_Toc380139458)

[3.1.2.4. Datubāzes versija 18](#_Toc380139459)

[3.1.2.5. Produktu versiju numuru izmaiņas nosacījumi 18](#_Toc380139460)

[3.1.3. Projekta informācijas datne 18](#_Toc380139461)

[3.1.4. Projekta izmaiņu aprakstīšanas datne 19](#_Toc380139462)

[3.2. Vienošanās par nosaukumu piešķiršanu (*naming convention*) 20](#_Toc380139463)

[3.3. Vienošanās par izvietošanas secību 22](#_Toc380139464)

[3.4. Labošanas iespēju nosacījumi 23](#_Toc380139465)

[3.4.1. C♯ avota faili (Sourcefiles) 23](#_Toc380139466)

[3.4.2. Katalogu izkārtojums (Directory Layout) 23](#_Toc380139467)

[3.5. C# izvietojuma nosacījumi 23](#_Toc380139468)

[3.6. Komentāru pievienošanas nosacījumi 26](#_Toc380139469)

[3.6.1. Blokveida komentāri (Block Comments) 26](#_Toc380139470)

[3.6.2. Vienrindas komentāri (Single Line Comments) 27](#_Toc380139471)

[3.6.3. Dokumentācijas komentēšana (Documentation Comments) 27](#_Toc380139472)

[3.6.4. Programmatūras koda datnes virsraksts (Copyright) 28](#_Toc380139473)

[4. Nosaukumu piešķiršana MS SQL datu bāzes objektiem 29](#_Toc380139474)

[5. MS SQL Serveru programmēšanas vadlīnijas 35](#_Toc380139475)

[5.1. Programmatūras koda lasāmība un formāts 35](#_Toc380139476)

[5.2. Datu tipi 35](#_Toc380139477)

[5.3. Saglabātās procedūras 36](#_Toc380139478)

[5.4. Veiktspēja 36](#_Toc380139479)

[5.5. Dažādi ieteikumi 38](#_Toc380139480)

[6. VISS vizuālā dizaina vadlīnijas 40](#_Toc380139481)

[6.1. Vispārīgas prasības 40](#_Toc380139482)

[6.2. Radiopoga 40](#_Toc380139483)

[6.2.1. Izmantošanas nolūks 40](#_Toc380139484)

[6.2.2. Konceptuāls apraksts 40](#_Toc380139485)

[6.2.3. Lietošanas rekomendācijas 40](#_Toc380139486)

[6.3. Hipersaite 41](#_Toc380139487)

[6.3.1. Izmantošanas nolūks 41](#_Toc380139488)

[6.3.2. Konceptuāls apraksts 41](#_Toc380139489)

[6.3.3. Lietošanas rekomendācijas 41](#_Toc380139490)

[6.4. Izvēles rūtiņa 41](#_Toc380139491)

[6.4.1. Izmantošanas nolūks 41](#_Toc380139492)

[6.4.2. Konceptuāls apraksts 41](#_Toc380139493)

[6.4.3. Lietošanas rekomendācijas 42](#_Toc380139494)

[6.5. Pogas 42](#_Toc380139495)

[6.5.1. Izmantošanas nolūks 42](#_Toc380139496)

[6.5.2. Konceptuāls apraksts 42](#_Toc380139497)

[6.5.3. Lietošanas rekomendācijas 42](#_Toc380139498)

[6.5.4. Pogu izkārtojuma rekomendācijas 42](#_Toc380139499)

[6.6. Režģis 43](#_Toc380139500)

[6.6.1. Lapošana 43](#_Toc380139501)

[6.6.2. Cilne 43](#_Toc380139502)

[6.7. Paziņojumi 44](#_Toc380139503)

[6.7.1. Izmantošanas nolūks 44](#_Toc380139504)

[6.7.2. Konceptuāls apraksts 44](#_Toc380139505)

[6.7.3. Lietošanas rekomendācijas 45](#_Toc380139506)

[6.8. Paskaidre 46](#_Toc380139507)

[6.8.1. Izmantošanas nolūks 46](#_Toc380139508)

[6.8.2. Konceptuāls apraksts 46](#_Toc380139509)

[6.8.3. Lietošanas rekomendācijas 47](#_Toc380139510)

[6.9. Brīvas ievades lauciņš 47](#_Toc380139511)

[6.9.1. Izmantošanas nolūks 47](#_Toc380139512)

[6.9.2. Konceptuāls apraksts 47](#_Toc380139513)

[6.9.3. Lietošanas rekomendācijas 48](#_Toc380139514)

[6.10. Izvēlne 48](#_Toc380139515)

[6.10.1. Izmantošanas nolūks 48](#_Toc380139516)

[6.10.2. Konceptuāls apraksts 48](#_Toc380139517)

[6.10.3. Lietošanas rekomendācijas 49](#_Toc380139518)

[6.11. Datums 49](#_Toc380139519)

[6.12. Gaitas līnija un ielāde 49](#_Toc380139520)

[6.12.1. Izmantošanas nolūks 49](#_Toc380139521)

[6.12.2. Konceptuāls apraksts 49](#_Toc380139522)

[6.12.3. Lietošanas rekomendācijas 50](#_Toc380139523)

[6.13. Citi elementi 50](#_Toc380139524)

[6.13.1. Izbraucošā izvēle 50](#_Toc380139525)

[6.13.2. Izkrītošais saraksts 50](#_Toc380139526)

[6.13.3. Saraksts 50](#_Toc380139527)

[7. VISS koplietojamo komponenšu izmantošana 51](#_Toc380139528)

[7.1. VISS Meklētāja koplietojamās datu shēmas apraksts 51](#_Toc380139529)

[7.1.1. Dokumenta metadati 51](#_Toc380139530)

[7.1.2. Dokumenta saturs 52](#_Toc380139531)

[7.2. Lietotāju profilu īpašību izgūšanas pakalpju apraksts 53](#_Toc380139532)

[7.2.1. Lietotāju profilu īpašības 54](#_Toc380139533)

[7.2.2. Lietotāju identificēšana un drošība 56](#_Toc380139534)

[7.2.3. Pakalpju operācijas 57](#_Toc380139535)

[7.2.3.1. Pakalpes UserProfileByIdentifierService operāciju izsaukumu piemēri 67](#_Toc380139536)

[7.2.4. Tipiskie lietošanas scenāriji 68](#_Toc380139537)

[7.3. No pārlūka izmantojamu (AJAX) tīmekļa pakalpju izstrāde (AJAX enabled services) 68](#_Toc380139538)

[7.3.1. Risinājuma arhitektūra 68](#_Toc380139539)

[7.3.2. ajaxCallProcessor Sitecore konfigurācija 69](#_Toc380139540)

[7.3.3. Servisu izstrāde 71](#_Toc380139541)

[7.3.3.1. REST serviss 71](#_Toc380139542)

[7.3.3.2. SOAP serviss 71](#_Toc380139543)

[7.3.4. Piemēri 73](#_Toc380139544)

# Attēlu saraksts

[1.attēls. Tīmekļa servisu arhitektūra 16](#_Toc380139545)

[2.attēls. Radiopogas vizuālais izskats 40](#_Toc380139546)

[3.attēls. Radiopogas ar paskaidri 40](#_Toc380139547)

[4.attēls. Hipersaite 41](#_Toc380139548)

[5.attēls. Hipersaites tabulā 41](#_Toc380139549)

[6.attēls. Izvēles rūtiņa 41](#_Toc380139550)

[7.attēls. Izvēles rūtiņa ar paskaidri 41](#_Toc380139551)

[8.attēls. Svarīgākā poga, aktīva 42](#_Toc380139552)

[9.attēls. Mazāk svarīga poga, aktīva 42](#_Toc380139553)

[10.attēls. Neaktīva poga 42](#_Toc380139554)

[11.attēls. Aktīva svarīgā poga ar paskaidrojošo tekstu 42](#_Toc380139555)

[12.attēls. Lapošana 43](#_Toc380139556)

[13.attēls. Lapošana tabulā 43](#_Toc380139557)

[14.attēls. Cilnes 43](#_Toc380139558)

[15.attēls. Informatīvs bloķējošs paziņojums, kam vajadzīgs apstiprinājums 45](#_Toc380139559)

[16.attēls. Informatīvs bloķējošs paziņojums ar apstiprinājumu (pogas dažādas nozīmības) 45](#_Toc380139560)

[17.attēls. Informatīvs paziņojums par veiksmīgi veiktu darbību 45](#_Toc380139561)

[18.attēls. Informatīvs paziņojums – informācija lietotājam 45](#_Toc380139562)

[19.attēls. Kļūdas paziņojums 46](#_Toc380139563)

[20.attēls. Skaidrojošs informatīvs paziņojums 46](#_Toc380139564)

[21.attēls. Informatīva paskaidre (neizvērsta) 46](#_Toc380139565)

[22.attēls. Informatīva paskaidre (izvērsta) 46](#_Toc380139566)

[23.attēls. Validācijas kļūdas paziņojums 46](#_Toc380139567)

[24.attēls. Paskaidre par kļūdu (neizvērsta) 47](#_Toc380139568)

[25.attēls. Paskaidre par kļūdu (izvērsta) 47](#_Toc380139569)

[26.attēls. Brīvās ievades lauciņš 47](#_Toc380139570)

[27.attēls. Brīvās ievades lauciņš ar paskaidrojošo tekstu 47](#_Toc380139571)

[28.attēls. Obligāti ievadāmie lauki 48](#_Toc380139572)

[29.attēls. Paroles ievade 48](#_Toc380139573)

[30.attēls. Kreisā izvēlne savērsta 48](#_Toc380139574)

[31.attēls. Kreisā izvēlne izvērsta (aktuālais ieraksts treknrakstā) 48](#_Toc380139575)

[32.attēls. Datuma informācijas izsaukšanas funkcionalitāte. 49](#_Toc380139576)

[33.attēls. Datuma ievade/izvēle. 49](#_Toc380139577)

[34.attēls. Izvēlēts datums. 49](#_Toc380139578)

[35.attēls. Gaitas līnija 49](#_Toc380139579)

[36.attēls. Ielādējas, gaidiet 50](#_Toc380139580)

[37.attēls. Izbraucošā izvēle ar teksta lauku blakus 50](#_Toc380139581)

[38.attēls. Izkrītošais saraksts 50](#_Toc380139582)

[39.attēls. Saraksts 50](#_Toc380139583)

[40.attēls. Lietotāja profila īpašības dažādās lietotnēs 54](#_Toc380139584)

[41.attēls. Lietotāju profilu īpašību izgūšanas pakalpju saskarne 56](#_Toc380139585)

[42.attēls. Drošības talonu apmaiņa, griežoties pie *PFAS.STS FP* 57](#_Toc380139586)

[43.attēls. XML elementa *GetPropertyRequest* struktūras shēma 59](#_Toc380139587)

[44.attēls. XML elementa *GetPropertyByIdentifierRequest* struktūras shēma 60](#_Toc380139588)

[45.attēls. XML elementa *GetPropertyResponse* struktūras shēma 60](#_Toc380139589)

[46.attēls. XML elementa *GetMultiplePropertiesRequest* struktūras shēma 61](#_Toc380139590)

[47.attēls. XML elementa *GetMultiplePropertiesByIdentifierRequest* struktūras shēma 61](#_Toc380139591)

[48.attēls. XML elementa *GetMultiplePropertiesResponse* struktūras shēma 62](#_Toc380139592)

[49.attēls. XML elementa *GetAllPropertiesRequest* struktūras shēma 62](#_Toc380139593)

[50.attēls. XML elementa *GetAllPropertiesByIdentifierRequest* struktūras shēma 62](#_Toc380139594)

[51.attēls. XML elementa *GetAllPropertiesResponse* struktūras shēma 63](#_Toc380139595)

[52.attēls. XML elementa *SetPropertyRequest* struktūras shēma 63](#_Toc380139596)

[53.attēls. XML elementa *SetPropertyByIdentifierRequest* struktūras shēma 64](#_Toc380139597)

[54.attēls. XML elementa *SetMultiplePropertiesRequest* struktūras shēma 65](#_Toc380139598)

[55.attēls. XML elementa *SetMultiplePropertiesByIdentifierRequest* struktūras shēma 66](#_Toc380139599)

[56.attēls. XML elementa *GetPropertiesDefinitionsRequest* struktūras shēma 66](#_Toc380139600)

[57.attēls. XML elementa *GetPropertiesDefinitionsResponse* struktūras shēma 66](#_Toc380139601)

[58. attēls. REST servisa izmantošanas diagramma 69](#_Toc380139602)

[59. attēls. Net.pipe servisa izmantošanas diagramma 69](#_Toc380139603)

# Ievads

## Dokumenta nolūks

Dokuments „VISS izstrāde” satur vadlīnijas un noteikumus, kas izstrādātājiem ir obligāti jāņem vērā izstrādājot IVIS projekta programmnodrošinājumus. Dokumentā aprakstīti nosacījumi, kas ņemami vērā piešķirot nosaukumus, kā arī izstrādes rekomendācijas. Šis dokuments ir paredzēts SIA „ABC software” izstrādātājiem, kā arī citiem iesaistītajiem izstrādātājiem, kas līdzdarbojas VISS programmnodrošinājumu izstrādē un pilnveidošanā.

## [Definīcijas un saīsinājumi](#_Toc65487056)

Apzīmējumu un terminu vārdnīca pieejama saistītajā dokumentā [2].

## Saistītie dokumenti

Dokuments ir izstrādāts, balstoties uz šādiem dokumentiem:

1. .NET Programming Standards and Naming Conventions (atrodams: <http://www.irritatedvowel.com/Programming/Standards.aspx>).
2. „Valsts informācijas sistēmu savietotāja, Latvijas Valsts portāla www.latvija.lv un elektronisko pakalpojumu izstrāde un uzturēšana”. Iepirkuma priekšmeta 3.daļa - VISS un Portāla jaunu un esošo moduļu papildinājumu izstrāde, ieviešana, garantijas apkalpošana un uzturēšana saskaņā ar tehnisko specifikāciju. Terminu un saīsinājumu indekss. (VRAA-6\_15\_11\_58-VISS\_2010-TSI).
3. Database object naming conventions (atrodams: <http://vyaskn.tripod.com/object_naming.htm>).
4. VISS sistēmas žurnāls. Koplietojuma bibliotēku apraksts(VRAA-6\_15\_11\_58-VISS\_2010-VISS\_ZUR-KBA)

1. <http://code.msdn.microsoft.com/sourceanalysis>
2. Metadatu un e-pakalpojumu identifikācija. Standarts ( VRAA-6\_15\_11\_58-VISS\_2010-MET\_EPAK\_ID-STD-V1.03-21.02.2012.).
3. Clusterpoint dokumentācija (atrodama: <http://www.clusterpoint.com/>).
4. SOAP12. [http://www.w3.org/TR/2003/REC-soap12-part1-20030624/](http://mail.abcsoftware.lv/exchweb/bin/redir.asp?URL=http://www.w3.org/TR/2003/REC-soap12-part1-20030624/).
5. Par Valsts informācijas sistēmu savietotāja, Latvijas valsts portāla www.latvija.lv un elektronisko pakalpojumu izstrāde un uzturēšana. 3.daļa "VISS un Portāla jaunu un esošo moduļu papildinājumu izstrāde, ieviešana, garantijas apkalpošana un uzturēšana saskaņā ar tehnisko specifikāciju". VISS Drošības talonu serviss. Programmētāja rokasgrāmata. (VRAA-6\_15\_11\_58-VISS\_2010-DTS-PR).

## Dokumenta pārskats

Dokuments sastāv no 7 nodalījumiem:

* Dokumenta ievads – aprakstīts dokumenta nolūks, termini un pieņemtie apzīmējumi, kā arī norādīta saistība ar citiem dokumentiem un materiāliem;
* Kopējās programmatūras izstrādes vadlīnijas – aprakstītas kopējās prasības, konfigurācijas vadību, izpildes trasēšanu un piekļuvi datu bāzei;
* .NET programmēšanas standarti – aprakstītas vienošanās par nosaukumu piešķiršanu, izvietošanas secību, labošanas iespēju nosacījumiem, C# izvietojuma nosacījumiem;
* Nosaukumu piešķiršana MS SQL datubāzes objektiem – aprakstīta nosaukumu piešķiršana MS SQL datubāzes objektiem;
* MS SQL Serveru programmēšanas vadlīnijas – aprakstīta programmatūras koda lasāmība un formāts, datu tipi, saglabātās procedūras, veiktspēja un citi ieteikumi;
* VISS vizuālā dizaina vadlīnijas – aprakstītas vispārējās prasības un atsevišķu kontroļu lietošanas nosacījumi;
* VISS meklētāja koplietojamās datu shēmas apraksts – aprakstītas prasības dokumentu metadatiem un dokumenta saturam.

# Kopējas programmatūras izstrādes vadlīnijas

## Kopējas prasības

Šajā nodaļā aprakstītas galvenās projektēšanas un realizācijas metodes, kas tiek izmantotas visām VISS risinājuma komponentēm.

Vadlīnijas, kas saistītas ar dažādiem VISS risinājumu izveides aspektiem, tiek iedalītas vairākās grupās:

1. Programmatūras izstrādes vadlīnijas.
2. .NET programmēšanas standarti un vienošanās par nosaukumu piešķiršanu.
3. Vienošanās par nosaukumu piešķiršanu Microsoft (turpmāk tekstā MS) SQL datu bāzes objektiem.
4. MS SQL Serveru programmēšanas vadlīnijas.

Atsevišķos gadījumos, veicot programmatūras izstrādi, varētu tikt definētas citas izstrādes rekomendācijas vai nosaukumu piešķiršanas noteikumi. Ja tas tiek darīts, tad jāsaskaņo ar VISS turētāju.

### Programmēšanas vide

Programmatūras risinājuma moduļu izstrāde notiek *Visual Studio* 2010 vai 2012 vidē. Risinājuma komponentes izstrādāšanai jāizmanto *Visual* C#, izņemot specifiskus līdzekļus, kuri prasa ciešu integrāciju ar Windows zema līmeņa lietojumprogrammas saskarni (*Application Program Interface*, turpmāk tekstā API).

### Komponentes atkārtota izmantošana

Visur, kur vien iespējams, komponentes jāprojektē, paredzot atkārtotas izmantošanas iespēju. Tas nozīmē, ka:

1. gadījums, kad izstrādātās komponentes programmatūras kods tiek pārkopēts citā komponentē, ir uzskatāms par sliktu programmēšanas stilu. Gadījumos, kad atkārtojamais programmatūras koda fragments nav lietots vispārēji, pieļaujams līdzīgā programmatūras koda pielietojums, lai padarītu divas funkcijas pilnīgi neatkarīgas;
2. izveidojot protokola informācijas apmaiņas specifiskās komponentes, tādas kā: Web *Service*, ASPX *page* vai klienta lietojumprogrammatūra (*front-end*) *WinForms*, programmatūras kods, kas izpilda biznesa pārbaudes, pieprasījumus citām servera puses komponentēm un IS servisiem, ir jāievieto atsevišķā biznesa klasē (komponentē), lai to būtu iespējams izmantot atkārtoti (arī šī protokola specifiskās komponentes ietvaros);

Risinājuma komponentes programmatūras kodam ir jābūt bāzētam uz *pašdokumentējoša* programmatūras koda pieeju, t.i., programmatūras kodam ir jābūt viegli lasāmam un jābūt komentētam tikai tad, kad nepieciešams skaidrāk aprakstīt algoritma ideju. Programmatūras kodā nedrīkst būt komentāri, kuri apraksta acīmredzamas lietas, piemēram, vērtību 5 nododam mainīgam *loopCounter.*

Programmatūras koda dokumentēšanas iespējas *Visual* *Studio* – atbalstīt programmatūras koda dokumentēšanu, izmantojot iegultos XML tagus (*embeded XML tags*), kas tiek izvietoti, pirms metodes apraksta vai pēc citiem elementiem. API dokumentācija varbūt ģenerēta, izmantojot rīkus, kuriem ir piekļuve informācijai XML komentāros (piemēram, Sandcastle, http://sandcastle.codeplex.com). Lai ieslēgtu dokumentēšanas iespēju *Visual* *Studio*, jāizvēlas *Projects* | <Projekta> *properties* | *Build* | „XML documentation file”. Ievadot „///” *Visual* *Studio*, programmatūras kods automātiski tiks papildināts ar nepieciešamiem tagiem.

Dokumentācijas valoda ir latviešu valoda. *Visual* *Studio* automātiski ģenerētie komentāri jāatstāj nemainītā veidā, kur tas ir nepieciešams.

### Komponenšu piegāde

Komponenšu piegādei jābūt veidotai, izmantojot *Microsoft Windows Installer* (turpmāk tekstā MSI) paketes. Papildus ir jāpiegādā tehniskais apraksts un uzstādīšanas instrukcija, ja tas ir prasīts. Nepieciešamības gadījuma piegāde varētu būt nodrošināta jauninājumu (paču) veida.

## Konfigurācijas vadība

Šajā nodaļā aprakstītas risinājuma komponenšu informācijas konfigurācijas vadības metodes.

### Konfigurācijas krātuve

Risinājuma komponenšu konfigurācija ir bāzēta uz failiem failu sistēmā, t.i., visai konfigurācijas informācijai jābūt pierakstītai XML failos. Darbvirsmas (*desktop*) aplikācijām tas ir app.config, Web aplikācijām tas ir web.config. Ja nepieciešams, var konkrēto konfigurāciju saglabāt cita veida failā XML.

### Konfigurācijas informācijas raksturs

Konfigurācijas informācijai ir jābūt statiskai, t.i, lai tiktu pielietoti jaunie uzstādījumi, nepieciešams iedarbināt komponenti. Tas nodrošinās labāku izpildi, jo trasēšanas informācija var būt kešēta, un arī vienkāršumu, jo dažas konfigurācijas izmaiņas varētu būt kritiskas tās komponentes iekšējam stāvoklim.

Konfigurācijas informācijas statiskais raksturs ietekmē komponenšu pieejamību konfigurācijas informācijas izmaiņu veikšanas laikā. Augstas pieejamības komponentēm jābūt vairākos eksemplāros un konfigurāciju maina vienam eksemplāram pēc otra, pārliecinoties, ka strādā vismaz viens aktīvais eksemplārs, kamēr tiek mainīta citu eksemplāru konfigurācija.

Konfigurācijas informācijai nav jābūt aizsargātai kādā speciālā veidā, ja tajā nav sensitīvo datu.

Ja servera vai klienta pusē konfigurācijā glabājas savienojuma rindu lietotāja vārds/parole vai cita autorizācijas informācija nepieciešams izmantot konfigurācijas datnes atbilstošās sekcijas šifrēšanas iespējas:

* Izmantojot DPAPI (Windows Data Protection application programming interface, http://msdn.microsoft.com/en-us/library/ff647398.aspx);
* Izmantojot RSA (http://msdn.microsoft.com/en-us/library/ff650304.aspx).

Citas paroles vai autorizācijas informācijas šifrēšanai jānodrošina saskaņā ar: http://support.microsoft.com/kb/329290.

## Izpildes trasēšana

Dažām risinājuma komponentēm ir jāiztur ļoti liela slodze un liels skaits vienlaicīgu lietotāju, tāpēc ir svarīgi, lai papildus pastāvīgai trasēšanai (skat. dokumentā [4] aprakstīto koplietojamas bibliotēkas), kuru izmanto risinājuma atkļūdošanai, tiktu izmantota arī izpildes trasēšana.

Izpildes trasēšana risinājumā jāizpilda, izmantojot:

1. Specifiski izpildes skaitītāji (*performance counters*), kurus eksponē izmantojot .NET rīki, un
2. Standarta izpildes skaitītāji, kurus eksponē ar Windows servisiem un MS standarta programmatūru, piemēram, .NET Framework, ASP.NET, SQL Server, Certificate Services.

Izpildes skaitītājus nepieciešams izmantot servera puses komponentēs, bet specifiskos scenārijos tie var tikt izmantoti arī klienta pusē.

### Izpildes skaitītāja izvietojums

Ņemot vērā, ka visām veidojama produkta komponentēm ir biznesa komponentes daļa, kas atbildīga par komponentes pamata funkcionalitāti, specifiski izpildes skaitītāji jāievieno biznesa komponentes programmatūras kodā. Tas nodrošinās, ka, vieni un tie paši skaitītāji tiks izmantoti atkārtoti gadījumos, kad viena komponente tiek darbināta, izmantojot vairākus kanālus.

### Izpildes skaitītāju nosaukšana

Specifiski izpildes skaitītāji, kurus eksponē risinājuma komponentē, jāpievieno izpildes skaitītāju grupām, kuras nosauktas atbilstoši šablonam "<namespace>.<klases nosaukums>:<metodes nosaukums>".

### Izpildes skaitītāja datu analizēšana

Trasēšanas informācija, kas eksponēta, izmantojot izpildes skaitītājus, jāanalizē:

1. izmantojot *Windows* *Performance* *Monitor* rīku reālajā laikā un/vai
2. reālā laikā vai vēlāk, izmantojot *Microsoft* *Operations* *Management* serveri, kurš savāc visu izpildes informāciju no visiem infrastruktūras serveriem.

### Tipiskie eksponētie izpildes skaitītāji

.NET WCF eksponē standarta izpildes skaitītāji. Visu WCF izpildes skaitītāju monitorings notiek ieslēdzot

<system.serviceModel>

<diagnostics performanceCounters="all">

…

Citādi var iedarbināt tikai servisā definētie izpildes skaitītāji (kas patērē mazāk resursu):

<system.serviceModel>

<diagnostics performanceCounters="ServiceOnly">

…

Detalizēti sk. sekojošā resursā: <http://msdn.microsoft.com/en-us/library/ms735098.aspx>

## Datu bāzes piekļuve

Nodaļā aprakstītas risinājuma datu bāzes informācijas piekļuves programmatūras koda realizācijas metodes.

### Uzticamais lietotājs

Uzticama datu bāzes lietotāja metode tiek izmantota, lai piekļūtu datu bāzes informācijai, t.i., neskatoties uz gala lietotāja izpildāmo operāciju, lietojuma servera programmatūras kods, izpildot operāciju, tiek autentificēts datu bāzē kā tā paša servisa lietotājs.

Lai apliecinātu servisu datu bāzē, tiek izmantots *Windows Integrated Security*. Lai serviss spētu izpildīt darbības, par kurām tas ir atbildīgs, servisa kontam ir jābūt piešķirtām datubāzes līmeņa tiesībām izpildīt („Execute”) noteiktu glabājamo procedūru tvērumu (stored procedures scope).

### Noklusētais datu bāzes izolācijas līmenis

Noklusētais datu bāzes izolācijas līmenis ir *Read Committed* izolācija. Tas nodrošina balansētu caurlaidspēju un izpildi tipiskai OLTP sistēmai.

Gadījumā, ja sistēmā paredzēti ilgstošie lasīšanas vaicājumi paralēli ar liela daudzuma izmaiņas vaicājumiem, vai sistēma prasa augstāka līmeņa izolācijas veidus transakcijām, nepieciešams izskatīt iespēju ieslēgt datubāzes opcijas „READ\_COMMITTED\_SNAPSHOT” un/vai „ALLOW\_SNAPSHOT\_ISOLATION”.

Rakstīšanas operācijas saskaņa ir realizēta ar manuāli specificētu izolācijas līmeni, definējot transakcijas izolācijas līmeni lietojuma programmatūras kodā vai glabājamā procedūrā.

### Glabājamās procedūras

Glabājamās procedūras jāizmanto piekļuvei visām datu bāzēm, t.sk., arī lasīšanas operācijām. Specifiskos scenārijos, kur tiešas SQL SELECT izteiksmes varētu būt efektīvākas, atļauts izmantot SQL izteiksmes ar arhitektūras komandas apstiprinājumu.

### Datu bāzes resursi

*Acquiring late* un *releasing soon* princips jāizmanto programmatūras kodā, lai piekļūtu datu bāzei. Tas nozīmē, ka saskarei ar datu bāzi ir jābūt atvērtai tikai pirms glabājamās procedūras izpildes un jābūt atbrīvotai, kad izpilde ir pabeigta. Ja tiek lietots SQL *Connection Pool,* tadsaskare ar datu bāzi tiek kontrolēta automātiski.

### Transakcijas

Ja vairākām datu bāzes operācijām, kuras tiek izsauktas no lietojuma servera komponentēm, jābūt vienā datu bāzes transakcijā, tad datu bāzes operācijas ir jāapvieno, izmantojot *System.Transactions.TransactionScope* objektu. Piemēram, bāziskas datu izgūšanas operācijas, kas izpilda tikai vienu glabājamo procedūru, nav apvienotas, izmantojot *TransactionScope*. Šīm vienkāršajām lasīšanas operācijām nav jādarbojas ar SQL transakciju vai transakcijas kopas objektiem.

Kad sarežģītākas operācijas tiek izpildītas, izmantojot vienkāršas lasīšanas operācijas (vai vairākas lasīšanas operācijas), tad vienkāršo operāciju izsaukšana jāizpilda *TransactionScope* definētā izsaucējā. Tas ļauj atkārtoti izmantot vienkāršās operācijas sarežģītākos scenārijos un arī mainīt sēriju veidošanas metodi, lai nodrošinātu nepieciešamo resursu aizsardzību.

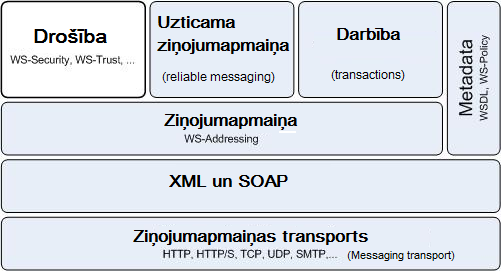
## Tīkla servisu izstrāde

### Kopējas prasības

Tīkla servisu izstrādē jāizmanto SOAP (vairāk [8] saistītajā dokumentā) 1.2.versija (Vienkāršais objektu piekļuves protokols) ir vieglpiekļuves protokols, kas paredzēts, lai apmainītos ar strukturētu informāciju decentralizētā.

### Drošības profils (security profile)

Izmantojot XML, SOAP un WSDL paplašināmos modeļus, tos savstarpēji kombinējot, var tikt izstrādātas arī servisu papildus izmantošanas iespējas, nodrošinot plašu tīmekļa servisu arhitektūru, kas ilustrēta 1.attēlā.



.attēls. Tīmekļa servisu arhitektūra

Detalizēti par drošo servisu izstrādi un izmantošanu skat. [9].

### Kļūdu un izņēmuma situāciju apstrāde

Tīkla servisu komunikācijā VISS infrastruktūras ietvaros pavisam tiek izdalīti divi kļūdu apstrādes gadījumi:

* Biznesa kļūdas. Ar biznesa kļūdām šajā kontekstā tiek saprastas visas kļūdas, kuras tiek apstrādātas lietotnes ietvaros, un kurām tiek piešķirts atbilstošais identifikators.
* Sistēmas kļūdas. Visas pārējās kļūdas, kas ir saistītas ar sistēmas darbību un kas netika apstrādātas tīkla servisā darbības kodā.

Biznesa kļūdas apstrāde ietver to saglabāšanu sistēmas žurnālā un atbilstošās pārraides apvalka kļūdu sekcijas aizpildīšanu kā tas ir paredzēts servisa projektējumā. Parasti, kļūdu aprakstam izmanto *IVISResponse* shēmas *ErrorListStructure* struktūru (http://ivis.eps.gov.lv/XMLSchemas/100001/IVIS/v1-0/IVISResponse.xsd).

Ar sistēmas kļūdām, tiek saprasta visas speciāli neapstrādātas kļūdas tiek uzskaitītas par sistēmas kļūdām un parasti automātiski tiek transformētas uz SOAP *Fault.*

Svarīgi, lai tīkla servisa kods nesaturētu izņēmumu situācijas apstrādes bloku, kurš pārveidotu visas saņemtās kļūdas par biznesa kļūdām ar vienotu identifikatoru un aprakstu: „Notika sistēmas kļūda!”.

# .NET programmēšanas standarti

## C# projektu izstrādes vadlīnijas

### Projektu nosaukumi

Izstrādes gaitā tiek izstrādāts produkts. Produkts var sastāvēt no daudziem projektiem. Projektiem rekomendēts izmantot šādus nosaukumus:

1. **Product** – produkta nosaukums vai produkta sagrupētā daļa;
2. **Product.<ProjectType>** - produkta projekta nosaukumi, kur <ProjectType>:
   1. WebService – Web servisa projekts;
   2. WebSite – Web lietojuma projekts;
   3. DataBase – Datubāzes projekts;
   4. BusinessLogic – Biznesa loģikas projekts;
   5. Utt.
3. **Product.<ProjectType>.UnitTests** – produkta projekta testa projekts;
4. **Product.<ProjectType>.Deployment** – projekta publikācijas projekts.
5. **Product.<ProjectType>.Setup** – produkta laidiena projekts;
6. **Product.<ProjectType>.Update** – produkta laidiena izmaiņas projekts.

### Produktu versiju piešķiršana

Produktam un tā projektiem tiek uzstādītas attiecīgas versijas.

#### Produkta versija

Produkta versijas formāts „M.m”, kur

1. M – pamatversijas numurs;
2. m – apakšversijas numurs.

Sākotnēji izstrādātam produktam versija ir 1.0.

#### Projekta versija

Projekta versijas formāts „M.m.b.r”, kur

1. M – pamatversijas numurs;
2. m – apakšversijas numurs;
3. b – kompilācijas numurs;
4. r – revīzijas numurs.

Pirmoreiz projekta versija ir 1.0.*build*.0, kur *build* ir dienu skaits no 01/01/2000.

#### Laidiena versija

Projekta versijas formāts „M.m.b.r” vai „M.m.r”, kur

1. M – pamatversijas numurs;
2. m – apakšversijas numurs;
3. b – kompilācijas numurs;
4. r – revīzijas numurs.

Pirmoreiz izveidotam laidienam versija ir 1.0.0 vai 1.0.*build*.0 atkarībā kāds laidiens tiek izveidots (*build* ir dienu skaits no 01/01/2000).

#### Datubāzes versija

Datubāzes versijas formāts „M.m.rrrr”, kur

1. M – pamatversijas numurs;
2. m – apakšversijas numurs;
3. rrrr – revīzijas numurs no četriem cipariem.

Pirmoreiz izstrādātai datubāzei versija ir 1.0.0000.

#### Produktu versiju numuru izmaiņas nosacījumi

Komponentes revīzijas numuru maina, ja tiek veiktas izmaiņas. Izmaiņas var būt saistītas ar kļūdu novēršanu vai programmatūras koda pārstrādāšanu, kas nepasliktina tā funkcionalitāti.

Laidiena revīzijas numurs tiek mainīts pēc jaunā laidiena sagatavošanas. Parasti produkta laidiena versija tiek mainīta pēc izmaiņām produkta galvenajā komponentē.

Apakšversijas versijas numurs tiek mainīts, ja tiek veiktas izmaiņas, kas saistītas ar funkcionalitātes izmaiņām vai jaunas funkcionalitātes pievienošanu.

Pamatversijas numurs tiek mainīts, ja tiek mainīta produkta struktūra un izveides principi.

### Projekta informācijas datne

Projektam jāpievieno informācijas datne. Tās piemērs:

// ----------------------------------------------------------------------------

// <copyright file="AssemblyInfo.cs" company="ABC software">

// Copyright © ABC SOFTWARE. All rights reserved.

// The source code or its parts to use, reproduce, transfer, copy or

// keep in an electronic form only from written agreement ABC SOFTWARE.

// </copyright>

// ----------------------------------------------------------------------------

using System;

using System.Reflection;

using System.Runtime.CompilerServices;

using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following

// set of attributes. Change these attribute values to modify the information

// associated with an assembly.

[assembly: AssemblyTitle("Product.BusinessLogic")]

[assembly: AssemblyDescription("Product business logic.")]

[assembly: AssemblyCompany("ABC software")]

[assembly: AssemblyProduct("Product")]

[assembly: AssemblyCopyright("Copyright \x00a9 ABC software 2007-2008")]

[assembly: AssemblyTrademark("")]

[assembly: AssemblyCulture("")]

[assembly: AssemblyConfiguration("")]

// Setting ComVisible to false makes the types in this assembly not visible

// to COM components. If you need to access a type in this assembly from

// COM, set the ComVisible attribute to true on that type.

[assembly: ComVisible(false)]

[assembly: CLSCompliant(true)]

// The following GUID is for the ID of the typelib if this project is exposed to COM

[assembly: Guid("6c05645c-69c5-46d6-a030-f2029cbc5abc")]

// Version information for an assembly consists of the following four values:

//

// Major Version

// Minor Version

// Build Number

// Revision

[assembly: AssemblyVersion("1.0")]

[assembly: AssemblyFileVersion("1.0.0.2")]

Projekta informācijas datnes atribūtu saraksts:

1. **AssemblyVersion** – produkta versija;
2. **AssemblyFileVersion** – projekta versija;
3. **AssemblyCulture** – asamblejas (*assembly*, citiem vārdiem sākot bibliotēkas) kultūra;
4. **AssemblyTitle** – projekta nosaukums;
5. **AssemblyDescription** – projekta apraksts;
6. **AssemblyConfiguration** – asamblejas konfigurācija, kas definē papildus informāciju par laidienu;
7. **AssemblyCompany** – uzņēmuma nosaukums, kura izveidoja projektu;
8. **AssemblyProduct** – produkta nosaukums;
9. **AssemblyCopyright** – autortiesības;
10. **AssemblyTrademark** – preču zīme;
11. **AssemblyInformationVersion** – asamblejas informatīvā versija (parasti sakrīt ar AssemblyFileVersion);
12. **AssemblySateliteVersion** – parasti sakrīt ar produkta versiju; izmanto daudzu valodu resursu izmantošanas gadījumā.

### Projekta izmaiņu aprakstīšanas datne

Katram projektam jābūt pievienotai izmaiņu aprakstīšanas datnei History.txt. Datne tiek veidota teksta formātā. Vēstures datnes formāts:

Date: dd/mm/yyyy

Assembly: assemblyname

[File: filename]

BUGFIX | NEW | CHANGE : comment

Release Vx.x.x.x at dd/mm/yyyy

Tajā:

1. x.x.x.x – projekta relīzes versija;
2. *dd/mm/yyyy* – projekta izmaiņu vai relīzes izlaiduma datums;
3. *assemblyname* – asamblejas nosaukums (obligāts, ja versija tiek mainīta produkta grupai);
4. filename – datnes nosaukums, kurā tika veiktas izmaiņas (neobligāts);
5. BUGFIX – izlabotās kļūdas;
6. NEW – papildinājumi;
7. CHANGE – veiktās izmaiņas;
8. *comment* – darbību apraksts (komentārs).

Vēstures datnes piemērs:

Date: 5/02/2007

Assembly: Product.BusinessLogic.dll

NEW: Pievienota jauna funkcija

CHANGE: Izmainīts vecās funkcijas formāts.

NEW: XMLSchemaTransformation klasei pievienoja īpašība XSLTFile

Release 1.0.2344.0 no 06/02/2007

## Vienošanās par nosaukumu piešķiršanu (*naming convention*)

Dokumentā ir aprakstīti projektā pieņemtie nosaukumu piešķiršanas standarti, kuri attiecas uz programmām, kas rakstītas, izmantojot J#, C# un (*Visual Basic*, turpmāk tekstā VB) VB.NET (papildinformācijai skatīt Microsoft izstrādātāju tīkla (*Microsoft Developer Network*, turpmāk tekstā MSDN) palīgdokumentācijā). Pastāv Microsoft rekomendācijas publiskiem un privātiem izstrādātājiem, tomēr individuālās vienošanās par nosaukumu piešķiršanas kārtību kompānijas iekšienē var atšķirties. Paredzams, ka 1. tabulā ietverto vadlīniju saraksts drīz kļūs par nozares *de-facto* standartu un ir balstīts uz *StyleCop Visual Studio plug-in* nosacījumiem [5]. Šos nosaukumu piešķiršanas standartus nepieciešams pielietot visiem .NET izstrādājumiem, t.sk., ASP.NET Web lietojumos un .NET *Windows Forms* lietojumos. Visi nosaukumi jāveido angļu valodā.

1.tabula

C# un VB.NET nosaukumu vadlīniju saraksts

| Tips | Standarts / Noteikums | Piemērs |
| --- | --- | --- |
| Vārdtelpas (*namespace*) | Pascal reģistrā bez pasvītrojuma. Projektiem tiek izmantots izstrādātāja nosaukums, piemēram, Abc, ZzDats, Rix utt. Visiem nosaukumiem jābūt pascal reģistrā (izmantojam Xml, nevis XML). | Abc.NotificationAndAudit  ZzDats.Framework  Rix.PFAS.AUTH |
| Asamblejas | Ja asambleja satur tikai vienu vārdtelpu, tad asambleja ir jānosauc tāpat kā vārdtelpa. | Rix.NotificationAndAudit.dll |
| Klases un struktūras | *Pascal* reģistrā bez pasvītrojuma. Nosaukuma sākumā nedrīkst likt "C" vai "cls". Klases nosaukums var sākties ar "I" tikai tad, ja nākamais burts ir mazais, pretējā gadījumā tas izskatīsies kā saskarne.  Klases nosaukumam nav jābūt līdzīgam vārdtelpai, kurā tā uzturas. Visiem saīsinājumiem, kuri sastāv no trim vai vairāk burtiem, ir jābūt *Pascal* reģistrā, nevis ar lielajiem burtiem. Jācenšas izvairīties no saīsinājumu lietošanas un vienmēr jācenšas izmantot lietvārdus. | Widget InstanceManager XmlDocument MainForm DocumentForm HeaderControl CustomerListDataSet (typed dataset) |
| Kolekciju klases | Jāseko klases nosaukumu piešķiršanas noteikumiem un nosaukuma beigās jāpievieno *Collection.* | WidgetCollection |
| Klases *Delegate* | Jāseko klases nosaukumu piešķiršanas noteikumiem un nosaukuma beigās jāpievieno *Delegate*. | WidgetCallbackDelegate |
| *Exception Classes* | Jāseko klases nosaukumu piešķiršanas noteikumiem un nosaukuma beigās jāpievieno *Exception*. | InvalidTransactionException |
| *Attribute Classes* | Jāseko klases nosaukumu piešķiršanas noteikumiem un nosaukuma beigās jāpievieno *Attribute*. | WebServiceAttribute |
| *Interfaces* | Jāseko klases nosaukumu piešķiršanas noteikumiem, pirms nosaukuma jāpievieno "I", pēc "I" nākamais burts ir lielais. | IWidget |
| Uzskaitījums (*Enumerations*) | Jāseko klases nosaukumu piešķiršanas noteikumiem. Uzskaitījuma beigās nepievieno *Enum*. Ja uzskaitījums reprezentē bitu karodziņu grupu, tad nosaukums jāpabeidz daudzskaitlī. | SearchOptions (bitwise flags)  AcceptRejectRule (normal enum) |
| Funkcijas un apakšfunkcijas  (*Functions and Subs*) | *Pascal* reģistrs bez pasvītrojuma, izņemot notikumu apdarinātājus (*handlers*).  Jācenšas izvairīties no saīsinājumu lietošanas. Funkcijām un apakšfunkcijām ir jāatšķiras ne tikai ar burtu reģistru, lai tās varētu izmantot reģistru nejutīgās valodas, piemēram, *VisualBasic* .NET . | VB: Public Sub DoSomething(...)  C#: public void DoSomething(...) |
| Īpašības un Publiskie \* locekļu mainīgie  (*Properties and Public \* Member Variables*) | *Pascal* reģistrs bez pasvītrojuma. Jācenšas izvairīties no saīsinājumu lietošanas. Locekļiem ir jāatšķiras ne tikai ar burtu reģistru, lai tos varētu izmantot reģistru nejutīgās valodas, piemēram, *VisualBasi*c .NET . | VB: Public Property RecordId As Integer  C#: public int RecordId |
| Parametri  (*Parameters*) | *Camel* reģistrs. Jācenšas izvairīties no saīsinājumu lietošanas. Parametriem ir jāatšķiras ne tikai ar burtu reģistru, tos varētu izmantot reģistru nejutīgās valodas, piemēram, *VisualBasic* .NET . | VB: ByRef recordId As Integer  C#: ref int recordId |
| Procedūras līmeņa mainīgie  (*Procedure-Level Variables*) | *Camel* reģistrs. | VB: Dim recordId As Integer  C#: int recordId; |
| Klases līmeņa privātie un aizsargātie mainīgie  (*Class-Level Private and Protected Variables*) | *Camel* reģistrs ar pirmo simbolu pasvītrojumu.  VB.NET vienmēr jānorāda "Aizsargāts" vai "Privāts", nevar izmantot "Dim".  "m\_" izmantošana nav ieteicama, tāpat arī tādu mainīgo nosaukumu izmantošana, kuri atšķiras no īpašības vienīgi ar reģistru, īpaši ar aizsargātiem mainīgiem, jo tādējādi tiks apgrūtināts darbs, ja programmēšanu realizēs VB.NET, jo locekļus būs nepieciešams nosaukt citādi kā *accessor/mutator* īpašības.  Microsoft rekomendē izmantot m\_ (un arī \_), kā arī savos programmatūras kodos izmanto abus variantus. Ir pieļaujams "m" likt arī sākumā. Protams, kamēr programmē galvenokārt ar C#, tiem varētu būt privāti locekļi, kuri atšķiras no īpašībām tikai ar reģistru.  Lai nemēģinātu un nenolīdzinātu katrai valodai speciālus reģistrus, tiek rekomendēts pievienot priedēkli – pasvītrojuma simbolu visās valodās, kuras to atbalsta. Ja nepieciešams, lai klase būtu pilnīgi pakļāvīga CLS, varētu atmest priedēkli visiem C# aizsargātiem locekļu mainīgiem. | VB: Private \_recordId As Integer  C#: private int \_recordId; |
| Formu kontroles (*Controls on Forms*) | Formu kontroļu nosaukumiem jāizmanto nosaukumi, kas skaidri apraksta kontroles izmantošanas būtību un ja nepieciešams, tās pamattipu.  Nedrīkst izmantot Ungārijas pierakstu vai modificētu Ungārijas pierakstu. | UserIDText, HeaderLabel, ChoiceList, SubmitButton |
| Konstantes  (*Constants*) | Jālieto *Pascal* reģistrs. Nevar izmantot AUGŠĒJĀ\_REĢISTRA\_BURTUS. | SomeClass.SomePublicConstant |

## Vienošanās par izvietošanas secību

*Using* direktīvām jābūt ievietotām vardtelpās. Programmatūras koda datnes elementiem jābūt sakārtotiem šādā secībā:

1. Extern Alias Directives;
2. Using Directives;
3. Vārdutelpas *(Namespaces*);
4. Delegāti (*Delegates*);
5. Uzskaitījumi (*Enums*);
6. Saskarnes (*Interfaces*);
7. Struktūras (*Struct*s);
8. Klases (*Classes*);

Klasēs, struktūrās un saskarnēs elementiem jābūt sakārtotiem šādā secībā:

1. Mainīgie (*Fields*);
2. Konstruktori (*Constructors*);
3. Finalizers (*Destructors*);
4. Delegāti (*Delegates*);
5. Notikumi (*Events*);
6. Uzskaitījumi (*Enums*);
7. Saskarnes (*Interfaces*);
8. Īpašības (*Properties*);
9. Indeksu īpašības (*Indexers*);
10. Funkcijas (*Methods*);
11. Struktūras (*Structs*);
12. Klases (*Classes*).

Elementiem (*class members*, klases locekļi) jābūt sakārtotiem pēc pieejas līmeņa šādā secībā:

1. Publiskie elementi (*public*);
2. Iekšējie elementi (*internal*);
3. Aizsargātie iekšējie elementi (*protected internal*);
4. Aizsargātie elementi (*protected*);
5. Privātie elementi (*private*).

Pirms konstantiem elementiem jābūt nekonstantiem elementiem. Pirms statiskiem elementiem jābūt parastiem elementiem.

Viena datne nedrīkst saturēt vairākas deklarācijas klasēm, interfeisiem utt.

## Labošanas iespēju nosacījumi

### C♯ avota faili (Sourcefiles)

Klases/failus jāveido īsus, nepārsniedzot 2000 rindiņu, kods jāsadala, jāveido skaidra struktūra. Katru klasi jāievieto atsevišķā failā un failu jānosauc līdzīgi klasei (izmanto paplašinājumu .cs).

### Katalogu izkārtojums (Directory Layout)

Katrai vārdtelpai izveido katalogu (priekš MyProject.TestSuite.TestTier lieto MyProject/TestSuite/TestTier kā ceļu)*,* atbilstoši katalogu nosaukumos nelieto kopā ar punktiem ("."). Tas atvieglo vārdtelpu kartēšanu atbilstoši kataloga izkārtojumam.

## C# izvietojuma nosacījumi

Instrukcijām jābūt ievietotām starp atvērtām un aizvērtām iekavām

PIEMĒRAM:

if (true)

{

return this.value;

}

SLIKTS STILS:

if (true)

return this.value;

Instrukcijai ar atvērtajām un aizvērtajām iekavām jābūt atsevišķā rindā

PIEMĒRAM:

public object Method()

{

lock (this)

{

return this.value;

}

}

...

public object Method()

{

return null;

}

SLIKTS STILS:

public object Method()

{

lock (this) { return this.value; }

}

...

public object Method() { return null; }

Elementiem jābūt atdalītiem ar tukšo rindu

PIEMĒRAM:

public void Method1()

{

}

public bool Property

{

get { return true; }

}

/// <summary>

/// Gets a value indicating whether the control is enabled.

/// </summary>

public bool Enabled

{

get { return this.enabled; }

}

SLIKTS STILS:

public void Method1()

{

}

public bool Property

{

get { return true; }

}

/// <summary>

/// Gets a value indicating whether the control is enabled.

/// </summary>

public bool Enabled

{

get { return this.enabled; }

}

Īpašībām un notikumiem jābūt vienā rindā vai atsevišķās rindās

PIEMĒRAM:

public bool Enabled

{

get { return this.enabled; }

set { this.enabled = value; }

}

public bool Enabled

{

get

{

return this.enabled;

}

set

{

this.enabled = value;

}

}

SLIKTS STILS:

public bool Enabled

{

get { return this.enabled; }

set

{

this.enabled = value;

}

}

Pirms vienas rindas komentāra jāievieto tukšā rinda

PIEMĒRAM:

public bool Enabled

{

get

{

Console.WriteLine("Getting the enabled flag.");

// Return the value of the 'enabled' field.

return this.enabled;

}

}

SLIKTS STILS:

public bool Enabled

{

get

{

Console.WriteLine("Getting the enabled flag.");

// Return the value of the 'enabled' field.

return this.enabled;

}

}

Nelietot liekas tukšās rindas:

* nelietot tukšo rindu pirms atvērtas iekavas;
* nelietot tukšo rindu pēc atvērtas iekavas;
* nelietot tukšo rindu pēc elementa dokumentācijas;
* nelietot vairākas vienas tukšās rindas starp instrukcijām;
* nelietot tukšo rindu pirms aizvērtas iekavas;
* nelietot tukšo rindu pēc vienas rindas komentāra.

PIEMĒRAM:

/// <summary>

/// Gets a value indicating whether the control is enabled.

/// </summary>

public bool Enabled

{

get

{

Console.WriteLine("Getting the enabled flag.");

// Return the value of the 'enabled' field.

return this.enabled;

}

}

SLIKTS STILS:

/// <summary>

/// Gets a value indicating whether the control is enabled.

/// </summary>

public bool Enabled

{

get

{

Console.WriteLine("Getting the enabled flag.");

// Return the value of the 'enabled' field.

return this.enabled;

}

}

## Komentāru pievienošanas nosacījumi

### Blokveida komentāri (Block Comments)

No blokveida komentāriem vajadzētu izvairīties. Aprakstiem rekomendēts lietot /// komentēšanas stilu, lai iegūtu C ♯ standarta aprakstus. Ja vēlas izmantot blokveida komentārus, jālieto šāds stils:

/\* Line 1

\* Line 2

\* Line 3

\*/

Tas lasītājam (cilvēkam) ļaus vizuāli bloku atšķirt no programmatūras koda. Pastāv alternatīva iespēja lietot novecojušo C stilu vienrindas komentāriem, tomēr tas nav ieteicams. Ja izvēlas lietot šo stilu, tad pēc komentāra jāseko rindas aplauzumam, pretējā gadījumā būs grūti ieraudzīt programmatūras kodu, pirms kura tajā pašā rindā atrodas komentāri:

/\* teksts teksts teksts \*/

Vispārīgā gadījumā blokveida komentārus izmanto, lai komentētu lielas programmatūras koda sekcijas. Šie komentāri jāizmanto pēc iespējas retāk, piemēram, saistībā ar *TechNote* „The fine Art of Commenting”.

### Vienrindas komentāri (Single Line Comments)

Tiek rekomendēts lietot // komentēšanas stilu, lai pievienotu komentāru programmatūras kodam (šim nolūkam *SharpDevelop* izmanto taustiņu kombināciju - Alt+/). To varētu izmantot arī, lai komentētu programmatūras koda sekcijas.

Vienrindas komentāros nepieciešams ievērot atkāpes līmeni, ja šie komentāri tiek lietoti koda dokumentēšanai.

Komentētu programmatūras kodu būtu vēlams komentēt pirmajā rindā, lai palielinātu komentēta programmatūras koda redzamību.

Pārlapošanas (*thumb*) likums nosaka, ka vispārīgā gadījumā komentāra garums nedrīkst daudz pārsniegt programmatūras koda garumu, jo tā ir pārāk komplicētu, potenciāli kļūdainu (*potentially buggy*) programmatūras kodu pazīme.

### Dokumentācijas komentēšana (Documentation Comments)

.Net ietvaros Microsoft ir ieviesis uz XML komentāriem bāzētu dokumentācijas ģenerēšanas sistēmu. Formāli šos komentārus var uzskatīt par vienrindas C♯ komentāriem, kas satur XML tagus. Vienrindas komentāru veidošanā seko šādam paraugam:

/// <summary>

/// This class...

/// </summary>

Daudzrindu XML komentārus veido šādi:

/// <exception cref=”BogusException”>

/// This exception gets thrown as soon as a

/// Bogus flag gets set.

/// </exception>

Ikvienai rindai jāsākas ar 3 slīpsvītrām, lai tās tiktu akceptētas kā XML komentāra rindas. Izšķir divu kategoriju tagus:

1. Dokumentācijas vienumi (*Documentation items*).
2. Formatēšana/Atsauču veidošana (*Formatting/Referencing*).

Pirmajā kategorijā ietilpst tādi tagi kā, piemēram, <*summary*>, <*param*> vai <*exception*>. Tie reprezentē vienumus, kuri pārstāv programmas saskarnes elementus, kurus nepieciešams dokumentēt, lai programma būtu derīga citiem programmētājiem. Parasti šiem tagiem ir tādi atribūti kā nosaukums vai *cref*, kā tas redzams iepriekš minētajā daudzrindu piemērā. Šos atribūtus pārbauda kompilators, līdz ar to, tiem ir jābūt ticamiem.

Otra kategorija pārvalda dokumentācijas izkārtojumu, lietojot tādus tagus kā: <*code*>, <*list*> vai <*para*>.

Pēc tam dokumentāciju ir iespējams ģenerēt, izmantojot izvēlnes #develop 'build' vienumu 'dokumentēšana'. Dokumentācija tiek ģenerēta HTMLHelp formātā.

Detalizētāks XML komentāru skaidrojums pieejams Microsoft .net framework programmatūras izstrādātāja rīkkopa (*Software Developer's Kit*, turpmāk tekstā SDK) dokumentācijā. Informācija par ieteicamo komentēšanas praksi un turpmākajiem jautājumiem saistībā ar komentēšanu atrodama TechNote 'The fine Art of Commenting' (http://www.icsharpcode.net/technotes/commenting20020413.pdf).

Jābūt komentētiem šādiem elementiem:

1. klases publiskiem locekļiem un tās parametriem un izejas datiem;
2. uzskaitījuma locekļiem;
3. *generic* tipa parametriem.

### Programmatūras koda datnes virsraksts (Copyright)

Katrā programmatūras koda datnē jābūt ievietotam virsrakstam. Virsrakstam jābūt pirmajai rindai datnē un komentētam xml:

//-----------------------------------------------------------------------

// <copyright file="Widget.cs" company="Sprocket Enterprises">

// Copyright (c) Sprocket Enterprises. All rights reserved.

// </copyright>

//-----------------------------------------------------------------------

Līnijas nav obligātas, un virsraksts var izskatīties arī šādi:

// <copyright file="Widget.cs" company="Sprocket Enterprises">

// Copyright (c) Sprocket Enterprises. All rights reserved.

// </copyright>

Ir iespēja pievienot papildu xml tagus:

//-----------------------------------------------------------------------

// <copyright file="Widget.cs" company="Sprocket Enterprises">

// Copyright (c) Sprocket Enterprises. All rights reserved.

// </copyright>

// <author>John Doe</author>

//-----------------------------------------------------------------------

# Nosaukumu piešķiršana MS SQL datu bāzes objektiem

Pašlaik pastāv ļoti daudz dažādu vienošanos par nosaukumu piešķiršanu datu bāzes objektiem. Neviena no tām nav kļūdaina, tā vairāk ir izstrādātāju personīga izvēle. Tomēr uzņēmumā viens darbinieks (vai darbinieku grupa) nosaka datu bāzes nosaukumu piešķiršanas noteikumus, standartizē tos, un citiem tie jāievēro. VRAA pieņemtā nosaukumu piešķiršanas vienošanās MS SQL datu bāzes objektiem ir aprakstīta 2.tabulā.

2.tabula

MS SQL nosaukumu vadlīniju saraksts

| Tips | Standarts / Noteikums/ Piemērs | Nosaukumu piemērs |
| --- | --- | --- |
| Tabulas | Tabulas reprezentē entītiju instances. Piemēram, informācija par visiem klientiem tiek glabāta vienā tabulā. Tātad 'klients' (*customer*) šajā gadījumā būs entītija un visas rindas klientu tabulā reprezentē entītijas 'klients' (*customer*) instances. Kāpēc nevar nosaukt tabulu pēc tās entītijas nosaukuma, '*Customer*'? Ņemot vērā, ka tabula satur daudzas klienta instances, tās nosaukumam ir jābūt daudzskaitlī – '*Customers*'.  Šī metode ir piemērotāka, nekā nosaukt tabulas, piemēram, *tblCustomers*. Skatoties uz vaicājumiem, ir skaidrs, ka atsevišķs nosaukums atteicas uz tabulu, jo tabulas nosaukuma priekšā vaicājumos vienmēr ir noteikums FROM (SELECT izteikuma daļa).  Ja datu bāzē ir dažādas loģiskas funkcijas un tabulas ir nepieciešams sagrupēt atbilstoši loģiskajām grupām, kurām tās pieder, tad var izmantot divu vai trīs burtu priedēkļus tabulas nosaukumā, kurš identificēs tabulas grupu.   Piemēram, datu bāzē ir tabulas, kuras glabā informāciju par Cilvēku resursu (Human resource) nodaļu un arī par Realizācijas (Sales) nodaļu. Visas tabulas, kuras attiecas uz Realizācijas nodaļu, var nosaukt, piemēram, tā:  SL\_NewLeads SL\_Territories SL\_TerritoriesManagers  Visas tabulas, kuras attiecas uz Cilvēku resursu nodaļu, var nosaukt šādi:  HR\_Candidates HR\_PremierInstitutes HR\_InterviewSchedules  Tāda nosaukumu piešķiršanas metode garantē, ka visas saistītās tabulas ir sagrupētas kopā un ka visas tabulas tiek kārtotas alfabētiskā secībā. Tomēr, ja datu bāzē ir tikai viena tabulu loģiskā grupa, tad nav vajadzības izmantot šo metodi.  Jāņem vērā, ka dažreiz tabulas vertikāle tiek sadalīta divās vai vairāk tabulās, taču šīs daļas efektīvi reprezentē vienu un to pašu entītiju. Tādā gadījumā, entītijas nosaukumam jāpievieno vārds, kas vislabāk identificē konkrēto sadaļu. | Customers  Orders  ErrorMessages  SL\_NewLeads SL\_Territories SL\_TerritoriesManagers  HR\_Candidates HR\_PremierInstitutes HR\_InterviewSchedules |
| Skati (*view)* | Jebkurai lietojumprogrammai, kura piekļūst skatam, skats ir tas pats, kas tabula. Tātad, nosaukumu veidošanas noteikumus, kas attiecas uz tabulām, var izmantot arī skatiem, tomēr ne visos gadījumos.  Izņēmumi:   1. Skati ne vienmēr reprezentē vienu realitāti. Skats var būt divu tabulu kombinācija, kas bāzēta uz JOIN nosacījuma, tādējādi, efektīvi reprezentējot uzreiz divas entītijas. Tādā gadījumā var kombinēt abu tabulu nosaukumus. Piemēram, ja skats apvieno tabulas '*Customers*' un ‘*Addresses*’, tad nosaukums ir 'CustomersAddresses'. Šo metodi var pielietot palīgtabulām, kuras tiek izmantotas, lai realizētu saiti daudzi-pret-daudziem starp tabulām. 2. Skati var apkopot esošo bāzes tabulu datus atskaites formā. Šīm atskaitēm iespējams izmantot, piemēram, šādus nosaukumus:   'Product\_Sales\_for\_1997’  'Summary\_of\_Sales\_by\_Year'  Atstarpes vietā objektu nosaukumos jāizmanto pasvītra ( \_ ). | CustomersAddresses  Product\_Sales\_for\_1997  Summary\_of\_Sales\_by\_Year |
| Glabājamās procedūras | Glabājamās procedūras ir orientētas uz darbībām. Tātad to nosaukumiem jāattēlo darbs, kuru procedūra izpilda. Procedūras nosaukumā jāizmanto darbības vārds.  Piemēram, procedūru, kura atgriež informāciju par klientu, ja tai ir nodots klienta identifikācijas numurs, vēlams nosaukt '*GetCustomerDetails*'.  Līdzīgi, procedūra, kura realizē informācijas par klientu ievadi, būtu jānosauc 'I*nsertCustomerInfo*'. Vēl daži procedūru nosaukumu piemēri: ‘WriteAuditRecord', 'ArchiveTransactions', 'AuthorizeUser'  Kā jau tika minēts tabulu gadījumā, iespējams izmantot priedēkli, lai glabājamās procedūras sagrupētu arī atkarībā no tā, kurai loģiskai grupai procedūra pieder. Piemēram, visām procedūrām, kuras ir saistītas ar pasūtījumu apstrādi (*'Order processing'*) varētu būt priedēklis ORD\_ (ar lieliem burtiem):  ORD\_InsertOrder ORD\_InsertOrderDetails ORD\_ValidateOrder  Izmantojot Microsoft SQL Server, procedūrai nekad nevajag pievienot priedēkli 'sp\_', ja vien procedūra netiek glabāta galvenajā datu bāzē. Ja procedūras nosaukumam būs pievienots priedēklis 'sp\_', SQL Server vienmēr meklēs šo procedūru galvenajā datu bāzē *(Master database*), un tikai pēc galvenās datu bāzes pārbaudes (ja tur nebūs atrasts) meklēs dotajā datu bāzē.  Nav ieteicams glabājamas procedūras nosaukumam pievienot priedēkli, piemēram, 'sproc\_', lai tikai parādītu, ka objekts ir glabājama procedūra. Jebkurš datu bāzes attīstītājs vai administrators var identificēt glabājamas procedūras, jo tām priekšā vienmēr ir atslēgvārds EXEC vai EXECUTE. | GetCustomerDetails  WriteAuditRecord  ArchiveTransactions  AuthorizeUser  ORD\_InsertOrder  ORD\_InsertOrderDetails ORD\_ValidateOrder |
| Lietotāja definētas funkcijas | Microsoft SQL Serverī lietotāja definētas funkcijas (turpmāk tekstā LDFas) ir gandrīz līdzīgas glabājamām procedūrām, izņemot faktu, ka LDFas var būt izmantotas SELECT izteikumos. Citādi glabājamās procedūras un LDFas ir līdzīgas. Tātad, nosaukumu piešķiršanas metodes, kuras tika aprakstītas glabājamu procedūru gadījumā, var pielietot arī LDFām. Iespējams izmantot priedēkli (ar maziem burtiem), lai sagrupētu LDFas. Piemēram, visas LDFas manipulācijām ar virknēm var nosaukt šādā veidā:  str\_MakeProperCase str\_ParseString | ProcessDate  str\_MakeProperCase str\_ParseString |
| Trigeri | Neskatoties uz to, ka trigeri ir speciāls glabājamās procedūras veids, nav jēgas izmantot tādu pašu nosaukumu piešķiršanas metodi, kā glabājamām procedūrām.  Piešķirot nosaukumu trigeriem, izšķir divus variantus:   1. Trigeri vienmēr ir atkarīgi no bāzes tabulas un nevar eksistēt paši par sevi. Tātad, labāk savienot bāzes tabulas nosaukumu ar trigera nosaukumu. 2. Trigeri parasti ir asociēti ar vienu vai vairākām šādām operācijām: *Insert, Update, Delete*. Tātad trigera nosaukumam jāatspoguļo to būtība.   Daži trigeru nosaukumu piemēri:  titles\_instrg titles\_updtrg titles\_deltrg *Microsoft SQL Server* atļauj vairāk nekā vienu trigeri katrai darbībai ar tabulu. Tātad, ja tas ir nepieciešams, varētu būt 2 trigeri ar INSERT, 3 ar UPDATE un 4 ar DELETE vienai tabulai. SQL Serverī nevar kontrolēt trigeru palaišanas kārtību, tomēr palaišanas kārtības kontrole ir iespējama, sakot ar SQL Serverī 2000 versiju. Tātad, ja tabulai 'Titles' ir 3 trigeri ar INSERT, nepieciešams izmantot šādu nosaukumu piešķiršanas metodi, lai tos atšķirtu:  titles\_ValidateData\_instrg titles\_MakeAuditEntries\_instrg Ja viens trigeris ir saistīts vairāk nekā ar vienu darbību (viens trigeris INSERT un UPDATE vai UPDATE un DELETE, vai jebkura cita kombinācija), trigera nosaukumā kopā jāizmanto vārdi 'ins','upd','del'.  Piemēram, ja tabulai Title viens trigeris ir INSERT un UPDATE, tad trigerim jāpiešķir šāds nosaukums  titles\_InsUpdtrg | titles\_instrg titles\_updtrg titles\_deltrg  titles\_ValidateData\_instrg titles\_MakeAuditEntries\_instrg  titles\_InsUpdtrg |
| Parametri | Procedūru, trigeru un lietotāja definētas funkcijas parametru identificēšanai kā piedēkli, jāizmanto ‘p’, bet parametru nosaukumu veidot, pielietojot *Camel* reģistru. | p\_title  p\_customerID |
| Indeksi | Tāpat kā trigeri, arī indeksi nevar eksistēt atdalīti un tie ir atkarīgi no bāzes tabulas. Tātad, ir vērts indeksa nosaukumā iekļaut tabulas un kolonnas nosaukumu, kurai ir izveidots indekss. Ir divu tipu indeksi: sagrupēti (*clustered*) un nesagrupēti (*nonclustered*). Šie divi indeksu tipi var būt unikāli vai neunikāli. Tātad nosaukumu piešķiršanas noteikumiem jāapraksta arī indeksu tipi.  Indeksu nosaukuma piešķiršanas noteikuma piemērs:  Tabulas nosaukums + Kolonnas nosaukums(i) + unikāls/neunikāls + sagrupēts/nesagrupēts (*clustered/nonclustered*)  Piemēram, unikālam, sagrupētam indeksam, kas bāzēts uz tabulas Titles kolonnu TitleID, nosaukums būtu šāds: Titles\_TitleID\_U\_Cidx Unikālam, nesagrupētam indeksam, kas bāzēts uz tabulas *Publishers* kolonnu *PubID*, nosaukums būtu šāds:  Publishers\_PubID\_U\_Nidx  Neunikālam, nesagrupētam indeksam, kas bāzēts uz tabulas *OrderDetai*ls kolonnu *OrderID*, nosaukums būtu šāds:  OrderDetails\_OrderID\_NU\_Nidx  Indeksi var būt arī kompleksi, t.i., indekss var būt bāzēts uz vairāk kā vienu kolonnu. Tādā gadījumā visi kolonnu nosaukumi jāapvieno kopā tāpat, kā tas tika aprakstīts iepriekš pie palīgtabulām un skatiem.  Kompleksam, unikālam, sagrupētam indeksam, kas bāzēts uz tabulas *OrderDetails* kolonnām *OrderID* un *OrderDetailID*, nosaukums būtu šāds:  OrderDetails\_OrderIDOrderDetailID\_U\_Cidx  Protams, tādi indeksu nosaukumi ir gari un nepatīkami. Bet, līdz ar to, nebūs nepieciešamības dot uz tiem norādes programmatūras kodā, ja vien nav vajadzības indeksu izveidot/nodzēst/pārveidot. | Titles\_TitleID\_U\_Cidx  Publishers\_PubID\_U\_Nidx  OrderDetails\_OrderID\_NU\_Nidx  OrderDetails\_  OrderIDOrderDetailID\_U\_Cidx |
| Kolonnas | Kolonnas ir entītijas atribūti, t.i., kolonnas apraksta entītijas īpašības. Tātad kolonnas nosaukumam jābūt dabiskam un jēgpilnam. Pieļaujami šādi nosaukumu varianti:   1. Piemēram, tabulai '*Customers*' varētu būt kolonnas: CustomerID CustomerFirstName CustomerAddress Kā redzams, būtu labi kolonnas nosaukumam pievienot priedēkli ar entītijas apzīmējumu, kuru tie reprezentē. 2. Otrs variants. Katrai tabulai no datu bāzes jāizvēlas 2-4 burtu kombinācija un jāpārliecinās, ka katra kombinācija ir unikāla. Piemēram, '*Cust*' tabulai '*Customers*', '*Ord*' tabulai '*Orders*', '*OrdD*' tabulai '*OrderDetails*', '*Ad*t' tabulai '*Audit*' utt. Tabulas kodu izmanto kā priedēkli kolonnas nosaukumā. Šī varianta priekšrocība ir tā, ka vaicājumos daudzām tabulām ar kompleksiem JOIN, nav jāuztraucas par divdomīgiem kolonnu nosaukumiem un nav jāizmanto tabulas pseidonīmi, lai pievienotu priedēkli kolonnas nosaukumam. Tas padara vaicājumus viegli lasāmus.  Piemēram, *CustomerID* kolonna tabulā '*Customer*':  Cust\_CustomerID; vai kolonna *CustomerID* un tabula '*Orders*':  Ord\_CustomerID. 3. Trešais variants (*Northwind*), kolonnu nosaukumus veido, nepievienojot tabulas nosaukumu vai priedēkli, izņemot primārās atslēgas lauku, kur jālieto entītijas nosaukums:  CustomerID FirstName Address | CustomerID CustomerFirstName CustomerAddress  Cust\_CustomerID  Ord\_CustomerID  CustomerID FirstName Address |
| Lietotāja definēti datu tipi | Lietotāja definēti datu tipi, ir tikai apvāks bāzes tipiem, kurus piedāvā datu bāzes pārvaldības sistēma. Tos izmanto, lai uzturētu datu tipu secību dažādu tabulu līdzīgajiem atribūtiem. Piemēram, ja kolonna *CustomerID* parādās desmit tabulās, visos gadījumos jāizmanto līdzīgi datu tipi. Līdz ar to, lietotāja definēti datu tipi būtu ļoti ērti. Nepieciešams izveidot lietotāja definētu datu tipu kolonnai *CustomerID*, un tas jāizmanto kā datu tips visos *CustomerID* sastapšanas gadījumos.  Vienkāršākā metode, lai nosauktu lietotāja definētus tipus, ir: Kolonnas nosaukums + '\_type'.  Piemēram, *CustomerID* tips būtu:  CustomerID\_type | CustomerID\_type |
| Primārās atslēgas | Primārā atslēga ir kolonna(s), kura var unikāli identificēt katru rindu tabulā. Tātad, primārās atslēgas nosaukumam var izmantot entītijas nosaukumu, pievienojot ‘*ID*’.  Primārai atslēgai kolonnas nosaukums no tabulas '*Customers*' būtu šāds: PK\_CustomerID,  ko nepieciešams saskaņot ar izvēlētās kolonnas nosaukuma shēmu. | PK\_CustomerID |
| Ārējās atslēgas | Ārējās atslēgas tiek izmantotas, lai reprezentētu attiecības starp tabulām, kuras ir saistītas.  Ārējā atslēga, kura norāda uz tabulas '*Customers*' kolonnu *CustomerID* no tabulas '*Orders*' kolonnas '*CustomerID*':  FK\_CustomerID  FK\_Cust\_CustomerID | FK\_CustomerID  FK\_Cust\_CustomerID |
| *Default* un *Check* ierobežojumi | Katram *default/check* ierobežojumam jāizmanto kolonnas nosaukums un jāpievieno priedēkļi '*def*' un '*chk*'.  Piemēram, *default* ierobežojums kolonnai *OrderDat*e būtu def\_OrderDate , bet *check* ierobežojums kolonnai *OrderDate* būtu chk\_OrderDate. | def\_OrderDate  chk\_OrderDate |
| Mainīgie | Mainīgiem, kuri glabā kolonnu saturu, varētu pielietot to pašu nosaukumu piešķiršanas metodi, kura tika aprakstīta saistībā ar kolonnu nosaukumiem. Jāizmanto *Camel* reģistrs. | @orderDate |
| Daži pamatnoteikumi | 1. Pēc iespējas neizmantot sarežģītus, garus nosaukumus tabulām, kā arī citiem datu bāzes objektiem. 2. Izmantot SajauktoReģistu, nevis pasvītrojumu, lai atdalītu vārdus nosaukumā. 3. Izmantot pasvītrojumu tikai starp piedēkli/priedēkli un objekta vārdu. 4. Neizmantot atstarpes datu bāzes objekta nosaukumā. Ja tomēr ir nepieciešams izmanto atstarpi, jāpārliecinās, ka nosaukums tiek ieslēgts kvadrātiekavās (Microsoft SQL Serverī). Piemēram, [Order Details]. 5. Neizmantot rezervētus vārdus datu bāzes objektu nosaukšanā, jo tas var radīt neparedzamas situācijas. 6. Ja datubāzē eksistē dažādas objektu grupas, kas nav vai ļoti maz sasaistīti viens ar otru, vēlams katrai no šim grupām izmantot atsevišķu shēmas nosaukumu „dbo” vietā, piemēram: AUD.AuditDetails LOG.LogDetails |  |

# MS SQL Serveru programmēšanas vadlīnijas

Šajā nodaļā ir aprakstītas SQL servera programmēšanas vadlīnijas un labās prakses paņēmieni. Šīm vadlīnijām un praktiskajiem paņēmieniem ir universālu likumu nozīme, bet iespējamas arī izņēmuma situācijas.

## Programmatūras koda lasāmība un formāts

1. Jāraksta komentārus saglabājamās procedūrās, trigeros un SQL paketēs ikreiz, kad ir kaut kādas neskaidrības. Tas palīdzēs citiem programmētājiem izprast programmatūras kodu, detalizētāk skat. 3.6.2.sadaļu.
2. Vienmēr konsekventi jālieto burtene (*case*) programmatūras kodos. Ja programmatūras kodā burtene nebūs konsekventi ievērota, tad, iespējams, uz burteni nejutīga servera programmatūras kods darbosies labi, bet uz burteni jutīga SQL servera tas cietīs neveiksmi. Piemēram, ja jāveido SQL servera tabula vai datu bāze, kura ir burteni jutīga vai bināri sakārtota secībā, visām norādēm uz tabulu jāizmanto burtene, kas ir specifēta priekšrakstā CREATE TABLE.
3. Nelietot kolonnu numurus (skaitļus) komandā (*clause*) ORDER BY.
4. Vecā stila savienojumu vietā izmantot vieglāk lasāmos ANSI standarta *Join* nobeigumus. Ar ANSI savienojumiem komanda WHERE tiek lietota tikai datu filtrēšanai. Ar vecāka stila savienojumiem komanda WHERE regulē (vada) gan savienojuma stāvokli (nosacījumu), gan datu filtrēšanu. Pirmajā no abiem zemāk esošajiem piemēriem parādīta vecā stila savienojuma sintakse, savukārt otrajā piemērā - jaunā ANSI savienojuma sintakse.

1. piemērs:

SELECT a.au\_id, t.title

FROM titles t, authors a, titleauthor ta

WHERE a.au\_id = ta.au\_id AND

ta.title\_id = t.title\_id AND

t.title LIKE '%Computer%'

2. piemērs:

SELECT a.au\_id, t.title

FROM authors a

INNER JOIN titleauthor ta ON a.au\_id = ta.au\_id

INNER JOIN titles t ON ta.title\_id = t.title\_id

WHERE t.title LIKE '%Computer%'

1. Lai SQL priekšraksti būtu vieglāk lasāmi, kad nepieciešams, katru komandu jāsāk jaunā rindā un jālieto atkāpe, piemēram:

SELECT title\_id, title

FROM titles

WHERE title LIKE '%Computer%' AND

title LIKE '%cook%'

1. Tāpat kā jebkurā citā programmēšanas valodā, nelietojiet GOTO vai arī lietojiet to ierobežoti. Pārmērīga GOTO lietošana var radīt grūti lasāmu un grūti saprotamu programmatūras kodu.

## Datu tipi

1. Ja atsevišķa kolonna atkārtojas daudzās tabulās, tad jāizmanto lietotāja definētie datu tipi, lai šīs kolonnas datu tipi būs atbilstoši visās tabulās.
2. NCHAR datu tipu kolonnai jāizmanto vienīgi tad, ja kolonna nav nenozīmīga (*non-nullable*). Ja NCHAR kolonna ir nenozīmīga, tā tiks apstrādāta kā fiksēta garuma SQL Servera 7.0+ kolonna. Līdz ar to, NCHAR(100), ja ir NULL, izmantos 100 baitus, radot telpas zudumu. Šādā gadījumā jāizmanto NVARCHAR(100). Protams, mainīga garuma kolonnām ir ļoti mazs apstrādes virstēriņš salīdzinājumā ar fiksēta garuma kolonnām. Izvēlei starp NCHAR un NVARCHAR jābūt pārdomātai, balstītai uz saglabājamo datu garumu.
3. Jāizmanto *Unicode* datu tipu, kā, piemēram: NCHAR un NVARCHAR, gadījumā, ja ir iespējams saglabāt kolonnā teksta datus vairākas valodas, kas izmanto ne-latīņu burtus. CHAR un VARCHAR datu tipi jālieto gadījumos, kad kolonna tiks saglabātas tikai angļu rakstzīmes vai teksts tikai vienā valoda. Šajā gadījuma jāpārliecinās, kā datubāzes vai atsevišķas kolonnas kodējums (collation) atbilst izmantotai valodai.
4. Nedrīkst lietot TEXT ,NTEXT un IMAGE datu tipus, jo tie būs izņemti no aprites nākamas MS SQL *Server* versijās. Tajā vieta jāizmanto tipi VARCHAR, NVARCHAR un VARBINARY. VARCHAR un VARBINARY tipa kolonnām jānorada fiksēto izmēru gadījumā, ja datu garums nepārsniedz 8000 simboli. NVARCHAR tipa kolonnām jānorada fiksēto izmēru gadījumā, ja datu garums nepārsniedz 4000 simboli.

## Saglabātās procedūras

1. Funkcijas, kuru rezultāts nemainās visu koda izpildes ciklā, nedrīkst izsaukt vairākkārt saglabāto procedūru, trigeru, funkciju pakešu iekšienē. Piemēram, virknes mainīgā garums var būt vajadzīgs vairākās procedūras vietās, tāpēc jāizsauc LEN (datu garums) funkciju tikai vienreiz, bet rezultāts jāsaglabā mainīgajā, tā vēlākai izmantošanai. Nedrīkst izsaukt LEN funkciju vairākkārtīgi.
2. Jāpārliecinās, ka saglabātās procedūras vienmēr atgriež vērtību, kas apzīmē to statusu. Saglabāto procedūru atgrieztās vērtības par sekmīgu izpildi un kļūdu nepieciešams standartizēt. RETURN priekšraksts ir paredzēts vienīgi izpildes statusa, nevis datu atgriešanai. Ja ir nepieciešams atgriezt datus, jāizmanto OUTPUT parametri.
3. Ja saglabātā procedūra vienmēr atgriež vienrindas rezultātu kopu, jāapsver iespēja rezultātu kopas atgriešanai izmantot OUTPUT parametrus SELECT priekšraksta vietā, jo ADO izvades parametrus vada ātrāk nekā SELECT priekšrakstu atgrieztās rezultātu kopas.
4. Kā jau tika minēts 4. nodalījumā, nedrīkst lietot prefiksu 'sp\_' saglabāto procedūru nosaukumos, jo prefikss 'sp\_' ir rezervēts sistēmas saglabātajām procedūrām, kas sadarbojas ar SQL serveri. Vienmēr, kad SQL serveris saskaras ar procedūras nosaukumu, kas sākas ar 'sp\_', tas šo procedūru vispirms mēģina saglabāt pamata datu bāzē, pēc tam - meklē paredzētos kvalifikatorus (datubāzi, īpašnieku), pēc tam mēģina izturēties kā īpašnieks. Neizmantojot prefiksu 'sp\_', var ietaupīt laiku saglabātās procedūras lokalizācijai (arī skat. 3.3.nodaļu).
5. Nepieļaut, ka klienta (*front-end*) aplikācijas pieprasa/manipulē ar datiem tieši, izmantojot SELECT vai INSERT/UPDATE/DELETE priekšrakstus. Tā vietā, jāizveido saglabātās procedūras un jāļauj savām aplikācijām piekļūt šīm procedūrām. Tas saglabās datu pieejamību ‘tīru’ un konsekventu visos aplikācijas moduļos, tajā pašā laikā centralizējot biznesa loģiku datu bāzē.

## Veiktspēja

1. Datu bāzes izstrādes laikā ir svarīgi domāt par aplikācijas veiktspēju. Reāli veiktspēju nav iespējams saskaņot vēlāk, kā vien datu bāzes izstrādes laikā, jo tas ietver tabulu un indeksu pārbūvi, pieprasījumu pārrakstīšanu u.c. Pieprasījumu analīzei jāizmanto grafisko ierobežojumu plānu Query Analyzer vai SHOWPLAN\_TEXT un SHOWPLAN\_ALL komandās. Jāpārliecinās, ka pieprasījumi izpilda indeksu meklēšanu ("Index seek"), nevis indeksu izvēršanu ("Index scan") vai tabulu izvēršanu ("Table scan”). Cik vien iespējams, vajag izvairīties no tabulu izvēršanas vai indeksu izvēršanas. Attiecīgajām kolonnām jāizvēlas pareizos indeksus.
2. Sākumā datiem jābūt normalizētiem vismaz līdz trešajai normālformai. Ja pēc tam kādus no datiem nepieciešams renormalizēt, lai uzlabotu veiktspēju, to var darīt, bet tikai pēc 3.normālformas. Nepieciešams dokumentēts loģisks pamatojums visām ar režīma pārkāpumu saistītām darbībām.
3. Pieprasījumos nedrīkst izmantot 'SELECT \*'. Nepieciešamās kolonnas nosaukums vienmēr jāraksta pēc SELECT priekšraksta, kā tas redzams piemērā:

SELECT CustomerID, CustomerFirstName, City

Šāda pieeja ļauj reducēt I/O disku un uzlabot veiktspēju.

1. Iespēju robežās jāizvairās veidot pagaidu tabulas, kamēr notiek datu apstrāde, jo pagaidu tabulu izveidei nepieciešams lielāks I/O disks. Jāapsver iespēja pagaidu tabulu vietā izmantot uzlabotu SQL, skatus, SQL servera 2000+ tabulu mainīgo vai atvasinātās tabulas.
2. Jāizvairās izmantot aizstājējzīmju rakstzīmes vārdu sākumā meklēšanas laikā, lietojot atslēgvārdu LIKE, kā rezultātā rodas pilns tabulas izvērsums, kas anulē indeksa mērķi, Pirmais, zemāk esošais, piemērs dod indeksa izvēršanu, savukārt otrais – indeksa meklēšanu.

1. piemērs:

SELECT LocationID

FROM Locations

WHERE Specialties LIKE '%pples'

2. piemērs:

SELECT LocationID

FROM Locations

WHERE Specialties LIKE 'A%s'

Funkciju pielietojums SELECT priekšrakstos neradīs priekšrocības indeksēšanai.

1. Turklāt meklēšanai jāizvairās lietot neatbilstošus operatorus (<> and NOT), jo tie rezultātā dod tabulu un indeksu izvērses.
2. Jālieto atvasinātās tabulas (derived tables) un CTE (Common Table Expressions), īslaicīgo tabulu un apakšvaicājumu „WHERE” apgalvojumā vietā, jo atvasinātās tabulas un CTE izpilda labāk. Kā piemēru var minēt šādu pieprasījumu, kuru izmanto, lai atrastu otru augstāko algu Personālsastāva tabulā:

SELECT MIN(Salary)

FROM Employees

WHERE EmpID IN (SELECT TOP 2 EmpID FROM Employees ORDER BY Salary Desc)

Izmantojot CTE, šo pieprasījumu iespējams pārrakstīt, kā redzams zemāk, un tas tiek izpildīts divreiz ātrāk, kā augstāk norādītais pieprasījums:

SELECT MIN(Salary)

FROM (SELECT TOP 2 Salary FROM Employees ORDER BY Salary Desc) AS A

Rezultāti var atšķirties dažādos scenārijos atkarībā no datu bāzes projektējuma, indeksiem, datu ietilpības utt. Jāizmēģina visus iespējamos pieprasījuma pieraksta veidus un jāizvēlas efektīvāko no tiem.

1. SQL pakešu sākumos jālieto SET NOCOUNT ON, vides izstrādei - saglabātās procedūras un trigerus, līdzīgi kā noklusētos ziņojumus '(1 row(s) affected)' pēc INSERT, UPDATE, DELETE un SELECT priekšrakstu izpildes. Tas uzlabo saglabāto procedūru izpildi, reducējot tīkla trafiku.
2. Jāizpilda visas integritātes pārbaudes un datu validāciju, lietojot ierobežojumus (svešas atslēgas un pārbaudes ierobežojumi) nevis trigerus, jo tie ir lēnāki. Trigerus jāizmanto vienīgi auditēšanai, klienta uzdevumiem un validācijai, ko nav iespējams izpildīt, lietojot ierobežojumus. Ierobežojumi ietaupa laiku, jo nav jāraksta programmatūras kodi šīm validācijām, viss darbs tiek atstāts *Relational Database Management System* (turpmāk tekstā – RDBMS) ziņā.

## Dažādi ieteikumi

1. Iespēju robežās jāmēģina izvairīties no servera puses kursoriem. Datu pieejamībai un manipulācijām (*procedural approach*) vietā vienmēr jāpielieto (*set-based approach*). Bieži no kursoriem var izvairīties, lietojot SELECT priekšrakstu. Ja kursors ir nenovēršams, jāizmanto WHILE cilpu, jo tā vienmēr ir ātrāka par kursoru. Lai kursoru aizstātu ar WHILE cilpu, nepieciešama kolonna (primārā atslēga vai unikālā atslēga) katras rindas unikālai identifikācijai. Jebkurā gadījumā ikvienai tabulai nepieciešama primārā atslēga vai unikālā atslēga.
2. Skatus parasti lieto, lai specifiskiem lietotājiem, balstoties un viņu interesēm, parādītu specifiskus datus. Skatus parasti lieto arī, lai ierobežotu pieeju datu bāzes tabulām, piešķirot atļauju vienīgi uz skatiem. Vēl viena nozīmīga skatu lietošanas iespēja – tie vienkāršo pieprasījumus. Iekļaujot bieži nepieciešanos, komplicētos savienojumus un aprēķinus skatā, tie nebūs jāatkārto katrā pieprasījumā, bet vienkārši jāizvēlas no skata.
3. Ja vien ir iespējams, jāizvēlas bināros vai attēlu failus (*Binary Large Objects or BLOBs*) nesaglabāt datu bāzē. Tā vietā jāsaglabā datu bāzē ceļš pie binārā vai attēla faila un jāizmanto to kā norādi uz aktuālo, kaut kur citur uz servera saglabāto, bināro failu. Ir labāk, ja šo lielo bināro failu izguve un darbības ar tiem notiek ar ārpus datu bāzes. Jāpatur prātā, ka datu bāze nav paredzēta failu uzglabāšanai. VISS aplikāciju ietvaros plānots bināros failus saglabāt Elektroniskajā dokumentu krātuvē.
4. Iespēju robežās jāizvairās no dinamiskajiem SQL priekšrakstiem. Dinamiskie SQL tiecas būt lēnāki par statiskajiem, tā rezultātā SQL serverim jāģenerē izpildes plāns katram izpildlaikam. IF un CASE priekšraksti ir parocīgi, lai izvairītos no dinamiskajiem SQL. Otra dinamisko SQL lietošanas nepilnība ir nepieciešamība, ļaut lietotājam tieši piekļūt visiem piekļuves objektiem., līdzīgi kā tabulām un skatiem. Parasti lietotājiem piešķir pieeju saglabātām procedūrām, kurās ir norāde uz tabulām, bet nevis piešķir piekļuvi pašām tabulām. Šajā gadījumā dinamiskie SQL nedarbosies.
5. Iespēju robežās jāizvairās no NULL vērtības izmantošanas. Piemēram, ja nav uzdota teksta vērtība, jāizmanto tukšo simbolu virkne. Šī pieeja samazina lieko konvertējumu daudzumu starp priekšgalā (front-end) attēloto informāciju un datubāzē glabājamo informāciju.
6. INSERT priekšrakstā vienmēr jālieto kolonnas saraksts. Tas palīdzēs izvairīties no problēmām, mainot tabulas struktūru (pievienojot vai izslēdzot kolonnu).
7. Piekļuve pie tabulām visās glabājamās procedūrās un trigeros vienmēr notiek vienā un tajā pašā veidā. Tas ļauj izvairīties no strupsaķerēm (*deadlock)*. Tālāk aprakstīti citi pasākumi, kas jāņem vērā, lai izvairītos no strupsaķerēm:
   1. Transakcijām jābūt iespējami īsām.
   2. Datu apmēram transakcijas laikā jābūt iespējami mazākam.
   3. Nedrīkst gaidīt, ka lietotājs ievadīs kādu informāciju transakcijas laikā.
   4. Nedrīkst izmantot augstākā līmeņa slēdzošas norādes (*locking hints*) vai ierobežojošus izdalīšanas līmeņus (*restrictive isolation levels*), ja vien tie nav kritiski nepieciešami.
   5. Priekšgala (*front-end*) lietotnes jāveido strupsaķeres inteliģentas (*deadlock-intelligent*), t.i., šīm lietotnēm jāspēj atkaliesniegt (*resubmit*) transakciju, ja gadījumā iepriekšējā transakcija ir cietusi kļūdu 1205.
   6. Visi SQL servera atgrieztie rezultāti nekavējoties ir jāapstrādā lietojumā, lai iespējami ātrāk atbrīvotu bloķētos datu bāzes ierakstus.
8. Uzdevumi, piemēram, virknes (*string*) manipulācijas, konkatenācija, rindu numurēšana, burtenes konversija u.c., jāpārnes uz lietojumu no datu bāzes servera pie nosacījuma, ka tie patērēs pietiekami daudz CPU laika. Turklāt vienkāršajām validācijām jānotiek datu ievades laikā lietojumā vai lietojuma biznesa loģikas klasē. Tas samazinās nevajadzīgu tīkla apriti (*roundtrips*).
9. MSSQL 2000 serveris tūlīt pēc datu manipulācijas priekšraksta (INSERT/UPDATE/DELETE) izpildes vienmēr pārbauda MSSQL 2000 globālo mainīgo, līdz ar to, kļūdas gadījumā iespējams atritināt (*rollback*) transakciju (kļūdas gadījumā @@ERROR vienmēr būs lielāks par 0). Tas ir svarīgi, jo pēc noklusējuma SQL serveris nevar atritināt visas iepriekšējās izmaiņas transakcijas ietvaros, ja specifiskais priekšraksts neizdodas. Šādu norisi iespējams mainīt, izpildot XACT\_ABORT ON. MSSQL 2005 serverim izmantojiet “*try – catch*” blokus.
10. @@ROWCOUNT mainīgajam parasti ir liela nozīme, lai noteiktu, cik rindas ir skāris iepriekšējais datu manipulācijas priekšraksts (arī izguve) un, balstoties uz to, var izvēlēties nodot (*commit*) vai atritināt (*rollback*) specifisku transakciju.
11. Datumos 2 ciparu gadskaitļa vietā vienmēr jāsaglabā 4 ciparu gadskaitli (īpaši, ja tiek lietotas CHAR vai INT datu tipu kolonnas), lai izvairītos no problēmām un neskaidrībām. DATETIME kolonnu gadījumā tā nebūs problēma, jo gadsimts tiks saglabāts pat, ja būs specificēts 2 ciparu gadskaitlis. Vienmēr labāk ir saglabāt 4 ciparu gadskaitli, arī DATETIME kolonnu gadījumā.
12. Nedrīkst aizmirst izpildīt unikālos ierobežojumus savām paralēlatslēgām (*alternate keys*).
13. Gadījumā, kad datubāzes manipulācijām izmantoti SQL skripta datnes, tiem jābūt saglabātiem izmantojot UTF-8 vai UNICODE kodējumu, lai neizrāsit problēmas sistēmām, kur sistēmas valoda nav Latviešu.
14. Ja paredzēts, kā izpildot SQL skripta datnes, tajos var mainīties objektu nosaukumi atkarībā no vidēm – tos jāimplementē ka skripta mainīgos pielietojot SQLCMD skripta izpildes režīmu. Šinī gadījumā skriptā sākumā jābūt pārbaudei, vai lietotājs ir ieslēdzis SQLCMD režīmu atbilstošam skriptam:

/\*

Detect SQLCMD mode and disable script execution if SQLCMD mode is not supported.

To re-enable the script after enabling SQLCMD mode, execute the following:

SET NOEXEC OFF;

\*/

:setvar \_\_IsSqlCmdEnabled "True"

GO

IF N'$(\_\_IsSqlCmdEnabled)' NOT LIKE N'True'

BEGIN

PRINT N'SQLCMD mode must be enabled to successfully execute this script.';

SET NOEXEC ON;

END

GO

# VISS vizuālā dizaina vadlīnijas

## Vispārīgas prasības

1. Lietojumi jāizvieto pa visu vidējo un labo sleju. Labā sleja izdalīta tikai sākotnējās sadaļās.
2. Izstrādājot konkrētu lietojumu nepieciešams norādīt, kas šajā lietojumā notiks nospiežot klaviatūras taustiņu Enter”.
3. Visiem teksta laukiem jābūt rediģējamiem (tie nedrīkst būt *hard - coded*).
4. Ja tiek ņemts sadaļu saturs no kāda ārēja resursa un netiek izmantota Sitecore vārdnīca, tad nepieciešams paredzēt resursa lokalizāciju un mehānismu kā notiks pārslēgšanās daudzvalodu režīmā.
5. Decimālatdalītāja uzstādījumi jāņem no ASP.NET lokalizācijas uzstādījumiem.
6. Datu izvēles risinājumiem, piemēram, kad nepieciešams izvēlēties kādu elementu no liela saraksta, ir jāveido modālā forma izvēlei, kas regulē šī saraksta atspoguļošanu un nodrošina ērtu informācijas meklēšanu datu sarakstā.

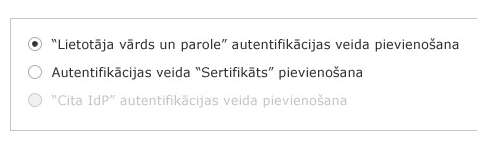
## Radiopoga

### Izmantošanas nolūks

Lietotājam jāizvēlas vienu no piedāvātajiem variantiem no grupā esošām radiopogām, kas parasti ir savstarpēji izslēdzošas: vienlaikus var izvēlēties tikai vienu no tām. Ja kāds elements ir jau izvēlēts, jaunā izvēle atcels iepriekšējo.

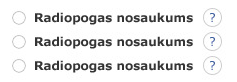
### Konceptuāls apraksts

Izvēlētais elements tiek apzīmēts ar melnu punktu (skat. 2.attēlu radiopogu „Lietotāja vārds un parole” autentifikācijas veida pievienošana). Neatļautie elementi ir pelēkā krāsā (skat. 2.attēlu radiopogu „Cita IdP” autentifikācijas veida pievienošana). Iespējams, neatļautos elementus neizvadīt ekrānā.



.attēls. Radiopogas vizuālais izskats

Ja nepieciešams, un ir vairākas sarežģīti nosauktas radiopogas, tad tām iespējams pievienot arī paskaidres, (skat. 6.8. nodaļu).

****

.attēls. Radiopogas ar paskaidri

### Lietošanas rekomendācijas

Lietojama tad, ja ir vismaz divas atšķirīgas izvēles, savstarpēji izslēdzošas, kas nav izsakāmas jautājuma formā ar atbildi „JĀ/NĒ”. Vienas izvēles gadījumā jālieto izvēles rūtiņa (skat. 6.4.nodaļu).

## Hipersaite

### Izmantošanas nolūks

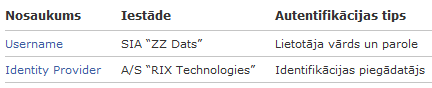
Izmanto, lai norādītu, ka notiks pāreja uz citu informācijas avotu/ dokumentu/ dokumenta daļu.

### Konceptuāls apraksts

Ar gaiši zilu krāsu izceļams vārds vai frāze, uz kura noklikšķinot peli, tiek parādīts kāds cits dokuments vai atvērta kāda cita interneta lapa.



.attēls. Hipersaite



.attēls. Hipersaites tabulā

### Lietošanas rekomendācijas

Jāizmanto visur, kur pēc lietojuma uzbūves nav iespējams izmantot parastās pogas. Piemēram, režģa *(grid)* satura izvadē. Izmantošana atkarīga no analīzes fāzes rezultātiem.

## Izvēles rūtiņa

### Izmantošanas nolūks

Izvēles rūtiņa (*Check Box*) ir kvadrātiņš blakus kādam parametra nosaukumam. Ja šajā kvadrātiņā ir ķeksītis, parametrs ir uzstādīts.

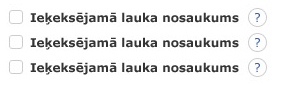
### Konceptuāls apraksts

Izvēles rūtiņai, lai norādītu, ka tā tiek ieķeksēta, vajadzētu lietot stilu, kas ir jaunākajās pārlūku versijās, lai ieķeksēta rūtiņa attēlotos līdzīgi (ķeksīša veids un iziešana ārpus rāmja) kā 6.attēlā.



.attēls. Izvēles rūtiņa

Pēc nepieciešamības iespējams lietot arī izvēles rūtiņu sarakstu, kā arī pievienot tām paskaidres, kas palīdz izprast atšķirību starp divām izvēles rūtiņām.



.attēls. Izvēles rūtiņa ar paskaidri

### Lietošanas rekomendācijas

Vienā dialoga logā var būt vairākas izvēles rūtiņas. Parasti tās ir savstarpēji neatkarīgas, t.i., vienas izvēle neietekmē pārējās, tāpēc, ja nepieciešams, lai būtu iespējams izvēlēties tikai vienu izvēli, tad jālieto radiopogas (skat. 6.1.nodaļu). Izmantošana atkarīga no analīzes fāzes rezultātiem.

## Pogas

### Izmantošanas nolūks

Pogas norāda uz izpildāmo darbību, kas aprakstīta uz tām (kas tiks veikta nospiežot šo pogu).

### Konceptuāls apraksts

Iespējams izmantot trīs veidu pogas, kuras aprakstītas un attēlotas turpmāk (skat. 8., 9. un 10.attēlus). Ja nepieciešams pievienot arī paskaidrojošu tekstu pogai, to iespējams darīt, ievērojot 11.attēlā redzamo izskatu.



.attēls. Svarīgākā poga, aktīva



.attēls. Mazāk svarīga poga, aktīva



.attēls. Neaktīva poga



.attēls. Aktīva svarīgā poga ar paskaidrojošo tekstu

### Lietošanas rekomendācijas

Turpmāk dots vispārējs pogu sadalījums:

1. Zila (Darbojas, pa visu formu, svarīgāka (piemēram, Saglabāt/Atcelt));
2. Pelēka (Mazāk svarīga poga, lieto visur, kur poga pieejama);
3. Balta ar pelēku (neaktīva poga, ko lietotājs nevar nospiest, kamēr nav izpildīti kādi nosacījumi).

Konkrētā pogu veida izvēle ir atkarīga no konkrētā lietojuma loģikas. Šīs rekomendācijas vajadzētu ņemt vērā projektējot formas, tomēr nepieciešamības gadījumā ir iespējams šo loģiku mainīt.

### Pogu izkārtojuma rekomendācijas

Informatīvs bloķējošs paziņojums par izmaiņām – pogas - Jā, Nē, Atcelt.

Vednī – pogas – Uz priekšu, Atpakaļ, Attīrīt.

Rediģēšana – pogas – Saglabāt, Atcelt.

Rediģēšana – pogas – Saglabāt, Aizvērt.

Meklēšana – pogas – Meklēt, Attīrīt, Atcelt,

## Režģis

Jāizmanto *DevExpress* pieejamā funkcionalitāte (skat. <http://www.devexpress.com/>, <http://demos.devexpress.com/ASPxGridViewDemos/> un http://demos.devexpress.com/aspxperienceDemos/TabControl/Features.aspx). Ja ir vajadzības izmantot citas komponentes, par to atsevišķi jāvienojas ar Pasūtītāju.

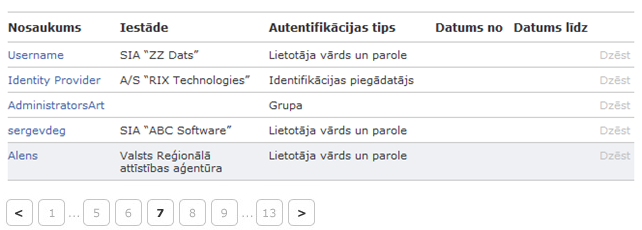
### Lapošana

Jāpielieto, ja lapaspušu skaits lielāks par dažām. Pieejamā funkcionalitāte: aiziet uz sākumu, aiziet uz beigām, kā arī pārvietoties pa sarakstu (skat 12.attēlu).



.attēls. Lapošana

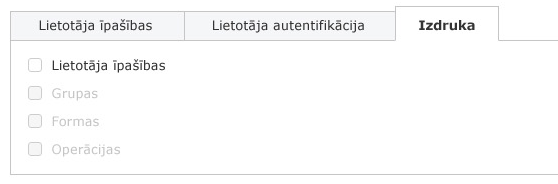
Darbības mehānisms – vienmēr tiek attēloti tikai pieci vidus elementi, kā arī pirmais un pēdējais elements. Visi elementi attiecīgi tiek nobīdīti uz vienu vai otru pusi, atkarībā no aktuālās vaļā atvērtās sadaļas. Mainot apskatāmo lapu (skat. 13.attēlu) tiek pārlādēta tikai konkrētā lapa, piemēram, pārejot, uz nākamo lapu – 6, tiks pārlādēts tikai režģis. Jāizmanto visiem sarakstiem, kuru ierakstu skaits ir liels un tos nav iespējams attēlot vienā lapā.



.attēls. Lapošana tabulā

### Cilne

Cilņu izmantošanas galvenais nolūks ir datu strukturizācija vai apvienošana.



.attēls. Cilnes

Pirmā cilne attēlojas un ielādējas vienmēr, pārējās var ielādēties pa vienai (to atverot). Cilņu rediģēšanu iespējams realizēt šādi (atkarībā no funkcionalitātes):

1. Katrā cilnē ievadāmo informāciju iespējams saglabāt ar tajā pašā cilnē izvietotās pogas „Saglabāt” palīdzību.
2. Ja cilnes izvietotas vedņa veidā – notiek loģiska pāreja no vienas cilnes uz citu, tad pēdējam vedņa solim jābūt visās cilnēs ievadītās informācijas attēlošanai un iespējai to kopīgi saglabāt ar vienas pogas nospiešanu.

## Paziņojumi

### Izmantošanas nolūks

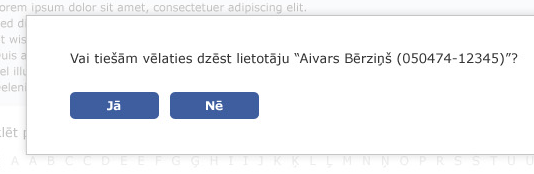
Lai viegli sakārtotu portāla paziņojumus, tiek ieviesta šāda klasifikācija:

1. Bloķējošs paziņojums – informē lietotāju par programmatūras izpildes laikā notikušajām darbībām, kā rezultātā tiek bloķēts lietotāja saskarne (piemēram: „Notiek pieprasījums”), lai ierobežotu ievades funkcionalitāti un netraucētu serverī palaistus procesus. Bloķējošie paziņojumi redzami 15. un 16.attēlos.
2. Informatīvs paziņojums – apziņo lietotāju par izmaiņām lietotāja darba vietā, piemēram, tiek iesūtīts paziņojums. Šajā gadījumā paziņojums netraucē veikt citas darbības portālā, t.i. pēc savas būtības šis paziņojums nav bloķējošs, skat. 17.attēlu. Šāda veida paziņojumi var tikt attēloti tikai uz brīdi un pēc tam pazust. Ir arī citādi informatīvi paziņojumi, ko iespējams attēlot portāla pirmajā un citās lapās, kas, piemēram, vari noformēt lietotāju par pašreiz nepieejamām sadaļām (skat. 18.attēlu).
3. Kļūdas paziņojums par kļūdu portāla sistēmā (bloks ar kļūdas paziņojumu) tiks realizēti gan ar pop-up funkcionalitāti, gan arī ar paziņojuma attēlošanu pašā formā, un tas tiek realizēts, kā atsevišķas formas kontrole. Konkrētā VISS lietojuma izstrādātājs pie tā prasību definēšanas varēs izvēlēties, kuru izmantot. Lauka validācijas kļūda un tās attēlojums pieejams 23.attēlā.

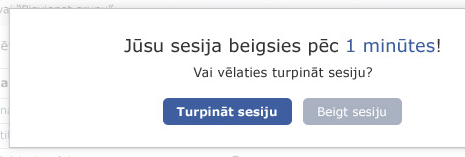
### Konceptuāls apraksts

Tehniskās nianses sistēmas lietošanai:

1. Bloķējošie paziņojumi tiks implementēti konkrētā funkcionālā bloka ietvaros, kura bloķēšana ir vajadzīga. Tas nozīmē, ka bloķējošā paziņojuma attēlošana ir atkarīga no bloķējamā funkcionālā bloka realizācijas.
2. Informatīvie paziņojumi tiks nodrošināti caur vienoto portāla API saskarni, kura nosakot lietotāja pārlūkprogrammas iespējas, pieņēma lēmumu par paziņojuma attēlošanas kārtību. Gadījumā, ja lietotājs izmanto pārlūkprogrammu, kas atbalsta HTML5 standartu, tad tiks veikta apziņošana ar šī standarta atbalstāmo iespēju palīdzību (piemēram, izsaucot metodi window.webkitNotifications.createNotification). Ja lietotāja izmantojamā pārlūkprogramma neatbalsta vajadzīgo iespēju, tad izstrādātājam katrā konkrētā lietojumā jāparedz statusa josla, kurā tiks izvadīts šis paziņojums. Informatīvie paziņojumi lietotājiem var tikt attēloti gan ar šī standarta atbalstāmo iespēju palīdzību (metode *window.webkitNotifications.createNotification*), ja to neizmanto, tad paredzēta atsevišķa statusa josla, kurā tiks izvadīts šis paziņojums. Tiks paredzēti abi gadījumi (arī, ja ir HTML5, var attēlot caur joslu), lai pēc vajadzības tos varētu izvēlēties (tie ir konfigurējami).
3. Katram paziņojumam var definēt trīs īpašības:
   1. Tekstu;
   2. Redzamību;
   3. Informatīvo paziņojumu veidu:
      1. Informatīvs paziņojums par veiksmīgi veiktu darbību (skat. 17.attēlu);
      2. Informatīvs paziņojums – brīdinājums (skat. 18.attēlu);
      3. Kļūdas paziņojums (skat. 19.attēlu);
      4. Informatīvs paziņojums (skat. 20.attēlu).



15.attēls. Informatīvs bloķējošs paziņojums, kam vajadzīgs apstiprinājums



.attēls. Informatīvs bloķējošs paziņojums ar apstiprinājumu (pogas dažādas nozīmības)

Katra lietojuma izstrādātājs var savam lietojumam definēt nepieciešamos un atbilstošos paziņojumus, kā arī veidu, kā tie jāattēlo. Katrs paziņojums ir kā objekts.

Ja rediģēšanas ekrānformā tiek atvērta poga (hipersaite) uz citu rediģēšanas ekrānformu un pirmajā ir nesaglabātas izmaiņas, tad atkarībā no konkrētā lietojuma izstrādātājiem iespējami trīs varianti:

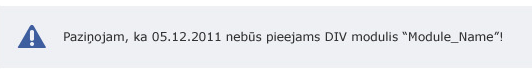
1. Ievadītās vērtības, pārejot no vienas ekrānformas uz otru, tiek saglabātas automātiski;
2. Ja lietojuma prasībās automātiska saglabāšana netika definēta, tad pārejot vai nospiežot pogu tiek attēlots brīdinošs informatīvs paziņojums – „Lūdzu saglabāt izmaiņas” un tikai tad, iespējams, pāriet uz jaunu ekrānformu. Šādā gadījumā formā, kurā iespējams labot un rediģēt datus jāparedz poga „Saglabāt”;
3. Iespējams arī nesaglabāt ievadīto informāciju, bet ka tas ir atkarīgs no biznesa procesiem, kuri tiek saskaņoti analīzes laikā.

### Lietošanas rekomendācijas

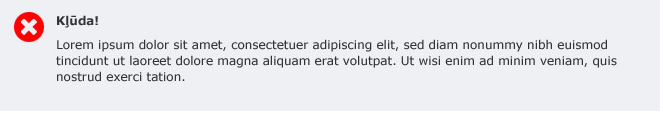
1. Bloķējošie paziņojumi var bloķēt visu mājaslapas saturu (skat. 15.attēlu).
2. Kļūdu gadījumā nepieciešams lietot sarkanu ikonu (skat. 19.attēlu), informatīvajiem paziņojumiem – zilu (skat. 18.attēlu). Paziņojumi, kas parādās un pazūd (piemēram, informējot par veiksmīgi izpildītu darbību) var būt krāsaini (piemēram, zaļi, kā 17.attēlā).



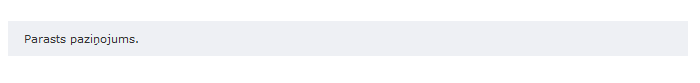
.attēls. Informatīvs paziņojums par veiksmīgi veiktu darbību



.attēls. Informatīvs paziņojums – informācija lietotājam



.attēls. Kļūdas paziņojums



.attēls. Skaidrojošs informatīvs paziņojums

Pie ieraksta saglabāšanas lietotājam jāattēlo informatīvs paziņojums par veiksmīgi izpildītu darbību (skat. 17.attēlu). Kad ieraksts tiek dzēsts lietotājam tiek attēlots informatīvs bloķējošs paziņojums (skat. 15.attēlu). Nospiežot dzēšanas apstiprinājuma pogu, lietotājam tiek attēlots atbilstošs informatīvs paziņojums – „ Dzēšana izdevusies veiksmīgi” (skat. 17.attēlu). Analoģisks paziņojums tiek attēlots arī datu atjaunošanas gadījumā. Izmantošana atkarīga no analīzes fāzes rezultātiem.

## Paskaidre

### Izmantošanas nolūks

Paskaidre paredzēta, lai varētu lietotājam sniegt paskaidrojumu par konkrēto pogu vai lauku, nevis vispārīgi par vairākiem. Ar tās palīdzību iespējams konkretizēt konkrētā formas elementa īpašības sīkāk. Paskaidre tiek aktivizēta uz tās uzklikšķinot, analoģiski aizvērt paskaidri iespējams ar klikšķi.

### Konceptuāls apraksts

Tātad, ja kādā laukā rodas validācijas kļūda, tad šī lauka saturs tiek attēlots ar sarkanu krāsu, paskaidres zīme no zilas (skat.21.attēlu) kļūst sarkana (skat. 24.attēlu) un pašā paskaidrē ar sarkanu tiek izvadīts kļūdas apraksts, bet lejā dota skaidrojoša informācija par lauku aizpildīšanu (skat. 22.un 25.attēlus salīdzinoši).



.attēls. Informatīva paskaidre (neizvērsta)



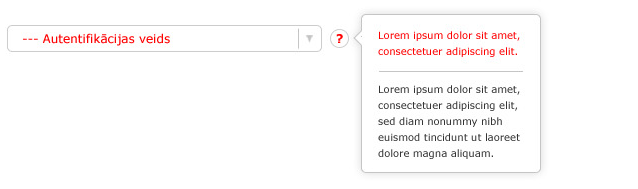
.attēls. Informatīva paskaidre (izvērsta)



.attēls. Validācijas kļūdas paziņojums



.attēls. Paskaidre par kļūdu (neizvērsta)



.attēls. Paskaidre par kļūdu (izvērsta)

### Lietošanas rekomendācijas

Ja formā ir tikai viena kļūda, tad, tas attēlojas apakšā zem pogas ar konkrētu tekstu, kur ir problēmas (skat.23.attēlu). Ja vairākas kļūdas, tad attēlojas vispārīgs kļūdas paziņojums, bet informācija par konkrētu kļūdu redzamam blakus kļūdainajam laukam, uzklikšķinot uz paskaidres (skat. 25.attēlu). Tiek izveidota paskaidre, ja iepriekš tur tā nav bijusi. Ja pirms tam jau bijusi informatīva paskaidre, tad kļūdaini aizpildot lauku, ko tā paskaidro, paskaidre tiek modificēta – augšdaļā ir kļūdas paziņojums, bet zem svītras tiek attēlots paskaidres iepriekšējais teksts (skat. 25.attēlu). Katra lietojuma izstrādātājs var definēt savam lietojumam piemērotāko paskaidres kļūdas gadījumu attēlojumu.

## Brīvas ievades lauciņš

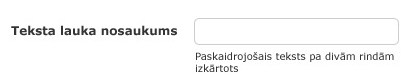
### Izmantošanas nolūks

Ja nepieciešams ievadīt kādu nestrukturētu tekstu, tiek izmantots brīvās ievades lauciņš, kas tiek aktivizēts, tajā ieklikšķinot.

### Konceptuāls apraksts



.attēls. Brīvās ievades lauciņš



.attēls. Brīvās ievades lauciņš ar paskaidrojošo tekstu



.attēls. Obligāti ievadāmie lauki



.attēls. Paroles ievade

### Lietošanas rekomendācijas

Lieto pēc vajadzības, ja nepieciešams iespējams arī izvietot paskaidrojošu tekstu zem lauciņa (skat. 27. attēlu). Obligāti aizpildāmie lauciņi tiek atzīmēti ar sarkanu zvaigznīti (skat. 28.attēlu). Ja jāievada parole (slēpts teksts), tad tas redzams kā 30.attēlā. Izmantošana atkarīga no analīzes fāzes rezultātiem.

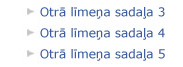
Ievadītajiem meklēšanas nosacījumiem jāsaglabājas, kā arī, vednī iepriekš izmantotajiem laukiem pārejot, atpakaļ ir jāsaglabājas.

## Izvēlne

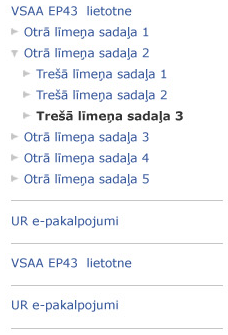
### Izmantošanas nolūks

Šī izvēlne attiecas uz konkrētu VISS lietojumu un strukturizē lietojuma iespējas/darbības. Paredzēta kreisās izvēlnes satura atspoguļošanai.

### Konceptuāls apraksts



.attēls. Kreisā izvēlne savērsta



.attēls. Kreisā izvēlne izvērsta (aktuālais ieraksts treknrakstā)

### Lietošanas rekomendācijas

Tiek izmantota *DevExpress* komponente un tās pieejamās iespējas. Lielu apjomu gadījumā (5 minūtēs komponente neielādējas) tiek izmantota ielādēšana pa daļām; no sākuma I līmenis, tad tālākie. Ielādēšanas pa daļām funkcionalitāte atkarīga no katra konkrētā lietojuma.

## Datums

Š iespēju var lietot datuma ievadei no kalendāra, lai tas nebūtu jādara manuāli. Lai sāktu datuma ievadi no kalendāra nepieciešams izvēlēties kalendāra atvēršanas simbolu, skat. 32.attēlā labajā pusē. Pēc atvēršanas tiek attēlots 33.attēlā redzamais kalendārs.



.attēls. Datuma informācijas izsaukšanas funkcionalitāte.



.attēls. Datuma ievade/izvēle.

Lietotājs var izvēlēties datumu, uz tā uzklikšķinot. Pēc noklusējuma izvēlēta šodiena. Iespējams no kreisās apakšējās daļas izsaukt saiti, kas izvēlas šodienu. Pēc datuma izvēles atliek uz tā uzklikšķināt un kalendārs tiek aizvērts un izvēlētais datums saglabāts, kā redzams 34.attēlā.



.attēls. Izvēlēts datums.

## Gaitas līnija un ielāde

### Izmantošanas nolūks

Gaitas līnija un ielādes simbols lietojams, gadījumos, kad nepieciešams sistēmas lietotājam parādīt, ka process notiek, kā arī demonstrēt tā izpildes progresu, lai lietotājs nesāktu vēl kādu citu procesu. Ielādes laikā (gan gaitas līnijas, gan ielādes simbola gadījumā) visiem kontroliem jābūt nobloķētiem, lai servera pusē nebūtu papildus darbību. Ilgi strādājošiem procesiem atbilde jāsniedz asinhroni.

### Konceptuāls apraksts



.attēls. Gaitas līnija



.attēls. Ielādējas, gaidiet

### Lietošanas rekomendācijas

Par vērtību aizpildīšanu ir atbildīga lietojuma iekšējā loģika. Izmantošana atkarīga no analīzes fāzes rezultātiem.

## Citi elementi

### Izbraucošā izvēle

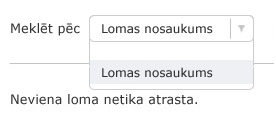
Iespējams pievienot vairākus lauciņus un definēt to vērtības. Izmantošana atkarīga no konkrētā lietojuma.



.attēls. Izbraucošā izvēle ar teksta lauku blakus

### Izkrītošais saraksts

Izkrītošais saraksts ir taisnstūris, kurā redzama pēdējā izvēlne. Lai sarakstu atvērtu, jānoklikšķina uz bultiņas lauka malā. Izmantojama kā alternatīva radiopogu sarakstam. Rekomendējama ja ir vairāk kā trīs izvēles iespējas.



.attēls. Izkrītošais saraksts

### Saraksts

Lauks, kurā ir saraksts ar vairākām vērtībām. Izmantošana atkarīga no konkrētā lietojuma.



.attēls. Saraksts

# VISS koplietojamo komponenšu izmantošana

## VISS Meklētāja koplietojamās datu shēmas apraksts

VISS Meklētāja kopējā datu shēma nodrošina datu agregāciju VISS portāla kopējas meklēšanas vajadzībām. Shēma ir izstrādāta ievērojot VISS metadatu un identifikācijas standartu (skat. [6]), *Clusterpoint* rekomendācijas un tehniskās vadlīnijas (skat. [7]). VISS koplietojamā datu shēma sastāv no šādām divām sadaļām:

1. Dokumenta metadati – apraksta dokumenta avotu, izcelsmi un mērķi.
2. Dokumenta saturs (valodas atkarīgs) – dokumenta tekstuāls saturs indeksācijai.

### Dokumenta metadati

3.tabulā ir aprakstīts saknes elements *document*.

3.tabula

Dokumenta metadati

| **Elements** | **Fac.[[1]](#footnote-1)** | **Izm.** | **Apraksts** | **Piemērs** |
| --- | --- | --- | --- | --- |
| Id |  | 1..1 | Pasaules mēroga unikāls identifikators, kurš viennozīmīgi identificē objektu *Clusterpoint* vidē. Ja tiek lietots IVIS URN tad tas varētu sakrist ar *objectId* elementu. | <id>URN:IVIS:100001:XSD-HL7V3.2011-lvext-LVCR\_MT000009UV01</id> |
| systemId | F | 1..1 | Informācijas sistēmas vai moduļa identifikators no VISS Informācijas sistēmas klasifikatora[[2]](#footnote-2) | <systemId>VISS.RC</systemId> |
| objectId |  | 1..1 | Sistēmas (pēc *systemId*) mēroga unikāls identifikators, kurš viennozīmīgi identificē objektu sistēmā. Ja tiek lietots IVIS URN tad tas varētu sakrist ar id elementu. | <objectId>12342334</objectId> |
| contributor |  | 1..1 | Iestāde[[3]](#footnote-3) objekta īpašnieks. Struktūra, kas sastāv no iestādes numura un iestādes nosaukuma, skat. pakārtotus elementus *authorityId* un *authorityName.* | <contributor>  <authorityId>165803</authorityId>  <authorityName>E-veselības aktivitāte</authorityName>  <authorityTypeId>MIN</authorityTypeId>  </contributor> |
| contributor. authorityId | F | 1..1 | VISS iestāžu klasifikatora reģistrētas iestādes identifikators *visscore:authorityIdentifier* |
| contributor. authorityName | F | 1..1 | VISS iestāžu klasifikatora reģistrētas iestādes nosaukums – atribūta *skos:preflabel* vērtība |
| contributor. authorityTypeId | F | 1..1 | Iestādes tipa identifikators no VISS iestāžu tipu klasifikatora |  |
| audience | F | 0..n | Aprakstīta dokumenta mērķauditorija, kuru veido šādi subjekti:  *Legal* (apzīmē juridiskās personas),  *Private* (apzīmē privātas personas) | <audience>Legal</audience>  <audience>Private</audience> |
| chargeable | F | 0..1 | Maksas pakalpojuma loģiska pazīme, kura varētu būt interpretēta kā saistīta ar doto objektu maksas veikšanas nepieciešamība | <chargeable>true</chargeable> |
| copyright |  | 0..n | Autortiesības ieraksts par dokumenta piederību | <copyright>(c) Nacionālais veselības dienests </copyright> |
| spatialCoverage |  | 0..n | Telpiskais resursa aptvērums |  |
|  |  |  |  | <spatialCoverage> |
| referenceName |  | 0..1 | Teritorijas tekstuāls apzīmējums | <referenceName>Rīga un Rīgas rajons</ referenceName> |
| spatialCoverage.atvk |  | 0..n | VISS ATVK klasifikatora[[4]](#footnote-4) reģistrēta vērtība | <atvk>  <code>010000</code>  <name>Rīga</name>  </atvk>  <atvk>  <code>130000</code>  <name>Jūrmala</name>  </atvk> |
| spatialCoverage.atvk.code | F | 1..1 | Administratīvas teritorijas ATVK kods (*vissadm:atvkCode*) |
| spatialCoverage.atvk.name | F | 1..1 | Administratīvas teritorijas nosaukums (*skos:preflabel*) |
| spatialCoverage.position |  | 0..n | Aprakstāma objekta ģeotelpiskas koordinātes | <position>  <latitude>56.925054</latitude>  <longitude>24.110779</longitude>  </position>  </spatialCoverage> |
| spatialCoverage.position.latitude |  | 1..1 | Aprakstāma objekta *latitude’s* koordināte |
| spatialCoverage.position.longitude |  | 1..1 | Aprakstāma objekta *longitude’s* koordināte |
| temporalCoverage |  | 0..1 | Temporālais resursa aptvērums. | <temporalCoverage> |
| temporalCoverage.created |  | 0..1 | Dokumenta izveidošanas laiks, atbilstoši formātam:  YYYY/MM/DD [HH:MM:SS [am/pm]] | <created>2012/08/29 00:00:00</created> |
| temporalCoverage.modified |  | 0..1 | Dokumenta modifikācijas datums, ja dokuments ir pievienots pirmo reizi, tad šis datums sakrīt ar *created* elementu. Laika formāts:  YYYY/MM/DD [HH:MM:SS [am/pm]] | <modified>2012/08/29 00:00:00</modified> |
| temporalCoverage.validFrom |  | 0..1 | Derīgs no. Ja elements nav noradīts, tad skaitās, ka dokuments ir derīgs no pievienošanas datuma. Laika formāts:  YYYY/MM/DD [HH:MM:SS [am/pm]] | <validFrom>2012/08/29 00:00:00</validFrom> |
| temporalCoverage.validTo |  | 0..1 | Derīgs līdz. Ja elements nav noradīts, skaitās, ka dokuments ir derīgs bez termiņa ierobežojumiem. Laika formāts:  YYYY/MM/DD [HH:MM:SS [am/pm]] | <validTo>2022/08/28 23:59:59</validTo>  </temporalCoverage> |

### Dokumenta saturs

4.tabulā ir aprakstīts elementa *document*.*langDependableContent* saturs. Katrai valodai ir domāta sava datu struktūra, kas identificējas ar ISO kodu.

4.tabula

Dokumenta saturs

| **Elements** | **Fac.[[5]](#footnote-5)** | **Izm.** | **Apraksts** | **Piemērs** |
| --- | --- | --- | --- | --- |
| lv |  | 1..n | Identificē aprakstošo valodu, elementa nosaukums ir valodas ISO kods. | <lv> |
| name |  | 1..1 | Dokumenta nosaukums | <name>Dokumenta nosaukums</name> |
| alternateName |  | 0..1 | Cits dokumenta nosaukums | <alternateName>Cits dokumenta nosaukums</alternateName> |
| description |  | 0..1 | Dokumenta apraksts. | <description>Apraksts</description> |
| url |  | 0..1 | Dokumenta atrašanas vieta, absolūta. | <url>http://server/document\_lv</url> |
| keyword | F | 1..n | Atslēgvārdu saraksts | <keyword>dokuments</keyword>  <keyword>pase</keyword> |
| category |  | 0..n | Kategorija, kurai pieder dokuments | <category> |
| category.code |  | 0..1 | Kategorijas kods (no klasifikatora, ja eksistē), kurai pieder dokuments. | <code>LIVE\_PLACE</code> |
| category.name |  | 0..1 | Kategorijas nosaukums, kurai pieder dokuments. | <name>Dzīvesvieta</name>  </category> |
| version |  | 0..n | Dokumenta versija | <version>1.0</version> |
| format |  | 0..n | Oriģināla dokumenta formāts[[6]](#footnote-6), piemēram: text/html, text | <format>text</format>  </lv> |
| categoryType |  | 0..n | Aprakstāma objekta informācijas kategoriju veids (piemēram, PPK pakalpojums, e-pakalpojums latvija.lv) | <categoryType><typeId>EPAK</typeId><redirectUrl>https://lvptest.vraa.gov.lv/Lvp/Epakalpojumi/EP38</redirectUrl></categoryType> |
| categoryType. typeId | F | 1..1 | Kategorijas veida identifikators no VISS kategorijas veidu klasifikatora |  |
| categoryType.redirectUrl |  | 0..1 | URL adrese, kas atbilst aprakstāmā kategorijas veida pārsūtīšanas adresei |  |

## Lietotāju profilu īpašību izgūšanas pakalpju apraksts

VISS un LVP infrastruktūrā lietotnēs (piemēram, VISS, LVP portālos), kurās var strādāt autentificētie lietotāji, katram lietotājam ir piesaistīts savs lietotāju profils. Katrā lietotāja profilā ir īpašības – pāri „atslēga – vērtība”. Piemēram, ja lietotājs *X* ir izmantojis lietotnes VISS, LVP un *Lietotne\_A*, tad viņa profils eksistē šajās lietotnēs – skat. 40.attēlu.



.attēls. Lietotāja profila īpašības dažādās lietotnēs

### Lietotāju profilu īpašības

Katrai profila īpašībai ir noteikts datu tips, kas nosaka, kādas vērtības var pieņemt attiecīgās īpašības vērtības. Īpašības datu tips var būt viens no šādiem:

* *String* (teksta virkne);
* *Int* (vesels skaitlis);
* *DateTime* (datums un laiks);
* *Bool* (Būla tips: iespējamās vērtības ir ‘true’ un ‘false’).

Profilu īpašību kopa ir atkarīga ne tikai no lietotnes, bet arī no profila veida. Iespējamie profilu veidi ir šādi:

* *Inhabitant* – iedzīvotāja kā fiziskās personas profils;
* *Employee* – iedzīvotāja kā konkrētās iestādes darbinieka profils;
* *Authority* – iestādes profils;
* *LegalEntity* – uzņēmuma profils;
* *UserDeclaredIdentity* – pašdeklarētās fiziskās personas profils.

Lietotnēs LVP un VISS definēto profila īpašību saraksts, atkarībā no profila veida, ir sniegts 5.tabulā.

5.tabula

Lietotņu LVP un VISS lietotāju profilu īpašības (atkarībā no profila veida)

| Lietotne | Profila īpašība | Īpašības esamība profilā atkarībā no profila veida | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| *Inhabitant* (PK) | *Employee* (PK+AU) | *Authority* (AU) | *LegalEntity* (UR) | *UserDeclared­Identity* |
| LVP | Identifier | + |  |  | + | + |
| FirstName | + |  |  |  | + |
| LastName | + |  |  |  | + |
| AuthorityId | + |  |  |  |  |
| AuthorityName | + |  |  |  |  |
| Country | + |  |  | + | + |
| Region | + |  |  | + | + |
| City | + |  |  | + | + |
| Village | + |  |  | + | + |
| Street | + |  |  | + | + |
| HouseNumber | + |  |  | + | + |
| FlatNumber | + |  |  | + | + |
| PostIndex | + |  |  | + | + |
| FullAddress | + |  |  | + | + |
| Phone | + |  |  | + | + |
| GetSmsOnPhone | + |  |  | + | + |
| Email | + |  |  | + | + |
| GetInfoOnEmail | + |  |  | + | + |
| BankAccountNumber | + |  |  | + | + |
| Territory | + |  |  |  | + |
| UseTerritory | + |  |  |  | + |
| ItemsPerPage | + |  |  |  | + |
| CompanyName |  |  |  | + |  |
| CompanyRegistration­Number |  |  |  | + |  |
| AuthenticationType | + |  |  |  |  |
| Position | + |  |  |  |  |
| RoleRestrictions | + |  |  |  |  |
| FavoritePPKServices | + |  |  |  | + |
| FavoriteEPAKServices | + |  |  |  | + |
| VISS | Identifier |  | + | + |  |  |
| FirstName |  | + |  |  |  |
| LastName |  | + |  |  |  |
| AuthorityId |  | + | + |  |  |
| AuthorityName |  | + | + |  |  |
| Country |  | + |  |  |  |
| Region |  | + |  |  |  |
| City |  | + |  |  |  |
| Village |  | + |  |  |  |
| Street |  | + |  |  |  |
| HouseNumber |  | + |  |  |  |
| FlatNumber |  | + |  |  |  |
| PostIndex |  | + |  |  |  |
| FullAddress |  | + |  |  |  |
| Phone |  | + |  |  |  |
| Email |  | + |  |  |  |
| GetInfoOnEmail |  | + |  |  |  |
| ItemsPerPage |  | + |  |  |  |
| Position |  | + |  |  |  |

VISS ietvarā eksistē divas drošās tīmekļa pakalpes: *UserProfileService* un *UserProfile­ByIdentifier­Service*, – kas ļauj izgūt lietotāju profilu īpašības (skat. 41.attēlu). Abas pakalpes kā ieeju saņem *XML* elementu ar attiecīgās operācijas pieprasījuma struktūru kopā ar *PFAS.STS* (PFAS drošības talonu pakalpes) izsniegto drošības talonu (*token*). Kā rezultātu abas pakalpes atgriež *XML* elementu ar attiecīgo atbildi vai arī *SOAP Fault* ziņojumu kļūdas gadījumā.



.attēls. Lietotāju profilu īpašību izgūšanas pakalpju saskarne

### Lietotāju identificēšana un drošība

Pakalpe *UserProfileService* identificē lietotāju, attiecībā uz kura profilu ir jāizpilda operācija, ņemot lietotāja identifikatoru no atbilstošā drošības talona apliecinājuma (*claim*). Tādā veidā, pakalpe *UserProfileService* var iegūt informāciju tikai par pašreiz autentificētā lietotāja profilu.

Pakalpe *UserProfile­ByIdentifier­Service*, atšķirībā no pakalpes *UserProfileService*, var iegūt informāciju par jebkura lietotāja profilu; tomēr šajā gadījumā drošības talonā ir jābūt atbilstošiem apliecinājumiem, kas atļauj izgūt (‘Read’) vai modificēt (‘Write’) informāciju par konkrētās lietotnes jebkura lietotāja profilu. Tikai apliecinājuma ar šādu nosaukumu:

**‘Full’ + ‘Read’ / ‘Write’ + ‘Of’ + (lietotnes nosaukums)**

– esamība PFAS autorizācijas apgabalā *UserProfileService* atļauj izgūt / modificēt informāciju par visu atbilstošās lietotnes lietotāju profilu informāciju; pretējā gadījumā, pieprasot operāciju bez attiecīgā apliecinājuma esamības, tiks atgriezta kļūda. Lietotāja, uz kura profila ir jāizpilda konkrētā operācija, identifikators tiek norādīts *XML* pieprasījumā elementā *UserIdentifier 🡪 NameIdentifier* (visām šīs pakalpes operācijām).

No drošības viedokļa ir ieteicama tieši pakalpes UserProfileService izmantošana: ja lietotnei ir jāpiekļūst pie citas lietotnes (piemēram, LVP vai VISS) lietotāja profila informācijas, tad, ja pastāv šāda iespēja, atbilstošais pieprasījums ir jāveic tā lietotāja vārdā, pie kura profila informācijas ir jāpiekļūst. Pakalpes UserProfile­ByIdentifier­Service izmantošana paredz šīs izmantošanas iepriekšējo saskaņošanu ar Pasūtītāju (kas ir atbildīgs par lietotāju profilu datu drošību), jo nekontrolēta piekļuve pie citas lietotnes patvaļīgu lietotāju profilu informācijas rada potenciālu ievainojamību lietotāju informācijas drošībā. Tāpēc pakalpe UserProfile­ByIdentifier­Service ir jāizmanto tikai tajā gadījumā, kad nepastāv iespējas veikt pieprasījumus lietotāja profila informācijas izgūšanai attiecīgā lietotāja vārdā.

Lietotāji (personas) – vai nu PFAS.STS drošības talona apliecinājumā (pakalpe *UserProfileService*), vai nu XML elementā *UserIdentifier 🡪 NameIdentifier* (pakalpe *UserProfile­ByIdentifier­Service*) – tiek identificēti pēc identifikatoriem, kas tiek formēti pēc šādas shēmas:

**{PK:[personas kods]}{-}{AU:[iestādes kods]}{-}{UR:[uzņēmuma kods]}**,

kur:

* *personas kods* – personas personas kods bez atdalītāja; piemēram, ‘10098610000’;
* *iestādes kods* – personas iestādes VISS identifikators; piemēram, ‘100000’;
* *uzņēmuma kods* – personas uzņēmuma kods no Uzņēmumu reģistra; piemēram, ‘40003627089’.

Personas identifikatorā PK, AU un UR atribūtu secība var būt patvaļīga. Atkarībā no identifikatora atribūtiem tiek atšķirts personas tips: iedzīvotājs, iestādes darbinieks, uzņēmuma paraksttiesīgā persona vai sistēma – skat. 6.tabulu.

6.tabula

Iespējamie personu identificēšanas tipi

|  |  |  |
| --- | --- | --- |
| Personas tips | Identifikatora atribūti | Identifikatora piemērs |
| Iedzīvotājs | PK | PK:10098610000 |
| Iestādes darbinieks | PK+AU | PK:10098610000-AU:100000 |
| Uzņēmuma paraksttiesīgā persona | PK+UR | PK:07017010000-UR:40003627089 |
| Sistēma (iestāde) | AU | AU:100000 |

Pakalpes *UserProfileService* un *UserProfile­ByIdentifier­Service* pieņem tikai PFAS.STS drošības talonus. Ja lietotājs ir autentificējies ar cita drošības talonu servisa (*STS*) palīdzību (piemēram, *LVP.STS*), tad pirms izsaukt šīs pakalpes, ir jāapmaina šis drošības talons uz *PFAS.STS* drošības talonu, griežoties pie *PFAS.STS* federācijas sniedzēja (*federation provider, FP*) (skat. 42.attēlu).



.attēls. Drošības talonu apmaiņa, griežoties pie *PFAS.STS FP*

### Pakalpju operācijas

Pakalpēm *UserProfileService* un *UserProfile­ByIdentifier­Service* ir šādas operācijas:

* *GetProperty* – izgūt profila īpašību;
* *GetMultipleProperties* – izgūt vairākas profila īpašības;
* *GetAllProperties* – izgūt visas profila īpašības;
* *SetProperty* – saglabāt / modificēt profila īpašību;
* *SetMultipleProperties* – saglabāt / modificēt vairākas profila īpašības;
* *GetPropertiesDefinitions* – izgūt profila īpašību definīcijas.

Izsaucot jebkuru pakalpes operāciju, ir obligāti jāaizpilda šādi elementi:

* *ApplicationName* – lietotnes, kuras lietotāja profila informācija tiek apstrādāta, identifikators (nosaukums), piemēram, ‘lvp’ (jebkuras operācijas izpilde ir atkarīga no lietotnes);
* *ProfileType* (var būt iezīmēts kā *nil=’true’*; iespējamās vērtības ir ‘Person’ un ‘Company’) – pazīme, kas ļauj konkretizēt profilu, ja pēc lietotāju identifikatora nevar noteikt konkrēto profilu. Šai pazīmei ir norāda, ar kuru no profiliem: personīgo (‘Person’ vai *null[[7]](#footnote-7)*) vai organizācijas (‘Company’) – strādāt, ja lietotāja identifikatorā lietotājs ir norādīts kā organizācijas (uzņēmuma vai iestādes) darbinieks. Pazīmes *ProfileType* atļautās vērtības atkarībā no norādītajām lietotāja identifikatora daļām un to interpretācija ir sniegtas 7.tabulā.

Izsaucot pakalpes *UserProfile­ByIdentifier­Service* jebkuru operāciju, papildus ir jānorāda šāds elements:

* *UserIdentifier 🡪 NameIdentifier* – lietotāja/sistēmas, kura(s) profila informācija ir jāapstrādā, identifikators (vērtību piemērus skat. 6.tabulā).

7.tabula

Iespējamie profilu veidi (atkarībā no lietotāju identifikatoriem un *ProfileType* vērtības)

| Norādītās lietotāja identifikatora daļas | Norādītā *ProfileType* vērtība | Atbilstošo profilu identificējošās daļas | Atbilstošā profila veida nosaukums |
| --- | --- | --- | --- |
| PK | null / Person | PK | Inhabitant |
| PK+AU | null / Person | PK+AU | Employee |
| Company | AU | Authority |
| AU | Company |
| PK+UR | null / Person | PK | Inhabitant |
| Company | UR | LegalEntity |
| UR[[8]](#footnote-8) | Company |
| (e-pasta adrese) | null / Person | (e-pasta adrese) | UserDeclaredIdentity |

Pakalpju operācijām atbilstošie ieejas un izejas XML elementu nosaukumi ir sniegti 8.tabulā; to struktūru shēmas (ar apakšelementu aprakstiem) ir sniegtas 43.–57.attēlos.

8.tabula

Pakalpju operācijām atbilstošie ieejas un izejas XML elementi

| Operācija | Ieejas XML elementi | | Izejas XML elementi | |
| --- | --- | --- | --- | --- |
| Pakalpe *UserProfile­Service* | Pakalpe *UserProfile­ByIdentifier­Service* | Pakalpe *UserProfile­Service* | Pakalpe *UserProfile­ByIdentifier­Service* |
| GetProperty | GetPropertyRequest | GetPropertyByIdentifierRequest | GetPropertyResponse | |
| GetMultiple­Properties | GetMultipleProperties­Request | GetMultiplePropertiesByIdentifier­Request | GetMultiplePropertiesResponse | |
| GetAll­Properties | GetAllPropertiesRequest | GetAllPropertiesByIdentifierRequest | GetAllPropertiesResponse | |
| SetProperty | SetPropertyRequest | SetPropertyByIdentifierRequest | – | |
| SetMultiple­Properties | SetMultipleProperties­Request | SetMultiplePropertiesByIdentifier­Request | – | |
| GetProperties­Definitions | GetPropertiesDefinitions–Request |  | GetPropertiesDefinitionsResponse | |

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\GetPropertyRequest.emf

.attēls. XML elementa *GetPropertyRequest* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\GetPropertyByIdentifierRequest.emf

.attēls. XML elementa *GetPropertyByIdentifierRequest* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\GetPropertyResponse.emf

.attēls. XML elementa *GetPropertyResponse* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\GetMultiplePropertiesRequest.emf

.attēls. XML elementa *GetMultiplePropertiesRequest* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\GetMultiplePropertiesByIdentifierRequest.emf

.attēls. XML elementa *GetMultiplePropertiesByIdentifierRequest* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\GetMultiplePropertiesResponse.emf

.attēls. XML elementa *GetMultiplePropertiesResponse* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\GetAllPropertiesRequest.emf

.attēls. XML elementa *GetAllPropertiesRequest* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\GetAllPropertiesByIdentifierRequest.emf

.attēls. XML elementa *GetAllPropertiesByIdentifierRequest* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\GetAllPropertiesResponse.emf

.attēls. XML elementa *GetAllPropertiesResponse* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\SetPropertyRequest.emf

.attēls. XML elementa *SetPropertyRequest* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\SetPropertyByIdentifierRequest.emf

.attēls. XML elementa *SetPropertyByIdentifierRequest* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\SetMultiplePropertiesRequest.emf

.attēls. XML elementa *SetMultiplePropertiesRequest* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\SetMultiplePropertiesByIdentifierRequest.emf

.attēls. XML elementa *SetMultiplePropertiesByIdentifierRequest* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\GetPropertiesDefinitionsRequest.emf

.attēls. XML elementa *GetPropertiesDefinitionsRequest* struktūras shēma

C:\Users\Vladislavs.Nazaruks\Documents\! work\projects\VISS 2010\UserProfile service\XML schemas\diagrams\GetPropertiesDefinitionsResponse.emf

.attēls. XML elementa *GetPropertiesDefinitionsResponse* struktūras shēma

#### Pakalpes *UserProfileByIdentifierService* operāciju izsaukumu piemēri

Pakalpes *UserProfileByIdentifierService* visu operāciju izsaukumu piemēri no C# programmas ir pieejami piesaistītajā arhīvā. Šajā programmā ir definētas šādas metodes:

* *CallGetPropertiesDefinitions* – izsauc pakalpes *UserProfileByIdentifierService* metodi *GetPropertiesDefinitions*, kas ļauj izgūt profila īpašību definīcijas;
* *CallGetProperty* – izsauc pakalpes *UserProfileByIdentifierService* metodi *GetProperty*, kas ļauj izgūt profila īpašību;
* *CallSetProperty* – izsauc pakalpes *UserProfileByIdentifierService* metodi *SetProperty*, kas ļauj saglabāt / modificēt profila īpašību;
* *CallGetMultipleProperties* – izsauc pakalpes *UserProfileByIdentifierService* metodi *GetMultipleProperties*, kas ļauj izgūt vairākas profila īpašības;
* *CallGetAllProperties* – izsauc pakalpes *UserProfileByIdentifierService* metodi *GetAllProperties*, kas ļauj izgūt visas profila īpašības;
* *CallSetMultipleProperties* – izsauc pakalpes *UserProfileByIdentifierService* metodi *SetMultipleProperties*, kas ļauj saglabāt / modificēt vairākas profila īpašības.

Pirms strādāt ar šo demonstrācijas programmu, ir jāveic tās konfigurēšana, rediģējot datni *app.config*.

Pirms izsaukt attiecīgo metodi, ir jāizveido pakalpes klienta objekts:

UserProfileByIdentifierServiceClient client =

new UserProfileByIdentifierServiceClient("ws2007FederationNoSct");

kurš jānodod kā pirmo parametru metodēm.

Metožu izsaukumu piemēri:

CallGetPropertiesDefinitions(client, "lvp", ProfileType.Person, "PK:01010110101");

CallGetProperty(client, "viss", ProfileType.Company, "AU:100000", "AuthorityName");

CallSetProperty(client, "lvp", ProfileType.Person, "PK:01010110101", "City", "City\_1");

CallGetMultipleProperties(client, "lvp", ProfileType.Person, "PK:01010110101", new string[]{"FirstName", "LastName", "City"});

CallGetAllProperties(client, "lvp", ProfileType.Person, "PK:01010110101");

CallSetMultipleProperties(client, "lvp", ProfileType.Person, "PK:01010110101", new Dictionary<string, string>() {{"FirstName", "Ivars"}, {"LastName", "Ozols"}});

Piemēram, 4.piemērs izsauc pakalpes *UserProfileByIdentifierService* operāciju *GetMultipleProperties*, pieprasot lietotāja (iedzīvotāja) ar kodu ‘PK:01010110101’ profila īpašību *FirstName*, *LastName* un *City* vērtības lietotnē *lvp*.

### Tipiskie lietošanas scenāriji

Tipisko pakalpju lietošanas scenāriju piemēri ir šādi:

1. Ir jāizgūst aktīvā lietotāja profila īpašību vērtības noteiktajā lietotnē. Šajā gadījumā ir jāizsauc pakalpes *UserProfileService* operācija *GetProperty*, *GetMultipleProperties* vai *GetAllProperties*.
2. Ir jāizgūst patvaļīga lietotāja profila īpašību vērtības noteiktajā lietotnē. Šajā gadījumā ir jāizpilda šādas darbības:

* lietotnē *PFAS.AUTH* autorizācijas apgabalā *UserProfileService* piereģistrēt operāciju *FullReadOf…* attiecīgajai lietotnei;
* lietotnē *PFAS.AUTH* piešķirt pilnvaras izsaukt iepriekšējā solī piereģistrēto operāciju lietotājam, kas veiks attiecīgo pieprasījumu;
* izsaukt pakalpes *UserProfileByIdentifierService* operāciju *GetProperty*, *GetMultipleProperties* vai *GetAllProperties*.

1. Ir jāsaglabā / jāmodificē aktīvā lietotāja profila īpašību vērtības noteiktajā lietotnē. Šajā gadījumā ir jāizsauc pakalpes *UserProfileService* operācija *SetProperty* vai *SetMultipleProperties*.
2. Ir jāsaglabā / jāmodificē patvaļīga lietotāja profila īpašību vērtības noteiktajā lietotnē. Šajā gadījumā ir jāizpilda šādas darbības:

* lietotnē *PFAS.AUTH* autorizācijas apgabalā *UserProfileService* piereģistrēt operāciju *FullWriteOf…* attiecīgajai lietotnei;
* lietotnē *PFAS.AUTH* piešķirt pilnvaras izsaukt iepriekšējā solī piereģistrēto operāciju lietotājam, kas veiks attiecīgo pieprasījumu;
* izsaukt pakalpes *UserProfileByIdentifierService* operāciju *SetProperty* vai *SetMultipleProperties*.

1. Ir jāpiereģistrē jauna lietotne, kuras lietotāju profiliem ir jāspēj piekļūt ar pakalpēm *UserProfileByIdentifierService*. Šajā gadījumā lietotnē *PFAS.AUTH* autorizācijas apgabalā *UserProfileService* ir jāpievieno operācijas *FullReadOfABC* un *FullWriteOfABC* (kur *ABC* vietā ir jābūt lietotnes nosaukumam), piešķirot tās konkrētajiem lietotājiem.

## No pārlūka izmantojamu (AJAX) tīmekļa pakalpju izstrāde (AJAX enabled services)

### Risinājuma arhitektūra

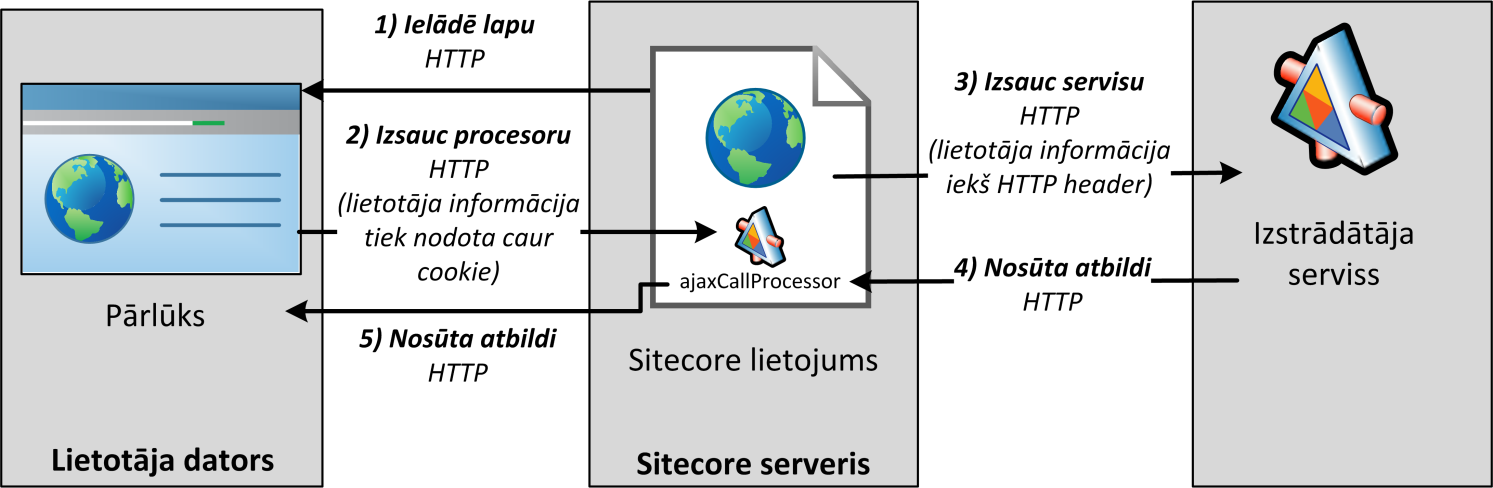
Piedāvātās infrastruktūras uzdevums ir piedāvāt izstrādātājiem iespēju veidot tīmekļa pakalpes, kuras

* tiek izsauktas no tīmekļa pārlūka (AJAX) un
* nav iekļautas (iekompilētas) VISS lietojumā.

Risinājuma pamatā ir *ajaxCallProcessor* komponente, kas ir iekļauta VISS lietojumā. Tā saņem pieprasījumus no pārlūka un šos pieprasījumus pārvirza uz izstrādātāja servisiem.

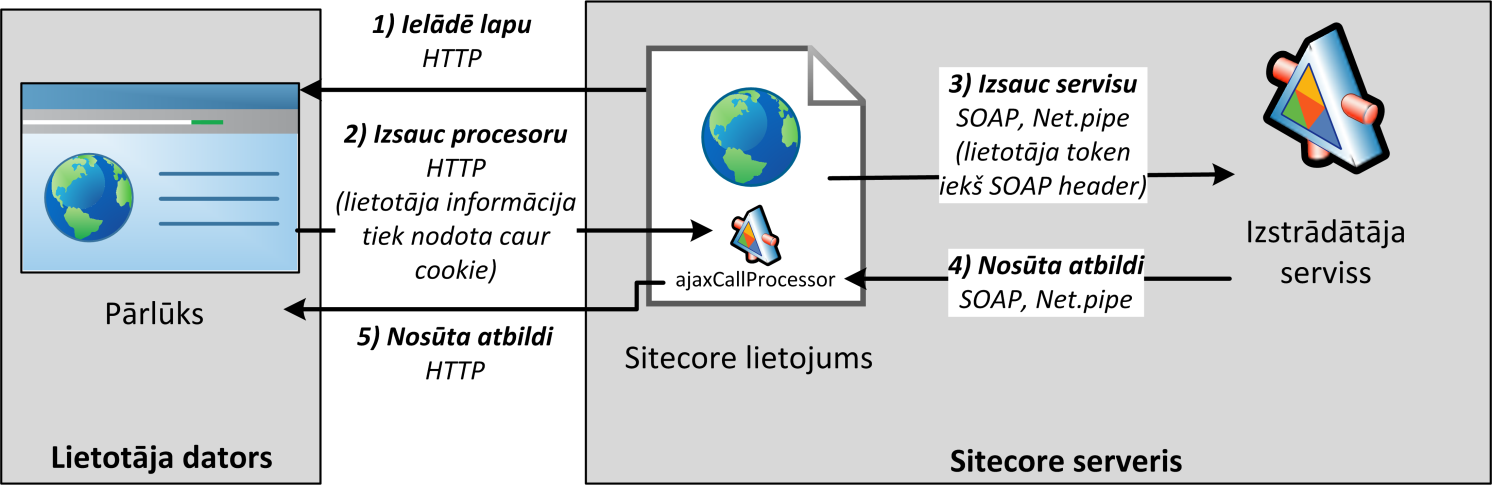
Servisu izstrādei tiek piedāvātas divas alternatīvas:

* REST serviss (caur HTTP protokolu)
* SOAP serviss (caur net.pipe protokolu)



. attēls. REST servisa izmantošanas diagramma

REST servisa gadījumā *ajaxCallProcessor* darbojas maksimāli caurspīdīgi (cenšas nemodificēt pieprasījumu un atbildi).



. attēls. Net.pipe servisa izmantošanas diagramma

SOAP servisa gadījumā *ajaxCallProcessor* HTTP pieprasījumu pārveido AjaxRequest objektā, kas ir obligāts servisa metodes ieejas parametrs. Servisa atbildes struktūrai jābūt AjaxResponse, kuru *ajaxCallProcessor* pārveidos par atbilstošu HTTP atbildi.

Pārlūki jau piedāvā risinājumu trešo pušu servisu izsaukšanai – *CORS* (*Cross-Origin Resource Sharing*). Tomēr problēmas rada novecojušas pārlūku versijas, kas ir iekļautas VISS portāla atbalstāmo pārlūku sarakstā (IE8, IE9). Nepilnīga standarta atbalsta dēļ to droša izmantošana (autentificētā lietotāja informācijas nodošana, neizmantojot *query string* un *post* vērtības) nav iespējama. Turklāt CORS konfigurācija arī var būt netriviāla. Tādēļ tika izstrādāta *ajaxCallProcessor* komponente.

### ajaxCallProcessor Sitecore konfigurācija

Lai padarītu izstrādāto servisu pieejamu, Sitecore aplikācijas konfigurācijas datnē (*App\_Config\Abc.AjaxCallProcessor.config*) ir jāpapildina *ajaxCallProcessor* iestatījumi.

Konfigurācijas piemērs:

<abc.ajaxCallProcessor>

  <switch path="/Ajax/Rest" type="Abc.Sitecore.Pipelines.AjaxCalls.Switches.RestSwitch, Abc.Sitecore">

    <!--http://lvpsc/Ajax/Rest/Test-->

    <service path="/Test"

             url="https://eservices:7777/Abc.AjaxEnabledTestService/Rest/RestService.svc"

             timeoutInMilliseconds="1000"

             requireUserData="true"/>

  </switch>

  <switch path="/Ajax/NetPipe" type="Abc.Sitecore.Pipelines.AjaxCalls.Switches.NetPipeSwitch, Abc.Sitecore">

    <service path="/Test"

             url="net.pipe://eservices/Abc.AjaxEnabledTestService/NetPipe/NetPipeService.svc"

             timeoutInMilliseconds="2000"

             requireUserData="true">

      <!--http://lvpsc/Ajax/NetPipe/Test/GetSimple-->

      <method path="/GetSimple"

              action="http://ivis.eps.gov.lv/ISS/NetPipeService/v1-0/GetSimple"

              replyAction="http://ivis.eps.gov.lv/ISS/NetPipeService/v1-0/GetSimpleResponse"/>

    </service>

  </switch>

</abc.ajaxCallProcessor>

9. tabula

**ajaxCallProcessor konfigurācijas elementi**

| **Iestatījums** | **Apraksts** |
| --- | --- |
| switch | Pieprasījumu pārvirzītāja konfigurācija. Pagaidām ir pieejami divi pārvirzītāji – REST un SOAP servisiem. |
| switch\path | Adreses daļa, caur kuru pārvirzītājs būs pieejams. Piemēram, ja ir norādīta vērtība „/Ajax/Rest” un Sitecore lietojuma adrese ir <http://viss.gov.lv>, tad reģistrētie servisi būs pieejami caur <http://viss.gov.lv/Ajax/Rest/>... . Šī elementa vērtībai jābūt unikālai katram pārvirzītājam. |
| switch\type | Klases, kura realizēs pārvirzīšanu, nosaukums. Klasei jāimplementē interfeiss ISwitch. Izstrādātājiem nav paredzēts realizēt savus pārvirzītājus. |

10. tabula

**REST pārvirzītāja (ajaxCallProcessor/switch[type=RestSwitch]) konfigurācijas elementi**

| **Iestatījums** | **Apraksts** |
| --- | --- |
| service | Apraksta vienu pārvirzāmo servisu. |
| service\path | Adreses daļa, caur kuru serviss būs pieejams. Šī elementa vērtībai jābūt unikālai katra servisa pārvirzītāja ietvaros. |
| service\url | Izsaucamā servisa adrese. |
| service\timeoutInMilliseconds | Laiks pēc kura nesaņemot atbildi no servisa pārvirzītājs pārlūkam atgriež HTTP kļūdu 504 (Gateway timeout). Pēc noklusējuma 1 minūte. |
| service\requireUserData | Vai servisa izsaukumam pievienot HTTP header Authorization ar aktīvā lietotāja identifikatoru. Pēc noklusējuma false. |

Servisa izsaukuma url veidošanas piemērs:

1. Pārlūks pieprasa adresi <https://viss.gov.lv/ajax/rest/test/LongGet?execTime=500>
2. *switch\path* ir norādīta vērtība *„/Ajax/Rest”*
3. *switch\service\path* ir norādīta vērtība *„Test”*
4. switch\service\url ir norādīta vērtība <https://myserver/myService/RestService.svc>
5. Pārsūtītājs izsauc adresi <https://myserver/myService/RestService.svc/LongGet?execTime=500>

11. tabula

**SOAP pārvirzītāja (ajaxCallProcessor/switch[type=NetPipeSwitch]) konfigurācijas elementi**

| **Iestatījums** | **Apraksts** |
| --- | --- |
| service | Apraksta vienu pārvirzāmo servisu. |
| service\path | Adreses daļa, caur kuru serviss būs pieejams. Šī elementa vērtībai jābūt unikālai katra servisa pārvirzītāja ietvaros. |
| service\url | Izsaucamā servisa adrese. |
| service\timeoutInMilliseconds | Laiks pēc kura nesaņemot atbildi no servisa pārvirzītājs pārlūkam atgriež HTTP kļūdu 504 (*Gateway timeout*). Pēc noklusējuma 1 minūte. |
| service\requireUserData | Vai servisa izsaukumam pievienot SOAP *header* ar aktīvā lietotāja talonu. |
| service\method | Izsaucamās servisa metodes konfigurācija. Atšķirībā no REST servisiem SOAP nepieciešams norādīt konfigurāciju katrai izsaucamajai metodei. |
| service\method\path | Adreses daļa, caur kuru metode būs pieejama. |
| service\method\action | Izsaucamās metodes pieprasījuma SOAP *action*. |
| service\method\replyAction | Izsaucamās metodes atbildes SOAP *action*. Pēc noklusējuma *service\method\action* vērtība + "Response". |

### Servisu izstrāde

#### REST serviss

REST servisu izmantošanai caur *ajaxCallProcessor* nevajadzētu ievērojami atšķirties no tiešas izsaukšanas. Tādēļ arī nepastāv īpaši izstrādes ierobežojumi.

Vienīgā specifika ir lietotāja informācijas saņemšana. Aktīvā lietotāja identifikators tiks iekļauts HTTP Authorization galvenē (ja konfigurācijā būs iespējota lietotāja informācija sūtīšana –skat. 7.3.2 nodalījumu). Pagaidām REST servisiem nav iespējams saņemt lietotāja talonu.

#### SOAP serviss

SOAP servisus pagaidām iespējams izmantot tikai caur *net.pipe* protokolu. Atbilstoši to uzstādīšana jāveic uz tā paša servera, kur Sitecore lietojums.

Katras izsaucamās metodes ieejas parametram jābūt ar tipu AjaxRequest un atgriežamajai vērtībai – AjaxResponse. Metožu nosaukumiem nekādi specifiski ierobežojumi netiek izvirzīti. Abas minētās klases un citi nepieciešamie datu tipi ir pieejami *Abc.AjaxEnabledServices.dll* bibliotēkā.

Metodes piemērs:

public AjaxResponse GetComplex(AjaxRequest request)

{

    MyParameterType param = request.GetContent<ComplexParam>();

    // TODO: real business logic

    MyParameterType result = param;

    return new AjaxResponse(result, request.AcceptedContentType);

}

Lai no servisa tālāk izsauktu citu drošo servisu ar autentificēto lietotāju, var

1. Izmantot SecuredOperationContextScope, ja izsaucamais serviss ir biznesa loģikas maršrutēšanas serviss (*BLRouter*):

var address = new EndpointAddress(

    "net.pipe://localhost/UserProfileRouter/RouterService.svc/UserProfileService");

var client = new UserProfileServiceClient(new RouterBinding(), address);

var request = new UP.RequestStructure { ApplicationName = "lvp" };

**using (new SecuredOperationContextScope(client.InnerChannel))**

{

    PropertiesStructure response = client.GetAllProperties(request);

}

1. STS apmainīt lietotāja talonu pret servisa izsaukšanai paredzēto talonu un servisa izsaukumu veikt ar šo jauno talonu. Šāda pieeja nepieciešama, drošo servisu izsaucot „pa tiešo” (bez *BLRouter*). Lietotāja talons pieejams caur AjaxRequest.UserToken parametru.

12.tabula

AjaxRequest īpašības un metodes

|  |  |
| --- | --- |
| **Īpašība vai metode** | **Apraksts** |
| ContentType ContentType  { get; set; } | HTTP pieprasījuma Content-Type galvenes vērtība. |
| ContentType AcceptedContentType  { get; set; } | HTTP pieprasījuma Accept galvenes vērtība. |
| string EndOfUrl { get; set; } | Pārvirzītāja saņemtās HTTP adreses nobeiguma daļa, kas atrodas aiz konfigurācijā norādītās metodes *path* daļas. |
| NameValueCollection QueryString  { get; } | Apstrādāta (*parsed*) query string vērtība no EndOfUrl īpašības. |
| string RawContent { get; set; } | Pārvirzītāja saņemtā HTTP pieprasījuma saturs. |
| NameValueCollection Form { get; } | Apstrādāta (*parsed*) pieprasījuma satura (RawContent) vērtība (pieņemot, ka pieprasījuma saturs ir x-www-form-urlencoded formātā). |
| SecurityToken UserToken { get; } | Lietotāja drošības talons. |
| bool HasUserInfo { get; } | Pazīme, vai lietotāja drošības talons ir pieejams. |
| T GetContent<T>() | Deserializē pieprasījuma saturu (RawContent) atbilstoši ContentType vērtībai. Saturam jābūt XML vai JSON formātā.   * T: datu tips, uz kuru veikt deserializāciju. * Atgriež: deserializēto objektu. |
| string GetUserClaim(  string claimType) | No lietotāja talona izgūst pretenzijas (*claim*) vērtību.   * claimType: pretenzijas identifikators. * Atgriež: pretenzijas vērtību. |

13.tabula

AjaxResponse īpašības un metodes

|  |  |
| --- | --- |
| **Īpašība vai metode** | **Apraksts** |
| ContentType ContentType  { get; set; } | Pārvirzītāja nosūtītās HTTP atbildes Content-Type galvenes vērtība. |
| string Content { get; set; } | HTTP atbildes saturs. |
| int? StatusCode { get; set; } | HTTP atbildes statusa kods. |
| AjaxResponse() | Konstruktors. |
| AjaxResponse(  object data,  ContentType contentType =  ContentType.Json) | Konstruktors.   * data: objekts, kuru serializēt atbilstoši contentType vērtībai un saglabāt Content īpašībā. * contentType: īpašības ContentType vērtība. |
| AjaxResponse(  object data,  ContentType contentType,  HttpStatusCode?  httpStatusCode = null) | Konstruktors.   * data: objekts, kuru serializēt atbilstoši contentType vērtībai un saglabāt Content īpašībā. * contentType: īpašības ContentType vērtība. * httpStatusCode: īpašības StatusCode vērtība. |

### Piemēri

Lai atvieglotu servisu izstrādi, ir pieejams *Visual Studio* servisu piemēra projekts *Abc.AjaxEnabledTestService*. To var lejupielādēt no dokumentu krātuves (līdzīgi kā šo dokumentu).

Piemērā realizēts viena tīmekļa pakalpe izmantošanai caur REST (*RestService.svc*) un otra – izmantošanai caur SOAP (*NetPipeService.svc*).

Lai izmēģinātu servisu darbībā nepieciešams:

1. Palaist servisu (ja ir vēlme darbināt SOAP servisus, visticamāk nāksies veikt izmitināšanu IIS, jo lietojumam nepieciešams *net.pipe binding*).
2. Sitecore aplikācijai pievienot *ajaxCallProcessor* konfigurāciju (konfigurācijas piemērs atrodams piemēra projekta datnē *Sitecore.Abc.AjaxCallProcessor.config*).
3. Jāuzstāda *Chrome* pārlūka paplašinājums *PostMan* un tajā jāielādē HTTP pieprasījumu piemēri no projekta (*\*.json* datnes). Jāmodificē pieprasījumu adreses (URL) atbilstoši konkrētajai videi.
4. Ja ir vēlme darbinātu lietotāju profila datu izgūšanu, projekta iestatījumos (appSettings) jānorāda videi atbilstoša UserProfileRouterUrl vērtība.

1. Facets – nodrošina elementa agregāciju (grupēšanu) meklēšanas pieprasījumos [↑](#footnote-ref-1)
2. Klasifikatora vārdtelpas identifikators http://ivis.eps.gov.lv/ClasificationScheme/informationSystemScheme [↑](#footnote-ref-2)
3. Klasifikatoru vārdtelpas identifikatori: http://ivis.eps.gov.lv/ClasificationScheme/Authority vai http://ivis.eps.gov.lv/ClasificationScheme/VirtualBody [↑](#footnote-ref-3)
4. Klasifikatora vārdtelpas identifikators ir http://ivis.eps.gov.lv/ClasificationScheme/ATVK [↑](#footnote-ref-4)
5. Facets – nodrošina elementa agregāciju (grupēšanu) meklēšanas pieprasījumos [↑](#footnote-ref-5)
6. skat. http://en.wikipedia.org/wiki/Internet\_media\_type [↑](#footnote-ref-6)
7. Šeit un tālāk šajā nodaļā vērtība *null* īpašībai/elementam *ProfileType* nozīmē to, ka attiecīgais XML elements ir tukšs un iezīmēts ar *nil=’true’*. [↑](#footnote-ref-7)
8. Lietotāja identifikatoru, kas sastāv tikai no UR daļas, ir iespējams norādīt, tikai izsaucot pakalpi *UserProfile­ByIdentifier­Service*. [↑](#footnote-ref-8)