LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

„Tīmekļbāzēts latviešu valodas pareizrakstības rīks”

MAĢISTRA DARBS

Autors: Jānis Akmentiņš
Stud. apl. nr.: ja07017
Darba vadītājs: Pēteris Paikens
pētnieks,
Latvijas Universitātes Matemātikas un informātikas institūts

Rīga, 2013

Anotācija

Pareizrakstības kļūdas ir liela problēma Latvijas tīmekļa resursos. Tās ir atrodamas pat ziņu portālu rakstos. Ziņu portālu komentāros šī problēma ir vēl aktuālāka. Rīku klāsts, ar kuriem cilvēki tīmekļa vidē varētu pārbaudīt latviešu valodas teksta pareizrakstību, ir stipri ierobežots.

Darba mērķis ir izstrādāt jauna rīka prototipu, kuru varētu brīvi integrēt tīmekļa lapās. Papildus tam, rīks paredz arī API funkcionalitāti, ar kuras palīdzību pareizrakstības rīku iespējams izmantot arī darbvirsmas (*Desktop*) un mobilajās lietotnēs. Šādā veidā varētu uzlabot vispārējo latviešu valodas kvalitāti Latvijas interneta resursos.

Darba ietvaros tika izpētītas 3 galvenās pareizrakstības rīka komponentes: kļūdu atrašana, labojumu piedāvāšana, kā arī labojumu kārtošana pēc to atbilstības. Rezultātā tika izveidots rīka prototips, kuru brīvi iespējams integrēt mājaslapās, kā arī API modulis, kas paver plašas iespējas rīka tālākajai attīstībai.

Atslēgas vārdi

Latviešu valoda, pareizrakstība, tīmekļa vietnes, mobilās lietotnes, PHP, Hunspell.

Abstract

Web based spellchecker for Latvian

Spelling problems are a major issue in Latvian internet. Spelling errors can be found even in news portal articles. The problem is even bigger if we look at the user comments of those articles. The ammount of tools which can be used by users to check their spelling is quite limited.

The puprose of this thesis is to develop a prototype of a complete new tool, which could be easily integrated in any web page. The tool also will have API functionality, so that it will be possible to integrate the tool in Desktop and mobile applications. In result of that the tool will help to improve the quality of text in latvian web reasources.

In this thesis author has been exploring 3 main components of spell checking tool: finding errors, suggesting corrections and sorting the corrections. In result author has developed a prototype of a new tool which can be easily integrated in any web page, and also an API module, which brings a lot more opportunities of improving the tool.

Keywords

Latvian, spelling, web pages, mobile applications, PHP, Hunspell.

Autoreferāts

Autors šī maģistra darba ietvaros ir:

1. Veicis pētījumu par pareizrakstības kļūdām Latvijas tīmekļa resursos (skat. 1.1. nod.).
2. Izpētījis pašlaik pieejamos latviešu valodas pareizrakstības rīkus (skat. 1.2. nod.).
3. Apskatījis kļūdu atrašanas risinājumus (skat. 4.1. nod.)
4. Izveidojis divus praktiskus pareizrakstības kļūdu noteikšanas risinājumus un tos savstarpēji salīdzinājis (skat. 4.2. nod.).
5. Izpētījis pareizrakstības kļūdu labojumu atrašanas paņēmienus (skat. 4.3. nod.).
6. Analizējis pareizrakstības kļūdu labojumu kārtošanas metodes (skat. 4.4. nod.).
7. Izveidojis valodas un kļūdu modeļus, uz kuriem tiek balstīts pareizrakstības rīka prototips (skat. 4.5.1. un 4.5.2. nod.).
8. Izstrādājis pareizrakstības rīka prototipu, kurš sastāv no 3 komponentēm: kļūdu atrašanas, kļūdu labojumu atrašanas un kļūdu labojumu kārtošanas (skat. 4.5.3., 4.5.4. un 4.5.5. nod.).
9. Veicis ātrdarbības testēšanu rīka prototipa komponentēm (skat. 4.2.5. un 4.5.6. nod.).
10. Izveidojis pareizrakstības rīka API moduli (skat. 4.6. nod.).

SATURA RĀDĪTĀJS

APZĪMĒJUMU SARAKSTS

Darbā izmantotie saīsinājumi un apzīmējumi redzami tabulā „Apzīmējumu saraksts”.

Apzīmējumu saraksts

Apzīmējums	**Skaidrojums**
APC	Alternative PHP Cache - bezmaksas atvērtā koda programmatūra, kura optimizē PHP mašīnkodu un noglabā to kešatmiņā.
WinRar	Datņu kompresēšanas rīks. Izplatīts gan Windows, gan UNIX vidēs.
gZip	Datņu kompresēšanas rīks. Izplatīts UNIX vidē.
API	Lietojumprogrammu programmēšanas interfeiss (application programming interface)
MySQL	Relāciju datubāzu vadības sistēma.
PHP	Hypertext preprocessor – tīmekļa programmēšanas valoda
ER	„Entity – Relationship model”. Relāciju modelis, kurš attēlo datubāzes struktūru un tabulu relācijas.
JSON	„JavaScript Object Notation”. Datu apmaiņas formāts.
HTTP	„Hypertext Transfer Protocol”. Datu apmaiņas protokols.
Javascript	Programmēšanas valoda, kura tiek izmantota tīmekļa vietnēs. Tā tiek izpildīta uz klienta datora.
AJAX	Tīmekļa izstrādes tehnoloģija, ar kuras palīdzību iespējams veikt asinhronus datu apmaiņas pieprasījumus ar serveri.

IEVADS

Ikdienā mēs ļoti daudz lietojam dažādas interneta pārlūkprogrammas dažādiem mērķiem. Lasām ziņas, lasām ziņu komentārus, iesaistāmies diskusijās sociālajos tīklos u.tml. Mēs ne tikai uzņemam informāciju, bet arī dalāmies ar to – piemēram, paužot savu viedokli komentāros. Ļoti bieži Latvijas ziņu portālos ir nācies lasīt rakstus, kuru autors steigā nav pamanījis vairākas pareizrakstības kļūdas. Vēl trakāk ir ar lietotāju komentāriem šajos portālos. Teju katrā otrajā, trešajā komentārā atrodamas vairākas pareizrakstības kļūdas.

Tomēr tīmekļa pārlūkprogrammas nav vienīgais veids, kā cilvēki dalās ar informāciju internetā. Mūsdienās arvien populārākas kļūst dažādas mobilās lietotnes. Bez, tam, ievadot tekstu ar telefonu, iespēja kļūdīties ir stipri vien lielāka, nekā ievadot tekstu ar datora klaviatūru.

Šī maģistra darba mērķis ir konceptuāli izstrādāt jaunu rīku, kuru iespējams integrēt jebkurā mājaslapā. Mājaslapu lietotājiem nebūs nepieciešamas nekādas papildus zināšanas, kā arī nebūs jāinstalē pārlūkprogrammas papildinājumi. Rīku mājaslapā ērti varētu integrēt mājaslapas izstrādātāji. Viena no rīka pamatidejām būs tā universālā integrēšanas iespēja – mājaslapas izstrādātājs jebkuru teksta ievadlauku savā mājaslapā varēs brīvi savienot ar pareizrakstības rīku. Šāda rīka izstrādāšana potenciāli uzlabotu situāciju Latvijas internetā – saturs kļūtu kvalitatīvāks un lasāmāks, jo tajā būtu mazāk pareizrakstības kļūdu.

Papildus tam, nepieciešams izstrādāt API risinājumu, ar kura palīdzību tiktu krietni palielināts rīka izmantošanas potenciāls. Izmantojot šo risinājumu, rīku būtu iespējams integrēt mobilajās lietotnēs, darbvirsmas lietotnēs u.c.

Darbā vispirms tiek padziļināti pētīta tēmas aktualitāte un pareizrakstības kļūdu problēmas Latvijas interneta resursos, pēc tam tiek apskatīti pieejamie pareizrakstības rīki un to darbība, savukārt, pēc tam autors pievēršas jauna rīka izstrādei, kuras ietvaros tiek pētītas 3 pareizrakstības rīka komponentes: pareizrakstības kļūdu atrašana, kļūdu labojumu atrašana, kā arī atrasto kļūdu labojumu kārtošana pēc to piemērotības. Darba rezultātā tiek iegūts jauna pareizrakstības rīka prototips.

1 SITUĀCIJAS APRAKSTS

1.1 Pareizrakstības kļūdas Latvijas interneta resursos

Bieži vien, lasot jaunākos ziņu ierakstus populārākajos Latvijas ziņu portālos, nākas uzdurties uz autora nepamanītajām pareizrakstības kļūdām. Ir redzēti gadījumi, kad tās ir pat ziņas virsrakstā. Šādas kļūdas visbiežāk tiek pieļautas steigā rakstītiem rakstiem, piemēram, sporta spēļu apskatiem, kuri tiek publicēti dažas minūtes pēc spēles beigām. Acīmredzot, ziņu portāla redaktors šos rakstus ir rakstījis steigā, un portāls tehniski nenodrošina iespēju pirms raksta publicēšanas pārliecināties, vai tajā nav pareizrakstības kļūdu.

Tomēr šādas kļūdas tiek salīdzinoši ātri izlabotas. Ziņu portālus apmeklē daudzi lasītāji, kuri šīs kļūdas pamana un vai nu ziņo uzreiz redakcijai, vai izsaka aizrādījumus raksta komentāros.

Sliktāka situācija ir ar rakstu komentāriem. Šo saturu rada cilvēki ar visdažādāko izglītību no visdažādākajiem sociālajiem slāņiem. Pārsvarā vienīgā korekcija no tīmekļa vietņu administratoru puses ir rupjo un aizvainojošo vārdu cenzēšana. Tomēr ar pareizrakstības kļūdu labošanu tīmekļa resursu komentāros neviens īpaši nenodarbojas.

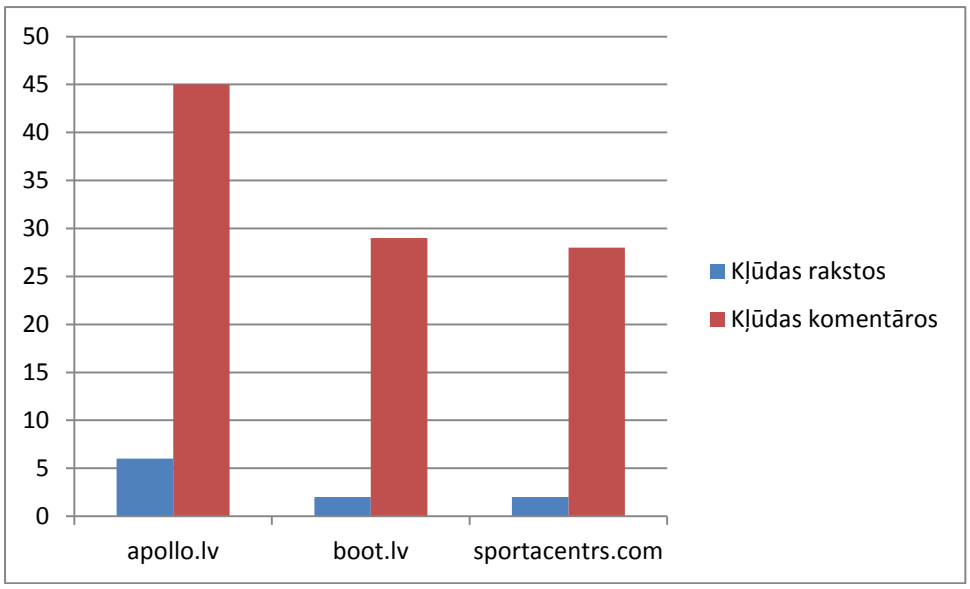
Lai aptuveni novērtētu pareizrakstības kļūdu sastopamību Latvijas interneta resursos, tika izvēlēti daži no populārākajiem ziņu portāliem Latvijā:

- apollo.lv
- boot.lv
- sportacentrs.com

Katrs no šiem portāliem ir tendēts uz savu mērķauditoriju. Apollo.lv ir plaša spektra ziņu portāls, kurš tendēts uz visa veida auditoriju. Boot.lv ir IT sfēras ziņu portāls, kura auditorija pārsvarā ir IT nozares pārstāvji un tehnoloģiju entuziasti. Sportacentrs.com satur rakstus par aktuālajiem notikumiem sportā.

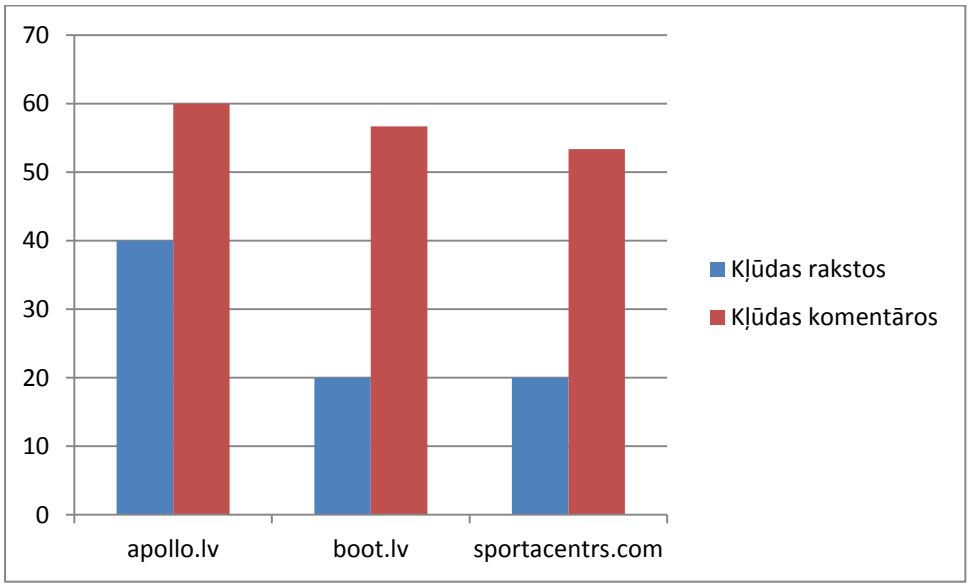
Nejauši tika izvēlēti 10 raksti katrā no šiem portāliem. Tika noteikts pareizrakstības kļūdu skaits katrā no šiem rakstiem, kā arī pareizrakstības kļūdu skaits 3 nejauši izvēlētiem komentāriem katram no šiem rakstiem. Pareizrakstības kļūdas tika noteiktas empīriskā ceļā – iekopējot tekstu OpenOffice Writer programmā, kurai ir uzstādīts latviešu valodas pareizrakstības kļūdu noteikšanas spraudnis.

Rezultāti parādīja (skatīt attēlu 1.1.1.), ka pēc autora mērījumiem, apollo.lv ir viskļūdainākie raksti un viskļūdainākie komentāri. Ir redzama kopējā tendence – ziņu portālos pareizrakstības kļūdu skaits komentāros ir diezgan liels.



1.1.1. att. **Pareizrakstības kļūdu skaits Latvijas ziņu portālos**

Ja mēs paskatāmies šo statistiku citā griezumā – cik procentuāli no šiem katra portāla 10 apskatītajiem rakstiem un 30 apskatītajiem komentāriem bija vismaz 1 pareizrakstības kļūda, tad mērījumu rezultātus varam redzēt attēlā 1.1.2.



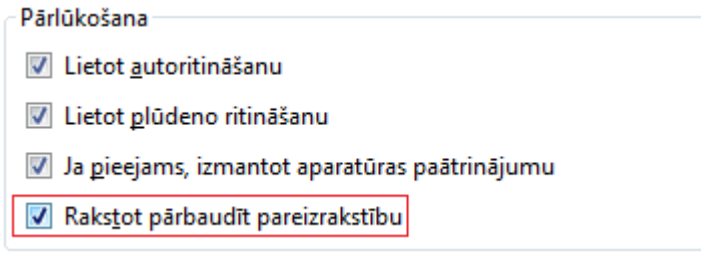
1.1.2. att. **Procentuālais kļūdaino rakstu skaits Latvijas ziņu portālos**

Jāpiezīmē, ka viens no faktoriem, kas ietekmē šos kļūdu skaita rādītājus, ir tas, ka cilvēki komentārus raksta translitā. Piemēram, „šis ir mans komentārs" vietā raksta „shis ir mans komentaars". Interneta vidē šis paradums ir diezgan izplatīts.

1.2 Pieejamie latviešu valodas pareizrakstības rīki

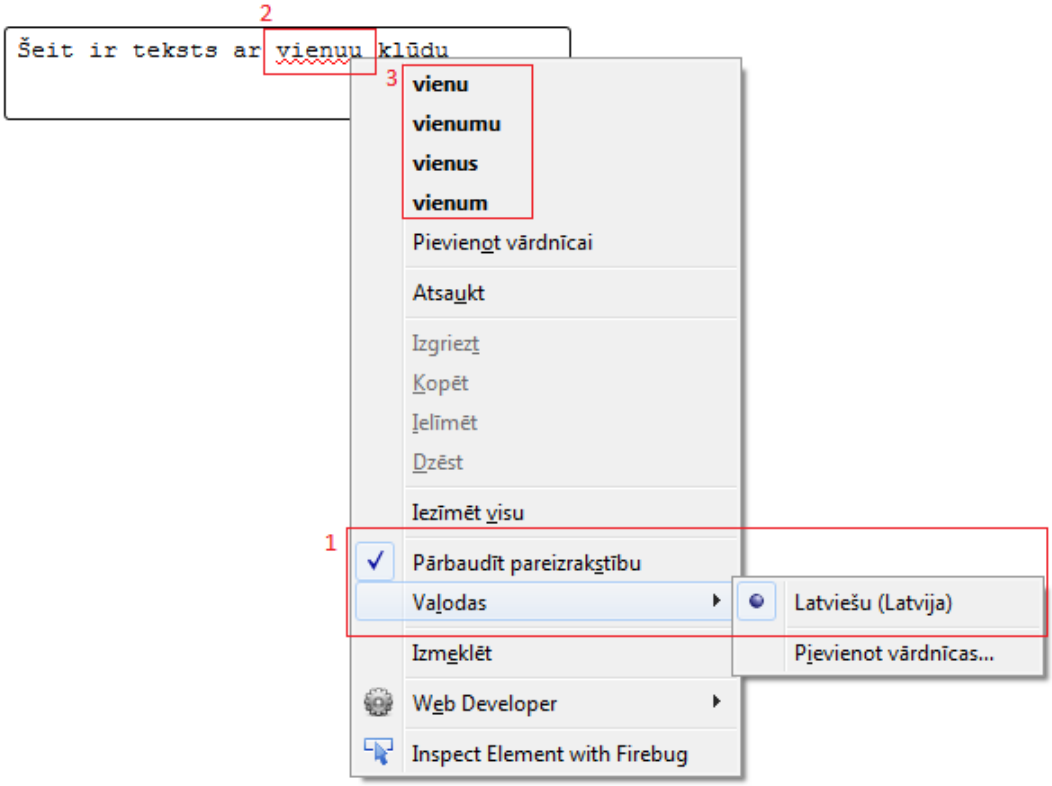
1.2.1 Interneta pārlūkprogrammas

Vairākās mūsdienu interneta pārlūkprogrammās ir jau iebūvēti pareizrakstības rīki. Konkrēti apskatīsim Mozilla Firefox (versija 20.0.1), Google Chrome (versija 26.0.1410.64 m) un Opera (versija 12.12). Visām minētajām pārlūkprogrammām ir iespēja uzstādījumu logā iespējot pareizrakstības pārbaudes rīku – skatīt attēlu 1.2.1.1.



1.2.1.1. att. **Firefox uzstādījumu logs**

Kad šī opcija ir iespējota, atveram jebkuru mājaslapu, kurā ir kāds teksta ievadlauks. Demonstrāciju var apskatīt attēlā 1.2.1.2. Blokā, kurš apzīmēts ar ciparu 1, redzama rīka konfigurācija. Blokā, kurš apzīmēts ar ciparu 2, redzama Firefox rīka atrastā pareizrakstības kļūda, kas pasvītrota ar viļņotu līniju, savukārt, blokā, kurš apzīmēts ar ciparu 3, redzami pareizrakstības rīka piedāvātie potenciālie kļūdainā vārda labojumi.



1.2.1.2. att. **Fifefox pareizrakstības kļūdu labošanas rīks**

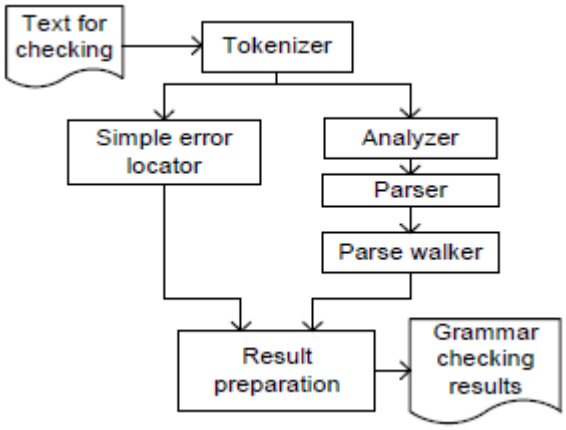
Visām minētajām pārlūkprogrammām pareizrakstības rīks ir balstīts uz Hunspell [1] pareizrakstības rīku. Šis rīks detalizēti tiks apskatīts šī darba turpmākajās nodaļās.

1.2.2 Tildes birojs

Tildes birojs ir maksas programmatūra, kurā kā viena no komponentēm ietilpst pareizrakstības pārbaudes rīks, kurš darbojas kā spraudnis Microsoft Office, Open Office un Libreoffice programmās [2]. Šis rīks pārbauda katra vārda pareizrakstību, kļūdaini uzrakstītu vārdu piedāvā aizstāt ar pareizu. Viennozīmīgi labojamus vārdus izlabo automātiski [3]. Rīka darbības piemēru Microsoft Word vidē var aplūkot attēlā 1.2.2.1. Pirmo latviešu valodas teksta pareizrakstības pārbaudītāju izstrādāja Tilde 1995. gadā [4]. Šo gadu laikā rīks ir attīstījies līdz teicamam līmenim, tāpēc uzskatāms par šīs sfēras līderi.

1.2.2.1. att. **Tildes biroja pareizrakstības pārbaudes rīks**

Gramatikas pārbaudes rīks sastāv no vairākām atsevišķām komponentēm. Katrai ir savs uzdevums. Vairums komponenšu tiek izmantotas noteiktā secībā, jo viena komponente izmanto citas komponentes rezultātus. Gramatikas pārbaudītāja arhitektūra apskatāma attēlā 1.2.2.2.



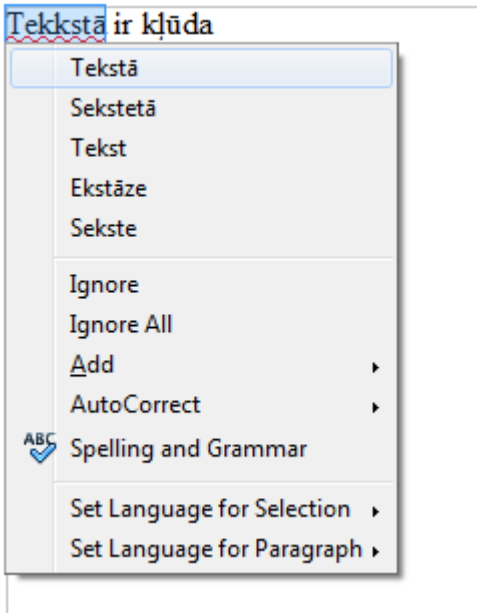
1.2.2.2. att. **Gramatikas pārbaudītāja arhitektūra**

Gramatikas pārbaudītājs neapstrādā visu tekstu uzreiz. Teksts tiek sadalīts atsevišķās daļās. To nodrošina „Tokenizer" komponente. Tālāk „Simple error locator" komponente ar regulāro izteiksmju palīdzību atrod vienkāršas formatējuma kļūdas. „Analyzer" komponente veic morfoloģisko analīzi katrai dotajai teksta daļai. „Parser" komponente apstrādā tekstu, izmantojot gramatikas likumu kopu. „Parse walker" komponente apstrādā „Parser" komponentes atrastās kļūdas, kā arī ģenerē iespējamos kļūdu labojumu variantus. „Result preparation" komponente apkopo visus iegūtos rezultātus [5].

1.2.3 OpenOffice latviešu valodas pareizrakstības pārbaudes papildinājums

Open office ietver iespēju uzstādīt dažādus papildinājumus. Tos var lejupielādēt Open office mājaslapā. Katrs papildinājums ir atsevišķa „oxt" datne, kuru var uzstādīt ar „Papildinājumu Menedžeri". Viens no šādu papildinājumu veidiem ir vārdnīcas [6]. Tās sevī iekļauj pareizrakstības kļūdu pārbaudes mehānismu, un ir lejupielādējamas dažādām valodām, tajā skaitā arī latviešu valodai.

„Latviešu valodas pareizrakstības pārbaudes moduli" ir izstrādājis Jānis Eisaks un šobrīd ir pieejama šī moduļa 0.9.6 versija, kura ir izstrādāta 2013. gada 8. aprīlī [7]. Šī rīka darbības piemēru Open office Writer vidē var aplūkot attēlā 1.2.3.1.



1.2.3.1. att. **Open office pareizrakstības pārbaudes rīks**

Tāpat, kā iepriekš aprakstītie interneta pārlūkprogrammās iebūvētie pareizrakstības pārbaudes rīki, arī šis Open office papildinājums izmanto Hunspell programmatūru, kuras funkcionalitāte tiks izklāstīta turpmāk tekstā.

1.3 Nepieciešamība pēc jauna rīka

Iepriekš aprakstītais Latvijas tīmekļa portālu kļūdu skaita pētījums norāda uz to, ka cilvēki ikdienā īpaši neizmanto pareizrakstības pārbaudes rīkus. Cilvēku rakstības stils ir dažāds – daži vienkārši ir aizmirsuši vai nezin gramatiku, citi vienkārši pieļauj neuzmanības kļūdas. Ņemot vērā to, ka aktualizēta tiek konkrēti pareizrakstības kļūdu problēma tīmekļa vidē, ir saprotams, ka cilvēki neizmanto Tildes Biroja vai Open Office piedāvātos

risinājumus, lai pārliecinātos par sava teksta pareizību. Tas ir ļoti neērti un laikietilpīgi – vispirms uzrakstīt tekstu kādā teksta redaktorā (piemēram, Microsoft Word) un pēc tam iekopēt tīmekļa vietnē. Kas attiecas uz pārlūkprogrammās iebūvētajiem pareizrakstības pārbaudītājiem – iespējams, ka cilvēki vienkārši nav uzlikuši pareizos iestatījumus, lai iespējotu latviešu valodas pareizrakstības kļūdu pārbaudītāju, vai arī vienkārši nemaz nenojauš par šādas iespējas esamību.

Respektīvi, pašlaik tīmekļa vidē neeksistē neviens bezmaksas rīks, kuru jebkurš mājaslapas administrators vai programmētājs varētu brīvi integrēt savā mājaslapā un kurš atbalstītu latviešu valodu. Eksistē līdzīgs rīks - http://www.webspellchecker.net, kurš piedāvā gan integrāciju mājaslapā, gan arī API funkcijas, tomēr tam ir viens liels mīnuss – tas neatbalsta latviešu valodu.

Otrs virziens, kurā pašlaik nav labu bezmaksas pareizrakstības noteikšanas rīku latviešu valodai, ir mobilās lietotnes. Ir dažādas mobilās ierīces, ar kurām var pārlūkot tīmekļa vietnes un ievadīt tajās tekstuālu informāciju, tomēr nav universāla veida, kā visas šīs ierīces varētu aprīkot ar pareizrakstības rīku.

Šo iemeslu dēļ autors uzskata, ka problēma Latvijas tīmekļa vidē ir ļoti aktuāla. Ir nepieciešamība izstrādāt rīku, kuru varētu brīvi integrēt mājaslapās. Līdz ar to, piemēram, ziņu portālu komentāru modulis spētu lietotājam paziņot to, ka komentāra teksts ir kļūdains. Papildus tam, rīks sevī iekļautu API funkcionalitāti. Tas nozīmē, ka, izmantojot HTTP pieprasījumus, jebkurš mobilo vai darbvirsmas lietotņu izstrādātājs varēs savā lietotnē brīvi integrēt pareizrakstības rīku savā realizācijā. Ar šāda rīka palīdzību varētu uzlabot vispārējo situāciju Latvijas tīmekļa resursos - uzlabot lietotāju rakstības kultūru, kā arī tīmekļa informatīvo saturu kā tādu.

2 PAREIZRAKSTĪBAS KĻŪDU PROBLĒMA

Pastāv vairāku veidu pareizrakstības kļūdas:

- Ne-vārdu kļūdas
- Gramatikas kļūdas
- Vārdu kļūdas (gramatikas kļūdu apakškopa)

Ne-vārdu kļūdas ir visvienkāršāk atrodamas. Piemēram, ir viegli secināt, ka latviešu valodā nav vārda „studnts" (pareizais vārds – „students"). Vienkāršākais šādu kļūdu atrašanas veids – salīdzināt doto vārdu ar visiem zināmajiem vārdiem latviešu valodā. Ja salīdzināšanas beigās secinām, ka šāds vārds mūsu zināmo vārdu krājumā nav bijis, tad secinām, ka pieļauta kļūda [8].

Vārdu kļūdas atrast ir ievērojami grūtāk. Šīs kļūdas ir gramatikas kļūdu grupas apakškopa, tomēr tīmekļa vidē tās ir tik izplatītas, ka tiks apskatītas atsevišķi. Mēs zinām, ka latviešu valodas gramatikas likumi nosaka to, ka saiklis „vai" saista vienlīdzīgus teikuma locekļus, kuri rakstīti ar dažādu nozīmi. Piemēram, „Es braukšu mājās ar autobusu vai trolejbusu". Tomēr bieži vien cilvēki kļūdās un šī saikļa vietā lieto saikli „jeb", kurš saista divus vienlīdzīgus teikuma locekļus, kas izteikti ar vienādas nozīmes vārdiem vai vārdu savienojumiem. Korekta lietojuma piemērs būtu: „Esmu gandarīts jeb apmierināts ar savu darbu". Respektīvi, piemēros minētās vārdu kļūdas ar saikļu „jeb" un „vai" lietojumiem ir ievērojami grūtāk atrodamas, nekā ne-vārdu kļūdas, jo nepieciešama teksta sintaktiskā un semantiskā analīze.

Gramatikas kļūdas, līdzīgi, kā vārdu kļūdas, arī ir ļoti sarežģīti atrast. Piemēram, noteikt, kur teikumā ir divdabja teiciens un norādīt, kurās vietās ir jāliek komati. Šeit, tāpat, kā iepriekšējā gadījumā nepieciešama teksta sintaktiskā un semantiskā analīze. Izpētot tīmekļa resursus latviešu valodā, nākas secināt, ka tipiskākās gramatikas kļūdas ir sekojošas:

- vārdu kļūdas
- interpunkcijas kļūdas
- nepareiza vārdu formu lietošana

Piemērs pēdējai tipiskāko kļūdu grupai ir pagātnes un tagadnes formu jaukšana. Piemēram, tagadnes formā nepareizi lietots „dators darbojās", kas patiesībā ir pagātnes forma. Korekti būtu „dators darbojas".

Latviešu valodas aģentūra savā mājaslapā ir apkopojusi biežāk uzdotos jautājumus par gramatikas kļūdām. Šī mājaslapa ietver kļūdu uzskaitījumu ar atbilstošajiem skaidrojumiem, kas sastādīti, pamatojoties uz latviešu valodas gramatikas likumiem [9].

Tomēr kļūdas konstatēšana ir tikai viens no pilnvērtīga pareizrakstības kļūdu pārbaudītāja uzdevumiem. Brīdī, kad pārbaudāmais vārds tiek atzīts par nepareizu, rīka uzdevums ir piedāvāt potenciāli pareizos šī vārda labojumus. Svarīgi, lai šie varianti nebūtu daudzi un lietotājam būtu ērti tajos orientēties. Tas nozīmē, ka šos variantus nepieciešams sakārtot secībā pēc to ticamības.

Minētās darbības aplūkojamas kodā, kurš redzams attēlā 2.1.

```
pareizrakstibasParbaude(vārds v) {
	if (navPareizs( v )) {
		pareizieVarianti = getPareizieVarianti( v )
		piedavatieVarianti = filtretUnSakartot( pareizieVarianti )
		return piedavatieVarianti
	}
	else return true;
}
```

2.1. att. **Pareizrakstības pārbaudes pseidokods**

3 SERVERA PROGRAMMNODROŠINĀJUMS UN KONFIGURĀCIJA

Pētījuma praktiskās daļas realizācija tika veikta uz autora personīgā datora. Galvenais iemesls šādai izvēlei ir tas, ka uz autora personīgā datora ir brīva iespēja instalēt jebkādu programmatūru un tās papildinājumus un dators pēc tā tehniskajiem parametriem atbilst programmatūras prasībām, kuras uz tā paredzēts instalēt, kas nozīmē, ka visi veiktspējas testi būs korekti.

Datoram ir uzstādīta Ubuntu Linux operētājsistēma. Šī sistēma tika izvēlēta tāpēc, ka darba praktiskajā daļā ir paredzēts instalēt PHP programmatūras papildinājumus, kuri ir paredzēti tikai Linux videi. Konkrēti Ubuntu distribūcija tika izvēlēta tāpēc, ka tā ir vispopulārākā Linux distribūcija, kā rezultātā, būtu pieejams liels daudzums papildus literatūras, kura palīdzētu atrisināt problēmas, kuras potenciāli varētu rasties darba gaitā.

Populārākā tīmekļa servera programmatūra Linux vidē ir Apache. Šī iemesla dēļ, tīmeklī ir atrodams liels daudzums dažādas papildus literatūras, kā arī rīku, piemēram, iepriekšminētais tīmekļa programmu izpildes laika testēšanas rīks. Ubuntu vidē Apache programmatūra tiek uzstādīta ar komandu, kas redzama attēlā 3.1.

```
sudo apt-get install apache2
```

3.1. att. **Apache programmatūras uzstādīšana**

Praktiskā daļa tiks programmēta PHP programmēšanas valodā. Šis risinājums ir izvēlēts tāpēc, ka mērķis ir izveidot rīku, kurš ir maksimāli pieejams mājaslapu izstrādātājiem, un tā kā PHP ir viena no izplatītākajām mājaslapu programmēšanas valodām, tika pieņems lēmums izvēlēties šo valodu. PHP instalācija tiek veikta ar komandu, kas redzama attēlā 3.2.

```
sudo apt-get install php5
```

3.2. att. **PHP pakotnes uzstādīšana**

Tā kā tīmekļa serveris ir Apache, nepieciešams uzstādīt arī PHP Apache pakotni, kuras uzstādīšanas komanda redzama attēlā 3.3.

```
sudo apt-get install libapache2-mod-php5
```

3.3. att. **PHP Apache pakotnes uzstādīšana**

Ņemot vērā to, ka rīkam būs nepieciešama pieeja lielam informācijas apjomam – pareizo vārdu vārdnīcai, ir nepieciešama datubāze, kur šos vārdus glabāt. Tā kā vispopulārākā datubāzes programmatūra ir MySQL, tika pieņemts lēmums par labu tai. Papildus faktors, kurš spēlēja par labu šādai izvēlei, ir tas, ka MySQL izmanto tādas tīmekļa autoritātes, kā Youtube, Wikipedia, Facebook u.c. [10]. MySQL pakotnes instalācija tiek veikta izpildot komandu, kas redzama attēlā 3.4.

```
sudo apt-get install mysql-server
```

3.4. att. **MySQL pakotnes uzstādīšana**

Līdz ar to, praktiskajā daļā izmantojamā programmatūras servera tehniskie parametri ir sekojoši:

- Procesors – Intel i3 2,27 GHz
- Citā diska apjoms – 500 GB
- Operatīvā atmiņa – 3 GB
- Operētājsistēma – Ubuntu Linux 12.10
- Tīmekļa serveris - Apache 2.2.16
- Tīmekļa programmatūras valodas atbalsts - PHP 5.3.3
- Datubāzes programmatūra - MySQL 5.1.49

4 PAREIZRAKSTĪBAS RĪKS

Šī darba mērķis ir izpētīt, kā darbojas pareizrakstības rīki un izveidot speciālu tīmekļa videi piemērotu latviešu valodas pareizrakstības rīku. Svarīgi ir saprast galvenās komponentes no kurām sastāv pareizrakstības rīks. Var izdalīt trīs galvenās komponentes:

1. Kļūdu atrašana
2. Kļūdu labojumu ģenerēšana
3. Kļūdu labojumu sakārtošana

Visas trīs komponentes savā starpā ir cieši saistītas, jo vienas komponentes gala rezultāts kalpo par citas komponentes ieejas datiem.

Pirmās komponentes uzdevums ir saprast, vai dotais vārds ir vai nav pareizs latviešu valodas vārds. Komponente ieejā saņem vārdu un izejā atgriež loģisko *true* vai *false* rezultātu.

Gadījumā, ja pirmā komponente paziņo, ka ieejā dotais vārds nav pareizs, tad darbu sāk otrā komponente, kuras uzdevums ir atrast visus potenciālos kļūdu labojumus nepareizajam vārdam. Piemēram, labojumi vārdam „studentz" būtu „students", „studenti", „studenta" utt.

Kad otrā komponente ir beigusi darbu, tā nodod vadību trešajai komponentei. Tās uzdevums ir sašķirot visus iespējamos kļūdu labojumus pēc to ticamības. Ir svarīgi, lai lietotājam, kurš lietos pareizrakstības rīku, būtu ērti labot kļūdas – lai kļūdu labojumu saraksta augšgalā ar vislielāko varbūtību parādās tieši tas vārds, ko lietotājs bija domājis.

Turpmākajā darba tekstā tiks aprakstīta katra no minētajām komponentēm, apskatīti dažādi kļūdu tipi un to atrašanas varianti, kā arī pamatots, kuri no apskatītajiem variantiem reāli tiks izmantoti pareizrakstības rīka izstrādē.

4.1 Kļūdu atrašanas risinājumi

4.1.1 Latviešu valodas specifika

Kā skaidrots Ingunas Skadiņas, Andreja Veisberga, Andreja Vasiļjeva, Tatjanas Gornostajas, Ivetas Keišas un Aldas Rudzītes kopīgi veidotajā grāmatā „Latviešu valoda digitālajā laikmetā", latviešu valodu raksturo šādas pazīmes: „

- Izruna gandrīz pilnībā atbilst rakstībai
- Locījumu dēļ pastāv daudz dažādu gramatisko formu un galotņu

- Liels apjoms atvasinātu vārdu un atvasināšanas veidu
- Brīva vārdu kārtība teikumā
- Gramatiskais un intonatīvais interpunkcijas princips

Latviešu valodā tiek izmantots fonomorfoloģiskais rakstības princips. Latviešu rakstība gandrīz pilnībā atbilst izrunai (skaņas garuma, mīkstinājuma, kā arī šņācošas skaņas apzīmēšanai tiek lietotas diakritiskās zīmes), tādēļ tā tiek uzskatīta par vienu no labākajām ortogrāfijas sistēmām.

Latviešu valodas struktūras dēļ pastāv ļoti bagātīgas vārddarināšanas iespējas. Vārdus visbiežāk darina morfoloģiski - pievienojot vārda celmam afiksus jeb priedēkļus un piedēkļus (vārda sastāvdaļas); retāk jaunus vārdus veido, darinot salikteņus; ir arī citi jaunu vārdu darināšanas paņēmieni.” [4].

4.1.2 Ne-vārdu kļūdu noteikšanas algoritms

Visa pamatā, lai noteiktu, vai dotais vārds ir pareizs vai nav, ir jābūt zināšanu bāzei – visiem pareizajiem vārdiem. Respektīvi, ir vārdu krātuve, kas satur pareizos vārdus. Sauksim šo krātuvi par vārdnīcu. Ideālā gadījumā vārdnīcā vajadzētu būt pilnīgi visiem konkrētajā valodā esošajiem vārdiem. Ja mums ir šāda vārdnīca, tad viss, kas nepieciešams, lai pārliecinātos par dotā vārda korektumu – vārdu pa vārdam jāsalīdzina dotais vārds ar vārdnīcā esošajiem vārdiem un jāmeklē sakritības. Ja vārdnīcā šāds vārds ir, tātad tas ir pareizs. Pseidokods šai metodei redzams attēlā 4.1.2.1.

```
parbaude(vards v){
     visiVardi = read('vardnica.txt');
     foreach(vardnicasVards in visiVardi){
           if(vardnicasVards == v){
                 return true;
           }
     }
     return false;
}
```

4.1.2.1. att. **Ne-vārdu kļūdu atrašanas pseidokods**

Tomēr reāli dzīvē ir ļoti sarežģīti izveidot vārdnīcu, kura ietvertu pilnīgi visas iespējamās vārdformas konkrētajā valodā. Ja tas arī izdotos, šāda vārdnīca būtu milzīga (piemēram, latviešu valodā – būtu nepieciešams vairāk nekā 14 miljonu ierakstu), kas nozīmē to, ka tās apstrāde un pareizo vārdu meklēšana prasītu salīdzinoši lielus resursus.

Eksistē kompaktāks vārdnīcas veids, kurš vienlīdz labi satura ziņā pārklāj iepriekš minēto. Vārdnīca tiek sadalīta vairākās komponentēs – vārda saknē un vārda afiksos - prefiksos un postfiksos. Piemērs redzams tabulā 4.1.2.1.

4.1.2.1. tabula

Vārdnīcas afiksi

Afikss	**Piemērs**
Prefikss	**maz**dēls
Postfikss	futbol**ists**

Protams, ir nepieciešams definēt likumus, pēc kuriem vārda saknei drīkst piemērot konkrētos afiksus. Šis process ir ļoti laikietilpīgs un prasa milzīgu darbu. Šāda metode ļauj krietni samazināt vārdnīcas izmēru, kā rezultātā – tiek palielināta meklēšanas procesa ātrdarbība.

Jāpiebilst, ka šāda vārdnīca regulāri ir jāatjauno ar aktuālajām izmaiņām latviešu valodas gramatikā. Tas ir skaidrojams ar faktu, ka latviešu valodā regulāri tiek veidoti arvien jauni vārdi. Piemēram, terminoloģijas komisijas ieviestie jaunvārdi, kā arī īpašvārdi.

4.1.3 Hunspell

Hunspell ir bezmaksas pareizrakstības pārbaudes rīks un morfoloģijas analizators, kurš paredzēts valodām, kuras ir morfoloģiski sarežģītas. Rīks spēj apstrādāt dažādu kodējumu alfabētus. Sākotnēji tas tika izstrādāts priekš ungāru valodas. Tas ir rakstīts valodā C++. Hunspell ir bāzēts uz MySpell [11] un spēj izmantot MySpell rīka vārdnīcas. Tomēr Hunspell privilēģija ir tā, ka tas spēj strādāt ar UTF-8 kodētām vārdnīcām, kamēr MySpell nepieciešamas *Single Byte Character Set* [12] vārdnīcas.

Hunspell ir ļoti izplatīts – to kā pamatrīku pareizrakstības pārbaudei izmanto šādas programmatūras:

- Apple's Mac OS X 10.6 Snow Leopard un vēlākās versijas
- Eclipse
- Google Chrome
- Mozilla produkti (t.s. Firefox un Thunderbird)
- OpenOffice
- Opera 10+
- u.c.

Lai Hunspell spētu korekti pārbaudīt pareizrakstības kļūdas, tam ir nepieciešamas divas datnes: vārdnīcas datne un afiksu datne. Vārdnīcas datne satur sarakstu ar vārdiem – vienā rindā ir viens vārds. Papildus tam, katram vārdam ir iespējams definēt likumus, pēc kuriem tam tiek piekārtots atbilstošais afikss. Šos likumus definē uzreiz aiz vārda, atdalot vārdu no likumiem ar „/" simbolu. Vārdnīcas datnē pirmā rinda vienmēr būs skaitlis – tas norāda, cik kopā vārdnīcā ir vārdu. Tas ir nepieciešams tehnisku apsvērumu dēļ – lai programma efektīvāk spētu operēt ar atmiņu. Vienas vārdnīcas rindiņas piemērs ir redzams attēlā 4.1.3.1.

```
suns/Pj
```

4.1.3.1. att. **Rindiņas piemērs Hunspell vārdnīcas datnē**

Redzams, ka vārdnīcā ir definēts vārds „suns". Papildus tam, šim vārdam ir definēti divi likumi – „P" un „j".

Afiksu datne satur likumus, no kuriem iespējams uzbūvēt pareizo vārdu variantus. Papildus tam, šajā datnē ir iespējams definēt arī dažādus papildus uzstādījumus, piemēram, teksta kodējumu. Vienā datnes rindiņā tiek definēts viens likums. Vienas afiksu datnes rindiņas piemērs ir redzams attēlā 4.1.3.2.

```
SFX P   s     im     [lns]s
```

4.1.3.2. att. **Rindiņas piemērs Hunspell afiksu datnē**

Skaidrojumi:

„SFX" – likums definē sufiksu

P – likuma identifikators

s – simbols vārda beigās, kurš tiks aizstāds

im – simboli, kuri tiks ievietoti aizstātā simbola vietā

[lns]s – regulārā izteiksme, kas nosaka to, ka likums attiecas uz vārdiem, kuru pēdējais simbols ir „s", bet pirmspēdējais ir „l", „n" vai „s".

Līdz ar to, ja mēs gribētu noskaidrot, vai vārds „sunim" ir pareizs, tad darbības gaita būtu sekojoša:

- vārdnīcā atrodam vārdu „suns"
- redzam, ka šim vārdam ir likums P
- afiksu datnē atrodam likumu P
- redzam, ka regulārā izteiksme ir spēkā

- tā, kā šis ir sufikss (uz to norāda „SFX") aizvietojam pēdējo burtu „s" ar burtiem „im" – „suns" => „sunim"
- secinām, ka vārds ir korekts

Šādā veidā ir uzskatāmi redzams, ka Hunspell lieliski izmanto iepriekš apskatīto pieeju, kurā vārdnīca nebūt nesatur visus konkrētajā valodā iespējamos vārdus, taču tā vietā ir definēti likumi, pēc kuriem algoritms spēj uzbūvēt visas iespējamās pareizo vārdu kombinācijas.

4.2 Pareizrakstības kļūdu atrašana

4.2.1 Izmantojamie dati

Pirmais solis rīka izstrādē ir vārdnīcas sagatavošana. Kā iepriekš tika aprakstīts – izveidot vārdnīcu, kura satur pilnīgi visus iespējamos vārdus visās to formās, ir ļoti sarežģīti un laikietilpīgi. Šī iemesla dēļ tiks izmantots Hunspell piedāvātais vārdnīcas formāts.

Veicot pieejamo tīmekļa resursu izpēti, tika pieņemts lēmums izmantot iepriekš aprakstīto Jāņa Eisaka izstrādāto latviešu valodas pareizrakstības pārbaudes rīka moduli Open Office programmatūrai. Tas ir brīvi pieejams lejupielādei Open Office mājaslapā. Modulis ir pieejams vienā datnē ar paplašinājumu *.oxt*. Izmantojot kādu no datņu arhivatoriem (piemēram, WinRar – Windows vidē, vai gZip – Linux vidē), šo datni iespējams atarhivēt. Tā sastāv no vairākām specifiskām datnēm, kuras ir paredzētas Open office vajadzībām. Visa nepieciešamā informācija atrodas tikai divās datnēs:

- lv_LV.dic (Hunspell vārdnīca, 156215 rindiņas)
- lv_LV.aff (Hunspell vārdu afiksi, 2689 rindiņas)

Turpmākajā pētījuma daļā par pamatu tiks izmantota šajās divās datnēs atrodamā informācija.

4.2.2 Datu konvertēšana no Hunspell formāta uz MySQL

Ņemot vērā to, ka vārdnīcas datne satur ļoti lielu rindiņu skaitu, būtu ļoti neefektīvi ar PHP pie katra pieprasījuma ielasīt visu šīs datnes informāciju atmiņā un pēc tam daudzos programmas ciklos veikt nepieciešamo vārdu meklēšanu. Šī iemesla dēļ tika pieņemts lēmums pārnest datus no Hunspell formāta uz MySQL datubāzi. Rezultātā PHP nevajadzēs apstrādāt tik lielu datu apjomu, jo MySQL paredz iespēju atlasīt noteiktu datu apakškopu, izmantojot

vaicājuma nosacījumus, piemēram, „LIKE” nosacījumu. Piemēram, ja ir nepieciešams pārbaudīt, vai vārds „students” ir pareizs, ir iespēja izmantot „LIKE” nosacījumu, lai pārliecinātos, vai datubāzē ir vārdi, kuri atbilst „stud%” kritērijam. Rezultātā, PHP saņem ievērojami mazāku datu apjomu, un ir nepieciešams mazāks iterāciju skaits. „LIKE” nosacījums bija viens no galvenajiem faktoriem, kāpēc praktiskās daļas ietvaros tika izvēlēta MySQL datubāze, nevis kāda „key-value pair” datubāze, piemēram, Memcached. Problēma ir tā, ka šī datubāzes programmatūra informāciju uzglabā operatīvajā atmiņā un indeksēšana ir realizēta veseliem vārdiem, kā rezultātā nav iespējama datu atlasīšana pēc vārda fragmentiem, kā to var darīt ar LIKE vaicājumu.

MySQL datubāzē tika izveidotas divas tabulas, kuru ER modelis redzams attēlā 4.2.2.1.

4.2.2.1. att. **Datubāzes ER modelis**

Tabula „words” satur informāciju par vārdnīcas vārdiem. Tie ir dati no datnes „lv_LV.dic”. Tabula „rules” satur informāciju par vārdu veidošanas likumiem, jeb afiksus. Tie ir dati no datnes „lv_LV.aff”.

Kā redzams ER modelī, starp šīm abām tabulām nav relāciju. Lai izveidotu relācijas, būtu nepieciešama starptabula, kura saturētu id no tabulas „words” un id no tabulas „rules”. Tomēr no ātrdarbības viedokļa šāda relācija nav vajadzīga.

4.2.3 PHP un MySQL bāzētā rīka izstrāde

Izanalizējot Hunspell datu formātu un iepazīstoties ar tā specifikāciju, tika pieņemts lēmums izveidot pareizrakstības kļūdu noteicēju, kurš spētu izmantot minētos datus un būtu veidots PHP programmēšanas valodā, glabājot datus MySQL datubāzē.

Šis rīks sastāv no vairākām loģiskām daļām:

- Teksta apstrāde – vārdu atrašana
- Pareizo vārdu tiešā meklēšana
- Pareizo vārdu meklēšana, kombinējot variantus

Turpmāk apskatīsim katru no šīm daļām.

4.2.3.1 Vārdu meklēšana

Rīkam jāspēj tikt galā ar dažāda garuma tekstu – jānosaka, kuri vārdi šajā tekstā nav pareizi. Līdz ar to, pirmais uzdevums ir ievadā doto tekstu sadalīt atsevišķos vārdos. Tas nozīmē, ka algoritmam ir jābūt pietiekoši komplicētam, lai tas spētu nodalīt, piemēram, tīmekļa vietņu adreses, e-pasta adreses, telefona numurus, skaitļus, speciālos simbolus utt. no parastiem vārdiem, kuros jāmeklē kļūdas. Šajā praktiskā darba daļā tas ir realizēts ar PHP regulāro izteiksmu palīdzību.

4.2.3.2 Pareizo vārdu tiešā meklēšana

Pārbaudot katru atsevišķo vārdu, meklējot tajā kļūdas, pirmkārt ir nepieciešams pārliecināties, vai vārdnīcā jau neeksistē šī vārda pamatforma. Respektīvi, vai ir iespējams atrast vārdu, nepielietojot vārdu kombināciju ģenerēšanu, vadoties pēc Hunspell vārdu veidošanas likumiem, kas definēti afiksu tabulā. Līdz ar to, pirmajā solī ir jāveic vienkāršs datubāzes vaicājums. Piemēram, ja mums ir jānosaka, vai vārda „students" pamatforma jau eksistē vārdnīcā, tad tas tiek darīts ar datubāzes vaicājumu,kurš redzams attēlā 4.2.3.2.1.

```
SELECT * FROM `words` WHERE `word` = 'students'
```

4.2.3.2.1. att. **Vārda pamatformas atrašanas MySQL vaicājums**

Ja ievaddatos tiek padots teksts, kurš satur vairākus, vārdus, tad būtu ļoti neefektīvi katram vārdam veidot jaunu datubāzes vaicājumu – tas rada lieku slodzi datubāzes serverim. Tā vietā ir jāizmanto „IN" nosacījums. Piemēram, ja ievaddatos ir teksts „students mācās", tad jāizmanto datubāzes vaicājums, kurš redzams attēlā 4.2.3.2.2.

```
SELECT * FROM `words` WHERE `word` IN ('students', 'mācās')
```

4.2.3.2.2. att. **Vairāku vārdu pamatformas atrašanas MySQL vaicājums**

Līdz ar to, šādā veidā ir iespējams zibenīgi noteikt, vai ievaddatos dotie vārdi jau eksistē vārdnīcā. Ja tie eksistē, tad tiek atgriezta pozitīva atbilde. Ja nē, tad nepieciešama potenciāli pareizo vārdu kombināciju ģenerēšana.

4.2.3.3 Pareizo vārdu meklēšana, kombinējot variantus

Ja iepriekšējā punktā aprakstītais risinājums nav devis rezultātus, un vārdnīcā vārds nav atrasts, tad nepieciešams sākt potenciāli pareizo vārdu kombināciju ģenerēšanu, izmantojot afiksu tabulā atrodamos vārdu veidošanas likumus.

Tā kā ir zināms, ka šajā brīdī dotais vārds tā pamatformā nav pareizs, ir nepieciešams izmainīt tā prefiksu vai postfiksu, jeb vārda sākumdaļu vai beigu daļu. Tehniski tas ir realizēts tā, ka, atkarībā no vārda garuma, tiek aprēķināta šī vārda apakšvirkne. Pēc tam, tiek veikts datubāzes vaicājums, lai iegūtu visus potenciālos kandidātus pareizajam vārdam. Piemēram, ievaddatos ir dots vārds „studentam". Iepriekšējā punktā minētā metode rezultātu nedeva, jo datubāzē vārds šādā formā neeksistē. Līdz ar to, nākamajā solī tiek aprēķināta vārda apakšvirkne, pēc kuras tiks meklēti vārdu kandidāti. Apakšvirkne šajā gadījumā būs „stud". Attiecīgi, datubāzes vaicājums redzams attēlā 4.2.3.3.1.

```
SELECT * FROM `words` WHERE `word` LIKE 'stud%'
```

4.2.3.3.1. att. **MySQL vaicājums vārda atrašanai pēc tā apakšvirknes**

Šeit arī uzskatāmi ir redzama MySQL datubāzes priekšrocība attiecībā uz key-value pair datubāzēm. Datubāzes vaicājuma rezultātā tiek iegūts potenciāli pareizo vārdu kandidātu saraksts.

Kā tas ir parādīts datubāzes ER modelī, tabula "words" satur kolonnu „affixes". Tehniskā darba realizācija paredz, ka visi afiksi šajā brīdī jau ir ielasīti atmiņā. Līdz ar to, nepieciešams ciklā iet cauri visiem atrastajiem kandidātiem, skatīties visus šo kandidātu vārdu veidošanas likumus, veidot kombinācijas un salīdzināt tās ar ieejā doto vārdu. Šī algoritma pseidokods ir redzams attēlā 4.2.3.3.2.

```
checkCandidates(vards v){
     subtext = getSubstring(v);
     kandidati = dbQuery('SELECT * FROM `words` WHERE `word` LIKE
     'subtext%');
     foreach(kandidats in kandidati){
           foreach(affix in kandidati[affixes]){
                 $check = checkCombinations(v, kandidats, affix);
                 if(in_array(v, $check)) return $check;
           }
     }
     return false;
}
```

4.2.3.3.2. att. **Vārda pareizības noteikšanas algoritms**

Pseidokodā minētā funkcija „checkCombinations” ģenerē visas iespējamās vārda kombinācijas pēc dotajiem likumiem. Šīs funkcijas pamatā ir vārdu ģenerēšanas algoritms, kurš aprakstīts šī darba punktā 4.1.3 – „Hunspell”.

Ja arī šīs metodes rezultāts nav devis pozitīvu rezultātu, vēl atliek pēdējā darbība. Izanalizējot pieejamo Hunspell vārdnīcu latviešu valodai, ir secināts, ka tā satur tikai vienu prefiksu. Tā ir vārda nolieguma forma „ne”. Tātad atliek pārbaudīt, vai vārda pirmie divi simboli nav „ne”. Ja ir, tad pirmie divi simboli tiek nogriezi nost, un meklēšana sākta ar jauno vārdu. Idejas pseidokods ir redzams attēlā 4.2.3.3.3.

```
if(substr(v, 0, 2) == ‘ne’){
     v = substr(v, 2); // nogriežam nost pirmos 2 simbolus
}
return checkCandidates(v);
```

4.2.3.3.3. att. **Vārda nolieguma formas pārbaude**

Ja arī šī metode nedod pozitīvu rezultātu, tad atliek secināt, ka ievaddatos dotais vārds nav pareizs. PHP + MySQL realizētā pareizrakstības kļūdu noteikšanas rīka pirmkods atrodams darba pielikumā Nr 1 – „PHP + MySQL pareizrakstības kļūdu noteikšanas kods”.

4.2.4 Hunspell un PHP

PHP ir interpretējama valoda, kuras kods katrā tīmekļa servera pieprasījumā tiek kompilēts. Tas nozīmē to, ka mājaslapai neeksistē jau gatavs nokompilēts baitu kods, kuru tīmekļa serveris var uzreiz izpildīt. Tīmekļa serverim PHP kods vispirms ir jānokompilē, jeb jāpārvērš baitu kodā, un tikai pēc tam to var izpildīt. Protams, pastāv dažādi risinājumi, piemēram, APC (Alternative PHP Cache) [13], ar kura palīdzību daļu PHP koda iespējams nokompilēt tā, lai katrā mājaslapas ielādes gadījumā tas nebūtu atkārtoti jākompilē.

Ja mēs šādā griezumā salīdzinām PHP ar C programmēšanas valodu, tad C gadījumā ir citādāk – programma pirms izpildes ir jānokompilē par mašīnkodu un tikai pēc tam to var izpildīt. Ja programma vienreiz ir nokompilēta, tad to atkārtoti darbinot, tā vairs netiek kompilēta. Līdz ar to C variants ir efektīvāks gadījumos, kad ir kritiski svarīga programmatūras ātrdarbība. Ņemot vērā to, ka pareizrakstības kļūdu noteikšanas rīkam ir jāspēj apstrādāt arī lielus teksta apjomus, ir ļoti svarīgi, lai izpildes laiks būtu pēc iespējas mazāks. Šī iemesla dēļ tiks apskatīta iespēja, kā var apvienot PHP mājaslapu un C valodā rakstītu pareizrakstības kļūdu noteicēju.

4.2.4.1 Hunspell PHP papildinājums

PHP ir iespējams pieslēgt dažādus papildinājumus, jeb moduļus. Moduļi ir jau nokompilēts kods, kurš ir gatavs lietošanai. PHP instalācijā ir ietverts liels daudzums dažādu moduļu, kuri noklusētajā instalācijā nemaz nav pieejami. Tos var iespējot, izmainot *php.ini* konfigurācijas datni. Piemēram, *mysql_query* funkcija, kuru PHP valodā izmanto lai izpildītu vaicājumu MySQL datubāzē, patiesībā ir viena no funkcijām, kas ietilpst Mysql PHP papildinājumā.

Tīmekļa resursā Github (https://github.com) ir publiski pieejams Hunspell PHP papildinājuma pirmkods, kura autors ir Sergejs Lomakovs [14]. Šis papildinājums izmanto Hunspell funkcionalitāti un padara pieejamas Hunspell funkcijas, rakstot izsaukumus no PHP. Tas nozīmē, ka tīmekļa serverim nepieciešama libhunspell pakotne, lai PHP papildinājums to spētu izmantot. Respektīvi, šī darba praktiskās daļas ietvaros Ubuntu tīmekļa serverim ir uzstādīta libhunspell pakotne. Šo pašu pakotni izmanto arī, piemēram, Open office, kurš arī ir iekļauts noklusētajā Ubuntu instalācijā.

Lai uzstādītu Github vietnē pieejamo papildinājumu nepieciešams to lejupielādēt un pēc tam izpildīt komandas, kuras redzamas attēlā 4.2.4.1.1.

```
cd /hunspell-pakotne/
phpize
./configure
make
```

4.2.4.1.1. att. **Hunspell pakotnes izveidošana**

Pēc šo komandu izpildīšanas, tekošajā direktorijā tiek izveidota jauna direktorija „modules", kur atrodama jaunā nokompilētā PHP papildinājuma datne „hunspell.so". Tagad atliek šo papildinājumu padarīt redzamu PHP. Lai to izdarītu, nepieciešams šo datni iekopēt PHP papildinājumu direktorijā, kas tiek izdarīts ar komandu, kas redzama attēlā 4.2.4.1.2.

```
cp ./hunspell.so /usr/lib/php5/20060613+lfs/
```

4.2.4.1.2. att. **Hunspell papildinājuma kopēšana PHP papildinājumu direktorijā**

Nākamais solis, kas jāizdara – PHP konfigurācijas datnē php.ini jāiespējo šis papildinājums, pieliekot klāt rindiņu, kas redzama attēlā 4.2.4.1.3.

```
extension=hunspell.so
```

4.2.4.1.3. att. **php.ini datnes izaiņas, lai iespējotu Hunspell**

Visbeidzot nepieciešams restartēt Apache tīmekļa serveri ar komandu, kas redzama attēlā 4.2.4.1.4.

```
/etc/init.d/apache2 restart
```

4.2.4.1.4. att. **Apache tīmekļa servera restartēšana**

4.2.4.2 Hunspell funkciju izmantošana PHP

Tā, kā tagad PHP programmatūrai ir pieslēgts Hunspell modulis, ir iespēja izsaukt Hunspell funkcijas uzreiz no PHP koda. Piemērs ir sekojošs:

Ar *hunspell_create* funkciju tiek inicializēta Hunspell instance, kā arī norādītas vārdnīcas un afiksu datnes. Šīs darbības piemērs ir redzams attēlā 4.2.4.2.1.

```
$hs = hunspell_create('lv_LV.aff', 'lv_LV.dic');
```

4.2.4.2.1. att. **Hunspell instances izveidošana**

Tagad, viss, kas nepieciešams, lai noteiktu, vai vārds ir pareizs – jāizsauc *hunspell_spell* funkcija, kuras izsaukuma piemērs ir redzams attēlā 4.2.4.2.2.

```
hunspell_spell($hs, ’students’);
```

4.2.4.2.2. att. ***hunspell_spell*** **funkcijas izsaukums**

Šīs funkcijas rezultāts būs loģiskais *true*, jo vārds ir pareizs.

Detalizētu PHP + Hunspell demontrācijas piemēru var apskatīt pielikumā Nr 2 – „PHP + Hunspell pareizrakstības kļūdu noteikšanas kods”.

4.2.5 Ātrdarbības testēšana

Tā kā pareizrakstības kļūdu pārbaudes rīks tika izstrādāts divos variantos, izmantojot dažādas tehniskās realizācijas, to ātrdarbība ir atšķirīga. Lai noskaidrotu, cik ļoti tā atšķiras un kurš no izstrādātajiem variantiem (PHP + MySQL vai PHP + Hunspell) ir labāks, tika veikta ātrdarbības testēšana, izmantojot vienu un to pašu testa kopu. Testa kopa sastāv no 3 dažādām datnēm, kuras satur informāciju no apollo.lv jaunumu tekstiem. Katrā datnē informācijas apjoms ir atšķirīgs (pirmajā datnē – 193 vārdi, otrajā – 405, bet trešajā – 1942 vārdi). Testēšana tiek veikta manuāli – salīdzinot laiku pirms vārdu pārbaudes izsaukuma un pēc.

Vispirms tika veikta PHP + MySQL risinājuma testēšana. Ņemot vērā to, ka MySQL savas darbības optimizācijas ietvaros veic kešdarbi, kā rezultātā pirmais vaicājums būs lēnāks attiecībā uz nākamajiem identiski veiktajiem vaicājumiem, testa kopas tiks testētas izmantojot divus paņēmienus – testēšana bez MySQL kešdarbes un testēšana ar MySQL kešdarbi. MySQL serveris tiks restartēts pēc katra testa veikšanas, lai pārliecinātos, ka tas nav saglabājis datus kešatmiņā. Rezultāti apskatāmi tabulā 4.2.5.1.

4.2.5.1. tabula

PHP + MySQL rīka ātrdarbība

Testa datne	MySQL bez kešdarbes (sekundes)	MySQL ar kešdarbi (sekundes)
1	16,787	6,325
2	20,582	2,139
3	66,991	15,153

Kā redzams, MySQL kešdarbe spēj paātrināt rezultātus konkrētajai testu kopai līdz pat 9,6 reizēm. Tiesa, šis rezultāts nav vispārināms. Tas ir attiecināms tikai uz konkrētiem testa datiem. Respektīvi, šajos testa datos vārdu atkārtošanās biežums bija pietiekoši liels, lai sasniegtu šādus rezultātus. Papildus jāpiezīmē tas, ka otrā testa datne tika apstrādāta ātrāk, par spīti tam, ka tajā ir vairāk vārdu. Tas skaidrojams ar to, ka šajā datnē atrodamie vārdi biežāk atkārtojas, kā arī tajos ir mazāk pareizrakstības kļūdu, kā rezultātā sistēma ātrāk spēj noteikt to, ka vārds ir pareizs.

Tagad apskatīsim otru praktiskā risinājuma varianta ātrdarbību. Arī šajā gadījumā vispirms tiks veikts pirmais kontroltests, lai pārliecinātos par to, ka rezultāti nav iekešoti. Rezultāti apskatāmi tabulā 4.2.5.2.

4.2.5.2. tabula

PHP + Hunspell rīka ātrdarbība

Testa datne	Pirmais mērījums (sekundes)	Atkārtoto mērījumu vidējais laiks (sekundes)
1	0,0234	0,0048
2	0,0148	0,0081
3	0,0287	0,0254

Kā redzams, arī šajā gadījumā pirmie mērījumi ir lēnāki, jo programma nav saglabājusi datus kešatmiņā. Efektīvākais ieguvums atkārtoti veiktajiem mērījumiem attiecībā pret pirmo mērījumu ir 1. testu datnei – 4,9 reizes.

Produkcijas vidē faktiski nevar paredzēt, kādi ieejas dati tiks padoti programmai. Šī iemesla dēļ ļoti svarīgs ir tieši pirmais rādījums, kad nekādu datu vēl nav kešatmiņā. Tabulā 4.2.5.3. apskatāmi uzskatāmi rezultāti abu rīka versiju ātrdarbību salīdzinājumā.

4.2.5.3. tabula

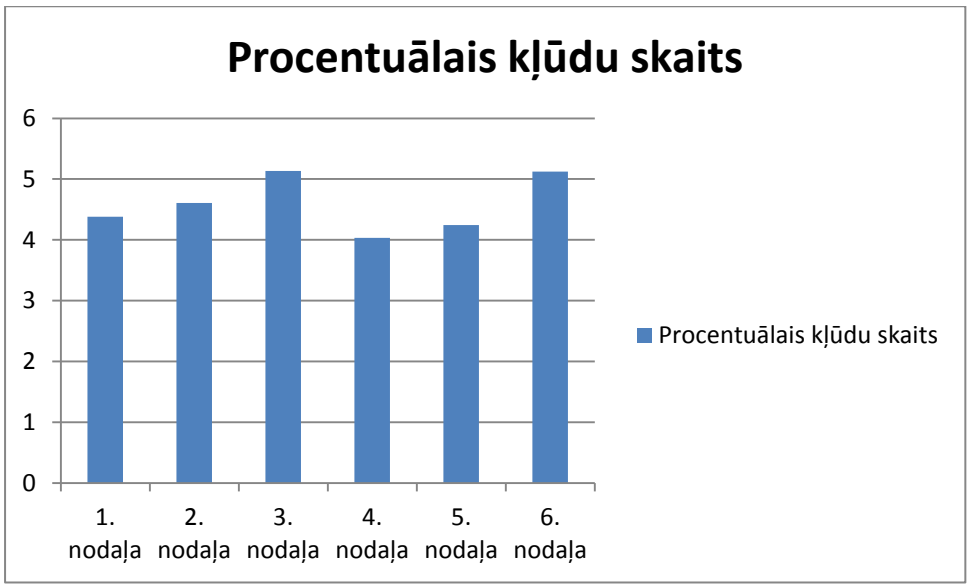
Abu rīka versiju salīdzinājums

Testa datne	PHP + MySQL	PHP + Hunspell	PHP + Hunspell pārsvars
1	16,787	0,0234	717 reizes
2	20,582	0,0148	1390 reizes
3	66,991	0,0287	2334 reizes

Kā redzams, pieaugot teksta apjomam un sarežģītībai, strauji pieaug PHP + MySQL rīka izpildes laiks. Savukārt, PHP + Hunspell rīka variantā izpildes laika pieaugums attiecībā uz testa kopas izmēru ir, tomēr ar, salīdzinājumā ar pirmo rīka variantu ir ļoti niecīgs.

4.2.6 Rīka darbības korektuma novērtējums

Lai novērtētu, cik korekti PHP + Hunspell risinājums tiek galā ar pareizrakstības kļūdu atrašanu, tika izmantots tīmeklī publiski pieejamais Brāļu Kaudzīšu romāns „Mērnieku laiki" [15]. Tika apskatītas pirmās daļas pirmās sešas nodaļas. Uzdevums bija novērtēt, vai rīks spēj tikt galā ar šādu pārbaudītu tekstu, kā arī to, cik pilnīga ir tīmeklī pieejamā latviešu valodas Hunspell vārdnīca. Rezultāti apskatāmi attēlā 4.2.6.1.



4.2.6.1. att. **Procentuālais kļūdu daudzums Mērnieku laikos**

Lai gan pirmajā brīdī šķiet, ka kļūdas procents ir diezgan liels, tomēr pēc nepareizo vārdu apskatīšanas tika izdarīts secinājums, ka tās ir t.s. „false positive" kļūdas, respektīvi, īpašvārdi, kā arī senvārdi, kuri mūsdienās tiek ļoti reti izmantoti, tāpēc vārdnīcā nav iekļauti. Līdz ar to nākas secināt, ka PHP + Hunspell rīks pēc būtības strādā korekti un bez kļūdām, tomēr liela nozīme ir vārdnīcā iekļauto vārdu apjomam.

4.3 Pareizrakstības kļūdu labojumu atrašana

Kad pirmā komponente ir beigusi darbu, tā paziņo, vai dotais vārds ir vai nav bijis pareizs. Gadījumā, ja vārds nav bijis pareizs, darbu sāk otrā komponente, kuras uzdevums ir atrast visus iespējamos pareizrakstības kļūdu labojumus. Šajā etapā nav svarīgi, kādā secībā šie labojumi tiks atrasti. Ar to nodarbosies trešā komponente.

Iemesli, kādēļ cilvēki pieļauj pareizrakstības kļūdas, ir visdažādākie. Cilvēki var kļūdīties, nejauši nospiežot blakus taustiņu uz klaviatūras, steigā rakstot, nospiest taustiņus nepareizā secībā, vai arī gluži vienkārši, nezinot, kā pareizi latviešu valodā jāraksta kādu vārdu (piemēram, „kaut vai" rakstot kā vienu vārdu – „kautvai"). Līdz ar to, pareizrakstības kļūdas var iedalīt trijās kategorijās:

- Tipogrāfiskās (studnts - students)
- Kognitīvās – (diezvai – diez vai)
- Fonētiskās – (dzeinieks - dzejnieks)

Tipogrāfiskās kļūdas rodas gadījumos, kad cilvēks nospiež (vai gluži otrādi - nenospiež) nepareizos taustiņus uz klaviatūras, vai arī nospiež tos nepareizā secībā.

Kognitīvās kļūdas parasti tiek pieļautas tad, kad cilvēkam nav īsti skaidrs, kā konkrētais vārds pareizi ir jāraksta. Respektīvi, zināšanu trūkuma gadījumos. Fonētiskās kļūdas rodas gadījumos, kad cilvēks raksta tekstu precīzi tā, kā tas izklausās. Minētajā piemērā „dzeinieks" nav korekts vārds. Tā ir kļūdaina vārda „dzejnieks" versija, kas radusies pozicionālo skaņu pārmaiņu rezultātā, kad līdzskanis „j" izrunā kļūst par patskani „i". Līdz ar to fonētiskās kļūdas faktiski ir kognitīvo kļūdu apakškopa, tomēr, tā kā šīs kļūdas ir pietiekoši izplatītas un turpmākajā darbā ir aprakstīts algoritms, kurš nodarbojas tieši ar šādu kļūdu atrašanu, šis tips tomēr tiek nodalīts atsevišķi.

4.3.1 Pareizrakstības kļūdu tipi

Analizējot cilvēku pieļautās pareizrakstības kļūdas, var ievērot dažādas likumsakarības, pēc kurām iespējams kļūdas iedalīt dažādos tipos. Konkrēti latviešu valodas gadījumā autors pareizrakstības kļūdas ir iedalījis sešos dažādos tipos:

1. Transpozīcija
2. Izlaists simbols
3. Lieks simbols
4. Substitūcija
5. Translits
6. Divi pareizi vārdi kopā

Tabulā 4.3.1.1. redzami piemēri katram no šiem tipiem.

4.3.1.1. tabula

Pareizrakstības kļūdu tipi

Tips	**Kļūdainais vārds**	**Pareizais vārds**	**Skaidrojums**
Transpozīcija	Stud**ne**ts	Stud**en**ts	Divi blakus esoši burti samainīti vietām
Izlaists simbols	Studnts	Stud**e**nts	Vārdā izlaists burts „e"
Lieks simbols	Stud**f**ents	Students	Aiz burta „d" ir nevajadzīgs burts „f"
Substitūcija	Stude**m**ts	Stude**n**ts	Burta „n" vietā uzrakstīts burts „m"
Translits	Ma**gj**istrs	Ma**ģ**istrs	Burta „ģ" vietā translitā uzrakstīts „gj"
Divi pareizi vārdi kopā	Kautgan	Kaut gan	Kaut gan jāraksta atsevišķi

Turpinājumā tiek piedāvāti nelieli pseidokoda fragmenti katra gadījuma detalizētai izpratnei no tehniskā viedokļa.

Transpozīcijas noteikšanas pseidokods redzams attēlā 4.3.1.1.

```
function transposition(word){
    for(i = 0; i < length(word); i++){
         letter1 = word[i];
         letter2 = word[i+1];
         test = substr(word,0,i) + letter2 + letter1 + substr(word, i+2,
length(word)-i);
         if( checkWord(test) ){
             // Vārdā bija transpozīcijas kļūda
         }
    }
}
```

4.3.1.1. att. **Transpozīcijas noteikšanas pseidokods**

Izlaista simbola noteikšanas pseidokods redzams attēlā 4.3.1.2.

```
function deletion(word){
   alphabet = array(); // latviešu valodas alfabēts
   for(i = 0; i < length(word); i++){
       foreach(letter in alphabet){
          test = substr(word,0,i) + letter + substr(word, i,
length(word)-i);
          if( checkWord(test) ){
             // Vārdā bija izlaista simbola kļūda
          }
       }
   }
}
```

4.3.1.2. att. **Izlaista simbola noteikšanas pseidokods**

Lieks simbola noteikšanas pseidokods redzams attēlā 4.3.1.3.

```
function insertion(word){
    for(i = 0; i < length(word); i++){
         test = substr(word,0,i) + substr(word, i+1, length(word)-i);
         if( checkWord(test) ){
             // Vārdā bija lieka simbola kļūda
         }
    }
}
```

4.3.1.3. att. **Lieka simbola noteikšanas pseidokods**

Substitūcijas noteikšanas pseidokods redzams attēlā 4.3.1.4.

```
function substitution(word){
   alphabet = array(); // latviešu valodas alfabēts
   for(i = 0; i < length(word); i++){
       foreach(letter in alphabet){
          test = substr(word,0,i) + letter + substr(word, i+1,
length(word)-i);
          if( checkWord(test) ){
             // Vārdā bija substitūcijas kļūda
          }
       }
   }
}
```

4.3.1.4. att. **Substitūcijas noteikšanas pseidokods**

Translita noteikšanas pseidokods redzams attēlā 4.3.1.5.

```
function translit(word){
   from = array('aa', 'ch', 'ee', 'gj', 'ii', 'kj', 'lj', 'nj', 'sh',
'uu', 'zh');
   to   = array('ā', 'č', 'ē', 'ģ', 'ī', 'ķ', 'ļ', 'ņ', 'š', 'ū', 'ž');

   foreach(from as item => letters){
      word = str_replace(letters, to[item], word);
   }
   if( checkWord(word) ){
      // Vārdā bija translita kļūda
   }

}
```

4.3.1.5. att. **Translita noteikšanas pseidokods**

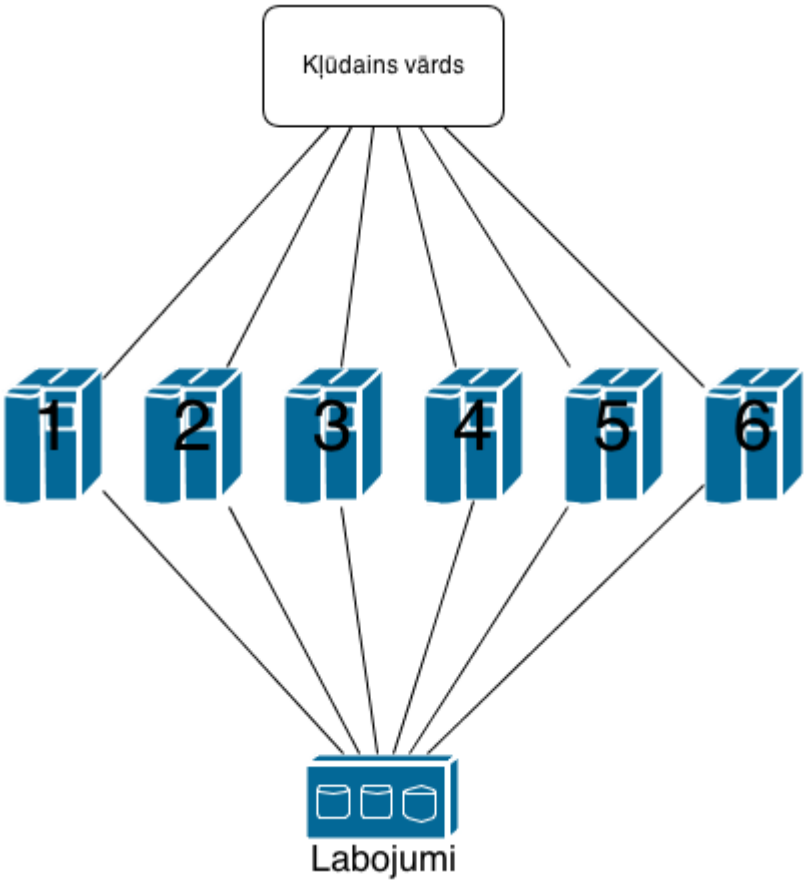
Divu pareizu kopā rakstītu vārdu noteikšanas pseidokods redzams attēlā 4.3.1.6.

```
function twoWords(word){
    if(length(word) <= 3) return false; // pārāk īss vārds
    for(i = 2; i < length(word)-1; i++){
         word1 = substr(word1, 0, i);
         if( checkWord(test) ){
             //Ja pirmais vārds ir pareizs, tikai tad ir jēga pārbaudīt
otru
             word2 = substr(word1, i, length(word) - i);
             if( checkWord(word2) ){
                 // Otrais vārds arī ir pareizs
                 // Tātad 2 pareizi vārdi uzrakstīti kopā
             }
         }
    }
}
```

4.3.1.6. att. **Divu pareizu kopā rakstītu vārdu noteikšanas pseidokods**

Pareizrakstības kļūdu labojumu atrašanas komponentes galvenais uzdevums ir pielietot algoritmus, lai katram no minētajiem sešiem kļūdu tipiem atrastu potenciāli pareizos labojumus. Šis ir ļoti resursietilpīgs process, jo katrā no šo kļūdu tipu algoritmiem tiek ciklā izsaukta funkcija no pirmās komponentes. Respektīvi, tiek pārbaudīti visi iespējamie varianti – vai gadījumā šāds kļūdas labojums nav pareizs.

Tomēr, tā kā otrās komponentes galvenais uzdevums ir sagatavot potenciālos kļūdu labojumus, neņemot vērā to secību, paveras iespēja šos sešus kļūdu noteikšanas algoritmus izpildīt nevis secīgi vienu pēc otra, bet gan paralēli. Šim mērķim ir nepieciešams viens kopējs atmiņas apgabals (*shared memory*), kurā katrs process pieglabātu atrastos kļūdu labojumus. Savukārt, katram algoritmam tiek izveidots savs paralēlais process. Respektīvi, viena procesa vietā, kurš secīgi pēc kārtas izpilda sešus dažādus algoritmus, tiek izveidoti seši procesi, kuri katrs strādā ar savu algoritmu. Šīs metodes shēma atspoguļota attēlā 4.3.1.7.



4.3.1.7. att. **Kļūdu labojumu atrašana ar paralēliem procesiem**

4.3.2 Soundex algoritms

Latviešu valodā ir specifiskas pozicionālās skaņu pārmaiņas. Skaņas izruna ir atkarīga no skaņas atrašanās vietas jeb pozīcijas. Tas atspoguļojas tikai izrunā. Respektīvi, gramatiski

pareizi uzrakstīts teksts atšķiras no tā, kā cilvēki to izrunā. Piemēram, viena no pozicionālajām skaņu pārmaiņām ir līdzskaņu *–ts* saplūšana izrunā, kā rezultātā dzirdams burts *c*. Piemēram, rakstiski pareizi ir „pats", bet izrunā dzirdam „pac".

Šī iemesla dēļ ir jāapsver iespēja, ka cilvēki pieļauj pareizrakstības kļūdas tāpēc, ka raksta precīzi tāpat, kā dzird. Tam var būt dažādi iemesli, piemēram, cilvēks steigā nepamana, ka uzrakstījis tekstu tā, kā tas skan, nevis tā, kā tas ir gramatiski pareizi, vai arī vienkārši nezina, kā konkrētais vārds pareizi ir jāraksta, tāpēc raksta to tā, kā dzird.

Eksistē algoritms, kurš diezgan labi noder šādos gadījumos. Soundex algoritms tika izstrādāts un patenēts 1918. gadā. Tā autori ir Roberts C. Rassels un Margareta K. Odella [16]. Algoritma idejas pamatā ir vārdu kodēšana, līdzīgiem vārdiem piešķirot vienādus kodus. Šis algoritms tika izmantots, piemēram, lai atrastu līdzīgus vārdus telefongrāmatā. Tas noder piemēram tādos gadījumos, kad cilvēks nosauc savu uzvārdu, kurš ir sarežģīts un grūti uzrakstāms. Šādā gadījumā iespējams meklēt vārdus datubāzē pēc to izrunas līdzības.

Eksistē Soundex algoritma varianti dažādās valodās, tomēr visizplatītākais ir tieši angļu valodas variants, jo Soundex tika būvēts speciāli šai valodai. Algoritma ideja angļu valodas vārdiem ir sekojoša.

Katram vārdam atbilstošais Soundex kods sastāv no viena burta un trīs cipariem. Burts vienmēr ir konkrētā vārda pirmais burts, savukārt, cipari tiek veidoti kodējot atlikušos burtus pēc noteikta algoritma. Burti, kuri izklausās līdzīgi, tiek kodēti ar vienādiem cipariem. Piemēram, burti B, F, P un V tiek kodēti ar ciparu 1. Pilnais Soundex algoritms angļu valodas vārdiem ir šāds:

1. Dotajam vārdam atstāj pirmo burtu. Visus *a, e, i, o, u, y, h,* un *w* burtus izdzēš.
2. Visus līdzskaņus vārdā aizstāj ar cipariem pēc sekojošas sistēmas:
   2.1. b, f, p, v => 1
   2.2. c, g, j, k, q, s, x, z => 2
   2.3. d, t => 3
   2.4. l => 4
   2.5. m, n => 5
   2.6. r => 6
3. Ja divi vai vairāki burti ar vienādu ciparu atrodas blakus (pirms 1. soļa), tad jāatstāj tikai pirmais no tiem. Ja starp diviem burtiem ar vienādu ciparu atrodas burti *h* vai *w*, tad tie jākodē tikai par vienu ciparu. Šie noteikumi ir jāņem vērā arī vārda pirmajam burtam.

4. Process jāturpina, kamēr tiek iegūts kods, kas sastāv no viena burta un trīs cipariem. Ja vārdā ir pārāk maz burtu, kā rezultātā nav iespējams izveidot 3 ciparu kodējumu, tad koda beigās trūkstošo ciparu vietā jāliek 0.

Pēc šī algoritma pielietošanas var secināt, ka, piemēram, divi vārdi angļu valodā „Robert" un „Rupert" tiek kodēti vienādi – to kods ir R163, savukārt, piemēram, „Rubin" tiek kodēts kā R150.

Ir veikti vairāki pētījumi par Soundex precizitāti. Piemēram, Alans Staniers 1990. gadā izanalizēja ASV iedzīvotāju uzvārdu datubāzi, kas sastāvēja no 411716 vārdiem un izpētīja, ka, meklējot vārdus ar Soundex algoritmu, tikai 33% no visiem atgrieztajiem rezultātiem bija pareizi. Ir cits pētījums, kuru veica A.J. Laits un B. Randells 1996. gadā, izmantojot uzvārdu datubāzi, kas sastāvēja no 5600 vārdiem. Šie vārdi bija īpaši atlasīti, ņemot vērā to, lai tie sāktos ar visiem alfabēta burtiem, tiem būtu dažādi garumi utt. Pētījums parādīja, ka tikai 36,37% no rezultātiem bija pareizi [17]. Tas nozīmē to, ka Soundex atgriež lielu daudzumu informācijas, bet tikai aptuveni 1/3 daļa ir reāli izmantojama.

Tomēr minētie algoritma nosacījumi īsti neder latviešu valodai. Iepriekš minētajā piemērā vārdi „pats" un „pac" ar noklusēto Soundex algoritmu tiks kodēti divos dažādos veidos. Latviešu valodā jāpievērš uzmanība pozicionālajām skaņu pārmaiņām. Tās ir uzskaitītas tabulā 4.3.2.1.

4.3.2.1. tabula

Pozicionālās skaņu pārmaiņas

Pārmaiņas	**Skaidrojums**	**Piemērs**
z -> s g -> k	Balsīgie līdzskaņi nebalsīgo priekšā kļūst nebalsīgi.	I**z**saukt Drau**g**s
p -> b t -> d	Nebalsīgie līdzskaņi balsīgo priekšā kļūst balsīgi.	A**p**ziņa A**t**bildēt
ts -> c ds -> c	Līdzskaņi *–ts* un *–ds* izrunā saplūst un kļūst par *c*.	Pa**ts** Sir**ds**
žs -> š šs -> š	Līdzskaņi *–žs, -šs* izrunā saplūst un kļūst par *š*.	Me**žs** A**šs**
v -> u	Līdzskanis *v*, ja tas atrodas vienā zilbē ar īsu iepriekšējo patskani, izrunā kļūst par *u*.	Zi**v**s Sa**v**s
j -> i	Līdzskanis *j*, ja tas atrodas vienā zilbē ar īsu iepriekšējo patskani, izrunā kļūst par *i*.	Dze**j**nieks Muze**j**s

Ņemot vērā šo informāciju autors nolēma, ka nav nepieciešamības veidot speciāli latviešu valodai paredzētu Soundex algoritmu tā tiešajā izpratnē. Tā vietā tika izstrādāts algoritms, kurš pēc dotās skaņu pārmaiņu tabulas aizstāj vienus burtus ar citiem, pārbaudot, vai rezultātā neveidojas korekts vārds.

4.3.3 Reālo vārdu kļūdas

Reālo vārdu (jeb vienkārši - vārdu) kļūdas atrast ir ievērojami grūtāk, nekā ne-vārdu kļūdas. Iepriekš aprakstītās metodes lieliski tiek galā ar ne-vārdu kļūdām. Respektīvi, nav problēmu saprast, ka latviešu valodā nav vārda „studnts". Arī potenciāli pareizos kļūdu labojumu variantus atrast nav tik sarežģīti, kā vārdu kļūdu gadījumā. Vārdu kļūdas var būt visdažādākās. Var būt teikums, kurš sastāv no vārdiem, kuri katras atsevišķi ir pareizi, bet neveido jēgu teikumā. Piemēram: „Es citu tev". Šeit vārda „citu" vietā jābūt vārdam „ticu". Mēdz būt gadījumi, kad cilvēki lieto nepareizas vārdu formas. Piemēram: „Bērni, nākat tūlīt mājās!". Šajā gadījumā vārds „nākt" lietots nepareizā formā, jo jālieto pavēles izteiksme „nāciet".

Ja ne-vārdu kļūdu gadījumā pietika salīdzināt doto vārdu ar visiem zināmajiem vārdiem, kas ir vārdnīcā, tad vārdu kļūdu gadījumā tas nav tik vienkārši. Lai atrastu šāda veida kļūdas, nepieciešama konteksta analīze teikumā. Ideālā gadījumā būtu nepieciešams rīks, kurš spēj analizēt katru teikuma vārdu, nosakot vārdšķiras, vārdu locījumus u.tml. Šādā gadījumā būtu nepieciešams glabāt ievērojamu daudzumu papildus datu, kā arī programmas darbība prasītu lielus sistēmas tehniskos resursus.

Tomēr eksistē cits veids, kā var meklēt vārdu kļūdas tekstā. Nav nepieciešams analizēt katru konkrēto vārdu teikumā un meklēt tā atbilstību. Tā vietā var nodefinēt potenciāli kļūdaino vārdu kopu (*confusion set*) – tos vārdus, kurus teorētiski teksta autors varētu būt nepareizi uzrakstījis. Attiecīgi pēc tam programmai, pārbaudot doto tekstu, atliek paskatīties, vai tekstā ir kāds vārds no potencāli kļūdaini vārdu kopas. Un tikai gadījumā, ja šāds vārds ir atrasts, meklēt šim vārdam korektākos labojumus.

Katrai valodai vārdu kļūdas ir diezgan specifiskas. Piemēram, angļu valodā ir ļoti daudz iespēju pieļaut vārdu kļūdas. Kaut vai ar prievārdu (*prepositions*) lietojumiem – „in time", „on time", „at time" utt. Angļu valoda ir specifiska ar to, ka, pieļaujot kaut vai tikai vienu kļūdu – uzrakstot viena burta vietā citu, var iegūt pavisam citu vārdu. Piemēram, „from - form", „theses - these", „met - meat" utt.

Latviešu valodai nav tik daudz iespēju pieļaut vārdu kļūdas. Tomēr arī latviešu valodā var izdalīt dažādas vārdu kļūdu grupas. Vienas no visbiežāk pieļautajām vārdu kļūdām ir vārdu formu nepareiza lietošana. Piemēram, tagadnes un pagātnes formu jaukšana.

Potenciāli kļūdaino vārdu kopas piemērs latviešu valodai ir šāds:

- Nākat, nāciet
- Ejat, ejiet
- Ticu, citu
- Paiet, apiet
- Ka, kad
- utt.

Respektīvi, tas ir grupēts vārdu saraksts. Katrā grupā atrodas vairāki vārdi. Ja pareizrakstības rīks, analizējot tekstu, atrod kādu vārdu no grupas, tad tālāk uzdevums ir saprast, vai dotajā teikumā labāk neiederas kāds cits vārds no šīs grupas.

Minētā metode apraksta pareizrakstības kļūdu labojumu atrašanu. Šos labojumus ir nepieciešams sakārtot pēc to ticamības. Tas ir aprakstīts šī darba turpmākajās nodaļās.

4.4 Kļūdu labojumu kārtošana

Ir divu veidu pareizrakstības rīki – interaktīvie un automātiskie. Interaktīvie pareizrakstības rīki pēc būtības ir vienkāršāki. Tie atrod visus iespējamos kļūdas labojumus, un piedāvā lietotājam sarakstu ar vārdiem. Pēc tam lietotājs pats izvēlas, kuru vārdu viņš grib lietot kļūdainā vārda vietā. Automātiskie pareizrakstības rīki ir ievērojami sarežģītāki, jo to uzdevums ir automātiski atrast vispiemērotāko vārdu no visiem iespējamajiem kļūdu labojumiem un bez lietotāja starpniecības aizstāt kļūdaino vārdu ar visprecīzāko labojumu. Šāda opcija ir iespējama, piemēram, Microsoft Word pareizrakstības rīkam. Tomēr nav iespējams 100% gadījumu atrast vispiemērotāko kļūdas labojumu, tāpēc šī opcija ir iespējama tikai noteiktai vārdu apakškopai [18].

Šī darba mērķis ir iegūt pusautomātisku pareizrakstības rīku. Respektīvi, tādu, kurš lietotājam piedāvā kļūdu labojumus, taču tie ir sakāroti prioritārā secībā, lai lietotājam būtu pēc iespējas vieglāk un ātrāk izlabot kļūdu.

Kad pareizrakstības rīks ir atradis visus iespējamos kļūdas labojumus, nepieciešams tos sakārtot. Ja piedāvāto labojumu skaits ir pārāk liels, tad cilvēkam tajos ir grūti orientēties. No lietošanas ērtuma viedokļa šāds variants ir slikts. Ja tiek ierobežots labojumu skaits, tad rodas

problēma ar to – kur tad ir robeža, kad pārējos labojumus vairs nepiedāvā? Lai pareizrakstības rīku būtu ērti lietot, kļūdu labojumus jāsakārto tā, lai paši pirmie būtu visticamākie labojumi. Kā saprast, kuri labojumi ir visticamākie, un kuri nē? Acīmredzot, ir jābūt sistēmai, pēc kuras katram vārdam var piekārtot ticamības koeficientu, jeb skaitli, kurš raksturo iespējamību tam, ka šis vārds ir tieši tas, kuru cilvēks ir gribējis uzrakstīt. Tomēr šeit ir problēma – kādā veidā katram vārdam aprēķināt šo koeficientu? Tas varētu tikt aprēķināts, piemēram, pēc tā, cik bieži konkrētais vārds ir sastopams literatūrā, vai arī pēc tā, cik ticama ir bijusi lietotāja pieļautā kļūda, ņemot vērā to, ka dotais vārds ir tieši tas, kuru lietotājs bija domājis. Metodes ir dažādas. Turpmākajās nodaļās šīs metodes tiks apskatītas un analizētas.

4.4.1 Minimālā labojumu distance

Iepriekšējās nodaļās tika apskatīti paņēmieni, kurus pielietojot, var izsecināt pareizos vārdus no dotā nepareizā vārda. Minimālās labojumu distances tehnikas pamatā ir vienas simbolu virknes transformēšana par citu, izmantojot pēc iespējas mazāk operāciju. Citiem vārdiem sakot, minimālā labojumu distance starp vārdiem v_1 un v_2 ir mazākais simbola virkņu apstrādes operāciju skaits, kurš ir nepieciešams, lai vārdu v_1 pārveidotu par vārdu v_2. Šī tehnika paredz šādas labojumu operācijas:

- Burta ievietošana
- Burta izņemšana
- Substitūcija (viena burta aizstāšana ar citu)
- Transpozīcija (blakus esošu burtu samainīšana vietām)

Piemēram, lai no nepareizi uzrakstīta vārda „studennts" iegūtu pareizi vārdu „students", nepieciešams veikt vienu operāciju – izdzēst vienu no „n" burtiem. Tātad šim vārdam labojumu distance ir 1. Savukārt, lai no vārda „sdudenz" iegūtu vārdu „students", nepieciešams veikt divus labojumus – pirmo burtu „d" aizstāt ar burtu „t" un pēdējo burtu „z" aizstāt ar burtu „s". Attiecīgi, šī vārda labojumu distance ir 2.

Šo metodi 1965. gadā pirmais izgudroja krievu zinātnieks Vladimirs Levenšteins. Viņam par godu metode tika nosaukta par „Levenšteina distanci". Tiesa gan, šī metode paredzēja tikai 3 no minētajām simbola virkņu apstrādes operācijām – burtu ievietošanu, burtu izņemšanu un substitūciju [19]. Vēlāk Frederiks J. Damerau, kurš bija pazīstams kā dabiskās valodu apstrādes un datizraces pētnieks, šo metodi uzlaboja, ieviešot papildus simbolu virkņu apstrādes operāciju – transpozīciju. Damerau apgalvoja, ka viņa metode ļauj atrast 80% no visām cilvēku pieļautām pareizrakstības kļūdām. Šī metode tika nosaukta par „Damerau –

Levenšteina distanci". Lai gan sākotnēji šī metode bija paredzēta cilvēka pieļauto pareizrakstības kļūdu mērījumiem, to var arī izmantot bioloģijā – DNS variāciju mērījumiem [20].

Atgriežoties pie problēmas ar nepareizā vārda labojumu sakārtošanu, šo metodi lieliski var izmantot, lai noteiktu, kurš no labojumiem ir atbilstošāks. Piemēram, ja ir dots nepareizais vārds „mama" un tiek piedāvāti labojumi: „manta", „mannā" un „mamma", tad minimālā labojumu distance tiek aprēķināta sekojoši:

- Manta – 2
- Mannā – 3
- Mamma – 1

Sakārtojot šos labojumus atbilstošā secībā pēc to labojumu distances, redzams, ka visatbilstošākais labojums vārdam „mama" ir „mamma".

Šīs metodes pseidokods ir redzams attēlā 4.4.1.1.

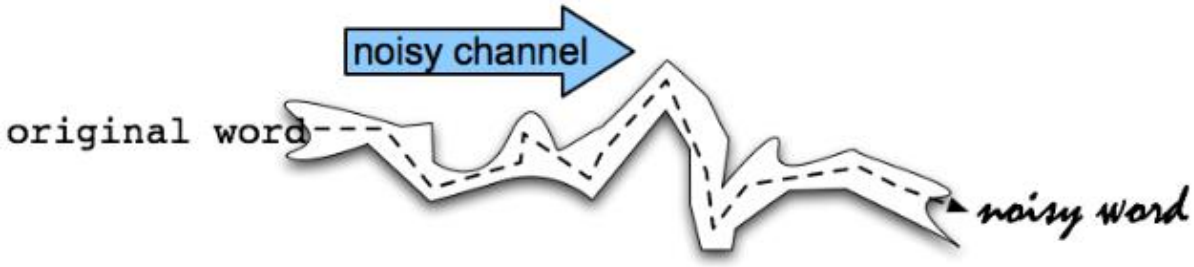
```
function sortByMinEditDistance( mistake, array $candidates ){
     $distances = array()
     foreach($candidates as $candidate){
       $distances[$candidate] = getMinEditDistance(mistake, $candidate);
     }
     return sort($distances);
}
```

4.4.1.1. att. **Labojumu kārtošana pēc minimālās labojumu distances**

4.4.2 Noisy channel modeļa metode

Noisy channel modelis tiek izmantots pareizrakstības kļūdu rīkos, balss atpazīšanas rīkos, kā arī mašīntulkošanā. Tā galvenā ideja ir atrast to vārdu, kuru lietotājs gribēja uzrakstīt, ja ir zināms kļūdainais vārds, kuru lietotājs ir uzrakstījis. Šis modelis tika izstrādāts ap 1990. gadu divās neatkarīgās laboratorijās – IBM un AT&T Bell Labs [21].

Attēlā 4.4.2.1. redzams grafiski uztverams skaidrojums šim modelim [21]. Mēs varam iedomāties, piemēram, situāciju, kad notiek divu cilvēki telefonsaruna, kuras laikā notiek sakaru traucējumi, kā rezultātā otrs cilvēks tikai daļēji spēj saklausīt to, ko pirmais ir teicis. Pareizrakstības kļūdu gadījumā ir līdzīgi. Tikai šiet, atšķirībā no telefonsarunas piemēra, informācijas troksni rada cilvēka pieļautās kļūdas, ievadot tekstu ar klaviatūru. Troksni var radīt nepareizi nospiesti taustiņi uz klaviatūras, kā rezultātā autors ir domājis vienu vārdu, bet tiek uzrakstīts pavisam cits - nepareizs vārds.



4.4.2.1. att. **Noisy channel ideja**

Noisy channel modeļa uzdevums ir, izmantojot rezultātā iegūto trokšņaino vārdu, mēģināt modelēt, kā darbojās kanāls, pa kuru šis vārds ir nācis. Pēc tam pa šo kanālu tiek laisti pareizie vārdi ar mērķi saprast, kurš no tiem galarezultātā būs tāds pats, kā sākotnēji dotais nepareizais vārds. Respektīvi, tiek izveidots dekoderis, kurš ieejā saņem vārdu kopu $\{v_1, v_2, v_3, ..., v_n\}$, modelē šo vārdu ceļu cauri kanālam (*noisy channel*) un skatās, kurš no rezultātā iegūtajiem vārdiem visvairāk līdzinās sākotnējam vārdam.

Matemātiskais šī modeļa skaidrojums apskatāms attēlā 4.4.2.2 [21].

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(w \mid x)$$

$$= \underset{w \in V}{\operatorname{argmax}} \frac{P(x \mid w)P(w)}{P(x)}$$

$$= \underset{w \in V}{\operatorname{argmax}} P(x \mid w)P(w)$$

4.4.2.2. att. **Noisy channel modeļa matemātiskais skaidrojums**

x – pieļautā kļūda

V - vārdnīca

w – vārds, kuru rakstot pieļauta kļūda

ŵ – vārds, kuru cilvēks visticamāk gribējis uzrakstīt

Šajā attēlā redzamā skaidrojuma pārveidošanai tiek pielietots Beisa likums (*Bayes rule*). Rezultātā tiek iegūta izteiksme, kura sastāv no divām daļām:

- P(w) – *Language model*. Norāda iespējamību tam, ka konkrētais vārds varētu tikt lietots.

- P(x|w) – *Error model*. Ja gadījumā būtu lietots vārds w, cik liela iespējamība būtu pieļaut kļūdu x. Šeit faktiski tiek modelēts Noisy channel, kurš pareizo vārdu pārvērš kļūdainajā.

Lai izveidotu abus minētos modeļus – valodas modeli (*language model*) un kļūdu modeli (*error model*), nepieciešams veikt teksta analīzi latviešu valodai. Valodas modelis satur informāciju par vārdu lietojumu biežumiem, savukārt, kļūdu modelis – par pareizrakstības kļūdu iespējamību.

Noisy channel modeļa realizācija paredz unigrammu izmantošanu. Unigrammas ir n-grammu (secīga n elementu virkne tekstā) apakštips, kas sastāv no viena vārda. Pretstatā, piemēram, bigrammas sastāv no diviem vārdiem, trigrammas – no trīs vārdiem utt. Unigrammas satur informāciju par vārdu lietošanas biežumiem konkrētajā valodā. Tas ļauj aprēķināt iespējamību, ar kādu teksta autors varētu būt gribējis šo vārdu uzrakstīt [22]. Angļu valodā eksistē teksta korpuss (Corpus of Contemporary English), kurš satur informāciju par 404 253 213 vārdiem. Šajā korpusā ir atrodams katra vārda lietošanas biežums angļu valodas tekstos [21].

Lai aprēķinātu P(w) varbūtību, jeb iespējamību, ar kādu teksta autors patiesībā ir gribējis uzrakstīt vārdu w, nepieciešams izdalīt teksta korpusā sastopamā vārda w lietošanas biežumu ar kopējo vārdu skaitu korpusā. Ja pareizrakstības kļūdu labojumu modelis kļūdainam angļu valodas vārdam „acress" atrod sekojošus potenciālos labojumus: „actress", „cress", „caress", „access", „across", „acres", tad nepieciešams aprēķināt katra šī vārda iespējamību P(w). To var apskatīt attēlā 4.4.2.3 [21].

word	Frequency of word	P(word)
actress	9,321	.0000230573
cress	220	.0000005442
caress	686	.0000016969
access	37,038	.0000916207
across	120,844	.0002989314
acres	12,874	.0000318463

4.4.2.3. att. **Vārdu lietošanas iespējamības**

Kad ir aprēķināta varbūtība P(w), nepieciešams aprēķināt varbūtību P(x|w). Iepriekš tika minēti vairāki kļūdu pieļaušanas tipi: transpozīcija, substitūcija u.t.t. Kļūdu modelis satur informāciju par šo kļūdu veidu pieļaušanas biežumiem. Tas nozīmē, ka paralēli jau apskatītajam unigrammu modelim dotajā teksta korpusā, pastāv arī kļūdu modelis. Šajā

modelī tiek apkopota informācija atbilstoši katram kļūdas pieļaušanas tipam. To izmantojot, iespējams noteikt, cik liela iespējamība ir pieļaut šo kļūdu. To sauc par „Confusion matrix" jeb pārpratumu matricu. Piemēram, transpozīcijas gadījumā – burts „c" tiek aizvietots ar burtu „a" 6 reizes, burts „d" aizvietots ar burtu „c" 13 reizes utt. Attēlā 4.4.2.4. var apskatīt šādas matricas piemēru angļu valodai [21].

sub[X, Y] = Substitution of X (incorrect) for Y (correct)

X \ Y (correct)	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0

4.4.2.4. att. **Pārpratumu matrica substitūcijas tipam**

Attēlā attēloti nepareizi uzrakstītie burti - X, un pareizie burti - Y. Respektīvi, tās ir kļūdas, kur Y vietā uzrakstīts X.

Kad ir iegūta informācija no kļūdu modeļa, iespējams aprēķināt P(x|w) varbūtību. Šī varbūtība tiek aprēķināta pēc metodes, kura parādīta attēlā 4.4.2.5 [21].

$$P(x|w) = \begin{cases} \frac{\text{del}[w_{i-1}, w_i]}{\text{count}[w_{i-1} w_i]}, & \text{if deletion} \\ \frac{\text{ins}[w_{i-1}, x_i]}{\text{count}[w_{i-1}]}, & \text{if insertion} \\ \frac{\text{sub}[x_i, w_i]}{\text{count}[w_i]}, & \text{if substitution} \\ \frac{\text{trans}[w_i, w_{i+1}]}{\text{count}[w_i w_{i+1}]}, & \text{if transposition} \end{cases}$$

4.4.2.5. att. **P(x|w) varbūtības aprēķināšana**

- Izdzēsta simbola gadījumā dalāmais ir skaitlis, kurš norāda, cik bieži burts *i* ir izlaists pēc burta *i*-1, savukārt, dalītājs ir skaitlis, kurš norāda, cik bieži teksta korpusā kopā parādās burti *i*-1 un *i*.
- Lieka simbola gadījumā dalāmais ir skaitlis, kurš norāda to, cik bieži kļūdainais burts *i* ir ierakstīts pēc burta *i*-1, savukārt, dalītājs ir skaitlis, kurš norāda, cik bieži teksta korpusā parādās burts *i*-1.
- Substitūcijas gadījumā, dalāmais ir skaitlis, kurš norāda, cik bieži burta w_i vietā tiek uzrakstīts kļūdainais burts x_i, savukārt, dalītājs ir skaitlis, kurš norāda, cik bieži teksta korpusā parādās burts w_i.
- Transpozīcijas gadījumā, dalāmais ir skaitlis, kurš norāda, cik bieži burts w_i tiek sajaukts vietām ar burtu w_{i+1}, savukārt, dalītājs ir skaitlis, kurš norāda, cik bieži teksta korpusā kopā parādās burti w_i un w_{i+1}

Ņemot vērā minētos nosacījumus, iespējams aprēķināt kļūdas modeli P(x|w) katram no piedāvātajiem nepareizā vārda labojumiem. „Acress" vārda labojumu gadījumu var aplūkot attēlā 4.4.2.6 [21].

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|word)
actress	t	-	c|ct	.000117
cress	-	a	a|#	.00000144
caress	ca	ac	ac|ca	.00000164
access	c	r	r|c	.000000209
across	o	e	e|o	.0000093
acres	-	s	es|e	.0000321
acres	-	s	ss|s	.0000342

4.4.2.6. att. **Kļūdu modeļa aprēķina piemēri**

Attēlā 4.4.2.7. redzami aprēķini katram no potenciāli pareizajiem kļūdu labojumiem [21]. Gala rezultāts tiek normalizēts, reizinot to ar 10^9, lai tas būtu vizuāli lasāmāks.

Candidate Correction	Correct Letter	Error Letter	x|w	P(x|word)	P(word)	10^9 *P(x|w)P(w)
actress	t	-	c|ct	.000117	.0000231	2.7
cress	-	a	a|#	.00000144	.000000544	.00078
caress	ca	ac	ac|ca	.00000164	.00000170	.0028
access	c	r	r|c	.000000209	.0000916	.019
across	o	e	e|o	.0000093	.000299	2.8
acres	-	s	es|e	.0000321	.0000318	1.0
acres	-	s	ss|s	.0000342	.0000318	1.0

4.4.2.7. att. **Noisy channel modeļa aprēķinu rezultāti**

Līdz ar to varam secināt, ka dotajā gadījumā vārdam „acress" pēc Noisy channel modeļa metodes visatbilstošākais labojums ir „across". Tātad, pareizrakstības rīkam šie labojumi būtu jāpiedāvā secībā: „across", „actres", „acress", „access", „caress" un „cress".

Ja šim pašam gadījumam pielietotu bigrammu modeli, tad būtu iespējams precīzāk noteikt, kurš no piedāvātajiem labojumiem labāk iederas kontekstā. Dans Jurafskis savā prezentācijā par *Noisy channel* min sekojošu piemēru: „a stellar and versatile **acress** whose combination of sass and glamour.." [21]. Tā kā šis bigrammu modeļa piemērs, šiet jāpievērš uzmanība vārdu kombinācijām ar potenciāli pareizo vārdu – vārdiem pirms un pēc tām. Uzdevums ir noskaidrot, kuriem no kļūdu labojumu vārdiem šo kombināciju varbūtība ir vislielākā.

Ņemot vērā „Corpus of Contemporary English" teksta korpusa datus Dans Jurafskis demonstrē sekojošu piemēru.

Vārdam „actress":

- P(actress|versatile) = 0.000021
- P(whose|actress) = 0.0010

Vārdam „across":

- P(across|versatile) = 0.000021
- P(whose|across) = 0.000006

Var redzēt, ka varbūtība pirmajā gadījumā:

P("versatile actress whose") = 0.000021 * 0.0010 = 210 * 10^{-10}

ir lielāka nekā otrā gadījumā:

P("versatile across whose") = 0.000021 * 0.000006 = 1 * 10^{-10}

Līdz ar to varam secināt, ka, izmantojot bigrammu modeli, atbilstošāks vārds būtu „actress", savukārt, izmantojot unigrammu modeli – „across" [21].

4.4.3 Reālo vārdu kļūdu labojumu kārtošana

Iepriekš aprakstīta metode, ar kuras palīdzību iespējams noteikt, vai dotajā teikumā eksistē kāds vārds, kurš pieder potenciāli kļūdaino reālo vārdu kopai. Piemēram, zināms, ka ir kļūdaino vārdu kopa, kas sastāv no vārdiem „nākat, nāciet". Šī kopa paredzēta, lai būtu iespējams atrast reālo vārdu kļūdas, kad tiek nepareizi lietotas vārda „nākt" īstenības un pavēles izteiksmes formas. Attiecīgi, ja ir dots sekojošs teikums: „Bērni, lūdzu, nākat ātri mājās!", tad pareizrakstības rīka uzdevums ir atrast vārdu „nākat", kurš ietilpst potenciāli kļūdaino vārdu kopā {„nākat", „nāciet"} un pēc tam izvēlēties, kurš vārds no šīs kopas labāk iederas dotajā teikumā.

Šajā gadījumā nepieciešams izmantot vārdu n-grammas, līdzīgi, kā tas ir aprakstīts iepriekšējā nodaļā. Viens veids, kā to darīt, ir izmantot vārdu unigrammas. Respektīvi, skatīties, cik bieži latviešu valodas modelī ir sastopami vārdi „nākat" un „nāciet" un attiecīgi izvēlēties to, kurš ir sastopams visbiežāk. Tomēr šāds risinājums ne vienmēr būs korekts. Šī iemesla dēļ labāk ir pielietot bigrammu modeli. Respektīvi, pārbaudīt doto vārdu kopā ar vārdu, kurš teikumā atrodas pirms tā, kā arī to vārdu, kurš teikumā atrodas pēc tā.

Līdz ar to, šajā gadījumā nepieciešams aprēķināt divas sekojošas varbūtības katram no potenciāli kļūdaino vārdu kopas vārdiem:

- P(lūdzu nākat) * P(nākat mājās)
- P(lūdzu nāciet) * P(nāciet mājās)

Attiecīgi, tam vārdam, kuram ir vislielākā varbūtība pēc dotās shēmas, būs lielāka prioritāte.

4.5 Pareizrakstības rīka izstrāde

Apkopojot iepriekš uzkrātās zināšanas, tik pieņemts lēmums uzbūvēt savu pareizrakstības rīku, kurš būtu brīvi pieejams tīmekļa vidē. Iepriekšējās nodaļās aprakstītas visas trīs pareizrakstības rīka komponentes: kļūdu atrašana, labojumu meklēšana un labojumu kārtošana. Apskatot kļūdu atrašanas komponentes dažādos tehniskos risinājumus, tika pieņemts lēmums izmantot PHP Hunspell risinājumu, jeb PHP papildinājumu (extension),

kurš spēj darboties ar Hunspell. Tas ir skaidrojams ar to, ka PHP Hunspell risinājums ātrdarbības testu laikā uzrādīja krietni labākus rezultātus, nekā PHP + MySQL risinājums.

Lai izstrādātu pareizrakstības rīka otro un trešo komponenti, nepieciešams ievākt informāciju par latviešu valodas vārdiem, biežāk pieļautajām kļūdām utt. Respektīvi, ir nepieciešams analizēt lielu teksta apjomu. Šim tekstam ir jābūt aktuālam un mūsdienīgam, lai ievāktie dati maksimāli atspoguļotu cilvēku rakstības stilu internetā mūsdienās. Šī iemesla dēļ, tika analizēti portāla *draugiem.lv* Dienasgrāmatu sadaļas dati. Kā zināms, portāls *draugiem.lv* ir Latvijā populārākais sociālais portāls, kuru ik dienu apmeklē vairāki simti tūkstošu lietotāju. Lietotāji ne vien aplūko, bet arī paši rada jaunu saturu. Cilvēki, kuri šo portālu apmeklē ietilpst visās vecuma grupās un visās izglītības līmeņa kategorijās, līdz ar to ir pamats uzskatīt, ka šī portāla Dienasgrāmatu sadaļas informācija maksimāli precīzi atspoguļo mūsdienu cilvēku rakstības stilu, vārdu lietojumu, kā arī kļūdu pieļaušanas tendences.

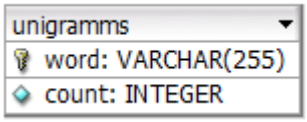
Iepriekš aprakstīti dažādi veidi, ar kuriem iespējams veikt kļūdu labojumu kārtošanu. Ņemot vērā Sarmada Husseina un Tahiras Nasemas prezentācijā atrodamo informāciju, kas ietver dažādu pareizrakstības kļūdu labojumu kārtošanas metožu salīdzinājumu [23], tika pieņemts lēmums izmantot *Noisy Channel* modeļa variantu, jo tā problēmas risināšanas precizitāte ir 99%, salīdzinājumā, piemēram, ar minimālās labojumu distances (minimal edit distance) metodi, kuras precizitāte tiek raksturota tikai 84% .

4.5.1 Valodas modeļa veidošana

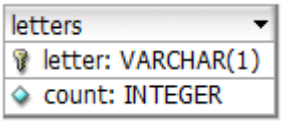
Tā kā darbā tiek izmantots *Noisy channel* modeļa variants, tad ir nepieciešams sagatavot valodas modeli. Lai to izdarītu, tiks izmantota jau iepriekš pieminētā informācija no portāla *draugiem.lv* Dienasgrāmatu sadaļas. Valodas modelim nepieciešama sekojoša informācija:

- Unigrammu (atsevišķu vārdu) lietojumu biežumi
- Burtu lietošanas biežumi
- Divu kopā rakstītu burtu lietošanas biežumi

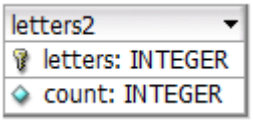
Šī informācija tiek apkopota MySQL datubāžu tabulās. Šo tabulu struktūra apskatāma attēlos 4.5.1.1, 4.5.1.2. un 4.5.1.3.



4.5.1.1. att. ***Unigramms*** **– unigrammu biežumi**



4.5.1.2. att. ***Letters*** **- burtu lietošanas biežumi**



4.5.1.3. att. ***Letters2*** **- divu kopā rakstītu burtu lietošanas biežumi**

Lai minēto informāciju apkopotu, tika uzrakstīts skripts, kurš ielasa *draugiem.lv* Dienasgrāmatas sadaļas datubāzi no MySQL formāta, sadala tekstu vārdos, burtos un divu burtu kombinācijās, un beigās apstrādāto informāciju ieraksta iepriekšminētajās valodas modeļa MySQL tabulās. Rezultātā tika iegūti lietošanas biežumi 35415 unigrammām, visiem latviešu valodas alfabēta burtiem, kā arī 702 dažādiem kopā uzrakstītiem diviem burtiem.

4.5.2 Kļūdu modeļa veidošana

Lai izveidotu kļūdu modeli, vispirms nepieciešams uzprogrammēt risinājumu, kurš spēj atrast kļūdas, kā arī noteikt, kurš kļūdas tips konkrētajā vārdā tika pieļauts. Kļūdu atrašanas risinājums jau ir gatavs un tā darbība ir aprakstīta iepriekšējās nodaļās. Līdz ar to, šajā gadījumā nepieciešams izveidot sistēmu, kura spēj identificēt un uzskaitīt kļūdu tipus. Iepriekš aprakstītā *Noisy channel* modeļa metode paredz to, ka kļūdu modelis satur informāciju par četriem kļūdu tipiem:

- Izlaists simbols
- Lieks simbols
- Substitūcija
- Transpozīcija

Līdz ar to, kļūdu modeļa veidošanas pseidokods izlaista simbola kļūdas tipa gadījumā ir redzams attēlā 4.5.2.1.

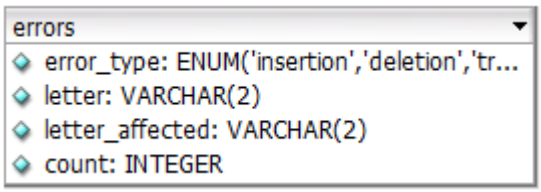
```
$mistakes = findMistakes($text);

foreach($mistakes as $mistake){
    // Izlaisti simboli
    // Atrodam visas kombinācijas jaunajam vārdam
    $possible = findDeletions($mistake);
    foreach($possible as $pword){
        if( isWord($pword) ){ // ja vārds ir pareizs
            writeErrorModel('deletion', $mistake, $pword);
        }
    }
}
```

4.5.2.1. att. **Kļūdu modeļa veidošanas pseidokods izlaista simbola kļūdas tipam**

Pēc līdzīga principa kļūdu modelis tiek veidots arī pārējiem trijiem pareizrakstības kļūdu pieļaušanas tipiem.

Kļūdu modeļa informācija tiek uzglabāta MySQL datubāzes tabulā. Tās struktūra ir apskatāma attēlā 4.5.2.2.



4.5.2.2. att. **Kļūdu modeļa datubāzes tabula**

Šīs tabulas detalizēts skaidrojums apskatāms tabulā 4.5.2.1.

4.5.2.1. tabula

Kļūdu modeļa datubāzes tabulas skaidrojums

Kolonna	Tips	Skaidrojums
error_type	ENUM('insertion', 'deletion', 'transposition', 'substitution')	Apzīmē vienu no četriem kļūdas tipiem
letter	VARCHAR(2)	Burts(i), kurš izraisīja kļūdu. *Insertion* gadījumā – lieki uzrakstītais burts *Deletion* gadījumā – burts, kurš tika izlaists *Substitution* gadījumā – burts, kurš tika uzrakstīts pareizā burta vietā *Transposition* gadījumā – nepareizi uzrakstītā divu burtu kombinācija
letter_affected	VARCHAR(2)	Burts(i), kuri tika ietekmēti, pieļaujot kļūdu. *Insertion* gadījumā – burts, pirms kura uzrakstīts liekais burts.

		Deletion gadījumā – burts, pirms kura tika izlaists nepieciešamais burts. *Substitution* gadījumā – burts, kura vietā tika uzrakstīts kļūdainais burts. *Transposition* gadījumā – pareizā divu burtu kombinācija, kuras vietā tika uzrakstīta nepareizā.
count	INTEGER	Cik bieži teksta korpusā šāds kļūdas tips šiem konkrētajiem burtiem ir sastopams

Rezultātā tika iegūts kļūdu modelis, kurš satur informāciju par 1387 dažādiem kļūdu veidiem.

4.5.3 Rīka integrēšana mājaslapā

Iepriekš tika minēts, ka pareizrakstības rīks būtu ļoti noderīgs dažādiem mājaslapās atrodamiem teksta ievadlaukiem. Tie ir, piemēram, jaunas ziņas pievienošanas lauks administrācijas režīmā, vai arī jauna komentāra pievienošanas lauks. Tā kā šie lauki var būt daudz un dažādi, ir nepieciešams izveidot rīku, kuru būtu iespējams pielāgot jebkuram no šiem teksta laukiem. Rīkam jābūt universālam, jo pretējā gadījumā būtu nepieciešams katram teksta ievadlaukam programmēt rīka savietojamību, kas būtu ļoti nepraktiski un sarežģīti. Šī iemesla dēļ tika pieņemts lēmums rīka būvēšanā izmantot AJAX tehnoloģiju [24] – rīka integrēšana mājaslapā notiek ar Javascript palīdzību, savukārt, visi servera puses pieprasījumi tiek veikti ar AJAX tehnoloģiju.

Vispirms nepieciešams mājaslapas kodā ievietot trīs rindiņas, ar kuru palīdzību tiek ielādēts pareizrakstības rīka kods. Tas ir redzams attēlā 4.5.3.1.

```
<link rel="stylesheet" type="text/css" href="spellchecker.css" />

<script type="text/javascript"
src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"
></script>

<script type="text/javascript" src="spellchecker.js"></script>
```

4.5.3.1. att. **Pareizrakstības rīka ielādēšanas HTML kods**

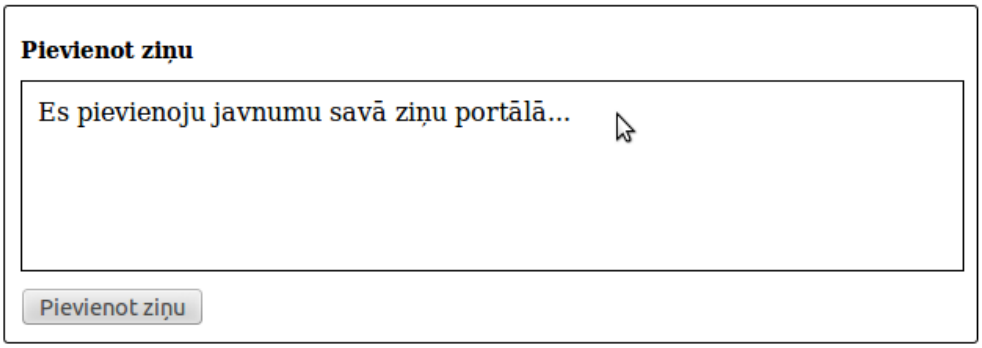
Nākošais solis ir pareizrakstības rīka piesaistīšana konkrētiem ievadlaukiem. Tā, kā ievadlauki var būt dažādi, nepieciešams universāls veids, kā tiem piesaistīt pareizrakstības rīku. Tas tiek demonstrēts attēlā 4.5.3.2.

```
<textarea id="ievadlauks"></textarea>

<script type="text/javascript">
   $(document).ready(function(){
       SP.bind( $('#ievadlauks') );
   });
</script>
```

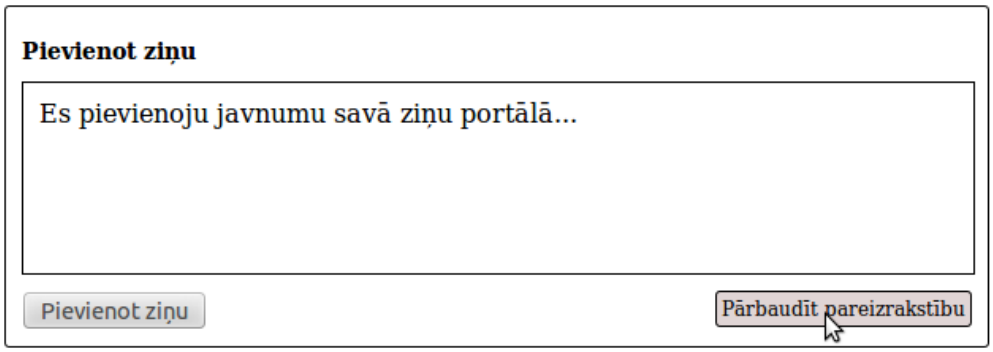
4.5.3.2. att. **Pareizrakstības rīka piesaistīšana teksta ievades laukam**

Šo piemēru uzskatāmi demonstrē nākoši divi attēli. Attēlā 4.5.3.3. redzams parasts teksta ievadlauks, pirms tam ir piesaistīts pareizrakstības pārbaudes rīks.



4.5.3.3. att. **Teksta ievadlauks**

Nākošajā attēlā 4.5.3.4. redzams šis pats ievadlauks pēc tam, kad ir izpildīta javascript komanda, kura teksta laukam piesaista pareizrakstības pārbaudes rīku.



4.5.3.4. att **Teksta ievadlauks ar piesaistītu pareizrakstības pārbaudes rīku**

Attēlā ir redzams, ka ir parādījusies poga, ar kuras palīdzību ir iespējams pārbaudīt, vai konkrētajā teksta laukā ir kāda pareizrakstības kļūda.

4.5.4 Kļūdu atrašana

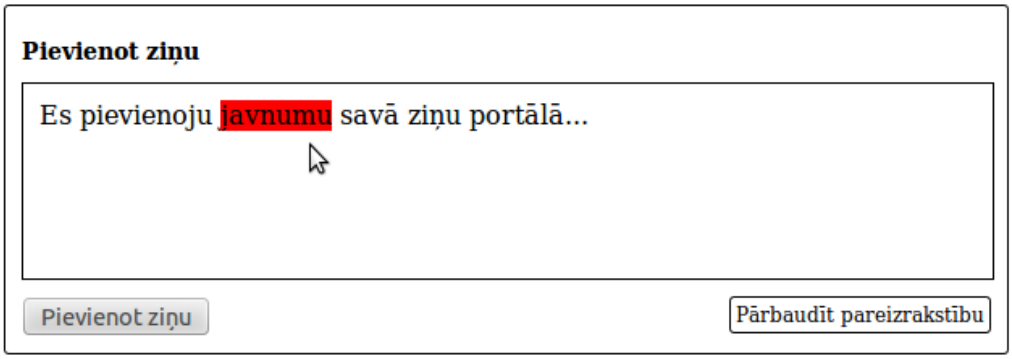
Pēc tam, kad lietotājs ir uzrakstījis tekstu un nospiedis pogu „Pārbaudīt pareizrakstību", tiek izsaukta javascript funkcija, kura, izmantojot AJAX tehnoloģiju, nosūta

pieprasījumu uz serveri. Šis pieprasījums satur konkrētā ievadlauka datus. Servera pusē notiek informācijas apstrāde sekojošā secībā:

1. Teksta sadalīšana vārdos
2. Katra vārda pārbaudīšana
3. Nekorekto vārdu masīva atgriešana

Konkrētajā piemērā redzams, ka teksts satur vienu nepareizu vārdu – „javnumu". Servera pusē tiek apstrādāts viss dotais teksts un secināts, ka šāds vārds latviešu valodas vārdnīcā neeksistē. Tāpēc serveris atbildē atgriež masīvu, kurš satur vienu elementu – šo kļūdaino vārdu.

Nākošais solis pēc tam, kad ir saņemta atbilde no servera – kļūdaino vārdu iezīmēšana. Attēlā 4.5.4.1. redzams, kā pareizrakstības pārbaudes rīks ir iezīmējis vārdu, kurš nav pareizs.



4.5.4.1. att. **Kļūdaino vārdu iezīmēšana**

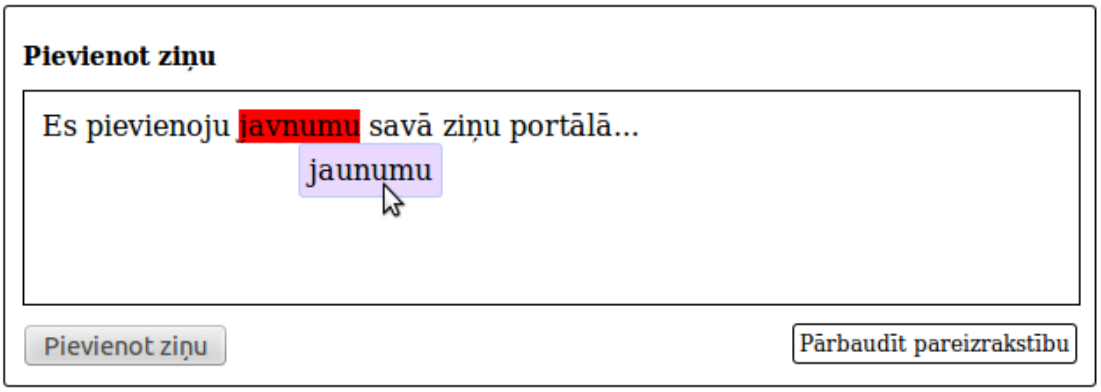
Šādā veidā uzreiz tiek piesaistīta lietotāja uzmanība un lietotājs tiek mudināts kļūdaini uzrakstīto vārdu labot.

4.5.5 Kļūdu labojumu piedāvāšana

Pēc tam, kad ir iezīmēti visi nepareizie vārdi, lietotājam tiek piedāvāta iespēja šos vārdus izlabot. Brīdī, kad lietotājs noklikšķina un iezīmētā kļūdainā vārda, tiek veiktas sekojošas darbības:

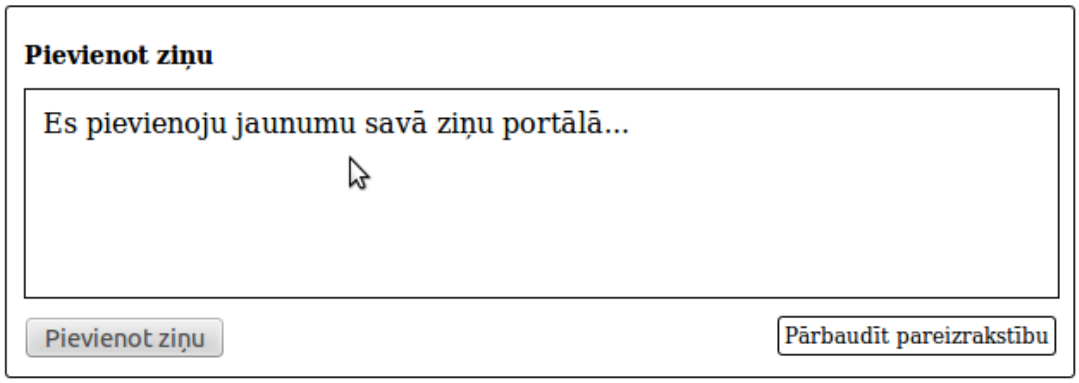
1. Pieprasījuma sūtīšana uz serveri. Nosūta kļūdaino vārdu.
2. Servera pusē tiek sameklēti visi potenciālie kļūdu labojumi.
3. Atrastie kļūdu labojumi tiek sakārtoti pēc to ticamības.
4. Sakārtotie kļūdu labojumi tiek atgriezti un apstrādāti ar javascript.

Šajā konkrētajā gadījumā vārdam „javnumu” ir atrasts tikai viens kļūdas labojums – „jaunumu”. To var redzēt attēlā 4.5.5.1.



4.5.5.1. att. **Kļūdu labojumu piedāvāšana**

Pēc tam, kad lietotājs izvēlas kādu no piedāvātajiem kļūdu labojumiem, kļūdainais vārds automātiski tiek aizstāts ar pareizo. Attēlā 4.5.5.2. var redzēt, kā pareizrakstības rīks nepareizā vārda vietā ir ievietojis pareizo vārdu.



4.5.5.2. att. **Kļūdas izlabošana**

4.5.6 Ātrdarbības testēšana

Darba sākumā tika testēta ātrdarbība pareizrakstības kļūdu atrašanai. Tas tika darīts ar mērķi izvēlēties, kurš no iespējamajiem risinājumiem ir visefektīvākais. Šo testu rezultāti bija pamats turpmāko pētījumu un izstrādes gaitai. Šajā apakšnodaļā tiks veikta ātrdarbības testēšana pareizrakstības rīka prototipa gala variantam, kur pareizrakstības kļūdu labojumi tiek atrasti izmantojot Hunspell mehānismu, savukārt, labojumu kārtošanas nodrošināšanai papildus tiek izmantots MySQL datubāzes risinājums. Šī pētījuma mērķis ir noskaidrot, cik daudz laika procentuāli aizņemt pareizrakstības labojumu sakārtošana pēc tam, kad ir atrasts viss potenciālo labojumu nesakārtotais saraksts.

Pētījuma laikā tiks veikti testi katram no iepriekšminētajiem gadījumiem ar diviem dažādiem datu apjomiem – sākotnēji ar mazu, bet pēc tam ar lielu datu apjomu. Tabulā 4.5.6.1. var aplūkot ātrdarbības testu rezultātus pareizrakstības labojumu atrašanai un sakārtošanai, strādājot ar maziem datu apjomiem (kopā pareizrakstības modulim tika padoti 895 vārdi, no kuriem 75 bija nepareizi – tiem bija nepieciešams meklēt labojumus).

4.5.6.1. tabula

Ātrdarbības mērījumi maziem datu apjomiem

Testa numurs	**Laiks (s) – labojumu atrašana**	**Laiks (s) – labojumu atrašana un sakārtošana**
1	0,5393	0,6147
2	0,5307	0,5987
3	0,5301	0,607
Vidējais laiks	0,5334	0,6068

Līdz ar to varam secināt, ka pie maziem apjomiem, pareizrakstības rīks patērē par 13,7% vairāk laika, lai atrastos kļūdu labojumus sakārtotu pareizā secībā, nevis atgrieztu nesakārtotus.

Savukārt, tabulā 4.5.6.2 var aplūkot šādus pašus mērījumus lieliem datu apjomiem – kopējais vārdu skaits 20828, no kuriem nepareizi ir 2722 vārdi.

4.5.6.2. tabula

Ātrdarbības mērījumi lieliem datu apjomiem

Testa numurs	**Laiks (s) – labojumu atrašana**	**Laiks (s) – labojumu atrašana un sakārtošana**
1	17,5716	19,1501
2	17,5163	19,6912
3	19,914	19,5073
Vidējais laiks	17,6673	19,4495

Šajā gadījumā varam secināt, ka pie lieliem apjomiem, pareizrakstības rīks patērē par 10,08% vairāk laika, lai atrastos kļūdu labojumus sakārtotu pareizā secībā, nevis atgrieztu nesakārtotus.

4.6 Pareizrakstības rīka API

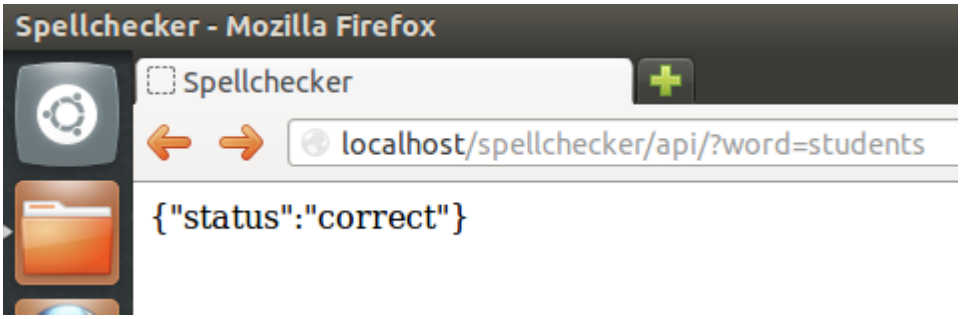
Ņemot vērā to, ka rīka iespēju potenciāls neaprobežojas tikai ar to, ka šo rīku ir iespējams integrēt mājaslapā, tika pieņemts lēmums izveidot API mehānismu, ar kura palīdzību būtu iespējams krietni paplašināt rīka iespējas. API būtu publiski pieejams un to brīvi jebkurš varētu izmantot savām vajadzībām. Dotajā brīdi ir apzināti sekojoši virzieni, kuros būtu iespējams izmantot pareizrakstības rīka potenciālu:

- Iespēja integrēt rīku mājaslapās, neuzstādot to uz sava servera, bet gan veicot attālinātus pieprasījumus API serverim
- Izmantot rīku dažādās darbvirsmas lietotnēs
- Izveidot pārlūkprogrammu spraudņus, kuri ir balstīti uz šo API
- Izmantot šo API mobilajās ierīcēs
- Teorētiski šāds rīka API varētu noderēt dažādiem pētījumiem par pareizrakstības kļūdām latviešu valodā

Pašlaik ir izstrādāts API mehānisma prototips, kurš darbojas pēc sekojoša principa:

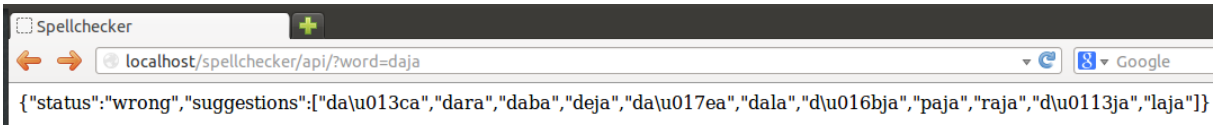
- Tiek veikts HTTP GET pieprasījums uz pareizrakstības rīka serveri. Pieprasījumā tiek norādīts vārds, kuru nepieciešams pārbaudīt.
- Serveris analizē vārdu
    - Ja vārds ir pareizs, tiek atgriezts rezultāts – {*status: correct}*
    - Ja vārds nav pareizs, tiek atgriezts rezultāts – {*status: wrong, suggestions: [* ...]}, kur *suggestions* ir nepareizā vārda potenicālie labojumi, sakārtoti pēc to ticamības

Turpmākajos divos attēlos parādīts piemērs, kā notiek pareizrakstības rīka API izmantošana. Attēlā 4.6.1. redzams, kā tiek veikts HTTP GET pieprasījums uz API serveri, norādot vārdu, kurš ir pareizs.



4.6.1. att. **API pieprasījums ar pareizu vārdu**

Savukārt, attēlā 4.6.2. redzams, kā tiek veikts HTTP GET pieprasījums uz API serveri, norādot vārdu, kurš nav pareizs.



4.6.2. att. **API pieprasījums ar nepareizu vārdu**

Kā redzams šajā attēlā, vārds „daja" nav pareizs. Līdz ar to, tiek atgriezts rezultāts *{status: wrong},* kā arī tiek piedāvāti sakārtoti potenciālie kļūdu labojumi – *suggestions*. Serveris datus atgriež JSON formātā.

Turpmākajās apakšnodaļās tiks detalizētāk apskatītas iepriekš uzskaitītās potenciālās pareizrakstības rīka API izmantošanas iespējas.

4.6.1 Pareizrakstības rīka integrēšana mājaslapā, izmantojot API

Gadījumā, ja mājaslapa ir liela – to regulāri apmeklē daudzi apmeklētāji, kā rezultātā mājaslapai ir palielināta noslodze, tad, visticamākais, mājaslapas veidotāji būs ieinteresēti pareizrakstības rīku integrēt savā serverī, nevis izmantot API risinājumu. Tas ir skaidrojams ar to, ka, gadījumā, ja tiek izmantots API risinājums, tad mājaslapa, kurā tas tiek integrēts, nevar lietotājam garantēt vienmērīgu pakalpojuma pieejamību bez aizturēm. Ņemot vērā to, ka tas būtu papildus pieprasījums uz citu serveri tīmeklī, jārēķinās ar potenciālām aizturēm.

Tomēr, gadījumā, ja mājaslapa nav tik liela un tās apmeklētāju skaits ir mērāms, piemēram, dažos tūkstošos apmeklētāju dienā, tad vieglāk ir izmantot API risinājumu. Šādā gadījumā mājaslapas programmētājam pašam nav jānodarbojas ar Hunspell un MySQL datubāzes (valodas un kļūdu modeļu) uzstādīšanu, bet atliek tikai veikt HTTP GET pieprasījumu pareizrakstības rīka API serverim. Šis variants ir daudz vienkāršāks – tā realizēšanai nepieciešams mazāk zināšanu un laika resursu.

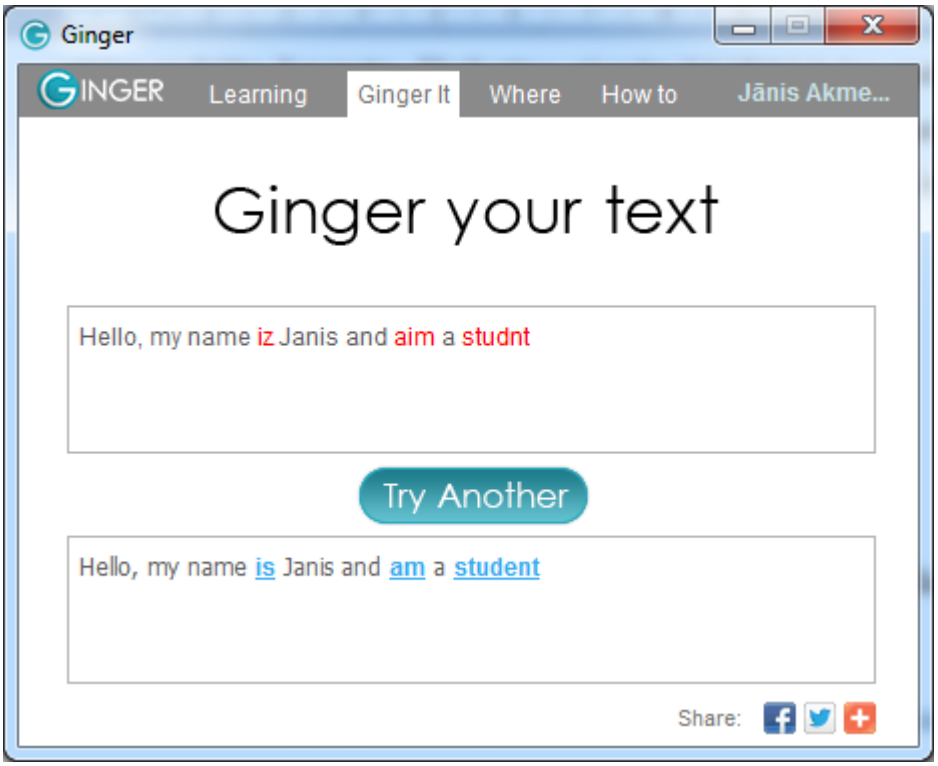
Papildus tam, eksistē arī citi veidi, kā mājaslapas izstrādātājs var izmantot API. Piemēram, ja mājaslapā ir meklēšanas bloks. Gadījumā, ja lietotājs ievada kļūdainu frāzi, meklēšanas rezultātos, visticamāk, neparādīsies tas, ko lietotājs vēlas sagaidīt. Šajā gadījumā mājaslapas izstrādātājs ērti var izveidot risinājumu, kurš lietotājam piedāvā kļūdaino vārdu izlabot, lai iegūtu pareizus meklēšanas rezultātus. Pēc līdzīga principa darbojas Google „did you mean" mehānisms.

4.6.2 Pareizrakstības rīka API izmantošana darbvirsmas lietotnēs

Ikdienā mēs strādājam ar ļoti daudzām darbvirsmas lietotnēm, kuras nodrošina lietotāja teksta ievadīšanu. Pareizrakstības rīka API varētu izmantot gan jaunu lietotņu veidošanā, gan arī jau esošajām lietotnēm. Uzskatāms piemērs šai idejai ir jau esošs līdzīgs risinājums angļu valodai – Ginger (http://www.gingersoftware.com). Šis rīks izmanto API serveri, kuram tiek sūtīti pieprasījumi un iegūtas atbildes par teksta korektumu. Rīks ir savietojams ar sekojošām lietotnēm:

- Microsoft Office Word
- Microsoft Office Outlook
- Microsoft Office PowerPoint
- Internet Explorer
- Firefox

Bez tam, šo rīku var lietot arī kā autonomu darbvirsmas lietotni, ar kuras palīdzību iespējams pārbaudīt teksta pareizrakstību. Rīka demonstrācija ir apskatāma attēlā 4.6.2.1.



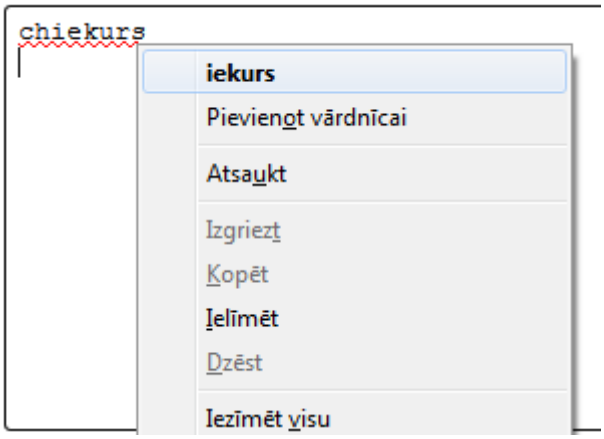
4.6.2.1. att. **Ginger lietotnes demonstrācija**

Kā jau tika minēts, šī lietotne sūta pieprasījumus uz API serveri, tāpēc, gadījumā, ja lietotājam nav pieejams interneta pieslēgums, pakalpojumu lietot nebūs iespējams.

Pakalpojums ir pieejams tikai angļu valodai, tāpēc autora piedāvātais API risinājums latviešu valodai būtu izmantojams līdzīgu rīku radīšanai.

4.6.3 Pārlūkprogrammu pareizrakstības spraudņu savietošana ar API

Darba sākumā minēts, ka eksistē spraudņi, kuri nodrošina pareizrakstības pārbaudes funkcionalitāti latviešu valodai. Šie spraudņi izmanto Hunspell risinājumu, kā arī iepriekš aprakstīto vārdnīcu, kuru autors izmanto arī savam risinājumam. Tomēr, veicot padziļinātus pētījumus, piemēram, Firefox spraudņa darbībā, atklājas, ka tas nespēj tik labi piedāvāt potenciālos kļūdu labojumus, kā autora piedāvātais risinājums. Konkrēti šajā gadījumā netiek ņemta vērā translita problēma latviešu valodai. Piemērs apskatāms attēlā 4.6.3.1.



4.6.3.1. att. **Firefox spraudņa problēmas**

Kā redzams attēlā, spraudnis nepiedāvā vārdu „chiekurs" izlabot uz „čiekurs". Šī iemesla dēļ, autora piedāvātais variants būtu potenciāls risinājums šādu spraudņu uzlabotajām versijām.

4.6.4 Pareizrakstības rīka API izmantošana mobilajās ierīcēs

Mobilo ierīču izplatībai tirgū ir tendence arvien pieaugt. Tiek ražoti arvien jauni mobilie telefoni un planšetdatori. Tomēr nav universāla mehānisma, kā šajās ierīcēs atrodamajām lietotnēm varētu pieslēgt latviešu valodas pareizrakstības rīku. Autora piedāvātais risinājums spētu lieliski tikt galā ar šādām problēmām. Par piemēru ņemot *draugiem.lv* mobilās versijas lietotni. Tā paredz iespēju gan rakstīt lietotājiem vēstules, gan pievienot ierakstus savā dienasgrāmatā utt. Tā kā lietotne pēc būtības ir paredzēta darbam

internetā, tad ir skaidrs, ka tās lietošanai ir nepieciešams interneta pieslēgums. Attiecīgi, viss, kas nepieciešams – lietotnes izstrādātājam uz sava servera ir jāizvieto API modulis, savukārt, pašā aplikācijā pēc tam brīvi jebkurā skatā var iestrādāt pareizrakstības pārbaudes rīka interfeisu, kurš veic informācijas apmaiņu ar API serveri.

4.6.5 Pareizrakstības rīka API izmantošana dažādu pētījumu veikšanai

Ņemot vērā to, ka pašlaik tīmekļa vidē neeksistē bezmaksas risinājums, kurš būtu ērti un universāli izmantojams latviešu valodas pareizrakstības pārbaudīšanai, ir diezgan sarežģīti veikt dažādus pētījumus par latviešu valodas pareizrakstību. Izmantojot autora piedāvāto API risinājumu, šādus pētījumus būtu iespējams veikt. Bez tam, ja pārbaudāmais vārds nav pareizs, teorētiski pastāv iespēja ne tikai atgriezt tā statusu un potenciālos kļūdu labojumus, bet arī pieļautās kļūdas veidu – transpozīcija, substitūcija u.tml.

5 SECINĀJUMI

Internetā ir pieejams milzīgs daudzums informācijas latviešu valodā. Tomēr itin bieži nākas secināt, ka latviešu valodā pieejamā tekstuālā informācija satur daudzas pareizrakstības kļūdas. Analizējot tīmekļa ziņu portālus, tika secināts, ka pašos jaunumu rakstos arī eksistē pareizrakstības kļūdas, tomēr kļūdu apjoms ir daudz reižu mazāks, nekā lietotāju komentāru tekstā. Šāds kļūdains teksts ir grūti lasāms un tā izprašana prasa iedziļināšanos.

Darba mērķi bija izpētīt esošās latviešu valodas pareizrakstības rīku izmantošanas iespējas un izstrādāt jauna rīka prototipu, kuru varētu brīvi integrēt tīmekļa vietnēs, kā arī attīstīt plašāk, izmantojot API funkcionalitāti.

Darbā tika veikts esošo pareizrakstības pārbaudes rīku pētījums, kura ietvaros tika apskatīti dažādi risinājumi: Tildes Birojs, tīmekļa pārlūkprogrammu papildinājumi, kā arī Open office pareizrakstības papildinājumi. Tomēr atklājās, ka nav ērta bezmaksas veida, ar kura palīdzību būtu iespējams integrēt pareizrakstības rīku tīmekļa vietnēs. Bez tam, Open office pareizrakstības rīka kļūdu labojumu piedāvāšanas iespējas reālos testos nedeva vēlamo rezultātu.

Tālāk tika sākts darbs pie jauna rīka prototipa izstrādes, kura tika sadalīta trīs komponentēs: pareizrakstības kļūdu atrašanā, labojumu piedāvāšanā un labojumu kārtošanā pēc to piemērotības. Vispirms tika veikta pareizrakstības kļūdu atrašanas iespēju izpēte. Tika izveidoti divi praktiski risinājumi: PHP + MySQL un PHP + Hunspell risinājums. Lai novērtētu, kurš no risinājumiem ir efektīvāks un izmantojams turpmākajā darba procesā, tika veikti ātrdarbības mērījumi, kuru rezultātā atklājās, ka PHP + Hunspell risinājums ir krietni ātrāks.

Pētot labojumu piedāvāšanas komponentes darbību, tika padziļināti izpētīta latviešu valodas specifika un analizētas kļūdu pieļaušanas iespējas. Rezultātā tika apskatītas dažādas metodes, ar kuru palīdzību iespējams atrast kļūdainā vārda potenciālos labojumus.

Pēc tam tika apskatīti dažādi pareizrakstības kļūdu labojumu kārtošanas komponentes risinājumi. Tika secināts, ka eksistē vairāki labi funkcionējoši risinājumi angļu valodai, kuri diemžēl nav savietojami ar latviešu valodu. Visbeidzot tika secināts, ka *Noisy channel* metode ir visefektīvākā un pilnībā piemērota latviešu valodai. Šīs metodes realizācijai tika izveidoti valodas un kļūdu modeļi, kuru datu avots ir *draugiem.lv* dienasgrāmatu sadaļas ieraksti.

Visbeidzot tika izveidots pareizrakstības rīka prototips, kuru ir iespējams ērtā veidā savienot ar mājaslapā esošajiem teksta ievades laukiem. Salīdzinājumā ar *Open office* un tīmekļa pārlūkprogrammās pieejamajiem pareizrakstības rīkiem, kas bāzēti uz *Hunspell*, autora izstrādātajam rīkam ir jauns kļūdu labojumu kārtošanas algoritms, kurš darbojas pēc

Noisy channel modeļa metodes. Papildus tam, tika izveidots API risinājums, kas paver iespējas rīku izmantot ne tikai tīmekļa vietnēs, bet arī dažādās darbvirsmas un mobilajās lietotnēs u.c.

Ņemot vērā iegūtos rezultātus, autors uzskata, ka izvirzītie darba mērķi tika pilnībā sasniegti – ir izstrādāts jauns pareizrakstības rīka prototips. Ar šāda rīka palīdzību nākotnē būs iespējams ērti aprīkot jebkuru mājaslapu ar pareizrakstības rīku, kā arī, pateicoties izstrādātajam API mehānismam, pareizrakstības rīka funkcionalitāti iespējams integrēt arī darbvirsmas un mobilajās lietotnēs. API mehānisms paver ļoti plašas iespējas attīstīt pareizrakstības rīka potenciālu, kā rezultātā tiktu krietni uzlabota Latvijas tīmekļa resursos pieejamā teksta kvalitāte.

6 IZMANTOTĀS LITERATŪRAS SARAKSTS

1. Hunspell rīks [tiešsaiste][19.01.2013]. Pieejams internetā: http://hunspell.sourceforge.net
2. Tildes biroja programmatūra [tiešsaiste][19.01.2013]. Pieejams internetā: http://www.tilde.lv/tildes-birojs-2011
3. Tildes biroja pareizrakstības pārbaudes rīks [tiešsaiste][19.01.2013]. Pieejams internetā: http://www.tilde.lv/tildes-birojs/pareizrakstibas-parbaude
4. I. Skadiņa, A. Veisbergs, A. Vasiļjevs, T. Gornostaja, I. Keiša, A. Rudzīte, „Latviešu valoda digitālajā laikmetā", 2012.
5. D. Deksne, R. Skadiņš, „CFG Based Grammar Checker for Latvian", 2011.
6. Most Popular | Repository for Apache OpenOffice Extensions [tiešsaiste] [19.01.2013]. Pieejams internetā: http://extensions.services.openoffice.org/dictionaries
7. Latviešu valodas pareizrakstības pārbaudes modulis [tiešsaiste][13.05.2013]. Pieejams internetā: http://extensions.services.openoffice.org/en/project/dict_lv_LV
8. M. Choudhury, M. Thomas, A. Mukherjee, A. Basu, N. Ganguly, „How Difficult is it to Develop a Perfect Spell-checker? A Cross-linguistic Analysis through Complex Network Approach", 2007.
9. Pareizrakstība [tiešsaiste]. Latvijas valodas aģentūra. [19.01.2013]. Pieejams internetā: http://valoda.lv/Biezak_uzdotie_jautajumi/Pareizrakstiba/726/mid_585
10. MySQL [tiešsaiste]. Wikipedia, 11.03.2013 [19.01.2013]. Pieejams internetā: http://lv.wikipedia.org/wiki/MySQL
11. MySpell [tiešsaiste]. Wikipedia, 18.01.2013 [19.01.2013]. Pieejams internetā: http://en.wikipedia.org/wiki/MySpell
12. SBCS [tiešsaiste]. Wikipedia, 2.06.2012 [19.03.2013]. Pieejams internetā: http://en.wikipedia.org/wiki/Single-byte_character
13. APC [tiešsaiste], The PHP Group, 6.01.2013 [19.01.2013]. Pieejams internetā: http://pecl.php.net/package/APC
14. php-hunspell [tiešsaiste]. Sergejs Lomakovs, 29.09.2011 [19.01.2013]. Pieejams internetā: https://github.com/sapfeer0k/php-hunspell
15. Brāļi kaudzītes, „Mērnieku Laiki" [tiešsaiste][19.01.2013]. Pieejams internetā: http://www.letonika.lv/literatura/read.aspx?f=1&r=160
16. Soundex [tiešsaiste]. Wikipedia, 8.05.2013 [16.05.2013]. Pieejams internetā: http://en.wikipedia.org/wiki/Soundex

17. IBM Information Management Software, „Is Soundex Good Enough for You? The Hidden Risks of Soundex-Based Name Searching.”, 2006.
18. H.L. Liang, „SPELLCHECKERS AND CORRECTORS: A UNIFIED TREATMENT”, master’s thesis, Faculty of engeneering, built environment and information technology, University of Pretoria, 2008.
19. Levenshtein distance [tiešsaiste]. Wikipedia, 13.05.2013 [16.05.2013]. Pieejams internetā: http://en.wikipedia.org/wiki/Levenshtein_distance
20. Damerau–Levenshtein distance [tiešsaiste]. Wikipedia, 1.05.2013 [16.05.2013]. Pieejams internetā: http://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance
21. D. Jurafsky, „Spelling Correction and the Noisy Channel”, 2012.
22. Language model [tiešsaiste]. Wikipedia, 15.05.2013 [16.05.2013]. Pieejams internetā: http://en.wikipedia.org/wiki/Language_model
23. S. Hussain, T. Naseem, „Spell checking”, 2012.
24. Ajax (programming) [tiešsaiste]. Wikipedia, 9.05.2013 [16.05.2013]. Pieejams internetā: https://en.wikipedia.org/wiki/Ajax_%28programming%29

1. pielikums - PHP + MySQL pareizrakstības kļūdu noteikšanas kods

```php
<?
class Spellchecker{
    private static $checkedWords = array(); // cache sistēma vienreiz jau
pārbaudītajiem vārdiem
    private static $rules = array(); // visus noteikumus ielādēsim atmiņā

    public function __construct(){
        //izveidojam DB konekciju
        $conn = mysql_connect('localhost', 'root', 'pass');
        if (!$conn) {
            die('Could not connect: ' . mysql_error());
        }
        mysql_select_db('spellcheck', $conn);
        mysql_query("SET NAMES 'utf8'");

        $q = mysql_query("SELECT * FROM `rules`");
        while($rul = mysql_fetch_assoc($q)){
            self::$rules[$rul['rule']][] = array(
                'replace' => $rul['replace'],
                'ending' => $rul['ending'],
                'regexp' => $rul['regexp']
            );
        }
    }

    private function saveCheckedWord($word, $val){
        if(!in_array($word, array_keys(self::$checkedWords))){
            self::$checkedWords[$word] = $val;
        }
    }

    private function getCheckedWord($word){
        if(in_array($word, array_keys(self::$checkedWords))){
            return self::$checkedWords[$word];
        }
    }

    // Pārbauda, vai vārds ir pareizs
    public function checkWord($word, $prefix = false){
        if($this->getCheckedWord($word)){
            return true; // cache sistēma
        }

        $word = $this->strtolower($word);

        //nogriežam vārda pirmo daļu
        $len = $this->strlen($word);
        if($len > 13){
            $celms = $this->substr($word, 0, 7);
        }elseif($len > 9){
            $celms = $this->substr($word, 0, 4);
        }elseif($len > 5){
            $celms = $this->substr($word, 0, 3);
        }elseif($len == 3){
            $celms = $this->substr($word, 0, 2);
        }elseif($len == 2){
            $celms = $word;
        }else{ // speciālgadījumi latviešu valodā
            if(in_array($this->substr($word, 0, 1), array('b', 'z', 'm'))){
```

```
                $celms = $this->substr($word, 0, 1);
            }elseif(in_array($this->substr($word, 0, 2), array('ta',
'ie'))){
                $celms = $this->substr($word, 0, 2);
            }elseif(in_array($this->substr($word, 0, 2), array('nā'))){
                $celms = $this->substr($word, 0, 2);
            }else{
                $celms = $this->substr($word, 0, 3);
            }
        }

        // Iegūstam visus iespējamos variantus ar vārda celmu, kā arī visu
šo variantu afiksus
        $q = mysql_query("SELECT * FROM `words` WHERE LOWER(`word`) LIKE
'".mysql_escape_string($celms)."%' ");
        $possible = array(); // visi iespējamie vārdi, jo tie sākas ar
celmu
        $affixes = array(); // visu šo iespējamo vārdu afixi
        while($poss = mysql_fetch_assoc($q)){
            $possible[] = $poss['word'];
            $aff = $poss['affixes'];
            if($aff != ''){
                for($a = 0; $a < $this->strlen($aff); $a++){
                    $affixes[] = $aff[$a];
                }
            }
        }

        foreach($affixes as $affix){
        if(!isset(self::$rules[$affix]))continue; // "ne" pārbaude ir
atsevišķi
            foreach(self::$rules[$affix] as $rul){
            $len_ar_ko_aizvietot = $this->strlen($rul['ending']);
            $len_word = $this->strlen($word);
            $galotne = $this->substr($word, ($len_word -
$len_ar_ko_aizvietot), $len_ar_ko_aizvietot);
              if($galotne == $rul['ending']){
                    if($rul['ending'] != '0'){
                        $seek = $this->substr($word, 0, ($len_word -
$len_ar_ko_aizvietot));
                    }

                    if($rul['replace'] !== "0") {
                        $seek .= $rul['replace'];//
                    }
                    if(in_array($seek, $possible) && ($rul['regexp'] ===
"." || preg_match("/".$rul['regexp']."$/", $seek))){
                        $this->saveCheckedWord($word, true);
                        unset($possible[$seek]);
                        return true;
                    }
              }elseif($rul['ending'] == '0'){ // "vēl". DB ir "vēlēt". No
vārda "vēl" neko nevajag griest nost, lai dabūtu "vēlēt".
                $seek = $word . $rul['replace'];
                if(in_array($seek, $possible)){
                    $this->saveCheckedWord($word, true);
                    unset($possible[$seek]);
                    return true;
                }
              }
        }
        }
```

```
            if($prefix){ // vienreiz jau pārbaudījām ar "ne" prefiksu
                $this->saveCheckedWord($word, false);
                return false;
            }
            if($this->substr($word, 0, 2) == 'ne'){
                $word = $this->substr($word, 2, $this->strlen($word)-1);
                return $this->checkWord($word, true);
            }
            $this->saveCheckedWord($word, false);
            if(isset($seek)){
                unset($possible[$seek]);
            }
            return false;
        }
        // Pārbauda, vai dotajā tekstā ir pareizrakstības kļūdas
        public function checkText($text){
            $words = $this->textToWords($text);
            foreach($words as $k => $word){
                $safeword = ($this->strtolower($word));
                if($safeword != '' && $this->strlen($word) > 1){
                    $words[$k] = $safeword;
                }else{
                    unset($words[$k]);
                }
            }
            // ja vārds uzreiz ir DB
            $q = mysql_query("SELECT * FROM `words` WHERE LOWER(`word`) IN
('".implode("','", $words)."') ");
            while($w = mysql_fetch_assoc($q)){
                $this->saveCheckedWord($w['word'], true);
            }
            $wrong = array();
            foreach($words as $word){
                if(!$this->checkWord($word)){
                    $wrong[] = $word;
                }
            }
            return $wrong;
        }
        // Teksts tiek sadalīts atsevišķos vārdos
        protected function textToWords ($text) {
            $patern = '/[\s\[\].*_!"\/():\-0-
9#$%&\'\'\+,;<=>\?\@\^{}\|\`\\\]+/s';
            return preg_split($patern, $text, -1, PREG_SPLIT_NO_EMPTY);
        }

        // Palīgfunkcijas darbam ar uft-8 tekstu

        private function strtolower($s){
            return mb_strtolower($s, 'utf-8');
        }

        private function substr($s, $from, $to = false){
            return mb_substr($s, $from, $to, 'utf-8');
        }

        private function strlen($s){
            return mb_strlen($s, 'utf-8');
        }


    }
    ?>
```

2. pielikums - PHP + Hunspell pareizrakstības kļūdu noteikšanas kods

```
<?
class Spellchecker{
    private static $hs;

    public function __construct(){
        $dir = dirname(__FILE__);
        self::$hs = hunspell_create($dir.'/lv_LV.aff', $dir.'/lv_LV.dic');
    }

    // Pārbauda, vai vārds ir pareizs
    public function checkWord($word){
        return hunspell_spell(self::$hs, $word);
    }

    // Pārbauda, vai dotajā tekstā ir pareizrakstības kļūdas
    public function checkText($text){
        $words = $this->textToWords($text);

        $wrong = array();
        foreach($words as $word){
            if(!$this->checkWord($word) && !$this->checkWord($this-
>strtolower($word))){
                $wrong[] = $word;
            }
        }
        return $wrong;
    }

    // Teksts tiek sadalīts atsevišķos vārdos
    public function textToWords($text){
        $patern = '/[\s\[\].*_!"\/():\-0-
9#$%&\'\'\+,;<=>\?\@\^{}\|\`\\\]+/s';
        return preg_split($patern, $text, -1, PREG_SPLIT_NO_EMPTY);
    }

    // Palīgfunkcijas darbam ar uft-8 tekstu
    public function strtolower($s){
        return mb_strtolower($s, 'utf-8');
    }

    public function substr($s, $from, $to){
        return mb_substr($s, $from, $to, 'utf-8');
    }

    public function strlen($s){
        return mb_strlen($s, 'utf-8');
    }
}
?>
```

Dokumentārā lapa

Maģistra darbs: **Tīmekļbāzēts latviešu valodas pareizrakstības rīks**

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: _____
(Autora paraksts)

Ar savu parakstu apliecinu, ka esmu lasījis augstāk minēto maģistra darbu un atzīstu to par **p i e m ē r o t u / n e p i e m ē r o t u** (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu maģistrantūrā.

Darba vadītājs: _____
(Vadītāja paraksts)

Darbs iesniegts **maģistrantūras sekretariātā** _____.
(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Studiju metodiķe: _____.
(Metodiķes paraksts)

Recenzents: _____
(Akad.amats, zin.grāds, vārds, uzvārds)

Darbs aizstāvēts maģistra gala pārbaudījuma komisijas sēdē

_____ prot. Nr. _____
(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____
(Sekretāra paraksts)