LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

CENTRALIZĒTAS UZDEVUMU UZSKAITES SISTĒMAS IEVIEŠANAS IESPĒJAS UZŅĒMUMĀ

BAKALAURA DARBS

Autore: Laura Bobule
Studenta apliecības Nr.: lb16046
Darba vadītājs: Prof.Mg.Sc.Ing. Haralds Gribusts

RĪGA 2020

ANOTĀCIJA

Centralizētās uzdevumu uzskaites sistēmas mērķis, ir nodrošināt ērtu, vienkārši pārskatāmu kopskatu par uzdevumiem, pieteikumiem vai citiem apstrādājamiem resursiem, kuri ir piešķirti konkrētam lietotājam gala sistēmās.

Bakalaura darba „Centralizētas uzdevumu uzskaites sistēmas ieviešanas iespējas uzņēmumā" izstrādes laikā apkopotas galvenās sistēmas prasības, izpētīti gatavie risinājumi, kas nodrošina nepieciešamo funkcionalitāti, izpētītas pielietojamās tehnoloģijas starpsistēmu komunikācijas nodrošināšanai, izvēlēts datubāzes veids un piedāvāti divi realizācijas piedāvājumi, balstoties uz izpētītajiem risinājumiem.

Darbs satur 38 lappuses, 16 attēlus, 1 tabulu, 19 avotus.

Atslēgvārdi: WebSockets, Tīmekļa aizķeres, starpsistēmu komunikācija, centralizācija, integrācija

ABSTARCT

CENTRALIZED TASK TRACKING SYSTEM IMPLEMENTATION OPPORTUNITIES IN THE COMPANY

The purpose of a centralized task tracking system is to provide a convenient, easy-to-understand overview of the tasks or other resources that are assigned to a particular user in the end systems.

During the development of the bachelor's thesis "Centralized task tracking system implementation opportunities in the company" the main system requirements are summarized, three ready-made solutions that provide the required functionality are researched, applied technologies for cross-system communication are researched, database type is selected and offers are offered based on the researched solutions.

Bachelor thesis contains 38 pages, 16 images, 1 table, 19 sources of information.

Keywords: WebSockets, Webhooks, cross-system communication, centralization, integration

SATURS

APZĪMĒJUMU SARAKSTS

ADFS - *active directory federation services* - aktīvās direktorijas federācijas pakalpojums.

AJAX - *Asynchronous JavaScript And XML* - asinhronais Javascript un XML.

API - *application programming interface* - lietojumprogrammas saskarne.

Centrāle - saīsinājums no "Centralizētā uzdevumu uzskaites sistēma".

HTTP - *Hypertext Transfer Protocol* - hiperteksta transporta protokols.

Huginn - sistēma automatizētu procesu veikšanai.

IBM - *International Business Machines Corporation* - datoru tehnoloģiju un konsultāciju korporācija.

Iframe - *Inline Frame* - peldošā ietvara elements.

Java - objektorientēta programmēšanas valoda.

JavaScript - skriptu progammēšanas valoda.

Jira - darbu pārvaldības un projektu vadības sistēma.

JSON - *JavaScript Object Notation* - datu apmaiņas formāts.

Klasiskā aptaujas metode - *polling*, arī *short polling* - metode, kurā klients pieprasa servera datus regulāros intervālos.

LDAP - *Lightweight Directory Access Protocol* - direktoriju viegļpiekļuves protokols.

LEports - iekšējā informācijas portāla nosaukums.

Laravel - PHP koda satvars.

MIME - *Multipurpose Internet Mail Extensions* - interneta pasta vairākmērķu paplašinājumi, interneta standarts e-pasta formātam.

NoSQL - *Not only SQL/Non SQL* – ne tikai SQL datubāze/ ne SQL datubāze.

Oracle - datubāzu pārvaldības sistēma.

PHP - *Hypertext Preprocessor* - pirmkoda skriptu valoda.

RDBMS - *Relational Data Base Management System* - relāciju datu bāzu vadības sistēma.

RDSMS - *Relational Data Stream Management System* - relāciju datu plūsmu vadības sistēma.

REST - *REpresentational State Transfer* -

Ruby - programmēšanas valoda.

Ruby on Rails - programmēšanas valodas Ruby satvars.

SSE - *Service-Sent Events*

SSO - *Single Sign On* - vienotā pierakstīšanās.

SQL - *Structured Query Language* - strukturēta vaicājumvaloda.

TCP - *Transmission Control Protocol* - pārraides vadības protokols.

Tīmekļa aizķeres - *Webhooks*

Tīmekļa pakalpes - *Web Services* - vienots veids, kā savstarpēji sazināties tīkla lietojumprogrammām

TLS - *Transport Layer Security* - transporta slāņa drošība

URL - *Universal Resource Locator* - vienotais resursu vietrādis. Adrese, kas pārlūkā norāda, kur var atrast kādu konkrētu interneta resursu.

WebSockets - Tīmekļa ligzdas

W3C - *World Wide Web Consortium* - Globālā tīmekļa konsorcijs

XML - *Extensible Markup Language* - paplašināmās iezīmēšanas valoda

Zapier - rīks integrācijas un automatizētu procesu veikšanai.

IEVADS

Uzņēmuma darbinieki ikdienā izmanto dažādas sistēmas, kur katrai no tām ir atšķirīgs mērķis, darba plūsmas un atšķiras kopējais uzdevumu apstrādes process. Brīdī, kad jauns darbinieks uzsāk darbu uzņēmumā, lielais saņemtās informācijas apjoms no dažādām sistēmām var radīt apjukumu. Tāpat no sistēmām, kuras tiek izmantotas retāk, saņemtie darba uzdevumi un termiņi var tikt piemirsti un rodas nepieciešamība atgādināt par to izpildi. Lai atvieglotu darbinieku ikdienu, ir izteikta ideja, par vienotas sistēmas ieviešanu.

Centralizētās uzdevumu uzskaites sistēmas mērķis, kuras ieviešanas iespējas tiek aplūkotas bakalaura darbā, ir nodrošināt ērtu, vienkārši pārskatāmu kopskatu par uzdevumiem, pieteikumiem vai citiem apstrādājamiem resursiem, kuri ir piešķirti konkrētam lietotājam gala sistēmās.

Bakalaura darba izstrādes laikā nepieciešams veikt sekojošus uzdevumus:

- noformulēt galvenās centralizētās uzdevumu uzskaites sistēmas prasības un pamatot nepieciešamību sistēmas ieviešanai,
- aplūkot saistītas sistēmas un to piedāvātas iespējas,
- noskaidrot, vai pastāv jau gatavi risinājumi, kas nodrošina nepieciešamo funkcionalitāti, to ieviešanas izmaksas, piedāvātās iespējas, un nepieciešamo pielāgojumu veikšanu,
- analizēt pielietojamās tehnoloģijas starpsistēmu komunikācijas nodrošināšanai, lai maksimāli samazinātu resursu izmantošanu gala sistēmās,
- izpētīt kāda datubāzes struktūra risinājumam būtu atbilstošākā.

Bakalaura darba mērķis ir aplūkot iespējamos variantus centralizētās uzdevumu uzskaites sistēmas ieviešanai uzņēmumā un nobeigumā sniegt realizācijas piedāvājumu, balstoties uz izpētītajiem risinājumiem

Darbs sastāv no piecām nodaļām:

1. Centralizētās sistēmas apraksts, kurā iekļauts problēmas apraksts, sistēmas prasību vispārīgs apraksts, saistīto sistēmu funkcijas, izmantotās tehnoloģijas un to loma saistībā ar centrāli.
2. Nodaļā izpētīti gatavie risinājumi un citas alternatīvas, kas varētu nodrošināt nepieciešamo funkcionalitāti.

3. Izpētīti, aprakstīti un salīdzināti seši risinājumi starpsistēmu komunikācijas nodrošināšanai, sniegts to salīdzinājums un atbilstība centralizētās uzdevumu uzskaites sistēmas kontekstā.
4. Salīdzinātas divas dažādas datubāzu struktūras - relāciju un nerelāciju datu bāzes, aprakstīta populārākā dokumentu glabāšanas datubāze - MongoDB
5. Ņemot vērā visu veikto izpēti, piedāvāts risinājums centralizētās uzdevumu uzskaites sistēmas realizācijai.

1. CENTRALIZĒTĀS SISTĒMAS APRAKSTS

Sistēmas izstrāde sākas ar prasību apkopošanu. Pēc sarunām ar iesaistītajām personām, saņemts aptuvens apraksts ar pašreizējās situācijas problēmām, no kurām izveidots prasību saraksts. Pirmajā apakšnodaļā "Problēmas apraksts" pamatota vajadzība pēc šādas sistēmas un ieguvumi no tās ieviešanas. Nodaļā "Vispārīgas prasību apraksts" apkopotas galvenās centralizētās uzdevumu uzskaites sistēmas funkcijas, kuras vēlētos sistēmas lietotāji. Tā kā vēlamās funkcionalitātes nodrošināšanai nepieciešama integrācija ar citām uzņēmumā lietotajām sistēmām, šīs sistēmas un to tehnoloģijas aprakstītas apakšnodaļā "Saistītās sistēmas".

1.1. Problēmas apraksts

Darbinieki, veicot savus ikdienas darba pienākumus, strādā ar daudzām sistēmām, līdz ar to, lai redzētu savus veicamos darbus un citus uzdevums, tiem jāpieslēdzas katrai sistēmai atsevišķi. Apstrādājot uzdevumus vienā sistēmā, var nepamanīt jaunus uzdevumus, kas piešķirti citās sistēmās.

Par saņemtu uzdevumu no vairākām sistēmām tiek nosūtīti arī e-pasti, taču ne visās sistēmās tie ir pieejami un, saņemot daudzus e-pastus ir grūti izsekot, kurā brīdī nepieciešams veik kādu darbību, un, kurš ir tikai informatīvs ziņojums. Attēlā 1.1.1. redzams ekrānuzņēmums no e-pasta par saņemtajiem ziņojumiem saistībā ar vienu pieteikumu, vienas dienas laikā no sistēmas, kas ar e-pasta palīdzību paziņo par visām izmaiņām saistībā ar kādu pieteikumu. Tas ir noderīgi, ja pieteikumu nav daudz, taču, palielinoties pieteikumu apjomam, lietotāji bieži vien novirza šos e-pastus uz kādu atsevišķu mapi, līdz ar to, dažreiz palaižot garām arī noderīgu informāciju saistībā ar pieteikumiem.

1.1.1. att. **Saņemtie e-pasta ziņojumi par pieteikumu**

Īpaši mulsinošs lielais informācijas apjoms var būt darbiniekiem, kuri tikko uzsākuši strādāt uzņēmumā un vēl nepārzina katras sistēmas īpatnības, vispārējos procesus un darba plūsmas.

Lai atvieglotu darbinieku ikdienu, plānots ieviest centralizētu uzdevumu uzskaites sistēmu, kurā ērtā veidā iespējams aplūkot kopskatu ar lietotājam aktuālajiem darba uzdevumiem un piešķirtajiem pieteikumiem gala sistēmās. Ieguvumi no šādas sistēmas ieviešanas būtu sekojoši:

- nav nepieciešams regulāri pieslēgties vairākām sistēmām, lai redzētu savus veicamos uzdevumus,
- darbiniekiem, redzot kopainu, iespēja labāk plānot un prioritizēt darbus,
- ātrāks uzdevumu apstrādes process,
- atgādinājums par veicamajiem darbiem.

1.2. Vispārīgs prasību apraksts

Centralizētā sistēma jāveido kā atsevišķs modulis sistēmā, kuru ikdienā jau lieto lielākā daļa darbinieku, lai tā būtu ērti pieejama visiem darbiniekiem un nebūtu nepieciešama vēl viena papildu pieslēgšanās kādai jaunai sistēmai, taču lielā informācijas apmaiņas apjoma dēļ, tās datubāzei jāglabājas uz atsevišķa servera. Sistēmā būtu jāredz visi konkrētajam lietotājam piešķirtie darba uzdevumi un pieteikumi no citām ikdienā lietojamajām sistēmām, kas palīdzētu darbiniekam prioritizēt aktuālos darbus un atgādinātu par citiem veicamajiem uzdevumiem. Jauniem datiem no sistēmām jāparādās bez lapas pārlādes, pietiekami īsā laikā no uzdevuma piešķiršanas brīža.

Tā kā uzņēmumā tiek lietotas daudzas un uz atšķirīgām tehnoloģijām balstītas sistēmas, tad svarīgākais uzdevums ir atrast optimālu veidu kā saņemt vai pieprasīt datus no visām šīm sistēmām. Svarīgi, lai veicot izmaiņas gala sistēmās un papildinot plūsmas nebūtu nepieciešamas būtiskas izmaiņas centrālajā sistēmā vai nepieciešamība katrā plūsmā norādīt vajadzību sūtīt datus uz to. Lielāko daļu loģikas jāizstrādā centrālajā sistēmā, tādā veidā maksimāli samazinot gala sistēmu noslodzi un izmaiņu pieprasījumu izmaksas tajās sistēmās, kuras uztur ārpakalpojums.

Nav plānots, ka centralizētajā uzdevumu sistēmā tiek veiktas darbības ar piešķirtajiem uzdevumiem, tie tikai pieejami aplūkošanai un satur būtiskāko informāciju - sistēmu, no kuras nācis pieteikums, izveidošanas un izpildes datumus, nosaukumu, statusu, aprakstu kā arī ar saiti uz konkrēto uzdevumu gala sistēmā. Gala sistēmās, kā līdz šim, veicamas visas pieteikumu statusu pārējas, kas tikai attēlosies centrālē. Tāpat sākotnēji sistēmā nav plānota arī statistika par pieteikumu apstrādi vai to apstrādei patērēto laiku, taču nākotnē, saņemot

lietotāju atsauksmes un ieteikumus, iespējams izstrādāt funkcionalitāti, kas piedāvātu papildu statistiku interaktīvā veidā.

1.3. Saistītās sistēmas

Apkopojot uzņēmuma informācijas sistēmas, izlemts, ka sākotnēji jānodrošina starpsistēmu komunikācija ar biežāk izmantotajām un būtiskākajām sistēmām, kurās redzami aktuālie darbi un uzdevumi. Katra no tām ir balstīta uz atšķirīgām tehnoloģijām, līdz ar to, atšķiras pieejamā funkcionalitāte datu apmaiņai.

1.3.1. Jira Core

Kā pirmā sistēma izvēlēta Jira Core, kura uzņēmumā ir divās instancēs. Jira ir galvenā darbu pārvaldības un projektu vadības sistēma, uzņēmuma gadījumā tā tiek izmantota kā galvenais rīks lietotāju problēmu un izmaiņu pieprasījumu pieteikšanai pārējām informācijas sistēmām. Laika gaitā ir ievērojami palielinājies tajā pārvaldāmo projektu skaits, katram no tiem ir savas darba plūsmas un dažādi statusi. Līdz ar to, lai funkcionalitāte atbilstu vēlamajām darījumprasībām, veikti daudzi pielāgojumi un ieviesti tādi risinājumi, kas neietilpst Jira standarta funkcionalitātē. Šāda veida izmaiņas sarežģī sistēmas atjaunināšanu uz jaunākām versijām, tādēļ centralizētās uzdevumu sistēmas risinājumam jāizmanto pieejamā Jira funkcionalitāte.

Uzņēmumā tiek izmantota Jira Core Server, tā ir Java programmēšanas valodā veidota programmatūra. Šobrīd tiek izmantotas versijas 6.3.15 un 7.2.7, taču šī gada laikā tiek plānota migrācija uz jaunāku versiju. Tām tiek izmantota Oracle datubāze. Ir pieejama plaša dokumentācija no izstrādāju mājas lapas, kurā aprakstītas pieejamās integrācijas, API un citi rīki. Lietotāju autentifikācija tiek nodrošināta izmantojot LDAP.

Centralizētajā uzdevumu sistēmā būtu jāparādās tikai tiem Jira pieteikumiem, ko nepieciešams apstrādāt (tātad, ne slēgtiem, apstrādātiem vai, piemēram, lietošanā saņemtai datortehnikai). Loģiskākais risinājums ir filtrēt atrādāmos pieteikumus pēc to statusa. Kopumā šobrīd vienā no Jira instancēm ir 159 statusi, otrā 210, daļa no tiem abās sistēmās atkārtojas, . Attēlā 1.3.1.1 attēlota daļa no analīzes procesa - apkopoti visi unikālie statusi, to rezolūcijas un lēmums, vai šo statusu atrādīt centrālē (kolonnā “Vai atrādīt” atzīmēts ar “1”) , vai nē (attēlā atzīmēts ar “0”). Redzams, ka rezolūcija nav viennozīmīga un var gadīties, ka vienā gadījumā to vajadzētu atrādīt centrālē, bet citā nē, tādēļ šo vērtību. Apkopojot visus statusus un analizējot, kurus no tiem ir nepieciešams atrādīt centrālē, iegūti 72 atrādāmie statusi.

Nosaukums	Rezolūcija	Vai atrādīt
Reģistrēts	JAUNS Jauns	1
Izpildē	RISINĀŠANĀ Risināšanā	1
Atvērts atkārtoti	JAUNS Jauns	1
Izpildīts	PABEIGTS Pabeigts	0
Slēgts	PABEIGTS Pabeigts	0
Apstrādāts	PABEIGTS Pabeigts	0
Novērtēšanā	JAUNS Jauns	1
Saskaņošanā	PABEIGTS Pabeigts	1
Ārpakalpojums	RISINĀŠANĀ Risināšanā	0
Uzstādīšana	JAUNS Jauns	1
Jauns	JAUNS Jauns	0
Lietošanā	RISINĀŠANĀ Risināšanā	0
Noliktavā	JAUNS Jauns	0
Sagāde	JAUNS Jauns	1

1.2.1.1. att. **Jira statusu analīze**

Jāņem vērā arī tas, ka pieteikumam var mainīties tā numurs un saite, ja, piemēram, pieteikums sākotnēji izveidots neīstajā projektā un vēlāk tas pārvietots uz pareizo.

1.2.2. IBM Notes/Domino

IBM Notes/Domino tiek izmantots, lai uzņēmumā nodrošinātu ne tikai Notes e-pasta klientu, bet arī uz Lotus Domino bāzēto biznesa vadības sistēmu un organizatorisko dokumentu izstrādes sistēmu. Tajās tiek nodrošināta līgumu pārvaldība un to saskaņošana, dokumentu un procesu vadība, rīkojumu, līgumu un citu organizatorisko dokumentu reģistra uzturēšana.

IBM Lotus programmatūra ir veidota LotusScript objektorientētajā programmēšanas valodā, uzņēmumā izmantota 9.0.1 FP10 versija, kam šobrīd vairs nav izstrādātāja atbalsts, taču ir pieejamas dokumentācijas.

Centralizētajā uzdevumu sistēmā jāatrāda dokumenti no biznesa vadības sistēmas un organizatorisko dokumentu izstrādes sistēmas, kuru statusā nepieciešams veikt to tālāku apstrādi. Tas iekļauj līgumus, rīkojumus, nolikumus, vēstules, ziņojumus un citus dokumentus, kas adresēti uzņēmuma darbiniekiem.

1.2.3. Mazās sistēmas

Uzņēmumā ir ieviestas arī vairākas mazās informācijas sistēmas, kas paredzētas dažādu specifisku darbu veikšanai un pārvaldībai:

- caurlaižu sistēma,
- prakšu pārvaldības sistēma,
- risku pārvaldības sistēma.

Par mazām tās tiek uzskatītas tādēļ, ka tās lieto mazs skaits darbinieku un tādēļ, ka ar tām ikdienā, kā galvenajām sistēmām, strādā tikai daži cilvēki. Tieši tāpēc šo sistēmu datu atrādīšana centrālē ir būtiska - darbi, kas piešķirti šajās sistēmās var tikt palaisti garām, jo sistēmas netiek pārbaudītas pietiekami regulāri un, tajās piešķirtie uzdevumi, mēdz būt ar ilgu izpildes termiņu - anketas aizpildīšana prakses beigās vai pusgada atskaite.

Tās visas veidotas izmantojot vienu bāzi - Laravel satvaru - primāri tāpēc, lai tās būtu vieglāk uzturēt un realizēt izmaiņu pieprasījums funkcionalitātes papildināšanai. Tāpat tām visām tiek izmantota MySQL datubāze. Līdz ar to, visu šo sistēmu integrācija ar centralizēto uzdevumu sistēmu būs līdzīga. Laravel satvaram pieejama plaša dokumentācija un pakotnes dažādu risinājumu izstrādes atvieglošanai. Arī šajā gadījumā lietotāju autentifikācija tiek nodrošināta izmantojot LDAP.

Tāpat kā citām sistēmām, svarīgi ir atlasīt tos statusus, kuros dati jāatrāda centrālē, taču tā kā sistēmas savstarpēji ir atšķirīgas, arī tajās izmantoto plūsmu realizācija atšķiras. No caurlaižu sistēmas būtu jāatrāda apstrādei nodotie, saskaņošanā un piešķiršanā esošie pieteikumi kā arī iesniegtie, reģistrētie un saskaņošanā esošie dokumenti. No prakšu pārvaldības sistēmas centrālē jāparādās neaizpildītajām prakšu vērtēšanas anketām, lietotāju ērtības dēļ tās varētu atrādīt tikai sākot ar, piemēram, nedēļu pirms prakses pēdējās dienas. No risku reģistra jāatrāda saskaņošanā esošie vērtējumi un risku mazināšanas pasākumi izpildē.

1.2.4. LEports

Plānots, ka sistēma, kurā tiks atrādīti darbinieka uzdevumi būs uzņēmuma informācijas portāls – LEports – tā ir iekšējā sistēma, kurā darbinieki ātri un vienkārši var atrast sev interesējošo informāciju, jaunākās ziņas, kolēģu kontaktinformāciju un uzņēmuma aktualitātes. Sistēma ir piemērota šādas informācijas atrādīšanai, jo to ikdienā izmanto lielākā daļa darbinieku un tajā pieejama aktuālā informācija, šajā gadījumā tā tiktu papildināta ar aktuālajiem darba uzdevumiem.

LEports izstrādāts izmantojot PHP programmēšanas valodu un Laravel satvaru. Tam tiek izmantota MySQL datubāze. Taču tam ir viens mīnuss - nav pieejama nekāda dokumentācija izstrādātājiem, kas sarežģīt funkcionalitātes papildināšanu. LEportā tiek izmantots ADFS vienotās pierakstīšanās (SSO) risinājums.

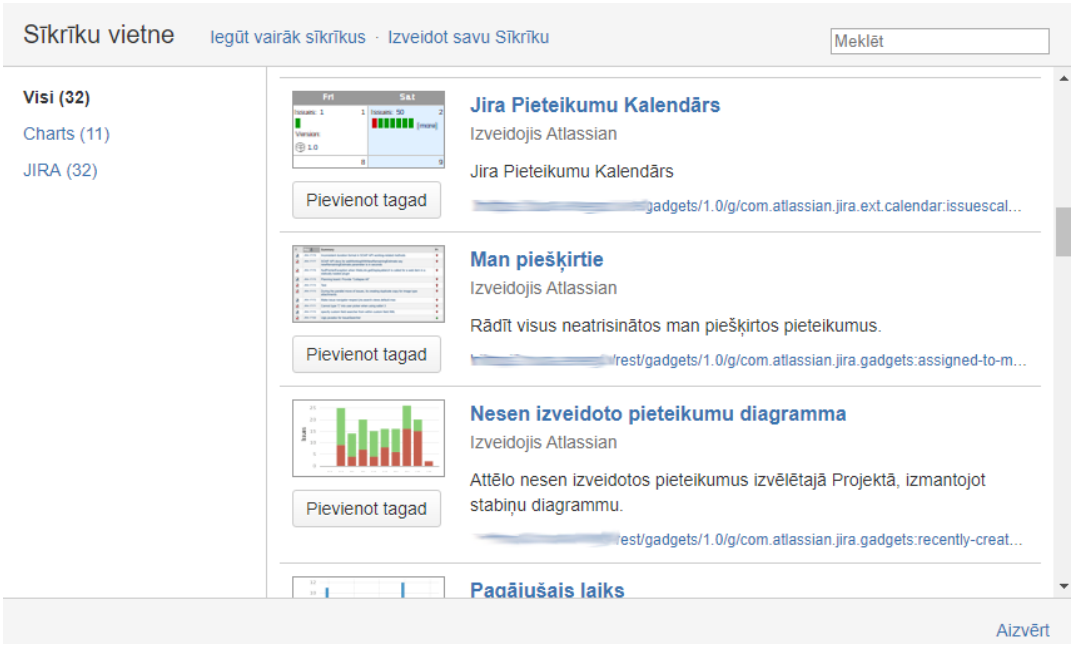
2. GATAVIE RISINĀJUMI UN ALTERNATĪVAS

Saprotams, ka neeksistē gatava sistēma, kura funkcionālā ziņā uzreiz pilnībā atbilstu izvirzītajām prasībām, jo saistītās sistēmas ir specifiskas uzņēmuma vajadzībām. Jāņem vērā arī tas, ka tās ar laiku var mainīties, ja tiek ieviestas jaunas sistēmas.

Taču eksistē sistēmas un risinājumi, ko iespējams pielāgot tā, lai tie atbilstu izvirzītajām prasībām un nodrošinātu nepieciešamo funkcionalitāti. Nodaļā aplūkoti iespējamie risinājumi un alternatīvas, kas varētu vai nu aizstāt jaunas sistēmas izstrādi, vai atvieglot šo procesu. Aplūkoti trīs iespējamie varianti - esošas sistēmas izmantošana, maksas rīks un bezmaksas, atvērtā pirmkoda risinājums.

2.1. Jira

Kā viens no alternatīvajiem variantiem ir Jira, kas jau tiek izmantota uzņēmumā un tās lietošana ir aprakstīta nodaļas 1.3.1. sākumā. Sistēmā ir pieejamas plašas iespējas personalizēt savu darba virsmu - veidojot specifiskus filtrus, pielāgojot pārskatus, atlasot pieteikumus un vizualizējot tos grafiku formā, kā arī, izmantojot pieejamos sīkrīkus. Attēlā 2.1.1. redzami gatavie Jira sīkrīki, kas piedāvā vizualizēt interesējošos datus. No tehniskā viedokļa, Atlassian (Jira izstrādātāji) veikalā ir pieejami vairāki integrāciju rīki ar citām sistēmām, kas nozīmē, ka Jira varētu kļūt par centrālo sistēmu, saņemot datus no pārējām sistēmām. Tāpat ir iespējams savienot arī vairākas Jira instances.



2.1.1. att. **Jira pieejamo sīkrīku skats**

Tā kā to jau izmanto liela daļa darbinieku, tad funkcionalitāte daudziem jau ir zināma un saprotama. Tomēr no uzņēmumā veiktajiem lietotāju apmierinātības pētījumiem, ar sistēmu pilnībā apmierināti ir tikai 58% no 640 respondentiem un liela daļa uzskata, ka sistēmas saskarne nav draudzīga lietotājam. Neskatoties uz to, šim risinājumam ir vēl vairāki mīnusi:

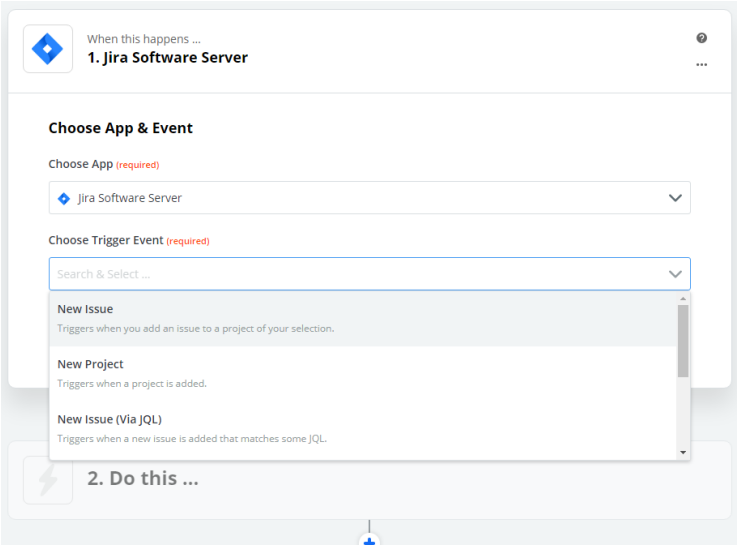
- daudzi no integrācijas spraudņiem ir maksas,
- šie spraudņi bieži vien ir saderīgi tikai ar jaunākajām Jira versijām,
- Jira sistēmas licenču maksa ir atkarīga no lietotāju skaita,
- ja kādu integrāciju nav iespējams izveidot no veikalā pieejamajiem spraudņiem, tad tam nepieciešama atsevišķa izstrāde no ārpakalpojuma,
- sistēma ir liela un izmanto daudz resursu, kas ieviešot šādu risinājumu var radīt veiktspējas problēmas.

Principā šāda risinājuma ieviešana nozīmē, ka būtu nepieciešama visu Jira instanču migrācija uz jaunāko versiju un papildu izdevumi nepieciešamo spraudņu iegādei kā arī sistēmas infrastruktūras nodrošināšanai.

2.2. Zapier

Zapier ir integrācijas rīks, kas ļauj automatizēt procesus starp dažādām lietotnēm un sistēmām. Tas ļauj savienot divas un vairāk lietotnes, veidojot plūsmas, kurās izpildoties notikumiem tiek veiktas darbības. Šo savienojumu plūsmu izveide ir vienkārša un nav nepieciešamas programmēšanas zināšanas, ja savienošanai izvēlētās sistēmas ir iekļautas starp tām, kuru integrācija ir atbalstīta. Šobrīd tiek piedāvāta integrācija starp vairāk nekā 2000 lietotnēm [1].

Kā redzams attēlā 2.2.1. izvēloties, piemēram, sistēmu Jira, tiek piedāvāti notikumi, uz kuriem reaģēt. Nākamais solis ir izvēlēties sistēmu, kurai šos datus padot un darbību, ko ar tiem veikt. Maksas plānos, plūsmām iespējams pievienot arī vēl papildu soļus, tādā veidā izveidojot sarežģītākas plūsmas un apstrādājot datus vairākās lietotnēs.



2.2.1. att. **Plūsmu izveide Zapier rīkā**

Pieejams arī atbalsts iekšējām sistēmām divos variantos. Pirmajā gadījumā veidojot savienojumu starp datu bāzēm, veicot savstarpējas datu apmaiņas un pēc tam veicot saņemto datu apstrādi, taču tas ir maksas pakalpojums, jo liela daļa no datu bāzēm, piemēram MySQL, ir atzīmētas kā premium lietotnes. Otrais variants ir izmantot Zapier Platfrom un definēt savas lietotnes Node.js failos, šajā gadījumā būs nepieciešams programmēšanas darbs, taču to atvieglo pieejamā dokumentācija un pamācības.

Turpinot par pieejamajiem maksas un bezmaksas plāniem, šis risinājums uzņēmuma kontekstā nebūs finansiāli izdevīgākais, jo tiek piedāvāti 5 plāni, no kuriem viens ir bezmaksas. Tas iekļauj 100 datu apmaiņas (Tasks) un 5 savienojumus (Zaps) mēnesī. Turpmākajos plānos šīs vērtības palielinās, iekļaujot vēl dažādus bonusus, kā piemēram, lietotāju pievienošanu un piekļuves atļauju piešķiršanu. Veicot vienkāršus aprēķinus, lai iegūtu aptuveno datu apmaiņu skaitu mēnesī (sareizinot vidējo jaunu darba uzdevumu skaitu sistēmās vienā dienā ar vidējo statusu pāreju skaitu un darba dienu skaitu mēnesī, tiek iegūtas vairāk nekā 100 000 apmaiņas), būtu nepieciešams “Company” plāns, kas izmaksās aptuveni 830 eiro mēnesī (samaksa ASV dolāros, konvertācija pēc aktuālā kursa), gandrīz 10 000 eiro gadā, ja tiek veikts ikgadējs maksājums vai 1030 eiro mēnesī izvēloties ikmēneša maksājumu. Šis skaitlis var palielināties līdz 18 000 gadā, ja tiek izlemts pievienot vēl citas sistēmas, tādā veidā palielinot datu apjomu [1].

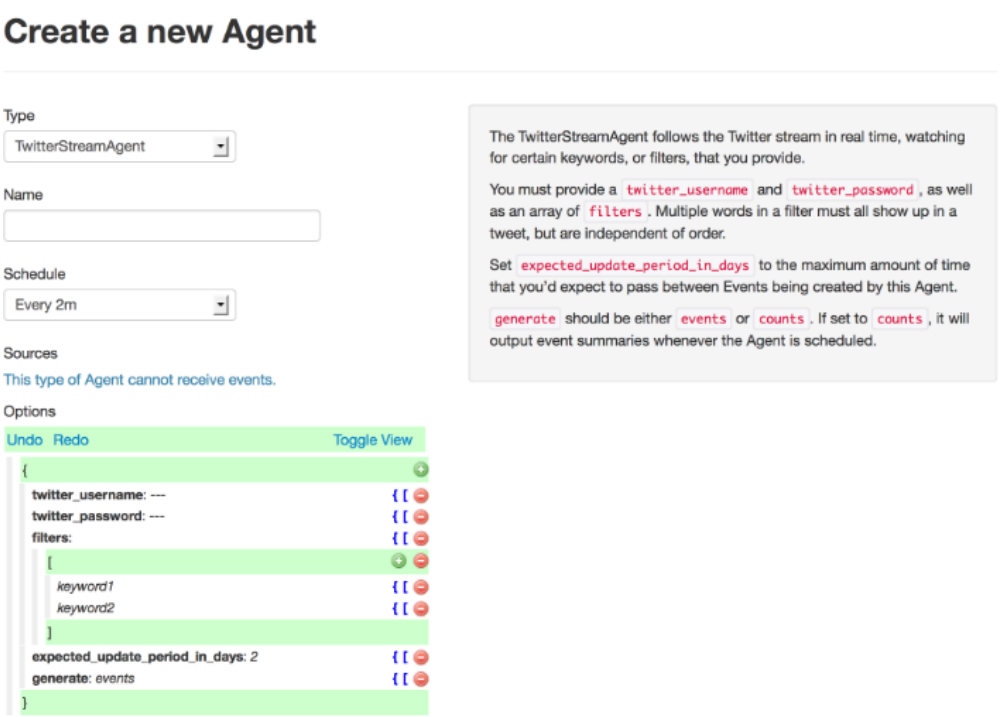
Kā galvenās risinājuma priekšrocības var minēt tā vieglo un saprotamo lietošanu, plašo dokumentāciju un pamācības.

Taču ir vairāki mīnusi:

- Risinājums ir maksas rīks un tā izmantošana ilgtermiņā nav lēta,
- ir nepieciešams veikt konfigurēšanu un visu plūsmu izveidi,
- iekšējo sistēmu savienojumam nepieciešama vai nu datu bāzē saglabāto datu apstrāde, vai savu lietotņu definēšana,
- vienmēr pastāv datu drošības riski, ja tiek izmantotas ārējās sistēmas datu apmaiņai.

2.3. Huginn

Huggin ir sistēma starpnieku veidošanai, kuri veic automatizētus procesus. Tā piedāvāta funkcionalitāte ir līdzīga iepriekšējā nodaļā aplūkotajam Zapier rīkam, tikai atšķirībā no Zapier, Huginn ir atvērtā pirmkoda, bezmaksas risinājums. Attēlā 2.3.1. redzams process, kādā Huginn sistēmā tiek veidotas automatizācijas plūsmas - jāizvēlas starpnieks, tas, cik bieži šo procesu atkārtot un var norādīt papildu parametrus. Ir arī pieejamas veidnes, kas ļauj veidot un konfigurēt savus starpniekus.



2.3.1. att. **Huginn plūsmu izveide [2]**

Huginn rīks ir vienkāršs un piedāvā vairākas konfigurēšanas iespējas, lai to pielāgotu nepieciešamajām prasībām. Tāpat risinājums ir atvērtā pirmkoda, dažas no jau pieejamajām funkcijām varētu atvieglot centralizētās uzdevumus uzskaites sistēmas izstrādi,

Tomēr risinājumam ir vairāki mīnusi:

- tas izstrādāts Ruby on Rails ietvarā, uzņēmuma iekšējām sistēmām, lai atvieglotu uzturēšanu, cenšas izvēlēties Laravel ietvaru,
- nepieciešams laiks, lai apgūtu sistēmas funkcijas, to uzstādītu un veiktu visus nepieciešamos pielāgojumus,
- pieejamā dokumentācija nesniedz skaidru ieskatu sistēmas funkcijās un to darbībā, ir pieejams vispārīgs apraksts un dažas lietotāju veidotas pamācības,
- jāveido un jākonfigurē atbilstoši starpnieki, jo ar pieejamajiem nevar nodrošināt nepieciešamo funkcionalitāti.

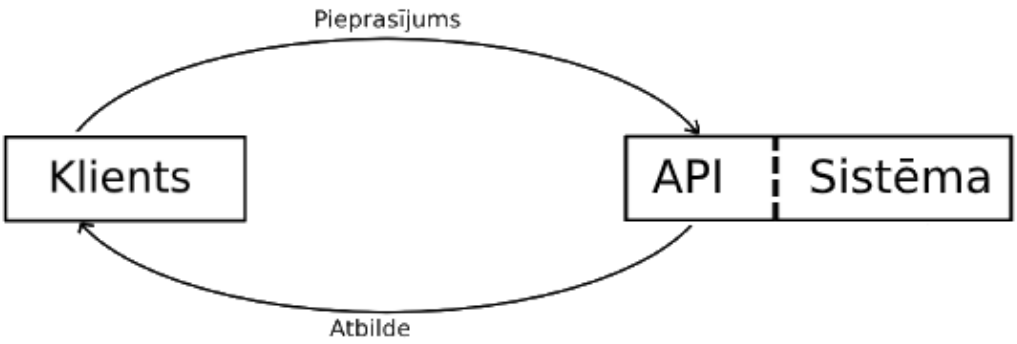
Kopumā sistēma nerada uzticību un tā vairāk būtu piemērota nelielu uzdevumu automatizēšanai, kam, pēc tās autora komentāriem, tā sākumā bija paredzēta. Iespējams, pēc kāda laika tā būs pietiekami attīstīta, lai to varētu izmantot lielos projektos, taču šobrīd tā nepiedāvā visus nepieciešamos rīkus, lai nodrošinātu liela uzņēmuma sistēmu sasaisti.

3. STARPSISTĒMU KOMUNIKĀCIJAS IESPĒJAS

Efektīva datu iegūšana no gala sistēmām ir viena no svarīgākajām centralizētās uzdevumu uzskaites sistēmas funkcijām. No izvēlētās metodes atkarīga gan pārējo sistēmu veiktspēja, gan centrāles lietojamība. Respektīvi, vai dati tiek iegūti reāllaikā un to iegūšana būtiski neietekmē pārējo sistēmu noslodzi. Nodaļā aprakstīti izpētītie iespējamie risinājumi, kas varētu nodrošināt reāllaika starpsistēmu komunikāciju, aplūkotas un salīdzinātas to priekšrocības un trūkumi, kā arī atbilstība centralizētas sistēmas risinājumam.

3.1. API

API - lietojumprogrammas saskarne - ir protokolu, procedūru un citi rīku kopums, kas ļauj piekļūt servera datiem un funkcijām. API padara pieejamus konkrētus servera datus un funkcijas, dodot tiem piekļuvi un tādā veidā atvieglojot sistēmu datu apmaiņu [3]. Izmantojot API ir iespējams nodrošināt saziņu starp starp sistēmām, kuras veidotas izmantojot atšķirīgas programmēšanas valodas un tehnoloģijas. Attēlā 3.1.1. grafiski redzama API darbība - klients nosūta datu pieprasījumu serverim un serveris nosūta tam atbildi. Datu apmaiņai visbiežāk tiek izmantoti HTTP pieprasījumi.



3.1.1. att. **Datu apmaiņa izmantojot API**

Klasiskā API izpildījumā ir iespējams saņemt un apstrādāt datus, taču nav iespējas reaģēt uz izmaiņām datos, tāpat šie dati vairumā gadījumu tiek saņemti tikai pēc klienta pieprasījuma. Šajā gadījumā ir nepieciešams risinājums, kas spēj reaģēt uz izmaiņām gala sistēmās (pieteikumu un resursu apstrāde) un nosūtīt tās centrālei.

R. Fīldings (R. Fielding) - amerikāņu datorzinātinieks, kurš ir arī viens no HTTP protokola autoriem, 2000. gadā savā disertācijas darbā minēja terminu REST (REpresentational State Transfer), tas ir programmatūras arhitektūras stils, kas nosaka sešus ierobežojumus HTTP lietojumprogrammu izstrādei un tīmekļa pakalpēm [3].

REST raksturo:

- uz resursiem orientēta pieeja,
- katram resursam ir unikāls URL,
- vairāki resursa attēlojumi (JSON, XML, MIME),
- vienota HTTP saskarne (GET, POST, PUT, DELETE).

Mūsdienās REST API ir viens no biežāk izmantotajiem paņēmieniem API izstrādei, jo principu ievērošana paātrina to izstrādes laiku un padara tos lietošanu saprotamāku izstrādātājiem.

3.2. Klasiskā aptaujas metode

Izmantojot klasisko aptaujas metodi, ko mēdz saukt arī par īso aptaujas metodi, klients regulāros intervālos nosūta pieprasījumu serverim un, ar katru pieprasījumu, cenšas veikt pieejamo datu atgādāšanu. Ja jauni dati nav pieejami, tad serveris atgriež tukšu atbildi, klients uzgaida konkrētu laika intervālu un veic atkārtotu atgādāšanas pieprasījumu. Šis intervāls ir atkarīgs no latentuma, kādu klients var pieļaut, iegūstot atjaunoto informāciju no servera [4]. Serverim ir jānodrošina API, kurš pados nepieciešamo informāciju un parasti tas tiek izsaukts izmantojot AJAX asinhronos pieprasījumus un norādot noildzes vērtību. Attēlā 3.2.1. redzams implementācijas piemērs no klienta puses, izmantojot AJAX pieprasījumu, funkcija, kad saņemta atbilde, tiek izsaukta rekursīvi pēc norādītā laika intervāla.

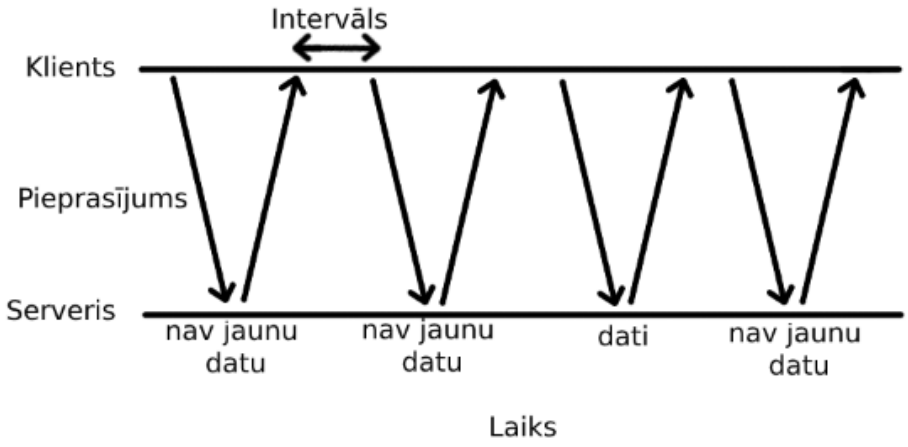
```
(function short_poll() {
    setTimeout(function () {
        $.ajax({
            url: "server",
            success: function (data) {

                //Datu apstrāde

                short_poll(); //Rekursīvi izsauc funkciju
            },
            dataType: "json"
        });
    }, 30000);
})();
```

3.2.1. att. **Klasiskās aptaujas metodes implementācijas piemērs.**

Tā kā klasiskās aptaujas metodes galvenais trūkums ir patērētie servera un tīkla resursi, atbilstoša intervāla izvēle, ir ļoti svarīga. Ja intervāls ir dažas sekundes, tad pieprasījumu biežums var radīt lielu noslodzi, taču tādā veidā var nodrošināt gandrīz reāllaika datu sinhronizāciju. Šis risinājums nav piemērots sistēmām, kurām ir liels lietotāju skaits [5].



3.2.2. att. **Klasiskās aptaujas metodes laika skala**

Attēlā 3.2.2. redzama klasiskās aptaujas metodes laika skala - klients nosūta datu pieprasījumu serverim, uz ko serveris vienmēr sniedz atbildi, pēc noteikta laika intervāla šī darbība tiek atkārtota.

Risinājuma izmantošanas plusi:

- implementācija nav sarežģīta,
- ja nav nepieciešams risinājums, kas nodrošina datus reāllaikā, tad, palielinot atjaunošanas intervālu, risinājums var nodrošināt nepieciešamo datu apmaiņu,

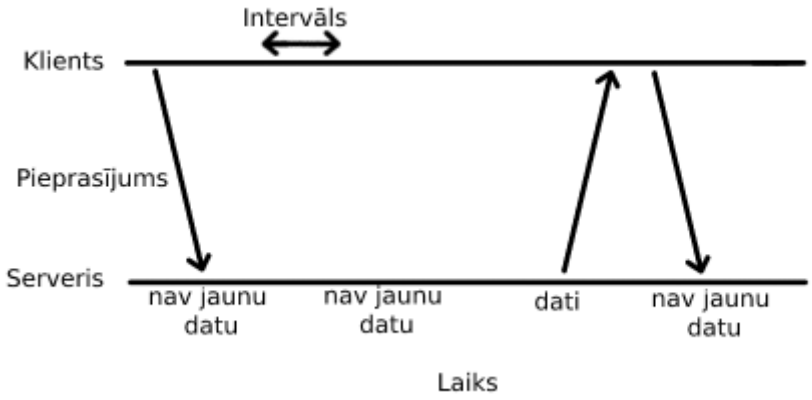
Risinājuma izmantošanas mīnusi:

- intervāla dēļ, saņemot datus, nav zināms kurā mirklī tieši tie parādījās,
- patērētie servera un tīkla resursi, ja liela datu apmaiņa.

3.3. Garā aptaujas metode

Garā aptaujas metode mēģina samazināt gan servera-klienta ziņojumu piegādes latentumu, gan apstrādes un tīkla resursu izmantošanu. Klients nosūta sākotnējo pieprasījumu taču serveris atbild uz pieprasījumu tikai tad, ja ir notikušas izmaiņas datos vai statusā. Tiklīdz ir notikušas izmaiņas, tiek saņemta atbilde no servera. Šajā brīdī no klienta puses nekavējoties tiek nosūtīts jauns datu pieprasījumus. Ja izmaiņas nenotiek un ir pagājis maksimālais dīkstāves laiks, iestājas savienojuma noildze un tas tiek pārtraukts. Līdz ar to, klientam jānosūta jauns pieprasījums, lai turpinātu saņemt datus. Tātad, klients vienmēr nosūta pieprasījumus un serveris atbild tikai tad, kad ir pieejama jauna informācija, ko padot klientam [4].

Attēlā 3.3.1. redzama garās aptaujas metodes laika skala - klients nosūta sākotnējo datu pieprasījumu serverim, uz ko serveris sniedz atbildi tikai tad, ja pieejami jauni dati, pēc datu saņemšanas nekavējoties tiek nosūtīts jauns pieprasījums.



3.3.1.att. **Garās aptaujas metodes laika skala**

Viens no biežāk izmantotajiem implementācijas variantiem garajai aptaujas metodei ir izmantojot AJAX. Šo pieeju atbalsta visi pārlūki, kam ir XHR atbalsts [6]. Attēlā 3.3.2. redzams implementācijas piemērs no klienta puses. Izmantojot AJAX, vispirms klients nosūta serverim pieprasījumu un tad gaida atbildi, kad tā tiek saņemta, funkcija tiek izsaukta atkārtoti.

```
$.ajax(
    {
        type: 'GET',
        url: 'http://127.0.0.1/php-long-polling/server/server.php',
        data: queryString,
        success: function(data){
            // put result data into "obj"
            var obj = jQuery.parseJSON(data);
            // put the data_from_file into #response
            $('#response').html(obj.data_from_file);
            // call the function again, this time with the timestamp we just got from server.php
            getContent(obj.timestamp);
        }
    }
);
```

3.3.2. att. **Garās aptaujas metodes implementācijas piemērs izmantojot AJAX [7]**

Risinājuma izmantošanas plusi:

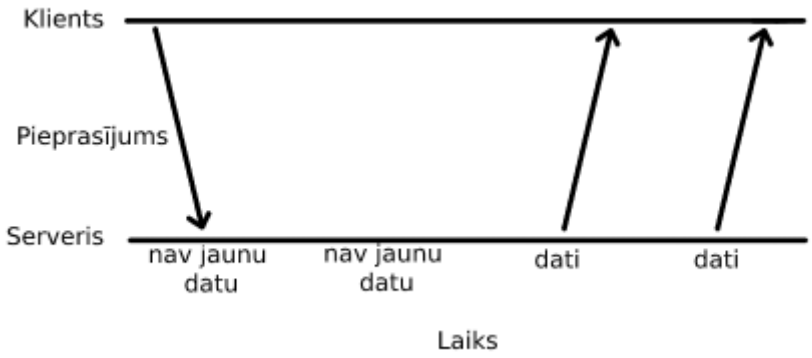
- Dati var tikt saņemti reāllaikā, jo nav jāgaida savienojuma izveide un nav gaidīšanas intervāls,

Risinājuma izmantošanas mīnusi:

- Lai gan garās aptaujas metode resursus izmanto taupīgāk, tik un tā tiek patērēti lieli servera un tīkla resursi [3],
- veiktspēja var pat samazināties, ja pieprasījumu skaits ir liels [6].

3.4. HTTP straumēšana

Abās aptaujas metodēs, kas aprakstītas iepriekšējās nodaļās, pēc datu saņemšanas savienojums tiek slēgts un jauns savienojums tiek izveidots tikai līdz ar jauna pieprasījuma nosūtīšanu. HTTP straumēšanas metodē savienojums tiek saglabāts nenoteiktu laiku. Pēc tam, kad klients ir nosūtījis sākotnējo pieprasījumu, serveris nesniedz atbildi līdz ir pieejami jauni dati, kad parādās dati, tiek nosūtīta atbilde, bet serveris neslēdz savienojumu pēc šo datu nosūtīšanas. Tas ļauj pārlūkam turpināt gaidīt jaunu informāciju. Līdz ar to, nākamajā reizē, kad ir pieejami jauni dati, nav nepieciešams veidot jaunu savienojumu un datu pārraidei tiek izmantots tas pats izveidotais savienojums [8]. Ideālā gadījumā šis savienojums nekad netiek pārtraukts, taču realitātē, var rasties kļūdas, līdz ar to, klientam jāspēj izveidot jaunu savienojumu [5].



3.4.1.att. **HTTP straumēšanas laika skala**

Eksistē divi biežāk sastopamie implementācijas veidi - paslēpta <iframe> elementa izmantošana pārlūkprogrammā un XHR objekta izmantošana vairākdaļu atbildei. Peldošā ietvara elements nav paredzēts datu straumēšanai, tādēļ šo tehniku uzskata par nestandarta apkārtceļu un tā izmantošana nav vēlama. Tāpat tā implementācija lielām sistēmām kļūst ar vien sarežģītāka un to ir grūti mērogot [6].

Risinājuma izmantošanas plusi:

- Pieprasījumiem ir mazs virstēriņš, jo nav jānosūta dati savienojuma izveidei.
- Risinājums ir labs, ja nepieciešams nosūtīt datus ļoti bieži vienam un tam pašam saņēmējam.

Risinājuma izmantošanas mīnusi:

- Notikumiem netiek piedāvātas datu struktūras, ja tās nepieciešamas, tad tās jānodoršina notikumu veicējiem [5].
- Risinājuma implementācija ir sareģīta, tas slikti mērogojas lielam datu apjomam.

3.5. Server-Sent Events

Server-Sent Events ir protokols, kas nodrošina vienvirziena komunikāciju starp serveri un klientu. Tehniski, šī metode ir balstīta uz HTTP straumēšanas modeli, un tajā pārlūks saņem automātiskus servera datu atjauninājumus izmantojot atvērto HTTP savienojumu. W3C ir standartizējuši SSE EventSource API kā daļu no HTML5 [9].

API izmantošanai nepieciešams izveidot EventSource objektu un reģistrēt notikumu uztvērēju, tā realizācijas piemērs redzams attēlā 3.1.5.

```
var source = new EventSource('updates.cgi');
source.onmessage = function (event) {
  alert(event.data);
};
```

3.5.1. att. **Server-Sent Events API izmantošanas piemērs**

SSE definē ziņojumu formātu notikumiem, kas tiek sūtīti no servera. Tam tiek izmantots text/event-stream un tiek nosūtīti MIME tipa dati [10]. Ziņojuma formātu veido teksta līnija, ko atdala rakstzīmju plūsma. Līnijas, kurās ir ziņojuma pamatteksts vai dati, sākas ar data: un beidzas ar \ n \ n simboliem, kā tas redzams attēlā 3.5.2. [9].

```
event: bookavailable\n
data: {"name" : "Game of Thrones"}\n\n

event: newbookadded\n
data: {"name" :"Storm of Swords"}\n\n
```

3.5.2. att. **Server-Sent Events datu piemērs**

Risinājuma izmantošanas plusi:

- Izmantojot šo risinājumu, dati tiek pārraidīti ar mazu latentumu.
- Risinājumu var izmantot ar EventSource API.

Risinājuma izmantošanas mīnusi:

- Serverim nav pieejama informācija, kad klients atslēdzas.

- Nav pieejams nekāds Internet Explorer atbalsts.

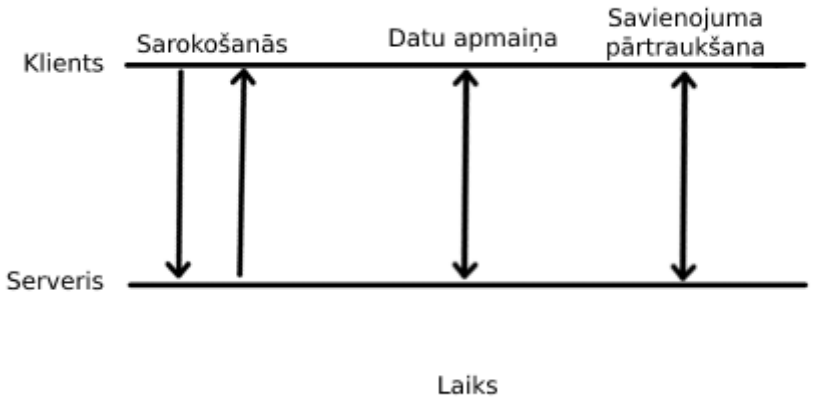
3.6. WebSockets

WebSockets ir TCP protokols, kas nodrošina pilndupleksa savienojumu starp klientu un serveri. Tas nozīmē, ka tiek izveidota divvirzienu datu apmaiņa un abas puses var vienlaicīgi sūtīt datus viena otrai izmantojot vienu TCP savienojumu [11]. Apvienojumā ar WebSocket API, tiek nodrošināta alternatīva HTTP aptaujāšanai ar divvirzienu komunikāciju starp klientu un serveri. WebSocket API atbalsta gandrīz visi pārlūki gan datoros, gan viedierīcēs, izņemot OperaMini, tas ir iekļauts arī HTML5 standartā. [12]

WebSockets ir pieejami divu veidu savienojumi - neaizsargātais (ws://) un TLS aizsargātais (wss://). Protokols sastāv no divām daļām:

- sarokošanās, kuras laikā klients nosūta ziņu serverim, un serveris sniedz sarokošanās atbildi,
- un datu apmaiņas [13].

Attēlā 3.6.1. redzama WebSockets laika skala, kā redzams, vispirms no klienta puses tiek izveidots savienojuma pieprasījums un pēc atbildes saņemšanas var notikt divvirzienu datu apmaiņa, līdz viena no pusēm šo savienojumu pārtrauc.



3.6.1. att. **WebSockets laika skala**

Risinājuma izmantošanas plusi:

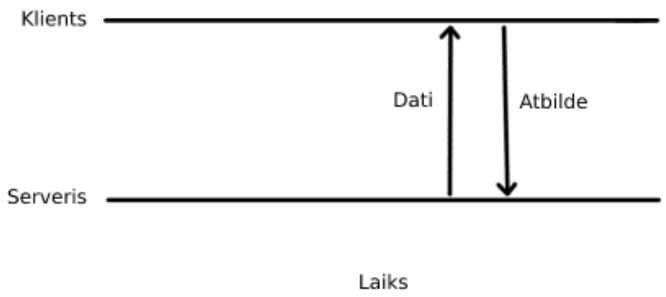
- Pieejams atbalsts lielākajā daļā pārlūkprogrammu.
- Lielākais ieguvums no tīmekļa ligzdu izmantošanas ir izmantoto resursu samazinājums gan klienta, gan servera pusēs [11].

Risinājuma izmantošanas mīnusi:

- Nav pieejama kļūdu apstrāde.

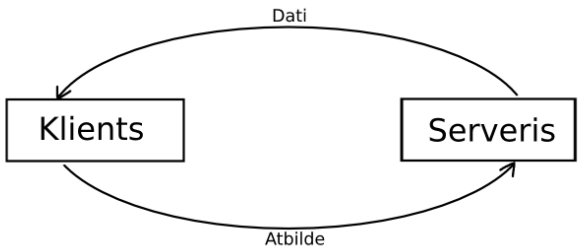
3.7. Tīmekļa aizķeres

Tīmekļa aizķeres ir risinājums, kur, tā vietā, lai klients pieprasītu datus serverim, serveris paziņo klientam, ka ir pieejami jauni dati. Tādā veidā tiek samazināta servera noslodze, jo nav vairāku API klientu, kas regulāri pieprasa datus, jo serveris pats paziņo par aktuālajiem datiem. Kad ir noticis vēlamais notikums, serveris nosūta informāciju par to visiem, kas reģistrēti kā ieinteresēti. Attēlā 3.7.1. redzama laika skala izmantojot tīmekļa aizķeres.



3.7.1. att. **Laika skala izmantojot tīmekļa aizķeres**

Parasti tas ir JSON tipa POST pieprasījumu ar pamattekstu, kas satur visu pieprasīto informāciju [14]. Lai serveris zinātu, ka klients ir saņēmis datus, klienta galapunktam jāatgriež 200 OK HTTP statusa kods, ja šāda atbilde netiks saņemta, API var mēģināt nosūtīt datus atkārtoti [9]. Attēlā 3.7.2. redzams vizuāls attēlojums datu apmaiņai izmantojot tīmekļa aizķeres.



3.7.2. att. **Datu apmaiņa izmantojot tīmekļa aizķeres**

Tīmekļa aizķeres atvieglo vienvirziena saziņu starp serveri un klientu, taču bez tā, nepieciešama servera konfigurācija datu pārraidīšanai, bet klienta pusē jāveic saņemto datu apstrāde. Tīmekļa aizķeres ir labs risinājums serveru datu apmaiņai, bet ne datu atrādīšanai lietotājiem.

3.8. Iespējamo risinājumu salīdzinājums

Salīdzinot visus aplūkotos risinājumus, redzams, ka klasiskā aptaujas metode nav piemērota centralizētās uzdevumu uzskaites risinājumam, jo plānots liels datu apjoms, pieprasījumu biežums var radīt noslodzi un veiktspējas problēmas, tapāt nav iespējas lietotājam paziņot, ka piešķirts jauns pieteikums.

Garā aptaujas metode nesniedz būtiskus ieguvumus, salīdzinot ar klasisko aptaujas metodi, lielam informācijas daudzumam, tā var būt pat sliktāka, jo nodrošina reāllaika datu apmaiņu.

HTTP straumēšana, līdzīgi kā abas iepriekšējās, arī nav paredzēta lielām sistēmām, tās implementācija kļūst arvien sarežģītāka, palielinoties sistēmu skaitam.

Atlikušo 3 risinājumu salīdzinājums redzams tabulā 3.8.1., salīdzinājums veikts pēc 9 kritērijiem:

- Vai risinājums piedāvā asinhronu, reāllaika komunikāciju -vai piegādātie dati tiks saņemti reālā laikā, kas atbilst visiem salīdzinātajiem risinājumiem,
- Vai tiek veidots ilgstošs savienojums datu apmaiņai - tātad, vai jaunu datu saņemšanai nav nepieciešams veidot atkārtotu savienojumu. Tīmekļa aizķeres datus padod tikai notikuma iestāšanās gadījumā, līdz ar to, katram notikumam izpildoties, jānosūta jauns pieprasījums un jāsaņem uz to atbilde.
- Vai izveidotais savienojums ir divvirzienu, kas nozīmē, ka abas puses var vienlaicīgi sūtīt datus viena otrai, tas iespējams tikai izmantojot WebSockets.
- Vai risinājumam ir pieejama kļūdu apstrāde - ja datu apmaiņas laikā notikusi kļūda, vai sistēma turpinās saņemt datus. Tīmekļa aizķerēm tas nav pieejams, taču to salīdzinoši vienkārši var pievienot.
- Vienkārša implementācija un uzturēšana, šajā gadījumā domāta kā nepieciešamība specifisku rīku izmantošanai, lai nodrošinātu datu apmaiņu. Tā kā tīmekļa aizķeres nodrošina ērtu datu apmaiņu starp serveriem, tad šajā gadījumā nepieciešams API un datu apstrāde. Abiem pārējiem risinājumiem, kas nodrošina datu atrādīšanu lietotāju pārlūkos, jāpievērš uzmanība atbalstītajām pārlūkprogrammu versijām.
- Nepieciešams rezerves variants ar aptaujas metodi - ja izmantotais risinājums kādā brīdī nav pieejams, vai funkcionalitātes atjaunošana jānodrošina ar aptaujas metodi, lai nesaņemtu datu gadījumā pilnībā neapturētu sistēmas darbību.
- Vai šis rīks tiek atbalstīts saistīto sistēmu izmantotajās tehnoloģijās, kas aprakstītas nodaļas 1.3. apakšnodaļās.

3.8.1.tabula

Izpētīto risinājumu salīdzinājums

	Server-Sent events	**WebSockets**	**Tīmekļa aizķeres**
Asinhrona reāllaika komunikācija	Jā	Jā	Jā
Ilgstošs savienojums	Jā	Jā	Nē
Divvirzienu savienojums	Nē	Jā	Nē
Kļūdu apstrāde	Jā	Jā	Nē
Vienkārša implementācija un uzturēšana	Nav InternetExplorer atbalsts	Nepieciešams pārlūka un starpniekservera atbalsts	Jā
Nepieciešams rezerves variants ar aptaujas metodi	Nē	Jā	Nē
Jira	Jā	Nē	Jā
IBM	Jā	Jā	Jā
Laravel	Jā	Jā	Jā

Risinājumam ir jābūt atbalstītam Internet Explorer un Google Chrome pārlūkos, kas ir oficiālās uzņēmumā izmantojamās pārlūkprogrammas. Ļoti daudzi darbinieki ikdienā izmanto Internet Explorer pārlūku, tādēļ Server-Sent Events nebūs piemērots risinājums.

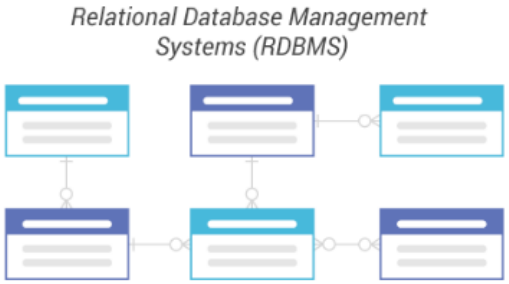
Starp Tīmekļa aizķerēm un WebSockets - katram ir savi plusi un mīnusi, bet vislabākais variants - izmantot abus kopā - tīmekļa aizķeres, lai iegūtu datus no gala sistēmām un websockets šo datu atrādīšanai lietotājiem. Līdz ar to, tiks nodrošināts, ka no visām sistēmām var saņemt datus, un visus datus varēs atrādīt centrālē, jo WebSocket risinājums datus iegūs no atsevišķas datubāzes.

4. DATUBĀZE

Lielā datu apjoma dēļ, centralizētajai uzdevumu uzskaites sistēmai nepieciešama sava datubāze. Šobrīd uzņēmumā tiek izmantotas dažādas datubāzes - gan maksas gan bezmaksas produkti. Kā aprakstīts nodaļas 1.3. apakšnodaļās, saistīto sistēmu datubāzes ir savstarpēji atšķirīgas un ir nepieciešams izvēlēties risinājumu, kurā šos, dažādo datu tipu, datus saglabāt. Nodaļā salīdzinātas divas datubāzu struktūras - relāciju datubāzes ar SQL un nerelāciju datubāzes NoSQL.

4.1. SQL

SQL ir strukturēta vaicājumvaloda, kas paredzēta datubāzu piekļūšanai un manipulēšanai relāciju datubāzu pārvaldības sistēmās (RDBMS) vai straumes apstrādei relāciju datu plūsmas pārvaldības sistēmā (RDSMS). Tās tiek izmantotas, apstrādājot strukturētus datus, kas ietver attiecības starp entītijām un mainīgajiem. Dati tiek glabāti datu bāzes objektos- tabulās. Tabulas ir saistītu datu ierakstu kolekcija, un tās sastāv no kolonnām un rindām [15]. Attēlā 4.1.1. redzams vizuāls relāciju datubāzu piemērs.



4.1.1. att. **RDBMS datu bāzes [16]**

Šo risinājumu ir iespējams izmantot centralizētas sistēmas datubāzei, taču tā tā pēc
savas būtības ir labāk piemērota strukturētiem datiem.

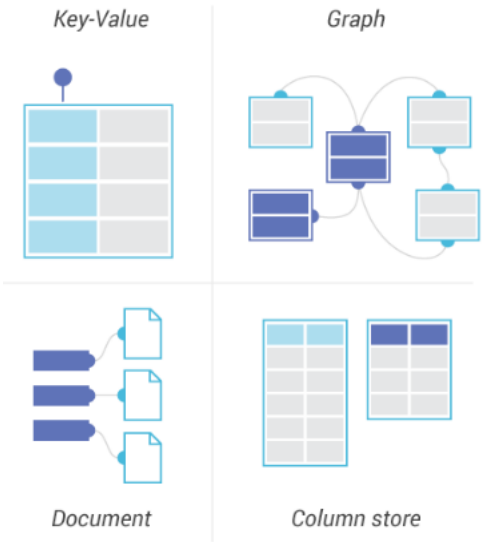
4.2. NoSQL

NoSQL ir nerelāciju datubāzu pārvaldības sistēma. Izmantojot terminu NoSQL parasti tiek domāta jebkura nerelāciju datubāze. Tā kā nerelāciju datubāzes neievēro RDBMS principus, neglabā datus tabulās, tām nav fiksēta shēma un ir salīdzinoši vienkāršs datu modelis, tās ir piemērotas nestrukturētu datu uzglabāšanai [17].

Tiek izdalīti četri dažādi datu glabāšanas veidi:

- Atslēgu-vērtību glabāšana - kā var noprast no nosaukuma, datubāzē tiek glabāti indeksēti atslēgu- vērtību pāri. Atslēgas ir unikāli identifikatori, kas norāda uz attiecīgo vērtību. Dati tajās tiek kārtoti alfabētiskā secībā.
- Dokumentu glabāšana - ieraksti tajās tiek glabāti kā dokumenti, parasti tiek glabāti nestrukturēti teksti vai daļēji strukturēti dokumenti. Tiem iespējams piekļūt, līdzīgi kā atslēgu-vērtību glabāšanā, izmantojot atslēgu, kas reprezentē dokumentu.
- Plašu kolonnu glabāšana - šajā gadījumā tiek izmantotas tabulas, rindas un kolonnas, taču to formāts un nosaukumi var atšķirties vienas tabulas ietvaros.
- Grafu datubāze - atrāda savstarpēji saistītos datus kā loģisku grafu. [18]

Attēlā 4.2.1. redzams šo četru veidu vizuāls salīdzinājums.



4.2.1. att. **NoSQL datu glabāšanas veidi [16]**

Risinājumam piemērotākā ir dokumentu datu bāze, jo, izvēloties tīmekļa aizķeru risinājumu datu saņemšanai, tie tiks saņemti JSON formātā - kā daļēji strukturēti resursi. Dokumentu glabāšanas datubāzes ir paredzētas tieši šādu datu saglabāšanai.

Šobrīd populārākā dokumentu glabāšanas datubāze ir MongoDB - tajā dati tiek glabāti JSON datu tipam līdzīga veida dokumentos. MongoDB nav nepieciešms definēt dokumentu datu struktūras, tie var tikt viegli mainīti un saturēt jebkādu vērtību[19].

5. PIEDĀVĀTIE RISINĀJUMI

Apkopojot visu izpētīto informāciju, tiek piedāvāti divi centralizētās uzdevumu uzskaites sistēmas realizācijas varianti. Pirmais, kas aprakstīts apakšnodaļā 6.1., centrāles realizācija izmantojot gatavu rīku un otrais, aprakstīts nodaļā 6.2., veidot sistēmu izmantojot uzņēmuma resursus.

5.1. Gatava rīka izmantošana

No 2. nodaļā aplūkotajiem risinājumiem, vispiemērotākais ir Zapier integrācijas rīks, procesu automatizēšanai. Tā būtiskākie plusi ir lietotājam draudzīgās saskarnes, kas tā lietošanu padara vieglu un saprotamu, plašā dokumentācija izstrādātajiem, kas atvieglotu savu lietotņu definēšanu. Lai gan tas ir maksas rīks, ja pieņem, ka gala produktu izmantotu 65% uzņēmuma darbinieku, tas izmaksātu tikai 5 eiro/gadā vienam lietotājam, kas ir diezgan maza summa. Tāpat pie mīnusiem tika minēti drošības riski, taču risinājumu izmanto miljoniem klientu visā pasaulē, oficiālajā mājas lapā ir aprakstīti kādi pasākumi tiek veikti datu drošības nodrošināšanai, tādēļ datu noplūdes risks ir ļoti neliels.

5.2. Sistēmas izstrāde uzņēmuma iekšienē

Balstoties uz izpētītajiem un salīdzinātajiem risinājumiem starpsistēmu komunikācijas nodrošināšanai un datu bāzes izvēlei, tiek piedāvāts otrs variants sistēmas, kurā sistēmas izstrāde notiek uzņēmuma iekšienē.

Centralizētās uzdevumu uzskaites sistēmas datu iegūšanai no gala sistēmām izmantot tīmekļa aizķeres. Tās ieviest izstrādājot API, kas balstās uz HTTPS pieprasījumu veikšanu no katras konkrētas gala sistēmas uz centrāli. Gala sistēmām jāpadod šos pieprasījumus datu tipa JSON formātā, reaģējot uz notikumiem sistēmā - pieteikuma izveide, statusa maiņa vai kāda cita darbība, atkarībā no sistēmas. Pieprasījumam jāsatur visi nepieciešamie dati par resursu, lietotāju, kurš veicis darbību un, iespējams, citu specifisku informāciju, kas nepieciešama resursa korektam aprakstam vai pilnīgam kontekstam centrālē.

Saņemto JSON tipa datu glabāšanai izmantot dokumentu glabāšanas datu bāzi MongoDB, kurā no visiem saņemtajiem datiem saglabāt tikai datus derīgos statusos. Lai taupītu resursus, būtu nepieciešams izstrādāt risinājumu, kas pārbaudītu, vai resursu ar saņemto statusu vēl ir nepieciešams atrādīt centrālē, ja nē - to neatgriezeniski dzēst no datubāzes.

Datu atrādīšanu veikt LEports sistēmā, tā veikšanai, klienta pusē jāimplementē uz Node.js balstītā bibliotēka Socket.io, kas no centralizētās uzdevumu uzskaites sistēmas datubāzes ar WebSocket API , reāllaikā atrādīs saņemtos datus lietotājiem. Lietotāju ērtībai, sistēmu var papildināt ar paziņojumu vai signālu, kas norādītu, ka ir piešķirts jauns resurss.

Šāda risinājuma pluss būtu arī iespēja nākotnē diezgan vienkāršā veidā palielināt saistīto sistēmu skaitu. Problēmu gadījumā, ja nav iespējams realizēt augstāk minētās darbības, ir iespējas veikt pielāgojumus centrālās sistēmas pusē, lai samazinātu izstrādes laiku vai risinājumu sarežģītību.

REZULTĀTI

Bakalaura darba izstrādei definētie mērķi un uzdevumi ir izpildīti. Darba gaitā ir izpētīta nepieciešamība centralizētās uzdevumu uzskaites sistēmas ieviešanai, izveidots un apkopots vispārīgs prasību apraksts, veikta saistīto sistēmu analīze - to pielietojums, tehnoloģiju apraksts un daļēji apkopots, kādi dati jāpadod centrālei. Izvēlēta piemērota sistēma, kurā saņemtos datus atrādīt darbiniekiem.

Tāpat izpētīti 3 dažādi gatavie risinājumi, kas var atvieglot centrāles izstrādi vai pat to aizstāt. Visiem risinājumiem izpētītas to piedāvātās iespējas, apkopoti plusi un mīnusi no katra risinājuma ieviešanas.

Veikts pētījums, kurā salīdzinātas pieejamās tehnoloģijas starpsistēmu komunikācijas nodrošināšanai reāllaikā. Ņemts vērā, ka izvēlētais risinājums nevar radīt būtisku noslodzi gala sistēmām. Rezultātā pieņemts lēmums par labākajiem un piemērotākajiem rīkiem, kuri izrādījās tīmekļa aizķeres un websockets.

Izpētītas divas dažādas datubāzu struktūras - relāciju un nerelāciju datu bāzes, no tām izlemts, ka centralizētajai uzdevumu uzskaites sistēmai piemērotāka būtu nerelāciju datubāze, jo, balstoties uz iepriekš pieņemto lēmumu par tīmekļa aizķeru izmantošanu, saņemtie dati būtu nestrukturēti. Līdz ar to, piedāvāts izmantot populārāko dokumentu glabāšanas datubāzi - MongoDB.

Darba beigās piedāvāti divi risinājumi centralizētās uzdevumu uzskaites sistēmas ieviešanai uzņēmumā, kur pirmajā variantā piedāvāts labākais gatavais rīks Zapier, bet otrajā - izpētīto tehnoloģiju izmantošanas apraksts, lai sistēmu varētu izstrādāt uzņēmuma iekšienē.

SECINĀJUMI

Darba gaitā secināts, ka centralizētas uzdevumu uzskaites sistēmas ieviešana varētu atvieglot pašreizējās darbinieku problēmas. Izpētītas uzņēmuma sistēmas un, lai gan sistēmai jāiegūst dati no vairākām nesaistītām sistēmām, eksistē daudzi paņēmieni starpsistēmu komunikācijas nodrošināšanai. Kā pierādījās darba gaitā, lai gan visiem paņēmieniem jāveic viens uzdevums - datu apmaiņa - katrs no tiem ir atšķirīgs un tā implementācija varētu radīt gan pozitīvus, gan negatīvus efektus. Tādēļ izpētes process pirms darba uzsākšanas ir svarīgs kvalitatīvas sistēmas ievešanai.

No iegūtajiem rezultātiem var secināt, ka bakalaura darba izstrāde bijusi veiksmīga, jo gala rezultātā piedāvāti divi realizējami risinājumi, kas spētu nodrošināt vēlamās prasības. Tāpat sistēmas funkcionalitāti nākotnē būtu iespējams papildināt, jo piedāvātie varianti to neierobežo.

Gūts plašs ieskats starpsistēmu komunikācijas nodrošināšanā, lai veiktu datu apmaiņu uz dažādām tehnoloģijām balstītām sistēmām. Darba laikā gūtas zināšanas par sistēmu izstrādes plānošanu, informācijas analīzi un risinājumu salīdzināšanu, kas nākotnē var palīdzēt profesionālajā jomā. Šobrīd, balstoties uz veikto darbu, ir uzsākta sistēmas izstāde.

IZMANTOTĀ LITERATŪRA UN AVOTI

1. Learn key concepts in Zapier [tiešsaiste] – [skatīts 15.04.2020.] Pieejams: https://zapier.com/help/create/basics/learn-key-concepts-in-zapier
2. Huginn dokumentācija [tiešsaiste] – [skatīts 18.04.2020.] Pieejams: https://huginnio.herokuapp.com/documentation
3. Mark Masse. REST API Design Rulebook. O'Reilly Media, Inc., 2011.
4. Loreto, S., Saint-Andre, P., Salsano, S., Wilkins, G.Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP. [tiešsaiste] - [skatīts 24.04.2020.] Pieejams: https://www.ietf.org/rfc/rfc6202.txt
5. Matthias Biehl. Webhooks – Events for RESTful APIs, CreateSpace Independent Publishing Platform, 2017.
6. Eliot Estep. Mobile html5: Efficiency and performance of websockets and server-sent events. 2013
7. Php long polling [tiešsaiste] – [skatīts 28.04.2020.] Pieejams: https://github.com/panique/php-long-polling/
8. R.Hema. COMET, THE REVERSE AJAX MECHANISM [tiešsaiste] – [skatīts 01.05.2020.] Pieejams: https://ideaexchange.uakron.edu/cgi/viewcontent.cgi?referer=https://scholar.google.com/&httpsredir=1&article=1908&context=mechanical_ideas#page=88
9. Bhakti Mehta. RESTful Java Patterns and Best Practices. Packt Publishing, 2014
10. HTML Server-Sent Events API [tiešsaiste] – [skatīts 24.04.2020.] Pieejams: https://html.spec.whatwg.org/multipage/server-sent-events.html#eventsource
11. A Lombardi. WebSocket: lightweight client-server communications. 2015
12. The Web Sockets API [tiešsaiste] – [skatīts 07.05.2020.] Pieejams: https://html.spec.whatwg.org/multipage/web-sockets.html
13. V Pimentel, BG Nickerson. Communicating and Displaying Real-Time Data with WebSocket, IEEE Internet Computing, 2012, 46-47
14. Lorna Jane Mitchell. PHP Web Services, 2nd Edition. O'Reilly 2016
15. Sangeeth Raj. SQL vs NoSQL [tiešsaiste] – [skatīts 18.05.2020.] Pieejams: https://medium.com/@sangeethraaj_70563/sql-vs-nosql-faef10e3852d
16. SQL vs NoSQL Difference [tiešsaiste] – [skatīts 20.05.2020.] Pieejams: https://www.scylladb.com/resources/nosql-vs-sql/
17. C. Győrödi, R. Győrödi, G. Pecherle. A comparative study: MongoDB vs. MySQL 2015

18. V Sharma, M Dave. Sql and nosql databases. International Journal of Advanced Research, 2012
19. NoSQL Databases Explained [tiešsaiste] – [skatīts 22.05.2020.] Pieejams: https://www.mongodb.com/nosql-explained

Bakalaura darbs „Centralizētas uzdevumu uzskaites sistēmas ieviešanas iespējas uzņēmumā" izstrādāts LU Datorikas fakultātē.

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

Autors: _____ Laura Bobule

Rekomendēju/nerekomendēju darbu aizstāvēšanai
Vadītājs: Prof.Mg.Sc.Ing. Haralds Gribusts_____ 25.05.2020.

Recenzents: prof. Juris Borzovs

Darbs iesniegts Datorikas fakultātē 25.05.2020.

Dekāna pilnvarotā persona: vecākā metodiķe Ārija Sproģe _____

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

____.06.2020. prot. Nr. _____

Komisijas sekretārs(-e): _____