LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

# KOMUNIKĀCIJAS RĪKS GALA LIETOTĀJU PLATFORMU TESTĒŠANAI

KVALIFIKĀCIJAS DARBS

Autore: **Karīna Seile-Zundāne**
Studenta apliecības nr.: ks16063
Darba vadītājs: B.dat. Artūrs Vaļenieks

RĪGA 2018

# ANOTĀCIJA

Kvalifikācijas darba mērķis ir izstrādāt komunikācijas rīku gala lietotāju platformu testēšanai - atgriezenisko izsaukumu datu īslaicīgās glabāšanas serveri.

Serveris uzglabā datus par atgriezeniskajiem izsaukumiem, kuri ir nepieciešami testēšanas procesā; tajā ir iespējams arī ievietot jaunus datus, pieprasīt serverī esošos datus vai izdzēst neaktuālos datus.

Rīki ļauj optimizēt gala lietotāju platformu testēšanas procesu, vienuviet uzglabājot gala lietotāju platformu pakalpojumu vērtumus un apstrādājot tos, tādā veidā taupot resursus un laiku, kuri tiek patērēti, izmantojot servera alternatīvas, piemēram, aptaujas metodei. Tas ir pielietojams dažādu gala lietotāju platformu servisu, piemēram, tērzēšanas lietotņu testēšanā.

Atgriezenisko izsaukumu datu īslaicīgās glabāšanas serveris tika izstrādāts, izmantojot JavaScript valodu, kā arī Node.js un Express.js satvarus.

Atslēgvārdi: REST API, testēšana, komandrindas saskarnes rīks, HTTP.

# ABSTRACT

COMMUNICATION TOOL FOR END USER PLATFORM TESTING

The aim of the qualification thesis is to develop the communication tool for end user platform testing - the callback server.

The callback server keeps the data that is used in testing processes. It is possible to add new data, request data from a server's collection, as well as delete outdated callbacks.

The tool allows to improve end user platform testing processes by keeping and processing all end user services' payload in one place. Comparing with the alternatives of the tool, for example, polling method, the server uses less resource and processes received callbacks much faster. It can be used in testing of different end user platform services, for instance, chat applications.

The callback server was developed using JavaScript programming language, as well as Node.js and Express.js frameworks.



Keywords: REST API, testing, CLI tool, HTTP.

# SATURS

# APZĪMĒJUMU SARAKSTS

| Izmantotais termins (tulkojums) | Skaidrojums |
|---|---|
| Aptauja (polling) | Metode, ko izmanto datu apstrādes sistēmās, lai vadītu pieeju kopējai datu pārraides videi. Dators vai datoru tīkla centrālā stacija noteiktā kārtībā apprasa datoram pievienotās ierīces, lai noskaidrotu, vai tām ir informācija, ko nepieciešams nosūtīt. [1] |
| Atgriezeniskais izsaukums (callback) | Programmēšanas funkcija vai cits darbināms kods, kurš tiek padots citam kodam kā arguments vai parametrs. |
| base64 | Bināro datu pieraksta veids - dati tiek konvertēti par ciparu virkni ar bāzi 64. |
| CORS - Cross-origin resource sharing | Mehānisms, kurš izmanto papildu HTTP galvenes, lai ļautu lietotājam piekļūt pie izvēlētajiem resursiem uz domēna pat tad, ja vietne, no kurienes tiek sūtīts pieprasījums, funkcionē uz cita domēna. |
| DRY - “Don’t repeat yourself” | Programmatūras izstrādes princips: izstrādātājam mazāk jāatkārto viens un tas pats kods, tā vietā mēģinot normalizēt un strukturēt to. |
| Express.js | Satvars REST API izstrādei JavaScript valodā. |
| Git | Versiju kontroles sistēma. |
| GitLab | Git repozitoriju pārvaldnieks. |
| HTTP | Hiperteksta transporta protokols, kurš tiek izmantots datu apmaiņai starp pārlūkprogrammām un tīmekļa serveriem. Tas darbojas pēc principa pieprasījums – atbilde, t.i., lietotājs sūta uz serveri pieprasījumu un saņem no servera atbildi. |

| | |
|---|---|
| JavaScript | Skriptu programmēšanas valoda. |
| Jest | Testēšanas satvars JavaScript valodai. |
| JSON | Datu apmaiņas formāts, kurš tiek plaši pielietots JavaScript valodā izstrādātajās sistēmās. |
| Mocha | Testēšanas satvars JavaScript valodai. |
| Nepārtrauktā integrācija (Continuous integration) | Programmatūras izstrādes paņēmiens, kad vismaz reizi dienā visas izstrādājamā produkta darba kopijas tiek sapludinātas vienā atzarā. Tas atzars tiek testēts, lai pēc iepējas ātrāk to atkļūdotu. |
| Node.js | Satvars, kurš ļauj izpildīt JavaScript serveru puses programmēšanai. |
| npm | Moduļu (pakotņu) pārvaldnieks Node.js satvaram. |
| nsp | Node.js rīks, kas ļauj pārbaudīt, vai projektā ir nedroši moduļi (pakotnes). |
| pm2 | Rīks procesu pārvaldīšanai. |
| Postman | Lietojumprogrammu saskarņu izstrādes vide. |
| REST API | Lietojumprogrammatūras saskarne, ar kuras palīdzību izstrādātājs var sūtīt HTTP pieprasījumus un saņemt atbildi. |
| Statusa kods (status code) | Kods, kurš ļauj noteikt, vai HTTP pieprasījums tika veiksmīgi pabeigts. Atkarībā no koda var noteikt, vai pieprasījums tika izpildīts pareizi. |
| Supertest | Testēšanas satvars JavaScript valodai. |
| URL vietrādis, vietrādis (URL) | Resursa adreses standarts tīmeklī. |
| x-www-form-urlencoded | Pieprasījumā nosūtīto datu tips: sūtāmās atslēgas un vērtības tiek kodētas, atdalot katru atslēgas-vērtības pāri ar "&" simbolu, bet atslēgu no vērtības atdalot ar "=" simbolu. |

# IEVADS

*"Kods bez testiem nav tīrs. Nav svarīgi, cik tas ir elegants, lasāms un sasniedzams, ja tam nav testu, tas nevar būt tīrs. "[2]*

*Deivs Tomass*

Testēšana ir ļoti svarīga programmatūras izstrādes sastāvdaļa, it īpaši izstrādājot ar nozari nesaistīto klientu pasūtītos informācijas tehnoloģiju risinājumus. Testēšana dažādos programmatūras izstrādes dzīves cikla posmos ļauj atrast kļūdas un nepilnības pēc iespējas ātrāk un piegādāt pasūtītājam kvalitatīvu produkciju. Kvalitātes nodrošināšanas speciālistu darbs ir ļoti atbildīgs, jo tieši pēc viņu apstiprinājuma izstrādātā programmatūra nokļūst pie pasūtītājiem, un darbietilpīgs, jo ir uzmanīgi jānotestē visa programmatūras pieejamā funkcionalitāte.

Kvalifikācijas darba mērķis ir izstrādāt programmatūras testēšanas uzņēmumam ikdienā nepieciešamo rīku - atgriezenisko izsaukumu datu īslaicīgās glabāšanas serveri. Šis rīks optimizē testēšanas procesu, jo tas vienuviet apkopo gala lietotāju platformu servisu vērtumus un apstrādā tos. Rīks palīdz ātrāk un efektīvāk testēt gala lietotāju platformu pakalpojumus. Piemēram, tas var apstiprināt, ka tērzēšanas lietotnē ziņa tika nogādāta adresātam vai lietotājs no rīta ir saņēmis pašpiegādes paziņojumu par laikapstākļiem Rīgā. Šāda veida mākoņpakalpojumi izmanto atgriezenisko izsaukumu parametrus no lietotājiem, un serveris ļauj tos dabūt ātrāk, nekā izmantojot aptaujas metodi, tajā pašā laikā patērējot mazāk resursu.

Strādājot par kvalitātes novērtēšanas speciālistu, autore kopā ar kolēģiem izsecināja, ka izmantojamais datu īslaicīgās glabāšanas serveris ir novecojis un tam ir nepieciešami uzlabojumi, tāpēc tika izlemts izveidot jaunu atgriezenisko izsaukumu datu īslaicīgās glabāšanas serveri, kas veicinātu autores izpratni par programmēšanu un uzlabotu testēšanas efektivitāti uzņēmumā.

Lai sasniegtu mērķi, tika izvirzīti sekojošie uzdevumi:

1. apgūt JavaScript, Node.js, Express.js, Mocha, Supertest, pm2, npm, nsp, Git, Postman rīkus un tehnoloģijas;
2. izstrādāt programmatūras prasību specifikāciju;
3. izstrādāt programmatūras projektējuma aprakstu, balstoties uz izveidoto prasību specifikāciju;
4. izstrādāt rīku;
5. veikt rīka testēšanu, dokumentēt to;
6. ieviest rīku uzņēmuma darbībā.

Rīka izstrāde notika pēc ūdenskrituma programmatūras izstrādes dzīves cikla modeļa - sākumā tika nodefinētas prasības un izveidots projektējuma apraksts, un pēc šī plāna tika veikta rīka izstrāde. Autores pieredzes trūkuma dēļ notika nelielas novirzes no sākotnējā plāna.

Darbs sastāv no rīka programmatūras prasību specifikācijas, programmatūras projektējuma apraksta, testēšanas dokumentācijas, kā arī no projekta organizācijas, kvalitātes nodrošināšanas, konfigurāciju pārvaldības aprakstiem un darbietilpības novērtēšanas. Pielikumā ir aplūkojami pirmkoda fragmenti.

# 1. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA

## 1.1. Ievads

### 1.1.1. Nolūks

Šīs programmatūras prasību specifikācijas (PPS) nolūks ir noteikt un apkopot visas prasības, kuras ir jārealizē atgriezenisko izsaukumu datu īslaicīgās glabāšanas servera izstrādē.

Šis dokuments ir savstarpējā vienošanās starp pasūtītāju un izpildītāju, un rīks tiks izstrādāts atbilstoši dokumentā noteiktajām prasībām.

### 1.1.2. Darbības sfēra

Atgriezenisko izsaukumu datu īslaicīgās glabāšanas serveris ir komunikācijas rīks gala lietotāju platformu pakalpojumu testēšanai. Tā galvenās funkcijas ir īslaicīga testēšanas procesā atgriezenisko izsaukumu datu uzglabāšana serverī un nepieciešamo datu atgriešana lietotājam pēc pieprasījuma, izmantojot HTTP.

Izstrādājamo produktu ir paredzēts izmantot informācijas tehnoloģiju sfērā, un tā mērķis ir optimizēt un paātrināt gala lietotāju platformu kvalitātes novērtēšanu. Piemēram, tas var tikt izmantots tērzēšanas pakalpojumu vai pašpiegādes ziņojumu testēšanā, serverī uzglabājot datus par nosūtītajām ziņām vai pašpiegādes ziņojumiem un atgriežot tos testēšanas procesā.

Nepieciešamības gadījumā serveri var pielāgot savām vajadzībām un glabāt tajā citus datus, ja tie ir pierakstāmi JSON objektu formātā.

### 1.1.3. Saistība ar citiem dokumentiem

Dokuments tika izstrādāts un noformēts, ievērojot standarta LVS 68:1996 „Programmatūras prasību specifikācijas ceļvedis” [3] prasības un rekomendācijas.

### 1.1.4. Pārskats

Dokuments sastāv no 4 nodaļām:

1. Ievads. Nodaļā tiek aprakstīts dokumenta nolūks, produkta darbības sfēra, kā arī dokumenta struktūra un saistība ar citiem dokumentiem.
2. Vispārējais apraksts. Nodaļā tiek īsumā izklāstīta produkta perspektīva, funkcionalitāte, informācija par rīka potenciālajiem lietotājiem, kā arī definēti rīka ierobežojumi, pieņēmumi un atkarības.
3. Funkcionālās prasības. Nodaļā tiek plašāk aprakstīta rīka funkcionalitāte, funkciju mērķi, ievaddati, apstrāde, izvaddati un kļūdu paziņojumi.
4. Nefunkcionālās prasības. Nodaļa apraksta sistēmas drošības, veiktspējas, uzturamības un izmantojamības prasības.

## 1.2. Vispārējais apraksts

### 1.2.1. Produkta perspektīva

Izstrādājamais produkts ir neatkarīgs un spēj darboties patstāvīgi. Tas nav saistīts ar citām programmatūras sistēmām.

### 1.2.2. Projekta funkcijas

Atgriezenisko izsaukumu datu īslaicīgās glabāšanas serverim tiks nodrošinātas sekojošas funkcijas un metodes:

1. Konfigurācijas modulis
    - Komandrindas argumentu apstrāde
    - Pierakstīšanās informācijas pārbaude
    - Lietojumprogrammas saskarnes palaišana
2. Pieprasījumu modulis
    - Autentifikācija
    - Ļaunprātīgo URL vietrāžu filtrēšana
    - Atbildes nosūtīšana
    - Atrastās entītijas dzēšana
    - OPTIONS pieprasījuma apstrāde (pieejama gan autentificētajiem, gan neautentificētajiem lietotājiem)
    - GET pieprasījuma “Wait-Time” galvenes apstrāde
    - GET pieprasījuma apstrāde (pieejama tikai autentificētajiem lietotājiem)
    - POST pieprasījuma apstrāde (pieejama gan autentificētajiem, gan neautentificētajiem lietotājiem)
    - DELETE pieprasījuma apstrāde (pieejama tikai autentificētajiem lietotājiem)
    - Nederīgo URL vietrāžu pāradresēšana
3. Kešatmiņas modulis
    - Entītijas derīguma termiņa uzstādīšana
    - Entītijas derīguma termiņa dzēšana
    - Pieprasītās entītijas meklēšana kešatmiņā
    - Entītijas esamības kešatmiņā pārbaude
    - Entītijas pievienošana kešatmiņai
    - Entītijas dzēšana no kešatmiņas
    - Pēdējās kešatmiņā ievietotās entītijas meklēšana

- Kešatmiņas dzēšana

### 1.2.3. Lietotāja raksturiezīmes

Rīks ir paredzēts informāciju tehnoloģijas nozares darbiniekiem - kvalitātes nodrošināšanas speciālistiem un programmatūras izstrādātājiem, lai izmantotu testēšanas procesos. Rīka lietotājiem ir jābūt izpratnei par REST API darbības principiem un jāmāk sūtīt HTTP pieprasījumus, ir vēlamas programmēšanas prasmes.

Ir vairākas iespējas, kā sūtīt pieprasījumus serverim (jāizvēlas ērtākais veids atkarībā no rīka lietošanas nolūkiem):

- izmantojot lietojumprogrammatūras saskarņu izstrādes vides, piemēram, Postman;
- uzprogrammējot pieprasījumu;
- sūtot pieprasījumu no pārlūkprogrammas adreses joslas.

Rīka uzturētājiem ir vēlama pieredze darbā ar Node.js, npm un JavaScript, kā arī ar procesu pārvaldības rīku, piemēram, pm2, un JavaScript tīmekļa lietotņu satvaru Express.js. Tā kā rīks ir palaižams no komandrindas, un turpat var arī norādīt rīka konfigurācijas parametrus, uzturētājiem ir vēlama pieredze darbā ar komandrindu. Lai izprastu rīka darbības principus, rīka uzturētājiem ir vēlams iepazīties ar šo dokumentu.

### 1.2.4. Vispārējie ierobežojumi

Rīkam ir sekojošie ierobežojumi:

- rīks var būt palaists tikai no komandrindas;
- lai lietotu atgriezenisko izsaukumu datu īslaicīgās glabāšanas serveri, uz datora, uz kura rīks tiek palaists, ir jābūt uzinstalētam Node.js satvaram un JavaScript pakotņu menedžerim npm;
- rīka darbība tiek pārtraukta līdz ar termināļa loga aizvēršanu vai resursdatora izslēgšanu vai restartēšanu;
- tiek apstrādāti tikai autentificēto lietotāju nosūtītie GET un DELETE pieprasījumi;
- derīgi pieprasījumi ir nosūtāmi tikai no URL vietrāžiem, kuri sastāv no burtiem, cipariem, apakšējās svītras (_) vai punkta;
- rīks pieņem tikai tos pieprasījumus, kuri ir nosūtīti x-www-form-urlencoded formā.

### 1.2.5. Pieņēmumi un atkarības

Tiek pieņemts, ka uz datora, uz kura rīks tiek palaists, ir uzinstalēts Node.js satvars un JavaScript pakotņu pārvaldnieks npm.

Rīks var tikt izmantots lokāli, bet tiek pieņemts, ka tā darbībai nepārtrauktās integrācijas nolūkos ir nepieciešams interneta pieslēgums un domēns.

Rīka darbība tiek pārtraukta līdz ar termināļa loga aizvēršanu, kurā tika palaists rīks, vai datora izslēgšanu vai restartēšanu (sk. nodaļu 1.2.4), tāpēc, ja tas tiek izmantots nepārtrauktās integrācijas ietvaros, tiek pieņemts, ka tas ir jādēmonizē un automātiski jāpalaiž pēc katras resursdatora izslēgšanas vai restartēšanas. Tas ir izdarāms, piemēram, izmantojot pm2 procesu pārvaldnieku Node.js satvaram.

Rīka pieejamība ir atkarīga no resursdatora stāvokļa (vai tas ir ieslēgts), kā arī no interneta pakalpojumu sniedzēja, ja rīks tiek palaists uz domēna.

Tiek pieņemts, ka nosūtāmiem objektiem ir jābūt JSON formātā, bet to iekšējā uzbūve nav stingri definēta.

## 1.3. Funkcionālās prasības

Atgriezenisko izsaukumu datu īslaicīgās glabāšanas servera funkcijas pēc to nozīmes tiek sadalītas 3 moduļos:

- konfigurācijas modulis, kurš ietver sevī rīka konfigurēšanu, palaišanu un ārējo konfigurācijas mainīgo apstrādi;
- pieprasījumu modulis, kurā ir apkopotas lietotāja nosūtīto pieprasījumu apstrādes funkcijas;
- kešatmiņas modulis, kurā ir definētas uz kešatmiņu attiecināmās metodes.

### 1.3.1. Konfigurācijas modulis

Konfigurācijas moduļa funkcijas nodrošina rīka konfigurācijas uzstādīšanu pēc lietotāja preferencēm un rīka palaišanu.

Rīku ir iespējams pielāgot savām vajadzībām, padodot nepieciešamos parametrus komandrindas komandā:

- autentifikācijai nepieciešamie lietotājvārds un parole (parametri --username un --password) - ja tie netiek norādīti, serveris netiek palaists;
- kešatmiņā ievietotās entītijas derīguma termiņš (parametrs --timeout) - skaitlis minūtēs, pēc kura entītija tiek izdzēsta no kešatmiņas;
- porta numurs, kuru izmantos serveris (parametrs --port);
- GET pieprasījuma atbildes gaidīšanas laiks (parametrs --waitTime) - skaitlis sekundēs, pēc kura serveris obligāti atbild, pat ja meklētais objekts netiek atrasts;
- reģistrētāja līmenis (parametrs --logLevel) - norāda uz to, kāda tipa ziņojumi tiks izprintēti reģistrētājā.

Ja neobligātie parametri rīka palaišanas komandā nav norādīti, tiek izmantotas noklusētās vērtības, kas ir norādītas konfigurācijas argumentu apstrādes funkcijā.

Vēlāk komandrindā padotie parametri tiek apstrādāti, kā arī pārbaudīts, vai serverim tika uzstādīts lietotājvārds un parole. Ja serverim ir autentifikācijai nepieciešamā informācija, tas tiek palaists.

Šis modulis ir paredzēts sistēmas uzturētājam, lai sakonfigurētu rīku pēc lietotāju vajadzības, kā arī sekotu līdzi serverī notiekošajiem procesiem.

#### 1.3.1.1. Komandrindas argumentu apstrādes funkcija

| | |
|---|---|
| Klasifikators | CONF.ARG |
| Nosaukums | Komandrindas argumentu apstrādes funkcija |
| Apraksts | Funkcija paredzēta tam, lai iegūtu un apstrādātu lietotāja komandrindā ievadītos konfigurācijas argumentus. |
| Ievads | Funkcijas ievaddati tiek iegūti no lietotāja komandrindā ievadītās rīka palaišanas komandas.<br><br>Obligātie atribūti:<br>1. Lietotājvārds (--username). Patvaļīgā simbolu virkne.<br>2. Parole (--password). Patvaļīgā simbolu virkne.<br><br>Neobligātie atribūti:<br>1. Kešatmiņā ievietotās entītijas derīguma termiņš (--timeout). Vesels skaitlis minūtēs. Noklusētā vērtība: 1 diennakts jeb 1440 minūtes.<br>2. Izmantojamā porta numurs (--port). Vesels skaitlis. Noklusētā vērtība: 9000.<br>3. GET pieprasījuma atbildes gaidīšanas laiks (--waitTime). Vesels skaitlis sekundēs. Noklusētā vērtība: 10 sekundes.<br>4. Reģistrētāja līmenis (--logLevel). Viena no 5 iespējamām simbolu virknēm: “trace”, “debug”, “info”, “warn”, “error”. Noklusētā vērtība: “info”. |
| Apstrāde | Konsolē ierakstītā komanda tiek parsēta, meklējot argumentus ar attiecīgo atslēgu. Ja atribūts ar attiecīgo atslēgu tiek atrasts, tā vērtība tiek ierakstīta konfigurācijas objektā, kurš tiek eksportēts izmantošanai citās rīka funkcijās. Ja arguments ar attiecīgo atslēgu tika norādīts nepareizi vai netika norādīts vispār, neobligāto atribūtu gadījumā tiek izmantota norādītā noklusētā vērtība, bet obligātajiem atribūtiem tiek piešķirta vērtība “undefined” - simbolu virkne. |
| Izvaddati | Funkcijas izvaddati ir eksportējamais modulis JSON objekta formātā ar sekojošu struktūru:<br>1. Kešatmiņā ievietotās entītijas derīguma termiņš (lauks).<br>2. Izmantojamā porta numurs (lauks).<br>3. GET pieprasījuma atbildes gaidīšanas laiks (lauks).<br>4. Reģistrētāja līmeņa uzstādīšana (metode) - reģistrētājam tiek uzlikts norādītais līmenis.<br>5. Lietotājvārds (lauks).<br>6. Parole (lauks). |

| | |
|---|---|
| Kļūdu paziņojumi | - |

#### 1.3.1.2. Autentifikācijas datu validācijas funkcija

| | |
|---|---|
| Klasifikators | CONF.CRED.SPEC |
| Nosaukums | Autentifikācijas datu validācijas funkcija |
| Apraksts | Funkcija paredzēta tam, lai pirms servera palaišanas noskaidrotu, vai ir norādīts autentifikācijai nepieciešamais lietotājvārds vai parole. |
| Ievads | Obligātais atribūts:<br>1. Autentifikācijas atribūts (lietotājvārds vai parole). Patvaļīgā simbolu virkne. |
| Apstrāde | Funkcija pārbauda, vai autentifikācijas atribūts ir norādīts un definēts. |
| Izvaddati | Atgriež būla tipa vērtību. Ja atribūts ir norādīts un definēts, tiek atgriezta vērtība "true", ja nē - "false". |
| Kļūdu paziņojumi | - |

#### 1.3.1.3. Lietojumprogrammas saskarnes palaišanas funkcija

| | |
|---|---|
| Klasifikators | CONF.LAUNCH |
| Nosaukums | Lietojumprogrammas saskarnes palaišanas funkcija |
| Apraksts | Funkcija paredzēta tam, lai izveidotu jaunu lietojumprogrammas saskarni, nokonfigurētu to un palaistu uz attiecīgā localhost porta. |
| Ievads | Funkcijas ievaddati tiek iegūti no funkcijas CONF.ARG eksportētā moduļa.<br><br>Obligātie atribūti:<br>1. Lietotājvārds. Patvaļīgā simbolu virkne.<br>2. Parole. Patvaļīgā simbolu virkne.<br>3. Izmantojamā porta numurs. Vesels skaitlis. |
| Apstrāde | Funkcija pārbauda, vai autentifikācijas dati ir norādīti un definēti, lietotājvārdam un parolei izsaucot funkciju CONF.CRED.SPEC.<br>• Ja tā atgriež vērtību "true", kas nozīmē, ka autentifikācijas dati ir norādīti un atbilst prasībām, tiek izveidota jauna lietojumprogrammas saskarne un tiek piesaistīta ievaddatos padotajam portam.<br>• Ja funkcija atgriež vērtību "false", kas nozīmē, ka autentifikācijas dati nebija norādīti vai pareizi definēti, rīks netiek palaists, un lietotājam tiek izvadīts kļūdas paziņojums Nr. 1. |
| Izvaddati | • Ja apstrādes laikā ir radusies kļūda, tiek izvadīts kļūdas paziņojums Nr. 1.<br>• Ja apstrādes laikā kļūdas nav radušās, izvaddatu nav. |

| | |
|---|---|
| Kļūdu paziņojumi | 1. Auth credentials are not specified (Autentifikācijas dati nav norādīti). |

### 1.3.2. Pieprasījumu modulis

Pieprasījuma moduļa funkcijas nodrošina lietotāja HTTP pieprasījumu apstrādi un atbilžu nosūtīšanu, kā arī servera drošību.

Lietotājs var nosūtīt GET, POST, OPTIONS vai DELETE pieprasījumus, kā arī saņemt atbildes uz pieprasījumiem. Ja pieprasījumi tiek sūtīti no ļaunprātīgiem vai nederīgiem (sk. nodaļu 1.2.4) URL vietrāžiem, tie netiek apstrādāti. Sūtot GET vai DELETE pieprasījumus, lietotājam ir jāpierakstās.

- GET pieprasījums pieprasa serverim sameklēt kešatmiņā entītiju, kuras vērtumam ir pieprasījumā padotās vērtības. Tam ir iespējams norādīt atbildes gaidīšanas laiku galvenē "Wait-Time". Pēc šī laika beigām lietotājs garantēti saņems atbildi, neatkarīgi no tā, vai entītija tika atrasta kešatmiņā. Ja entītija ir atrasta, tā tiek dzēsta no kešatmiņas.
- POST pieprasījums ļauj ievietot entītiju kešatmiņā.
- OPTIONS pieprasījums lietotājam norāda uz pieejamām operācijām, nosūtot attiecīgās galvenes, tomēr tā primārā nozīme ir nodrošināt CORS mehānisma darbību.
- DELETE pieprasījums ļauj atbrīvot kešatmiņu vai tās konkrētus apgabalus.

Šis modulis ir paredzēts rīka lietotājiem - kvalitātes nodrošināšanas speciālistiem vai izstrādātājiem, kuri vēlas notestēt savu programmatūru.

#### 1.3.2.1. Autentifikācijas funkcija

| | |
|---|---|
| Klasifikators | REQ.AUTH |
| Nosaukums | Autentifikācijas funkcija |
| Apraksts | Funkcija paredzēta, lai lietotāji, kuri vēlas nosūtīt GET vai DELETE pieprasījumu, varētu pierakstīties. |
| Ievads | Obligātais atribūts:<br>1. GET vai DELETE pieprasījuma objekts.<br><br>Visi pārējie nepieciešamie dati tiek iegūti no padotā objekta. |
| Apstrāde | Funkcija apstrādā nosūtītā pieprasījuma "Authorization" galveni un pārbauda, vai tā vērtība atbilst funkcijas CONF.ARG eksportētajā modulī norādītajam lietotājvārdam un parolei. |

| | |
|---|---|
| Izvaddati | • Ja lietotājs ir veiksmīgi pierakstījies, viņa veiktais pieprasījums tiek nosūtīts uz serveri.<br>• Ja lietotājam nav izdevies autentificēties, reģistrētājā tiek izdrukāts kļūdu paziņojums nr. 1, un lietotājs saņem no servera atbildi ar 401. statusa kodu un JSON objektu ar kļūdu paziņojumu: {"message":"Access denied.","payload":""}. |
| Kļūdu paziņojumi | 1. Access denied (Piekļuve ir liegta). |

#### 1.3.2.2. Ļaunprātīgo URL vietrāžu filtrēšanas funkcija

| | |
|---|---|
| Klasifikators | REQ.MALWARE |
| Nosaukums | Ļaunprātīgo URL vietrāžu filtrēšanas funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai neļautu veikt pieprasījumus no ļaunprātīgajiem URL vietrāžiem. |
| Ievads | Obligātie atribūti:<br>1. Nosūtītā pieprasījuma objekts.<br>2. Nākamās paredzētās darbības objekts.<br><br>Visi pārējie nepieciešamie dati tiek iegūti no padotā pieprasījuma objekta. |
| Apstrāde | Funkcija tiek izsaukta pirms jebkura pieprasījuma nosūtīšanas uz serveri, lai novērstu ļaunprātīgo URL vietrāžu izmantošanu. Tā dekodē vienoto resursu identifikatoru URL vietrādim, no kura tika nosūtīts pieprasījums. Ja tas izdodas un vietrādis nav ļaunprātīgs, lietotāja pieprasījums tiek nosūtīts uz serveri, pretējā gadījumā lietotāja pieprasījums netiek nosūtīts un viņš saņem attiecīgo atbildi no servera. |
| Izvads | • Ja URL vietrādis, no kura tika nosūtīts pieprasījums, nav ļaunprātīgs, lietotāja pieprasījums tiek nosūtīts uz serveri.<br>• Ja URL vietrādis ir ļaunprātīgs, lietotājs saņem no servera atbildi ar 400. statusa kodu un JSON objektu ar paziņojumu: {"message":"URL is potentially malicious.","payload":{url: ${url}}}. |
| Kļūdu paziņojumi | - |

#### 1.3.2.3. Atbildes sūtīšanas funkcija

| | |
|---|---|
| Klasifikators | REQ.RES |
| Nosaukums | Atbildes sūtīšanas funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai nosūtītu atbildi aktīvajam pieprasījumam. |
| Ievads | Obligātie atribūti:<br>1. Aktīvā pieprasījuma atbildes objekts.<br>2. Atbildes ziņojums. Simbolu virkne. |

| | |
|---|---|
| | 3. Atbildes statusa kods. Vesels skaitlis.<br>4. Vērtums. JSON objekts. |
| Apstrāde | Funkcija pievieno servera atbildei padoto statusa kodu, izveido atbildes entītiju no atbildes ziņojuma un vērtuma, kurš tiks nosūtīts lietotājam atbildes ķermenī, un nosūta atbildi pieprasījumam. |
| Izvads | - |
| Kļūdu paziņojumi | - |

#### 1.3.2.4. Atrastās entītijas dzēšanas funkcija

| | |
|---|---|
| Klasifikators | REQ.REMOVE |
| Nosaukums | Atrastās entītijas dzēšanas funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai izdzēstu no kešatmiņas atrasto entītiju un atgrieztu tās vērtumu lietotājam, nosūtot attiecīgo atbildi. |
| Ievads | Obligātie atribūti:<br>1. Atrastā entītija. JSON objekts.<br>2. Aktīvā pieprasījuma atbildes objekts. |
| Apstrāde | Funkcija izsauc CACHE.OBJ.DELETE atrastajai entītijai, kā arī funkciju REQ.RES, lai atrastās entītijas vērtumu nosūtītu lietotājam. |
| Izvads | Lietotājs saņem no servera atbildi ar 200. statusa kodu un JSON objektu, kurš iekļauj sevī kešatmiņā atrasto objektu: {"message":"Requested object was found.","payload":${object}}. |
| Kļūdu paziņojumi | - |

#### 1.3.2.5. OPTIONS pieprasījuma apstrādes funkcija

| | |
|---|---|
| Klasifikators | REQ.OPTIONS |
| Nosaukums | OPTIONS pieprasījuma apstrādes funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai apstrādātu lietotāja nosūtīto OPTIONS pieprasījumu. |
| Lietotāju grupas | Autentificētie un neautentificētie lietotāji |
| Ievads | Obligātie atribūti:<br>1. Regulārā izteiksme, pēc kuras tiek pārbaudīts pieprasījuma URL vietrādis.<br>2. OPTIONS pieprasījuma objekts. |
| Apstrāde | Funkcija tiek izsaukta tikai tajā gadījumā, ja URL vietrādis atbilst padotajai regulārajai izteiksmei, pretējā gadījumā tiek izsaukta funkcija REQ.ALL.<br>Lietotājam tiek nosūtīta atbilde par to, ka pieprasījums ir saņemts. |

| | |
|---|---|
| Izvads | Lietotājs saņem no servera atbildi ar 200. statusa kodu un JSON objektu ar attiecīgo paziņojumu: {{"message":"Request received.","payload":""}. Atbilde satur galvenes ar atļautajām metodēm, kas ir nepieciešams CORS mehānismam. |
| Kļūdu paziņojumi | - |

#### 1.3.2.6. "Wait-Time" galvenes apstrādes funkcija

| | |
|---|---|
| Klasifikators | REQ.WAIT |
| Nosaukums | "Wait-Time" galvenes apstrādes funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai apstrādātu lietotāja nosūtītā GET pieprasījuma galveni "Wait-Time". |
| Ievads | Obligātais atribūts:<br>1. GET pieprasījuma objekts.<br><br>Visi pārējie nepieciešamie dati tiek iegūti no padotā pieprasījuma objekta. |
| Apstrāde | Funkcija apstrādā pieprasījuma galveni "Wait-Time", kur lietotājs var izmainīt noklusēto, no funkcijas CONF.ARGS iegūto aktīvā GET pieprasījuma atbildes gaidīšanas laiku. Ja galvenes vērtība nav skaitliska vai nav definēta, tā netiek ņemta vērā, pretējā gadījumā padotajam GET pieprasījumam tiks piemērots galvenē norādītais gaidīšanas laiks. |
| Izvads | • Ja galvene tika nosūtīta pareizajā formātā, funkcija atgriež tā vērtību sekundēs.<br>• Ja galvene tika nosūtīta kļūdaini vai netika nosūtīta vispār, funkcija atgriež CONF.ARGS funkcijas eksportējamā modulī norādīto aktīvā GET pieprasījuma derīguma termiņu sekundēs. |
| Kļūdu paziņojumi | - |

#### 1.3.2.7. GET pieprasījuma apstrādes funkcija

| | |
|---|---|
| Klasifikators | REQ.GET |
| Nosaukums | GET pieprasījuma apstrādes funkcija |
| Apraksts | Funkcija paredzēta tam, lai apstrādātu lietotāja nosūtīto GET pieprasījumu. |
| Lietotāju grupas | Autentificētie lietotāji |
| Ievads | Obligātie atribūti:<br>1. Regulārā izteiksme, pēc kuras tiek pārbaudīts pieprasījuma URL vietrādis.<br>2. Autentifikācijas funkcija.<br>3. GET pieprasījuma objekts. |

| | |
|---|---|
| | Visi pārējie nepieciešamie dati tiek iegūti no padotā pieprasījuma objekta. |
| Apstrāde | Funkcija tiek izsaukta tikai tajā gadījumā, ja URL vietrādis atbilst padotajai regulārajai izteiksmei, pretējā gadījumā tiek izsaukta funkcija REQ.ALL. Funkcija netiek izsaukta arī tad, ja netiek veiksmīgi izpildīta autentifikācijas funkcija REQ.AUTH.<br>Tiek izveidota GET pieprasījuma entītija, kurā pieglabāts pats pieprasītais objekts, iegūtais no URL vietrāža vaicājuma daļas, kā arī URL vietrādis, no kura tika veikts pieprasījums.<br>Tiek izsaukta funkcija CACHE.HAS ar tikko izveidoto entītiju, lai noskaidrotu, vai kešatmiņā ir atrodama entītija ar vērtumu, kurā ir pieprasītie parametri. Ja meklējamā entītija ir kešatmiņā, tiek izsaukta funkcija REQ.REMOVE, lai izdzēstu atrasto to no kešatmiņas un atgrieztu entītijas vērtumu lietotājam.<br>Ja pieprasītā entītija netiek atrasts uzreiz, GET pieprasījumam sākas gaidīšanas laika, kurš tiek iegūts no REQ.WAIT funkcijas, atskaite, un pieprasījums tiek pievienots aktīvo pieprasījumu kolekcijai. Ja pa šo laiku entītija ar pieprasīto objektu parādās kešatmiņā, tas tiek atgriezts lietotājam. Ja arī pēc gaidīšanas laika beigām entītija netiek atrasta, GET pieprasījums tiek dzēsts no aktīvo pieprasījumu kolekcijas, un lietotājs saņem attiecīgo atbildi no servera. |
| Izvads | • Ja pieprasītā entītija tiek atrasta, lietotājs saņem no servera atbildi ar 200. statusa kodu un JSON objektu ar kešatmiņā atrastās entītijas vērtumu: {"message":"Requested object was found.","payload": ${object}}.<br>• Ja entītija netiek atrasta arī pēc atbildes gaidīšanas laika beigām, reģistrētājā tiek izdrukāts kļūdu paziņojums nr. 1, un lietotājs saņem no servera atbildi ar 404. statusa kodu un JSON objektu ar kļūdu paziņojumu: {"message":"No results found.","payload":${object}} . |
| Kļūdu paziņojumi | 1. No matching results with url ${url} (Dotajā URL vietrādī pieprasītais objekts netika atrasts). |

### 1.3.2.8. POST pieprasījuma apstrādes funkcija

| | |
|---|---|
| Klasifikators | REQ.POST |
| Nosaukums | POST pieprasījuma apstrādes funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai apstrādātu lietotāja nosūtīto POST pieprasījumu. |
| Lietotāju grupas | Autentificētie un neautentificētie lietotāji |
| Ievads | Obligātie atribūti:<br>1. Regulārā izteiksme, pēc kuras tiek pārbaudīts pieprasījuma URL vietrādis.<br>2. POST pieprasījuma objekts.<br><br>Visi pārējie nepieciešamie dati tiek iegūti no padotā pieprasījuma objekta. |
| Apstrāde | Funkcija tiek izsaukta tikai tajā gadījumā, ja URL vietrādis atbilst padotajai regulārajai izteiksmei, pretējā gadījumā tiek izsaukta funkcija REQ.ALL.<br>Kešatmiņā ievietojamais objekts tiek iegūts no POST pieprasījuma ķermeņa. Tiek izveidota POST objekta entītija, kura iekļauj sevī ievietojamo objektu un URL vietrādi, no kura tika veikts pieprasījums. |

| | |
|---|---|
| | Tiek izsaukta funkcija CACHE.ADD ar izveidoto entītiju. Ja to neizdodas pievienot kešatmiņai, lietotājs saņem attiecīgo atbildi no servera.<br>Ja tas izdodas, tiek pārskatīta aktīvo GET pieprasījumu kolekcija, lai nepieciešamības gadījumā atgrieztu kādam no tiem tikko pievienotās entītijas vērtumu. Ja entītija atbilst kādam GET pieprasījumam, tiek izsaukta funkcija REQ.REMOVE, lai izdzēstu atrasto entītiju no kešatmiņas un atgrieztu tās vērtumu aktīvajam GET pieprasījumam, par ko lietotājs saņem attiecīgo atbildi no servera. Ja tā neatbilst nevienam aktīvajam GET pieprasījumam, lietotājs saņem attiecīgo atbildi no servera par to, ka entītija tika ievietota kešatmiņā. |
| Izvads | • Ja entītijas vērtums tika atgriezts aktīvajam GET pieprasījumam, lietotājs saņem no servera atbildi ar 200. statusa kodu un JSON objektu ar nosūtīto vērtumu: {"message":"Object was sent to active request.","payload":${object}}.<br>• Ja entītja tika ievietots kešatmiņā, lietotājs saņem no servera atbildi ar 201. statusa kodu un JSON objektu ar kešatmiņā ievietotās entītijas vērtumu: {"message":"Object was successfully added.","payload":${object}}.<br>• Ja POST pieprasījuma ķermenī padotais objekts nav derīgs ievietošanai, reģistrētājā tiek izdrukāts kļūdu paziņojums Nr. 1, un lietotājs saņem no servera atbildi ar 400. statusa kodu un JSON objektu ar kļūdu paziņojumu: {"message":"Wrong input.","payload":{}} . |
| Kļūdu paziņojumi | 1. Wrong input in an object ${object} (Objekts nav derīgs). |

#### 1.3.2.9. DELETE pieprasījuma apstrādes funkcija

| | |
|---|---|
| Klasifikators | REQ.DELETE |
| Nosaukums | DELETE pieprasījuma apstrādes funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai apstrādātu lietotāja nosūtīto DELETE pieprasījumu. |
| Lietotāju grupas | Autentificētie lietotāji |
| Ievads | Obligātie atribūti:<br>1. Regulārā izteiksme, pēc kuras tiek pārbaudīts pieprasījuma URL vietrādis.<br>2. Autentifikācijas funkcija.<br>3. DELETE pieprasījuma objekts.<br><br>Visi pārējie nepieciešamie dati tiek iegūti no padotā pieprasījuma objekta. |
| Apstrāde | Funkcija tiek izsaukta tikai tajā gadījumā, ja URL vietrādis atbilst padotajai regulārajai izteiksmei, pretējā gadījumā tiek izsaukta funkcija REQ.ALL. Funkcija netiek izsaukta arī tad, ja netiek veiksmīgi izpildīta autentifikācijas funkcija REQ.AUTH.<br>Tiek izsaukta funkcija CACHE.CLEAR ar pieprasījuma URL vietrādi. |
| Izvads | Pēc veiksmīgas kešatmiņas atbrīvošanas lietotājs saņem no servera atbildi ar 200. statusa kodu un JSON objektu ar paziņojumu: {"message":"Cache was successfully cleared.","payload":""}. |

| | |
|---|---|
| Kļūdu paziņojumi | - |

#### 1.3.2.10. Nederīgo URL vietrāžu pāradresēšanas funkcija

| | |
|---|---|
| Klasifikators | REQ.ALL |
| Nosaukums | Nederīgo URL vietrāžu pāradresēšanas funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai pāradresētu nederīgos URL vietrāžus un neļautu sūtīt no tiem pieprasījumus. |
| Ievads | Obligātie atribūti:<br>1. Regulārā izteiksme, pēc kuras tiek pārbaudīts pieprasījuma URL vietrādis.<br>2. Jebkura pieprasījuma objekts. |
| Apstrāde | Ja URL vietrādis, no kura tika nosūtīts pieprasījums, neatbilst regulārajai izteiksmei, lietotājam tiek nosūtīta atbilde par to, ka URL vietrādis ir nederīgs. |
| Izvads | Lietotājs saņem no servera atbildi ar 400. statusa kodu un JSON objektu ar attiecīgo paziņojumu par URL vietrāža nederīgumu: {"message":"Wrong URL.","payload":{"url":${url}}}. |
| Kļūdu paziņojumi | - |

### 1.3.3. Kešatmiņas modulis

Kešatmiņas moduļa funkcijas ir paredzētas iekšējām kešatmiņas metodēm. Ir iespējams pārbaudīt, vai kešatmiņā glabājas nepieciešamā entītija, atrast to, pievienot un izdzēst entītiju no kešatmiņas, iegūt pēdējo ievietoto entītiju, kā arī pilnīgi vai daļēji atbrīvot kešatmiņu.

Tāpat kā aktīvajiem pieprasījumiem (sk. nodaļu 1.3.2.), kešatmiņā ievietotajām entītijām tiek piešķirts derīguma termiņš - laiks, pēc kura entītija tiek automātiski izdzēsta no kešatmiņas. Ja entītija tiek atrasta un padota pēc pieprasījuma pirms šī laika beigām, derīguma termiņš tiek anulēts un izdzēsts.

Šis modulis ir rīka iekšējais modulis, kuram nevar piekļūt neviens rīka lietotājs.

#### 1.3.3.1. Entītijas derīguma termiņa uzstādīšanas funkcija

| | |
|---|---|
| Klasifikators | CACHE.SET.TIME |
| Nosaukums | Entītijas derīguma termiņa uzstādīšanas funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai uzstādītu entītijai derīguma termiņu - laiku, pēc kura tā tiks automātiski izdzēsta no kešatmiņas. |

| | |
|---|---|
| Ievads | Obligātie atribūti:<br>1. Entītija, kurai tiks uzstādīts derīguma termiņš. JSON objekts.<br>2. Kešatmiņa. JSON objektu masīvs.<br>3. Derīguma termiņu kolekcija. JSON objektu masīvs. |
| Apstrāde | Funkcijā padotajai entītijai tiek uzstādīts derīguma termiņš minūtēs, kurš tiek iegūts no funkcijas CONF.ARG eksportējamā moduļa. Pēc norādītā laika dotā entītija tiek izdzēsts no kešatmiņas. Derīguma termiņu kolekcijā tiek ievietota entītija, kura ietver sevī kešatmiņas entītiju, kā arī derīguma termiņa objektu - tā unikālo identifikatoru. |
| Izvads | - |
| Kļūdu paziņojumi | - |

#### 1.3.3.2. Entītijas derīguma termiņa dzēšanas funkcija

| | |
|---|---|
| Klasifikators | CACHE.DELETE.TIME |
| Nosaukums | Entītijas derīguma termiņa dzēšanas funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai izdzēstu uzstādīto entītijas derīguma termiņu specifiskos gadījumos, piemēram, kad kešatmiņā ievietotā entītija tiek atgriezta kādam GET pieprasījumam pirms tā derīguma termiņa beigām. |
| Ievads | Obligātie atribūti:<br>1. Entītija, kurai tika uzstādīts derīguma termiņš. JSON objekts.<br>2. Derīguma termiņu kolekcija. JSON objektu masīvs. |
| Apstrāde | Tiek pārbaudīts, vai padotajai entītijai eksistē derīguma termiņš padotajā kolekcijā. Ja tas eksistē, termiņš tiek atcelts, un ieraksts par to tiek izdzēsts no derīguma termiņu kolekcijas. |
| Izvads | - |
| Kļūdu paziņojumi | - |

#### 1.3.3.3. Entītijas meklēšanas funkcija

| | |
|---|---|
| Klasifikators | CACHE.FIND |
| Nosaukums | Entītijas meklēšanas funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai atrastu kešatmiņā pirmo sastopamo entītiju ar noteiktajām atslēgām vai atslēgas-vērtības pāriem. |
| Ievads | Obligātie atribūti:<br>1. Meklējamā entītija. JSON objekts.<br>2. Kešatmiņa. JSON objektu masīvs. |

| | |
|---|---|
| Apstrāde | Funkcija meklē pirmo kešatmiņā ievietoto entītju, kurā ir sastopami visi meklējamās entītijas atslēgas-vērtības pāri. Ja meklējamās entītijas vērtumā ir atribūti, kuriem ir tikai atslēga, tiek meklēts pirmā entītija, kurai eksistē šāda atslēga, neatkarīgi no tai piesaistītās vērtības.<br>Tiek pārbaudīts, vai entītijai kešatmiņā un meklējamajai entītijai sakrīt pieprasījuma URL vietrādis. Piemēram, ja pieprasījums tika veikts no vietrāža localhost:9000/test, tas tiks meklēts tikai starp tām kešatmiņas entītijām, kuras tika ievietotas kešatmiņā, izmantojot vietrādi localhost:9000/test. |
| Izvads | Ja entītija ar padotajiem parametriem tiek atrasta kešatmiņā, funkcija atgriež tās indeksu kešatmiņā. Ja tā netiek atrasta, tiek atgriezts skaitlis -1. |
| Kļūdu paziņojumi | - |

#### 1.3.3.4. Entītijas esamības kešatmiņā pārbaudes funkcija

| | |
|---|---|
| Klasifikators | CACHE.HAS |
| Nosaukums | Entītijas esamības kešatmiņā pārbaudes funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai noskaidrotu, vai entītija ar padotajiem atslēgas-vērtības parametriem ir kešatmiņā. |
| Ievads | Obligātais atribūts:<br>1. Meklējamā entītija. JSON objekts. |
| Apstrāde | Funkcija izsauc CACHE.FIND funkciju ar padoto entītiju un tās atgriežamo vērtību ievieto būla funkcijā. |
| Izvads | Tiek atgriezta būla tipa vērtība. Ja entītija tika atrasta kešatmiņā, tiek atgriezta vērtība “true”, ja nē - “false”. |
| Kļūdu paziņojumi | - |

#### 1.3.3.5. Entītijas pievienošanas funkcija

| | |
|---|---|
| Klasifikators | CACHE.ADD |
| Nosaukums | Entītijas pievienošanas funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai ievietotu entītiju kešatmiņā. |
| Ievads | Obligātais atribūts:<br>1. Pievienojamā entītija. JSON objekts. |
| Apstrāde | Funkcijā tiek pārbaudīts, vai entītijas vērtums ir JSON objekts un vai tas tukšs. Ja vērtums ir netukšs JSON objekts, entītija tiek ievietota kešatmiņā, un tai tiek uzstādīts derīguma termiņš, izsaucot funkciju CACHE.SET.TIME. Pārējos |

| | |
|---|---|
| | gadījumos entītija netiek ievietots kešatmiņā, un reģistrētājā tiek parādīts attiecīgais kļūdas paziņojums. |
| Izvads | Funkcija atgriež būla tipa mainīgo:<br>• Ja padotās entītijas vērtums ir netukšs JSON objekts, pēc entītijas ievietošanas kešatmiņā tiek atgriezta vērtība "true".<br>• Ja padotās entītijas vērtums ir tukšs objekts, tiek atgriezta vērtība "false" un reģistrētājā tiek izvadīts kļūdu paziņojums nr. 1.<br>• Ja padotās entītijas vērtums nav objekts, tiek atgriezta vērtība "false" un reģistrētājā tiek izvadīts kļūdu paziņojums nr. 2. |
| Kļūdu paziņojumi | 1. Object ${object} cannot be added to cache (empty object) (Tukšs objekts nevar tikt ievietots kešatmiņā).<br>2. Object ${object} cannot be added to cache (not an object) (Atribūts, kura tips nav objekts, nevar tikt ievietots kešatmiņā). |

#### 1.3.3.6. Entītijas dzēšanas funkcija

| | |
|---|---|
| Klasifikators | CACHE.OBJ.DELETE |
| Nosaukums | Entītijas dzēšanas funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai izdzēstu entītiju no kešatmiņas. |
| Ievads | Obligātais atribūts:<br>1. Dzēšamā entītija. JSON objekts. |
| Apstrāde | Tiek izsaukta CACHE.FIND funkcija, kura pārbauda, vai kešatmiņā ir entītija, kura kurā ir sastopami visi dzēšamās entītijas atslēgas-vērtības.<br>Ja entītija tika atrasta, tiek izdzēsts tās derīguma termiņš, izsaucot funkciju CACHE.DELETE.TIME, un arī pati entītija tiek izdzēsta no kešatmiņas. Ja tā netika atrasta, tiek atgriezts kļūdas paziņojums. |
| Izvads | • Ja apstrādes laikā dzēšamā entītija tika atrasta, funkcija atgriež to.<br>• Ja apstrādes laikā dzēšamā entītija netika atrasts, tiek atgriezts kļūdas paziņojums nr. 1. Tas tiek izprintēts arī reģistrētājā. |
| Kļūdu paziņojumi | 1. No objects with such parameters to remove (Kešatmiņā nav dzēšamo objektu ar dotajiem parametriem). |

#### 1.3.3.7. Pēdējās kešatmiņā ievietotās entītijas atgriešanas funkcija

| | |
|---|---|
| Klasifikators | CACHE.PEEK |
| Nosaukums | Pēdējās kešatmiņā ievietotās entītijas atgriešanas funkcija |
| Apraksts | Funkcija paredzēta tam, lai atgrieztu pēdējo kešatmiņā pievienoto entītiju. |
| Ievads | - |
| Apstrāde | Tiek pārbaudīts, vai kešatmiņa ir tukša. |

| | |
|---|---|
| Izvads | • Ja kešatmiņa nav tukša, funkcija atgriež entītijas, kurai ir vislielākais indekss kešatmiņā, vērtumu.<br>• Ja kešatmiņa ir tukša, funkcija atgriež tukšo objektu. |
| Kļūdu paziņojumi | - |

#### 1.3.3.8. Kešatmiņas dzēšanas funkcija

| | |
|---|---|
| Klasifikators | CACHE.CLEAR |
| Nosaukums | Kešatmiņas dzēšanas funkcija |
| Apraksts | Funkcija ir paredzēta tam, lai izdzēstu visas entītijas, kuras tika ievietotas, izmantojot konkrēto URL vietrādi, no kešatmiņas. |
| Ievads | Obligātais atribūts:<br>1. URL vietrādis. |
| Apstrāde | Ja URL vietrādis ietver sevī tikai slīpsvītru, no kešatmiņas tiek izdzēstas visas entītijas, kā arī tiek izdzēsti entītiju derīguma termiņi.<br>Pretējā gadījumā no kešatmiņas tiek izdzēstas visas tās entītijas, kuras tika pievienotas kešatmiņai, izmantojot ievaddatos padoto vietrādi vai tā "apakšvietrādi". Tiek izdzēsti arī entītiju derīguma termiņi. Piemēram, ja funkcijai tiek padota saite .../test, un kešatmiņā ir entītijas, kuru vērtums tika ievietots no saitēm .../test un .../test/test2, entītijas ar abiem vietrāžiem tiks izdzēstas no kešatmiņas. |
| Izvads | - |
| Kļūdu paziņojumi | - |

## 1.4. Nefunkcionālās prasības

### 1.4.1. Veiktspēja

Servera atbildes laiks, izpildot GET pieprasījumu, nedrīkst pārsniegt konfigurācijā norādīto vērtību vairāk kā par 1 sekundi, savukārt pārējiem pieprasījumiem atbildes laiks nedrīkst pārsniegt 1 sekundi.

### 1.4.2. Drošība

Rīka konfigurācijas parametriem, kuri tiek padoti komandrindā, nedrīkst būt "cieti" iekodētiem pirmkodā, izņemot noklusētās vērtības.

Izstrādātajam rīkam jānodrošina aizsardzība pret ļaunprātīgo URL vietrāžu pieprasījumiem.

Lai nosūtītu GET un DELETE pieprasījumus, kuri var ietekmēt kešatmiņā glabājamos ierakstus, ir jābūt piekļuvei tikai autentificētajiem lietotājiem. Šiem nolūkiem ir jāizmanto CORS metode.

Galvenē nosūtāmiem autentifikācijas datiem ir jābūt šifrētiem ar base64 metodi.

Rīka pirmkodam nedrīkst saturēt importētos Node.js moduļus, kuriem tika konstatēti drošības pārkāpumi. Tas ir pārbaudāms, izmantojot Node.js komandrindas rīku nsp.

Tā kā rīks tiek paredzēts izmantošanai nepārtrauktās integrācijas testēšanā, un dati mēdz ātri zaudēt aktualitāti, nekādas prasības pret rezerves kopiju veidošanu netiek izvirzītas.

### 1.4.3. Uzturamība

Rīkam jābūt izstrādātam tā, lai nepieciešamības gadījumā būtu iespējams to papildināt ar jaunu funkcionalitāti vai izlabot esošo, minimāli mainot pirmkodu. Šī iemesla dēļ tiek pieņemts, ka rīkam jābūt uzbūvētam modulāri, lai nepieciešamības gadījumā ieviestu jaunu funkcionalitāti, izveidojot jaunu moduli.

Rīka dokumentācijai ir jāatbilst aktuālajai rīka versijai. Tā regulāri ir jāatjaunina un jāpārskata pie jebkurām strukturālajām izmaiņām.

### 1.4.4. Izmantojamība

Rīkam jābūt dēmonizētam uz resursdatora, kā arī jānodrošina rīka automātiskā palaišana pēc katras resursdatora izslēgšanas vai restartēšanas, ja tas tiek izmantots nepārtrauktās integrācijas procesos.

# 2. PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS

## 2.1. Ievads

### 2.1.1. Nolūks

Šī programmatūras projektējuma apraksta (PPA) nolūks ir noteikt un apkopot visas prasības, pēc kurām tiks projektētas un realizētas atgriezenisko izsaukumu datu īslaicīgās glabāšanas servera funkcijas.

Šis dokuments ir paredzēts dokumentā aprakstītās sistēmas izstrādātājiem.

### 2.1.2. Darbības sfēra

Izstrādājamais produkts ir atgriezenisko izsaukumu datu īslaicīgās glabāšanas datu īslaicīgās uzglabāšanas serveris, kurš komunicē ar lietotāju, izmantojot HTTP. Tajā var ievietot jaunus datus, pieprasīt datus no serverī uzglabājamās kolekcijas, kā arī izdzēst nevajadzīgos datus.

Produkta mērķis ir optimizēt testēšanas procesus un paātrināt gala lietotāju platformu kvalitātes novērtēšanu. Ar šo rīku ir iespējams testēt, piemēram, tērzēšanas programmatūru vai pašpiegādes ziņojumus.

Rīks ir paredzēts izmantošanai informācijas tehnoloģiju nozarē.

### 2.1.3. Saistība ar citiem dokumentiem

Dokuments tika izstrādāts un noformēts, ievērojot standarta LVS 72:1996 “Ieteicamā prakse programmatūras projektējuma aprakstīšanai” [4] prasības un rekomendācijas.

### 2.1.4. Pārskats

Dokuments sastāv no 4 nodaļām:

1. Ievads. Nodaļā tiek aprakstīts dokumenta nolūks, produkta darbības sfēra, kā arī dokumenta struktūra un saistība ar citiem dokumentiem.
2. Dekompozīcijas apraksts. Nodaļā tiek aprakstīti rīka moduļi, to nolūks un funkcijas, kā arī datu dekompozīcijas apraksts, kurā tiek pieminētas rīkā izmantojamās kolekcijas un to entītiju struktūra.
3. Moduļu atkarības. Nodaļā tiek attēlotas moduļu savstarpējās atkarības, izmantojot datu plūsmu diagrammas.
4. Lietošanas diagrammas. Nodaļa satur lietošanas diagrammas, kuras ļauj labāk izprast servera izmantošanas scenārijus.

## 2.2. Dekompozīcijas apraksts

### 2.2.1. Moduļu dekompozīcija

Nodaļā tiek aprakstīts sistēmas sadalījums moduļos, katra moduļa nolūks un programmatūras prasību specifikācijā definēto prasību realizācija katrā modulī. Atkarības starp moduļiem ir apskatāmas nodaļā 2.3.

#### 2.2.1.1. Konfigurācijas modulis

| | |
|---|---|
| Moduļa nolūks | Modulis ir paredzēts rīka konfigurēšanai, apstrādājot un validējot komandrindā padotos parametrus, un palaišanai uz resursdatora. |
| Moduļa funkcijas | 1. Komandrindas argumentu apstrāde (CONF.ARG)<br>2. Pierakstīšanās informācijas pārbaude (CONF.CRED.SPEC)<br>3. Lietojumprogrammas saskarnes palaišana (CONF.LAUNCH) |

#### 2.2.1.2. Pieprasījumu modulis

| | |
|---|---|
| Moduļa nolūks | Modulis ir paredzēts pieprasījumu apstrādei un atbilžu nosūtīšanai, kā arī servera un tajā esošo datu drošības risku novēršanai. |
| Moduļa funkcijas | 1. Autentifikācija (REQ.AUTH)<br>2. Ļaunprātīgo URL vietrāžu filtrēšana (REQ.MALWARE)<br>3. Atbildes nosūtīšana (REQ.RES)<br>4. Atrastās entītijas dzēšana (REQ.REMOVE)<br>5. OPTIONS pieprasījuma apstrāde (REQ.OPTIONS)<br>6. GET pieprasījuma “Wait-Time” galvenes apstrāde (REQ.WAIT)<br>7. GET pieprasījuma apstrāde (REQ.GET)<br>8. POST pieprasījuma apstrāde (REQ.POST)<br>9. DELETE pieprasījuma apstrāde (REQ.DELETE)<br>10. Nederīgo URL vietrāžu pāradresēšana (REQ.ALL) |

#### 2.2.1.3. Kešatmiņas modulis

| | |
|---|---|
| Moduļa nolūks | Modulis ir paredzēts iekšējām kešatmiņas metodēm, kurām nevar piekļūt sistēmas uzturētāji un lietotāji. Moduļa funkcijas izmaina kešatmiņas saturu un struktūru. |
| Moduļa funkcijas | 1. Entītijas derīguma termiņa uzstādīšana (CACHE.SET.TIME)<br>2. Entītijas derīguma termiņa dzēšana (CACHE.DELETE.TIME)<br>3. Pieprasītās entītijas meklēšana kešatmiņā (CACHE.FIND)<br>4. Entītijas esamības kešatmiņā pārbaude (CACHE.HAS)<br>5. Entītijas pievienošana kešatmiņai (CACHE.ADD)<br>6. Entītijas dzēšana no kešatmiņas (CACHE.OBJ.DELETE)<br>7. Pēdējās kešatmiņā ievietotās entītijas meklēšana (CACHE.PEEK)<br>8. Kešatmiņas dzēšana (CACHE.CLEAR) |

### 2.2.2. Datu dekompozīcija

Rīkā nav paredzēts lietot datu bāzi, tā vietā izmantojot kolekcijas - viendimensijas masīvus, kuros tiek uzglabātas entītijas ar noteiktu struktūru. Šajā nodaļā tiks aprakstītas kolekcijas un to entītijas.

Par entītiju rīka kontekstā tiek uzskatīts JSON objekts.

*Tab. 2.1.*

**Rīkā izmantojamo kolekciju un to entītiju apraksts**

| **Kolekcijas nosaukums** | **Kolekcijas nozīme** | **Entītijas struktūra** |
|---|---|---|
| Cache | Glabā serverī ievietotos objektus. | Entītija sastāv no 2 laukiem:<br>• kešatmiņā ievietotais JSON objekts;<br>• URL vietrādis, no kura tika nosūtīts objekts. |
| Timeouts | Glabā objektu derīguma termiņus. | Entītija sastāv no 2 laukiem:<br>• kešatmiņā ievietotais JSON objekts;<br>• derīguma termiņa JSON objekts - unikālais identifikators. |
| Active requests | Glabā aktīvos pieprasījumus, kuriem vēl nav beidzies servera atbildes gaidīšanas laiks. | Entītija sastāv no 4 laukiem:<br>• pieprasītais JSON objekts (uzbūve ir identiska Cache kolekcijas entītijas uzbūvei);<br>• pieprasījuma JSON objekts;<br>• atbildes JSON objekts;<br>• servera atbildes gaidīšanas laika JSON objekts - unikālais identifikators. |

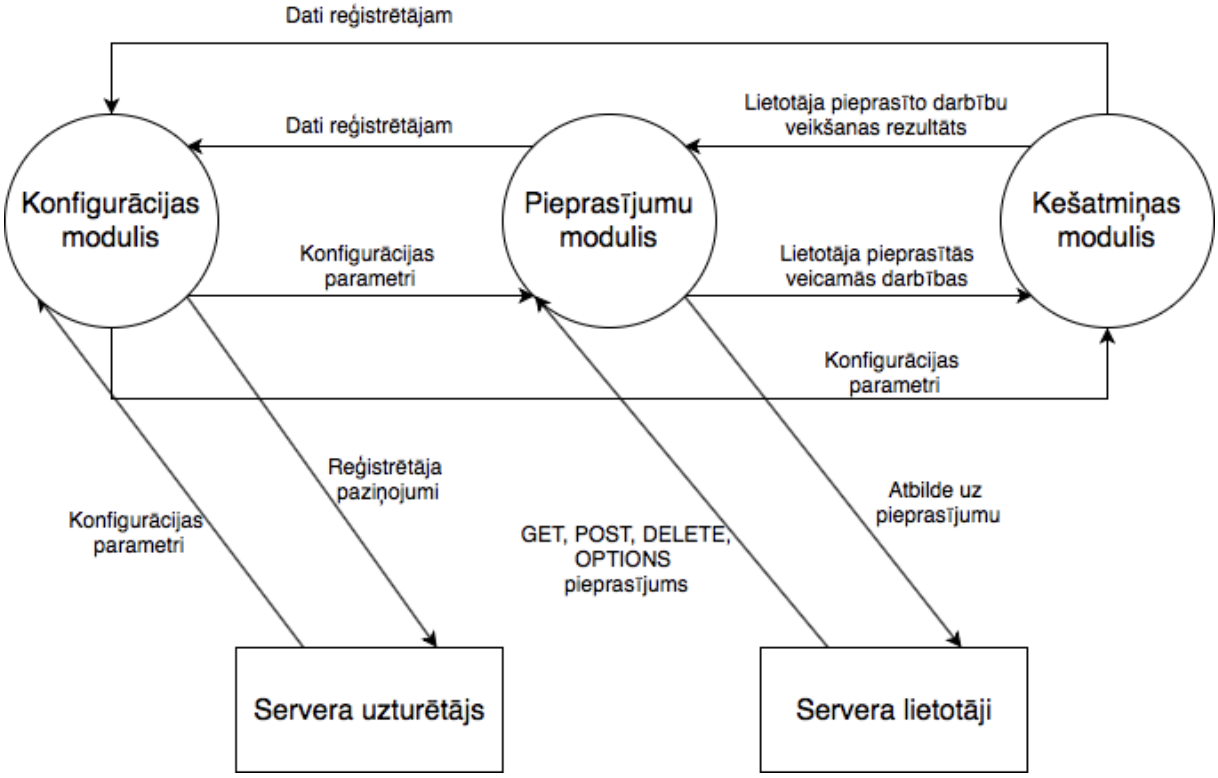
## 2.3. Moduļu atkarības

Atkarības starp moduļiem, kā arī rīka saistība ar lietotājiem tiek attēlota, izmantojot datu plūsmu diagrammas.

### 2.3.1. 0. līmeņa datu plūsmas diagramma

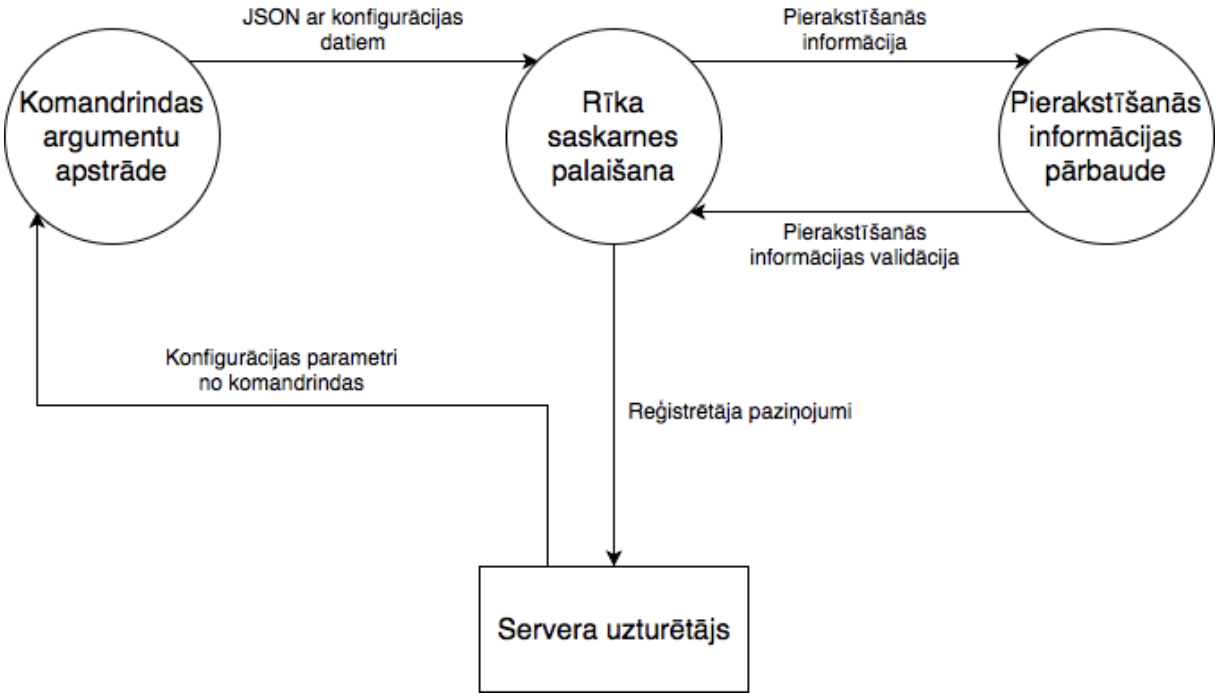

*Att. 2.1.* **0. līmeņa datu plūsmu diagramma**

### 2.3.2. 1. līmeņa datu plūsmas diagramma



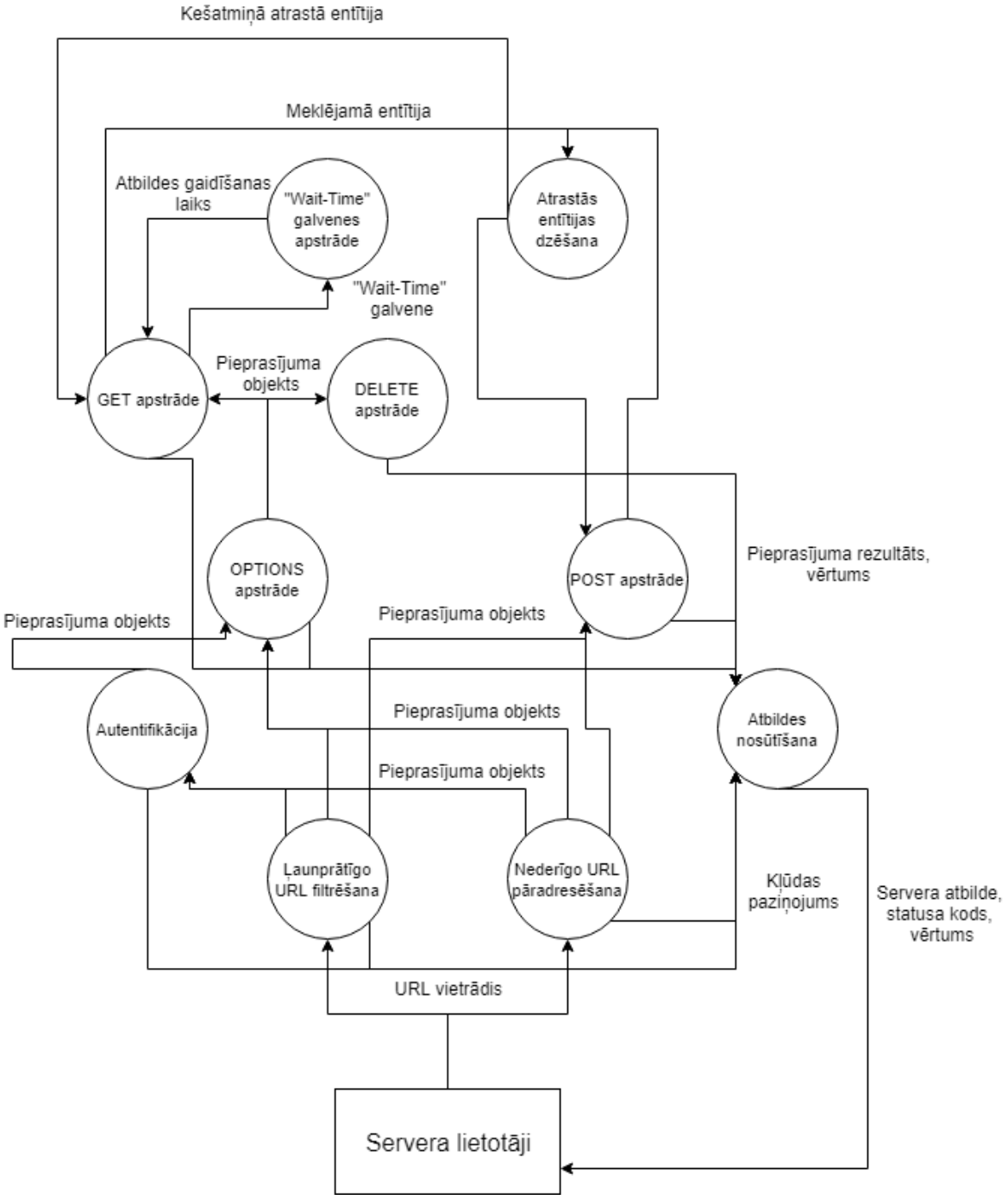

*Att. 2.2.* **1. līmeņa datu plūsmu diagramma**

### 2.3.3. 2. līmeņa konfigurācijas moduļa datu plūsmu diagramma



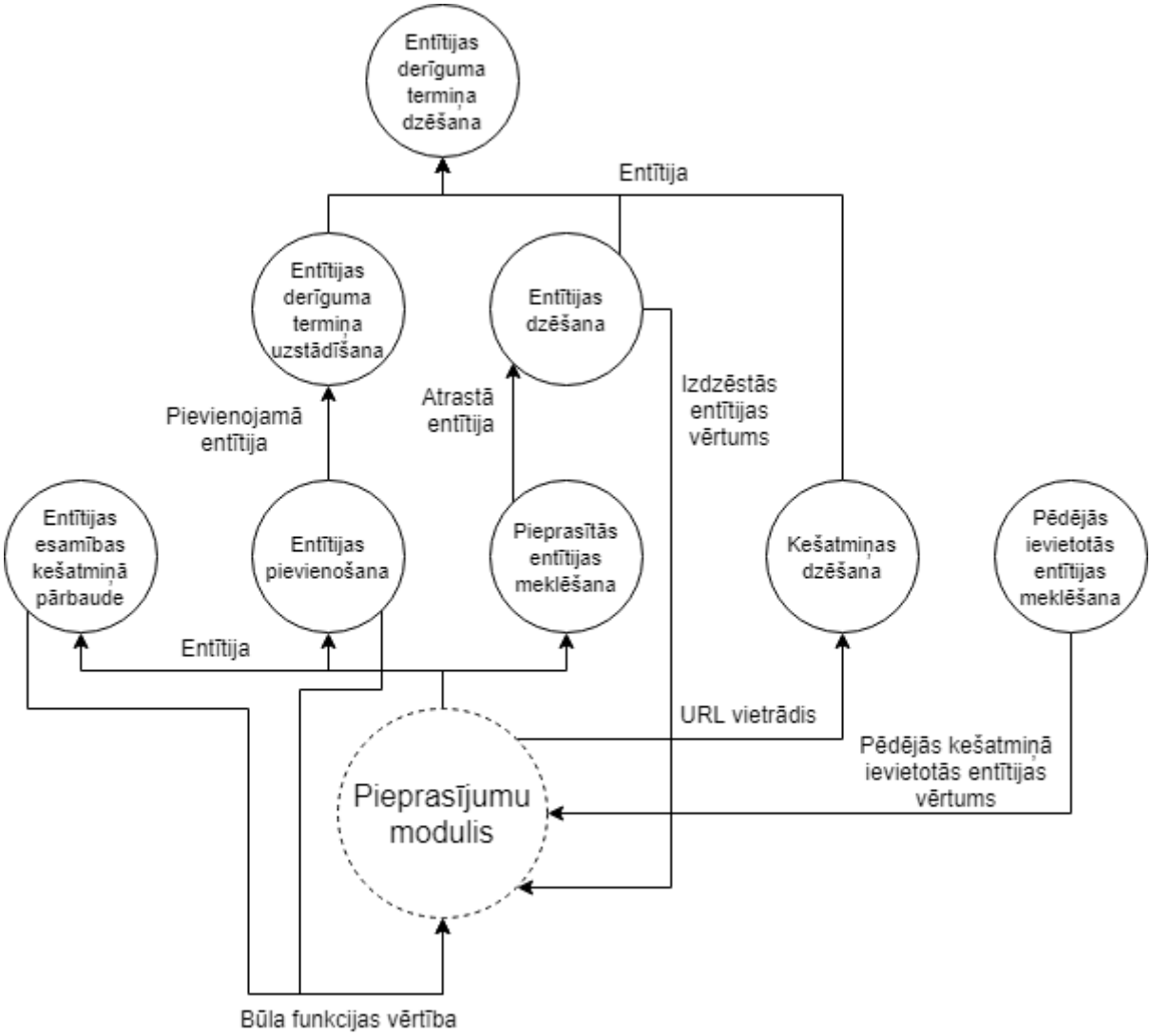

*Att. 2.3.* **2. līmeņa konfigurācijas moduļa datu plūsmu diagramma**

### 2.3.4. 2. līmeņa pieprasījumu moduļa datu plūsmu diagramma




*Att. 2.4.* **2. līmeņa pieprasījumu moduļa datu plūsmu diagramma**

### 2.3.5. 2. līmeņa kešatmiņas moduļa datu plūsmu diagramma

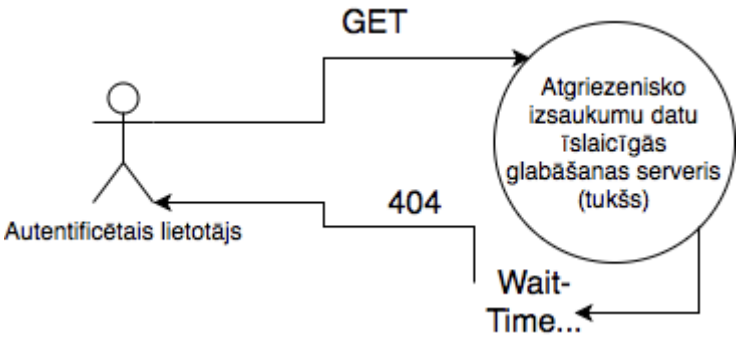



*Att. 2.5.* **2. līmeņa kešatmiņas moduļa datu plūsmu diagramma**

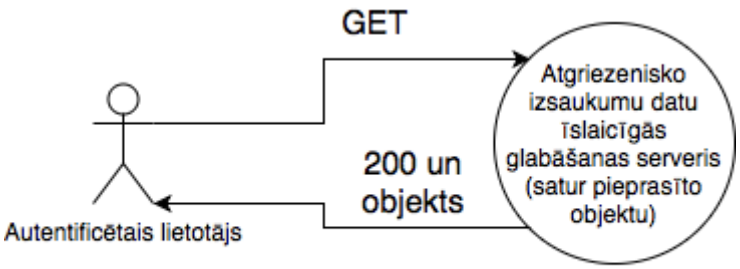
## 2.4. Lietošanas diagrammas

Lai attēlotu rīka lietotāju pamatdarbības ar serveri, tika izveidotas stilizētas lietošanas diagrammas. Tās parāda dažādus iespējamos pieprasījumu nosūtīšanas un atbildes iegūšanas scenārijus.

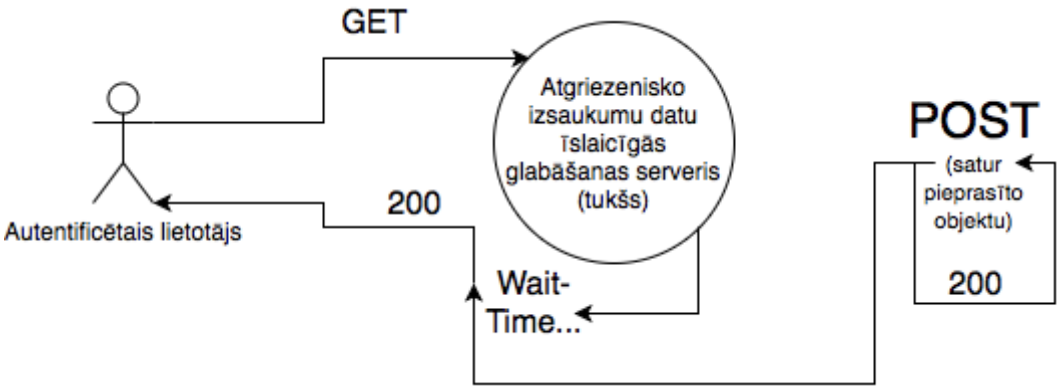
Par objektu nodaļas kontekstā tiek uzskatīts entītiju vērtums.



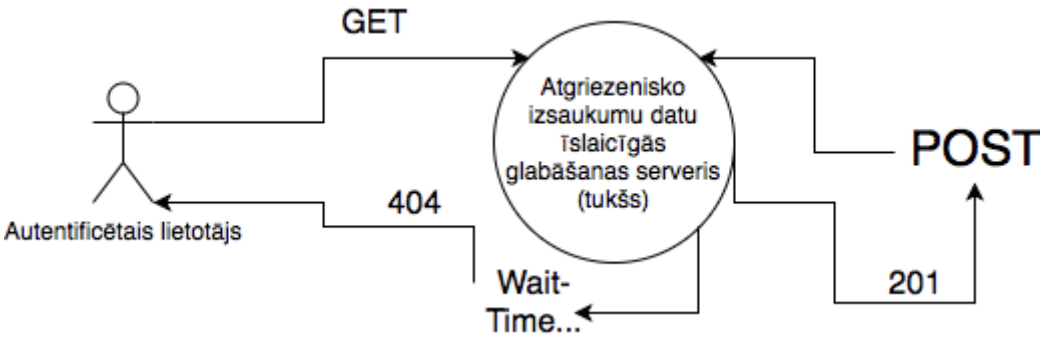

*Att. 2.6.* **Lietošanas diagramma gadījumā, kad tiek nosūtīts GET pieprasījums uz tukšo serveri**



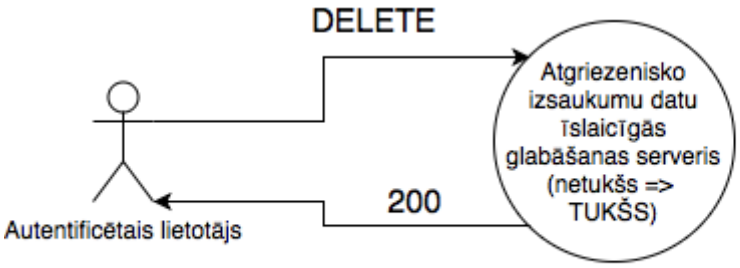

*Att. 2.7.* **Lietošanas diagramma gadījumā, kad serverī glabājas entītija ar pieprasīto objektu**



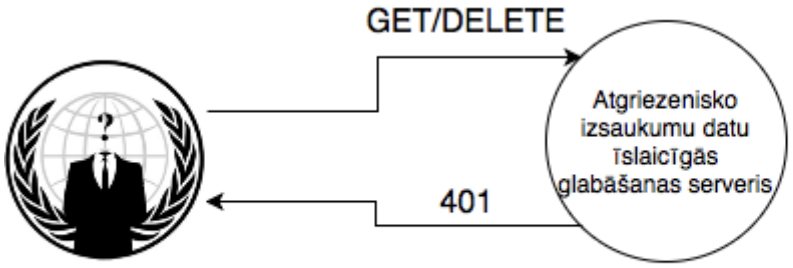

*Att. 2.8.* **Lietošanas diagramma gadījumā, kad aktīvā pieprasījuma laikā uz serverī parādās entītija ar pieprasīto objektu**




*Att. 2.9.* **Lietošanas diagramma gadījumā, kad aktīvā pieprasījuma laikā uz serverī parādās entītija, kuras vērtums neatbilst aktīvajam pieprasījumam**




*Att. 2.10.* **Lietošanas diagramma gadījumā, kad netukšajam serverim tiek nosūtīts DELETE pieprasījums**




*Att. 2.11.* **Lietošanas diagramma gadījumā, kad neautentificētais lietotājs mēģina nosūtīt GET vai DELETE pieprasījumu[5]**

# 3. TESTĒŠANAS DOKUMENTĀCIJA

## 3.1. Testēšanas apraksts

Rīka testēšana tika veikta, izmantojot "pēlēkās kastes" metodi, t.i., apvienojot "melnās kastes" un "baltās kastes" principu. Lai pārbaudītu rīka funkcionalitāti un darbību, tika izmantotas šādas testēšanas metodes:

- kešatmiņas moduļa vienībtestēšana;
- pieprasījumu moduļa integrācijas testēšana;
- rīka manuālā testēšana, izmantojot lietojumprogrammatūras saskarņu izstrādes vides, viedierīces un tīmekļa pārlūkprogrammas.

Kopumā uz šī dokumenta tapšanas brīdi rīkam ir 73 dažādi automatizētie testi, kuri nodrošina koda 100% pārklājumu. Testu izstrāde notika uzreiz pēc attiecīgā moduļa/funkcijas izstrādes vai arī izstrādes laikā.

Vismaz reizi divās nedēļas rīks tiek testēts manuāli, lai pārliecinātos, ka tas funkcionē atbilstoši prasībām. Manuālā testēšana tika veikta, balstoties uz programmatūras projektējuma apraksta lietojumu diagrammām (sk. nodaļu 2.4.).

Rīka testēšanai tiek izmantoti Node.js ietvari Mocha un Supertest.

## 3.2. Testpiemēri

Liela testpiemēru skaita dēļ, nodaļā tiks atspoguļoti tikai nozīmīgākie testpiemēri katra moduļa funkcijām.

Par objektu nodaļas kontekstā tiek uzskatīts entītiju vērtums.

### 3.2.1. Pieprasījumu moduļa testēšana

*Tab. 3.1.*

**Pieprasījumu moduļa (1.3.2.) testi**

| Nr. | Testējamā funkcija | Testa apraksts | Sagaidāmais rezultāts | Testa rezultāts |
|---|---|---|---|---|
| 1. | REQ.AUTH | Neautentificētais lietotājs nosūta GET pieprasījumu. | Pieprasījums netiek izpildīts, un lietotājs saņem atbildi no servera ar statusa kodu 401 un kļūdu paziņojumu. | + |
| 2. | REQ.AUTH | Lietotājs, kurš ir ievadījis nepareizus pierakstīšanās datus, nosūta DELETE pieprasījumu. | Pieprasījums netiek izpildīts, un lietotājs saņem atbildi no servera ar statusa kodu 401 un kļūdu paziņojumu. | + |

| | | | | |
|---|---|---|---|---|
| 3. | REQ.OPTIONS | Tiek nosūtīts OPTIONS pieprasījums. | Pieprasījums tiek izpildīts, un lietotājs saņem atbildi no servera ar statusa kodu 200 un attiecīgo paziņojumu. Atbildes galvenēs ir norādīti pieejamie pieprasījumu veidi. | + |
| 4. | REQ.GET | Tiek nosūtīts tukšs GET pieprasījums no URL vietrāža 'localost:9000/'. | Pieprasījums tiek izpildīts, un lietotājs saņem atbildi no servera ar statusa kodu 200 un pirmo objektu kešatmiņā, kurš ir pieejams šajā URL vietrādī. | + |
| 5. | REQ.GET | Tiek nosūtīts GET pieprasījums ar atslēgas-vērtības pāri, kurš ir kādam no kešatmiņas objektiem. | Pieprasījums tiek izpildīts, lietotājs saņem atbildi no servera ar statusa kodu 200 un pirmo objektu kešatmiņā, kurš ir pieejams šajā URL vietrādī un kuram ir pieprasītais atslēgas-vērtības pāris. | + |
| 6. | REQ.GET | Tiek nosūtīts GET pieprasījums tikai ar atslēgu. | Pieprasījums tiek izpildīts, un lietotājs saņem atbildi no servera ar statusa kodu 200 un pirmo objektu kešatmiņā, kurš ir pieejams šajā URL vietrādī un kuram ir pieprasītā atslēga. | + |
| 7. | REQ.GET | Tiek nosūtīts GET pieprasījums ar atslēgas-vērtības pāri, kuru nesatur neviens objekts dotajā URL vietrādī. | Pieprasījums tiek izpildīts, lietotājs saņem atbildi no servera ar statusa kodu 404 un kļūdu paziņojumu, jo dotajā URL vietrādī tāds objekts netika atrasts. | + |
| 8. | REQ.GET | Tiek nosūtīti 2 GET pieprasījumi ar vienādu atslēgas-vērtības pāri no viena un tā paša URL vietrāža. Serverī šajā URL vietrādī glabājas tikai viens šāds objekts. | Abi pieprasījumi tiek izpildīti. Lietotājs uz pirmo pieprasījumu saņem atbildi no servera ar statusa kodu 200 un meklējamo objektu, savukārt uz otro pieprasījumu viņš saņem atbildi no servera ar statusa kodu 404 un kļūdu paziņojumu, jo dotajā URL vietrādī tāds objekts vairs neeksistē. | + |
| 9. | REQ.GET | Tiek nosūtīts GET pieprasījums ar “Wait-Time” galveni, kura satur skaitli. | Pieprasījums izpildās, un lietotājs saņem servera atbildi pēc laika, kurš tika norādīts sekundēs pieprasījuma galvenē. | + |
| 10. | REQ.GET | Tiek nosūtīts GET pieprasījums sameklēt objektu, kuram ir beidzies derīguma termiņš. | Pieprasījums tiek izpildīts, lietotājs saņem atbildi no servera ar statusa kodu 404 un kļūdu paziņojumu, jo novecojis objekts tika izdzēsts. | + |

| | | | | |
|---|---|---|---|---|
| 11. | REQ.POST | Tiek nosūtīts POST pieprasījums ar tukšu objektu. | Pieprasījums tiek izpildīts, lietotājs saņem atbildi no servera ar statusa kodu 400 un kļūdu paziņojumu, jo tukšus objektus nedrīkst ievietot serverī. | + |
| 12. | REQ.POST | Tiek nosūtīts POST pieprasījums no ļaunprātīgā URL vietrāža. | Pieprasījums netiek izpildīts, un lietotājs saņem atbildi no servera ar statusa kodu 400 un kļūdu paziņojumu. | + |
| 13. | REQ.POST | Tiek nosūtīts POST pieprasījums ar ievietošanai derīgu objektu. | Pieprasījums tiek izpildīts, un lietotājs saņem atbildi no servera ar statusa kodu 201 un attiecīgo paziņojumu. | + |
| 14. | REQ.POST | Tiek nosūtīts POST pieprasījums ar objektu, kurš atbilst kādam no aktīvo GET pieprasījumu meklējamajam objektam. | Pieprasījums tiek izpildīts, POST objekts tiek nosūtīts aktīvajam GET pieprasījumam. Abos pieprasījumos lietotājs saņem atbildi no servera ar statusa kodu 200 un attiecīgo paziņojumu. | + |
| 15. | REQ.DELETE | Tiek nosūtīta DELETE pieprasījums izdzēst visus objektus, kuri glabājas uz servera. | Pieprasījums tiek izpildīts, un lietotājs saņem atbildi no servera ar statusa kodu 200 un attiecīgo paziņojumu. | |

### 3.2.2. Kešatmiņas moduļa testēšana

*Tab. 3.2.*

**Kešatmiņas moduļa (1.3.3.) testi**

| Nr. | Testējamā funkcija | Testa apraksts | Sagaidāmais rezultāts | Testa rezultāts |
|---|---|---|---|---|
| 1. | CACHE.SET.TIME | Tiek meklēts objekts, kurš tika izdzēsts no kešatmiņas pēc derīguma termiņa beigām. | Darbība ir neveiksmīga, tiek izvadīts attiecīgais kļūdu paziņojums. | + |
| 2. | CACHE.HAS, CACHE.FIND | No kešatmiņas tiek pieprasīts tukšais objekts. | Darbība ir veiksmīga, tiek atgriezts pirmais kešatmiņā ievietotais objekts ar attiecīgo URL vietrādi. CACHE.HAS atgriež vērtību “true”. | + |
| 3. | CACHE.HAS, CACHE.FIND | No kešatmiņas tiek pieprasīts kešatmiņā esošais objekts. Meklējamā objekta un kešatmiņā esošā objekta URL vietrāži atšķiras. | Darbība ir neveiksmīga, jo vietrāži atšķiras. CACHE.HAS atgriež vērtību “false”. | + |

| | | | | |
|---|---|---|---|---|
| 4. | CACHE.HAS, CACHE.FIND | No kešatmiņas tiek pieprasīts neeksistējošs objekts. | Darbība ir neveiksmīga. CACHE.HAS atgriež vērtību “false”. | + |
| 5. | CACHE.HAS, CACHE.FIND | No kešatmiņas tiek pieprasīts sameklēt kešatmiņā ievietoto objektu tikai pēc atslēgām. | Darbība ir veiksmīga, tiek atgriezts meklējamais objekts. CACHE.HAS atgriež vērtību “true”. | + |
| 6. | CACHE.PEEK | Tiek pieprasīts pēdējais kešatmiņā pievienotais objekts. | Darbība ir veiksmīga, metode atgriež pēdējo pievienoto objektu. | + |
| 7. | CACHE.ADD | Tiek mēģināts pievienot kešatmiņai derīgu objektu. | Darbība ir veiksmīga, funkcija CACHE.PEEK atgriež tikko pievienoto objektu. | + |
| 8. | CACHE.ADD | Tiek mēģināts pievienot kešatmiņai tukšu objektu. | Darbība ir neveiksmīga, jo ievietojamie objekti nedrīkst būt tukši. Funkcija atgriež vērtību “false”. | + |
| 9. | CACHE.ADD | Tiek mēģināts pievienot kešatmiņai simbolu virkni. | Darbība ir neveiksmīga, jo kešatmiņā var ievietot tikai objektus. Funkcija atgriež “ false”. | + |
| 10. | CACHE.OBJ.DELETE | Tiek mēģināts izdzēst kešatmiņā ievietoto objektu. | Darbība ir veiksmīga, objekts tiek izdzēsts no kešatmiņas un atgriezts. | + |
| 11. | CACHE.OBJ.DELETE | Tiek mēģināts izdzēst kešatmiņā ievietoto objektu, kuram ir beidzies derīguma termiņš. | Darbība ir neveiksmīga, jo objekts tika izdzēsts no kešatmiņas pēc derīguma termiņa beigām. Tiek atgriezts kļūdas paziņojums. | + |
| 12. | CACHE.OBJ.DELETE | Tiek mēģināts izdzēst no kešatmiņas simbolu virkni. | Darbība ir neveiksmīga, jo kešatmiņā glabājas tikai objekti. Tiek atgriezts kļūdas paziņojums. | + |
| 13. | CACHE.OBJ.DELETE | Tiek mēģināts izdzēst no kešatmiņas neeksistējošo objektu. | Darbība ir neveiksmīga. Tiek atgriezts kļūdas paziņojums. | + |
| 14. | CACHE.CLEAR | Tiek mēģināts izdzēst visus serverī esošos datus. | Darbība ir veiksmīga, CACHE.PEEK atgriež tukšu objektu. | + |
| 15. | CACHE.CLEAR | Tiek mēģināts izdzēst datus, kuri glabājas uz konkrētā URL vietrāža. | Darbība ir veiksmīga, CACHE.PEEK atgriež tikai tos objektus, kuri tika | + |

| | | | | |
|---|---|---|---|---|
| | | | piesaistīti pie citiem URL vietrāžiem. | |

## 3.3. Rezultātu kopsavilkums

Testēšanas laikā atrastās kļūdas tika novērstas, veicot uzlabojumus pirmkodā. Pēc kļūdu atrašanas un likvidācijas testēšana tika veikta atkārtoti, lai pārliecinātos, ka ieviestais risinājums ir stabils un efektīvs. Pārsvarā kļūdas radās neuzmanības dēļ.

Pārbaudot testu stabilitāti, tika izsecināts, ka arī pašam satvaram Supertest ir nepieciešami uzlabojumi, jo, atkārtoti izlaižot testus ar šo satvaru, periodiski dažu testu rezultāti atšķīrās no sagaidāmajiem. Līdzīgi testējot citu projektu, izmantojot citu satvaru, šī problēma nav radusies. Nākotnē testēšanai tiek plānots izmantot citu satvaru, kurš ir stabilāks par Supertest, piemēram, Jest.

# 4. PROJEKTA ORGANIZĀCIJA

Rīka izstrādē tika plānots pieturēties pie ūdenskrituma programmatūras izstrādes dzīves cikla modeļa. Sākot izstrādi, tika definētas prasības rīka izstrādei, bet izstrādes gaitā radās novirzes no izvēlētā modeļa, tā kā izstrādes procesā tika noteiktas dažas jaunas prasības.

Projekta sākumā tika izveidota rīka programmatūras prasību specifikācija un projektējuma apraksts.

Tā kā, sākot darbu pie projekta, autorei nebija pieredzes darbā ar JavaScript, izstrāde aizņēma vairāk laika nekā tika plānots (sk. nodaļu 7). Sākumā tika nokonfigurēta izstrādei nepieciešamā darba vide (uzinstalēti visi nepieciešamie satvari, JavaScript pakotņu pārvaldnieki, uzlikta un nokonfigurēta izstrādes vide). Tika apgūti JavaScript valodas pamati un iegūtas iemaņas darbā ar Node.js un Express.js satvariem, testēšanas satvariem Mocha un Supertest, kā arī iegūts priekšstats par jaunajām kvalitātes nodrošināšanas nozares aktualitātēm un specifiku.

Atgriezenisko izsaukumu datu īslaicīgās glabāšanas servera izstrādē kešatmiņas modulis tika izstrādāts vienlaikus ar pieprasījumu moduli, jo tāda ir rīka specifika. Vēlāk tika ieviesta prasība padot konfigurācijas parametrus no komandrindas vai galvenēs, lai rīks būtu elastīgāks un draudzīgāks pret lietotājiem. Vēlāk tika pievienota arī autentifikācijas prasība, lai cilvēkiem, kuri nav saistīti ar doto projektu, neļautu piekļūt datiem, kuri tiek uzglabāti serverī. Paralēli projekta izstrādei tika veidots un noformēts dotais dokuments.

Rīka testēšana notika vienlaikus ar izstrādi: funkcijas un metodes tika pārbaudītas uzreiz pēc uzrakstīšanas, lai noskaidrotu, vai tās darbojas atbilstoši prasību specifikācijai un vai ir nepieciešami to uzlabojumi. Dotā metode palīdzēja ātrāk un efektīvāk atkļūdot izstrādātos rīkus.

Rīka izstrādes beigās tika veikta manuālā testēšana, kurā iesaistījās arī potenciālie rīka lietotāji, lai apliecinātu, ka tam piemīt visa nepieciešamā funkcionalitāte.

Projekta pamatprasību formulēšana notika, konsultējoties ar prakses vadītāju un kvalitātes nodrošināšanas speciālistiem. Produkta izstrāde, testēšana un dokumentēšana tika veikta patstāvīgi.

Projekta izstrādē tika izmantota WebStorm izstrādes vide, pirmkoda uzturēšanai tika izmantota versiju kontroles sistēma Git.

Rīku testēšanā tika izmantotas Apple iPhone 8 (iOS 11.3), iPhone 8 Plus (iOS 11) un Samsung Galaxy S6 Edge+ (Android 7.0) ierīces, lietojumprogrammatūras saskarņu izstrādes vide Postman, kā arī pārlūkprogrammas Mozilla Firefox, Google Chrome, Safari un Opera.

# 5. KVALITĀTES NODROŠINĀŠANA

Lai nodrošinātu izstrādājamā produkta kvalitāti, tika veikti šādi pasākumi:

- rīka prasību specifikācija un projektējuma apraksts tika noformēti atbilstoši valsts standartiem (sk. nodaļas 1.1.3. un 2.1.3.);
- rīka pirmkodā tika ievērots vienots programmēšanas stils, kā arī ievērotas ECMAScript specifikācijas prasības;
- rīka pirmkods ir komentēts vietās, kur tas ir nepieciešams, kā arī iekļauts reģistrētājs, lai būtu vienkāršāk sekot līdzi procesu un funkciju izpildei un atkļūdot rīku;
- pirmkodā tika ievērots DRY princips, lai tas būtu elegants un viegli lasāms;
- pirmkods tika strukturēts modulāri;
- izstrādājot jaunu funkcionalitāti, tas tika veikts uz atsevišķajiem repozitorija zariem, kuri tika sapludināti ar pamatzaru tikai tad, kad funkcionalitātes izstrāde bija pabeigta un kad rīks spēja iziet visus testpiemērus, ieskaitot tos, kuri tika izstrādāti pirms dotās funkcionalitātes ieviešanas;
- pirms jaunās funkcionalitātes repozitorijas zaru sapludināšanas ar pamatzaru tika veikta koda apskate; sapludināšana notika tikai pēc projekta vadītāja apstiprinājuma.

# 6. KONFIGURĀCIJAS PĀRVALDĪBA

Izstrādājamā rīka pirmkoda konfigurāciju pārvaldībai tika izmantots versiju kontroles rīks Git, un tā rīka repozitorijs tika glabāts Git repozitoriju pārvaldniekā GitLab.

Izstrādes procesā tika izmantots Git Flow zarošanas princips. Repozitorijā tika uzturēts zars master, uz kura glabājās strādājoša un vadītāja akceptēta rīka versija. Ja bija nepieciešams veikt izmaiņas, tika veidots jauns zars, kurā tiek veiktas izmaiņas. Tā nosaukums bieži bija saistīts ar pašām izmaiņām, lai būtu vienkāršāk orientēties izveidotajos zaros un darba vadītājam būtu vieglāk izprast izmaiņu būtību. Pirms zara nosūtīšanas uz GitLab repozitoriju, ir jāpārliecinās, ka rīks veiksmīgi iziet visus vienībtestus: gan vecos, gan arī jaunos, kuri pārbauda jauno rīku funkcionalitāti.

Pēc izmaiņu veikšanas zars tiek saglabāts GitLab repozitorijā. Pirms jaunā zara sapludināšanas ar master zaru tiek veikta koda apskate. Sapludināšana tika veikta tikai tad, ja projekta vadītājs apstiprina izmaiņas.

Rīkā izmantojamo moduļu versiju pārvaldībai tiek izmantots nsp rīks. Aktuālās moduļu versijas tiek glabātas rīka galvenajā repozitorijā konfigurācijas failā package.json.

# 7. DARBIETILPĪBAS NOVĒRTĒŠANA

Ņemot vērā to, ka autorei pirms projekta uzsākšanas nebija pieredzes darbā ar izmantotajām tehnoloģijām, bija grūti precīzi novērtēt projekta darbietilpību, un rīka izstrāde autorei aizņēma vairāk laika nekā tika plānots.

Darbietilpības novērtēšanai tika izmantota trīskāršā metode: projekta izstrādes posmi tiek novērtēti no pesimistiskā, reālistiskā un optimistiskā skatu punkta, un kopējā darbietilpība tiek aprēķināta, izmantojot formulu:

$$\frac{S(optimistiskais) + 4S(reālistiskais) + S(pesimistiskais)}{6}$$

Par dienu tiek uzskatīta pilnās slodzes darbinieka darba diena - tā ir 8 stundas ilga.

*Tab. 7.1.*

**Darbietilpības novērtēšanas tabula**

| | Pesimistiskais novērtējums (dienas) | Reālistiskais novērtējums (dienas) | Optimistiskais novērtējums (dienas) |
|---|---|---|---|
| Iepazīšanās ar projektā izmantojamām tehnoloģijām | 15 | 10 | 5 |
| Konsultācijas ar darba vadītāju | 5 | 3 | 2 |
| Programmatūras prasību specifikācijas noformēšana | 7 | 5 | 3 |
| Programmatūras projektējuma apraksta noformēšana | 7 | 5 | 3 |
| Konfigurācijas moduļa izstrāde | 4 | 3 | 1 |
| Pieprasījumu moduļa izstrāde | 20 | 15 | 7 |
| Kešatmiņas moduļa izstrāde | 10 | 7 | 5 |
| Testpiemēru izveide | 15 | 10 | 5 |
| Testēšanas dokumentācijas noformēšana un rezultātu apkopošana | 5 | 3 | 1 |
| Kvalifikācijas darba noformēšana | 6 | 2 | 1 |
| **Summā:** | **94** | **63** | **33** |

Darbietilpība = (94 + (63 * 4) + 33)/6 = 63,167 dienas jeb 3,16 personmēneši.

Darbietilpības novērtējums pēc trīskāršās metodes ir vienāds ar 3,16 personmēnešiem, tātad projekta apjoms atbilst kvalifikācijas darba izstrādes prasībām.

# SECINĀJUMI

Izstrādājot darbu, tika izveidots funkcionējošais rīks, kurš atbilst prasību specifikācijai un projektējuma aprakstam. Tas tiek aktīvi pielietots uzņēmuma ikdienā, optimizējot testēšanu produktiem, kurus izmanto vairāk nekā pusmiljons aktīvo lietotāju. Pēc lietotāju ieteikumiem rīks nākotnē var tikt uzlabots un tam var būt pielikta jaunā funkcionalitāte, jo, mainoties testējamās programmatūras funkcionalitātei, mainās arī testēšanas stratēģija.

Rīka izstrādē tika apgūta JavaScript valoda, satvars Node.js, kurš ļāva izmantot JavaScript serveru puses programmēšanā, un satvars Express.js, kurš tiek izmantots tīmekļa lietojumprogrammatūras izstrādē. Autore ieguva testu automatizēšanas prasmes, izmantojot satvarus Supertest un Mocha, savukārt Postman vides apgūšana ļāva veiksmīgi veikt arī rīka manuālo testēšanu.

Palaižot atgriezenisko izsaukumu datu īslaicīgās glabāšanas datu īslaicīgās glabāšanas serveri uz resursdatora tā, lai to varētu izmantot nepārtrauktās integrācijas nolūkos, autore apguva nginx servera konfigurēšanas, kā arī procesu pārvaldnieka pm2 pamatus.

Tā kā autorei projektā izmantojamās tehnoloģijas pirms izstrādes uzsākšanas nebija pazīstamas, daudz laika tika patērēts to apgūšanai un dokumentācijas pētīšanai. Vislielākās grūtības sagādāja automatizēto testu rakstīšana, jo testēšanas satvaru darbības principi atšķiras, kā arī testēšanas procesā tika konstatētas nepilnības Supertest satvarā.

# IZMANTOTIE AVOTI

[1]Datu pārraides un apstrādes sistēmas. Angļu-krievu-latviešu skaidrojošā vārdnīca — R., SWH, 1995 (Pēdējo reizi skatīts: 28.05.2018)

[2]Robert C. Martin Clean Code: A Handbook of Agile Software Craftsmanship, ISBN-13: 978-0132350884 (Pēdējo reizi skatīts: 28.05.2018)

[3]https://estudijas.lu.lv/mod/resource/view.php?id=131427 (Pēdējo reizi skatīts: 28.05.2018)

[4]https://estudijas.lu.lv/mod/resource/view.php?id=131428 (Pēdējo reizi skatīts: 28.05.2018)

[5]https://upload.wikimedia.org/wikipedia/commons/thumb/a/a6/Anonymous_emblem.svg/1200px-Anonymous_emblem.svg.png (Pēdējo reizi skatīts: 28.05.2018)

# 1. pielikums. Rīka pirmkods

CACHE.FIND funkcijas realizācija:

```
'use strict';

module.exports = (get, objects) => {
  // loops through the cache with POST'ed objects' entities and returns
index of entity with the requested object, if found
  return objects.findIndex(post => {
    if(get.url !== post.url) {
      return false;
    }

    // loops through all GET object's keys to find the similar values in
POST'ed objects
    return Object.keys(get.object).reduce((value, key) => {
      // if only key is specified in requested object, returns the first
object in cache with this key
      if (!value) {
        return value;
      }

      if(get.object[key] === '') {
        return !!post.object[key];
      }

      // values are stringified as the cache receives POST'ed object as a
string
      return String(post.object[key]) === String(get.object[key]);
    }, true);
  });
};
```

Kešatmiņas moduļa funkciju CACHE.SET.TIME, CACHE.CLEAR.TIME, CACHE.HAS, CACHE.REMOVE, CACHE.ADD, CACHE.PEEK, CACHE.CLEAR realizācija:

```
'use strict';

const path = require('path');
const log = require('./logger')(path.relative(process.cwd(), __filename));
const timeout = require('./cli-arguments').timeout;
const findObjectIndex = require('./find-object');

const setTime = (object, collection, timeouts) => {
  // sets timeout for the entity; it will be deleted when deprecated
  const timeoutID = setTimeout(() => {
    const deprecatedObject = findObjectIndex(object, collection);
    collection.splice(deprecatedObject, 1);
    log.info(`Object ${JSON.stringify(object.object)} removed`);
  }, timeout);
  log.info(`Timeout for object ${JSON.stringify(object.object)} set:
${timeout}`);

  // timeout is added to the timeouts collection (needed in case of timeout
removing)
  timeouts.push({
    object: object,
    id: timeoutID
  });
};

const clearTime = (object, timeouts) => {
  // searches for a timeout to clean
  const timeoutId = timeouts.findIndex(timeout => {
    return (timeout.object === object);
  });

  // clears timeout and deletes it from the timeouts collection, if found
  if (timeoutId !== -1) {
    const timeoutToClear = timeouts[timeoutId];

    clearTimeout(timeoutToClear.id);
    timeouts.splice(timeoutId, 1);
    log.info(`Timeout for object ${JSON.stringify(timeoutToClear.object)}
cleared`);
  }
};

module.exports = (cache = []) => {
  const timeouts = [];
  // sets timeouts to all objects in case the cache is not empty
  cache.forEach(object => {
    setTime(object, cache, timeouts);
  });

  // findObjectIndex: boolean edition
  const has = object => {
    return findObjectIndex(object, cache) !== -1;
  };

  const remove = object => {
    const foundObjectId = findObjectIndex(object, cache);

    // if found requested object, removes it and returns it to the calling
function
```

```
    if (foundObjectId !== -1) {
      const foundObject = cache[foundObjectId];

      clearTime(foundObject, timeouts);
      cache.splice(foundObjectId, 1);
      log.info(`Object ${JSON.stringify(foundObject)} removed`);
      return foundObject.object;
    }
    log.error(`No objects with parameters ${JSON.stringify(object)} to
remove`);
    throw new Error(`No objects with such parameters to remove.`);
  };

  const add = object => {
    const objectKeys = Object.keys(object.object);
    const logObject = JSON.stringify(object); // needed for logger

    if (!objectKeys.length) {
      log.error(`Object ${logObject} cannot be added to cache (empty
object)`);
      return false;
    }

    if (typeof(object.object) !== 'object') {
      log.error(`Object ${logObject} cannot be added to cache (not an
object)`);
      return false;
    }

    // adds object to the cache and sets timeout for it in case object was
successfully validated
    cache.push(object);
    log.info(`Object ${logObject} added to cache`);
    setTime(object, cache, timeouts);
    log.info(`Object ${logObject} will be deleted in ${timeout} ms`);
    return true;
  };

  const peek = () => {
    if (!cache.length) {
      return {};
    }

    const lastObject = cache[cache.length - 1].object;

    log.info (`Last object in cache: ${JSON.stringify(lastObject)}`);
    return lastObject;
  };

  const clear = url => {
    const regex = new RegExp('^' + url + '(\\/(\\w+\\/?)*)*$'); //accepts
the original url + sub urls for this

    // clears the whole cache in case request sent from '/' URL
    if(url === '/') {
      cache.forEach(object => {
        clearTime(object, timeouts);
      });
      cache.splice(0);
    }

    // deletes only those entities in the cache, whose URL matches the
regex
```

```
    cache = cache.filter(object => {
      const matchRegex = regex.test(object.url);

      if(matchRegex) {
        clearTime(object, timeouts);
      }
      return !matchRegex;
    });

    log.info (`Cache for url ${url} cleared`);
  };

  return {
    has: has,
    remove: remove,
    add: add,
    peek: peek,
    clear: clear
  };
};
```

REQ.RES funkcijas realizācija:

```
'use strict';

const path = require('path');
const log = require('./logger')(path.relative(process.cwd(), __filename));

const responses =
  {
    error: 'No results found.',
    added: 'Object was successfully added.',
    wrongInput: 'Wrong input.',
    deleted: 'Cache was successfully cleared.',
    sent: 'Object was sent to active request.',
    found: 'Requested object was found.',
    unauthorized: 'Access denied.',
    wrongUrl: 'Wrong URL.',
    malwareUrl: 'URL is potentially malicious.',
    receivedRequest: 'Request received.'
  };

module.exports = (response, message, code, payload) => {
  //searches for a necessary message in a responses collection
  const responseMessage = responses[message];

  log.info(`Status code: ${code} Message: ${responseMessage} Payload:
${JSON.stringify(payload)}`);
  response
    .set({
      'Access-Control-Allow-Origin': '*',
      'Access-Control-Allow-Methods': 'GET, POST, DELETE, OPTIONS',
      'Access-Control-Allow-Headers': 'authorization, wait-time'
    })
    .status(code)
    .json({
      message: responseMessage,
      payload
    })
    .end();
};
```

REQ.GET funkcijas testēšanas fragments:

```
'use strict';

const assert = require('assert');
const request = require('supertest');
const createCache = require('../src/object-cache');
const timeout = require('../src/cli-arguments').timeout;
const waitTime = require('../src/cli-arguments').waitTime;
const sinon = require('sinon');
const createApp = require('../src/index.js');
const EventEmitter = require('events');
const {chuck, borat, singer, random, food, pizza} = require('./test-
objects');

const clock = sinon.useFakeTimers();

describe('index', () => {
  let options = null;
  let app = null;
  let cache = null;
  let credentials = null;
  const promises = [];

  beforeEach(() => {
    options = {
      emitter: new EventEmitter(),
      username: 'kefir',
      password: 'cucumbers'
    };
    credentials = new
Buffer(`${options.username}:${options.password}`).toString('base64');

    promises.splice(0);
    cache = createCache([chuck, borat]);
    app = createApp(cache, options);
  });

  after(() => {
    clock.restore();
  });

  describe('GET / (immediate response)', () => {
    const checkResponse = (request, object, done) => {
      request.expect(response => {
        assert.deepStrictEqual(JSON.parse(response.text).payload, object);
      })
        .expect(200, done);
    };

    it('Should return a 200 status code and the first object in the cache',
done => {
      const getRequest = request(app)
        .get('/')
        .set({
          'Accept': 'application/json',
          'Authorization': `Basic ${credentials}`
        });

      checkResponse(getRequest, chuck.object, done);
    });

    it('Should return a 200 status code and the requested object', done =>
```

```
{
      const getRequest = request(app)
        .get('/?name=Borat')
        .set({
          'Accept': 'application/json',
          'Authorization': `Basic ${credentials}`
        });

      checkResponse(getRequest, borat.object, done);
    });

    it('Should return a 200 status code and the requested object (keys
only)', done => {
      const getRequest = request(app)
        .get('/?country')
        .set({
          'Accept': 'application/json',
          'Authorization': `Basic ${credentials}`
        });

      checkResponse(getRequest, borat.object, done);
    });

    it('Should return a 200 status code and the requested object added
during GET request (several requests)', done => {
      promises.push(new Promise(resolve => {
        request(app)
          .get('/?name=Borat')
          .set({
            'Accept': 'application/json',
            'Authorization': `Basic ${credentials}`
          })
          .expect(response => {
            resolve(response);
          })
          .expect(200, () => {});
      }));

      promises.push(new Promise(resolve => {
        request(app)
          .get('/?name=Borat')
          .set({
            'Accept': 'application/json',
            'Authorization': `Basic ${credentials}`
          })
          .expect(response => {
            resolve(response);
          })
          .expect(200, () => {});
      }));

      promises.push(new Promise(resolve => {
        request(app)
          .post('/')
          .send(borat.object)
          .expect(response => {
            resolve(response);
          })
          .expect(200, () => {});
      }));

      Promise.all(promises).then(responses => {
        assert.deepStrictEqual(responses[0].statusCode, 200);
```

```
        assert.deepStrictEqual(JSON.parse(responses[0].text).payload,
borat.object);
        assert.deepStrictEqual(responses[1].statusCode, 200);
        assert.deepStrictEqual(JSON.parse(responses[1].text).payload,
borat.object);
        assert.deepStrictEqual(responses[2].statusCode, 200);
      })
        .then(done)
        .catch(done);
    });

    it('Should return a 200 status code and the requested object added
during GET request', done => {
      promises.push(new Promise(resolve => {
        request(app)
          .get('/?name=Kebab')
          .set({
            'Accept': 'application/json',
            'Authorization': `Basic ${credentials}`
          })
          .expect(response => {
            resolve(response);
          })
          .expect(200, () => {});
      }));

      promises.push(new Promise(resolve => {
        request(app)
          .post('/')
          .send(food.object)
          .expect(response => {
            resolve(response);
          })
          .expect(200, () => {});
      }));

      Promise.all(promises).then(responses => {
        assert.deepStrictEqual(responses[0].statusCode, 200);
        assert.deepStrictEqual(JSON.parse(responses[0].text).payload,
food.object);
        assert.deepStrictEqual(responses[1].statusCode, 200);
      })
        .then(done)
        .catch(done);
    });

    it('Should return a 404 status code when there is no requested object
in the given link', done => {
      promises.push(new Promise(resolve => {
        request(app)
          .get('/123?name=Kebab')
          .set({
            'Accept': 'application/json',
            'Authorization': `Basic ${credentials}`
          })
          .expect(response => {
            resolve(response);
          })
          .expect(404, () => {});
      }));

      promises.push(new Promise(resolve => {
        request(app)
```

```
          .post('/1')
          .send(food.object)
          .expect(response => {
            resolve(response);
          })
          .expect(201, () => {
            clock.tick(waitTime);
          });
      }));

      Promise.all(promises).then(responses => {
        assert.deepStrictEqual(responses[0].statusCode, 404);
        assert.deepStrictEqual(responses[1].statusCode, 201);
      })
        .then(done)
        .catch(done);
    });

    it('Should return a 200 status code and the requested object (added
before request)', done => {
      request(app)
        .post('/').send(food.object).end(() => {
          const getRequest = request(app)
            .get('/?price=3.5')
            .set({
              'Accept': 'application/json',
              'Authorization': `Basic ${credentials}`
            });

          checkResponse(getRequest, food.object, done);
        });
    });

    it('Should return a 200 status code and the requested object (requested
before timeout)', done => {
      request(app)
        .post('/').send(pizza.object).end(() => {
          clock.tick(timeout - 1);
          const getRequest = request(app)
            .get('/?place=Dorms')
            .set({
              'Accept': 'application/json',
              'Authorization': `Basic ${credentials}`
            });

          checkResponse(getRequest, pizza.object, done);
        });
    });

    it('Should return a 200 status code and the object to the first GET and
a 404 status code and an error to the second ' +
      '(same object requested)', done => {
      promises.push(new Promise(resolve => {
        request(app)
          .get('/?name=Kebab')
          .set({
            'Accept': 'application/json',
            'Authorization': `Basic ${credentials}`
          })
          .expect(response => {
            resolve(response);
          })
          .expect(200, () => {});
```

```
      }));

      promises.push(new Promise(resolve => {
        request(app)
          .get('/?name=Kebab')
          .set({
            'Accept': 'application/json',
            'Authorization': `Basic ${credentials}`
          })
          .expect('Content-Type', 'application/json; charset=utf-8')
          .expect(response => {
            resolve(response);
          })
          .expect(404, () => {});
      }));

      promises.push(new Promise(resolve => {
        request(app)
          .post('/')
          .send(food.object)
          .expect(response => {
            resolve(response);
          })
          .expect(200, () => {
            clock.tick(waitTime);
          });
      }));

      Promise.all(promises).then(responses => {
        assert.deepStrictEqual(responses[0].statusCode, 200);
        assert.deepStrictEqual(JSON.parse(responses[0].text).payload,
food.object);
        assert.deepStrictEqual(responses[1].statusCode, 404);
        assert.deepStrictEqual(responses[2].statusCode, 200);
      })
        .then(done)
        .catch(done);
    });

    it('Should return a 200 status code and the object to the first GET
request and return a 404 status code and an ' +
      'error to the second', done => {
      promises.push(new Promise(resolve => {
        request(app)
          .get('/?name=Kebab')
          .set({
            'Accept': 'application/json',
            'Authorization': `Basic ${credentials}`
          })
          .expect(response => {
            resolve(response);
          })
          .expect(200, () => {});
      }));

      promises.push(new Promise(resolve => {
        request(app)
          .get('/?age=0')
          .set({
            'Accept': 'application/json',
            'Authorization': `Basic ${credentials}`
          })
          .expect(response => {
```

```
            resolve(response);
          })
          .expect(404, () => {});
      }));

      promises.push(new Promise(resolve => {
        request(app)
          .get('/?surname=Reiniks&name=Lauris')
          .set({
            'Accept': 'application/json',
            'Authorization': `Basic ${credentials}`
          })
          .expect(response => {
            resolve(response);
          })
          .expect(200, () => {});
      }));

      promises.push(new Promise(resolve => {
        request(app)
          .post('/')
          .send(food.object)
          .expect(response => {
            resolve(response);
          })
          .expect(200, () => {});
      }));

      promises.push(new Promise(resolve => {
        request(app)
          .post('/')
          .send(singer.object)
          .expect(response => {
            resolve(response);
          })
          .expect(200, () => {
            clock.tick(waitTime);
          });
      }));

      Promise.all(promises).then(responses => {
        assert.deepStrictEqual(responses[0].statusCode, 200);
        assert.deepStrictEqual(JSON.parse(responses[0].text).payload,
food.object);
        assert.deepStrictEqual(responses[1].statusCode, 404);
        assert.deepStrictEqual(responses[2].statusCode, 200);
        assert.deepStrictEqual(JSON.parse(responses[2].text).payload,
singer.object);
        assert.deepStrictEqual(responses[3].statusCode, 200);
        assert.deepStrictEqual(responses[4].statusCode, 200);
      })
        .then(done)
        .catch(done);
    });
  });
});
```

## 2. pielikums. Izmantojamo HTTP pieprasījumu metožu tabula

Pastāv vairākas HTTP pieprasījuma metodes, kuras atšķiras pēc funkcionalitātes un veicamās darbības.

Darba ietvaros izstrādātajā rīkā tiek izmantotas šīs metodes:

| Pieprasījuma metodes nosaukums | Pieprasījuma metodes nozīme |
|---|---|
| GET | Metode pieprasa noteiktā resursa reprezentāciju. Metode tikai iegūst datus no resursa. Darba kontekstā GET tiek sūtīts, lai iegūtu vērtumus no servera. |
| POST | Metode tiek izmantota, lai nosūtītu entītiju uz noteikto resursu, kas izraisa pārmaiņas servera statusā. Darba kontekstā POST tiek sūtīts, lai ievietotu datus kešatmiņā. |
| DELETE | Metode dzēš noteikto resursu. |
| OPTIONS | Metode tiek izmantota, lai aprakstītu komunikācijas opcijas ar noteikto resursu, piemēram, nosūtot pieejamās pieprasījumu metodes. |

## 3. pielikums. Izmantojamo HTTP atbilžu statusa kodu tabula

Statusa kodi ļauj noteikt, ka nosūtītais pieprasījums tika izpildīts. Pateicoties statusa kodam, ir iespējams noskaidrot, vai serverim izdevās pareizi apstrādāt pieprasījumu, vai arī ir notikusi kāda kļūda.

Kvalifikācijas darba ietvaros izstrādātajā rīkā tiek izmantoti šādi statusa kodi:

| Statusa kods | Statusa koda nozīme |
|---|---|
| 200 OK | Pieprasījums tika veiksmīgi izpildīts. Piemēram, GET pieprasījuma gadījumā tika atrasti pieprasītie dati, savukārt POST gadījumā – dati, kuri tika nosūtīti pieprasījuma ķermenī, tika saņemti. |
| 201 Created | Pieprasījums tika veiksmīgi izpildīts, tika izveidots jaunais resurss, izstrādājamā rīka konekstā – entītija kešatmiņā. Šis statusa kods parasti ir raksturīgs POST pieprasījumiem. |
| 400 Bad Request | Pieprasījums netika izpildīts nepareizas sintakses dēļ, piemēram, nepareizi ievadīts URL vietrādis vai POST pieprasījumā nosūtītie ķermeņa dati neatbilst formātam, kuru akceptē serveris. |
| 401 Unauthorized | Lai saņemtu atbildi uz nosūtīto pieprasījumu, lietotājam ir jāautentificējas. Tas nozīmē, ka lietotājam nav piekļuves pieprasītajiem datiem. |
| 404 Not Found | Serveris nevar atrast pieprasīto resursu. Vai nu tiek izmantots nepareizs vietrādis, vai nu serverī netiek glabāti pieprasītie resursi. Izstrādājamā rīka kontekstā kods tiek nosūtīts tad, kad pieprasītie dati netiek atrasti serverī. |

Kvalifikācijas darbs „***Komunikācijas rīks gala lietotāju platformu testēšanai***” izstrādāts Latvijas Universitātes Datorikas fakultātē.

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

Autors: ***Karīna Seile-Zundāne*** _____ **___.05.2018**.

Rekomendēju darbu aizstāvēšanai

Darba vadītājs: ***B.dat Artūrs Vaļenieks*** _____ **___.05.2018.**

Recenzents: ***M.dat Roberts Vārtiņš***

Darbs iesniegts 28.05.2018.

Kvalifikācijas darbu pārbaudījumu komisijas sekretāre: *Darja Solodovņikova*_____

Darbs aizstāvēts kvalifikācijas darbu pārbaudījuma komisijas sēdē

___.06.2018. prot. Nr. _____

Komisijas sekretārs(-e): _____