LATVIJAS UNIVERSITĀTE

DATORIKAS FAKULTĀTE

PYTHON MODUĻA IZSTRĀDE UN INTEGRĀCIJA TĪMEKĻA OPERĒTĀJSISTĒMĀ WEBAPPOS DROŠĪBAS FUNKCIONALITĀTES NODROŠINĀŠANAI

KVALIFIKĀCIJAS DARBS

Autors: Iļja Repko

Studenta apliecības Nr.: ir19044

Darba vadītājs: Asoc.prof. Sergejs Kozlovičs

RĪGA 2022

ANOTĀCIJA

Tīmekļa operētājsistēma webAppOS ir infrastruktūra, kas ietver sevī vairākus tīmekļa procesorus jeb moduļus, kas ļauj vienā tīmekļa lietotnē izsaukt klašu metodes, kas ir realizētas dažādas programmēšanas valodās. Šī darba mērķis ir izveidot Python moduli un ieintegrēt to webAppOS vidē. Modulis dos izstrādātājiem iespēju veidot klašu metodes Python programmēšanas valodā. Moduli ir paredzēts lietot drošības funkcionalitātes nodrošināšanai, tas ir, automātiskai HTTPS komunikācijas protokola sertifikātu ģenerēšanai un sistēmas mapju šifrēšanai. Šiem nolūkiem tiks izmantoti, attiecīgi, RSA un AES algoritmi, kas ļauj sasniegt augstu drošības līmeni, netērējot daudz resursu.

Atslēgvārdi: webAppOS, python, kriptogrāfija, procesors, HTTPS protokols.

ABSTRACT

DEVELOPMENT AND INTEGRATION OF A PYTHON MODULE IN THE WEB OPERATING SYSTEM WEBAPPOS TO PROVIDE SECURITY FUNCTIONALITY

The web operating system webAppOS is an infrastructure that includes web processors, or modules, that provide the ability to invoke class methods written in different programming languages in one web application. The goal of this work is to develop a Python module and integrate it into webAppOS. That would allow developers to write functions and class methods in the Python programming language. The main usage of the module is to provide security functionality, i.e., to generate HTTPS communication protocol certificates for websites and to encrypt system directories. For that, the RSA and AES algorithms will be used, respectively. These algorithms achieve a high level of security without using to many resources.

Keywords: webAppOS , python, cryptography, processor, HTTPS protocol.

SATURA RĀDĪTĀJS

APZĪMĒJUMU SARAKSTS

WebAppOS – infrastruktūra tīmekļa lietojumprogrammu izstrādei un palaišanai.

RSA algoritms – [Rivest, Shamir, Adleman] populārs šifrēšanas algoritms, kas veidots uz publiskās atslēgas kriptogrāfijas principiem; algoritma drošība balstās uz to, ka ir grūti sadalīt pirmreizinātājos divu lielu pirmskaitļu reizinājumu.

AES algoritms – [Advanced Encryption Standard] cipardatu šifrēšanas standarts jeb algoritms, kas izmanto simetrisko atslēgu datu šifrēšanai un atšifrēšanai.

HTTP – [Hypertext Transfer Protocol] protokols, kas ļauj pārsūtīt datus tīmeklī.

HTTPS – [Hypertext Transfer Protocol Secure] TCP/IP lietojuma slāņa protokola HTTP paplašinājums, kas ļauj šifrēt savienojumus starp klientu un serveri.

SSL – [Secure Sockets Layer] kriptogrāfijas protokols, kas nodrošina drošu savienojumu starp klientu un serveri. SSL šobrīd tiek uzskatīs par nedrošu, to aizstāja TLS (Transport Layer Security); tomēr, saīsinājumu SSL turpina lietot, apzīmējot SSL/TLS protokolu saimi.

JSON – [JavaScript Object Notation] datu apmaiņas formāts, kas atbilst JavaScript objektu sintaksei.

API – [Application programming interface] programmēšanas saskarne lietojumprogrammās.

MINICA – [mini Certification authority] lietojumprogramma, kas izmanto RSA algoritmu, SSL sertifikātu ģenerēšanai un to parakstīšanai.

CRYFS – [A cryptographic filesystem] mapju šifrēšanas rīks, kas izmanto AES. CryFS ir iekļauts Ubuntu paku sistēmā, ir pieejamas versijas arī Mac un Windows vidēm.

Pamatmape – mape, kurā programma CryFS glabā šifrētus datus. Šo mapi iespējams “pieslēgt” (MOUNT) citā mapē, kurā dati būs pieejami nešifrētā veidā (CryFS nodrošina reālā laika šifrēšanu un atšifrēšanu starp šīm mapēm). Pēc UNMOUNT dati paliks tikai pamatmapē šifrēta veidā.

MOUNT – darbības vārds, nozīmē izveidot virtuālo datu nesēju, kur glabājas atšifrēti lietotāja dati.

UNMOUNT – darbības vārds, šifrēt mapes datus jeb dzēst virtuālo datu nesēju ar lietotāja datiem tajā.

DPD – datu plūsmu diagramma.

PPS – programmatūras prasību specifikācija.

PPA – programmatūras projektējuma apraksts.

IEVADS

Nolūks

Dokumenta nolūks ir apkopot Python moduļa programmatūras prasību specifikāciju (PPS), programmatūras projektējuma aprakstu (PPA), kā arī moduļa funkcionalitātes un testēšanas principus. Dokumenta mērķauditorija ir pasūtītājs (LU MII), kā arī potenciālie citi izstrādātāji, kas izmantos Python moduli, lai izstrādātu vai izsauktu Python kodu webAppOS vidē.

Darbības sfēra

Python moduļa integrācija webAppOS vidē dod iespēju gala lietotājam ērti un ātri izmantot šifrēšanas bibliotēku pašparakstītu (self-signed) SSL sertifikātu ģenerēšanai, lai turpmāk veidotu šifrētus HTTPS savienojumus datortīklos. Droši savienojumi tiks izmantoti, lai pasargātu gala lietotāju no uzbrukumiem un nesakcionētas piekļuves datiem vidēs (galvenokārt lokālajos datortīklos), kur nav iespējams izmantot sertifikātus, kurus automātiski paraksta ārējās sertifikācijas iestādes. Savukārt, lai mazinātu sekas datu noplūdes gadījumā, tiks nodrošināta papildus aizsardzība – mapju šifrēšana.

Saistība ar citiem dokumentiem

Dokumenta sastādīšanā tika ievērots standarts LVS 68:1996 „Programmatūras prasību specifikācijas ceļvedis" [1]. Dokumenta tehniskais noformējums ir izstrādāts saskaņā ar Latvijas Universitātes rīkojumu Nr. 1/38 "Prasības noslēguma darbu (bakalaura, maģistra darbu, diplomdarbu un kvalifikācijas darbu) izstrādāšanai un aizstāvēšanai Latvijas Universitātē" [2]. Programmatūras projektējuma apraksts ir saistīts ar LVS 72:1996 "Ieteicamā prakse programmatūras projektējuma aprakstīšanai"[3]. Testēšanas dokumentācija tika veidota, balstoties uz LVS 70:1996 „Programmatūras testēšanas dokumentācija"[4] prasībām.

Dokumenta pārskats

Dokuments sastāv no astoņām pamatdaļām. Pirmā daļa ir vispārīgs apraksts, kas apraksta sistēmu vispārējā līmenī, kā arī apkopo informāciju par sistēmas lietotājiem un pasūtītāju. Otrā daļa ir programmatūras prasību specifikācija, kas ietver sevī funkcionālās un nefunkcionālās prasības ar mērķi aprakstīt sistēmas funkcionalitāti, lietotāja ievadītus datus un izvaddatus. Trešajā daļā jeb programmatūras projektējuma aprakstā ir apkopota informācija par izvēlēta risinājuma integrāciju webAppOS vidē, tā saistību ar šifrēšanas sistēmu, kā arī funkciju projektējums, lai izveidotu priekšstatu par projektētas un galvenās sistēmas mijiedarbību.

Ceturtajā daļā ir testēšanas dokumentācija, kas aptver sevī testēšanas rezultātus un projektētus testpiemērus, lai izvērtētu aprakstītas prasības sistēmai. Testēšana galvenokārt ir sadalīta divās daļās – vienībtestēšana un integrācijas testēšana. Piektā daļa ir projekta organizācijas apraksts, kas sniedz informāciju par projekta veikšanu un izvēlēto dzīves cikla modeli. Sestā daļa ir darbietilpības novērtējums, kas apraksta sistēmas prognozējamo izstrādes laiku. Septītā daļa ir kvalitātes nodrošināšanas apraksts, kas sniedz informāciju par principiem un noteikumiem, kas bija ievēroti, lai izveidotu kvalitatīvu dokumentāciju un kodu. Astotā daļa ir konfigurāciju pārvaldība, kas apraksta darba versiju kontroles rīkus.

1. VISPĀRĪGS APRAKSTS

1.1. Esošā stāvokļa apraksts

Lielāka atšķirība starp webAppOS un parasto operētājsistēmu ir tīmekļa atmiņa. Salīdzinot to ar klasisko brīvpiekļuves atmiņu, tīmekļa atmiņa tiek automātiski sinhronizēta starp visiem iesaistītajiem klientiem un serveriem, radot ilūziju par tieši vienu koplietojamo atmiņu. Tādu operētājsistēmu var uztvert kā daudzprocesoru sistēmu, kurā katram tīmekļa procesoram ir dažādas funkcijas, bet viena vienīga datu koplietojama atmiņa.

Pašlaik webAppOS strādā tikai Java vidē. Rakstītais kods Java programmēšanas valodā tiek pārveidots zema līmeņa baitkodā, kas dod iespēju palaist tīmekļa operētājsistēmu uz jebkura datora, kas atbalsta Java platformu [5]. Uz doto brīdi nav implementēts tāds tīmekļa procesors, kas strādātu un izsauktu klašu metodes, kas rakstītas python programmēšanas valodā.

Ir vairākas starptautiskās organizācijas, kas nodrošina savienojuma HTTPS izveidi starp klientu un serveri. Piemēram, DigiCert, Sectigo, Let's Encrypt, GoDaddy. Projektā webAppOS ietvaros tiek izmantoti uzņēmuma Let's Encrypt pakalpojumi, lai izveidotu drošu savienojumu globālajos tīklos jeb vietnēs, kas ir brīvi pieejamas katram interneta lietotājam. Tomēr Let's Encrypt un citi pakalpojumu sniedzēji nevar šifrēt savienojumus, kas tiek izveidoti lokālajos tīklos, jo pašam uzņēmumam nav iespējas pārbaudīt domēna vārda pieejamību globālajā tīmeklī.

Pašlaik ir iespējams manuāli šifrēt lietotāja mapes Ubuntu vidē, piemēram, ar fscrypt vai CryFS rīkiem. Tomēr, ja gala lietotājam nav pietiekami daudz pieredzes vai iespējas lietot webAppOS Ubuntu vidē, var rasties problēmas ar manuālo mapju šifrēšanu.

Uz doto brīdi lokālo savienojumu un mapju šifrēšanas mehānisms projektā webAppOS nav automatizēts. Lietotājam pašam ir jāģenerē publiskas un privātas atslēgas, piemēram, izmantojot OpenSSL kriptogrāfijas bibliotēku. Šifrēšanas mehānisma automatizēšana arī bija galvenais iemesls python procesora implementācijai.

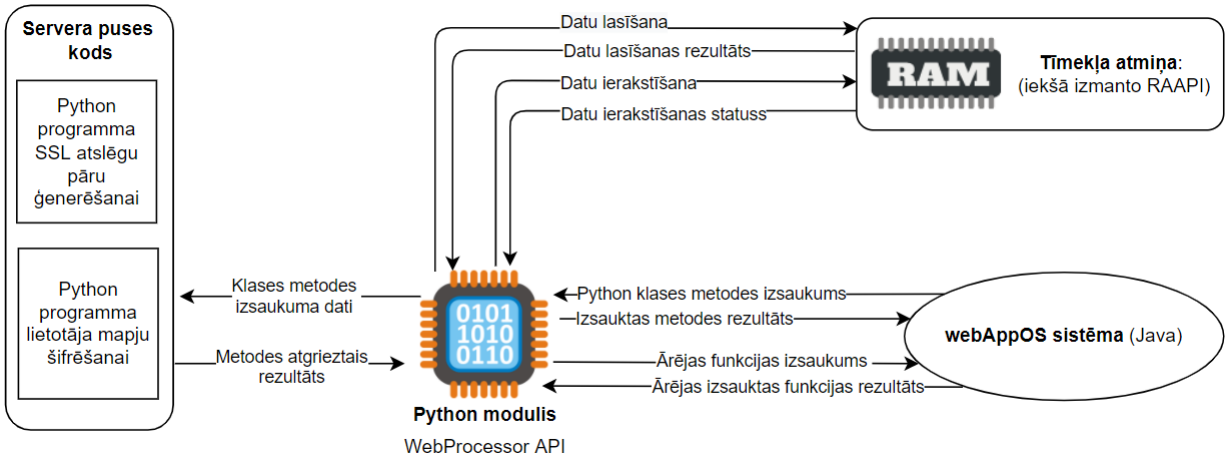
1.2. Pasūtītājs

Sistēma izstrādāta pēc tā vajadzības Latvijas Universitātes Matemātikas un Informātikas institūta projekta "Modeļu-bāzētā tīmekļa lietotņu infrastruktūra ar mākoņu tehnoloģiju

atbalstu" uzlabošanas ietvaros. Sistēma tika aprakstīta pēc autora iniciatīvas kvalifikācijas darba ietvaros.

1.3. Produkta perspektīva

Produkts nav neatkarīgs, tā ir daļa no Latvijas Universitātes Matemātikas un Informātikas institūta projekta “Modeļu-bāzētā tīmekļa lietotņu infrastruktūra ar mākoņu tehnoloģiju atbalstu", kas tika veidots webAppOS tīmekļa operētājsistēmas funkcionalitātes uzlabošanai. Python moduļa izstrāde ir veidota tāda veidā, lai nākotnē tīmekļa operētājsistēmā varētu izmantot arī citas uzrakstītas funkcijas python programmēšanas valodā.



1.1. att. **webAppOS, python moduļa savstarpējo sakaru un saskarņu diagramma**

Galvenās sistēmas un Python moduļa savstarpējus sakarus var aplūkot attēlā 1.1. Saņemot no webAppOS kādas python klases metodes izsaukumu, tiks izsaukts Python modulis kas realizē WebProcessor API. Turpmāk Python modulis pieslēdzas tīmekļa atmiņai, lai tā identificētu brīvu vietu, kur glabāsies izsauktas klases metodes dati. Piemēram, klases atribūtu vērtības. Tālāk metodes ievaddati jau servera pusē tiek padoti dotajai python klasei, lai izsauktu nepieciešamu metodi.

Python moduļa darba laikā var rasties nepieciešamība pēc ārējas funkcijas izsaukuma. Piemēram, saņemt un turpmāk izmantot funkcijas atgriezto rezultātu, kas ir uzrakstīta Java programmēšanas valodā. Tāda gadījumā Python modulis aizsūta pieprasījumu pēc ārējas funkcijas izsaukuma webAppOS vidē. Pēc kāda laika webAppOS aizsūta Python modulim atbildi. Python metodes darba laikā var vairākas reizes lasīt un ierakstīt datus tīmekļa atmiņā. Katru reizi tīmekļa atmiņa informēs par datu lasīšanas un ierakstīšanas statusu.

Beigās Python moduļa saņemtais metodes rezultāts būs aizsūtīts atpakaļ galvenajai sistēmai.

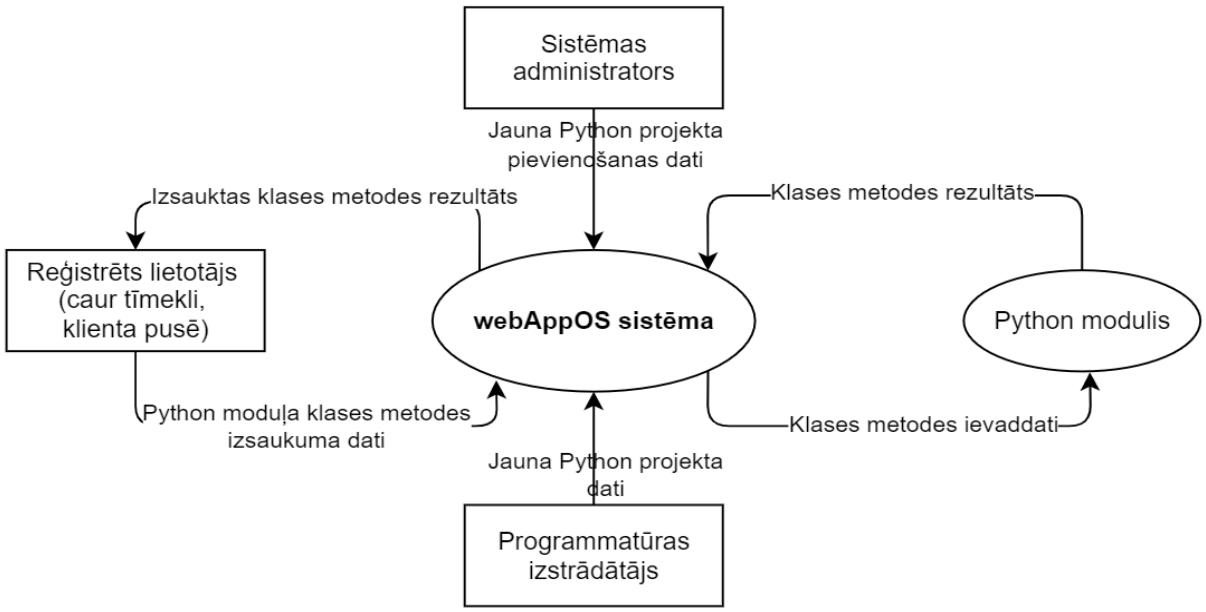
1.4. Darījumprasības

Sistēmas galvenās funkcijas un piedāvājumus webAppOS sistēmas lietotājiem var apskatīt zemāk:

1. iespēja izveidot jaunu SSL sertifikāta atslēgu konkrētajiem tīmekļa domēna vārdiem vai IP adresēm;
2. SSL sertifikāta un atslēgu pāra atjaunošana;
3. SSL sertifikāta publiskās atslēgas lasīšana;
4. SSL atslēgu pāra atrašanas vietas failu sistēmā definēšana;
5. SSL atslēgu pāra derīguma termiņa lasīšana;
6. domēna vārdu un IP adrešu definēšana, kas pieder noteiktajam SSL sertifikāta atslēgu pārim;
7. SSL atslēgu pāra izdevēja noteikšana;
8. darba mapes šifrēšana un atšifrēšana

Savukārt programmatūras izstrādātājiem ir iespējams pievienot jaunas klases un metodes Python modulim.

1.5. Sistēmas lietotāji



1.2. att. **0. līmeņa DPD**

Ir definētas trīs lietotāju grupas, tas ir, reģistrēts lietotājs, sistēmas administrators un programmatūras izstrādātājs. DPD var apskatīt 1.2. attēlā. Reģistrētam lietotājam ir tiesības lietot implementēta Python moduļa metodes. Sākumā reģistrēts lietotājs aizsūta atbilstošo pieprasījumu webAppOS vidē. Pēc tam galvenā sistēma savienojas ar Python moduli, kas savukārt aizsūtīs atpakaļ tīmekļa operētājsistēmai izsauktās metodes rezultātu. Programmatūras

izstrādātājs varēs pievienot jaunus Python projektus webAppOS vidē, kurus turpmāk uzstādīs sistēmas administrators. Tiek pieņemts, ka programmatūras izstrādātajam un sistēmas administratoram ir atbilstošs izglītības līmenis, lai, attiecīgi, izveidotu un pievienotu jaunus python projektus.

1.6. Vispārējie ierobežojumi

Python moduļa integrācija un izmantošana ietver sevī vairākus ierobežojumus:

1. reģistrētam lietotājam pieejams dators ar vienu no sekojošam operētājsistēmām: Windows, Linux Ubuntu, MacOS, kuram ir pieslēgta strādājošā klaviatūra, pele un ekrāns;
2. Python moduli ir iespējams izmantot, ja reģistrētam lietotājam ir viens no minētiem datora procesoriem: x86 (32 bitu), x86_64 (amd64) vai arm64 (aarch64);
3. pieejams stabils interneta savienojums, lai palaistu webAppOS;
4. iepriekš instalēta Java virtuālās mašīnas 11 vai jaunāka versija darbam ar webAppOS un Python moduli;
5. reģistrētam lietotājam ir nokonfigurēts savs lietotāja konts webAppOS [6], lai izmantotu webAppOS funkcijas, to starpā arī Python moduļa iespējas;
6. Python moduļa programmēšanas valodas versija ir Python3.

1.7. Pieņēmumi un atkarības

Python moduļa integrācija un izmantošana ietver sevī vairākus pieņēmumus:

1. sistēmas administratoram ir pietiekami daudz tiesību, lai integrētu Python moduļa funkcijas (*root* vai *sudo*);
2. reģistrētam lietotājam ir pietiekami daudz atmiņas, lai lejupielādētu webAppOS sistēmu;
3. serverī ir pieejams vismaz 1GB brīvpiekļuves atmiņas, kā arī 1GHz procesors.

2. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA

2.1. Prasību kopsavilkums

Sakarā ar to, ka webAppOS strādā java vidē, Python modulis tiks uzprojektēts, lai izmantotu python valodā uzrakstītas metodes. Modulim ir jābūt universālam, lai turpmāk papildinātu to ar jaunām funkcijām, kā arī saprotamam realizācijā, lai citiem programmētājiem būtu viegli saprast galvenās ārējas saskarnes ar webAppOS.

Šajā darba ietvaros tiks uzprojektēts mapju šifrēšanas modulis un SSL atslēgu pāra izveidošanas modulis, lai automatizētu gala lietotāja darbu. Līdz ar to Python modulim ir jāspēj ģenerēt jaunus, kā arī atjaunināt jau eksistējošus SSL atslēgu pārus attiecīgiem domēna vārdiem un IP adresēm. Python modulim automātiski jāprot šifrēt un atšifrēt attiecīgas darba mapes pēc sistēmas reģistrēta lietotāja pieprasījuma.

2.2. Funkcionālās prasības

Šī darba ietvaros PPS daļā tiks apskatītas Python moduļa prasības un to saistība ar galveno webAppOS sistēmu. Datu plūsmu diagrammu Python modulim var apskatīt 2.1. attēlā. Vispirms reģistrēts lietojas pieslēdzas galvenajai sistēmai caur tīmekli klienta pusē, tas ir, autentificējas sistēmā ar savu lietotājvārdu un paroli. Tālāk, lai izmantotu metodes, kas rakstītas python programmēšanas valodā, tiks izmantots tīmekļa procesors. WebAppOS specifikācija nosaka to, ka starp tīmekļa procesoru un galveno sistēmu tiks pārsūtīti JSON formātā.

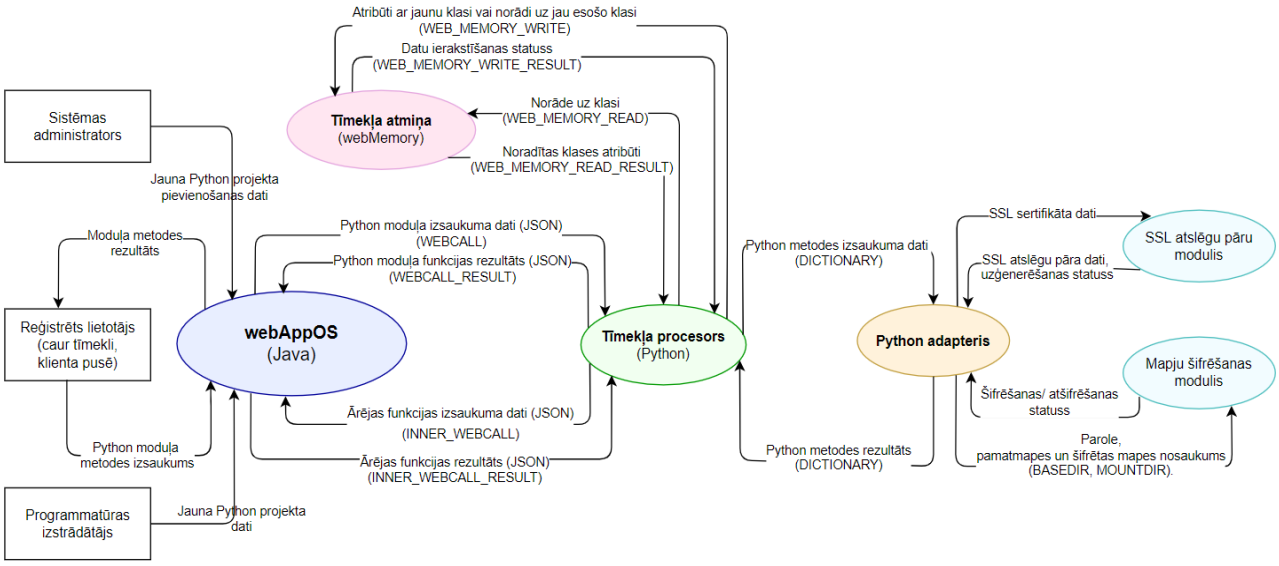
Sākumā webAppOS aizsūta tīmekļa procesoram Python moduļa un attiecīgas metodes izsaukuma datus (WEBCALL). Ja tīmekļa procesoram būs aizsūtīts tikai klases nosaukums, tad tas pieslēgsies tīmekļa atmiņai, lai ierakstītu tajā jaunas klases datus (WEB_MEMORY_WRITE). Pēc pieprasījuma apstrādes tīmekļa atmiņa informēs procesoru par datu ierakstīšanas statusu un norādi uz atmiņu, kur glabāsies klases dati (WEB_MEMORY_WRITE_RESULT). Savukārt, ja python procesors saņems gan klases, gan metodes nosaukumu, tas nozīmēs, ka tīmekļa atmiņā jau eksistē norādītas klases objekts un atribūtu vērtības, un procesors lasīs šos datus no tīmekļa atmiņas (WEB_MEMORY_READ). Tīmekļa atmiņa aizsūtīs attiecīgus klases datus (WEB_MEMORY_READ_RESULT).

Turpmāk tīmekļa procesors pārsūta izsauktas metodes datus python adapterim. Adapteris saņem un lasa datus, lai noteiktu kādu moduli ir jāizmanto metodes izpildei. Šī darba ietvaros tika uzprojektēts SSL atslēgu pāra modulis un mapju šifrēšanas modulis. Izsaucot SSL

atslēgu pāra moduli, tam tiks aizsūtītш SSL sertifikāta dati, piemēram, domēna vārdu un IP adrešu kopa. Savukārt mapju šifrēšanas modulim tiks aizsūtīta parole mapju šifrēšanai, kā arī pamatmapes un šifrētas mapes datus.

Ja kāda Python moduļa metodes izpildei būs vajadzība pēc kāda cita moduļa funkcijas, piemēram, Java programmēšanas valodā, tad tīmekļa procesors aizsutis datus webAppOS par attiecīgas funkcijas izpildi (INNER_WEBCALL). Pēc kāda laika procesors saņems funkcijas rezultātu (INNER_WEBCALL_RESULT).

Beigās saņemot python metodes rezultātu, adapteris to atgriež atpakaļ tīmekļa procesoram, bet tas savukārt rezultātu aizsutīs webAppOS (WEBCALL_RESULT). Python modulim var pievienot jaunus projektus, ko arī veic programmatūras izstrādātājs. Beigās projektu webAppOS videi pievieno sistēmas administrators.



2.1. att. **1. līmeņa DPD**

2.2.1. Paziņojumu saraksts

Paziņojumu grupu definējumus var apskatīt tabulā 2.1. Ir definētas četras galvenās paziņojumu grupas: ievades kļūda (INPUT), webAppOS un procesora datu apmaiņas kļūda (COMMUNICATION), vispārējo ierobežojumu kļūda (RESTRICTION), kas definētas 1.6. sadaļā un vispārīga kļūda (GENERAL).

Katrs paziņojuma identifikators sastāv no pieciem simboliem. Pirmais simbols sakrīt ar paziņojuma grupas apzīmējumu. Otrais simbols ir domuzīme, kas atdala paziņojuma grupas simbolu no skaitītāja. Pārējie trīs simboli ir skaitītāja apzīmējums veselos pozitīvos skaitļos. Ja

paziņojums satur tekstu <Python kļūda>, tad projektējumā šis teksts tiks aizvietots ar python interpretatora kļūdu. Paziņojumu saraksts ir definēts tabulā 2.2.

2.1. tabula. **Paziņojumu grupas**

Paziņojuma grupa	Grupas definējums
I *(INPUT)*	Kļūda ievadē
C *(COMMUNICATION)*	WebAppOS un tīmekļa procesora datu apmaiņas/komunikācijas kļūda
R *(RESTRICTION*)	Vispārējo ierobežojumu kļūda
G *(GENERAL)*	Vispārīga kļūda

2.2. tabula. **Paziņojumu saraksts**

Paziņojuma identifikators	Funkcijas, kuras izmanto paziņojumu	Paziņojuma teksts
I-001	WP_WEBCALL_NEW WP_WEBCALL_OLD	“Invalid input json format in WEBCALL”
I-002	WP_WEBCALL_NEW WP_WEBCALL_OLD	"Invalid input json format in WEBCALL: <Python kļūda> "
I-003	PA_OBJECT_CREATE PA_METHOD_CALL	"Error during file import in the adapter: <Python kļūda>"
I-004	PA_OBJECT_CREATE PA_METHOD_CALL	" Error during file import in the adapter: The argument <argument> was not passed to the function"
I-005	KP_SINGLE_CERT KP_DIFFERENT_CERT	“All invalid inputs or empty input”
I-006	KP_DEFINE_CONTENT KP_DEFINE_ISSUER	"Error during reading PEM file in: <Directory>."
C-001	WP_WB_WRITE	“Unable to serialize the WEB_MEMORY_WRITE input object to json <python kļūda>”

Paziņojuma identifikators	Funkcijas, kuras izmanto paziņojumu	Paziņojuma teksts
C-002	WP_WB_WRITE WP_WB_READ	“web processor communication error”
C-003	WP_WB_WRITE	“Unable to load WEB_MEMORY_WRITE_RESULT output from webMemory <python kļūda>”
C-004	WP_WB_READ	“Unable to serialize the WEB_MEMORY_READ input object to json <python kļūda>”
C-005	WP_WB_READ	“Unable to load WEB_MEMORY_READ_RESULT output from webMemory <python kļūda>”
C-006	WP_INNER_WEBCALL	“Unable to serialize the INNER_WEBCALL input object to json in <methodName>”
C-007	WP_INNER_WEBCALL	“Unable to load INNER_WEBCALL_RESULT output from webMemory in <methodName>”
R-001	KP_SINGLE_CERT KP_DIFFERENT_CERT	“The program does not support your operating system: <OS>”
R-002	KP_SINGLE_CERT KP_DIFFERENT_CERT	“The program does not support your CPU: <CPU> “
G-001	KP_SINGLE_CERT KP_DIFFERENT_CERT	“Error during creating SSL KeyPair”
G-002	KP_SINGLE_CERT KP_DIFFERENT_CERT	“Error during creating SSL KeyPair <Python kļūda>”
G-003	KP_READ_PUBLIC_NAME KP_READ_PRIVATE_NAME	"Firm path is not accesible"

Paziņojuma identifikators	Funkcijas, kuras izmanto paziņojumu	Paziņojuma teksts
G-004	KP_READ_PUBLIC_NAME KP_READ_PRIVATE_NAME	'Directory <Directory> is not accesible'
G-005	KP_READ_PUBLIC_NAME KP_READ_PRIVATE_NAME	"Error during finding KeyPair in path: <Directory>, <Python kļūda>"
G-006	KP_READ_PUBLIC_FILE	'cert.pem file in <Directory> is not accessible for reading'
G-007	KP_EXPIRATION_DATE	"cannot import crypto library to decode SSL cert.pem file <python kļūda>"
G-008	DE_INIT_MOUNT	"<Directory> is not empty, cannot init CryFS"
G-009	DE_INIT_MOUNT	"<Directory> has already mounted, cannot run init with the mounted folder"
G-010	DE_INIT_MOUNT DE_MOUNT	"Password cannot be empty"
G-011	DE_INIT_MOUNT DE_MOUNT	"Base directory can't be inside the mount directory"
G-012	DE_INIT_MOUNT DE_MOUNT	"Error during mounting: <Python kļūda>"
G-013	DE_UNMOUNT	"<ErrorLine>, error with mountdir: <MountDirectory"
G-014	DE_UNMOUNT	"Unmounted directory is not empty, something went wrong in <MountDirectory>"
G-015	DE_UNMOUNT	"Failed to unmount directory <MountDirectory> by CryFS"
G-016	DE_MOUNT	"Basedir not found before CryFS mounting"

Paziņojuma identifikators	Funkcijas, kuras izmanto paziņojumu	Paziņojuma teksts
G-017	DE_MOUNT	"CryFS configuration file not found in basedir <BaseDirectory>"
G-018	DE_MOUNT	"Cannot mount <MountDirectory> because it is not empty"
G-019	DE_MOUNT	"<MountDirectory> has already mounted, cannot mount it again"

2.2.2. Funkciju sadalījums pa moduļiem

Moduļi un moduļu funkcijas, kas tiek projektēti darbā ir pieejami tabulā 2.3.

2.3. tabula. **Moduļu funkciju saraksts**

Moduļa / artefakta nosaukums	Moduļa identifikators	Funkcijas nosaukums	Funkcijas identifikators
Tīmekļa procesors (python)	WP	Jaunas metodes izsaukums (WEBCALL)	WP_WEBCALL_NEW
		Iepriekš izmantotas metodes izsaukums (WEBCALL)	WP_WEBCALL_OLD
		Datu ierakstīšana tīmekļa atmiņā (WEB_MEMORY_WRITE)	WP_WB_WRITE
		Datu lasīšana no tīmekļa atmiņas (WEB_MEMORY_READ)	WP_WB_READ
		Ārējas funkcijas izsaukums (INNER_WEBCALL)	WP_INNER_WEBCALL
Python adapteris	PA	Python objekta izveidošana	PA_OBJECT_CREATE
		Python metodes izsaukums	PA_METHOD_CALL

Moduļa / artefakta nosaukums	Moduļa identifikators	Funkcijas nosaukums	Funkcijas identifikators
SSL atslēgu pāru modulis	KP	Kopīga SSL atslēgu pāra izveidošana	KP_SINGLE_CERT
		Dažādu SSL atslēgu pāru izveidošana	KP_DIFFERENT_CERT
		Publiskas atslēgas atrašanas vietas lasīšana	KP_READ_PUBLIC_NAME
		Privātas atslēgas atrašanas vietas lasīšana	KP_READ_PRIVATE_NAME
		Publiskas atslēgas faila lasīšana	KP_READ_PUBLIC_FILE
		Atslēgu pāra derīguma termiņa lasīšana	KP_EXPIRATION_DATE
		Atslēgu pāra atjaunošana	KP_UPDATE_CERT
		Atslēgu pāra domēna vārdu un IP adrešu lasīšana	KP_DEFINE_CONTENT
		Atslēgu pāra izdevēja lasīšana	KP_DEFINE_ISSUER
Mapju šifrēšanas modulis	DE	Mapes izveidošana turpmākai šifrēšanai	DE_INIT_MOUNT
		Mapes šifrēšana	DE_UNMOUNT
		Mapes atšifrēšana	DE_MOUNT

2.2.3. Tīmekļa procesors

2.2.3.1. Jaunas metodes izsaukums (WEBCALL)

Identifikators

WP_WEBCALL_NEW

Ievads

Jaunas klases instances (objekta) un atribūtu ierakstīšana tīmekļa atmiņā.

Ievade

Tīmekļa procesors saņem no webAppOS:

1. klases nosaukums (simbolu virkne) ;
2. klases atribūtu kopa konstruktora izveidošanai (simbolu virknes).

Apstrāde

Sākumā notiek pārbaude, vai saņemtie dati no webAppOS, tas ir, klases nosaukums un atribūtu kopa atbilst JSON formātam. Ja ievade atbilst formātam, tad tīmekļa procesors savienojas ar tīmekļa atmiņu, lai ierakstītu tajā klasi un atribūtu kopu, kas sīkāk aprakstīts 2.2.3.3. nodaļā (WP_WB_WRITE**).** Ja dati tiks ierakstīti veiksmīgi, tad tīmekļa procesors saņems atbilstošo rezultātu no tīmekļa atmiņas.

Izvade

Ja datu ierakstīšana tīmekļa atmiņā notika veiksmīgi vai tika saņemta kļūda, tad tīmekļa procesors atgriež iegūto rezultātu no tīmekļa atmiņas webAppOS operētājsistēmai.

Paziņojumi

1. Ja ievade neatbilst JSON formātam, tad tiks attēlots paziņojums I-001;
2. ja JSON nesatur visus nepieciešamos datus vai dati ir kļūdaini, tad tiks izvadīts paziņojums I-002 ar tekstu, kas atbilst python interpretatora kļūdai.

2.2.3.2. Iepriekš izmantotas metodes izsaukums (WEBCALL)

Identifikators

WP_WEBCALL_OLD

Ievads

Iepriekš izmantotas klases instances (objekta) metodes izsaukums python modulī.

Ievade

Tīmekļa procesors saņem no webAppOS:

1. klases nosaukums (simbolu virkne) ;
2. klases metodes argumentu kopa (simbolu virknes);
3. norāde uz tīmekļa atmiņu, kur glabājas objekta atribūti (vesels pozitīvs skaitlis);
4. klases metodes precīza atrašanas vieta (simbolu virkne).

Apstrāde

Sākumā notiek pārbaude, vai saņemtie dati no webAppOS, tas ir, klases nosaukums, argumentu kopa un norāde atbilst JSON formātam. Ja ievade atbilst formātam, tad tīmekļa procesors savienojas ar tīmekļa atmiņu, lai nolasītu klases atribūtus konstruktora izveidošanai, kas tiks aprakstīts 2.2.3.4. nodaļā (WP_WB_READ**).** Ja datu ieguve notika veiksmīgi, tad tīmekļa procesors izsauc python adapteri jauna objekta izveidošanai, kas tiks aprakstīts 2.2.4.1. nodaļā (PA_OBJECT_CREATE). Pēc tam izsauc klases metodi, ko apraksta 2.2.4.2. nodaļa (PA_METHOD_CALL). Beigās, ja klases metode tiks izpildīta veiksmīgi, tad tīmekļa adapteris atgriež klases metodes rezultātu (PA_METHOD_CALL) tīmekļa procesoram. Citādi, tiek atgriezta kļūda.

Izvade

Ja python klases metodes izpilde notika veiksmīgi vai tika saņemta kļūda, tad tīmekļa procesors pārsūtīs iegūto rezultātu no python adaptera webAppOS operētājsistēmā.

Paziņojumi

1. Ja ievade neatbilst JSON formātam, tad tiks attēlots paziņojums I-001;
2. ja neizdodas izlasīt ievades datus JSON formātā, tad tiks izvadīts paziņojums I-002 ar tekstu, kas atbilst python interpretatora kļūdai;
3. ja metodes izsaukšanas laikā notika kļūda, tiek atgriezts teksts, kas atbilst python interpretatora kļūdai.

2.2.3.3. Datu ierakstīšana tīmekļa atmiņā (WEB_MEMORY_WRITE)

Identifikators

WP_WB_WRITE

Ievads

Saņemto no webAppOS vai python adaptera datu ierakstīšana tīmekļa atmiņā.

Ievade

Ja objekts ir jauns un nav iepriekš izveidots tīmekļa atmiņā, tad tīmekļa procesors saņem no webAppOS vai python adaptera sekojošo informāciju:

1. ievades dati (simbolu virknes) (var būt gan atribūtu kopa, gan metodes argumentu kopa);
2. klases nosaukums (simbolu virkne).

Ja objekts ir iepriekš izveidots tīmekļa atmiņā, tad:

1. ievades dati (simbolu virknes) (var būt gan atribūtu kopa, gan metodes argumentu kopa);
2. norāde uz tīmekļa atmiņu, kur glabājas iepriekš saglabāti dati (vesels pozitīvs skaitlis).

Apstrāde

Sākumā ievades dati tiks konvertēti JSON formātā. Ja JSON formāts ir veiksmīgi iegūts, tad dati tiks virzīti tīmekļa atmiņā. Tālāk tīmekļa atmiņa atgriezīs procesoram JSON objektu, kas satur vai nu norādi, kur glabājas ierakstītie dati vai nu kļūdas paziņojumu. Beigās iegūtais JSON objekts tiks konvertēts vārdnīcas formātā.

Izvade

Nav.

Paziņojumi

1. Ja neizdodas ievades datus pārveidot JSON formātā, tad atgriezts paziņojums C-001 ar tekstu, kas sakrīt ar python kļūdu;
2. ja tīmekļa atmiņa nekorekti informēs tīmekļa procesoru par ierakstīšanas pabeigšanu, tad tīmekļa procesors attēlos paziņojumu C-002;
3. ja neizdodas rezultātu pārveidot vārdnīcas formātā, tad tiks atgriezts paziņojums C-003 ar tekstu, kas sakrīt ar python kļūdu.

2.2.3.4. Datu lasīšana no tīmekļa atmiņas (WEB_MEMORY_READ)

Identifikators

WP_WB_READ

Ievads

Tīmekļa atmiņā esošo datu lasīšana.

Ievade

Tīmekļa procesors no webAppOS vai python adaptera saņems norādi uz tīmekļa atmiņu, kur glabājas iepriekš saglabāti dati (vesels pozitīvs skaitlis).

Apstrāde

Sākumā ievades dati tiks konvertēti JSON formātā. Ja JSON formāts ir veiksmīgi iegūts, tad dati tiks virzīti tīmekļa atmiņā. Tālāk tīmekļa atmiņa atgriezīs procesoram vai nu visu esošo informāciju, kas glabājas noradītajā atmiņas vietā, vai nu kļūdas paziņojumu. Beigās dati tiks konvertēti no JSON vārdnīcas formātā.

Izvade

Nav.

Paziņojumi

1. Ja neizdodas ievades datus pārveidot JSON formātā, tad būs attēlots paziņojums C-004 ar tekstu, kas sakrīt ar python interpretatora kļūdu;
2. ja tīmekļa atmiņa nekorekti informēs tīmekļa procesoru par datu izguves pabeigšanu, tad tīmekļa procesors attēlos paziņojumu C-002;
3. ja beigās neizdodas rezultātu pārveidot vārdnīcas formātā, tad tiks attēlots paziņojums C-005 ar tekstu, kas sakrīt ar python kļūdu.

2.2.3.5. Ārējas funkcijas izsaukums (INNER_WEBCALL)

Identifikators

WP_INNER_WEBCALL

Ievads

Funkcijas izsaukšana, kas uzprogrammēta cita programmēšanas valodā (ne Python), piemēram, Java.

Ievade

WebAppOS saņems no tīmekļa procesora datus:

1. norāde tīmekļa atmiņā, kur glabājas dati par izsaukto funkciju (vesels pozitīvs skaitlis);
2. izsaucamās klases nosaukums (simbolu virkne);
3. izsaucamās metodes nosaukums (simbolu virkne);
4. izsaucamās metodes argumenti (simbolu virknes);

Apstrāde

Sākumā ievades dati tiks konvertēti JSON formātā. Ja JSON formāts ir veiksmīgi iegūts, tad dati tiks virzīti webAppOS operētājsistēmā. Pēc noradītas funkcijas darba pabeigšanas, tīmekļa procesors saņems no webAppOS funkcijas rezultātu. Beigās iegūtie dati atpakaļ tiks konvertēti no JSON vārdnīcas formātā.

Izvade

WebAppOS atgriezīs tīmekļa procesoram funkcijas rezultātu JSON formātā.

Paziņojumi

1. Ja neizdodas ievades datus pārveidot JSON formātā, tad būs atgriezts paziņojums C-006 ar tekstu, kas sakrīt ar esošas metodes nosaukumu;
2. ja tīmekļa atmiņa nekorekti informēs tīmekļa procesoru par metodes izpildes beigām tad tīmekļa procesors attēlos paziņojumu C-002;
3. ja beigās neizdodas rezultātu pārveidot vārdnīcas formātā, tad tiks atgriezts paziņojums C-007 ar tekstu, kas sakrīt ar esošas metodes nosaukumu.

2.2.4. Python adapteris

2.2.4.1. Python objekta izveidošana

Identifikators

PA_OBJECT_CREATE

Ievads

Metode izveido python objektu, kas ļaus palaist attiecīgas klases metodes pēc reģistrēta lietotāja pieprasījuma.

Ievade

Adapteris saņem no python procesora sekojošu informāciju:

1. klases metodes precīza atrašanas vieta (simbolu virkne). To starpā arī *fileName* – faila ".py" paplašinājumā nosaukums, kurā glabājas nepieciešama klase objekta izveidošanai;
2. *className* (simbolu virkne)– klases nosaukums, kurai ir jāizveido objekts;
3. klases *className* konstruktora argumentu kopa (simbolu virknes), kas tika iepriekš izlasīta no tīmekļa atmiņas (WP_WEBCALL_OLD, WP_WB_READ);

Apstrāde

Paša sākumā python adapteris sakārto saņemtu konstruktora argumentu kopu tāda veidā, lai ievades datu secība sakristu ar klases konstruktora argumentu secību. Tālāk adapteris pēc norādītiem metodes datiem, tas ir, faila un klases nosaukuma, meklē klasi *className*, kas savukārt glabājas failā *fileName.py*. Beigās adapteris izveido attiecīgu python objektu.

Izvade

Ja meklētie faila un klases nosaukumi ir veiksmīgi atrasti un python objekts ir izveidots, tad tiks atgriezta norāde uz jaunizveidoto python objektu.

Paziņojumi

1. Ja meklētais fails vai klases nosaukums nav atrasts, tad tiks atgriezts paziņojums I-003;
2. ja ievadē trūkst kāda konstruktora argumenta, tad tiks atgriezts paziņojums I-004 ar tekstu, kas atbilst šim argumentam.

2.2.4.2. Python metodes izsaukums

Identifikators

PA_METHOD_CALL

Ievads

Python programmēšanas valodā uzrakstītas klašu metodes izsaukšana.

Ievade

Adapteris saņem no python procesora datus:

1. *methodName* (simbolu virkne)– klases metodes nosaukums;
2. metodes *methodName* argumentu kopa, kas tika iepriekš saņemta no webAppOS (WP_WEBCALL_OLD);
3. python objekts, kas tika izveidots 2.2.4.1. nodaļā (PA_OBJECT_CREATE)

Apstrāde

Sākumā python adapteris sakārto saņemtus metodes argumentus tāda veidā, lai ievades datu secība sakristu ar klases metodes argumentu secību. Tālāk, ja visi argumenti ir veiksmīgi izlasīti, tiks izsaukta attiecīga python klases metode. Beigās adapteris saņem un atgriež tīmekļa procesoram python metodes rezultātu.

Izvade

Ja saņemtā python metode ir atrasta, tad tiks atgriezts izpildītas metodes rezultāts. Kļūdas gadījumā, tiek atgriezts kļūdas ziņojums.

Paziņojumi

1. Ja meklētas klases metodes nosaukums nav atrasts, tad būs atgriezta attiecīga python interpretatora kļūda;
2. ja ievadē trūkst kāda metodes argumenta, tad tiks atgriezts paziņojums I-004 ar tekstu, kas atbilst argumentam.

2.2.5. SSL atslēgu pāru modulis

2.2.5.1. Kopīga SSL atslēgu pāra izveidošana

Identifikators

KP_SINGLE_CERT

Ievads

Metode ģenerē vienu vienīgu publisko un privāto atslēgu. Atslēgu pāris var būt izmantots jebkuram noradītam domēnam, IP adresei. (iespējami vairāki domēni un IP adreses).

Ievade

Tiks saņemti ievades dati no python adaptera:

1. domēnu vārdu kopa (simbolu virknes);
2. IP adrešu kopa (simbolu virknes);
3. sertifikācijas iestādes (uzņēmuma) nosaukums *caName*, kura vārdā tiks parakstīts jaunizveidotais SSL atslēgu pāris (simbolu virkne).

Apstrāde

Sākumā notiek pārbaude, vai reģistrēta lietotāja operētājsistēma, procesors atbilst prasībām, kas aprakstītas 1.6. nodaļā. Ja atbilst, tad tiks izveidota mape *caName*, kur glabāsies izveidotais atslēgu pāris. Tālāk pēc kārtas jālasa ievadīti domēna vārdi un IP adreses. Privāta un publiskā atslēga būs izveidota tikai korektiem domēna vārdiem un tikai tam IP adresēm, kas atbilst IPv4 standartam. Ja tāds atslēgu pāris jau eksistē, tad tas būs atjaunots tikai pie nosacījuma, ka jauns atslēgu pāris izveidots bez kļūdām.

Izvade

Ja atslēgu pāris ir izveidots, tad būs atgriezts objekts ar izveidota atslēgu pāra atribūtiem: domēna vārdi, IP adreses un *caName*. Objektam jābūt tādam, ka tam būtu iespējams izsaukt 2.2.5.3 – 2.2.5.7. nodaļās minētās metodes.

Paziņojumi

1. Ja lietotāja operētājsistēma nav Windows, Linux vai MacOS, tad būs atgriezts R-001;
2. ja lietotāja procesors neatbilst prasībām, būs atgriezts R-002 ar procesora nosaukumu;
3. ja visi domēna vārdi un IP adreses nav korektas, tiks atgriezts I-005 paziņojums;
4. ja notiks atslēgu pāra ģenerēšanas kļūda, būs atgriezts G-001 paziņojums;
5. ja notiks kļūda ar jauno atslēgu izvietošanu mapē *caName*, būs atgriezts G-002.

2.2.5.2. Dažādu SSL atslēgu pāru izveidošana

Identifikators

KP_DIFFERENT_CERT

Ievads

Metode ģenerē atsevišķus SSL atslēgu pārus katram noradītajam domēna vārdam un katrai IP adresei.

Ievade

Tiks saņemti ievades dati no python adaptera:

1. domēnu vārdu kopa, kuriem ir jāizveido atslēgu pāri (simbolu virknes);
2. IP adrešu kopa (simbolu virknes);
3. sertifikācijas iestādes (uzņēmuma) nosaukums *caName*, kura vārdā tiks parakstīts jaunizveidotais SSL atslēgu pāris (simbolu virkne).

Apstrāde

Sākumā notiek pārbaude, vai reģistrēta lietotāja operētājsistēma un procesors atbilst norādītajam prasībām, kas aprakstītas 1.6. nodaļā. Ja atbilst, tad tiks izveidota mape ar nosaukumu *caName*, kur glabāsies izveidotie atslēgu pāri. Tālāk pēc kārtas tiks lasīti ievadīti domēna vārdi, ja domēna vārds ir korekts, tad tiks uzģenerēta atbilstošā privāta un publiskā atslēga. Pēc tam tiks lasīta katra IP adrese. Ja tā atbilst IPv4 standartam, tad arī tai būs uzģenerēts savs atslēgu pāris. Ja kāds no atslēgu pārim jau eksistē, tad tas būs atjaunots, tomēr, ja notiks kļūda, tad mapē saglabāsies iepriekš izveidotais atslēgu pāris.

Izvade

Ja vismaz viens atslēgu pāris ir uzģenerēts, tad tiks atgriezta *boolen True* vērtība. Citādi, tiek atgriezta vērtība *False*.

Paziņojumi

1. Ja lietotāja operētājsistēma nav Windows, Linux, MacOS, tad būs atgriezts R-001;
2. ja lietotāja procesors neatbilst prasībām, būs atgriezts R-002 ar procesora nosaukumu;
3. ja visi domēna vārdi un IP adreses nav korektas, tiks atgriezts I-005 paziņojums;
4. ja notiks vismaz viena atslēgu pāra ģenerēšanas kļūda, būs atgriezts G-001;
5. ja notiks kļūda, kas ir saistīta ar jauno atslēgu izvietošanu mapē *caName*, būs atgriezts G-002 paziņojums ar tekstu, kas sakrīt ar python interpretatora kļūdu.

2.2.5.3. Publiskas atslēgas atrašanas vietas lasīšana

Identifikators

KP_READ_PUBLIC_NAME

Ievads

Metode definē un atgriež publiskas atslēgas atrašanas vietu kopā ar faila nosaukumu.

Ievade

Iepriekš izveidots objekts tīmekļa atmiņā (sk. 2.2.5.1. nodaļu) (KP_SINGLE_CERT), kas ietvers sevī sekojošos datus:

1. izveidota atslēgu pāra domēnu vārdu kopa (simbolu virknes);
2. izveidota atslēgu pāra IP adrešu kopa (simbolu virknes);
3. izveidota atslēgu pāra uzņēmuma nosaukums *caName* (simbolu virkne).

Apstrāde

Izmantojot saņemto *caName,* tiks definēta atslēgu pāra atrašanas vieta (mape) uz datora. Sakarā ar to, ka vienam uzņēmumam var būt vairāki SSL sertifikāti, tiks izmantoti ievades dati par domēna vārdiem un IP adresēm, lai precīzi definētu atslēgu pāra mapi. Tā kā vienmēr ievades dati sniegs informāciju tieši par vienu atslēgu pāri, tas ir, vienu privāto un publisko atslēgu, vienmēr varēs precīzi definēt publiskas atslēgas atrašanas vietu.

Izvade

Ja publiskai atslēgai ir veiksmīgi noteikta tā atrašanas vieta, būs atgriezts ceļš kopā ar faila nosaukumu (simbolu virkne). Citādi, tiks atgriezts kļūdas paziņojums.

Paziņojumi

1. Ja nevar atvērt *caName* mapi, tiks atgriezts G-003 paziņojums;
2. ja varēs atvērt mapi *caName,* bet nevarēs atvērt kādu no apakšmapēm, kas veido ceļu līdz publiskas atslēgas failam, būs atgriezts G-004 paziņojums ar ceļa, kuru nav iespējams atvērt, nosaukumu;
3. ja noteiktajā mapes *caName* apakšmapē netika atrasts publiskas atslēgas fails, tad būs atgriezts G-005 paziņojums ar ceļa nosaukumu un tekstu, kas sakrīt ar python interpretatora kļūdu.

2.2.5.4. Privātas atslēgas atrašanas vietas lasīšana

Identifikators

KP_READ_PRIVATE_NAME

Ievads

Metode definē un atgriež privātas atslēgas atrašanas vietu kopā ar faila nosaukumu.

Ievade

Iepriekš izveidots objekts tīmekļa atmiņā (sk. 2.2.5.1. nodaļu) (KP_SINGLE_CERT), kas ietvers sevī sekojošos datus:

1. izveidota atslēgu pāra domēnu vārdu kopa (simbolu virknes);
2. izveidota atslēgu pāra IP adrešu kopa (simbolu virknes);
3. izveidota atslēgu pāra uzņēmuma nosaukums *caName* (simbolu virkne).

Apstrāde

Izmantojot saņemtu *caName*, tiks definēta atslēgu pāra mape. Ir zināms, ka vienam uzņēmumam var būt daudz SSL sertifikātu, tāpēc tiks izmantoti ievades dati par domēna vārdiem un IP adresēm, lai precīzi definētu atslēgu pāra mapi. Tā kā vienmēr ievades dati sniegs informāciju tieši par vienu atslēgu pāri, tas ir, vienu privāto un publisko atslēgu, vienmēr varēs precīzi definēt privātas atslēgas atrašanas vietu.

Izvade

Ja privātai atslēgai ir veiksmīgi atrasta tā atrašanas vieta, būs atgriezts ceļš līdz privātai atslēgai kopā ar faila nosaukumu (simbolu virkne). Citādi, tiks atgriezts kļūdas paziņojums.

Paziņojumi

1. Ja nevarēs atvērt *caName* mapi, tiks atgriezts G-003 paziņojums;
2. ja varēs atvērt mapi *caName,* bet nevarēs atvērt kādu no apakšmapēm, kas veido ceļu līdz privātas atslēgas failam, būs atgriezts G-004 paziņojums ar ceļa, kuru nav iespējams atvērt, nosaukumu;
3. ja noteiktajā mapes *caName* apakšmapē nebūs atrasts privātas atslēgas fails, tad būs atgriezts G-005 paziņojums ar ceļa nosaukumu un tekstu, kas sakrīt ar python interpretatora kļūdu.

2.2.5.5. Publiskas atslēgas faila lasīšana

Identifikators

KP_READ_PUBLIC_FILE

Ievads

Metode lasa un atgriež publiskas atslēgas faila saturu.

Ievade

Iepriekš izveidots objekts tīmekļa atmiņā (sk. 2.2.5.1. nodaļu) (KP_SINGLE_CERT), kas ietvers sevī sekojošos datus:

1. izveidota atslēgu pāra domēnu vārdu kopa (simbolu virknes);
2. izveidota atslēgu pāra IP adrešu kopa (simbolu virknes);
3. izveidota atslēgu pāra uzņēmuma nosaukums *caName* (simbolu virkne).

Apstrāde

Sākumā tiks izsaukta metode publiskas atslēgas faila ceļa noteikšanai, kas aprakstīta 2.2.5.3. nodaļā (KP_READ_PUBLIC_NAME). Metodes kļūdas gadījumā būs atgriezts atbilstošs kļūdas paziņojums. Ja metode atgriezīs faila atrašanas vietu, tad būs atvērts publiskas atslēgas fails. Ja fails būs veiksmīgi atvērts, sāksies faila satura lasīšana. Lasīšana notiek tāda veidā, lai atbilde saglabātos viena rindā jeb viena simbolu virknē. Metodes beigās publiskas atslēgas fails ir jāaizver.

Izvade

Ja fails ir veiksmīgi izlasīts, python adapterim būs atgriezts faila saturs viena rindā (simbolu virkne Base64 kodējumā), kuru pēc tam tiešā veidā var izmantot SSH konfigurācijai.

Paziņojumi

Ja nevar atvērt publiskas atslēgas failu, tad būs atgriezts paziņojums G-006 ar faila ceļa nosaukumu.

2.2.5.6. Atslēgu pāra derīguma termiņa lasīšana

Identifikators

KP_EXPIRATION_DATE

Ievads

Metode noteikta atslēgu pāra derīguma termiņu noteikšanai.

Ievade

Python adapteris, pateicoties iepriekš izveidotam objektam, kas aprakstīts 2.2.5.1. nodaļā (KP_SINGLE_CERT), ietvers sevī šīs metodes ievades datus:

1. izveidota atslēgu pāra domēnu vārdu kopa (simbolu virknes);
2. izveidota atslēgu pāra IP adrešu kopa (simbolu virknes);
3. izveidota atslēgu pāra parakstošā uzņēmuma nosaukums *caName* (simbolu virkne).

Apstrāde

Paša sākumā tiks izsaukta metode, lai uzzinātu publiskas atslēgas faila ceļu, kas aprakstīts 2.2.5.3. nodaļā (KP_READ_PUBLIC_NAME). Metodes kļūdas gadījumā būs atgriezts atbilstošs kļūdas paziņojums. Faila ceļa uzzināšanas gadījumā būs atvērts publiskās atslēgas fails. Tālāk notiks faila derīguma termiņa lasīšana. Metodes beigās publiskas atslēgas fails ir jāaizver. Tā kā publiska un privāta atslēga bija izveidota vienlaicīgi (KP_SINGE_CERT vai KP_DIFFERENT_CERT), pietiks izlasīt tikai publiskas atslēgas failu.

Izvade

Ja faila derīguma termiņš ir veiksmīgi izlasīts, tad metode atgriež datumu formātā “yyyy-mm-ddThh:mm:ss”.

Paziņojumi

1. Ja nevar atvērt publiskas atslēgas failu, tad būs atgriezts paziņojums G-006 ar faila ceļa nosaukumu;
2. ja varēs atvērt, bet nevarēs izlasīt publiskas atslēgas failu, būs atgriezts G-007 paziņojums.

2.2.5.7. Atslēgu pāra atjaunošana

Identifikators

KP_UPDATE_CERT

Ievads

Metode privātas un publiskas atslēgas atjaunošanai.

Ievade

Python adapteris, pateicoties iepriekš izveidotam objektam, kas aprakstīts 2.2.5.1. nodaļā (KP_SINGLE_CERT), ietvers sevī šīs metodes ievades datus:

1. izveidota atslēgu pāra domēnu vārdu kopa (simbolu virknes);
2. izveidota atslēgu pāra IP adrešu kopa (simbolu virknes);
3. izveidota atslēgu pāra parakstošā uzņēmuma nosaukums *caName* (simbolu virkne).

Apstrāde

Metode saņemto ievades datus pārsūtīs kopīga SSL atslēgu pāra izveidošanas metodei, kas aprakstīta 2.2.5.1. nodaļā (KP_SINGLE_CERT). Kā bija minēts tajā nodaļā, ja saņemtais domēna vārds vai IP adrese jau eksistē *caName* uzņēmumā, tad vecais atslēgu pāris būs aizstāts ar jauno pie nosacījuma, ka tas būs veiksmīgi izveidots.

Izvade

Ja atslēgu pāris ir atjaunots, tad būs atgriezts objekts ar atjaunota atslēgu pāra atribūtiem: domēna vārdi, IP adreses un *caName*. Objektam jābūt tādam, lai varētu izsaukts 2.2.5.3 – 2.2.5.7. nodaļās monētās metodes. Gadījumā ja KP_SINGLE_CERT metodē radīsies kļūda, tad tiks atgriezts atbilstošs kļūdas paziņojums.

Paziņojumi

Gadījumā ja KP_SINGLE_CERT metodē radīsies kļūda, tiks atgriezts atbilstošs kļūdas paziņojums.

2.2.5.8. Atslēgu pāra domēna vārdu un IP adrešu lasīšana

Identifikators

KP_DEFINE_CONTENT

Ievads

Domēna vārdu un IP adrešu definēšana, kas pieder tieši dotajam atslēgu pārim.

Ievade

No python adaptera būs saņemts ceļš, kur glabājas noradīts atslēgu pāris (simbolu virkne).

Apstrāde

Sākumā notiek pārbaude, vai eksistē atslēgu pāris, kas atrodas pēc noradīta ceļa. Ja dati ir korekti, tad notiek publiskas atslēgas lasīšana jeb tam piederošo domēna vārdu un IP adrešu definēšana.

Izvade

Ja noradītajā ceļā eksistē atslēgu pāris un fails ir veiksmīgi izlasīts, tad būs atgriezts saraksts ar visiem domēna vārdiem un IP adresēm, kas pieder dotajam pārim. Vispirms būs izvietoti visi domēna vārdi, bet pēc tiem sekos IP adreses.

Paziņojumi

Ja ievadē noradītajā ceļā nav publiskas atslēgas faila vai to nav iespējams izlasīt, tad būs attēlots paziņojums I-006 ar tekstu, kas sakrīt ar ceļu.

2.2.5.9. Atslēgu pāra izdevēja lasīšana

Identifikators

KP_DEFINE_ISSUER

Ievads

Metode paredzēta SSL sertifikāta atslēgu pāra izdevējiestādes noteikšanai.

Ievade

No python adaptera būs saņemts ceļš, kur glabājas noradīts atslēgu pāris (simbolu virkne).

Apstrāde

Sākumā notiek pārbaude, vai eksistē atslēgu pāris, kas atrodas pēc noradīta ceļa. Ja ceļš ir korekts, tad notiek publiskas atslēgas lasīšana. Lasot failu, būs noteikts tā izdevējs. Tā kā gan publiskas, gan privātas atslēgas fails bija izveidots vienlaicīgi (KP_SINGLE_CERT, KP_DIFFERENT_CERT), nav lielas starpības kādu failu lasīt. Tika izvēlēts publiskas atslēgas fails, lai izvairītos no privātas informācijas lasīšanas kļūdām.

Izvade

Ja noradītajā ceļā eksistē atslēgu pāris un fails ir veiksmīgi izlasīts, tad būs atgriezts atslēgu pāra izdevējs (simbolu virkne). Citādi, tiks atgriezts kļūdas paziņojums.

Paziņojumi

Ja ievadē noradītajā ceļā nav publiskas atslēgas faila vai to nav iespējams izlasīt, tad būs attēlots paziņojums I-006 ar tekstu, kas sakrīt ar ceļu.

2.2.6. Mapju šifrēšanas modulis

2.2.6.1. Mapes izveidošana turpmākai šifrēšanai

Identifikators

DE_INIT_MOUNT

Ievads

Jaunas mapes izveidošana, kas paredzēta datu šifrēšanai 2.2.6.2. nodaļā (DE_UNMOUNT) un datu saglabāšana pamatmapē turpmākai atšifrēšanai.

Ievade

Tiks saņemti ievades dati no python adaptera:

1. pamatmapes (BASE) ceļš, kur glabāsies šifrēti dati (simbolu virkne);
2. atšifrētas (MOUNT) mapes ceļš, dati būs dzēsti pēc DE_UNMOUNT (simbolu virkne);
3. parole, lai aizsargātu šifrēto mapi no nesankcionētas piekļuves (simbolu virkne).

Apstrāde

Mapju šifrēšana sastāv no divām mapēm. Pamatmapē glabāsies šifrēti dati, kas atjaunosies reāla laikā pēc katras izmaiņas atšifrētajā mapē. Sākumā, ja mapju nav, būs izveidota gan pamatmape, gan atšifrētā mape. Tālāk pirms pirmās šifrēšanas notiks pārbaude, vai dotas mapes ir tukšas un nav jau šifrētas. Notiks pārbaude, vai mape (MOUNT) neatrodas pamatmapē jeb nav pamatmapes apakšmape. Ja parole nav tukša, ar tas palīdzību tiks izveidota saite starp pamatmapi un šifrēto mapi. Ja nosacījumi ir izpildīti, lietotājs saņems izvadi. Pēc metodes darba pabeigšanas lietotājs var strādāt mapē (MOUNT), pievienot, izmainīt, dzēst failus. Kad darbs būs pabeigts, lietotājs izsauks DE_UNMOUNT, lai pabeigtu datu šifrēšanu.

Izvade

Ja pamatmape un mape (MOUNT) izveidota veiksmīgi, tad būs atgriezta *True* vērtība. Citādi, tiks atgriezts kļūdas paziņojums.

Paziņojumi

1. Ja sākumā kaut kāda mape nebūs tukša, būs atgriezts G-008 ar mapes atrašanas vietu;
2. ja sākumā kaut kāda mape ir jau šifrēta, būs atgriezts G-009 ar mapes atrašanas vietu;
3. ja parole šifrēšanai nesatur nevienu simbolu, būs atgriezts paziņojums G-010;
4. ja atšifrēta mape atrodas pamatmapē, tad būs atgriezts paziņojums G-011;
5. ja šifrēšanas laikā notiks kļūda, tad atgriež G-012 ar python kļūdas paziņojumu.

2.2.6.2. Mapes šifrēšana

Identifikators

DE_UNMOUNT

Ievads

Atšifrētas mapes (MOUNT), datu šifrēšana un atšifrētas mapes datu slēpšana.

Ievade

Tiks saņemti ievades dati no python adaptera:

1. pamatmapes (BASE) ceļš, kur glabājas jau šifrēti dati pēc DE_INIT_MOUNT vai DE_MOUNT metodēm (simbolu virkne);
2. atšifrētas (MOUNT) mapes ceļš, no kurienes dati būs dzēsti (simbolu virkne);

Apstrāde

Mapju šifrēšana ietver sevī divas mapes. Pamatmapē tagad glabājas šifrēti dati, bet otrajā mape pagaidām satur to pašu informāciju, bet tikai nešifrēta veidā. Notiek atšifrētas (MOUNT) mapes šifrēšana. Šifrēšanas laikā visi dati no tās mapes (MOUNT) pazudīs un saglabāsies tikai šifrēti lietotāja dati pamatmapē (BASE). Pēc šifrēšanas pabeigšanas notiks pārbaude vai atšifrētā mape (MOUNT) ir tukša. Ja mape ir tukša, tad šifrēšana notika veiksmīgi. Tagad nevarēs lasīt datus no šifrētas mapes (MOUNT), jo dati pazuda. Dati pamatmapē ir šifrēta veidā un tos varēs atšifrēt tikai ar paroli, kas ir aprakstīts 2.2.6.3. nodaļā (DE_MOUNT).

Izvade

Ja šifrēšana pabeigta veiksmīgi, tad tiks atgriezta *boolen True* vērtība. Citādi, tiks atgriezts kļūdas paziņojums.

Paziņojumi

1. ja šifrēšanas procesa laikā notiks kļūda, tad būs paziņots par G-013 ar attiecīgu kļūdas tekstu un šifrētas mapes ceļu (MountDirectory);
2. ja atšifrēta mape nebūs tukša procesa beigās tad tiks attēlots paziņojums G-014 ar atšifrētas mapes ceļu (MountDirectory);
3. ja šifrēšanas process pabeidzas bez kļūdām un mape būs tukša, bet tā paliks atšifrēta, tad lietotājs tiks informēts ar G-015 ar šifrētas mapes atrašanas vietu.

2.2.6.3. Mapes atšifrēšana

Identifikators

DE_MOUNT

Ievads

Atšifrēto datu ievietošana atšifrētājā mapē (MOUNT).

Ievade

Tiks saņemti ievades dati no python adaptera:

1. pamatmapes (BASE) ceļš, kur glabājas jau šifrēti dati (simbolu virkne);
2. atšifrētas (MOUNT) mapes ceļš, kur dati būs ievietoti atšifrēta veidā (simbolu virkne);
3. parole, lai pamatmapes šifrēto saturu atšifrētu otrajā mapē (simbolu virkne).

Apstrāde

Mapju šifrēšana ietver sevī divas mapes. Pamatmapē tagad glabājas šifrēti dati, bet otrajai mapei jābūt tukšai. Sākumā notiek pārbaude vai pamatmape eksistē norādītajā ceļā un tajā iekšā ir šifrēšanas konfigurācijas fails, kur glabājas informācija par šifrētiem datiem. Tālāk pārbauda, vai šifrēta mape (MOUNT) eksistē, tā ir tukša, neatrodas pamatmapē un nav jau atšifrēta iepriekš. Pēc tam, ja parole nav tukša, ar tas palīdzību notiks mapes atšifrēšana. Šifrēti dati no pamatmapes (BASE) būs atšifrēti, jo iepriekš (DE_INIT_MOUNT) tika izveidota saite starp divām mapēm. Ja mape būs atšifrēta, tad lietotājs saņems attiecīgu izvadi un tagad var atkal strādāt mapē. Pēc lietotāja darba, var izsaukt DE_UNMOUNT, lai atkal paslēptu datus.

Izvade

Ja šifrēšana pabeigta veiksmīgi, tad tiks atgriezta *boolen True* vērtība.

Paziņojumi

1. Ja kāda mape nebūs atrasta, tad būs atgriezts G-016 ar mapes plānoto atrašanas vietu;
2. ja pamatmapē nav konfigurācijas faila, tad atgriež G-017 ar mapes atrašanas vietu;
3. ja mape datu atšifrēšanai nav tukša, tad tiks atgriezts G-018 ar mapes atrašanas vietu;
4. ja mape datu atšifrēšanai jau ir atšifrēta, tad atgriež G-019 ar mapes atrašanas vietu;
5. ja parole atšifrēšanai nesatur nevienu simbolu, būs atgriezts paziņojums G-010;
6. ja mape atšifrēšanai atrodas pamatmapē, tad būs atgriezts paziņojums G-011;
7. ja šifrēšanas procesa laikā notiks kļūda, tad būs paziņots par G-012 ar tekstu, kas sakrīt ar python interpretatora kļūdas paziņojumu.

2.3. Nefunkcionālās prasības

2.3.1. Veiktspējas prasības

1. Virtuāla datu nesēja izveidošana (DE_INIT_MOUNT un DE_MOUNT) aizņem ne vairāk kā 1 minūti, ja tas darīts ar procesoru ar vismaz 2GHz frekvenci un pie nosacījuma, ka nav citu resursietilpīgo procesu un ir pieejami vismaz 4GB brīvpiekļuves atmiņas (RAM);
2. IDL faila lasīšana ar ne vairāk kā 1000 rindiņām jāveic 1 sekundes laikā, ja tas darīts ar procesoru ar vismaz 2GHz frekvenci.

2.3.2. Izmantojamības prasības

1. Visiem kļūdas paziņojumiem un metodes rezultātiem jābūt pieejamiem angļu valodā;
2. metodes izsaukuma datiem, kas tiks pārsūtīti starp webAppOS un Python moduli jābūt UTF-8 kodējumā.

2.4. Papildinājumi

Kad projekts jau bija uzsākts, no pasūtītāja puses atnāca papildus funkcionālā prasība: iespēja ģenerēt kodu, kas ļaus WebAppOS lietotņu izstrādātājiem izmantot Python sintaksi, lai piekļūtu datiem tīmekļa atmiņā un ar WebAppOS līdzekļiem izsauktu funkcijas, kas uzrakstītas citās programmēšanas valodās.

Koda ģeneratora ievaddati

Fails C++/Java-sintaksē, kas satur klases, to atribūtus un metodes. Koda ģeneratora projektējumā ir nepieciešams formalizēt faila formātu.

Koda ģeneratora izvaddati:

Ģenerētajam kodam jānodrošina iespēja tikt klāt atribūtu vērtībām, kas glabājas tīmekļa atmiņā, kā arī jāspēj izsaukt funkcijas, kas uzrakstītas citās programmēšanas valodās un kuras izpildīs citi procesori.

Piezīmes

Ir atļauts ģenerēt palīgkodu (piemēram, objektu *factory* kodu). Koda ģeneratora projektējumā ir nepieciešams formalizēt ģenerētā koda struktūru.

3. PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS

3.1. Izmantotā pieeja

Drošības funkcionalitātes nodrošināšanai tika izvēlēta python3 programmēšanas valoda, kas ietver sevī vairākus šifrēšanas rīkus, piemēram, SSL un crypto bibliotēkas. Projektētas sistēmas integrēšana tīmekļa operētājsistēmā webAppOS tiks realizēta python valodā, kas dos iespēju procesoram vieglāk komunicēt ar adapteri, kas izsauks nepieciešamas metodes dinamiski, iepriekš nezinot to nosaukumu. WebAppOS neuzliek papildus ierobežojumus uz tehnisko python funkciju realizāciju. Tomēr darba ietvaros tiks izmantota objektorientēta programmēšana, kas savukārt ļaus sadalīt attiecīga moduļa funkcijas klasēs, kas veiks noteiktas funkcijas. Tāda pieeja ļaus programmētājiem ātrāk saskatīt kļūdas un pēc nepieciešamības pievienot jaunas klases metodes, kā arī saskatīt to saistību ar jau eksistējošām metodēm. Katrai adaptera izsauktai klases metodei ir jāatgriež gala lietotāja pieprasīti dati vai nu norāde uz klases objektu, vai nu vārdnīca ar atslēgu ‘error’ un vērtību, kas sakrīt ar attiecīgu kļūdas paziņojumu. Visus ievades datus no tīmekļa operētājsistēmas puses tīmekļa procesors saņems JSON formātā.

3.2. Tīmekļa procesors

Tīmekļa procesora un webAppOS komunikācijas sakaru projektējumu var apskatīt avotā [7]. Šajā nodaļā būs pievērsta uzmanība tieši python moduļa integrācijai un tas saistībai ar projektētiem drošības funkcionalitātes moduļiem. Python moduļa komunikācijai ar galveno sistēmu tiks izmantota standarta ieejas/izejas datu plūsma (stdin/stdout). Katrai komandai procesora un webAppOS komunikācijā jāsastāv no divām daļām, kur katra sākas ar jaunu rindu: komandas nosaukums un serializēts jeb teksta formāta JSON objekts. Lai dokumentācijā ērtāk lasītu ievades un izvades komandas, vienas rindas turpinājumu apzīmēsim ar ‘\’ simbolu.

Paša sākumā, pirms reģistrēta lietotāja darba sākuma, webAppOS palaidīs python procesoru, kas arī ir redzams 3.1. attēlā.

```
python3 /PATH_to_WEBAPPOS/usr/lib/localpython/Python_Processor.py "NAME"
```

3.1. att. **Python procesora palaišana**

- PATH_to_WEBAPPOS (simbolu virkne)– mapes atrašanas vieta (servera pusē), kur atrodas webAppOS;
- NAME (simbolu virkne)– tīmekļa procesora nosaukums, kas būs nepieciešams, ja turpmāk būs palaisti vairāki python procesori.

Kad python modulis būs palaists (3.1. att.), tīmekļa procesors informēs par veiksmīgu darba sākšanu, kas ir redzams 3.2. attēlā.

```
READY
{"instructionSets": ["python3"], \
"codeProtocols": ["local"], \
"webMemoryProtocols": ["pipe"]}
```

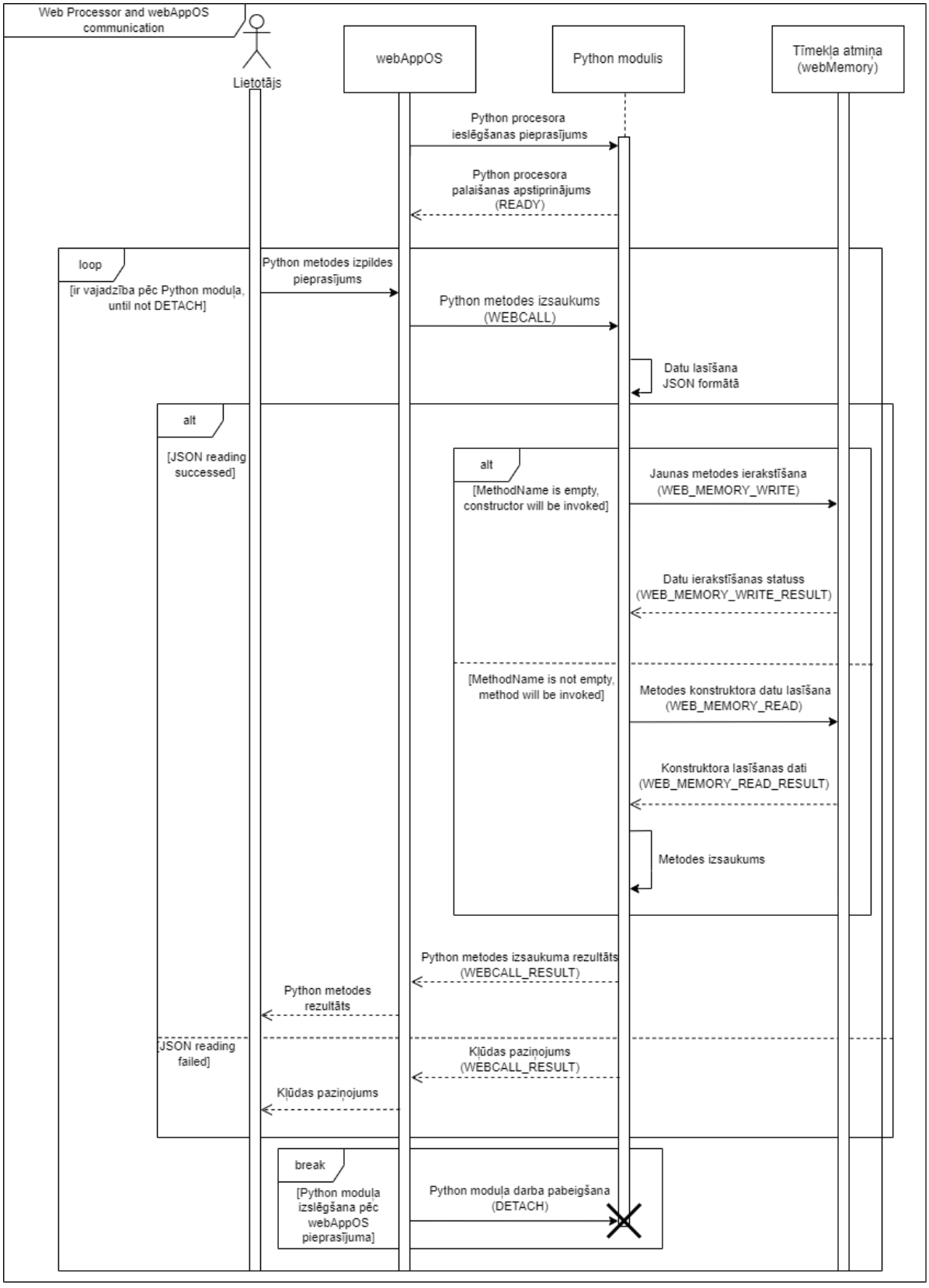
3.2. att. **Python procesora palaišanas apstiprinājums**

- READY – python procesora palaišanas apstiprinājuma komanda. Tīmekļa procesors gatavs turpmākajam darbam;
- instructionSets – programmēšanas valodas (python3) nosaukums;
- codeProtocols – datu apmaiņas protokols;
- webMemoryProtocols – "pipe" vai tukša kopa nozīmē to, ka datu apmaiņa starp tīmekļa atmiņu un webAppOS notiks tikai ar standarta ieejas/izejas plūsmu.

3.2.1. Python metodes izsaukums (WEBCALL)

Kāda veidā notiek komunikācija starp tīmekļa procesoru un webAppOS var apskatīt 3.3. attēlā. Secību diagramma ietver sevī reģistrēto lietotāju, webAppOS, python moduli un tīmekļa atmiņu. Paša sākumā, pēc veiksmīgas python moduļa palaišanas, reģistrēts lietotājs aizsūta savu metodes izpildes pieprasījumu. WebAppOS analizē doto pieprasījumu un, ja pieprasīta metode atrodas Python modulī, tad Python modulim būs aizsūtīti metodes izsaukuma dati (WEBCALL). Tālāk tīmekļa procesors apstrādās ievades datus (sk. 3.2.1.1. un 3.2.1.2. nodaļas). Ja ievades dati atbilst JSON formātam, tad Python modulis izgūs aizsūtītas metodes nosaukumu. Ja metode nav ievadīta, tas nozīmēs, ka dati par šo klases metodi pagaidām neglabājas tīmekļa atmiņā, līdz ar to notiks aizsūtīto datu ierakstīšana tīmekļa atmiņā, kas aprakstīts 3.2.2. nodaļā. Ja metode nav tukša, tad pirms norādītas metodes palaišanas Python modulī notiks konstruktora ievades datu lasīšana no tīmekļa atmiņas, kas detalizēti aprakstīta 3.2.3. nodaļā. Tagad, kad tīmekļa procesoram būs gan klases metodes dati no webAppOS, gan klases atribūti no tīmekļa atmiņas, būs iespējams nodot šos datus adapterim, lai tas izsauktu metodi Python modulī. Pēc rezultāta saņemšanas tīmekļa procesors aizsūtīs šos datu atpakaļ webAppOS (WEBCALL_RESULT). Ja pieprasījuma apstrādes laikā notiks kļūda, tad arī gala lietotājs saņems atbilstošo 2.2.1. nodaļai paziņojumu standarta ieejas/izejas plūsmā. Kad

webAppOS vajadzēs izslēgt Python moduli izslēgšanas, tā nosūtīts atbilstošos paziņojumus (DETACH).



3.3. att. **Python procesora un webAppOS komunikācijas secību**

3.2.1.1. Jaunas metodes izsaukums (WEBCALL)

Tiks aprakstīts metodes WP_WEBCALL_NEW projektējums. Kā redzams 3.3. attēlā webAppOS aizsūta python modulim metodes izsaukuma datus (WEBCALL). Ja tīmekļa atmiņā pagaidām nav informācijas par lietotāja pieprasītu metodi, tad webAppOS aizsūtīs tīmekļa procesoram informāciju, kas redzama 3.4. attēlā.

```
WEBCALL
{"webMemoryUrl": "1-2-3-4-5-6", \
"objectReference": 7890, \
"className": "CLASS_NAME", \
"methodName": "", \
"methodUrl": \ "python3:local:FILE_NAME.py#CLASS_NAME ", \
"arguments": "{ARGUMENTS}", \
"context": {"searchPaths": "DIR/FILE_NAME.py"}}
```

3.4. att. **Python jaunas metodes izsaukums**

- webMemoryUrl (simbolu virkne)– adrese, kur glabājas reģistrēta lietotāja dati tīmekļa atmiņā;
- objectReference (vesels pozitīvs skaitlis) – norāde uz atmiņas daļu, kur glabāsies informācija par jaunu klasi CLASS_NAME;
- className (simbolu virkne) – jaunas klases CLASS_NAME nosaukums;
- methodName – metodes nosaukums. Tukšs, pēc noklusējuma tas nozīmē, ka jāsaglabā klases atribūti tīmekļa atmiņā;
- methodUrl (simbolu virkne) – url, kas glabā faila FILE_NAME.py nosaukumu, klases un programmēšanas valodas nosaukumu;
- arguments (saraksts)– klases CLASS_NAME atribūtu kopa turpmākai konstruktora izveidošanai, ko arī apraksta 3.2.1.2. nodaļa;
- context (vārdnīca) – sastāv no atslēgas “searchPaths” ar vērtību, kur glabājas failu atrašanas vietas metodes izpildīšanai, to starpā arī FILE_NAME.py atrašanas vieta servera pusē.

3.2.1.2. Iepriekš izmantotas metodes izsaukums (WEBCALL)

Nodaļa paredzēta metodes WP_WEBCALL_OLD aprakstīšanai. Secību diagrammā 3.3. attēlā var apskatīt gadījumu, kad python metodes izsaukuma laikā (WEBCALL) metodes nosaukums nebūs tukšs. Tīmekļa procesors saņems datus, kas redzami 3.5. attēlā.

```
WEBCALL
{"webMemoryUrl": "1-2-3-4-5-6", \
"objectReference": 7890, \
"className": "CLASS_NAME", \
"methodName": "METHOD_NAME", \
"methodUrl": \ "python3:local:FILE_NAME.py#CLASS_NAME.METHOD_NAME", \
"arguments": "{ARGUMENTS}", \
"context": {"searchPaths": "DIR/FILE_NAME.py"}}
```

3.5. att. **Python iepriekš izmantotas metodes izsaukums**

- webMemoryUrl (simbolu virkne)– adrese, kur glabājas lietotāja dati tīmekļa atmiņā;
- objectReference (vesels pozitīvs skaitlis) – norāde uz atmiņu, kur jau glabājas informācija par esošo klasei CLASS_NAME instancei (objektu);
- className (simbolu virkne) – klases CLASS_NAME nosaukums;
- methodName (simbolu virkne)– metode, kuru ir jāizmanto lietotāja pieprasījuma izpildei;
- methodUrl (simbolu virkne) – url ar FILE_NAME.py faila nosaukumu, kur glabājas python klase CLASS_NAME ar metodi METHOD_NAME;
- arguments (saraksts)– metodes METHOD_NAME argumenti.
- context (vārdnīca) – sastāv no atslēgas “searchPaths” ar vērtību, kas ietver sevī faila FILE_NAME.py atrašanas vietu servera pusē.

3.2.1.3. URL adreses lasīšana

Tīmekļa procesors python faila FILE_NAME atrašanai izmantos methodURL vērtību. Vispirms notiks methodUrl ievades pārbaude, tas ir, simbolu virknei jāatbilst formātam "python3:local:FILE_NAME.py#CLASS_NAME". Lai pārbaudītu šo informāciju, būs izmantota regulāra izteiksme. Izteiksme ir sadalīta piecās daļās. Pirmajā daļā notiek pārbaude

vai norādīta programmēšanas valoda ir python3. Otrajā daļā seko protokols, tā var būt jebkura simbolu virkne (mūsu gadījumā *locali*, jo faili būs pieejami lokāli). Trešajā daļā notiek pārbaude, vai ievadītā programmēšanas valoda sakrīt ar tās paplašinājumu (mūsu gadījumā *python.py*). Ceturtajā daļā klases nosaukumam nedrīkst būt tukšam un jāsatur tikai mazie un lielie latīņu alfabēta burti, cipari vai '_' simbols. Piektā daļa var būt arī tukša, kā bija minēts 3.2.1.1. nodaļā, un aizpildīta, ko apraksta 3.2.1.2. nodaļa. Metodes nosaukums var saturēt mazos un lielos latīņu alfabēta burts, kā arī ciparus un '_' simbolu. Divi koli '**:**' un viens '**#**' simbols kalpos par atdalījuma zīmēm, lai ar regulāro izteiksmi vieglāk pārbaudītu ievades simbolu virkni.

3.2.1.4. Python metodes izsaukuma rezultāta aizsūtīšana (WEBCALL_RESULT)

Pēc veiksmīga python metodes izsaukuma apstrādes (WEBCALL) tīmekļa procesors aizsūta webAppOS atbildi, kas ir redzama 3.6. attēlā. Atbilde sastāv no JSON tipa objekta ar vienu atslēgu "result" un vērtību, kas sakrīt ar python metodes rezultātu.

```
WEBCALL_RESULT
{"result": "RESULT"}
```

3.6. att. **Python metodes veiksmīga izsaukuma rezultāts**

- result (simbolu virkne) – izpildītas metodes rezultāts.

Ja metodes izpildīšanas laikā notiks kaut kāda kļūda, tad tīmekļa procesors aizsutīs webAppOS JSON objektu ar vienu atslēgu "error" un atbilstošo paziņojumu.

```
WEBCALL_RESULT
{"error": "RESULT"}
```

3.7. att. **Python metodes neveiksmīga izsaukuma rezultāts**

- error (simbolu virkne) – izpildītas metodes rezultāts jeb kļūda, kas parādījās apstrādes laikā.

3.2.2. Datu ierakstīšana tīmekļa atmiņā (WEB_MEMORY_WRITE)

WP_WB_WRITE metode ir paredzēta, lai mainītu esošos datus, vai arī – lai ierakstītu jaunus datus tīmekļa atmiņā. Tīmekļa procesors aizsūtīs webAppOS informāciju, kuru var aplūkot 3.8. attēlā.

```
WEB_MEMORY_WRITE
{"webMemoryUrl": “1-2-3-4-5-6”, \
 "objectReference": 56980, \
 "className": CLASS_NAME, \
"state": { \
“KEY”: VALUE; … } }
```

3.8. att. **Datu ierakstīšana tīmekļa atmiņā**

- webMemoryUrl (simbolu virkne)– adrese, kur glabājas reģistrēta lietotāja dati tīmekļa atmiņā;
- objectReference (vesels pozitīvs skaitlis, **tikai izveidotiem objektiem**) – norāde uz atmiņas daļu, kur glabājas informācija par klasi, kuru dati būs rediģēti;
- className (simbolu virkne, **tikai jauniem objektiem**) – jaunas klases nosaukums;
- state (vārdnīca) – sastāv no objekta atribūtiem, kas atbilst atslēgām (KEY). Katrai atslēgai atbilst vērtība (VALUE). Ja vērtība ir null, tad informācija par atribūtu būs dzēsta. Būs atjaunoti tikai tie atribūti, kas parādās vārdnīcā. Daudzpunktes (**…**) vietā var būt vēl citas atslēgas ar vērtībām.

Pēc datu ierakstīšanas tīmekļa atmiņā, procesors saņems norādīto 3.9. attēlā informāciju

```
WEB_MEMORY_WRITE_RESULT
{"result": RESULT}
```

3.9. att. **Veiksmīgas datu ierakstīšanas statuss**

- result (vesels pozitīvs skaitlis) – norāde objectReference, sakrīt ar 3.8. attēlā. redzamo.

Ja notiks kļūda ierakstīšanas laikā, tad tīmekļa atmiņa aizsūtīs informāciju (3.10. att.)

```
WEB_MEMORY_WRITE_RESULT
{"error": RESULT}
```

3.10. att. **Neveiksmīgas datu ierakstīšanas statuss**

- error (simbolu virkne) – datu ierakstīšanas kļūda tīmekļa atmiņā

3.2.3. Datu lasīšana no tīmekļa atmiņas (WEB_MEMORY_READ)

Lai izlasītu klases atribūtu vērtības no tīmekļa atmiņas būs izmantota WP_WB_READ metode**.** Tīmekļa procesors aizsūtīs informāciju, kas redzama 3.11. attēlā.

```
WEB_MEMORY_READ
{"webMemoryUrl": “1-2-3-4-5-6”, \
"objectReference": 56980 \
}
```

3.11. att. **Datu lasīšana no tīmekļa atmiņas**

- webMemoryUrl (simbolu virkne)– adrese, kur glabājas reģistrēta lietotāja dati tīmekļa atmiņā;
- objectReference (vesels pozitīvs skaitlis) – norāde uz atmiņas daļu, kur glabājas informācija par klasi.

Pēc veiksmīgas datu izlasīšanas, tīmekļa atmiņa aizsūtīs informāciju, kuru var apskatīt 3.12. attēlā.

```
WEB_MEMORY_READ_RESULT
{"result": RESULT }
```

3.12. att. **Veiksmīgas datu lasīšanas rezultāts**

- result (vārdnīca) – sastāv no objekta atribūtiem, kas katrai vārdnīcas atslēgai atbilst atribūta vērtība.

Ja datu lasīšanas laikā notiks kļūda, tad tīmekļa atmiņa aizsūtīt tīmekļa procesoram kļūdas paziņojumu, kā redzams 3.13. attēlā.

```
WEB_MEMORY_READ_RESULT
{"error": RESULT }
```

3.13. att. **Neveiksmīgas datu lasīšanas rezultāts**

- error (simbolu virkne) – ietver sevī atbilstošo kļūdas paziņojumu (RESULT).

3.2.4. Datu ierakstīšanas/lasīšanas un ārējas funkcijas izsaukuma automatizēšana

Var gadīties, ka python moduļa darbības laikā būs nepieciešams izsaukt funkciju, kas uzrakstīta kaut kāda cita programmēšanas valodā, piemēram, Java, vai arī būs nepieciešams izlasīt vai ierakstīt atribūtu tīmekļa atmiņā vairākas reizēs. Tāda gadījumā, lai neapgrūtinātu programmētāja kodu ar vienveidīgam konstrukcijām, tiks izmantota koda ģenerēšana.

3.2.4.1. Koda ģenerēšana

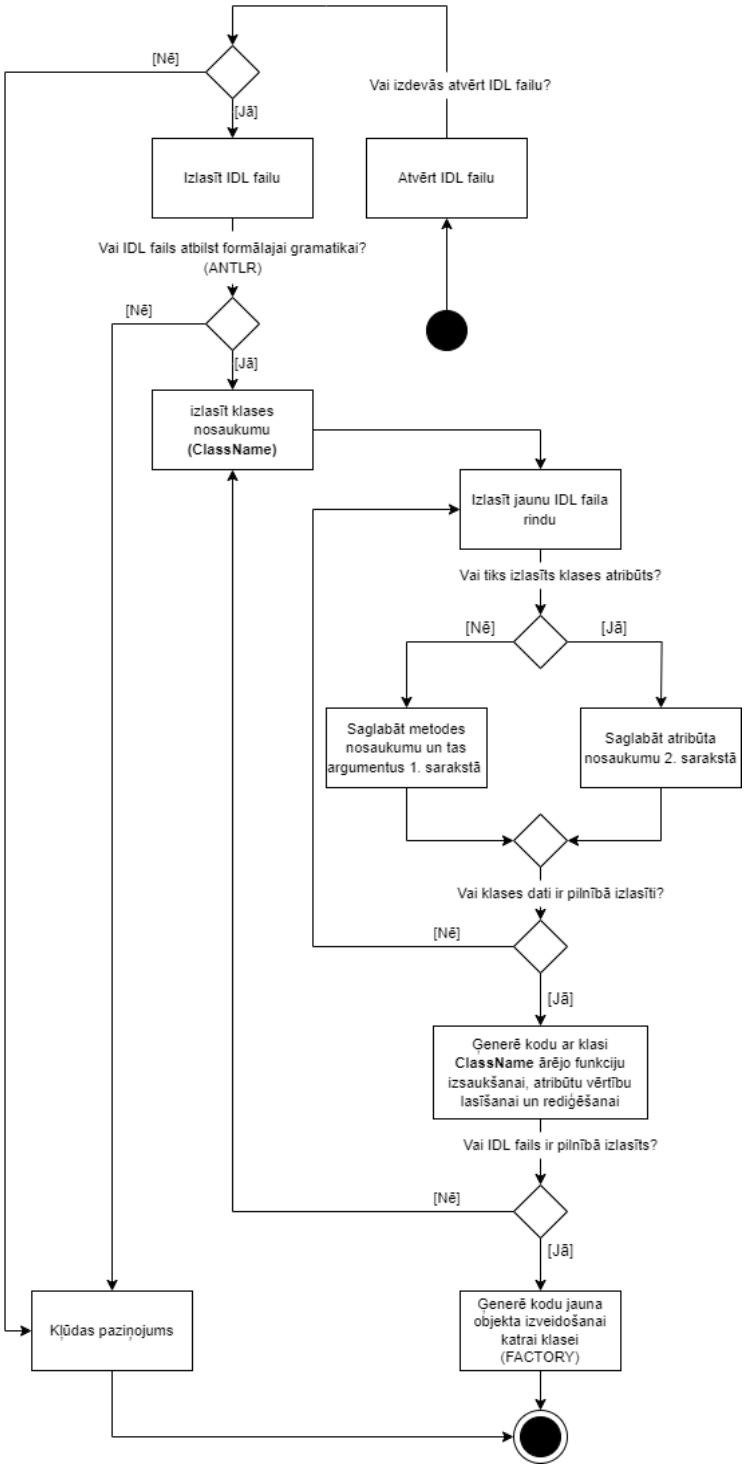
Lai uzģenerētu kodu, kas izsauks ārējas funkcijas un lasīs/ierakstīs atribūtus tīmekļa atmiņā, ir nepieciešams ieejas fails, kas aprakstītu kādi atribūti un metodes pieder noteiktai klasei. Ievades failam ir izvēlēts IDL formāts, Piemēru IDL failam var apskatīt 3.14. attēlā

```
Class Greeting {
   String text;
   Void say_hello_to_C(String name);
   String say_hello_to_java();
}
Class Java {
   String get_version();
}
```

3.14. att. **IDL faila piemērs**

1. Klase vienmēr sākas ar vārdu “Class” un pēc tas seko klases nosaukums;
2. informācija par klases atribūtiem un metodēm ir ierakstīta “**{ }**” figūriekavās;
3. atribūtu nosaukumi sākas ar datu tipu, kas var būt vienāds ar “Integer”, “String”, “Boolean”, “Real”;
4. pēc atribūta datu tipa vienmēr seko atribūta nosaukums (simbolu virkne);
5. metodes nosaukumi sākas ar datu tipu, kas var būt vienāds ar “Integer”, “String”, “Boolean”, “Real” vai “Void”;
6. pēc metodes datu tipa seko metodes nosaukums aiz kuras seko apaļas iekavas ‘**( )**’;
7. metode vai nu nesaturēt argumentus, tad iekavas ‘**( )**’ paliks tukšas, vai nu satur argumentus, kas atdalīti sava starpā ar komatiem;
8. metodes argumenti sākas ar datu tipu “Integer”, “String”, “Boolean”, “Real” pēc kura seko metodes argumenta nosaukums;
9. pēc metodes argumenta datu tipa seko argumenta nosaukums (simbolu virkne);
10. pēc atribūta vai metodes seko semikols ‘**;**’ ;
11. IDL failā var būt vairākas klases, vairāki tukšumi un jauno rindu simboli.

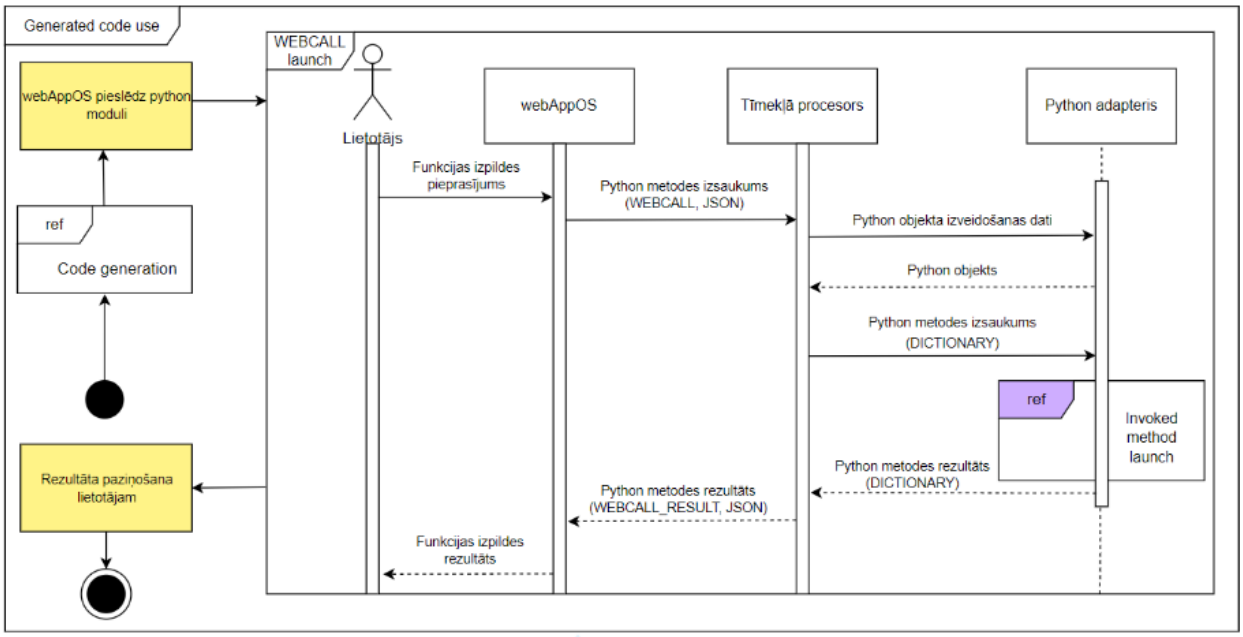
IDL ieejas faila pārbaudei tiks izmantots formālo valodu ģenerators ANTLR. Vispirms būs uzrakstīta formālā gramatika parsētāja būvei, kas izlasīs pa daļām doto IDL. Tāda veidā, kad katrai klasei būs izlasīti tas atribūti un metodes, būs iespējams uzģenerēt kodu. Koda ģenerēšanas soļus, kas palīdzēs izsaukt ārējas klases funkcijas (INNER_WEBCALL) un lasīt (WEB_MEMORY_READ) / ierakstīt (WEB_MEMORY_WRITE) klases atribūtus var apskatīt 3.15. attēlā.



3.15. att. **Koda ģenerēšanas soļi – UML aktivitāšu diagramma**

3.2.4.2. Uzģenerēta koda izmantošanas ceļvedis

Attēlā 3.16. var apskatīt datu apmaiņu starp webAppOS un python moduli, kā arī momentu, kad tiks uzģenerēts un lietots kods darba automatizēšanai. Pašā sākumā tiks uzģenerēts kods (Code generation), kas ir redzams UML aktivitāšu diagrammā 3.15. attēlā. Pēc python moduļa pieslēgšanas reģistrēts lietotājs var sūtīt funkcijas izpildes pieprasījumu webAppOS, kas ir apzīmēts secību diagrammā. Kad tīmekļa procesors saņems python metodes izsaukuma pieprasījumu, tas virzīs datus adapterim, lai izveidotu jaunu klases objektu. Turpmāk ar saņemta objekta palīdzību procesors varēs aizsūtīt python metodes izsaukuma pieprasījumu adapterim. Python adapteris izpilda saņemtu metodi ("Invoked method launch" 3.16. attēlā).

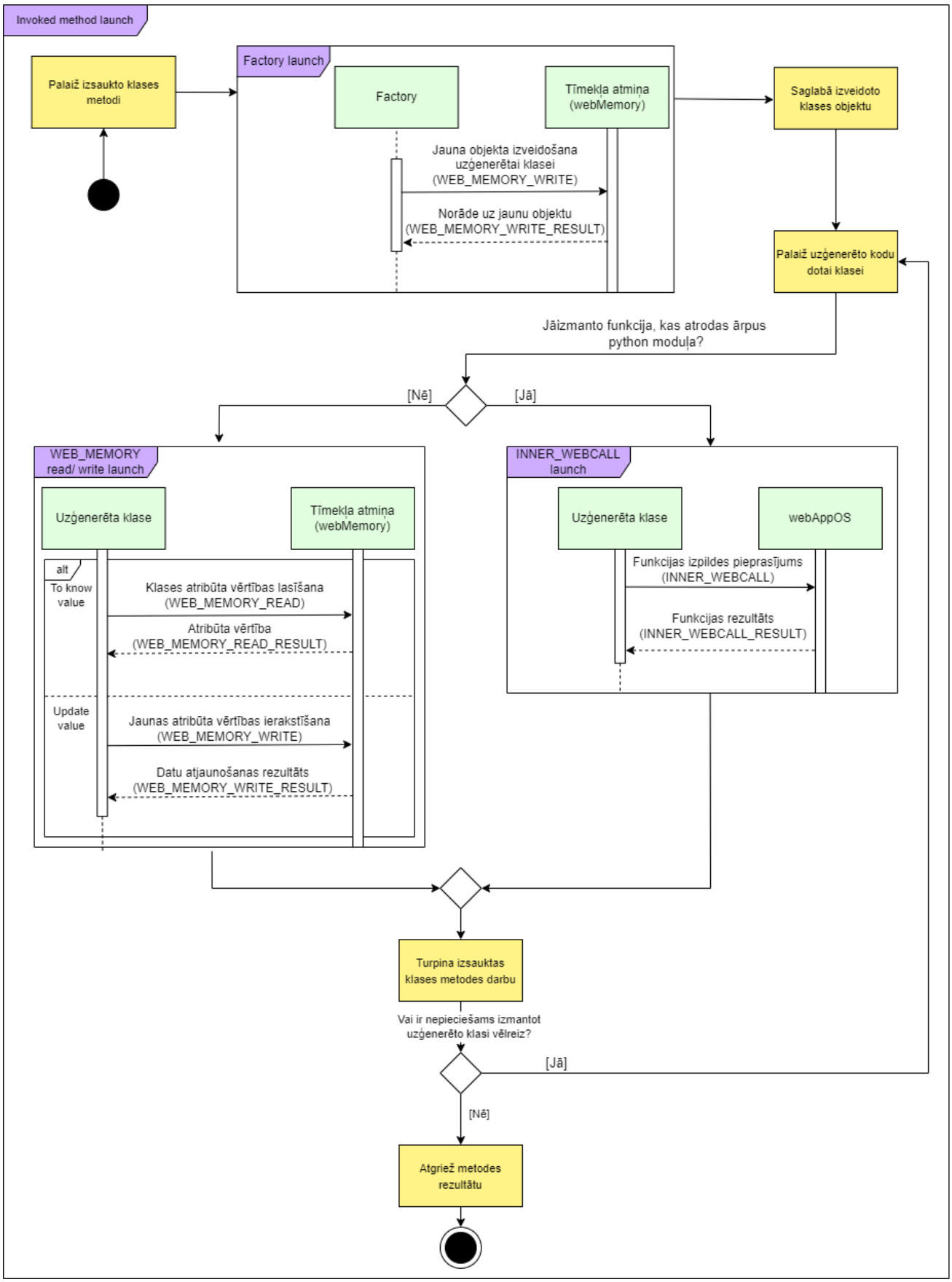


3.16. att. **Uzģenerēta koda izmantošana – UML jauktā diagramma**

Metodes izpildīšanu (Invoked method launch) var apskatīt 3.17. attēlā, kur redzams, ka tika izsaukta metode **MethodName**, kas atrodas klasē **ClassName**.

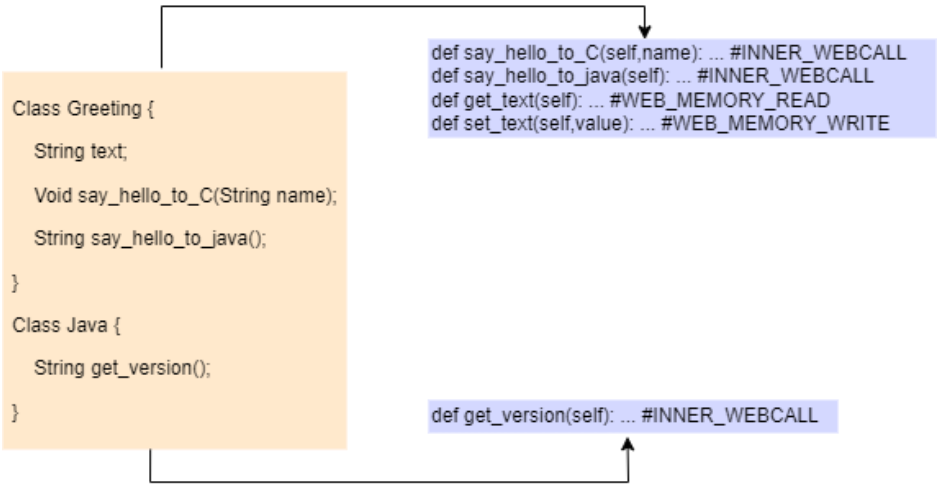
Paša sākumā notiek uzģenerētas klases **Factory** metodes **newClassName** palaišana. **Factory** pārsūtīs sekojošo informāciju (sk. 3.2.2. nodaļā): adresi, kur glabājas lietotāja dati (webMemoryUrl), kuru saņēma adapteris no tīmekļa procesora; izpildāmas klases nosaukumu (**ClassName**); izpildāmas klases atribūtus (state). Ja dati ir ierakstīti veiksmīgi, tad tīmekļa atmiņa aizsūtīs norādi objectReference, kur glabājas dati par klasi **ClassName**.

Beigās uzģenerētais kods **Factory** izveido objektu ar atribūtiem – webMemoryUrl un objectReference. Šo objektu atgriež metodei **MethodName**, kas norāda uz uzģenerēto kodu datu ierakstīšanas/lasīšanas un ārējas funkcijas izsaukuma automatizēšanai.



3.17. att. **Metodes, kas izmanto uzģenerēto kodu, izpildīšana – UML jauktā diagramma**

Attēlā 3.18. var vēlreiz apskatīt uzģenerētas metodes no IDL faila parauga (att. 3.14., šeit oranžā krāsa rāmītis). Violetos rāmīšos ir uzrakstītas uzģenerētas metodes IDL failam. Katra jauna klase atbilst jaunajam uzģenerētam kodam.



3.18. att. **Uzģenerētas metodes pēc IDL faila parauga**

Tagad, kad metode **MethodName** jau saņēma uzģenerētas klases objektu, ir iespējams palaist uzģenerēto kodu. Attēlā 3.17. redzamā diagramma apraksta situāciju, kad ir nepieciešams vismaz vienu reizi izmantot uzģenerēto kodu:

1. ja ir jāizmanto funkcija, kas atrodas ārpus python moduļa, tad metode **MethodName** izsauks uzģenerēto koda metodi, kuru nosaukums sakrīt ar **MethodName** (WP_INNER_WEBCALL). Uzģenerētais kods aizsūta webAppOS 3.19. attēla informāciju.

```
INNER_WEBCALL
{"webMemoryUrl": “1-2-3-4-5-6”, \
"objectReference": 56980, \
"className": CLASS_NAME, \
"methodName": METHOD_NAME, \
"arguments" : ARGUMENTS, \
"context" : {… }, \
"callNo" : 28590
"idle" : IDLE }
```

3.19. att. **Ārējas funkcijas izsaukums**

- webMemoryUrl (simbolu virkne)– adrese, kur glabājas reģistrēta lietotāja dati tīmekļa atmiņā (saņem no **MethodName**);
- objectReference (vesels pozitīvs skaitlis) – norāde uz atmiņas daļu, kur glabājas informācija par klasi CLASS_NAME objektu (saņem no **MethodName**);
- className (simbolu virkne) – klase, kur glabājas ārēja funkcija (lasa no IDL faila)
- methodName – metodes nosaukums (lasa no IDL faila);
- arguments (saraksts)– klases CLASS_NAME atribūti (lasa no IDL faila);
- context (vārdnīca) – var ietvert informāciju par esošo lietotāju, programmēšanas valodu vai vietu, kur atrodas izsauktais kods;
- callNo (vesels pozitīvs skaitlis) – vaicājuma numurs pēc kārtas;
- idle (boolen True/False) –tukšs vai false. Turpmāk glabās informāciju, vai tīmekļa progrsors varēs saņemt cirtus metodes izpildes pieprasījumus laikā, kad tas gaida atbildi no ārējas funkcijas.

Uzģenerētais kods saņems no webAppOS atbildi sekojošā veidā (3.20. att.)

```
INNER_WEBCALL_RESULT
{"webMemoryUrl": “1-2-3-4-5-6”, \
 "result": RESULT, \
 "error": RESULT, \
"callNo": 28590 }
```

3.20. att. **Ārējas funkcijas izsaukuma rezultāts**

- webMemoryUrl (simbolu virkne)– adrese, kur glabājas reģistrēta lietotāja dati tīmekļa atmiņā;
- result (simbolu virkne, **tikai veiksmīgs gadījums**) – funkcijas rezultāts;
- error (simbolu virkne, **tikai kļūdas gadījumā**) – kļūdas paziņojums;
- callNo (vesels pozitīvs skaitlis) – sakrīt ar aizsūtīto vaicājumu kārtas numuru.

2. ja ir jāizmaina kaut kāda atribūta vērtība (3.17. att. kreisā pusē, nosacījums [Nē]), tad tiks izsaukta datu ierakstīšana, kas aprakstīta 3.2.2. nodaļā.
3. ja ir jāizlasa aktuāla kaut kāda atribūta vērtība (3.17. att. kreisā pusē, nosacījums [Nē]), tad tiks izsaukta datu lasīšana uzģenerēta kodā. Datu lasīšana aprakstīta 3.2.3. nodaļā.

3.3. Python adapteris

Python adapteris pildīs divas galvenās funkcijas – python objekta izveidošana un metodes izsaukums. Paša sākuma nav zināms, kura metode būs palaista, adapterim būs jāpieslēdz attiecīgās klases dinamiski. Piemēram, lai dinamiski pieslēgtu metodes failu "import fileName" var izmantot python funkciju "exec", bet savukārt objekta izveidošanai var izmantot funkciju "eval". Vienam tīmekļa procesoram var būt vairāki adapteri dažādām programmēšanas valodām. Līdz ar to ir pieņemts lēmums izveidot divas atsevišķas moduļa daļas: tīmekļa procesors komunicē ar webAppOS, bet adapteris izsauc tieši python valodas metodes.

3.3.1. Python objekta izveidošana

Metode ir paredzēta jauna objekta izveidošanai (PA_OBJECT_CREATE), kas ļaus turpmāk palaist reģistrēta lietotāja izsauktu metodi. Python adapteris saņems no tīmekļa procesora:

1. **fileName** (simbolu virkne)– faila nosaukums, kur glabājas metode (procesors saņem to no methodUrl, ko apraksta 3.2.1.2. nodaļa);
2. **className** (simbolu virkne) – klases nosaukums, kur glabājas metode (procesors izlasa no methodUrl);
3. **savedArgsForConstructor** (saraksts ar simbolu virknēm) – argumenti klases konstruktora izveidošanai (lasa no tīmekļa atmiņas, sk. 3.2.1. nodaļā, 3.3. att.)

Lai izveidotu python objektu būs jāizpilda sekojoši soļi:

1. importē failu **fileName**, lai piekļūtu klases konstruktora datiem;
2. pieslēdz atbilstoši klasi **className**, kas atrodas failā **fileName**;
3. klases konstruktoram jāpadod atribūti **savedArgsForConstructor**, ko nav iespējams izdarīt uzreiz, jo atribūti var būt padoti jauktā secībā. Lai novērstu tādu problēmu, ir jāuzzina klases **className** atribūti pirms objekta izveidošanas. To palīdzes izpildīt "inspect.getfullargspec" funkcija. Kā parametru padosim konstruktoru **className**.__init__. Mainīgajā saglabā korektu atribūtu secību;
4. adaptera metode WriteMethodArgs katram klases atribūtam atrod atbilstošo vērtību no ievades datiem **savedArgsForConstructor**. Kā speciālais gadījums ir atribūts 'factory', kas neglabājas tīmekļa atmiņā, jo tā ir tehniskā informācija (sk. 3.2.4.2. nodaļu), Tādēļ, ja sarakstā **savedArgsForConstructor** ir tāds atribūts, tad tas arī būs pievienots funkcijas rezultātā pareizā vietā
5. beidzot var padot konstruktora argumentus, izveidot objektu klasei **className**.

3.3.2. Python metodes izsaukums

Metode ir paredzēta python metodes izsaukšanai (PA_METHOD_CALL). Python adapteris saņems no tīmekļa procesora sekojošus datus:

1. **obj** – objekts, kas tika izveidots 3.3.1. nodaļā aprakstītajā metodē;
2. **methodName** (simbolu virkne) – metodes nosaukums (izlasīts no methodUrl);
3. **argsForMethod** (saraksts ar simbolu virknēm) – argumenti klases metodes izpildei (procesora lasa no tīmekļa atmiņas, sk. 3.2.1. nodaļā, 3.3. att.)

Lai izsauktu atbilstošu klases metodi būs jāizpilda vairāki soļi:

1. līdzīgi python objekta izveidošanai, jāizlasa metodes **methodName** argumenti;
2. izsaucam funkciju WriteMethodArgs un padodam tajā izlasītus argumentus un **argsForMethod** sarakstu. Funkcija lasa katru metodes argumentu un sakārto tos pareizā secībā. Ja metodes argumentu sarakstā būs ‘factory’, tad arī būs ierakstīts tāds arguments. Kā parametru ‘factory’ metodei nodosim tīmekļa atmiņas Url (webMemoryUrl), lai, kad kods būs palaists, būtu izveidots objekts ‘factory’, kas būs nepieciešams metodei **methodName**;
   Piezīme: factory ir tehniska klase, kas ļauj veidot python objektus atbilstoši objektiem tīmekļa atmiņā. Šādi tiek realizēts Factory paraugs.
3. tagad izsaucam metodi **methodName**, padodot tai argumentus pareizā secībā;
4. tīmekļa procesors var atgriezt webAppOS tikai simbolu virknes, bet var gadīties, ka metodes **methodName** rezultāts ir kaut kādas klases objekts. Līdz ar to būs nepieciešams pārveidot objektu par norādi tīmekļa atmiņā (objectReference), kur glabāsies dati šī objekta atkārtotai izveidošanai. Programmētājam būs jāizveido papildus funkcija toWebMemoryObjectReference (factory) metodē **methodName**.
   4.1. Vispirms adapterī notiek pārbaude, vai iegūta atbilde ir objekts;
   4.2. ja rezultāts ir objekts, tad adapterī tiks izveidots Factory klases objekts;
   4.3. factory objekts tiks padots metodei toWebMemoryObjectReference. Šī metode savukārt izsauks factory uzģenerēto kodu, lai saņemtu jaunu objektu, kas glabās sevī norādi uz tīmekļa atmiņu (objectReference);
   4.4. adapteris lasa doto norādi un atgriež tīmekļa procesoram kā metodes rezultātu;
5. ja metode **methodName** atgriež ne objektu, tad arī objekts būs aizsūtīts procesoram.

3.4. SSL atslēgu pāru modulis

Visas SSL atslēgu pāru moduļa klases metodes ir sadalītas trīs klasēs. Pirmā klase ir atslēgu pāru ģenerēšanai (**MinicaSecurity**), kas ir aprakstīta 3.4.2. nodaļā. Otrā klase ir atslēgu pāra informācijas lasīšanai (**KeyPair**), kas ir definēta 3.4.3. nodaļā. Trešā klase ir paredzēta vienas atslēgas lasīšanai (**CertReading**), kuru var apskatīt 3.4.3. nodaļā.

SSL atslēgu pāru moduļa pirmkods būs izvietots mapē, kas attēlota 3.21. attēlā:

USER_PATH/webAppOS/dist/apps/CA.weblibrary/miniCA_python

3.21. att. **SSL atslēgu pāra pirmkoda izvietojums mapē**

- USER_PATH – reģistrēta lietotāja mape, kur glabājas webAppOS sistēma.

Visi izveidoti SSL atslēgu pāri glabāsies 3.22. attēlā norādītajā mapē:

USER_PATH/webAppOS/dist/etc/minica/caName

3.22. att. **SSL atslēgu pāra saglabāšana mapē**

- USER_PATH – reģistrēta lietotāja mape, kur glabājas webAppOS sistēma;
- caName – uzņēmuma nosaukums, kam ir jāizveido atslēgu pāris. Katram uzņēmumam būs sava mape, kur glabāsies tās atslēgu pāri.

3.4.1. Lietojumprogramma Minica

Šī darba ietvaros tiks izmantota miniCA lietojumprogramma, kas izmanto RSA algoritmu, publisko un privāto atslēgu ģenerēšanai, kuru ir iespējams izmantot HTTPS savienojuma izveidei mazajos uzņēmumos [8]. MiniCA strādā programmēšanas valodas go vidē.

Paša sākumā ir jāizvēlas mape SSL atslēgu pāru ģenerēšanai. Ja ievadītajā mapē pašlaik nav neviena lietojumprogrammas uzģenerēta atslēgu pāra, tad miniCA ģenerē root-sertifikātu, kas satur informāciju par izdevēju, versiju, sertifikātu sērijas numuru. Turpmāk miniCA ģenerē privātas un publiskas atslēgas, un uzreiz parakstīs tās ar root-sertifikātu. Tālāk SSL atslēgu pāru modulis aizsūtīs tās adapterim.

Let's Encrypt, uzņēmums, kas nodrošina bezmaksas kriptogrāfijas sertifikātu pieejamību, rekomendē lietojumprogrammu miniCA mazo uzņēmumu ikdienas darbā [9]. Līdz ar to miniCA var būt brīvi un droši izmantota arī šajā darbā. Tiks izmantota 1.0.2. versija.

3.4.3. SSL atslēgu pāra ģenerēšana (MinicaSecurity)

Klase atslēgu pāru ģenerēšanai saturēs vairākus atribūtus konstruktora būvei, kuru saņems no python adaptera:

1. **dictionary** (vārdnīca) – satur informāciju par visiem domēna vārdiem un IP adresēm, kuram būs jāuzģenerē atslēgu pāri;
2. **caName** (simbolu virkne) – uzņēmuma nosaukums, kuram ir jāizveido atslēgu pāri.

Konstruktorā arī jāidentificē mapi, kur glabāsies atslēgu pāri (att. 3.22.). Var izmantot '__file__', lai uzzinātu pirmkoda atrašanas vietu (att. 3.21.), no kā varam uzzināt ceļu līdz mapei 'dist', kas sastapās gan pirmajā, gan otrajā gadījumā. Beigās pievienojam ceļam 'etc/minica/**caName**'.

Gala lietotāja datorā var nebūt minica lietojumprogrammas, tāpēc datoram atkarībā no operētājsitēmas un procesora būs palaista sava minica. To var apskatīt 3.23. attēlā.

```
minica-CPU-OS .exe
```

3.23. att. **minica versija atkarībā no OS, CPU**

- CPU – lietotāja procesors. Var būt: 'x86_64' vai 'amd64';
- OS – lietotāja operētājsistēma. Varbūt: 'Windows', 'Linux' vai 'macos';
- .exe – paplašinājums būs pievienots, tikai ja operētājsistēma būs Windows.

Modulis var saņemt gan pareizus, gan kļūdainus domēna vārdus un IP adreses. Līdz ar to KP_SINGLE_CERT un KP_DIFFERENT_CERT metodēm ir jāpārbauda, vai ievaddati ir korekti. Domēna vārdu pārbaudei būs izmantota regulāra izteiksme:

1. domēna vārds var sastāvēt no vairākām daļām, kuras ir jāatbala ar punktiem;
2. augšējā un zemā līmeņa domēnam ir jāsastāv no lielajiem vai mazajiem latīņu alfabēta burtiem, cipariem, domuzīmēm, kuras nevar būt uzraksta sākumā un beigās;
3. domēna vārda paša sākumā var būt viena '*', kas nozīmēs jebkurus augšēja līmeņa domēnus;
4. pilnais domēna vārds nevar būt garāks par 253 simboliem;
5. maksimālais apakšdomēnu dziļums ir 127 (to var arī pārbaudīt ar 4.punktu);
6. katras uzraksts nevar pārsniegt 63 simbolus.

IP adrešu pārbaudei būs izmantota python funkcija 'ip_address (address)', kur address – IPv4 ievadīta IP adrese.

3.4.3.1. Dažādu SSL atslēgu pāra izveidošana

Metodei (KP_DIFFERENT_CERT) nav argumentu, kurus ir nepieciešams aizsūtīt adapterim. Visi nepieciešami dati tiks glabāti klases konstruktorā. Metodei ir jāizveido viens vienīgs atslēgu pāris, kas derētu kā atslēgas jebkuram no norādītiem domēna vārdiem vai jebkurai norādītajai IP adresei. Metodes izpildes soļi:

1. sākumā notiks pārbaude vai lietotāja operētājsistēma un procesors atbilst prasībām:
   1.1. ja operētājsistēma ir Windows, Linux vai macos, tad turpinām darbu, citādi būs atgriezta kļūda;
   1.2. ja procesors ir ‘x86_64’ vai ‘amd64’, tad turpinām darbu, pretējā gadījumā būs atgriezts kļūdas paziņojums;
2. atkarībā no operētājsistēmas un procesora būs izvēlēta pareiza minica lietojumprogrammas versija, to var vēlreiz izlasīt 3.4.3. nodaļās sākumā, 3.23. attēlā;
3. pēc kārtas tiks apstrādāti visi domēna vārdi. Ja domēna vārds ir korekts, tad tam būs izveidots atbilstošs atslēgu pāris 3.22. attēla ceļā. Katrs jaunais atslēgu pāris būs izveidots sava mapē, kuru nosaukums sakrīt ar domēna vārdu. Jāievēro, ka domēna vārds var sākties arī ar ‘*’ simbolu, kas nozīmēs, ka būs izveidots atslēgu pāris, kas derēs jebkuriem domēna vārdiem, kas ir apakšdomēni dotajam domēna vārdam.
4. pēc kārtas būs tiks apstrādātas visas IP adreses. Katrai korektai IP adresei būs izveidota sava mape 3.22. attēla ceļā (līdzīgi 3. punktam).
5. beigās, kad visi dati ir pārbaudi un ir izveidots vismaz viens SSL atslēgu pāris, tad metode atgriezīs rezultātu ‘True’. Ja visi ievades dati ir kļūdaini vai procesā ir sastapusies cita kļūda, tad metode atgriezīs šo kļūdu.

Kaut kādam domēna vārdam vai IP adresei var būt jau izveidots to atslēgu pāris līdz ar to būs izpildīti sekojoši soļi katra atslēgu pāra izveidošanā:

1. mapē (3.22. att.) būs izveidota pagaidu mape, izmantojot mkdtemp() funkciju;
2. šajā pagaidu mapē būs ģenerēts atslēgu pāris, lai izvairītos no kļūdām, kas var rasties, ja ģenerēšanas laikā notiks kļūda galvenajā mapē;
3. ja atslēgu pāris pagaidu mapē ir izveidots veiksmīgi, tad veca mape ar atslēgu pāri (ja tāda eksistē) būs dzēsta un tajā vietā būs izveidota jauna mape ar jaunu atslēgu pāri;
4. kad pagaidu mapes saturs būs pārvietots, tā būs dzēsta.

3.4.3.2. Kopīga SSL atslēgu pāra izveidošana

Metode (KP_SINGLE_CERT) ir paredzēta viena vienīga atslēgu pāra izveidošanai, kas derēs jebkuram konstruktora saglabātam domēna vārdam vai IP adresei. Metodei nav ievades datu.

1. sākumā notiks pārbaude vai lietotāja operētājsistēma un procesors atbilst prasībām:
   1.1. ja operētājsistēma ir Windows, Linux vai macos, tad turpinām darbu, citādi būs atgriezta kļūda;
   1.2. ja procesors ir ‘x86_64’ vai ‘amd64’, tad turpinām darbu, pretējā gadījumā būs atgriezts kļūdas paziņojums;
2. atkarībā no 1. punkta būs izvēlēta atbilstoša minica lietojumprogramma, to apraksta 3.4.3. nodaļa 3.23. attēlā;
3. pēc kārtas būs lasīti visi domēna vārdi. Notiks pārbaude vai izlasītais domēna vārds atbilst 3.4.3. nodaļā izvirzītam prasībām. Visus korektus domēna vārdus ir jāsaglabā **1. sarakstā**;
4. pēc kārtas būs izlasītas visas IP adreses. Ja adrese ir korekta, tad tā būs pievienota **2. sarakstam**;
5. kad visi dati ir izlasīti notiks viena SSL atslēgu pāra ģenerēšana. Atslēgu pāris varēs izmantot tikai korektiem iepriekš pārbaudītiem domēna vārdiem un IP adresēm.

Metodes beigās būs izveidots klases **KeyPair** objekts, kuram kā konstruktora parametri būs padota vārdnīca ar **izveidotiem sarakstiem** un uzņēmuma nosaukums. Metode atgriež python adapterim **KeyPair** objektu. Kā objekts tiek pārveidots par norādi tīmekļa atmiņā var izlasīt 3.3.2.nodaļā 4. punktā.

3.4.4. Atslēgu pāra informācijas lasīšana (KeyPair)

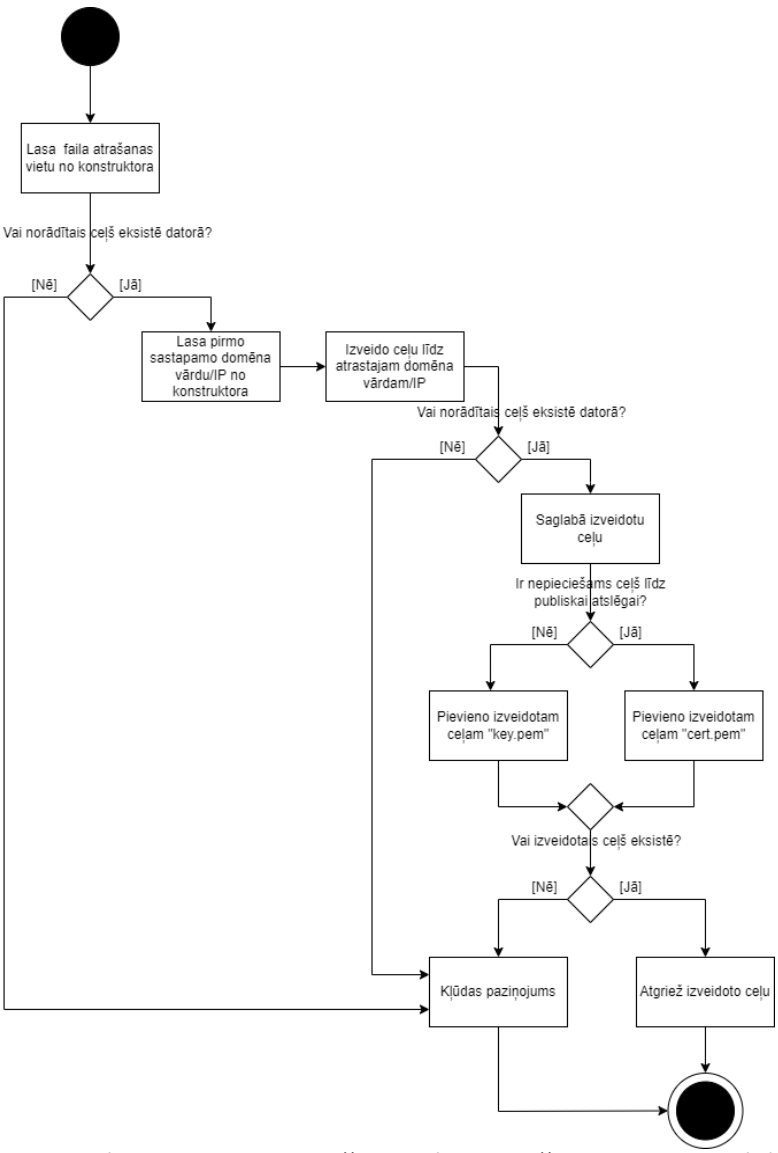
Klase **KeyPair** ir paredzēta atslēgu pāra informācijas lasīšanai. Klases konstruktora dati būs aizsūtīti pēc metodes KP_SINGLE_CERT izpildes. Savukārt šos datus adapteris aizsūtīs **KeyPair** klasei:

1. **domains_and_ips** (vārdnīca) – satur korektus domēna vārdus un IP adreses, kuram jau bija uzģenerēts atslēgu pāris;
2. **caName** (simbolu virkne) – uzņēmuma nosaukums, kam pieder atslēgu pāris.

Konstruktorā arī jāidentificē mapi, kur glabāsies atslēgu pāri (att. 3.22.). Var izmantot '__file__', lai uzzinātu pirmkoda atrašanas vietu (att. 3.21.).

3.4.4.1. Publiskas atslēgas atrašanas vietas lasīšana

Metodei KP_READ_PUBLIC_NAME nav ievades datu. Soļus, lai uzzinātu publiskās atslēgas atrašanas vietu jeb ceļu, var apskatīt 3.24. attēlā.



3.24. att. **Publiskas un privātas atslēgas atrašanas vietas lasīšana – UML aktivitāšu diagramma**

Vispirms notiek pārbaudē vai konstruktorā saglabātais ceļš līdz uzņēmuma mapes vispār eksistē. Tā kā visi domēna vārdi un IP adreses pieder tieši vienam atslēgu pārim, ir jāatrod pirmais sastopamais domēna vārds/IP adrese. Tas nosaukums arī sakritīs ar atslēgu pāra mapes nosaukumu. Atkal notiek pārbaude, vai vispār tāds ceļš eksistē un ir pieejams. Ja ir, tad esošam ceļam būs pievienots arī ‘cert.pem’ publiskas atslēgas faila nosaukums. Beigās notiks pēdējā pārbaude, vai ‘cert.pem’ fails vispār eksistē un nav dzēsts no tas mapes. Ja tāds fails eksistē, tad metode atgriež ceļu līdz šim failam.

3.4.4.2. Privātas atslēgas atrašanas vietas lasīšana

Metode KP_READ_PRIVATE_NAME ir ļoti līdzīga 3.4.4.1. nodaļai. Algoritmu privātas atslēgas atrašanas vietas noteikšanai arī var apskatīt 3.24. attēlā. Ja konstruktorā glabātais ceļš eksistē, tad būs definēts mapes nosaukums, kur glabājas privātā atslēga. Mapes nosaukumam ir jāsakrīt ar pirmo sastopamo domēna vārdu/IP adresi no konstruktora. Ja noradītais ceļš eksistē, tad tam būs pievienots faila ‘key.pem’ nosaukums. Metode atgriež norādīto ceļu, ja iepriekš nav iestājies kļūda.

3.4.4.3. Publiskas atslēgas faila lasīšana

Metode ar identifikatoru KP_READ_PUBLIC_FILE ir paredzēta, lai izlasītu publiskas atslēgas faila saturu un atgriezt to kā vienas rindas simbolu virkni Base64 kodējumā. Būs jāizpilda sekojoši soļi:

1. uzzināt publiskās atslēgas faila atrašanas vietu (sk. 3.4.4.1. nodaļu);
2. atvērt failu ‘cert.pem’ lasīšanas formātā;
3. lasīt failu pa rindām:
   3.1. izlaist faila pirmo un pēdējo rindu, jo tā satur tikai paziņojumus par atslēgas sākumu un beigām;
   3.2. nepievērst uzmanību jaunas rindas simboliem, jo atbildei jābūt vienā rindā (simbolu virkne)
4. aizvērt failu ‘cert.pem’;
5. ja lasīšanas laikā notiks kļūda, tad būs atgriezta attiecīgs kļūdas paziņojums.

3.4.4.4. Atslēgu pāra derīguma termiņa lasīšana

Metode KP_EXPIRATION_DATE nesaņem ievades datus no python adaptera. Metodes izpildīšanai būs nepieciešams tikai klases **KeyPair** atribūti. Tā kā gan privātas, gan publiskas atslēgas fails bija izveidots viena laikā, nav lielas atšķirības, kuru failu izlasīt. Tiks izvēlēts publiskas atslēgas fails 'cert.pem', lai izvairītos no kļūdām privāta faila lasīšanas gaitā.

Derīguma termiņa noteikšanai ir jāizpilda soļi:

1. uzzināt publiskās atslēgas faila atrašanas vietu (sk. 3.4.4.1. nodaļu);
2. atvērt failu 'cert.pem' lasīšanas formātā;
3. importē python bibliotēku 'crypto', kas dod iespēju izlasīt kriptogrāfijas failus;
4. ar funkciju 'load_certificate' izlasām visus datus, kas glabājas failā;
5. lasām faila derīguma termiņu 'get_notAfter()';
6. pēc noklusējuma datums var atšķirties no nepieciešama formāta, tāpēc derīguma termiņa datums jāpārveido formātā 'yyyy-mm-ddThh:mm:ss".
7. aizveram failu 'cert.pem'.

Beigās, ja nav kļūdu, metode atgriež derīguma termiņu (simbolu virkne).

3.4.4.5. Atslēgu pāra atjaunošana

Metode KP_UPDATE_CERT atjauno SSL atslēgu pāri. Metodei nav ievades datu, visa nepieciešama informācijā tiks paņemta no klases atribūtiem. Metode izpildes soļus var apskatīt zemāk:

1. vispirms būs izveidots klases **MinicaSecurity** objekts (sk. 3.4.3. nodaļu), kuram kā atribūtus tiks padota vārdnīca ar domēna vārdiem un IP adresēm, uzņēmuma nosaukumu, tas ir, klases **KeyPair** atribūti;
2. izsauc KP_SINGLE_CERT metodi (metodes projektējumu var redzēt 3.4.3.2. nodaļā);
3. atgriež metodes KP_SINGLE_CERT rezultātu python adapterim.

3.4.5. Vienas atslēgas lasīšana (CertReading)

Klase **CertReading** ir paredzēta vienas atslēgas informācijas lasīšanai. Python adapteris aizsūtīs klasei sekojošus datu konstruktora izveidei:

1. **PATH** (simbolu virkne) – ceļš līdz privātai vai publiskai atslēgai kopā ar faila nosaukumu un paplašinājumu.

3.4.5.1. Atslēgu pāra domēna vārdu un IP adrešu lasīšana

Metode KP_DEFINE_CONTENT lasa no faila visus domēna vārdus un IP adreses, kuram var izmantot doto atslēgu HTTPS savienojuma nodrošināšanai un atgriež sarakstu ar tiem elementiem. Būs jāizpilda sekojoši soļi:

1. importē bibliotēku ‘ssl’ faila lasīšanai;
2. atvērt un lasīt failu, kas atrodas **PATH** mapē ar ‘ssl._ssl._test_decode_cert’ funkciju;
3. ja izdevās atvērt un izlasīt failu, tad ierakstām visus atslēgas domēna vārdu un IP adreses (‘subjectAltName’) viena sarakstā;
4. aizvērt failu;
5. ja nav kļūdu, tad metode atgriež izveidoto sarakstu.

3.4.5.2. Atslēgu pāra izdevēja lasīšana

Metode KP_DEFINE_ISSUER ir paredzēta atslēgu pāra izdevēja lasīšanai. Ir zināms, ka gan publiska, gan privāta bija izveidota viena laikā, tāpēc nav svarīgi ceļš līdz kurai atslēgai glabājas kā atribūts. Lai uzzinātu atslēgas izdevēju:

1. importē ‘ssl’ bibliotēku faila lasīšanai;
2. atvērt failu, kas atrodas **PATH** mapē;
3. lieto ‘test_decode_cert’ funkciju un meklē izdevēju pēc atslēgas ‘issuer’;
4. aizvērt failu;
5. ja nebija kļūdu, tad metode atgriež izdevēja nosaukumu.

3.5. Mapju šifrēšanas modulis

Mapju šifrēšanas moduļa pirmkods būs izvietots mape, kas attēlota 3.25. attēlā:

USER_PATH/webAppOS/dist/apps/CA.weblibrary/cryFS_python

3.25. att. . **Mapju šifrēšanas pirmkoda izvietojums mapē**

- USER_PATH – reģistrēta lietotāja mape, kur glabājas webAppOS sistēma.

Modulī būs viena klase ar trim metodēm, kas konstruktorā no adaptera saņems:

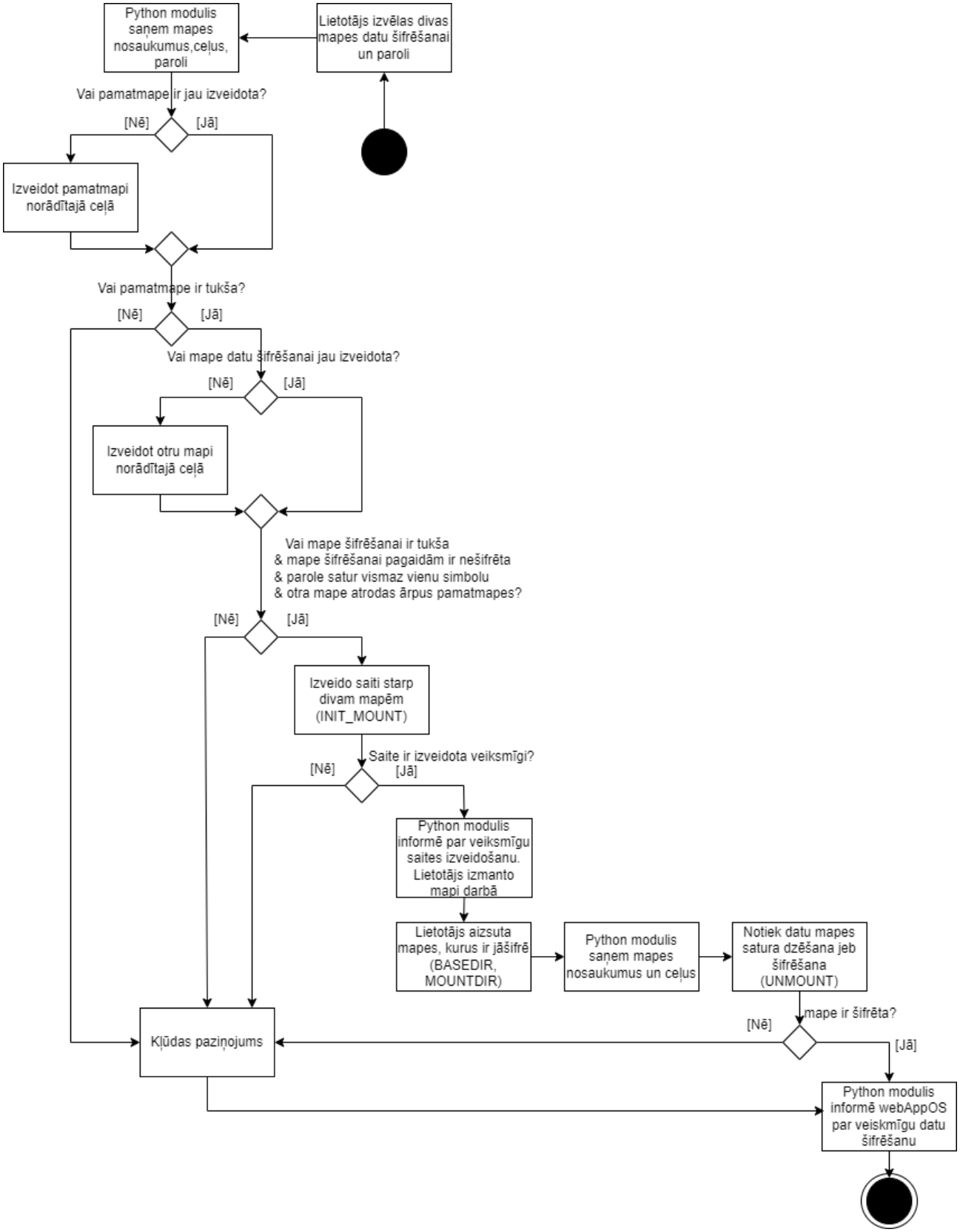
1. **BASEDIR** (simbolu virkne) – pamatmapes ceļš;
2. **MOUNTDIR** (simbolu virkne) – ceļš līdz datu mapei.

3.5.1. Lietojumprogramma CryFS

Projekta ietvaros tika izvēlēta lietojumprogramma CryFS, kas dod iespēju šifrēt un atšifrēt lietotāja mapes. Šifrēšana notiks ar divām mapēm: pamatmape (BASEDIR) un mape, kur glabāsies reģistrēta lietotāja informācija (MOUNTDIR). Paša sākumā būs izsaukta metode DE_INIT_MOUNT (sk. 3.5.2. nodaļā), kas izveidos saiti starp mapēm. Pēc metodes izpildes lietojumprogramma datu mapi (MOUNTDIR) pārveidos par virtuālo datu nesēju, kurā lietotājs var brīvi strādāt – pievienot, rediģēt un dzēst informāciju. Pēc darba pabeigšanas būs izsaukta DE_UNMOUNT (sk. 3.5.3. nodaļā), kas paslēps no datu mapes (MOUNTDIR) visu informāciju, pamatmapē paliks tikai šifrēti dati. Kad lietotājs gribēs atkal atšifrēt savu informāciju būs izsaukta DE_MOUNT (sk. 3.5.4. nodaļā), kas atkal, izmantojot pamatmapes šifrētus datus, pārveidos tukšu mapi (MOUNTDIR) par virtuālo datu nesēju ar atšifrētiem datiem. Var apskatīt cryFS lejupielādes [10] un lietošanas instrukciju [11].

Mapju šifrēšanu piedāvā arī citas lietojumprogrammas, piemēram, fscrypt. Tomēr viens no svarīgākiem kritērijiem mapju šifrēšanas programmas izvēlē bija iespēja izmantot to kā moduli, gan windows, gan linux, gan macos operētājsistēmā. CryFS ir iekļauts linux ubuntu pakotņu sarakstā, to ir iespējams lejupielādēt gan windows, gan macos vidē, tāpēc bija pieņemts lēmums izmantot cryFS mapju šifrēšanai.

CryFS ir atvērta koda programmatūra, kuru kodu var apskatīt ikviens [12]. PPA rakstīšanas un realizācijas laikā tika izmantota pēdējā 0.11.1. versija. CryFS var izmantot droši, jo tā izmanto pasaulē atzītos šifrēšanas algoritmus, tā ir atvērta koda programma (var pārbaudīt kodu), kā arī par CryFS ir uzrakstītas zinātniskās publikācijas.



3.26. att. . **Datu šifrēšanas ceļš – UML aktivitāšu diagramma**

3.5.2. Mapes izveidošana turpmākai šifrēšanai

Metode DE_INIT_MOUNT ir paredzēta jaunas mapes izveidošanai jeb jauna virtuāla datu nesēja izveidei, kur glabāsies lietotāja nešifrēti dati. No python adaptera metode saņems paroli (simbolu virkne).

Attēlā 3.26. var apskatīt ceļu pirmajai datu šifrēšanai (metodes DE_INIT_MOUNT un DE_UNMOUNT). Vispirms webAppOS aizsūta mapju šifrēšanas modulim mapes nosaukumus, kur būs izvietota pamatmape un datu mape. Ja tādu mapju nav, tad metode izveidos dotas mapes noradītajā ceļā. Tālāk notiks pārbaude, ka abas mapes ir tukšas, datu mape (MOUNTDIR) nav jau šifrēta veidā un neatrodas pamatmepē (BASEDIR) un parole nav tukša. Ja visi nosacījumi ir izpildīti, tad tiks palaista cryFS programma, lai izveidotu saiti starp mapēm. Ja nav kļūdu, tad lietotājs saņems rezultātu True.

Lai izveidotu saiti starp mapēm būs jālieto komanda, kas redzama 3.27.attēlā.

```
cryfs BASEDIR MOUNTDIR
```

3.27. att. . **Komanda mapes saites izveidei turpmākai šifrēšanai**

1. **BASEDIR** (simbolu virkne) – pamatmapes ceļš, kas būs saņemts konstruktorā;
2. **MOUNTDIR** (simbolu virkne) – ceļš līdz datu mapei, kas būs saņemts konstruktorā.

Ir jāievēro, ka moduļa metodei ir jādarbojas pat, ja saite starp mapēm ir jau izveidota, jo lietotājs strādās datu mapē jeb virtuāla datu nesējā (MOUNTDIR), Lai nepazaudētu saiti starp pamatmapi, kur dati paši atjaunosies reālajā laikā, un datu mapi, ir jānodrošina procesa darbību arī pēc rezultāta paziņošanas webAppOS par veiksmīgas metodes rezultātu. Pati klases metode beigs darbu uzreiz pēc rezultāta atgriešanas adapterī, tāpēc būs jāizveido otrs programmas pavediens (thread), kur arī palaižam cryFS:

1. adapteris izsauc metodi DE_INIT_MOUNT;
2. importē '**Queue**' un '**Thread**' bibliotēkas divām datu plūsmām;
3. rindā **Queue** būs ievietots programmas cryFS izpildes rezultāts vai kļūda, lai to turpmāk varēja pārsūtīt adapterim;
4. izveidojam jaunu datu plūsmu **Thread** un tajā palaižam cryFS. Pateicam galvenajai programmai gaidīt, kamēr rindā **Queue** būs redzama atbilde;
5. palaižam cryFS otrajā plūsmā. Izvēlamies visus iestatījumus pēc noklusējuma un ievadam paroli. Lietojumprogrammas rezultātu ievietojam rindā;
6. galvenā programma lasa rezultātu no rindas un aizsūta to python adapterim.

3.5.4. Mapes šifrēšana

Metode DE_UNMOUNT paredzēta datu šifrēšanai un informācijas paslēpšanai no datu mapes. Metode nesaņem ievades datus no python adaptera. Mapju ceļi tiks padoti klases konstruktorā.

Attēlā 3.26. var apskatīt turpinājumu, kāda solī dati būs šifrēti. Pēc metodes DE_INIT_MOUNT izpildes webAppOS paziņos par datu mapes šifrēšanas nepieciešamību. Metode izmantos cryFS, lai šifrētu mapi. Ja iepriekš nebūs kļūdas, tad metode aizsūtīs python adapterim rezultātu True.

Lai šifrētu datu mapi, būs jāizmanto cryFS komanda, kas redzama 3.28. attēlā.

```
cryfs-unmount BASEDIR MOUNTDIR
```

3.28. att. . **Komanda mapes šifrēšanai**

1. **BASEDIR** (simbolu virkne) – pamatmapes ceļš, kas būs izlasīts no konstruktora;
2. **MOUNTDIR** (simbolu virkne) – ceļš līdz datu mapei, kas būs izlasīts no konstruktora.

3.5.5. Mapes atšifrēšana

Metode DE_MOUNT ir paredzēta mapju atšifrēšanai, kuram ir saglabāti šifrēta veidā dati pamatmapē. Metode saņems no python adaptera paroli mapes atšifrēšanai.

Lai atšifrētu datus ar cryFS būs jāizmanto komanda, kuru var redzēt 3.29. attēlā.

```
cryfs BASEDIR MOUNTDIR
```

3.29. att. . **Komanda mapes atšifrēšanai**

1. **BASEDIR** (simbolu virkne) – pamatmapes ceļš, kas padots konstruktora;
2. **MOUNTDIR** (simbolu virkne) – ceļš līdz datu mapei, kas tiek padots konstruktora.

Pašā sākumā notiks pārbaude, vai pamatmapē (BASEDIR) eksistē norādītajā ceļā un satur konfigurācijas failu ‘cryfs.config’, kas būs nepieciešams atšifrēšanai. Tālāk notiks pārbaude, vai norādīta datu mape (MOUNTDIR) ir tukša un jau atšifrēta, citādi nav jēgas to atkal atšifrēt. Pēc tādam pārbaudēm, notiks lietojumprogrammas cryFS palaišana, pie

nosacījuma ka parole nav tukša un datu mape neatrodas pamatmapē. Ja programmas cryFS darbā nebija kļūdu un mape patiešām ir atšifrēta, tad būs atgriezta True vērtība.

Atceramies, ka metodei ir jādarbojas arī pēc mapes atšifrēšanas, jo pēc tas atkal būs izveidota saite starp divām mapēm un, lai dati atjaunotos pamatmapē reālajā laikā būs jāveido divi pavedieni (threads): pavediens metodei un pavediens cryFS programmas izpildei. Ļoti līdzīgi 3.5.2. nodaļā aprakstītajam algoritmam izpildīsim darbības:

1. adapteris izsauc metodi DE_MOUNT;
2. importē '**Queue**' un '**Thread**' bibliotēkas divām datu plūsmām;
3. rindā **Queue** būs ievietots programmas cryFS izpildes rezultāts vai kļūda, lai to pārsūtītu adapterim;
4. izveidojam jaunu pavedienu **Thread** un tajā palaižam cryFS lietojumprogrammu. Pateicam galvenajai programmai gaidīt, kamēr rindā **Queue** būs atbilde;
5. palaižam cryFS otrajā plūsmā. Izvēlamies visus iestatījumus pēc noklusējuma un ievadam paroli. Ja parole nesakritīs ar sākuma paroli, tad cryFS izdos atbilstošu paziņojumu. Lietojumprogrammas rezultātu ievietojam rindā;

Pēc veiksmīga rezultāta lietotājs atkal var izmantot mapi savam darbam un vairākas reizes atkārtot DE_UNMOUNT un DE_MOUNT metodes.

4. TESTĒŠANAS DOKUMENTĀCIJA

4.1. Testēšanas plāns

Testēšanas mērķis ir pārbaudīt, vai sistēma atbilst PPS izvirzītajām prasībām. Testēšanas dokumentācija sastāv no testēšanas žurnāla, kur ir ierakstīti testu izpildes statusi noteikta laika periodā, moduļu vienībtestēšanas, kur bija testēti tikai atsevišķi moduļi, integrācijas testēšanas, kur bija notestēta tīmekļa procesora komunikācija ar adapteri un uzprojektētiem moduļiem. Testēšanas dokumentācija ietver sevī arī nefunkcionālo prasību testēšanu. Tā kā projekts bija veidots pēc spējas izstrādes modeles, projekta moduļi bija vairākas reizes testēti arī Python moduļa projektēšanas laikā.

4.2. Testēšanas žurnāls

Testēšanas žurnālā '+' nozīme, ka tests rezultāts ir veiksmīgs, '-', ka neveiksmīgs, 'n', ka testēšana dotajā vidē nebija veikta noradītajā datumā. Testēšana veikta trīs operētājsistēmās. Testēšanas vidi var apskatīt 4.1. attēlā.

4.1. tabula. **Testēšanas vide**

Operētājsistēma	Windows	Linux	Mac
Operētājsistēmas versija	10 Education 1909	Ubuntu 20.04. LTS	MacOS 12.1. Monterey
Programmēšanas valoda	Python	Python	Python
Versija	3.8.2	3.8.2	3.8.2

4.2.1. SSL atslēgu pāru modulis

SSL atslēgu pāru moduļa vienībtestēšanas žurnālu var apskatīt 4.2. tabulā.

4.2. tabula. **SSL atslēgu pāru moduļa vienībtestēšanas žurnāls**

Testa numurs	Datums	Rezultāts		
		Windows	Linux	MacOS
1.1.	28.12.2021	+	n	n
	10.01.2021	+	+	+
1.2.	28.12.2021	+	n	n
	10.01.2021	+	+	+
1.3.	28.12.2021	+	n	n

Testa numurs	Datums	Rezultāts		
		Windows	Linux	MacOS
	10.01.2021	+	+	+
1.4.	28.12.2021	+	n	n
	10.01.2021	+	+	+
1.5.	28.12.2021	+	n	n
	10.01.2021	+	+	+
1.6.	28.12.2021	+	n	n
	10.01.2021	+	+	+
1.7.	28.12.2021	+	n	n
	10.01.2021	+	+	+
1.8.	28.12.2021	- Atgriež ‘True’ nevis ‘Firm path is not accesible’	n	n
	09.01.2021	+	n	n
	10.01.2021	+	+	+
2.1.	28.12.2021	+	n	n
	10.01.2021	+	+	+
2.2.	28.12.2021	+	n	n
	10.01.2021	+	+	+
2.3.	28.12.2021	- Atgriež ‘True’ nevis kļūdu. ‘All invalid inputs or empty input’	n	n
	09.01.2021	+	n	n
	10.01.2021	+	+	+
3.1.	28.12.2021	+	n	n
	10.01.2021	+	+	+
3.2.	28.12.2021	+	n	n
	10.01.2021	+	+	+
3.3.	28.12.2021	+	n	n
	10.01.2021	+	+	+
4.1.	28.12.2021	+	n	n

Testa numurs	Datums	Rezultāts		
		Windows	Linux	MacOS
	10.01.2021	+	+	+
4.2.	28.12.2021	+	n	n
	10.01.2021	+	+	+
5.1.	28.12.2021	+	n	n
	10.01.2021	+	+	+
5.2.	28.12.2021	+	n	n
	10.01.2021	+	+	+
6.1.	28.12.2021	+	n	n
	10.01.2021	+	+	+
7.1.	28.12.2021	+	n	n
	10.01.2021	+	+	+
7.2.	28.12.2021	+	n	n
	10.01.2021	+	+	+
8.1.	28.12.2021	+	n	n
	10.01.2021	+	+	+
9.1.	28.12.2021	+	n	n
	10.01.2021	+	+	+

4.2.2. Mapju šifrēšanas modulis

Mapju šifrēšanas moduļa testēšanas rezultāti ir aprakstīti 4.3. tabulā.

4.3. tabula. **Mapju šifrēšanas moduļa vienībtestēšanas žurnāls**

Testa numurs	Datums	Rezultāts		
		Windows	Linux	MacOS
1.1.	04.01.2021	+	n	n
	10.01.2021	+	+	+
1.2.	04.01.2021	+	n	n
	10.01.2021	+	+	+
1.3.	04.01.2021	+	n	n
	10.01.2021	+	+	+
1.4.	04.01.2021	+	n	n

Testa numurs	Datums	Rezultāts		
		Windows	Linux	MacOS
	10.01.2021	+	+	+
2.1.	04.01.2021	+	n	n
	10.01.2021	+	+	+
2.2.	04.01.2021	+	n	n
	10.01.2021	+	+	+
3.1.	04.01.2021	+	n	n
	10.01.2021	+	+	+
3.2.	04.01.2021	+	n	n
	10.01.2021	+	+	+
3.3.	04.01.2021	+	n	n
	10.01.2021	+	+	+

4.2.3. Integrācijas testēšanas žurnāls

Integrācijas testēšanas žurnālu var redzēt 4.4. tabulā. Tests ir izpildīts, kad tas atbilst sagaidāmajām rezultātam un kļūdas paziņojums/metodes rezultāts ir angļu valodā, pārsūtīti dati starp webAppOS un Python moduli ir UTF-8 kodējumā, kas atbilst nefunkcionālajām prasībām.

4.4. tabula. **Integrācijas testēšanas žurnāls**

Testa numurs	Datums	Rezultāts		
		Windows	Linux	MacOS
1.	05.01.2021	+	n	n
	10.01.2021	+	+	+
2.	05.01.2021	+	n	n
	10.01.2021	+	+	+
3.	05.01.2021	+	n	n
	10.01.2021	+	+	+
4.	05.01.2021	+	n	n
	10.01.2021	+	+	+
5.	05.01.2021	+	n	n
	10.01.2021	+	+	+

Testa numurs	Datums	Rezultāts		
		Windows	Linux	MacOS
6.	05.01.2021	+	n	n
	10.01.2021	+	+	+
7.	05.01.2021	+	n	n
	10.01.2021	+	+	+
8.	05.01.2021	+	n	n
	10.01.2021	+	+	+
9.	05.01.2021	+	n	n
	10.01.2021	+	+	+
10.	05.01.2021	+	n	n
	10.01.2021	+	+	+
11.	05.01.2021	+	n	n
	10.01.2021	+	+	+
12.	05.01.2021	+	n	n
	10.01.2021	+	+	+
13.	05.01.2021	+	n	n
	10.01.2021	+	+	+
14.	05.01.2021	+	n	n
	10.01.2021	+	+	+
15.	05.01.2021	+	n	n
	10.01.2021	+	+	+
16.	05.01.2021	+	n	n
	10.01.2021	+	+	+
17.	05.01.2021	+	n	n
	10.01.2021	+	+	+
18.	05.01.2021	+	n	n
	10.01.2021	+	+	+

4.3. Moduļu vienībtestēšana

Katram testpiemēram ir noradīts testētas metodes identifikators un pamatojums, kāpēc ir veikts dotais tests. Ērtākai lasīšanai katra testpiemēra ievaddatos ir tukšumi. Pirmā python testpiemēra izpildi var apskatīt 4.1. attēlā. Pārējie testi bija palaisti analoģiski.

```
>>> from miniCA_python import MinicaSecurity
>>> A = MinicaSecurity({"domains": ["universitate.lv","*.u.fakultate.lk","*.vm"],"ip":
["192.18.0.34","192.18.1.112","87.2.0.2"]},"f1")
>>> A.generate_key_pair_dif()
True
>>>
```

4.1. att. **SSL atslēgu pāru moduļa 1.1. testpiemērs**

4.3.1. SSL atslēgu pāru moduļa testi

Tabulā 4.5. var apskatīt uzrakstītus testus SSL atslēgu pāru modulim. Labajā kolonā var redzēt sagaidāmo rezultātu.

4.5. tabula. **SSL atslēgu pāru moduļa vienībtestēšana**

N.p. k.	Metodes identifikators	Pamatojums	Ievaddati	Sagaidāmais rezultāts
1.1.	KP_ DIFFERENT_ CERT	Ievada visus korektus domēna vārdus un IP adreses. Jauna uzņēmuma pievienošana.	A = MinicaSecurity({ "domains": ["universitate.lv", "*.u.fakultate.lk", "*.vm"], "ip": ["192.18.0.34", "192.18.1.112", "87.2.0.2"]}, "f1") A.generate_key_pair_dif()	Izveido mapi ‘f1’, kura glabājas 6 jaunas mapes ar atslēgu pāriem. Mapē ‘f1’ izveido root-sertifikātu. Atgriež ‘True’.
1.2.	KP_ DIFFERENT_ CERT	Jauna korekta domēna vārda, IP adreses pievienošana iepriekš izveidotam uzņēmumam.	**<Palaiž 1.1. testu>** A = MinicaSecurity({ "domains": ["fakultate.lv"], "ip": ["193.18.0.34"]}, "f1") A.generate_key_pair_dif()	Papildina mapi ‘f1’ ar vienu mapi jaunajam domēna vārdam un mapi IP adresei, kur glabājas atslēgu pāris. Atgriež ‘True’

N.p. k.	Metodes identifikators	Pamatojums	Ievaddati	Sagaidāmais rezultāts
1.3.	KP_ DIFFERENT_ CERT	Domēna vārdu pārbaude. Pievieno nekorektus domēna vārdus.	A = MinicaSecurity({ "domains": ["univers-.lv", "**.domain.com", "vasar*a.net", "ziemassvētki.lv", "rudens", ".lv", "ziema.*.lv"], "ip": ["11.12.0.1"]}, "f2") A.generate_key_pair_dif()	Izveido mapi ‘f2’ ar vienu apakšmapi ’11.12.0.1’, kurā glabājas atslēgu pāris. Mapē ‘f2’ izveido root-sertifikātu. Atgriež ‘True’.
1.4.	KP_ DIFFERENT_ CERT	IP adrešu pārbaude. Pievieno nekorektas IP adreses.	A = MinicaSecurity({ "domains": ["1-2-3.df"], "ip": ["a.12.0.1", "196.8.0", "7-12-0-1", "19217801 ", "13.14.15.0."]}, "f3") A.generate_key_pair_dif()	Izveido mapi ‘f3’ ar apakšmapi ‘1-2-3.df’, kurā glabājas atslēgu pāris. Mapē ‘f3’ izveido root-sertifikātu. Atgriež ‘True’.
1.5.	KP_ DIFFERENT_ CERT	Domēna vārda garums ir lielāks nekā 100. (110)	A = MinicaSecurity({ "domains": ["1.1.1.(…).co"]}, "f4") A.generate_key_pair_dif()	Izveido mapi ‘f4’ ar apakšmapi ‘1.1.(…).co’, kurā glabājas atslēgu pāris. Izveido mapē ‘f4’ root-sertifikātu. Atgriež True.

N.p. k.	Metodes identifikators	Pamatojums	Ievaddati	Sagaidāmais rezultāts
1.6.	KP_ DIFFERENT_ CERT	Nav korektu domēna vārdu un IP adrešu.	A = MinicaSecurity({ "domains": ["#"], "ip": ["256.1.1.1"]}, "f7") A.generate_key_pair_dif()	Paziņo par kļūdu ar tekstu 'All invalid inputs or empty input'
1.7.	KP_ DIFFERENT_ CERT	Pievieno jaunu uzņēmumu, kam nav neviena domēna vārda un IP adreses.	A = MinicaSecurity({ "domains": [ ], "ip": [ ]}, "f8") A.generate_key_pair_dif()	Paziņo par kļūdu ar tekstu 'All invalid inputs or empty input'
1.8.	KP_ DIFFERENT_ CERT	Uzņēmuma nosaukums satur nepieļaujamus simbolus.	A = MinicaSecurity({ "domains": ["test.lt"]}, "=f9?") A.generate_key_pair_dif()	Paziņo par kļūdu ar tekstu 'Firm path is not accesible'
2.1.	KP_ SINGLE_ CERT	Jaunā uzņēmuma pievienošana. Viena atslēgu pāra ģenerēšana. Ir nekorekti ievades domēna vārdi un IP adreses.	A = MinicaSecurity({ "domains": ["ziema.lv", "*.vasaras.lpp", "*#",], "ip": ["197.1.4.3", "19r.18.1.112", "-1.5.8.9"]}, "f10") A.generate_key_pair_s()	Izveido mapi 'f10' ar vienu apakšmapi 'ziema.lv', kas satur atslēgu pāri. Pāris der 'ziema.lv', '*vasaras.lpp' un '197.1.4.3'. Mapē 'f10' izveido root-sertifikātu. Atgriež klases 'KeyPair' objektu.

N.p. k.	Metodes identifikators	Pamatojums	Ievaddati	Sagaidāmais rezultāts
2.2.	KP_ SINGLE_ CERT	Pirmais sastopamais domēna vārds nav korekts. Mapes nosaukumam jāsakrīt ar pirmo korektu sastopamu domēna vārdu.	A = MinicaSecurity({ "domains": ["--.lv", "???.", "kastani.py",], "ip": ["19.19.0.0"]}, "f11") A.generate_key_pair_s()	Izveido mapi ‘f11’ ar apakšmapi ‘kastani.py’, kurā glabājas atslēgu pāris. Mapē ‘f11’ izveido root-sertifikātu. Atgriež klases ‘KeyPair’ objektu.
2.3.	KP_ SINGLE_ CERT	Visi ievadīti domēna vārdi un IP adreses nav korektas.	A = MinicaSecurity({ "domains": ["-.lv"], "ip": [".1w.0.0"]}, "f12") A.generate_key_pair_s()	Paziņo par kļūdu ar tekstu ‘All invalid inputs or empty input’
3.1.	KP_ DIFFERENT_ CERT KP_READ_ PUBLIC_ NAME	Visi ievaddati ir korekti. Publiskās atslēgas atrašanas vietas lasīšana.	**<Iepriekš izpilda 1.1.>** A = KeyPair({ "domains": ["universitate.lv", "*.u.fakultate.lk", "*.vm"], "ip": ["192.18.0.34", "192.18.1.112", "87.2.0.2"]}, "f1") A.public_key_file_name()	Atgriež publiskas atslēgas atrašanas vietu.

N.p. k.	Metodes identifikators	Pamatojums	Ievaddati	Sagaidāmais rezultāts
3.2.	KP_READ_ PUBLIC_ NAME	Uzņēmuma nosaukums neeksistē, kur jāglabājas publiskai atslēgai.	A = KeyPair({ "domains": [ ], "ip": ["192.18.0.35"]}, "NOTexist") A.public_key_file_name()	Paziņo par kļūdu ar tekstu 'Firm path is not accesible'
3.3.	KP_READ_ PUBLIC_ NAME	Uzņēmuma mapē nav publiskas atslēgas.	**< Izveido tukšu mapi 'f13'>** A = KeyPair({ "domains": [ ], "ip": ["192.18.0.35"]}, "f13") A.public_key_file_name()	Paziņo par kļūdu ar tekstu 'Directory of <map directory> is not accesible'.
4.1.	KP_ DIFFERENT_ CERT KP_READ_ PRIVATE_ NAME	Visi ievaddati ir korekti. Privātas atslēgas atrašanas vietas lasīšana.	**<Izpilda 1.1. testu>** A = KeyPair({ "domains": ["universitate.lv", "*.u.fakultate.lk", "*.vm"], "ip": ["192.18.0.34", "192.18.1.112", "87.2.0.2"]}, "f1") A.private_key_file_name()	Atgriež privātas atslēgas atrašanas vietu.
4.2.	KP_READ_ PRIVATE_ NAME	Uzņēmuma nosaukums neeksistē, kur jāglabājas privātai atslēgai.	A = KeyPair({ "domains": [ ], "ip": ["192.18.0.35"]}, "NOTexist") A.private_key_file_name()	Paziņo par kļūdu ar tekstu 'Firm path is not accesible'

N.p. k.	Metodes identifikators	Pamatojums	Ievaddati	Sagaidāmais rezultāts
5.1.	KP_ DIFFERENT_ CERT KP_READ_ PUBLIC_ FILE	Ievaddati ir korekti, lasa publiskas atslēgas faila saturu.	**< Izpilda 1.1. testu>** A = KeyPair({ "domains": ["universitate.lv", "*.u.fakultate.lk", "*.vm"], "ip": ["192.18.0.34", "192.18.1.112", "87.2.0.2"]}, "f1") A.public_key_string()	Atgriež publiskas atslēgas faila saturu (simbolu virkne).
5.2.	KP_ SINGLE_ CERT KP_READ_ PUBLIC_ FILE	Metodei ir jāpārbauda vai vispār publiskas atslēgas fails eksistē, kuru ir jāizlasa.	**<Izpilda 2.1. testu>** **<Dzēš publiskas atslēgas failu mapē 'ziema.lv'>** A = KeyPair({ "domains": ["ziema.lv", "*.vasaras.lpp"], "ip": ["197.1.4.3"]}, "f10") A.public_key_string()	Atgriež kļūdas paziņojumu ar tekstu 'cert.pem file in <map directory> is not accessible''
6.1.	KP_ SINGLE_ CERT KP_ EXPIRATION _DATE	Ievaddati ir korekti. Pārbaude, vai derīguma termiņš būs izlasīts.	**<Izpilda 2.1. testu>** A = KeyPair({ "domains": ["ziema.lv", "*.vasaras.lpp"], "ip": ["197.1.4.3"]}, "f10") A.expiration_date()	Atrod mapes 'ziema.lv' atrašanas vietu un atgriež derīguma termiņu "yyyy-mm-ddThh:mm:ss" formātā.

N.p. k.	Metodes identifikators	Pamatojums	Ievaddati	Sagaidāmais rezultāts
7.1.	KP_ SINGLE_ CERT KP_ UPDATE_ CERT	Ievaddati ir korekti. Pārbaude, vai atslēgu pāris būs atjaunots.	**<Izpilda 2.1. testu>** A = KeyPair({ "domains": ["ziema.lv", "*.vasaras.lpp"], "ip": ["197.1.4.3"]}, "f10") A.renew()	Metode atjauno 2.1. testā izveidoto atslēgu pāri un atgriež jaunu klases 'KeyPair' objektu.
7.2.	KP_ SINGLE_ CERT KP_ UPDATE_ CERT KP_ EXPIRATION _DATE	Lasām atslēgu pāra derīguma termiņu, Atjauno atslēgu pāri un atkal lasām derīguma termiņu. Jaunajam derīguma termiņam jābūt vēlākam.	**<Izpilda 2.1. testu>** A = KeyPair({ "domains": ["ziema.lv", "*.vasaras.lpp"], "ip": ["197.1.4.3"]}, "f10") A.expiration_date() A.renew() A.expiration_date()	Beigās otrajām derīguma termiņam jābūt vēlākam. Tāda veidā pārbaudām, ka atslēgu atjaunošana strādā korekti.
8.1.	KP_ SINGLE_ CERT KP_ DEFINE_ CONTENT	Metode lasa visus domēna vārdus un IP adreses, kas pieder norādītajam atslēgai. LOCAL_PATH – ceļš līdz webAppOS.	**<Izpilda 2.1. testu>** A = CertReading("LOCAL_PATH\webAppOS\dist\etc\minica\f10\ziema.lv") A.cert_read_domains_and_ips()	Atgriež vārdnīcu ar atslēgu 'domains' un vērtībām 'ziema.lv', '*vasaras.lpp', ar atslēgu 'ip' un vērtību '197.1.4.3.'

N.p. k.	Metodes identifikators	Pamatojums	Ievaddati	Sagaidāmais rezultāts
9.1.	KP_ SINGLE_ CERT KP_ DEFINE_ ISSUER	Pārbauda korektu izdevēja lasīšanu.	**<Izpilda 2.1. testu>** A = CertReading("LOCAL_PATH\webApp OS\dist\etc\minica\f10\zie ma.lv") A.issuer()	Atgriež izdevēja nosaukumu ‘minica’.

4.3.2. Mapju šifrēšanas moduļa testi

Tabulā 4.6. var apskatīt uzrakstītus testus mapju šifrēšanas modulim.

4.6. tabula. **Mapju šifrēšanas moduļa vienībtestēšana**

N.p. k.	Metodes identifikators	Pamatojums	Ievaddati	Sagaidāmais rezultāts
1.1.	DE_ INIT_ MOUNT	Metodei pašai ir jāizveido divas mapes un virtuālais datu nesējs mapē ‘mount-1’. Parole ‘secret’ nav tukša.	A = CryfsSecurity(‘LOCAL_PATH1\base-1’, ‘LOCAL_PATH2\mount-1’) A,initAndMount(‘secret’)	Izveido divas mapes ‘base-1’ un ‘mount-1’. Mape ‘mount-1’ kļūs par virtuālo datu nesēju, kur lietotājs varēs strādāt ar datiem. Atgriež ‘True’.
1.2.	DE_ INIT_ MOUNT	Metodei ir jāizveido ceļš no trim jaunām mapēm līdz pamatmapei.	A = CryfsSecurity(‘LOCAL_PATH1\new1\ new2\new3\base-1’, ‘LOCAL_PATH2\mount-1’) A,initAndMount(‘secret’)	Atgriež no programmas CryFS kļūdu, ka mape ‘new1’ neeksistē.

N.p. k.	Metodes identifikators	Pamatojums	Ievaddati	Sagaidāmais rezultāts
1.3.	DE_ INIT_ MOUNT	Parole ir tukša, Pārbaude, vai būs atgriezts attiecīgs paziņojums.	A = CryfsSecurity(‘LOCAL_PATH1\base-2’, ‘LOCAL_PATH2\mount-2’) A,initAndMount(‘’)	Atgriež kļūdu ar tekstu “Password cannot be empty”.
1.4.	DE_ INIT_ MOUNT	Pamatmape ir virtuāla datu nesēja mapes (MOUNT) apakšmape. Pārbaude uz attiecīgu kļūdu.	A = CryfsSecurity(‘LOCAL_PATH2\mount-2 \base-2’, ‘LOCAL_PATH2\mount-2’) A,initAndMount(‘secret’)	Atgriež kļūdu ar tekstu "<Directory> is not empty, cannot init CryFS"
2.1.	DE_ INIT_ MOUNT DE_ UNMOUNT	Pārbaude uz korektu datu šifrēšanu.	A = CryfsSecurity(‘LOCAL_PATH1\base-1’, ‘LOCAL_PATH2\mount-1’) A,initAndMount(‘secret’) A,unmount()	Mape ‘mount-1’ kļūs tukša, dati šifrēta veidā glabāsies ‘base-1’. Atgriež ‘True’.
2.2.	DE_ UNMOUNT	Mēģina šifrēt mapes, kuras neeksistē un iepriekš nav izveidotas.	A = CryfsSecurity(‘LOCAL_PATH1\ not_exist1\base’, ‘LOCAL_PATH2\ not-exist2\mount’) A,unmount()	Paziņo par kļūdu ar tekstu “<ErrorLine>, error with mountdir: <MountDirectory>”

N.p. k.	Metodes identifikators	Pamatojums	Ievaddati	Sagaidāmais rezultāts
3.1.	DE_INIT_ MOUNT DE_ UNMOUNT DE_MOUNT	Pārbaude, vai dati būs atšifrēti ar to pašu paroli, kura bija iepriekš noradīta. Izpildām DE_INIT_MOU NT, ievietojam failus mapē ‘mount-3’. Pēc tam šifrējam un atšifrējam mapi.	A = CryfsSecurity(‘LOCAL_PATH1\base-3’, ‘LOCAL_PATH2\mount-3’) A,initAndMount(‘secret’) A.unmount() A.mount(‘secret’)	Mapē ‘mount-3’ jābūt tiem pašiem failiem, kas bija ievietoti mapē ‘base-3’ pirms šifrēšanas (unmount). Atgriež ‘True’.
3.2.	DE_INIT_ MOUNT DE_ UNMOUNT DE_MOUNT	Mēģina atšifrēt datus ar citu paroli, kas nesakrīt ar pirmo noradīto.	A = CryfsSecurity(‘LOCAL_PATH1\base-3’, ‘LOCAL_PATH2\mount-3’) A,initAndMount(‘secret1’) A.unmount() A.mount(‘secret2’)	Atgriež kļūdas paziņojumu ar tekstu: "Error during mounting: <Python kļūda>".
3.3.	DE_INIT_ MOUNT DE_ UNMOUNT DE_MOUNT	Izveido un šifrē mapi. Pārbaude, vai metode atgriež kļūdu par to, ka nav iespējams atšifrēt datus pat ar pareizu paroli.	A = CryfsSecurity(‘LOCAL_PATH1\base-3’, ‘LOCAL_PATH2\mount-3’) A,initAndMount(‘secret’) A.unmount() **< Dzēš pamatmapes ‘base-3’ saturu. >** A.mount(‘secret’)	Atgriež kļūdu ar tekstu: "CryFS configuration file not found in basedir <BaseDirectory>"

4.4. Python moduļa integrācijas testēšana

Zemāk tika aprakstīti testpiemēri integrācijas testēšanai, lai pārbaudītu korektu tīmekļa procesora un webAppOS komunikāciju. 4.6. tabulā aprakstīti testi tika palaisti gan atsevišķi testēšanas vidēs (sk. 4.1. tabulu), gan kopā ar webAppOS sistēmu.

4.4.1. Integrācijas testi

Tabulā 4.7. var apskatīt testpiemērus integrācijas testēšanai.

4.7. tabula. **Integrācijas testēšana. Komunikācija starp webAppOS un tīmekļā procesoru**

N.p.k.	Metode	Pamatojums	Sagaidāmais rezultāts
1.	KP_DIFFERENT_CERT	Dažādu atslēgu pāru ģenerēšana. Ievada gan korektus, gan kļūdainus domēna vārdus.	Izveido mapi ‘firm-1’, kur glabāsies 6 mapes ar atsevišķiem atslēgu pāriem. Mapē ‘firm-1’ izveido root-sertifikātu. Atgriež ‘True’.
2.	KP_DIFFERENT_CERT	Python moduļa izsaukumā (WEBCALL) ir kļūda. Trūkst ‘webMemoryUrl’ lauka.	Atgriež kļūdu ar tekstu: 'Invalid input json format in WEBCALL: webMemoryUrl'
3.	KP_DIFFERENT_CERT	Saglabājot konstruktora datus tīmekļa atmiņā, procesors nesaņem korektu datu ierakstīšanas statusu no tīmekļa atmiņas.	Atgriež kļūdu: 'web processor communication error'.
4.	KP_DIFFERENT_CERT	Nav noradīts ceļš (‘searchPaths’) failam, kur jāglabājas klases ‘MinicaSecurity’ metodei.	Python adapterī notiks kļūda, būs atgriezts paziņojums: "Error during file import in the adapter: No module named 'miniCA_python'"

N.p.k.	Metode	Pamatojums	Sagaidāmais rezultāts
5.	KP_DIFFERENT_CERT	Atslēgu pāru ģenerēšanai nav padots uzņēmuma nosaukums.	Atgriež kļūdu: "Error during file import in the adapter: The argument 'caName' was not passed to the function"}
6.	KP_DIFFERENT_CERT	Noradītajā failā nevar atrast klasi. Tika padota klase ‘MinicaSekurity’ nevis ‘MinicaSecurity’.	Atgriež kļūdu ar tekstu: "Error during file import in the adapter: <Python kļūda>"}
7.	KP_DIFFERENT_CERT	Metodē ir drukas kļūda ‘generate_key_dif’. Pareizi būtu ‘generate_key_pair_dif’	Atgriež python interpretatora kļūdu. Noradītajā klasē tādas metodes neeksistē.
8.	KP_SINGLE_CERT	Viena atslēgu pāra ģenerēšana. Domēna vārdos ir arī kļūdas.	Izveido mapi ‘firm-2’ ar apakšmapi ‘sun.lv’ ar vienu atslēgu pāri. Mapē ‘firm-2’ izveido root-sertifikātu. Tā kā webAppOS nevar atgriezt ‘KeyPair’ objektu, klases atribūti būs saglabāti tīmekļa atmiņā un būs atgriezta norāde uz atmiņu.
9.	KP_SINGLE_CERT	Klases atribūtu secība nesakrīt ar saņemto no webAppOS atribūtu secību. Vispirms Python modulis saņem uzņēmuma nosaukumu un tikai pēc tam domēna vārdus ar IP adreses.	Neatkarībā no aizsūtītas atribūtu secības, rezultātam jābūt tādam pašam kā arī 8. testā.

N.p.k.	Metode	Pamatojums	Sagaidāmais rezultāts
10.	KP_SINGLE_CERT KP_READ_PUBLIC_NAME	Iepriekš palaiž 8. testu. Lasa publiskas atslēgas atrašanas vietu, kas atbilst tieši vienam iepriekš uzģenerētam atslēgu pārim.	Atgriež publiskas atslēgas atrašanas vietu.
11.	KP_SINGLE_CERT KP_READ_PRIVATE_NAME	Iepriekš palaiž 8. testu. Lasa privātas atslēgas atrašanas vietu, kas atbilst tieši vienam iepriekš uzģenerētam atslēgu pārim.	Atgriež privātas atslēgas atrašanas vietu.
12.	KP_SINGLE_CERT KP_READ_PUBLIC_FILE	Iepriekš palaiž 8. testu. Lasa publiskas atslēgas saturu.	Atgriež publiskas atslēgas tekstu UTF-8 kodējumā.
13.	KP_SINGLE_CERT KP_EXPIRATION_DATE	Iepriekš palaiž 8. testu. Lasa ieveidotā atslēgu pāra derīguma termiņu.	Atgriež iepriekš izveidota atslēgu pāra derīguma termiņu formātā: ‘yyy-mm-ddThh:mm:ss’.
14.	KP_SINGLE_CERT KP_EXPIRATION_DATE KP_UPDATE_CERT	Palaiž 8. testu. Lasām atslēgu pāra derīguma termiņu, Atjauno atslēgu pāri un atkal lasām derīguma termiņu.	Otrajam derīguma termiņam jābūt vēlākam, kas informēs par veiksmīgu atslēgu pāra atjaunošanu.
15.	KP_SINGLE_CERT KP_DEFINE_CONTENT	Palaiž 8. testu. Pārbaude uz domēna vārdu un IP adrešu definēšanu, kas bija ierakstīti 8. testā.	Atgriež domēna vārdu un IP adrešu kopu: 'sun.lv', 'kasa3ni.tree.lv', 'sunUMuiza.ly', '192.18.0.34', '192.18.1.112', '87.2.0.12', kas atbilst vienam atslēgu pārim.

N.p.k.	Metode	Pamatojums	Sagaidāmais rezultāts
16.	KP_SINGLE_CERT KP_DEFINE_ISSUER	Palaiž 8. testu. Pārbaude uz korektu izdevēja lasīšanu.	Atgriež izdevēju ‘minica’
17.	DE_INIT_MOUNT	Izveido divas jaunas mapes turpmākai šifrēšanai.	Mapē ‘base-5’ glabājas dati, kas atjaunojas reālajā laikā. Mape ‘mount-5’ kļūst par virtuālo datu nesēju, kur var strādāt ar datiem. Atgriež ‘True’.
18.	DE_INIT_MOUNT DE_UNMOUNT DE_MOUNT	Pārbaude, vai dati būs atšifrēti ar to pašu paroli, kura bija iepriekš noradīta. Vispirms izveidojām virtuālo datu nesēju, pēc tam šifrējam un atšifrējam mapi.	Beigās mapē ‘mount-5’ būs atšifrēta iepriekš izveidota mape ‘MySecrets’. Atgriež ‘True’ gan pēc datu šifrēšanas, gan pēc atšifrēšanas.

4.4.2. Ievaddati konstruktora izveidošanai

Tabulās 4.8., 4.9., 4.10., 4.11. var apskatīt komunikāciju starp webAppOS un Python moduli, attiecīgi, MinicaSecurity, KeyPair, CertReading un CryfsSecurity klašu konstruktoru izveidošanai un datu saglabāšanai tīmekļa atmiņā. Tabulas ir jālasa no kreisas puses uz labo.

4.8. tabula. **MinicaSecurity konstruktora izveidošana pirms metodes izsaukuma**

Ievaddati no webAppOS	Sagaidāmā atbilde no Python moduļa
WEBCALL {"webMemoryUrl": "9999","objectReference": 123,"className": "MinicaSecurity","methodName": "","methodUrl": "python3:miniCA_python.py#MinicaSecurity", "arguments": {"dictionary": {"domains": ["sun.lv","kasa3ni.tree.lv","*.#ni.lv","sun.ly.", "sunUMuiza.ly"],"ip": ["192.18.0.34","192.18.1.112","87.2.0.12"]},"caName": "firm-1"},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}}	**WEB_MEMORY_WRITE** {"webMemoryUrl": "9999", "className": "MinicaSecurity", "state": "{'dictionary': {'domains': ['sun.lv', 'kasa3ni.tree.lv', '*.#ni.lv', 'sun.ly.', 'sunUMuiza.ly'], 'ip': ['192.18.0.34', '192.18.1.112', '87.2.0.12']}, 'caName': 'firm-1'}"}
WEB_MEMORY_WRITE_RESULT {"result":124}	**WEBCALL_RESULT** {"result":124}

4.9. tabula. **KeyPair konstruktora izveidošana pirms metodes izsaukuma**

Ievaddati no webAppOS	Sagaidāmā atbilde no Python moduļa
WEBCALL {"webMemoryUrl": "9999","objectReference": 123,"className": "KeyPair","methodName": "","methodUrl": "python3:miniCA_python.py#KeyPair","arguments": {"domains_and_ips": {"domains": ["sun.lv","kasa3ni.tree.lv", "sunUMuiza.ly"],"ip": ["192.18.0.34","192.18.1.112","87.2.0.12"]},"caName": "firm-2"},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}}	**WEB_MEMORY_WRITE** {"webMemoryUrl": "9999", "className": "KeyPair", "state": {"domains_and_ips": {"domains": ["sun.lv","kasa3ni.tree.lv", "sunUMuiza.ly"],"ip": ["192.18.0.34","192.18.1.112","87.2.0.12"]} ,"caName": "firm-2"}}

Ievaddati no webAppOS	Sagaidāmā atbilde no Python moduļa
WEB_MEMORY_WRITE_RESULT {"result":130}	**WEBCALL_RESULT** {"result":130}

4.10. tabula. **CertReading konstruktora izveidošana pirms metodes izsaukuma**

Ievaddati no webAppOS	Sagaidāmā atbilde no Python moduļa
WEBCALL {"webMemoryUrl": "9999","objectReference": 123,"className": "CertReading","methodName": "","methodUrl": "python3:miniCA_python.py#CertReading","arguments": {"PATH":"C:\\Users\\iljar\\webAppOS\\dist\\etc\\minica\\firm-2\\sun.lv\\cert.pem"},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}} **WEB_MEMORY_WRITE_RESULT** {"result":150}	**WEB_MEMORY_WRITE** {"webMemoryUrl": "9999", "className": "CertReading", "state": "{'PATH': 'C:\\\\Users\\\\iljar\\\\webAppOS\\\\dist\\\\etc\\\\minica\\\\firm-2\\\\sun.lv\\\\cert.pem'}"} **WEBCALL_RESULT** {"result":150}

4.11. tabula. **CryfsSecurity konstruktora izveidošana pirms metodes izsaukuma**

Ievaddati no webAppOS	Sagaidāmā atbilde no Python moduļa
WEBCALL {"webMemoryUrl": "9999","objectReference": 123,"className": "CryfsSecurity","methodName": "","methodUrl": "python3:CryfsSecurity.py#CryfsSecurity","arguments": {"basedir": " C:\\Users\\iljar\\CryFS\\base-5","mountdir": "	**WEB_MEMORY_WRITE** {"webMemoryUrl": "9999", "className": "CryfsSecurity", "state": "{'basedir': ' C:\\\\Users\\\\iljar\\\\CryFS\\\\base-5', 'mountdir': ' C:\\\\Users\\\\iljar\\\\CryFS\\\\mount-5 '}"}

Ievaddati no webAppOS	Sagaidāmā atbilde no Python moduļa
C:\\Users\\iljar\\CryFS\\mount-5 "},"context": {"searchPaths": ["C:\\Users\\iljar\\SelfSignedCA.weblibrary\\src\\main\\python\\CryFS"]}} **WEB_MEMORY_WRITE_RESULT** {"result":400}	 **WEBCALL_RESULT** {"result":400}

4.4.2. Ievaddati testu palaišanai

Ievaddatus testpiemēriem, kas bija aprakstīti 4.7. tabulā, var redzēt 4.12. tabulā. Tabula ir jālasa no kreisās puses uz labo pusi. Piemēram, 1. testā webAppOS aizsūta WEBCALL, bet pēc tam Python modulis atbild ar WEB_MEMORY_READ.

4.12. tabula. **Integrācijas testēšanas ievaddati**

N.p.k.	Ievaddati no webAppOS (input)	Aizsūtītie dati no Python moduļa (output)
1.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "MinicaSecurity","methodName": "generate_key_pair_dif","methodUrl": "python3:miniCA_python.py#MinicaSecurity.generate_key_pair_dif","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}} **WEB_MEMORY_READ_RESULT** {"result":{"dictionary": {"domains": ["sun.lv","kasa3ni.tree.lv","*.#ni.lv","sun.ly.", "sunUMuiza.ly"],"ip": ["192.18.0.34","192.18.1.112","87.2.0.12"]},"caName": "firm-1"}}	**WEB_MEMORY_READ** {"webMemoryUrl": "9999", "objectReference": "123"} **WEBCALL_RESULT** {‘result’: True}

N.p.k.	Ievaddati no webAppOS (input)	Aizsūtītie dati no Python moduļa (output)
2.	**WEBCALL** {"objectReference": 123, "className": "MinicaSecurity","methodName": "generate_key_pair_dif","methodUrl": "python3:miniCA_python.py#MinicaSecurity.generate_key_pair_dif","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}}	**WEBCALL_RESULT** {‘error’: ‘Invalid input json format in WEBCALL: webMemoryUrl’}
3.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "MinicaSecurity","methodName": "generate_key_pair_dif","methodUrl": "python3:miniCA_python.py#MinicaSecurity.generate_key_pair_dif","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}} **SOMETHING_ELSE**	**WEB_MEMORY_READ** {“webMemoryUrl”: “9999”, “objectReference”: “123”} **WEBCALL_RESULT** {‘error’: ‘web processor communication error’}
4.	**WEBCALL** {“webMemoryUrl”: “9999”,”objectReference”: 123,”className”: “MinicaSecurity”,”methodName”: “generate_key_pair_dif”,”methodUrl”: “python3:miniCA_python.py#MinicaSecurity.generate_key_pair_dif”,”arguments”: {},”context”: {}}	**WEB_MEMORY_READ** {“webMemoryUrl”: “9999”, “objectReference”: “123”}

N.p.k.	Ievaddati no webAppOS (input)	Aizsūtītie dati no Python moduļa (output)
	WEB_MEMORY_READ_RESULT {"result":{"dictionary": {"domains": ["sun.lv","kasa3ni.tree.lv","*.#ni.lv","sun.ly.", "sunUMuiza.ly"],"ip": ["192.18.0.34","192.18.1.112","87.2.0.12"]},"caName": "firm-1"}}	**WEBCALL_RESULT** {‘error’: “Error during file import in the adapter: No module named ‘miniCA_python’”}
5.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "MinicaSecurity","methodName": "generate_key_pair_dif","methodUrl": "python3:miniCA_python.py#MinicaSecurity.generate_key_pair_dif","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}} **WEB_MEMORY_READ_RESULT** {“result”:{“dictionary”: {“domains”: [“sun.lv”,”kasa3ni.tree.lv”,”*.#ni.lv”,”sun.ly.”, “sunUMuiza.ly”],”ip”: [“192.18.0.34”,”192.18.1.112”,”87.2.0.12”]}}}	**WEB_MEMORY_READ** {“webMemoryUrl”: “9999”, “objectReference”: “123”} **WEBCALL_RESULT** {‘error’: “Error during file import in the adapter: The argument ‘caName’ was not passed to the function”}
6.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "MinicaSekurity","methodName": "generate_key_pair_dif","methodUrl": "python3:miniCA_python.py#MinicaSekurity.generate_key_pair_dif","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}}	**WEB_MEMORY_READ** {“webMemoryUrl”: “9999”, “objectReference”: “123”}

N.p.k.	Ievaddati no webAppOS (input)	Aizsūtītie dati no Python moduļa (output)
	WEB_MEMORY_READ_RESULT {"result":{"dictionary": {"domains": ["sun.lv","kasa3ni.tree.lv","*.#ni.lv","sun.ly.", "sunUMuiza.ly"],"ip": ["192.18.0.34","192.18.1.112","87.2.0.12"]},"caName": "firm-1"}}	**WEBCALL_RESULT** {‘error’: “Error during file import in the adapter: cannot import name ‘MinicaSekurity’ from ‘miniCA_python’ (C:\\Users\\iljar\\webAppOS\\dist\\apps\\CA.weblibrary\\miniCA_python\\miniCA_python.py)”}
7.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "MinicaSecurity","methodName": "generate_key_pair_dif","methodUrl": "python3:miniCA_python.py#MinicaSecurity.generate_key_dif","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}} **WEB_MEMORY_READ_RESULT** {“result”:{“dictionary”: {“domains”: [“sun.lv”,”kasa3ni.tree.lv”,”*.#ni.lv”,”sun.ly.”, “sunUMuiza.ly”],”ip”: [“192.18.0.34”,”192.18.1.112”,”87.2.0.12”]},”caName”: “firm-1”}}	**WEB_MEMORY_READ** {“webMemoryUrl”: “9999”, “objectReference”: “123”} **WEBCALL_RESULT** {‘error’: “’MinicaSecurity’ object has no attribute ‘generate_key_dif’”}

N.p.k.	Ievaddati no webAppOS (input)	Aizsūtītie dati no Python moduļa (output)
8.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "MinicaSecurity","methodName": "generate_key_pair_s","methodUrl": "python3:miniCA_python.py#MinicaSecurity.generate_key_pair_s","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}} **WEB_MEMORY_READ_RESULT** {"result":{"dictionary": {"domains": ["sun.lv","kasa3ni.tree.lv","*.#ni.lv","sun.ly.", "sunUMuiza.ly"],"ip": ["192.18.0.34","192.18.1.112","87.2.0.12"]},"caName": "firm-2"}} **WEB_MEMORY_WRITE_RESULT** {"result":130}	**WEB_MEMORY_READ** {“webMemoryUrl”: “9999”, “objectReference”: “123”} **WEB_MEMORY_WRITE** {“webMemoryUrl”: “9999”, “className”: “KeyPair”, “state”: {“domains_and_ips”: {“domains”: [“sun.lv”, “kasa3ni.tree.lv”, “sunUMuiza.ly”], “ip”: [“192.18.0.34”, “192.18.1.112”, “87.2.0.12”]}, “caName”: “firm-2”}} **WEBCALL_RESULT** {‘result’: 130}

N.p.k.	Ievaddati no webAppOS (input)	Aizsūtītie dati no Python moduļa (output)
9.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "MinicaSecurity","methodName": "generate_key_pair_s","methodUrl": "python3:miniCA_python.py#MinicaSecurity.generate_key_pair_s","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}} **WEB_MEMORY_READ_RESULT** {"result":{"caName": "firm-2","dictionary": {"domains": ["sun.lv","kasa3ni.tree.lv","*.#ni.lv","sun.ly.", "sunUMuiza.ly"],"ip": ["192.18.0.34","192.18.1.112","87.2.0.12"]}}} **WEB_MEMORY_WRITE_RESULT** {"result":130}	**WEB_MEMORY_READ** {“webMemoryUrl”: “9999”, “objectReference”: “123”} **WEB_MEMORY_WRITE** {“webMemoryUrl”: “9999”, “className”: “KeyPair”, “state”: {“domains_and_ips”: {“domains”: [“sun.lv”, “kasa3ni.tree.lv”, “sunUMuiza.ly”], “ip”: [“192.18.0.34”, “192.18.1.112”, “87.2.0.12”]}, “caName”: “firm-2”}} **WEBCALL_RESULT** {‘result’: 130}
10.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "KeyPair","methodName": "public_key_file_name","methodUrl": "python3:miniCA_python.py#KeyPair.public_key_file_name","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}}	**WEB_MEMORY_READ** {“webMemoryUrl”: “9999”, “objectReference”: “123”}

N.p.k.	Ievaddati no webAppOS (input)	Aizsūtītie dati no Python moduļa (output)
	WEB_MEMORY_READ_RESULT {"result":{"domains_and_ips": {"domains": ["sun.lv","kasa3ni.tree.lv", "sunUMuiza.ly"],"ip": ["192.18.0.34","192.18.1.112","87.2.0.12"]},"caName": "firm-2"}}	**WEBCALL_RESULT** {'result': 'C:\\Users\\iljar\\webAppOS\\dist\\etc\\minica\\firm-2\\sun.lv\\cert.pem'}
11.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "KeyPair","methodName": "private_key_file_name","methodUrl": "python3:miniCA_python.py#KeyPair.private_key_file_name","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}} **WEB_MEMORY_READ_RESULT** {"result":{"domains_and_ips": {"domains": ["sun.lv","kasa3ni.tree.lv", "sunUMuiza.ly"],"ip": ["192.18.0.34","192.18.1.112","87.2.0.12"]},"caName": "firm-2"}}	**WEB_MEMORY_READ** {"webMemoryUrl": "9999", "objectReference": "123"} **WEBCALL_RESULT** {'result': 'C:\\Users\\iljar\\webAppOS\\dist\\etc\\minica\\firm-2\\sun.lv\\key.pem'}
12.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "KeyPair","methodName": "public_key_string","methodUrl": "python3:miniCA_python.py#KeyPair.public_key_string","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}}	**WEB_MEMORY_READ** {"webMemoryUrl": "9999", "objectReference": "123"}

N.p.k.	Ievaddati no webAppOS (input)	Aizsūtītie dati no Python moduļa (output)
	WEB_MEMORY_READ_RESULT {"result":{"domains_and_ips": {"domains": ["sun.lv","kasa3ni.tree.lv", "sunUMuiza.ly"],"ip": ["192.18.0.34","192.18.1.112","87.2.0.12"]},"caName": "firm-2"}}	**WEBCALL_RESULT** {‘result’: ‘<PUBLIC_KEY>’}
13.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "KeyPair","methodName": "expiration_date","methodUrl": "python3:miniCA_python.py#KeyPair.expiration_date","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}} **WEB_MEMORY_READ_RESULT** {"result":{"domains_and_ips": {"domains": ["sun.lv","kasa3ni.tree.lv", "sunUMuiza.ly"],"ip": ["192.18.0.34","192.18.1.112","87.2.0.12"]},"caName": "firm-2"}}	**WEB_MEMORY_READ** {“webMemoryUrl”: “9999”, “objectReference”: “123”} **WEBCALL_RESULT** {‘result': '<yyy-mm-ddThh:mm:ss>'}
14.	<13. tests> **WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "KeyPair","methodName": "renew","methodUrl": "python3:miniCA_python.py#KeyPair.renew","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}}	**WEB_MEMORY_READ** {"webMemoryUrl": "9999", "objectReference": "123"}

N.p.k.	Ievaddati no webAppOS (input)	Aizsūtītie dati no Python moduļa (output)
	WEB_MEMORY_READ_RESULT {"result":{"domains_and_ips": {"domains": ["sun.lv","kasa3ni.tree.lv", "sunUMuiza.ly"],"ip": ["192.18.0.34","192.18.1.112","87.2.0.12"]},"caName": "firm-2"}}	**WEB_MEMORY_WRITE** {"webMemoryUrl": "9999", "className": "KeyPair", "state": {"domains_and_ips": {"domains": ["sun.lv", "kasa3ni.tree.lv", "sunUMuiza.ly"], "ip": ["192.18.0.34", "192.18.1.112", "87.2.0.12"]}, "caName": "firm-2"}}
	WEB_MEMORY_WRITE_RESULT {"result":300}	**WEBCALL_RESULT** {'result': 300}
	<13. tests>	
15.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "CertReading","methodName": "cert_read_domains_and_ips","methodUrl": "python3:miniCA_python.py#CertReading.cert_read_domains_and_ips","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}}	**WEB_MEMORY_READ** {"webMemoryUrl": "9999", "objectReference": "123"}
	WEB_MEMORY_READ_RESULT {"result":{"PATH": "C:\\Users\\iljar\\webAppOS\\dist\\etc\\minica\\firm-2\\sun.lv\\cert.pem"}}	**WEBCALL_RESULT** {'result': ['sun.lv', 'kasa3ni.tree.lv', 'sunUMuiza.ly', '192.18.0.34', '192.18.1.112', '87.2.0.12']}

N.p.k.	Ievaddati no webAppOS (input)	Aizsūtītie dati no Python moduļa (output)
16.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "CertReading","methodName": "issuer","methodUrl": "python3:miniCA_python.py#CertReading.issuer","arguments": {},"context": {"searchPaths": ["C:/Users/iljar/webAppOS/dist/apps/CA.weblibrary/miniCA_python"]}} **WEB_MEMORY_READ_RESULT** {"result":{"PATH": "C:\\Users\\iljar\\webAppOS\\dist\\etc\\minica\\firm-2\\sun.lv\\cert.pem"}}	**WEB_MEMORY_READ** {"webMemoryUrl": "9999", "objectReference": "123"} **WEBCALL_RESULT** {'result': 'minica root ca 2f60d8'}
17.	**WEBCALL** {"webMemoryUrl": "9999","objectReference": 123,"className": "CryfsSecurity","methodName": "initAndMount","methodUrl": "python3:CryfsSecurity.py#CryfsSecurity.initAndMount","arguments": {"password":"secret"},"context": {"searchPaths": ["C:\\Users\\iljar\\SelfSignedCA.weblibrary\\src\\main\\python\\CryFS"]}} **WEB_MEMORY_READ_RESULT** {"result":{"basedir": "C:\\Users\\iljar\\CryFS\\base-5","mountdir": "C:\\Users\\iljar\\CryFS\\mount-5"}}	**WEB_MEMORY_READ** {"webMemoryUrl": "9999", "objectReference": "123"} **WEBCALL_RESULT** {'result': True}

N.p.k.	Ievaddati no webAppOS (input)	Aizsūtītie dati no Python moduļa (output)
18.	<17. tests> <Izveido mapē ‘mount-5’ apakšmapi ‘MySecrets’>	
	WEBCALL {"webMemoryUrl": "9999","objectReference": 123,"className": "CryfsSecurity","methodName": "unmount","methodUrl": "python3:CryfsSecurity.py#CryfsSecurity.unmount","arguments": {},"context": {"searchPaths": ["C:\\Users\\iljar\\SelfSignedCA.weblibrary\\src\\main\\python\\CryFS"]}}	**WEB_MEMORY_READ** {"webMemoryUrl": "9999", "objectReference": "123"}
	WEB_MEMORY_READ_RESULT {"result":{"basedir": "C:\\Users\\iljar\\CryFS\\base-5","mountdir": "C:\\Users\\iljar\\CryFS\\mount-5"}}	**WEBCALL_RESULT** {'result': True}
	WEBCALL {"webMemoryUrl": "9999","objectReference": 123,"className": "CryfsSecurity","methodName": "mount","methodUrl": "python3:CryfsSecurity.py#CryfsSecurity.mount","arguments": {"password":"secret"},"context": {"searchPaths": ["C:\\Users\\iljar\\SelfSignedCA.weblibrary\\src\\main\\python\\CryFS"]}}	**WEB_MEMORY_READ** {"webMemoryUrl": "9999", "objectReference": "123"}
	WEB_MEMORY_READ_RESULT {"result":{"basedir": "C:\\Users\\iljar\\CryFS\\base-5","mountdir": "C:\\Users\\iljar\\CryFS\\mount-5"}}	**WEBCALL_RESULT** {'result': True}

4.5. Veiktspējas testēšana

PPS ir nefunkcionālās prasības, kas ir testētas šajā nodaļā. Veiktspēja bija notestēta uz datora ar 2.3 GHz procesoru Windows 10 Edu vidē. Veiktspējas testēšanas laikā bija pieejami 4GB brīvpiekļuves atmiņas (RAM) no iespējamiem 8 GB. Rezultāts '+', nozīmē ka darbības laiks atbilst nefunkcionālajai prasībai (<1 min), '-', ka neatbilst.

4.5.1. Mapju šifrēšana

4.13. tabulā var redzēt mapju šifrēšanas laiku. Pirmais izpildes laiks atbilst metodei DE_INIT_MOUNT, otrais DE_MOUNT. Mērījumiem vairākas reizēs bija izmantots 18. tests, kas aprakstīts 4.12. tabulā.

4.13. tabula. **Mapju šifrēšanas veiktspējas testēšana**

N.p.k.	Datums	Izpildes laiks	Rezultāts
1.	05.01.2021	7min 12s 0min 16s	-
2.		0min 18s 0min 17s	+
3.		0min 15s 0min 16s	+
4.		0min 15s 0min 15s	+
5.		0min 16s 0min 17s	+
6.		0min 15s 0min 17s	+
7.		0min 18s 0min 17s	+
8.		0min 16s 0min 16s	+
9.		0min 15s 0min 16s	+
10.		0min 16s 0min 15s	+

Datu mediāna ir 0min 16s, līdz ar to laiks atbilst nefunkcionālai prasībai.

4.5.2. Koda ģenerēšana

Tabulā 4.14. var apskatīt veiktspējas testēšanas rezultātus. IDL faila lasīšana ar ne vairāk kā 1000 rindiņām jāveic 1 sekundes laikā, ja tas darīts ar procesoru ar vismaz 2GHz frekvenci. Katrs tests bija izpildīts 3 reizes.

4.14. tabula. **Koda ģenerēšanas veiktspējas testēšana**

N.p.k.	Datums	Rindiņu skaits	Izpildes laiks	Rezultāts
1.	05.01.2021	9	<0.5s	+
2.		108	<0.5s	+
3.		216	<0.5s	+
4.		396	<0.7s	+
5.		549	<0.8s	+
6.		702	<0.8s	+
7.		864	<0.9s	+
8.		972	<0.9s	+
9.		981	<1s	+
10.		1000	<1s	+

Ja rindiņu skaits nepārsniedz 1000 rindiņas, tad izpildes laiks nepārsniedz 1 sekundi. Var secināt, ka nefunkcionāla prasība ir izpildīta.

5. PROJEKTA ORGANIZĀCIJA

Projekts bija izstrādāts pēc spējas izstrādes dzīves cikla modeļa, jo paša sākumā nebija zināms Python moduļa integrācijas risinājums. Katru nedēļu bija organizētas sapulces ar kvalifikācijas darba vadītāju un autoru. Pēc katra jauna Python moduļa integrācijas risinājuma bija apspriesti tas plusi un mīnusi, kā arī realizācija praksē. Ne visi teorētiski atrisinājumi strādāja praksē, līdz ar to vajadzēja operatīvi mainīt Python moduļa izstrādes stratēģijas un uzlabot uzprogrammētu kodu. Programmēšana un testēšana notika viena laikā pēc katras moduļa metodes uzrakstīšanas, kā arī pēc katras sapulces bija precizētas programmatūras prasības. Pašu Python moduli izstrādāja tikai šī darba autors, savukārt Python moduļa integrācija webAppOS vidē notika kopā ar darba vadītāju.

6. DARBIETILPĪBAS NOVĒRTĒJUMS

Tā kā iepriekš webAppOS vidē nebija projektēts un integrēts līdzīgs modulis, pasūtītājam nebija pieejams darbietilpības novērtējums šāda tipa modulim. Rindiņu skaits kā pierādījums darbietilpības novērtējumam neder, jo rindiņu skaits augsta līmeņa programmēšanas valodām, tas ir, python, ir atkarīgs no sistēmas prasībām un specifikas. Līdz ar to darbietilpības novērtējums bija veikts ar funkcijpunktu novērtējumu. Pirms darbietilpības novērtējuma darba autors iepazinās ar funkcijpunktu novērtējuma metodi [13]. Tāda metode nav atkarīga no programmēšanas valodas, līdz ar to derēs arī Python valoda. Funkcijpunktu aprēķināšanas formulu var apskatīt 6.1. attēlā:

$$\mathrm{FP} = \mathrm{UFP} \cdot \mathrm{VAF}$$

$$VAF = \left[0.65 + 0.01 \cdot \sum(F)\right]$$

FP −funkcijpunktu skaits

UFP – visu prognozēšanas parametru summa

VAF – vērtības korekcijas koeficients

$0.65\,, 0.01$ −konstantes

F −vērtības korekcijas parametrs

6.1. att. **Funkcijpunktu aprēķināšanas formula**

Prognozēšanas parametru vērtības var apskatīt tabulā 6.1.

6.1. tabula **Prognozēšanas parametru saskaitīšana**

Prognozēšanas parametri	Skaits	Sarežģītības vērtība			Kopsumma
		Vienkāršs	Vidējs	Sarežģīts	
Ārējo ievadu skaits (EIs)	3	3	4	6	18
Ārējo izvadu skaits (EOs)	1	4	5	7	4
Ārējo vaicājumu skaits (EQs)	3	3	4	6	9
Ieksējo loģisko failu skaits (ILF)	2	7	10	15	14
Ārējo saskarņu failu skaits (EIF)	1	5	7	10	7

Prognozēšanas parametri	Skaits	Sarežģītības vērtība			Kopsumma
		Vienkāršs	Vidējs	Sarežģīts	
Prognozēšanas parametru summa:					52

Gaiši oranža krāsā ir iezīmēta izvēlēta sarežģītības vērtība. Funkcijpunktu aprēķināšanai bija izmantoti sekojoši prognozēšanas parametri:

- Ārējo ievadu skaits:
  1. jaunas metodes izsaukums (WEBCALL) pirms metodes palaišanas;
  2. metodes izsaukums no webAppOS (WEBCALL). SSL atslēgu pāru moduļa izmantošana;
  3. metodes izsaukums no webAppOS (WEBCALL). Mapju šifrēšanas moduļa izmantošana.
- Ārējo izvadu skaits:
  1. metodes izsaukuma rezultāta paziņošana (WEBCALL_RESULT).
- Ārējo vaicājumu skaits:
  1. ārējas funkcijas izsaukuma projektēšana (INNER_WEBCALL, to skaitā koda ģenerēšana izsaukumam);
  2. datu lasīšana no tīmekļa atmiņas (WEB_MEMORY_READ);
  3. datu ierakstīšana tīmekļa atmiņā (WEB_MEMORY_WRITE).
- Iekšējo loģisko failu skaits
  1. objektu izveidošanas fabrika (Factory), kas izveido python objektu kas iekšēji izmanto metodes datu lasīšanai/ ierakstīšanai tīmekļa atmīņā, kā arī ārējo funkciju izsaukuma metode (sk. 3.2.4.2. nodaļu);
  2. python adapteris uzprojektēto klašu metožu izsaukumiem;
- Ārējo saskarņu failu skaits (EIF):
  1. IDL fails uzģenerēta koda veidošanai. IDL faila pārveidošana python programmēšanas valodā.

Vērtības korekcijas koeficientu aprēķina, summējot visas parametru vērtības, kas ietekmē darbietilpību, novērtējot katru no 0 (nav svarīgs) līdz 5 (svarīgs). Vērtības korekcijas koeficienta aprēķināšanu var apskatīt 6.2. tabulā. Ir apskatīti 4 kritēriji jeb jautājumi, kas ietekmē vērtības korekcijas koeficientu.

6.2. tabula **Vērtības korekcijas koeficienta aprēķināšana**

N.p.k.	Jautājums	Svarīgums (no 0 līdz 5)	Pamatojums
1.	Vai kods ir paredzēts atkārtotai lietošanai?	4	Python modulis ir paredzēts atkārtotai lietošanai. Tīmekļa procesoram ir paredzēts pieslēgt arī citu programmēšanas valodu adapterus. Python adapteris ir paredzēts citu programmatūras izstrādātāju izveidotu moduļu pieslēgšanai webAppOS videi.
2.	Vai lietojumprogramma ir paredzēta, lai atvieglotu lietotāja funkciju lietošanu?	4	SSL atslēgu pāru modulis un mapju šifrēšanas modulis ir paredzēts darba automatizēšanai. Ar Python moduli gala lietotājam nav nepieciešams manuāli izsaukt python interpretatoru, kā arī lietojumprogrammas Minica un CryFS.
3.	Vai iekšēja apstrāde ir sarežģīta?	3	Mapju šifrēšanas modulī ir izmantota datu struktūra Queue (rinda), lai izveidotu komunikāciju starp izsaukto metodi un metodi CryFS palaišanai, kas sava starpā ir neatkarīgas viena no otras un atrodas dažādos failos.
4.	Vai ievades, izvades pieprasījumi ir sarežģīti?	3	Metodes izsaukuma dati (WEBCALL) satur 7 dažādas vērtības, kas atsevišķi ir jāapstrādā. Ārējas funkcijas izsaukuma dati (INNER_WEBCALL) satur 8 vērtības.
	Summa:	14	

Ievietojam visus skaitļus formulā un attēlā 6.2. apskatām rezultātu.

$$\mathrm{FP} = \mathrm{UFP} \cdot \mathrm{VAF}$$

$$\mathrm{FP} = 52 \cdot [0.65 + 0.01 \cdot 14] = 52 \cdot 0.79 = 41{,}08$$

6.2. att. **Funkcijpunktu skaits**

Rindiņu skaitu labāk nelietot, lai novērtētu projekta darbietilpību, par to arī stāsta programminženierijas metriku speciālists Capers Jones, jo tas lielā mērā ir atkarīgas no projektētām funkcijām. Tā kā darbietilpības novērtējums nevar būt ļoti precīzs skaitlis, kas nemainās projektēšanas laikā, var paņemt, ka 10.4 funkcijpunkti python programmēšanas valodā atbilst 1 personmēnesim [14]. Tad projekta darbietilpība ir apmēram *3,9 personmēneši*. Ja paņemt R.S.Pressman grāmatā minētus datus [13], tad 1 personmēnesis atbilst apmēram 12 funkcijpunktiem. Tad projekta darbietilpība ir apmēram *3,4 personmēneši*.

Papildinājumi:

Pēc darba pabeigšanas, var apgalvot, ka faktiskais patērētais laiks bija *3,6 personmēneši,* kas arī aptuveni sakrīt ar plānoto darbietilpības novērtējumu.

7. KVALITĀTES NODROŠINĀŠANA

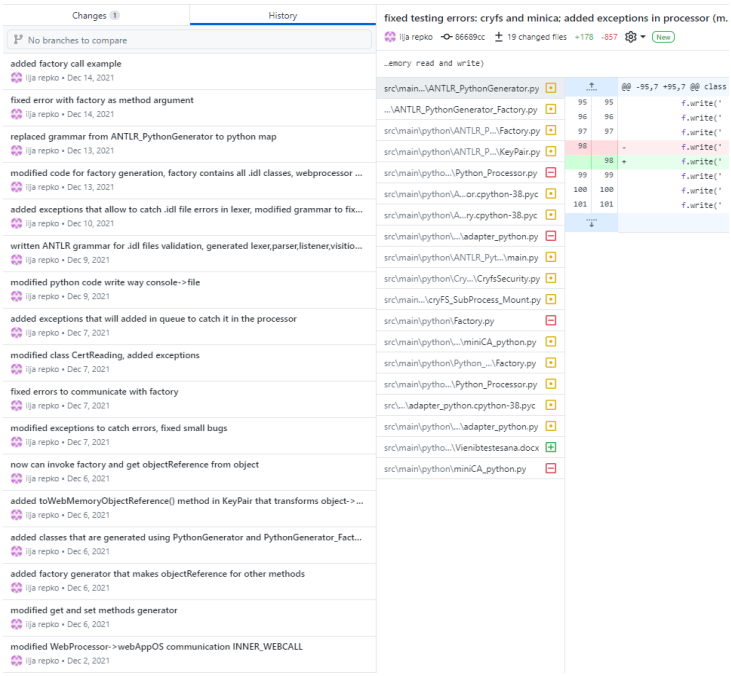
Python moduļa izstrādes laikā tiks nodrošināta kvalitāte, veicot sekojošas darbības:

1. Tika izmantota versiju kontroles sistēma Git, kas deva iespēju gan darba autoram, gan darba vadītājam ērti un ātri apskatīt jaunākas koda izmaiņas, kā arī kvalitatīvi nodrošināt koda izveidi;
2. jaunu metožu rakstīšana un testēšana notika paralēli, lai uzreiz saskatītu nepilnības un kļūdas;
3. programmatūras koda rakstīšanas laikā bija pievienoti vairāki komentāri, lai citiem programmatūras izstrādātajiem būtu viegli saprast metožu galvenās funkcijas;
4. regulāra pārbaude, ka programmatūra un dokumentācija atbilst specificētajām prasībām attiecībā pret Python moduli;
5. tika ievēroti PPS [1], PPA [3] un testēšanas dokumentācijas standarti [4], kā arī prasības noslēguma darbiem Latvijas Universitātē [2];
6. Python modulis bija izstrādāts tāda veidā, lai to varētu izmantot dažādās operētājsistēmās (Windows, Linux, MacOS);
7. tīmekļa procesors un python adapteris bija izveidots tāda veidā, lai tie nebūtu atkarīgi no izsaucamās klases metodes;
8. bija pārbaudīta lietojumprogrammu (Minica, CryFS) drošība pirms to lietošanas Python modulī (sk. 3.4.1. un 3.5.1. nodaļas);
9. dokumentācijas versijas automātiski bija saglabātas Latvijas Universitātes OneDrive serverī, kā arī regulāri bija veidotas rezerves kopijas darba autora datorā.

8. KONFIGURĀCIJU PĀRVALDĪBA

Python moduļa konfigurācijas pārvaldībai bija izvēlēta versiju kontroles rīks Git. Versiju kontroles rīks deva iespēju apskatīt iepriekšējas programmatūras koda versijas un vēsturi, kad un kas bija pievienots projektam. Versiju kontroles servisā GitHub bija izveidots privāts repozitorijs, kas palīdzēja gan darba autoram, gan vadītājam aplūkot izmaiņas projektā. Kopumā Python moduļa projekta moduļi bija atjaunoti un papildināti vairāk nekā 50 reizes. Attēlā 8.1. var apskatīt pirmās un pēdējās saglabātas koda versijas servisā GitHub.

Dokumentācijas versijas automātiski bija saglabātas Latvijas Universitātes OneDrive serverī. Darba autors arī saglabāja dokumentācijas rezerves kopijas sava datorā. Pirmo un pēdējo dokumentācijas versiju datumus no OneDrive var apskatīt 8.2. attēlā.



8.1. att. **Git versiju vēsture**

352.0	⋮	06.01.2022 15:15	Ilja Repko	2,60 MB
351.0	⋮	06.01.2022 13:07	Ilja Repko	2,60 MB
3.0	⋮	14.09.2021 16:41	Ilja Repko	38,3 KB
2.0	⋮	14.09.2021 16:14	Ilja Repko	37,4 KB
1.0	⋮	14.09.2021 16:11	Ilja Repko	37,6 KB

8.2. att. **Dokumentācijas versijas**

SECINĀJUMI

Darba rezultātā, platforma WebAppOS tika papildināta ar divām svarīgām iespējām:

1. iespēja ģenerēt ciparsertifikātus, kurus lokāli paraksta ar lokālās izdevējiestādes atslēgu;
2. iespēja šifrēt mapes un piekļūt to atšifrētam saturam reālā laikā.

Šīs iespējas padarīs WebAppOS par vairāk pievilcīgāku izmantošanai vidēs, kur datu šifrēšana ir kritiski svarīga. Piedāvātie python procesors un adapteris ļaus izstrādātājiem veidot WebAppOS-bāzētās tīmekļa lietotnes, kurām nepieciešams daļu no funkcionalitātes realizēt valodā Python.

Uz Python procesora piemēra tika aprobēts jaunais komunikācijas protokols, ko WebAppOS lietos saziņai arī ar citiem procesoriem, lai nodrošinātu iespēju izmantot citas programmēšanas valodas un ietvarus servera pusē, piemēram, Go, Node.js/Deno, PHP, u.c.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] Latvijas Valsts Standarts LVS 68:1996 “Programmatūras prasību specifikācijas ceļvedis.”

[2] Latvijas Universitātes rīkojums Nr. 1/38 “Prasības noslēguma darbu (bakalaura, maģistra darbu, diplomdarbu un kvalifikācijas darbu) izstrādāšanai un aizstāvēšanai Latvijas Universitātē.”

[3] Latvijas Valsts Standarts LVS 72:1996 „Ieteicamā prakse programmatūras projektējuma aprakstīšanai”.

[4] Latvijas Valsts Standarts LVS 70:1996 „Programmatūras testēšanas dokumentācija”.

[5] webAppOS API & Docs Source Documentation [tiešsaiste] – [atsauce 10.01.2022.]. Pieejams: https://webappos.org/dev/doc/#File:API_Specifications/Server-Side_APIs/API.java:Server-Side_API

[6] webAppOS izmantošanas ceļvedis [tiešsaiste] – [atsauce 10.01.2022.].

Pieejams: https://webappos.org/install/

[7] Tīmekļa procesora un webAppOS komunikācijas protokola izveidošana [tiešsaiste] – [atsauce 10.01.2022.].

Pieejams: https://www.notion.so/Web-Processor-webAppOS-Communication-Protocol-via-OS-pipes-ef7b961bd6344907870f811706d83096

[8] Minica lietojumprogrammas dokumentācija un pirmkods [tiešsaiste] – [atsauce 10.01.2022.].

Pieejams: https://github.com/jsha/minica

[9] Let’s Encrypt: Ciparsertifikāti lokālajos datortīklos [tiešsaiste] – [atsauce 10.01.2022.].

Pieejams: https://letsencrypt.org/docs/certificates-for-localhost/

[10] CryFS lietojumprogramma [tiešsaiste] – [atsauce 10.01.2022.].

Pieejams: https://www.cryfs.org/#download

[11] CryFS lietojumprogrammas izmantošanas ceļvedis [tiešsaiste] – [atsauce 10.01.2022.].

Pieejams: https://www.cryfs.org/tutorial

[12] CryFS lietojumprogrammas pirmkods [tiešsaiste] – [atsauce 10.01.2022.].

Pieejams: https://github.com/cryfs/cryfs

[13] R.S.Pressman Software Engineering - A Practitioner Approach, 7th ed. (620.lpp-623.lpp)

[14] Jones, C. Software Economics and Function Point Metrics (48.lpp – 52.lpp) [tiešsaiste] – [atsauce 10.01.2022.].

Pieejams: https://www.ifpug.org/wp-content/uploads/2017/04/IYSM.-Thirty-years-of-IFPUG.-Software-Economics-and-Function-Point-Metrics-Capers-Jones.pdf

PIELIKUMI

1. Pielikums

Factory koda ģenerēšana

```
class Factory:
    def PrintPythonFactory(self,factory): #factory - dictionary of {class:atributes}
        s = 'open("Factory.py","w")' #create code for input class
        f = eval(s)
        f.write('import json\n')
        f.write('class Factory:\n')
        f.write('  def __init__(self,webMemoryUrl):\n')
        f.write('    self.webMemoryUrl = webMemoryUrl\n')
        f.write('\n\n')

        for className in factory:
            atributes = factory[className]
            methodName = "new"+className
            s = '  def '+methodName+'(self'
            for i in atributes:
                s+= ',' +i
            s+= '):\n'
            f.write(s)

            s = '    from '+className+' import '+className + '\n'
            f.write(s)
            f.write('    print("WEB_MEMORY_WRITE")\n')
            f.write('    MemoryWriteDictionary = {\n')
            f.write('      "webMemoryUrl" : self.webMemoryUrl,\n')
            f.write('       "className" : "'+className+'",\n')
            f.write('       "state" : {\n')
            length = len(atributes)
            for i in atributes:
                if i != atributes[length-1]:
                    f.write('         "'+i+'" : '+i+',\n')
                else:
                    f.write('         "'+i+'" : '+i+'\n')
            f.write('       }\n')
            f.write('    }\n\n')

            f.write('    try:\n')
            f.write('      MemoryWriteJson = json.dumps(MemoryWriteDictionary)\n')
            f.write('      print(MemoryWriteJson)\n')
            f.write('    except TypeError as T:\n')
            f.write('      text = "Unable to serialize the WEB_MEMORY_WRITE input object to json T.args[0]"\n')
            f.write('      raise Exception(text)\n\n')

            f.write('    s=input()\n')
            f.write('    if(s=="WEB_MEMORY_WRITE_RESULT"):\n')
            f.write('      try:\n')
            f.write('        ResultJson=input()\n') #return result or error from webAppOS
            f.write('        ResultDic = json.loads(ResultJson)\n')
            f.write('      except ValueError as V:\n')
            f.write('        text = "Unable to load WEB_MEMORY_WRITE_RESULT output from webMemory V.args[0]"\n')
            f.write('        raise Exception(text)\n\n')
            f.write('      if("error" in ResultDic):\n')
            f.write('        raise Exception(ResultDic["error"])\n')
            f.write('      else:\n')
            f.write('        return '+className+'(self.webMemoryUrl,ResultDic["result"])\n\n')
            f.write('    else:\n')
            f.write('      text = "web processor communication error"\n')
            f.write('      raise Exception(text)\n')
            f.write('\n')
```

2. Pielikums

Python adapteris. Python objekta izveidošana

```
def CreatePythonObject(self,fileName,className,savedArgsForConstructor):
    try:
        s = "import "+fileName #to know PATH with __file__
        exec(s)
        s = "from "+fileName+" import "+className
        exec(s)
        s = fileName+".__file__" #PATH of the fileName. Like C:/../miniCA_python.py

        s = "inspect.getfullargspec("+className+"."+"__init__)[0]"
        args = eval(s)
        Arguments = self.__WriteMethodArgs(args,savedArgsForConstructor) #find given arg for a specific constructors arg

        s = className +"("+Arguments+")"
        obj = eval(s) #create object!
        return obj
    except Exception as E:  #invalid fileName or className, missed arguments
        text = "Error during file import in the adapter: "+E.args[0]
        raise Exception(text)
```

3. Pielikums

SSL atslēgu pāru modulis.

Publiskas atslēgas faila lasīšana un derīguma termiņa lasīšana

```
def public_key_string(self): #Return public key as string for KeyPair
    cert = ""
    f = None
    try:
        PATH_file = self.__Path_to_KeyPair() #else except Exception that has been raised
        PATH_file = os.path.abspath(os.path.join(PATH_file,"cert.pem"))

        f = open(PATH_file, 'r') #try open public cert file ->else not accesible
        lines = f.readlines()
        l = len(lines)
        for j in lines[1:l-1]: #read line by line
            cert += j.rstrip()
        return cert
    except FileNotFoundError as F: #if cert.pem cannot be found
        text = 'cert.pem file in "'+PATH_file+'" is not accessible for reading'
        raise Exception(text)
    finally:
        if(f is not None):
            f.close()

def expiration_date(self): #read exp date of the KeyPair
    try:
        PATH_file = self.__Path_to_KeyPair() #else except Exception that has been raised
        PATH_file = os.path.abspath(os.path.join(PATH_file,"cert.pem"))
        with open(PATH_file,"r") as f:
            cert_buf = f.read()

        from OpenSSL import crypto as c #cryptography module
        cert = c.load_certificate(c.FILETYPE_PEM,cert_buf)
        date_format = "%Y%m%d%H%M%SZ" #default date_format
        encoding = "ascii"
        Date = datetime.strptime(cert.get_notAfter().decode(encoding), date_format) #to know exp.dates
        Exp_Date = Date.strftime("%Y-%m-%dT%H:%M:%SZ") #need such date format
        return Exp_Date
    except FileNotFoundError: #if cert.pem cannot be found
        text = 'Directory "'+PATH_file+'" is not accesible'
        raise Exception(text)
    except ImportError as I: #catch error from __Path_to_KeyPair()
        text = "cannot import crypto library to decode SSL cert.pem file "+I.args[0] #cause crypto should be installed prev.
        raise Exception(text)
    finally:
        if(f is not None):
            f.close()
```

4. Pielikums

Mapju šifrēšanas modulis. Mapes izveidošana turpmākai šifrēšanai

```
class CryfsSecurity:
    def __init__(self,basedir,mountdir):
        self.basedir = os.path.normpath(basedir) #dir for encrypted files
        self.mountdir = os.path.normpath(mountdir) #dir for decrypted files - user works in this dir

    def initAndMount(self,password):
        q = Queue() #after mounting, there will be running subprocess, that will keep Cryfs running and mountdir working
        t1 = Thread(target = self.initAndMountQueue, args =(q,password, ))
        t1.start()
        t1.join()
        Result = q.get()
        if "error" in Result:
            raise Exception(Result["error"]) #except error that has been catched
        else:
            return Result["result"] #return positive result

    def initAndMountQueue(self,q,password):
        dic = {}
        arr = [self.basedir,self.mountdir]
        try:
            for i in arr:
                if(os.path.isdir(i)): #if basedir/mountdir exist
                    if(os.listdir(i)): #if basedir/mountdir is NOT empty
                        text = i + " is not empty, cannot init CryFS"
                        raise Exception(text) #error cause they must be empty for init

                    elif os.path.ismount(i): #if there is mounted dir with the same name
                        text = i + " has already mounted, cannot run init with the mounted folder"
                        raise Exception(text)
                else:
                    os.mkdir(i) # create new dir, if it is not exist

            if not password: #is password is empty
                text = "Password cannot be empty"
                raise Exception(text)

            elif self.mountdir not in self.basedir: #if dir are not same & basedir not in mountdir
                t = Thread(target = cryfsMounting, args =(q, self.basedir,self.mountdir,password,)) #Thread for cryfs mounting
                t.start() #start subprocess program,cause we do not want the whole program to freeze in Windows OS
                dic = q.get()
                if "error" in dic: #if error in the another subprocess program
                    raise Exception(dic["error"])
                else:
                    Result = {"result":True} #convert to dictionary, to get it in initAndMount
                    q.put(Result) #return successful result
            else:
                text = "Base directory can't be inside the mount directory"
                raise Exception(text)
        except Exception as E:
            Error = {"error":E.args[0]} #convert to dictionary, to except error in initAndMount
            q.put(Error)
```

Kvalifikācijas darbs **„Python moduļa izstrāde un integrācija tīmekļa operētājsistēmā webAppOS drošības funkcionalitātes nodrošināšanai”** 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: ***Iļja Repko*** **_____ ___.01.2022**.

Rekomendēju darbu aizstāvēšanai

Darba vadītājs: ***Asoc.prof. Sergejs Kozlovičs*** **_____ ___.01.2022.**

Recenzents: ***Krists Jirgens***

Darbs iesniegts 10.01.2022.

Kvalifikācijas darbu pārbaudījumu komisijas sekretārs: *Viesturs Vēzis*

Darbs aizstāvēts kvalifikācijas darbu pārbaudījuma komisijas sēdē

___.01.2022. prot. Nr. _____

Komisijas sekretārs(-e): _____