A blue and white logo

Description automatically generated with medium confidence

“Liepājas Valsts tehnikums”

**Lietojumprogrammu veidošanas, testēšanas un izvietošanas rīku kopums.**

Kvalifikācijas eksāmena praktiskās daļas tehniskā dokumentācija

Izglītības programma **33484011 Programmēšana**

Profesionālā kvalifikācija **Programmēšanas tehniķis**

Projekta izstrādātājs Mārtiņš Bardulis

Eksāmena datums 2023. gada 21. jūnijs

Liepāja 2023

**Saturs**

[**Ievads** 3](#_Toc131076689)

[1. Uzdevuma formulējums 4](#_Toc131076690)

[2. Programmatūras prasību specifikācija 6](#_Toc131076691)

[2.1. Produkta perspektīva 7](#_Toc131076692)

[2.2. Sistēmas funkcionālās prasības 8](#_Toc131076693)

[2.3. Sistēmas nefunkcionālās prasības 9](#_Toc131076694)

[2.4. Gala lietotāja raksturiezīmes 10](#_Toc131076695)

[2.5. Lietoto terminu un saīsinājumu skaidrojumi 11](#_Toc131076696)

[3. Izstrādes līdzekļu, rīku apraksts un izvēles pamatojums 12](#_Toc131076697)

[3.1 Iespējamo (alternatīvo) risinājuma līdzekļu un valodu apraksts 13](#_Toc131076698)

[3.2 Izvēlēto risinājuma līdzekļu un valodu apraksts 14](#_Toc131076699)

[4. Sistēmas modelēšana un projektēšana 15](#_Toc131076700)

[4.1 Sistēmas struktūras modelis 16](#_Toc131076701)

[4.2. Klašu diagramma / ER diagramma 17](#_Toc131076702)

[4.3. Funkcionālais un dinamiskais sistēmas modelis 18](#_Toc131076703)

[4.4. Aktivitāšu diagramma (Activity) 19](#_Toc131076704)

[4.5. Lietojumgadījumu diagramma (Use Case) 20](#_Toc131076705)

[4.6. Sistēmas moduļu apraksts un algoritmu shēmas 21](#_Toc131076706)

[5. Lietotāju ceļvedis 22](#_Toc131076707)

[6. Testēšanas dokumentācija 23](#_Toc131076708)

[6.1. Izvēlētās testēšanas metodes, rīku apraksts un pamatojums 24](#_Toc131076709)

[6.2. Testpiemēru kopa 25](#_Toc131076710)

[6.3. Testēšanas žurnāls 26](#_Toc131076711)

[8. Secinājumi 27](#_Toc131076712)

[9. Lietoto saīsinājumu saraksts 28](#_Toc131076713)

[10. Literatūras un informācijas avotu saraksts 29](#_Toc131076714)

# **Ievads**

Mūsdienu ātrajā digitālajā vidē uzņēmumi tiecas nodrošināt ātru un efektīvu lietojumprogrammu piegādi. Mūsu rīku kolekcija automatizē lietojumprogrammu izvietošanu, testēšanu un veidošanu darbības plūsmā. Mūsu produkts optimizē procesus, palielina produktivitāti, nodrošina uzticamību un paātrina augstas kvalitātes lietojumprogrammu piegādi.

Manuālie lietojumprogrammu ievietošanas procesi var būt laikietilpīgi, pakļauti kļūdām un kavē produktivitāti. Šie rīki izslēdz manuālo iejaukšanos, automatizējot svarīgās darbības plūsmas posmus. Vai nu ievietojat jaunāko lietojumprogrammas versiju, veicot visaptverošus testus vai izveidojot optimizētas pakotnes, produkts ar to visu tiks galā bez problēmām.

Darbības plūsmas automatizācija nodrošina vairākas priekšrocības. Tā samazina ievietošanas laiku, ļaujot ātri reaģēt uz tirgus pieprasījumu un iegūt konkurences priekšrocības. Izslēdzot cilvēku kļūdas, tiek nodrošināta vienmērīga un uzticama lietojumprogrammu ievietošana. Mūsu daudzpusīgie rīki integrējas ar dažādām izstrādes vides, padarot tos piemērotus gan maziem, gan lieliem komandu darbiem.

Secinājumā mūsu rīku kolekcija revolucionē lietojumprogrammu ievietošanas procesu, automatizējot svarīgos darbības plūsmas posmus. Palielinot produktivitāti, uzticamību un ātrumu, mūsu produkts ļauj uzņēmumiem veiksmīgi darboties digitālajā vidē, koncentrējoties uz inovāciju un izaugsmi.

# 1. Uzdevuma formulējums

**Programmatūras produkta nepieciešamība:**

Šis produkts aizpilda nozīmīgu vajadzību nozarē, automatizējot lietojumprogrammu izvietošanas procesu. Izstrādāts manā DevOps praksē, šis rīku kopums risina manuālās izvietošanas laikietilpīgumu un kļūdainības problēmas. Šis risinājums samazina laiku, kas nepieciešams, lai lietojumprogramma nonāktu tirgū, izslēdz kļūdas un palielina produktivitāti. Tas bez problēmām integrējas ar dažādām izstrādes vidēm, piemērots gan mazu, gan lielu komandu darbiem.

**Sasniedzamais rezultāts:**

Mūsu produkta galīgais mērķis ir optimizēt lietojumprogrammu izvietošanu, nodrošinot efektīvu un uzticamu augstas kvalitātes lietojumprogrammu piegādi. Automatizējot būtiskos uzdevumus, mēs vēlamies paātrināt lietojumprogrammu izvietošanu, izslēgt kļūdas, palielināt produktivitāti un uzlabot lietojumprogrammu kvalitāti.

**Kā tiks sasniegts:**

Galīgais mērķis - optimizēt lietojumprogrammu ievietošanu un sasniegt efektīvu un uzticamu augstas kvalitātes lietojumprogrammu piegādi - tiks sasniegts, veicot šādus soļus:

1. Automatizācija: Automatizējot būtiskos uzdevumus darbības plūsmā, piemēram, izvietošanu, testēšanu un izveidi, tiek izslēgta manuāla iejaukšanās, samazinot kļūdu risku un nodrošinot vienmērīgu un uzticamu lietojumprogrammu ievietošanu.

2. Optimizācija: Produkts koncentrējas uz lietojumprogrammu izstrādes darba plūsmas optimizēšanu, automatizējot laikietilpīgus uzdevumus, palielinot komandas produktivitāti un taupot vērtīgo laiku un resursus.

3. Testēšana un kvalitātes nodrošināšana: Ieviesti visaptveroši testēšanas procesi, kas ļauj identificēt un novērst problēmas, nodrošinot augstas kvalitātes lietojumprogrammu piegādi, kas atbilst noteiktajiem standartiem un lietotāju ekspektācijām.

4. Elastība un pielāgojamība: Produkts ir izstrādāts, lai atbalstītu dažādas izstrādes vides un pielāgotos dažādu lielumu komandām, nodrošinot elastību un pielāgojamību.

**Kā konstatēts, ka mērķis ir sasniegts:**

Mērķis tiks sasniegts, kad tiks novēroti šādi radītāji:

1. Optimizēta izvietošanas procesa organizācija: Lietojumprogrammu izvietošanas process kļūst ievērojami ātrāks un efektīvāks. Manuāla iejaukšanās ir izslēgta, un automatizētās darba plūsmas nodrošina vienmērīgu un uzticamu izvietošanu.

2. Samazināts laiks, lai lietojumprogramma nonāktu pie lietotājiem: Nepieciešamais laiks, lai piegādātu lietojumprogrammas tirgū, ir ievērojami samazināts. Lietojumprogrammas var tikt ātri izvietotas, ļaujot ātri reaģēt uz lietotāju prasībām.

3. Kļūdu samazināšana: Cilvēku radītas kļūdas izvietošanas procesā tiek samazinātas vai izslēgtas. Automatizēti testēšanas un kvalitātes nodrošināšanas procesi nodrošina lietojumprogrammu rūpīgu pārbaudi, rezultātā iegūstot augstas kvalitātes un uzticamu gala produktu.

Novērojot šos rādītājus mēs varam novērtēt, vai mērķis ir sasniegts, un veikt nepieciešamos uzlabojumus, lai sasniegtu izvirzīto mērķi.

**2. Programmatūras prasību specifikācija**

aaaaaaahfjhfjadhjkdhfkladskljfkldsjfmkldsnflskd.

**2.1. Produkta perspektīva**

Dslnfkdjfkjdlfjdsfjl‘dfdhhjdshhfjsdjuhjdskvfjhgfdzi.

**2.2. Sistēmas funkcionālās prasības**

**Administratora funkcijas**

**2.2.1. ggrhglhgrlghrlhrghlgr**

Mērķis:

lksnfgnglkfeglnkerqanu.

Ievaddati:

Tg;mwg;emrw;ergm.

Apstrāde:

G[otjrwpwogjnrpwogewor[epgj[weorjg[rweogj.

* **Pārbauda:**
  + Dmsfnldsanfldsfk;;
  + Djfsbds;bfnjkdsbfj;dsf;.

aa.aaaaaaaaaaaaaaaaaaaaaa

Izvaddati:

1. aaaaaaaaa.
2. aaaaaa
3. aaaaaaaaaaaaa

2.**3. Sistēmas nefunkcionālās prasības**

**2.3.1. Veiktspējas prasības**

L;dsmf;dsma;flmd;sfms;dlfm.

**2.3.2. Drošība**

Lkdfnslk/afndkl/fnas/klnf/slfl/kdnfslaf).

**2.3.3. Uzturamība**

Ddfldmfl;dmfl;dmfldsmfl;amfdl;mffDFMF ldmfl;dfklfu.

**2.3.4. Lokalizācija**

Ndlfa/slkfndfaklsndfklfnadknfkdlsnflasdfldināties).

**2.3.5. Pārnesamība**

dkfnlkdfklfnlkdffm.

**2.3.6. Lietojamība**

gfgtffgvhhfjhgjfhjgfgfgsfot.

**2.4. Gala lietotāja raksturzīmes**

Hjkjhjhjkj.gjgjk bvgjhj.kh

## 2.1. Produkta perspektīva

## 2.2. Sistēmas funkcionālās prasības

## 2.3. Sistēmas nefunkcionālās prasības

## 2.4. Gala lietotāja raksturiezīmes

## 2.5. Lietoto terminu un saīsinājumu skaidrojumi

# 3. Izstrādes līdzekļu, rīku apraksts un izvēles pamatojums

Šajā nodaļā tiek aprakstīti izmantotie līdzekļi, rīki un programmēšanas valodas, kas tika izmantotas produkta izstrādē. Šie rīki tika izvēlēti, pamatojoties uz to individuālajām stiprajām pusēm, savstarpējo saderību un spēju atbalstīt konkrētās produkta prasības, veicinot izstrādes darbplūsmas efektivitāti, automatizāciju un kodu kvalitāti.

**GitHub** ir plaši izmantota versiju kontroles sistēma, kas ļauj veidot programmatūru sadarbības veidā. Tas nodrošina platformu, kurā var glabāt un pārvaldīt kodu, veicinot efektīvu sadarbību starp izstrādātājiem. GitHub piedāvā daudzas un dažādas funkcijas, piemēram, zaru pārvaldību, "pull request" izvērtēšanu, koda pārskatu un problēmu izsekošanu, kas veicina efektīvu koda izstrādi un sadarbību. GitHub tika izvēlēts pateicoties tam, ka tam ir spēcīgas versiju kontroles iespējas, lietotājam draudzīga saskarne un tika plaši izmantots prakses vietā.

**Jenkins** ir atvērtā koda automatizācijas serveris, kurš tiek izmantots lietojumprogrammu veidošanai, testēšanai un izvietošanai. Tas piedāvā elastīgu un paplašināmu platformu nepārtrauktai integrācijai un nepārtrauktai izsūtīšanai (CI/CD). Jenkins ļauj automatizēt dažādas programmatūras izstrādes posmus, ieskaitot koda kompilāciju, testēšanu un izvietošanu, tādējādi optimizējot izstrādes un publicēšanas procesu. Tas tika izvēlēts pateicoties tam, ka tam ir lieliska pielāgojamība, plaša spraudņu ekosistēma un iespēja integrēties ar citiem rīkiem DevOps darbplūsmā, kā arī ikdienā tas tika plaši izmantots prakses vietā.

**Visual Studio Code (VS Code)** ir koda redaktors, kas nodrošina izcilu izstrādes vidi. Tas piedāvā plašu funkciju klāstu, piemēram, inteliģento koda aizpildīšanu, atkļūdošanu, versiju kontroles integrāciju un plašu paplašinājumu bibliotēku. VS Code atbalsta vairākas programmēšanas valodus un platformas, padarot to par daudzpusīgu rīku izstrādātājiem. Tas tika izvēlēts pateicoties tam, ka tam ir lietotājam draudzīga saskarne, plašas pielāgošanas iespējas un efektīva koda rediģēšanas funkcionalitāte. Šis rīks tika izmantots arī citos prakses un skolas projektos.

**SonarQube** ir populārs koda kvalitātes analīzes rīks, kas palīdz izstrādātājiem agrīnā izstrādes posmā atklāt un novērst koda problēmas. Tas piedāvā plašu koda analīzes noteikumu kopu, kas sniedz atziņas par koda kļūdām, drošības trūkumiem un dublējumiem. SonarQube palīdz uzturēt koda kvalitāti, uzlabot uzturējamību un ieviest kodēšanas standartus. Tas tika izvēlēts pateicoties tā spējai veikt koda analīzi, sniegt noderīgu atgriezenisko saiti un nodrošināt koda kvalitāti visā izstrādes ciklā. Šis rīks tika izmantots arī citos prakses projektos.

**Groovy** ir dinamiskā programmēšanas valoda kura bez problēmām integrējas ar Java. Groovy plaši tiek izmantots skriptēšanai un testēšanas automatizācijai. Tika izvēlēts pateicoties tam, ka tas ir viegli lasāms un saprotams, spēj rakstīt saīsinātus skriptus un piedāvā lieliskas iespējas automatizācijas uzdevumiem DevOps darbplūsmā.

**AWS (Amazon Web Services)** ir populāra mākoņa skaitļošanas platforma, kas piedāvā plašu pakalpojumu klāstu lietojumprogrammu un infrastruktūras veidošanai, izvietošanai un pārvaldībai. Tas nodrošina elastīgus un mērogojamus resursus, tai skaitā virtuālos serverus, datu glabāšanu, datu bāzes un tīklošanu, kas ļauj izveidot un palaist lietojumprogrammas mākoņa vidē. AWS tika izvēlēts pateicoties tās plašajam pakalpojumu spektram, drošībai, elastībai un uzticamībai, kas ļauj veiksmīgi īstenot lietojumprogrammu izstrādes un darbplūsmas vajadzības mākoņa vidē.

## 3.1 Iespējamo (alternatīvo) risinājuma līdzekļu un valodu apraksts

Šie alternatīvie rīki piedāvā līdzīgas funkcionalitātes iepriekš minētajiem rīkiem, sniedzot dažādas iespējas, kas atbilst konkrētajām vajadzībām un priekšrocībām.

**GitLab** ir tīmekļa bāzēta DevOps platforma, kas nodrošina versiju kontroli, nepārtrauktu integrāciju un nepārtrauktu izsūtīšanu (CI/CD). Līdzīgi kā GitHub, tas piedāvā kodu pārvaldīšanu, problēmu izsekošanu un koda pārskatu funkcijas. GitLab arī nodrošina iebūvētas CI/CD darbplūsmas, padarot to par visaptverošu alternatīvu GitHub.

**Travis CI** ir nepārtrauktas integrācijas platforma, kas ļauj viegli veidot, testēt un izvietot programmatūras projektus. Tas labi integrējas ar GitHub un piedāvā vienkāršu konfigurāciju un uzstādīšanu. Travis CI atbalsta dažādas programmēšanas valodas un nodrošina automātiskas izvietošanas iespējas uz dažādiem mākoņu risinājumiem.

**IntelliJ IDEA** ir integrēta izstrādes vide (IDE), kas atbalsta vairākas programmēšanas valodas un ietvarus. Tā piedāvā uzlabotas koda rediģēšanas funkcijas, inteliģentu koda aizpildīšanu, atkļūdošanas rīkus un iebūvētu versiju kontroles integrāciju. IntelliJ IDEA ir pazīstama ar savu spēcīgo spraudņu ekosistēmu un izcilajām produktivitātes funkcijām.

**Google Cloud Platform** ir Google piedāvāts mākoņu skaitļošanas pakalpojumu klāsts. Tas nodrošina dažādas rīkus un pakalpojumus lietojumprogrammu veidošanai, izvietošanai un pārvaldībai mākoņos. GCP piedāvā dažādas funkcijas, piemēram, virtuālās skaitļošanas mašīnas, datu glabāšanu, datu bāzes un mašīnmācīšanās pakalpojumus.

**ESLint** ir populārs atvērtā koda JavaScript rīks, kas palīdz atklāt un labot programmatūras koda kļūdas un uzturēt koda kvalitāti. Tas piedāvā pielāgojamu noteikumu kopu un atbalsta dažādus koda stilus un ietvarus. ESLint integrējas ar lielāko daļu koda redaktoru un būvēšanas sistēmu, padarot to par daudzpusīgu alternatīvu SonarQube koda analīzei un kvalitātes nodrošināšanai.

**Python** ir daudzpusīga un plaši izmantota programmēšanas valoda, kas pazīstama ar savu vienkāršību un vieglo lasāmību. Tā piedāvā lielu bibliotēku un ietvaru ekosistēmu, kas ļauj ātri veidot un automatizēt programmatūru. Python var izmantot dažādās jomās, tostarp skriptēšanai, tīmekļa izstrādei un datu analīzei. Tas kalpo kā alternatīva Groovy skriptēšanai un automatizācijai DevOps darbplūsmā.

## 3.2 Izvēlēto risinājuma līdzekļu un valodu apraksts

# 4. Sistēmas modelēšana un projektēšana

adsadad

## 4.1 Sistēmas struktūras modelis

sdadad

## 4.2. Klašu diagramma / ER diagramma

adadsad

## 4.3. Funkcionālais un dinamiskais sistēmas modelis

adadasd

## 4.4. Aktivitāšu diagramma (Activity)

asdadasd

## 4.5. Lietojumgadījumu diagramma (Use Case)

adsadsad

## 4.6. Sistēmas moduļu apraksts un algoritmu shēmas

adadad

# 5. Lietotāju ceļvedis

Lietotāja ceļvedis par to, kā izmantot būvēšanas darbplūsmu Jenkins vidē:

1. Piekļūt Jenkins:

- Atveriet tīmekļa pārlūku un dodieties uz Jenkins vietni, kuru esat saņēmuši.

- Pierakstieties, izmantojot savus akreditācijas datus.

2. Pārvietoties uz būvēšanas darbplūsmu:

- Jeknins informācijas panelī atrodiet būvēšanas darbplūsmas projektu, ko vēlaties izmantot.

- Noklikšķiniet uz darbplūsmas nosaukuma vai pārvietojieties uz projekta lapu.

3. Sākt darbplūsmu:

- Lai sāktu darbplūsmu, noklikšķiniet uz “Build now” pogas.

- Darbplūsma sāks izpildīties, sekojot secīgiem posmiem.

4. Uzraudzīt darbplūsmas izpildi:

- Darbplūsmas izpildes laikā varat uzraudzīt katra posma progresu.

- Jenkins sniedz reāllaika atjauninājumus par katra posma statusu.

5. Skatīt būvēšanas darbplūsmas rezultātus:

- Kad darbplūsma ir pabeigta, varat skatīt kopējos rezultātus un statusu.

- Jenkins parasti sniedz kopsavilkumu lapu ar informāciju par būvēšanu, testu rezultātiem un radītajiem artefaktiem.

Tas viss! Sekojot šiem soļiem, jūs varat izmantot būvēšanas darbplūsmu Jenkins vidē.

Lietotāja ceļvedis par to, kā izmantot izvietošanas darbplūsmu Jenkins vidē.

1. Piekļūt Jenkins:

- Atveriet tīmekļa pārlūku un dodieties uz Jemkins vietni, kuru esat saņēmuši.

- Pierakstieties, izmantojot savus akreditācijas datus.

2. Pārvietoties uz izvietošanas darbplūsmu:

- Jenkins informācijas panelī atrodiet izvietošanas darbplūsmas projektu, ko vēlaties izmantot.

- Noklikšķiniet uz darbplūsmas nosaukuma vai pārvietojieties uz projekta lapu.

3. Sākt izvietošanu:

- Lai sāktu izvietošanas procesu, noklikšķiniet uz “Build now" pogas.

- Darbplūsma sāks izpildīties, sekojot secīgiem posmiem.

4. Uzraudzīt izvietošanas darbplūsmas izpildi:

- Izvietošanas procesa laikā varat uzraudzīt katra posma progresu.

- Jenkins sniedz reāllaika atjauninājumus par katra posma statusu.

5. Skatīt izvietošanas rezultātus:

- Kad izvietošana ir pabeigta, varat skatīt kopējos rezultātus un statusu.

- Jenkins parasti sniedz kopsavilkumu lapu ar informāciju par izvietoto lietojumprogrammu, vides statusu un attiecīgajiem žurnāla ierakstiem.

Tas viss! Sekojot šiem soļiem, jūs varat izmantot izvietošanas darbplūsmu Jenkins vidē.

# 6. Testēšanas dokumentācija

adadsadas

## 6.1. Izvēlētās testēšanas metodes, rīku apraksts un pamatojums

adasdsad

## 6.2. Testpiemēru kopa

asdadsada

## 6.3. Testēšanas žurnāls

asdadsadsad

# 8. Secinājumi

1. Automatizācija un efektivitāte: Projekta galvenais mērķis ir automatizēt izvietošanas, testēšanas un aplikācijas būvēšanas procesus ražošanas plūsmā. Izmantojot dažādus rīkus, es vēlējos optimizēt šos uzdevumus un paaugstināt kopējo efektivitāti.

2. DevOps prakse: Ir svarīgi uzsvērt, ka šo projektu es izstrādāju, strādājot kā DevOps praktikants. Šī pieredze apliecina manas praktiskās prasmes un spēju sniegt ieguldījumu reālo projektu veidošanā DevOps jomā.

3. Rīku izvēle: Projekts izmanto piecus konkrētus rīkus - GitHub, Jenkins, Visual Studio Code, AWS, SonarQube un Groovy. Katrs rīks kalpo noteiktai funkcijai ražošanas plūsmā, un es tos izvēlējos, pamatojoties uz to funkcionalitāti un saderību ar projektu.

4. Vēlamais rezultāts: Projekta galvenais mērķis ir izveidot stabilu un automatizētu ražošanas plūsmu. Izmantojot atlasītos rīkus, es vēlējos panākt kļūdu un problēmu brīvu izvietošanu, efektīvu testēšanu un veiksmīgu aplikācijas būvēšanu.

5. Sasniegtā novērtēšana: Projekta veiksmi var novērtēt, vērtējot automatizācijas procesu efektivitāti un efektivitāti. Galvenie rādītāji var būt izvietošanas laika samazināšanās, uzlabota testēšanas precizitāte, stabilāka būvēšanas procesa stabilitāte un palielināta produktivitāte attīstības komandai.

Kopumā projekts risina automatizācijas nepieciešamību ražošanas plūsmā, un, izmantojot noteiktus rīkus, es centos izveidot efektīvu un optimizētu darba plūsmu. DevOps praksē esmu ieguvis praktisku pieredzi, kas dod projektam papildu vērtību.

# 9. Lietoto saīsinājumu saraksts

# 10. Literatūras un informācijas avotu saraksts