Automaattestide kasutamise nõuded

- Üldised nõuded
- Tehnilised nõuded

Üldised nõuded

Nõude nr	Nõude sisu	Seletused	Koostamise eest vastutaja	Testimise läbi viib või kinnitab
1. Automatiseeritavad testiliikide testid				

1.1	Automatiseeritakse kokkulepitud või minimaalse skoobi testiliikide testid	• Iga lahendus vaadatakse eraldi üle ja lepitakse kokku, milliseid testiliike peab lahenduse testimisel automatiseerima ja läbima.
		• **Minimaalne skoop -** komponenttestid (unit testid), integratsioonitestid, liideste testid, süsteemitestid (*end-to-end testid*), robustsuse testid ja koormustestid.
		• Komponenttestid (unit testid) ja integrtatsioonitestid käivitatakse enne reliisi kokku ehitamist.
		  ◦ **Buildi** skriptis on sammudes kirjas, et läbitakse komponenttestid (unit testid) ja integratsioonitestid, kui on edukalt läbitud, siis koostatakse **build**.
		• Liideste testid, süsteemitestid (*end-to-end testid*), robustsuse testid ja koormustestid läbitakse peale reliisi paigaldust keskkonda.
		  ◦ Peale **buildi** tegemist läbitakse liideste testid, süsteemitestid (*end-to-end testid*), robustsuse testid ja koormustestid.
			Arendaja	Arendaja testija
				TEHIK testija

1.2 Minimaalne skoop				

1.2.1	Komponenttestid (unittestid)	• Testimise eesmärk on kontrollida süsteemi komponendid töötavad erinevates olukordades ootuspäraselt.
		  ◦ Komponenttestid (unit testid) testitakse süsteemi üksikuid komponente. Eesmärk on kinnitada, et süsteemi koodi kõik komponendid toimivad ootuspäraselt. Komponenttestimine (unit testimine) toimub süsteemi väljatöötamise (süsteemi koodi kirjutamise etapis) arendajate poolt. Komponenttestid (unit testid) eraldavad süsteemi koodi osa ja testid, mis kontrollivad süsteemi koodi ootuspärasust. Komponent võib olla individuaalne funktsioon, meetod, protseduur, moodul või objekt.
		• Komponenttestid (unit testid) käivitatakse enne koodi kokku ehitamist.
		• Komponenttestid (unit testid) on üks osa üleantavast koodist.
		• Peab vastama MFN nõudele 4.21 (https://wiki.sm.ee/pages/viewpage.action?pageId=3834694).
		  ◦ Siin tuleb arvestada teekide mittetestimisega, muidu on koodi hindamise hinnangud valed.
		  ◦ Iga lahenduse juures otsustab arhitekt, kas aksepteeritav on madalam komponenttestide (unit testide) katvuse protsent kui on MFN nõue 4.21 kirjas.
		  ◦ Kui ei ole otsust, et on aksepteeritav ka madalam komponenttestide (unit testide) katvuse protsent, siis aksepteeritav on MFN nõue 4.21 kirjas olev katvuse protsent.
		• Koodi katvuse automaattestide komponenttestidega (unit testidega) kontrollimiseks kasutame tööriista *SonarQube* (sonar.tehik.ee).
		• *SonarQube*'s kontrollime bugide ja haavatavuse taset ning alla taset **A** ei ole aksepteeritav.
			Arendaja	Arendaja testija
				
				TEHIK testija

1.2.2	Integratsioonitestid	• Testimise eesmärk on kontrollida, kas integreeritud komponentide omavaheline kootöö toimib.
		  ◦ Integratsioonitestides on komponendid integreeritud ja testitakse komponentide omavahelist koostöö toimimist. Tüüpiline tarkvara projekt sisaldab mitmeid komponente, mida kirjutavad erinevad arendajad. Integratsioonitesti eesmärk on tuvastada vead nende komponentide integratsioonis. Keskendutakse komponentide omavahelise andmeside kontrollimisele.
		• Integratsioonitestid käivitatakse enne koodi kokku ehitamist.
		• Integratsioonitestid on üks osa üleantavast koodist.
		• Testitakse komponentide integreerumist süsteemis.
		  ◦ Vajalikke integreeritud komponentide omavahelist toimimise testimiseks kasutatakse, kas *mock*'e või käivitatakse konteineris etteantud andmestik. Saab kasutada nt. https://www.testcontainers.org/ võimalusi.
		• Käivitatakse koos komponenttestide (unit testide) töövoos.
		• Integratsioonitestid peavad olema projektis, kus hoitakse süsteemi koodi, et saaks neid teste kasutada.
			Arendaja	Arendaja testija
				
				TEHIK testija

1.2.3	Liideste testid	• Testimise eesmärk on kontrollida, et kas kaks erinevat süsteemi töötavad ja edastavad andmeid omavahel ootuspäraselt.
		  ◦ Liideste testid kontrollivad, kas kahe erineva süsteemi vaheline suhtlus toimib. Ühendus, mis integreerib kahte komponenti nimetatakse liideseks nagu nt. API'd, veebiteenused jne. Nende ühenduse teenuste või liidestega testimist nimetatakse liideste testimiseks. Liides on tarkvara, mis koosneb käskude komplektidest, teadetest ja muudest atribuutidest, mis võimaldavad teenuse ja kasutaja vahelist suhtlust.
		  ◦ Väliste süsteemidega liidestumise testimisel tuleb kontrollida kõigi liideste toimimist.
		• Liideste testid käivitatakse peale reliisi paigaldust keskkonda.
		• Lisaks positiivsele töövoole seatakse rõhku ka negatiivsele töövoole.
		• Testitakse kahe erineva süsteemiga nende omavahelist suhtlust ja edastatavaid andmeid.
		  ◦ Vajalike liideseid testimiseks ei *mock*'ta ega käivitata konteineris etteantud andmestikuga.
		  ◦ Vajalike liideste testimiseks kasutatakse *x-tee*'d.
		• Võimalik kasutada töövahendeid nagu nt. postman/newman (https://www.postman.com/) lahendust.
		• Liideste testid peavad olema projektis, kus hoitakse süsteemi koodi, et saaks neid teste kasutada.
			Arendaja	Arendaja testija
				
				TEHIK testija

1.2.4	Süsteemitestid (*end-to-end testid*)	• Testimise eesmärk on kontrollida, kas integreeritud süsteem vastab määratud nõuetele ehk testimine peab katma funktsionaalseid ja mitte-funktsionaalseid nõudeid.
		  • Süsteemitestid (*end-to-end testid*) kontrollitakse integreeritud süsteem vastab määratud funktsionaalsetele- ja mitte-funktsionaalsetele nõuetele ning seetõttu läbitakse testi tüübi testid funktsionaalne testimine ja mitte-funktsionaalne testimine.
		  • Funktsionaalse testimisega veendutakse, et arendatud funktsionaalsus ja/või muudetud funktsionaalsus on realiseeritud ja töötab ootuspäraselt.
		  • Mitte-funktsionaalse testimisega veendutakse, et arendatud funktsionaalsus ja/või muudetud funktsionaalsus vastab mitte-funktsionaalsetele nõuetele.
		• Vastavalt vajadusele läbitakse kasutajaliidese süsteemitestid (*end-to-end testid*).
		  • Mõnel lahendusel ei ole kasutajaliidest, siis läbitakse ainult backend tasemel süsteemitestid (*end-to-end testid*).
		  • Mõnel lahendusel on kasutajaliides, siis läbitakse nii front-end kui backend tasemel süsteemitestid (*end-to-end testid*).
		• Süsteemitestid (*end-to-end testid*) käivitatakse peale reliisi paigaldust keskkonda.
		• Peab vastama MFN nõudele 4.22 (https://wiki.sm.ee/pages/viewpage.action?pageId=3834694).
		  • Katvuse protsenti kontrollime hetkel manuaalselt ehk kontrollime automaatsete süsteemitestide (*end-to-end testide*) koodi ja hindame, kas automaattestidega on katvuse protsent vastavus funktsionaalsusega, mis on kirjas MFN nõudes 4.22.
		  • Plaan tulevikus kasutusele võtta katvuse protsendi kontrollimiseks tööriista *Xray* funktsiooni *Requirement Coverage*.
		• Hallatakse eraldi projektis, kus ei hoita süsteemi koodi ning mis sisaldab ainult süsteemiteste (*end-to-end teste*).
			Arendaja	Arendaja testija
				
				TEHIK testija

1.2.5	Robustsuse testid	• Testimise eesmärk on kontrollida süsteemi käitumist ning töö taastumist veaolukordades. • Robustsuse testid keskenduvad olukordadele, kus süsteemi enda mõni komponent või komponendid ei toimi osaliselt (nt. tagastavad valesid tulemusi) või ei toimi üldse (nt. liidese kaudu info edastamine on blokeeritud). • Robustsuse testid käivitatakse peale reliisi paigaldust keskkonda. • Hallatakse eraldi projektis, kus ei hoita süsteemi koodi ning mis sisaldab ainult robustsuse teste.	Arendaja	Arendaja testija TEHIK testija
1.2.6	Koormustestid	• Koormustestimise eesmärk on kontrollida süsteemi käitumist reaalsetes koormustingimustes ja mitu kasutajat korraga saavad teha paralleelselt tegevusi. • Koormustestide läbimisel määratakse süsteemi maksimaalne koormus, et kontrollida süsteemi käitumist reaalsetes koormustingimustes ja mitu kasutajat saavad teha paralleelseid tegevusi, Koormustestid on mitte-funktsionaalsete nõuete testimise tüüp. • Koormustestid läbitakse, et leida süsteemi koormuse piirkohtade leidmiseks. • Koormustestid käivitatakse sprindi lõpus, et veenduda süsteemi koormustaluvus ei ole kehvemaks muutunud. • Peab vastama MFN nõudele 4.17 (https://wiki.sm.ee/pages/viewpage.action?pageId=3834694). • Hallatakse eraldi projektis, kus ei hoita süsteemi koodi ning mis sisaldab ainult koormusteste.	Arendaja	Arendaja testija TEHIK testija
2	Automatiseeritud testiliikide testid kasutatakse vähemalt arenduskeskkonnas ja testkeskkonnas või kui on kokkulepitud, siis ka teistes keskkondades	• Arenduskeskkonnas kasutatakse automatiseeritud teste testiliikide testide testimiseks. • Arendaja ja TEHIK peavad koostöös panema arenduskeskkonnas olevad testiliikide automaattestid tööle ka vähemalt testkeskkonnas või kui on kokkulepitud, siis ka teistes olemasolevates keskkondades.	Arendaja TEHIK	Arendaja testija TEHIK tehniline testija TEHIK testija
3	Automatiseeritud testiliikide testide haldamine	• Automatiseeritud testiliikide haldamisega ja ajakohasena hoidmisega tegeleb arendaja.	Arendaja	Arendaja testija TEHIK tehniline testija TEHIK testija
4. Automaatestide haldamiseks kasutatakse *Domain Driven Design*'i *(DDD)*				
4.1	Koodihalduseks kasutatakse tööriista GitLab	• Kõik rakenduste lähtekoodid, automaattestid jm. seotud failid peavad asuma GitLab's (https://gitlab.sotsiaalministeerium.ee/).	Arendaja TEHIK	Arendaja Arendaja testija TEHIK arhitekt TEHIK administraator TEHIK tehniline testija TEHIK testija

4.2	Projektide haldamiseks kasutatakse *Domain Driven Design*'i *(DDD)*	• **Domain Driven Design (DDD) -** konseptsioon, mille kohaselt koodi struktuur ja keel (klassi nimed, klassi meetodid, klassi muutujad) peaksid ühtima äri domeeniga. Domain Driven Design (DDD) põhineb printsiipidel - põhirõhk on peamisel domeenil ja domeeni loogikal, keerukate disainide koostamine domeeni mudelile ning tehniliste ja domeeni ekspertidega koostöös täpsustatakse/lahendatakse iteratiivselt konseptuaalsete mudelite domeeni probleeme. Rohkem infot leiab siit.
		• Projektide haldamiseks kasutatatakse *Domain Driven Design (DDD)* kuna hallatakse projekti domeeni üleselt kui ka valdkonna põhiselt ehk iga domeeni eraldi.
		  • Projekti domeeni ülesel tehtud muudatused mõjutavad projekti domeeni üleselt.
		  • Iga domeen eraldi on kindla valdkonna domeen nagu nt. süsteemitestid (*end-to-end testid*).
		  • Seal tehtavad muudatused mõjutab ainult antud valdkonna domeeni mitte projekti ülest domeeni.
		• **Projekti haldamise struktuur, mida kasutame GitLab's:**
		  • Projekti jaoks koostatakse **grupp**.
		  • Projekti **gruppi** alla lisatakse peamine projekti **test repositoorium**, mis sisaldab:
		  • **Lähtekoodi.**
		  • **Komponentteste (unitteste).**
		  • **Integratsiooniteste.**
		  • **Liideste teste.**
		  • Projekti **gruppi** alla lisatakse projekti süsteemitestide (*end-to-end testid*) **test repositoorium**, mis sisaldab:
		  • **Süsteemiteste (*end-to-end teste*).**
		  • Projekti **gruppi** alla lisatakse projekti robustsuse testid **test repositoorium**, mis sisaldab:
		  • **Robustsuse teste.**
		  • Projekti **gruppi** alla lisatakse projekti koormustestid **test repositoorium**, mis sisaldab:
		  • **Koormusteste.**
		• Projekti **grupide** alla saab lisada ka **alam gruppe**, kuid sel juhul on **test repositooriumi** loogika kasutamine sama, mis peamise **gruppi** all.
		• Peamine projekti **test repositoorium** on domeeni ülene ehk kui seal tehakse muudatus, siis see mõjutab projekti domeeni üleselt.
		• Ülejäänud projekti **test repositoorium** on ühe domeeni põhised ehk kui tehakse muudatus nt. süsteemitestide (*end-to-end testide*) **test repositooriumis**, siis mõjutab ainult süsteemitestide (*end-to-end testide*) domeeni.
		
		GitLabis terminite vastavused *Domain Driven Design (DDD)* terminitega:
		• GitLab: grupp - DDD: projekt;
		• GitLab: test repositoorium - DDD: domeen.
			Arendaja
			
			TEHIK
				Arendaja
				
				Arendaja testija
				
				TEHIK arhitekt
				
				TEHIK administraator
				
				TEHIK tehniline testija
				
				TEHIK testija

5. *Continous Integratsioni (CI)* ja automaattestide käivitamise protsesside kasutamine				

5.1	Kasutatakse automaatset *Continous Integratsioni (CI)* protsessi	• **Continous Integratsion (CI) -** tarkvaraarenduses järgitav tava, kus arendajad pidevalt oma koodi ühisesse hoidlasse üles panevad. Iga muudatust kontrollib automaatne protsess, mis tagab arendusprotsessi stabiilsuse ning kui esineb vigu, leitakse need kiiresti. Selleks et arendajate tehtud muudatused koodis ei jääks isoleerituks, peavad arendajad oma muudatused ühishoidlas asuva koodiga liitma. Nende muudatuste edukust testitakse kohe, luues uus tarkvaraversioon (*build*), mida testivad automaattestid. Pidev integratsioon säästab aega, kuna identifitseerib varakult koodimuudatustest tulenevad konfliktid ja regressioonid, kuid selle edukaks läbiviimiseks on vaja palju automaatteste. Rohkem infot leiab siit.
		• **Automaatne *Continous Integratsion (CI)* protsess:**
		  • Läbitakse **buildi** koostamisel automaattestid - komponenttestid (unittestid) ja integratsioonitestid.
		  • Komponenttestid (unittestid) ja integratsioonitestid on läbitud edukalt ning koostatakse **build**.
		  • Komponenttestid (unittestid) ja integratsioonitestid on läbitud mitte-edukalt ning katkestatakse **buildi** koostamine.
		  • Tehakse **deploy**.
		  • Läbitakse automaattestid - liideste testid, süsteemitestid *(end-to-end* testid), robustsuse testid ja koormustestid.
		  • Liideste testid, süsteemitestid (*end-to-end testid*), robustsuse testid ja koormustestid on läbitud edukalt ning lõpetatakse protsess.
		  • Liideste testid, süsteemitestid (*end-to-end testid*), robustsuse testid ja koormustestid on läbitud mitte-edukalt ning tehakse keskkonnas **rollback**.
		  • **Rollback** ei tehta arenduskeskkonnas, aga testkeskonnas ja teistes kokkulepitud keskkondades tehakse **rollback**.
		• *Continious Integratsion (CI)* protsessi ehk reliisi üleandmise joonis Lisa 1. Automaattestide kasutamise nõuete joonised#Lisa1.1ContinousIntegration(CI)protsessehkreliis%C3%BCleandmiseksvalmis.
		• Lepitakse iga lahenduse juures eraldi kokku, milline peaks olema *Continous Integratsion (CI)* ehk peaks kõik automatiseeritud testiliigid läbima iga reliisi paigalduse korral või mõned automaattestid - robustsuste testid ja koormustestid - võiksid olla ka eraldi manuaalselt käivitatavad.
		• Projektis on mitu domeeni, kuid luuakse üks *Continous Integratsion (CI)* GitLab Pipeline.
		• **GitLab Pipeline struktuur:**
		  • **Tests**
		  • Läbitakse komponenttestid (unittestid) ja integratsioonitestid.
		  • **Build**
		  • Koostatakse **build**.
		  • **Deploy**
		  • Tehakse **deploy**.
		  • **Interface tests**
		  • Läbitakse liideste testid.
		  • **System tests (*end-to-end tests*)**
		  • Läbitakse süsteemitestid (*end-to-end testid*).
		  • **Robustness tests**
		  • Läbitakse robustsuse testid.
		  • **Load tests**
		  • Läbitakse koormustestid.
			Arendaja
			
			TEHIK
				Arendaja
				
				Arendaja testija
				
				TEHIK arhitekt
				
				TEHIK administraator
				
				TEHIK tehniline testija
				
				TEHIK testija

5.2	Tarnete paigaldus peab olema automatiseeritud	• Eeldus nõudele *Üldised nõuded nõue 4.1*.
		• **Tarne paigalduse automatiseerimiseks on kaks varianti:**
		  • Täiesti automaatne, et kui tarnitakse reliis, siis automaatselt käivitatakse reliisi paigaldus.
		  • Manuaalselt käivitatav ehk on tekitatud nupp reliisi paigalduse käivitamiseks, et kui kasutaja vajutab nuppu, siis paigaldatakse reliis automaatselt.
		• Automaatne reliisi paigaldus peab olema minimaalselt testkeskkonda tehtud, kui on kokkulepitud, siis ka teistesse olemasolevatesse keskkondadesse.
			Arendaja
			
			TEHIK
				Arendaja
				
				TEHIK arhitekt
				
				TEHIK administraator

5.3	Kasutatakse manuaalset automaattestide käivitamise protsessi	• **Manuaalne automaattestide käivitamise protsess:**
		  • Süsteemitestide (*end-to-end testid*) domeeni luuakse eraldi GitLab Pipeline, kus saab manuaalselt käivitada süsteemitestid (*end-to-end testid*).
		  • Robustsuste testid domeeni luuakse eraldi GitLab Pipeline, kus saab manuaalselt käivitada robustsuse testid.
		  • Koormustestid domeeni luuakse eraldi GitLab Pipeline, kus saab manuaalselt käivitada koormustestid.
			Arendaja
			
			TEHIK
				Arendaja
				
				TEHIK arhitekt
				
				TEHIK administraator

6	Testide automatiseerimiseks võib kasutada erinevaid tööriistu või raamistikke ning olemasolevaid kasutatavaid tööriistu või raamistikke	• Iga uue testimise automatiseerimise tööriista või raamistiku kasutusega tuleb eelnevalt TEHIKuga kokkuleppida, milline on antud tööriista või raamistiku arhitektuuri muster. • Olemasolevate tööriistadega või raamistike testide automatiseerimisel tuleb kasutada arhitektuuri mustrit, mis on kirjas nõuetes *Tehnilised nõuded nõue 4*.	Arendaja TEHIK	Arendaja testija TEHIK tehniline testija TEHIK testija TEHIK arhitekt TEHIK administraator
7	Automaattestide skriptide kood peab vastama MFN nõuetele 5	• MFN nõuded https://wiki.sm.ee/pages/viewpage.action?pageId=3834694.	Arendaja	Arendaja testija TEHIK testija
8	Automaattestide logimine ja debuggimine peab vastama MFN nõuetele 4 v.a nõuded 4.17, 4.21 ja 4.22	• MFN nõuded https://wiki.sm.ee/pages/viewpage.action?pageId=3834694.	Arendaja	Arendaja testija TEHIK testija
9. Dokumendid				
9.1	Automaattestide paigaldamisjuhend	• Koostatud peab olema automaattestide paigaldusjuhend, kuidas automaattestid integreerida testkeskkonda ja teistesse kokkulepitud keskkondadesse.	Arendaja	Arendaja testija TEHIK tehniline testija TEHIK testija TEHIK arhitekt TEHIK administraator
9.2	Automaattestide kasutusjuhend	• Koostatud peab olema automaattestide kasutusjuhend, kuidas automaattestid käivitada ning kus vajalikud logid asuvad jm. vajalikud failid.	Arendaja	Arendaja testija TEHIK tehniline testija TEHIK testija TEHIK arhitekt TEHIK administraator

Tehnilised nõuded

Nõude nr	Nõude sisu	Seletused	Koostamise eest vastutaja	Testimise läbi viib või kinnitab
1	Automatiseeritud testimiseks tuleks kasutada tööriista / raamistiku, mis võimaldab kergesti automaattestimist hallata.	**Põhinõuded:** • lisada uued testid; • kustutada testid; • välja/sisse lülitada testid; • parandada testid; • hallata teste.	Arendaja	Arendaja Arendaja testija TEHIK arhitekt TEHIK administraator TEHIK tehniline testija TEHIK testija
2	Kõik testide automatiseerimise tööriistad / raamistikud peavad oskama töötada GitLab PipeLine's ja Docker Container's.	• Kogu testimine peaks olema kavandatud töötama GitLab Pipelines automaatselt ja ilma manuaaltegevuseta.	Arendaja	Arendaja Arendaja testija TEHIK arhitekt TEHIK administraator TEHIK tehniline testija TEHIK testija

3	Automaattestide kasutamise tehniline loogika	• Testide automatiseerimise kasutamise tehniline loogika joonis Lisa 1. Automaattestide kasutamise nõuete joonised#Lisa1. 2Testideautomatiseerimiselekasutamisetehnilineloogika.	Arendaja TEHIK	Arendaja Arendaja testija TEHIK arhitekt TEHIK administraator TEHIK tehniline testija TEHIK testija
4. Docker image				
4.1	Kõik docker image tuleks eelnevalt ette valmistada.	• Vaata punkt 4.2, kõik 4.3 alampunktid ja 4.4.	Arendaja TEHIK	Arendaja TEHIK arhitekt TEHIK administraator TEHIK tehniline testija
4.2	Docker image peaks sisaldama kõiki vajalikke sõltuvusi konteineri kiireimaks käivitamiseks.	• Valmistatud docker image peab olema koos installitud sõltuvustega. • Java-testimiseks peab vajalikud library/raamistikud ette valmistama **local /.m2** kataloogis. • Docker image tuleks luua võimalikult kerge ja kõik **prügi** tuleks enne loomist eemaldada	Arendaja TEHIK	Arendaja TEHIK arhitekt TEHIK administraator TEHIK tehniline testija
4.3 Dockeri image testimise struktuurid				
4.3.1	Dockeri image testimise struktuurid	• Java baasil testimine. • Npm baasil testimine. • SoapUI baasil testimine. • Muude tööriistade baasil testimine.	Arendaja TEHIK	Arendaja TEHIK arhitekt TEHIK administraator TEHIK tehniline testija

4.3.2	Docker image Java baasil testimise struktuur	• Alpine või Ubuntu baasil image.
		• Java v8 või kõrgem.
		• **Maven või Gradle:**
		  • Gatling (**/.m2**).
		  • JUnit (**/.m2**).
		  • Selenium (**/.m2**).
		  • Selenid (**/.m2**).
		  • Teised java raamistikud.
			Arendaja
			
			TEHIK
				Arendaja
				
				TEHIK arhitekt
				
				TEHIK administraator
				
				TEHIK tehniline testija

4.3.3	Docker image Npm baasil testimise struktuur	• Alpine või Ubuntu baasil image.
		• **Node.js/Npm:**
		
		  • Cypress;
		  • Newman;
		  • Teised npm raamistikud.
			Arendaja
			
			TEHIK
				Arendaja
				
				TEHIK arhitekt
				
				TEHIK administraator
				
				TEHIK tehniline testija

4.3.4	Docker image SoapUI baasil testimise struktuur	• Alpine või Ubuntu baasil image. • Java v8 või kõrgem. • OpenSource SoapUI.	Arendaja TEHIK	Arendaja TEHIK arhitekt TEHIK administraator TEHIK tehniline testija
4.4	Kasutada igal võimalusel Docker'is uusimat tarkvara või testide automatiseerimise raamistiku versiooni.	• Kui testimine ei vaja vanu **Library** ja testide automatiseerimise raamistikke, peab kasutama viimast versiooni.	Arendaja TEHIK	Arendaja TEHIK arhitekt TEHIK administraator TEHIK tehniline testija

5. Automaattestide GitLab repository				

5.1	GitLab automaattestide repository struktuur	• Kõik PipeLine töötamiseks vajalikud failid peavad olema repository **root** kataloogis
		  • readme
		  • litsents
		  • gitlab-ci.yml
		  • .....
		• Testprojekt või testfailid peavad olema repository **test** kataloogis
		• Test kataloogi nimi peab pealkirjas sisaldama:
		  • "testi-tüüp"-
		  • -tests-
		  • -"projekt või mooduli-nimi"
			Arendaja
			
			TEHIK
				Arendaja
				
				Arendaja testija
				
				TEHIK arhitekt
				
				TEHIK administraator
				
				TEHIK tehniline testija
				
				TEHIK testija

5.2	GitLab automaatesti repository üldine struktuur	***/*** - GitLab Repository **root** kataloog. ***/'testi tüüp'-tests-'projekt või mooduli nimi/*** - automaattestide **root** kataloog. ***.../some-lib/*** - mõne automaattesti kataloog koos **Library** ja **Dependency**'ga. ***.../some-dir/*** - mõne automaattesti kataloog. ***.../some-dir/some-dir/*** - mõni teine automaattesti kataloog. ***.../some-dir/some-dir/some-dir/*** - automaattesti kataloog. ***.../some-file.file*** - mõne automaattesti failid. ***.../other*** - teised projekti automaattesti failid. ***/gitlab-ci.yml*** - GitLab Pipeline konfiguratsiooni fail. ***/README.md*** ***/.gitignore*** ***/LICENSE*** ***/Muud GitLab Pipline failid***	Arendaja TEHIK	Arendaja Arendaja testija TEHIK arhitekt TEHIK administraator TEHIK tehniline testija TEHIK testija
5.2 Front-end automaattestide GitLab repository				
5.2.1	Front-end Gitlab repository struktuur	• Selenium/Selenid repository struktuur - punkt 5.2.2.	Arendaja TEHIK	Arendaja Arendaja testija TEHIK arhitekt TEHIK administraator TEHIK tehniline testija TEHIK testija

5.2.2	Selenium/Selenide Gitlab repository struktuur	• Peab vastama punktile 5.1 ja 5.2.
		• Repository struktuur:
		  1. Juurkataloog testide jaoks on:
		  • **/test-kataloogi-nimi/src/test/java/***
		  2. Lisa library juurkataloog on:
		  • **/test-kataloogi-nimi/lib/***
		  3. Gradle ja Maven juurkataloog on :
		  • **/test-kataloogi-nimi/***
		• Muud failid ja kataloogid luuakse vabas vormis, kuid neid tuleb eelnevalt arutada.
		
		Selenium/Selenid struktuur:
		
		/...
		
		/ui-tests-'projekt või mooduli-nimi'/
		
		.../src/test/java/*
		
		.../lib/*
		
		.../other-dir/
		
		.../other-project.file
		
		.../Gradle-or-Maven.files
		
		/...
			Arendaja
			
			TEHIK
				Arendaja
				
				Arendaja testija
				
				TEHIK arhitekt
				
				TEHIK administraator
				
				TEHIK tehniline testija
				
				TEHIK testija

5.3 Back-end automaattestide GitLab repositpory				
5.3.1	Back-end Gitlab repository struktuur	• Postman Gitlab repository struktuur - punkt 5.3.2. • SoapUI Gitlab repository struktuur - punkt 5.3.3.	Arendaja TEHIK	Arendaja Arendaja testija TEHIK arhitekt TEHIK administraator TEHIK tehniline testija TEHIK testija
5.3.2	Postman Gitlab repository struktuur	• Peab vastama punktile 5.1 ja 5.2. • Repository struktuur: 1. Juurkataloog testide jaoks on: • **/test-kataloogi-nimi/postman-test-file.json** 2. Lisa library juurkataloog on: • **/test-kataloogi-nimi/lib/*** 3. Lisa scriptide juurkataloog on: • **/test-kataloogi-nimi/scripts/*** 4. Test data juurkataloog on: • **/test-kataloogi-nimi/custom/*** • Muud failid ja kataloogid luuakse vabas vormis, kuid neid tuleb eelnevalt arutada. **Postman struktuur:** ***/...*** ***/'api'-tests-'projekt või mooduli-nimi'/*** ***/'rest'-tests-'projekt või mooduli-nimi'/*** ***/'soap'-tests-'projekt või mooduli-nimi'/*** ***.../postman-test-file.json*** - automaattesti projekti andmed. ***.../lib/**** - automaattesti **Library**. ***.../lib/some-test-lib.file*** ***.../scripts/**** - kohandatud automaattestide skriptid. ***.../scripts/start.sh*** ***.../scripts/run-after-postman-end.js*** ***.../custom/**** - automaattesti andmed. ***.../custom/postman-data-csv.file*** ***.../other-dir/*** ***.../other-project.file*** ***/...***	Arendaja TEHIK	Arendaja Arendaja testija TEHIK arhitekt TEHIK administraator TEHIK tehniline testija TEHIK testija

5.3.3	SoapUI Gitlab repository struktuur	• Peab vastama punktile 5.1 ja 5.2. • Repository struktuur: 1. Juurkataloog testide jaoks on: • **/test-kataloogi-nimi/soapui-test-file.xml** 2. Lisa library juurkataloog on: • **/test-kataloogi-nimi/lib/*** 3. Lisa scriptide juurkataloog on: • **/test-kataloogi-nimi/scripts/*** 4. Test data juurkataloog on: • **/test-kataloogi-nimi/custom/*** • Muud failid ja kataloogid luuakse vabas vormis, kuid neid tuleb eelnevalt arutada **SoapUI struktuur:** ***/..*** ***/'api'-tests-'projekt või mooduli-nimi'/*** ***/'rest'-tests-'projekt või mooduli-nimi'/*** ***/'soap'-tests-'projekt või mooduli-nimi'/*** ***.../soapui-test-file.xml*** - automaattesti projekti andmed. ***.../lib/**** - automaattesti **Library**. ***.../lib/some-test.lib.file*** ***.../scripts/**** - kohandatud automaattesti skriptid. ***.../scripts/start.sh*** ***.../scripts/run-after-postman-end.js*** ***.../custom/**** - automaattesti andmed. ***.../custom/soapui-data-csv.file*** ***.../other-dir/*** ***.../other-project.file*** ***/...***	Arendaja TEHIK	Arendaja Arendaja testija TEHIK arhitekt TEHIK administraator TEHIK tehniline testija TEHIK testija
6. Käivitamine				
6.1	Kogu loogika ja käivitusjärjestus tuleb **gitlab-ci.yml** failis määratleda.	• Testid võivad töötada nii põhiprojektist eraldi kui ka põhiprojekti ajal. • Gitlab *continous integratsionit* tuleb konfigureerida nii, et see saaks iseseisvalt töötada (eraldi projektis). • Gitlab *continous integratsionit* tuleb konfigureerida nii, et see saaks töötada põhiprojekti osana (module or extension for main project).	Arendaja TEHIK	Arendaja Arendaja testija TEHIK arhitekt TEHIK administraator TEHIK tehniline testija TEHIK testija
6.2	Peaks olema võimalus testid läbi viia **osaliselt** või **ükshaa val**.	**Testi käivitamine võib olla:** • **kõik testid** (all tests/test suites); • **üks** valitud test (test case); • **mitu** valitud testi (test cases); • valitud **testikomplekt** (test suite); • valitud **testikomplektid** (test suites); • valitud **testikomplekt/testikomplektid** ja **üks/mitu** valitud testi.	Arendaja TEHIK	Arendaja Arendaja testija TEHIK arhitekt TEHIK administraator TEHIK tehniline testija TEHIK testija
6.3	GitLab PipeLine testi käsitsi valimiseks kasutatakse eelnevalt määratletud muutujaid, nii testide automatiseerimise raamistikus kui ka repositoriumis.	• Kõik muutujad tuleb edaspidiseks kasutamiseks eelnevalt kokku leppida.	Arendaja TEHIK	Arendaja Arendaja testija TEHIK arhitekt TEHIK administraator TEHIK tehniline testija TEHIK testija

7. Muud				
7.1	Kasutada **docker image** või eraldi **virtuaal masinat** mocki jaoks.	• Kui on vaja kasutada mock, siis ta peab olema **docker container** se es koos kõigi vajalike sõltuvustega. • Kui docker ei saa kasutada, siis mock saab panna käima virtuaal masinal.	Arendaja TEHIK	Arendaja Arendaja testija TEHIK arhitekt TEHIK administraator TEHIK tehniline testija TEHIK testija

7.2	Testide automatiseerimisel kasutades raamistiku, mis kasutab WebDriverit peab eemalt saama käivitada automaattestid.	• Testid tuleb esialgu ette valmistada kaugkäivitumiseks.
		• Testimiseks kasutatakse **Selenoid Hub**
		
		Selenoid Hub kasutamine:
		
		• **Local hub:**
		  • Isiklik hub
		  • Kohalik testimine
		• **Public Test hub:**
		  • Test Debuging
		  • GitLab PipeLine Test/Proto branch
		• **Public Live hub:**
		  • Testid
		  • GitLab PipeLine Master branch
		• **Eraldi Hub projekti jaoks:**
		  • GitLab PipeLine Master branch
		  • Testi isolatsioon
		  • Piiratud juurdepääs andmetele
		  • Testandmed
		
		Remote käivitamine kasutavad:
		
		• Selenium;
		• Selenide;
		• Muud.
			Arendaja
			
			TEHIK
				Arendaja
				
				Arendaja testija
				
				TEHIK arhitekt
				
				TEHIK administraator
				
				TEHIK tehniline testija
				
				TEHIK testija