Lisa 2
RMK ja Askend Estonia OÜ
Vahelise lepingu nr 1-18/2026/92
juurde

Arendusnõuded

Dokumendi punktis 2.1 viidatud *“RMK Töö- ja koodihaldus” ja punktis 5 viidatud “Logimise spetsifikatsioon” väljastatakse hanelepingu sõlmimise järel ning punktis 2.3 viidatud “Eelistatud tehnoloogiate nimekiri” väljastatakse konfidentsiaalsusleppe (NDA) või hankelepingu sõlmimise järel.*

Sisukord

1. Ülevaade ja skoop

Käesolev dokument kirjeldab kuidas RMK-s arendatakse, hangitakse ja juurutatakse tarkvara. See hõlmab nii tarkvara arendust kui ka karbitoodete ja SaaS (Software as a Service) teenuste kasutusele võttu. Nõuded ei kirjelda ärifunktsionaalsust.

Dokumendis viidatakse ka teistele välistele dokumentidele, mis moodustavad käesoleva dokumendiga ühtse terviku.

---

2. Üldnõuded

2.1 Töö- ja koodihaldus

- Kõik tööd peavad olema kirjeldatud piletina RMK JIRA-s. Täpsemad JIRA ja versioonihalduse kasutamise nõuded on kirjeldatud dokumendis „RMK Töö- ja koodihaldus“(välisele arenduspartnerile väljastatakse dokument peale hankelepingu sõlmimist).
- **Git repod:** RMK kood asub **Azure DevOps** repodes.

2.2. Operatsioonid, lokaalne käivitamine, CI/CD ja release

- **Lokaalne käivitus:** Rakendused peavad olema lokaalselt käivitatavad ilma väliste sõltuvusteta. Vajalikud sõltuvused peaksid olema lokaalselt käivitatavad Docker Compose-i konteineriga. Välised sõltuvused(API-d) võivad olla mockitud. Rakenduse käivitamiseks ja testimiseks vajalikud andmed peavad olema rakenduses seadistatud. Peale käivitamist peab olema rakendus kasutatav. Rakenduse
  Vajalikud käsud ja sammud rakenduse lokaalseks käivitamiseks peavad olema välja toodud koodi repos README.md failis.
- **Koodi formaadi kontroll:** koodi formaadi kontroll peab olema build protsessi osa. Kood peab vastama RMK-s kokku lepitud formaadile.
- **Staatiline analüüs:** staatiline analüüs(nt Sonar) peab olema build protsessi osa selliselt, et kui kvaliteedi lävendit ei ületata, siis build ei õnnestu. Uus kood ei tohi tekitada juurde Sonari vigu. Lokaalselt seda sammu ei ole(võimalusel kasutada Sonarit IDE pluginana, või minna eraldi Sonari liidesesse ja kontrollida vigasid).

- **Testimise üldnõuded:** vt peatükk 9.
- **Release strateegiad:** kasutusel võivad olla erinevad väljalaskmise mustrid (nt blue-green, canary). Tähtis on, et uue versiooni kasutuselevõtt oleks kontrollitud ja võimaldaks vajadusel kiiret tagasipööramist.
- ***Production* ressursivajadus:** arendusmeeskonna ülesanne on testida rakenduse jõudlust ja selle põhjal teha ettepanek rakenduse ressursivajaduse kohta production keskkonnas, mis on sisendiks haldusosakonnale.

2.3 Platvorm

- **Versioon:** Ükskõik mis programmeerimiskeelte, raamistike või teekidega infosüsteeme realiseeritakse, kehtib sama põhimõte: alati kasutatakse kas viimast LTS(Long Term Support) versiooni, või selle puudumisel viimast stabiilset versiooni. Kui arendustöö käigus tuleb uus LTS versioon välja, siis see uuendatakse kohe ära, kui ei esine seda takistavaid asjaolusid.
- **Eelistatud tehnoloogiate nimekiri:** Nimekiri programmeerimiskeeltest, raamistikest, teekidest ja nende versioonidest, mida võib ilma täiendava kooskõlastamiseta kasutada on kirjeldatud dokumendis „Eelistatud tehnoloogiate nimekiri“(välisele arenduspartnerile väljastatakse dokument riigihanke käigus NDA alusel või peale hankelepingu sõlmimist). Juhul kui on soov kasutada midagi muud, siis tuleb see valik kooskõlastada RMK tarkvaraarhitektiga.
- **Keskkond:** backend, frontend ja andmebaasid töötavad Azure **pilvekeskkonnas**.
- **CI/CD:** kasutatakse Azure CI/CD. Build protsessi osana käivitatakse koodi formaadi kontroll, staatiline analüüs, testid.
- **Sõltuvused:** kasutatakse ainult toetatud ja turvauuendusi saavaid teeke.
- **Skaleeruvus:** rakenduste arendamisel peab arvestama võimalusega, et need jooksevad mitme õla peal.
- **Docker:** rakendus peab olema tarnitav Dockeri konteineritena.

2.4 Koodi kvaliteet ja arendusprotsess

- **Põhiharu kvaliteet:** main haru peab alati olema testitud ja valmis production keskkonda paigaldamiseks.
- **Sisendi valideerimine:** kõik kasutajalt või teistest süsteemidest tulevad sisendid valideeritakse. Kui rakendusel on nii kliendi- kui serveripool, toimub valideerimine mõlemas komponendis. Samuti rakendatakse valideerimist teistes integratsioonipunktides(andmebaas, välised liidesed).
- **Väljundandmete kontroll:**
  Rakenduse poolt kasutajale või teisele süsteemile väljastatavad andmed peavad olema **õigesti vormindatud, äriliselt korrektsed ja turvalised**.
    - **Formaadi kontroll:** kõik väljastatavad väärtused vastavad kokkulepitud skeemile või standardile. Näiteks kuupäevad esitatakse ISO 8601 formaadis

(2025-09-02T12:00:00Z), rahasummad kahe komakohaga, ning API vastus vastab OpenAPI kirjeldatud struktuurile.

  - **Äriline korrektsus:** väljastatav info järgib ärireegleid ja loogikat. Näiteks ei väljastata negatiivset saldo väärtust, kui süsteemi reeglid seda ei luba; ID viitab alati reaalselt olemasolevale kirjetele; staatuse väljad sisaldavad ainult eeldefineeritud väärtusi.
  - **Turvalisus:** väljund ei tohi sisaldada tundlikke või sisemisi andmeid, mida kasutaja või partner ei pea nägema (nt paroole, võtmeid, sisemisi tehnilisi tunnuseid). Tekstid peavad olema korrektselt escape'itud, et vältida XSS või sarnaseid ründeid. Logidesse ei tohi sattuda täielikud isikuandmed ega salajased identifikaatorid.

- **Staatiline koodianalüüs:** kasutatakse staatilist koodianalüüsi(Sonar); kõrge prioriteediga turvavead tuleb lahendada enne live keskkonda paigaldamist. Kõik Sonari vead vaadatakse enne arenduse üleandmist üle, vead mida ei peeta otstarbekaks parandada kooskõlastatakse RMK tarkvaraarhitektiga.
- **Kompilaatori hoiatused:** build-i hoiatused ja staatilise analüüsi tulemused vaadatakse üle ja arvestatakse.
- **Kiirlahenduste vältimine:** ei kasutata testimata lahendusi, mis eiravad häid praktikaid või RMK nõudeid.
- **Testimise põhimõtted:** vt peatükk 9.

2.5 Dokumenteerimine

- **Dokumentatsiooni põhielemendid:**
  - süsteemi funktsionaalne ja tehniline kirjeldus,
  - kasutatavad liidesed (API-d, integratsioonid),
  - perioodilised tööd (schedulerid, batch-protsessid),
  - arenduskeskkonna seadistamine
  - andmebaasi struktuuri kirjeldus (tabelid, seosed, migratsioonid),
  - keskkonnad ja paigaldus
  - peamised hooldustegevused (nt logide ülevaatus, varunduse kontroll, sõltuvuste uuendamise tsükkel)
- **Dokumentatsiooni asukoht:** RMK Confluence keskkond
- **Diagrammid:** süsteemi arhitektuuri ja komponente visualiseeritakse eelistatult PlantUML või Mermaid formaadis.
- **Arhitektuur:** iga süsteemi kohta peab olema arhitektuuriülevaade, sealhulgas API spetsifikatsioonid (nt OpenAPI).

2.6 Haldus

- **Tehniline võlg:** arenduse käigus tuleb dokumenteerida teadaolevad piirangud ja tehniline võlg (nt TODO-d, ajutised lahendused) ning soovitused nende likvideerimiseks.
- **Sõltuvuste ja platvormi uuendused:** arenduse käigus tuleb kirjeldada süsteemi kaasaegsena hoidmise strateegiat (nt milliseid raamistikke ja teeke tuleb regulaarselt uuendada, millal tuleb planeerida üleminek LTS-versioonidele).
- **Monitooringu vajadused:** tarnija peab kirjeldama arendatava süsteemi monitooringu aspektid (nt vealogid, jõudlusmõõdikud, integratsioonide toimivus, job scheduler-i töö).
- **Teavitused:** tarnija peab defineerima soovitatud häirete künnised (nt kui API annab vea, kui mingi protsess ei lõpeta näiteks 15 minutiga).
- **Elutsükli planeerimine:** tarnija peab andma hinnangu, millised tehnoloogiad või komponendid võivad lähiaastatel vajada moderniseerimist või väljavahetamist.
- **Andmete säilitustähtajad:** kui süsteemis talletatakse andmeid, millel on määratud säilitustähtaja (vastavalt teabe liigitusskeemile Jira Assetis), tuleb see dokumenteerida. Dokumentatsioonis peab olema kirjeldatud, millised andmed kustutatakse automaatselt, millal ja kuidas.

2.7 Välised sõltuvused

- **Teekide kasutamine:** kasutada ainult laialdaselt toetatud ja turvauuendusi saavaid teeke.
- **Sõltuvuste miinimum:** sõltuvusi hoitakse minimaalsena, et vähendada turvariske ja halduskoormust.
- **Litsentsid:** sõltuvuste litsentsitingimusi, turvalisust ja ajalugu hinnatakse; eelistatakse vabavaralisi komponente.
- **Jätkusuutlikkus:** hinnatakse teekide jätkuarenduse ja toe kättesaadavust. Vajadusel analüüsitakse mõju RMK protsessidele, kui komponendi arendus lõpeb.
- **Ühilduvus:** analüüsitakse kasutatavate komponentide ühilduvust teiste süsteemidega.

2.8 Muud üldised nõuded

- **Kodeering ja locale:** kogu süsteemis (lähtekoodis, andmebaasis, jne) kasutatakse UTF-8 kodeeringut; vaikekeel on et-EE (laiendatav i18n); ajatsoon on UTC (UI-s kohalik aeg).

---

3. Frontend nõuded (UX/UI)

3.1 Tehnilised

- **Tehnoloogia:** kui rakenduse nõuded võimaldavad siis eelistame serveris renderdatud HTML-i (Thymeleaf, HTMX).
- **Raamistikud:** kui kasutatakse mingit raamistikku, siis eelistame laialdaselt toetatud raamistikku (nt React, Angular või Vue).

- **Raamistiku versioon:** projektis kasutatavad raamistikud peavad olema **toetatud ja saama turvauuendusi**.
- **Build:** sõltuvused hallatakse **paketihalduriga**, arvestades kasutatavat tehnilist lahendust:
    - **SPA:** npm / yarn / pnpm (lock fail versioonihalduses; build deterministlik).
    - **Java SSR (Server-Side Rendering nt Thymeleaf):** Gradle (või Maven) hoiab nii backendi kui frontendi build-i .
    - **PHP SSR:** Composer.
    - **Python SSR:** pip.
      Build peab olema **deterministlik** ja **versioonitav** (lock failid, build-config git-s).
- **Keskkonnaspetsiifilised seaded:** hoitakse **välises konfiguratsioonis** (nt .env).
- **API kasutus:** kõik API-päringud käivad läbi **konfiguratsiooni alusel määratud lõpp-punktide**; otselinke koodis ei kasutata.
- **Turvalisus:** tundlikke andmeid ei salvestata brauseri püsivasse vahemällu (LocalStorage, SessionStorage) ilma krüpteerimiseta.
- **Jõudlus:** rakendus peab laadima mõistliku aja jooksul – esmase laadimise vaikimisi nõue on 2s. Täpsem kontekst (nt mõõdik, tüüpiline seade ja võrguolud) lepitakse projekti alguses kokku ja fikseeritakse rakenduse/projekti dokumentatsioonis. Pildid, skriptid ja stiilid peavad olema optimeeritud (lazy load, cache, minify).
- **Telemeetria:** kriitilised kliendipoolsed vead (JavaScript errorid, katkised API päringud) peavad jõudma tsentraalsesse monitooringusse.

3.2 Kasutuskogemus

- **Reageeriv(responsive) disain:** rakendus peab olema kasutatav erineva ekraanisuurusega seadmetel (desktop, mobiil, tahvel).
- **Navigatsioon:** menüüd ja liikumisloogika on rakenduse eri vaadetes ühtsed. Sama funktsionaalsus on alati leitav samast kohast.
- **Nuppude paigutus ja järjestus:** sama tüüpi tegevused (nt “Salvesta”, “Tühista”) paiknevad alati samas järjekorras ja paigutusega.
- **Veateated:** esitatakse ühtses formaadis (struktuur, sõnastus, värvilahendus). Teated on arusaadavad ja ei sisalda tehnilist müra. Veateated ei sulgu automaatselt, et kasutajal oleks võimalik neid adekvaatselt edastada toe saamiseks.
- **Kinnitused ja dialoogid:** riskantsete tegevuste (nt kustutamine, lõplik kinnitamine) puhul küsitakse alati kinnitust. Dialoogid järgivad sama stiili ja loogikat.
- **Abitekstid ja juhendid:** tooltipid, sisestusväljade placeholderid ja abitekstid on stiililt ja positsioonilt ühtsed. Kõik väljad ja tegevused on varustatud abitekstiga, mis aitab kasutajal mõista, **mis juhtub** enne kui ta midagi teeb.
- **Laadimise tagasiside:** pikemate tegevuste puhul (nt päring serverisse, faili üleslaadimine) peab kasutaja nägema visuaalset indikaatorit (spinner, progress bar).

- **Etteaimatavus:** süsteemi käitumine peab olema **selge ja ootuspärane** – väldime olukorda kus kasutaja peab mõtlema „mis siis juhtub, kui ma siia vajutan“. Tegevuste tagajärjed on nähtavad või selgitatud enne sooritamist.
- **Ikonograafia ja sümbolid:** sama tegevuse tähistamiseks kasutatakse alati sama ikooni ja tähendust.
- **Stiil ja kujundus:** värvipalett, kirjatüübid ja komponentide kujundus vastavad ühtsele disainisüsteemile või stiilijuhendile.
- **Klahvikombinatsioonid ja kiirtoimingud (kui rakendatud):** töötavad järjepidevalt kõigis vaadetes.
- **Terminoloogia:** tekstid ja nimetused (nt “kasutaja”, “konto”, “fail”) on järjepidevalt samad kogu rakenduses.
- **Andmete sisestuse valideerimine:** sisestusväljade puhul kasutatakse valideerimist enne serverisse saatmist, et ilmseid vigu kasutajale kiiremini esile tuua.

---

4. Backend nõuded (API/teenused)

4.1 Konfiguratsioon ja deploy

- **Seadistused:** keskkonnaspetsiifilised väärtused hoitakse rakendusest väljas (**Azure Key Vault**, keskkonnamuutujad), aga konfiguratsioon ise on koodi juures vastavates failides.
- **Konfiguratsioon:** konfiguratsioon on **versioneeritav** ja muudatused **jälgitavad**.
- **Ehitamine:** Java projektide ehitamisel eelistatakse **Gradle**-t.

4.2 API disain ja dokumentatsioon

- **Versioonimine:** REST/HTTP liidesed on **versioonitud** (nt /v1).
- **Dokumentatsioon:** kõik REST/HTTP liidesed peavad olema **dokumenteeritud** nii, et partneritel ja RMK-l on võimalik neid kasutada ja testida. Dokumentatsiooni võib teha erineval viisil (nt kirjalik juhend, Postman collection, README näidised). **Soovitatavalt** kasutatakse masinloetavat standardit **OpenAPI** ja sellest genereeritud Swagger liidest.
- **Asukoht:** kui kasutatakse OpenAPI/AsyncAPI dokumentatsiooni, peab vastav fail asuma samas Git repo-s koos lähtekoodiga, et see oleks alati koodiga kooskõlas.
- **Koormuse juhtimine:** kasutatakse mehhanisme ülekoormuse vältimiseks (nt **rate limiting, throttling**).
- **Veakäsitlus:** veateated on **standardiseeritud**, inimloetavad ja ei sisalda tundlikku sisemist infot.
- **Sisend, väljund:** API sisend ja väljund peab jälgima REST põhimõtteid (https://restfulapi.net/).
- **Protokoll, formaat:** kasutatakse HTTP protokolli ja API väljund on JSON, välja arvatud binaarfailide puhul.

4.3 Töökindlus ja monitooring

- **Health-checkid:** rakendus peab pakkuma **health-check** ja **readiness** liideseid(Spring Boot Actuator).
- **Observability:** rakendused edastavad minimaalselt request count, response time, error rate mõõdikuid Azure Monitori(Micrometer)
- **API töökindlus:** API peab olema kasutatav (vastab päringutele) ja enamus päringuid peab lõppema mõistliku aja jooksul ning ilma veata. Täpsemad jõudluse või töökindluse eesmärgid määratakse projektipõhiselt.
- **Testimine:** Integratsiooniteste ja end-to-end teste võib kirjutada. Nende puudumisel tuleb enne release-i teha regressioonitestimist käsitsi.

4.4 Koodistruktuur ja arendus

- **Struktuur:** Java projektides kasutatakse kihipõhist package struktuuri, kus komponendid on jaotatud package-tesse tehnilise rolli järgi (nt controllers, services, repositories).
- **Perioodilised tööd:** kõik **schedulerid, batchid ja stored procedures** dokumenteeritakse README-s. Perioodiliste tööde käivitamine eraldi klassis äriloogikast.
- **Automaatne andmete kustutamine:** kui süsteemis on andmeid, millel on määratud säilitustähtaja (vastavalt teabe liigitusskeemile), tuleb rakendada automaatne kustutamise funktsionaalsus. Automaatne kustutamine toimub perioodilise tööna (scheduler, batch), mis kontrollib andmete vanust ja kustutab andmeid, mis on ületanud säilitustähtaja. Kustutamise protsess peab olema logitud.
- **Teenusklassid:** äriloogika peab olema realiseeritud Service kihis, mille sisendiks ja väljundiks on domeeniobjektid. Teenuseklassi testideks on unit testid.
- **Kontrollerid:** Kontrollerite testid tuleb kirjutada kasutades MockMvc raamistikku, kus teenuste kihi vastused on mockitud.
- **Presentatsioonikiht:** presentatsioonikiht ei tohi kasutada sisendiks ja väljundiks domeeniobjekte, vaid eraldi Request ja Response ressursse.
- **Autoriseerimine:** päringute autoriseerimine peab toimuma kontrollerite kihis.
- **Testimine:** koodi unit testide haru-kattuvus(branch coverage) peab olema 80%. Unit testid tuleb jooksutada CI/CD süsteemis iga buildiga uuesti. Build peab katki minema, kui testid pole edukad. Repository (ja messaging) kihi testid tuleb kirjutada integratsioonitestidena vastu mockitud (nt rest apid) või konteineris jooksva päris välise süsteemi vastu (näiteks rakenduse Postgres baas TestContainers abil). Iga test hoolitseb ise oma testandmete loomise ja esialgse seisu taastamise eest. Erinevates testides samade andmete kasutamine ei ole lubatud.
- **Koodistandard:** vaikimisi kasutatakse Google Java koodistandardit, mille kontrollimine on osa buildist.

4.5 Integratsioon

- **Andmebaasid, välised liidesed:** andmebaasi ja teiste väliste andmete küsimine peab olema lahendatud eraldi repository kihis, mis väljastab äriloogika kihile ainult domeeniobjekte.

- **Sõnumid, eventid:** eventide saatmine ja vastuvõtmine peab olema realiseeritud äriloogikast eraldi consumerites ja producerites.

---

5. Logimine

Logimisnõuded on kirjeldatud dokumendis „Logimise spetsifikatsioon“.

- NB! Välisele arenduspartnerile väljastatakse “Logimise spetsifikatsioon” dokument peale hankelepingu sõlmimist.

---

6. Andmebaasi ja andmetöötluse nõuded

6.1 Üldpõhimõtted

- **Andmete terviklikkus:** kõik andmebaasi struktuurid (tabelid, seosed, indeksid) peavad tagama andmete terviklikkuse ja vältima duplikaate või orvuks jäänud kirjeid.
- **Normaliseeritus:** andmemudel peab olema vähemalt 3. normaalkujus või põhjendatult denormaliseeritud (näiteks jõudluse kaalutlustel).
- **Järjepidevus:** andmete formaat ja tähistus (nt kuupäevad, valuutad, staatuse väljad) peab olema ühtne kogu süsteemis.
- **Äriloogika:** relatsioonilisse baasi tuleb suhtuda kui andmete hoidlasse, äriloogika peab olema implementeeritud rakenduskihis. Stored procedure kasutamine ei ole lubatud.

6.2 Identifikaatorid ja audit

- **Primaarvõtmed:** numbrilised ID-d on eelistatud, mitte UUID, välja arvatud kui on tehniline vajadus globaalseks unikaalsuseks.
- **Auditväljad:** igas tabelis, kus toimub andmete muutmine, peavad olema väljad created_by, created_at, modified_by, modified_at. Ajatemplid (created_at, modified_at) on vajalikud ka andmete automaatseks kustutamiseks vanuse järgi.
- **Kustutuse logimine:** kõik andmete kustutamise operatsioonid (nii automaatsed kui käsitsi) peavad olema logitud koos kustutatud andmete tüübi, kustutamise ajaga, põhjuseks ja tegijaga (süsteem või kasutaja).
- **Auditlogi või event sourcing:** kasutatakse sõltuvalt süsteemi iseloomust ja ärivajadusest.

6.3 Platvorm ja tööriistad

- **Andmebaasi tarkvara:** RMK eelistatud andmebaas on **PostgreSQL**. Arenduse ja testimise eesmärgil võib kasutada ka mäluandmebaase (nt **H2**, **SQLite3**).
- **Migratsioonid:** kõik struktuurimuudatused hallatakse migratsioonitööriistadega ja versioonihalduses (soovitatavalt **Flyway**). **Käsitsi tehtud muudatused ei ole lubatud**, välja arvatud erandkorras ja RMK-ga eelnevalt kooskõlastatult. Kõik sellised muudatused tuleb dokumenteerida ja kanda tagasi migratsiooniskriptidesse, et tagada keskkondade ühtsus.

6.4 Turvalisus ja juurdepääs

- **Paroolid ja autentimisandmed:** salvestatakse ainult **soolatud räsi** kujul (nt Argon2, bcrypt, scrypt). Lihttekstina talletamine on keelatud.
- **Õiguste haldus:** andmebaasis kasutatakse eraldi kontosid rakenduse ja haldustööde jaoks (väikseima privileegi põhimõte).

---

7. Turvanõuded

7.1 Üldised põhimõtted

- **Autentimine ja identiteet:** rakendused ja kasutajad tuvastavad end RMK autentimisnõuete alusel (Entra ID, TARA väliskasutajatele).
- **Autoriseerimine:** juurdepääs rakendustes toimub rollipõhiselt. Rollide nimetusvorming: APP-{env}-{teenus}-{roll}.
- **Andmete liigitus:** rakenduse planeerimisel määratakse andmete tüüp ja liigitus, dokumenteeritakse nende käsitlemise alused. Säilitustähtajad määratakse vastavalt teabe liigitusskeemile Jira Assetis ja andmesäilituse teemapoliitikale.
- **Turvaline arendus:** rakenduse arendamisel järgitakse RMK dokumente *Arenduse elutsükkel* ja *Turvaline arhitektuur*.
- **Andmekaitse:** mitteavalikud andmed kaitstakse vastavalt teabe liigitusskeemile ja andmekaitse nõuetele. Töötlemisel järgitakse kehtivaid andmekaitse õigusakte, sh **GDPR**.
- **Andmevahetus:** kogu andmevahetus on krüpteeritud (TLS 1.2+); kasutatakse RMK krüptograafia juhendis määratud protokolle ja vorminguid.
- **Auditväljad ja -logi:** rakendustes logitakse, millal ja kes kirje lõi ning viimati muutis (audit trail).
- **Saladuste haldus:** kõik võtmed ja paroolid hoitakse **Azure Key Vault**-s, mitte lähtekoodis või konfiguratsioonifailides.
- **Keskkondade eraldatus:** live keskkond peab olema rangelt eraldatud arendus- ja testkeskkondadest. **Toodanguandmeid ei tohi kasutada arendus- ega testkeskkondades.**
- **Vajaduspõhine ligipääs:** kõik ligipääsud lähtuvad vajaduspõhimõttest (default deny).
- **Vähima privileegi põhimõte:** kõik protsessid töötavad vähimate vajalike õigustega.
- **Autentimise integreerimine:** kasutajate õiguseid arvestatakse läbivalt igal tasemel (UI, äriloogika, andmebaas, integratsioonid).

7.2 Sessioonid

- **Timeout:** serveripoolne sessiooni idle timeout ≤ 30 minutit, absoluutne timeout ≤ 12 tundi.

- **Sessiooni turvalisus:** sessiooni ID logitakse vajadusel maskitult või räsitult.
- **Sessiooni lõpetamine:** sessioon suletakse automaatselt, kui kasutaja logib välja või jõuab timeout.

7.3 Lähtekood ja arendusvahendid

- **Ligipääs:** lähtekood asub RMK Azure DevOps keskkonnas, kus ligipääsu hallatakse projekti põhiselt.
    - RMK arendajatele antakse ligipääs vajaduspõhiselt töölepingu kehtivuse ajal.
    - RMK töötajatele antakse ligipääs ainult vajaduse korral ja vaikimisi lugemisõigus.
    - Välistele arendajatele antakse projektipõhine ligipääs, mis eemaldatakse projekti lõppedes.
- **Lähtekoodi turvalisus:** tuleb vältida lubamatute funktsioonide lisamist ja tagada, et lähtekood, arendusvahendid ja teegid on kaitstud tahtmatute või pahatahtlike muudatuste eest.
- **Koodi kvaliteet:** turvavigade automaatne analüüs (nt Sonar).

---

8. Hooldatavus ja toetatavus

- **Koodistandardid:** rakenduse kood peab olema ühtses stiilis, loetav ja järgitav, et ka uus arendaja saaks töö kiiresti üle võtta.
- **Juhendite praktilisus:** dokumentatsioon peab olema piisavalt selge, et uuel arendajal või administraatoril oleks võimalik süsteem kiiresti üle võtta.
- **Testid:** rakenduse põhitöövood peavad olema kaetud automaatsete testidega. See võimaldab veenduda, et süsteemi põhifunktsioonid töötavad ka pärast muudatusi. Täpsemalt vt peatükk 9.
- ***Runbook*-d (käitamis- ja taastamisjuhendid):** kriitiliste töövoogude ja häireolukordade jaoks peab olema lühike samm-sammuline juhend, mis kirjeldab, kuidas probleem lahendada või süsteem taastada.
- **Tugi ja eskaleerimine:** peab olema selge, kes ja kuidas reageerib süsteemi tõrgetele ning millised on reageerimisajad erineva tõsidusega olukordades.

---

9. Testimine

- **CI/CD ja build:** unit- ja integratsioonitestid peavad jooksma igal ehitusel. Build peab ebaõnnestuma, kui testid ei ole edukad.
- **Peaharu:** main haru peab alati olema testitud ja valmis production keskkonda paigaldamiseks.

- **Testide kattuvus:** unit testide haru-kattuvus (branch coverage) peab vaikimisi olema 80%. Projekti- või hankepõhisel kokkuleppel võib tase olla madalam ning erand tuleb fikseerida kirjalikult.
- **Testide liigid ja kasutus:**
    - **Unit testid:** puhta loogika testimiseks, kus on konkeete sisend ja väljund. Unit testid ei sobi testideks, mille osaks on I/O, võrguga seotud funktsionaalsus või aeg.
    - **Integratsioonitestid:** katavad liideseid ja sõltuvusi(andmebaas, failisüsteem, MQ, REST liidesed jne)
    - **End-to-end testid:** katavad kasutaja töövooge (nt Playwright); vaikimisi katame E2E testidega ainult põhitöövood.
- **Testandmed:** iga test loob ise testandmed ja vajadusel taastab esialgse seisu; erinevates testides samu testandmeid kasutada ei tohi.
- **Integreerimis- ja E2E testid:** kui neid ei ole, tuleb enne release'i teha regressioonitestimine käsitsi.
- **Testkeskkonnad ja -andmed:** testimise käigus ei tohi kasutada toodanguandmeid arendus- ega testkeskkondades (vt 7.1).
- **Turva- ja jõudlustestid:** tehakse projekti ja vajaduspõhiselt.