TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Alexander Smirnov 232792IAIB

Daniel Tjulinov 233031IAIB

«Mis? Kus? Millal?» võistkondlike mänguturniiride töövoogude digitaliseerimise uurimine ja veebipõhise platvormi prototüübi kavandamine, realiseerimine ning hindamine

Bakalaureusetöö

Juhendaja: Siim Rebane

Külalisõppejõud

Tallinn 2026

Autorideklaratsioon

Kinnitame, et oleme koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autorid: Alexander Smirnov ja Daniel Tjulinov

18.05.2026

Lühikokkuvõte

Käesoleva bakalaureusetöö raames analüüsiti paberipõhise «Mis? Kus? Millal?» mälumängu turniirikorralduse pudelikaelu ning kavandati, realiseeriti ja valideeriti veebipõhine prototüüp, mis pakub digitaalset alternatiivi senistele manuaalsetele töövoogudele. Töö on tehtud koostöös tellijaga MIS? KUS? MILLAL? MTÜ-ga, mis on Eesti üks aktiivsemaid mälumänguturniiride korraldajaid alates 1991. aastast.

Loodud süsteem koosneb NestJS-põhisest serveriprotsessist, mis haldab mängu olekumasinat ja reaalajas suhtlust Socket.IO kaudu, ning Expo / React Native klientrakendusest, mis pakub ühest koodibaasist kasutajaliideseid veebibrauseritele ja iOS/Android seadmetele. Andmete püsivus on tagatud PostgreSQL andmebaasiga, lühiajalist mänguseisundit hoitakse Redises. Süsteem rakendab serveripoolset oleku-juhtimise (ingl *backend-driven state*) põhimõtet ning serveripoolset taimerit, mis tagab võistkondadele tehnilise pariteedi.

Süsteemi jõudlust ja kasutatavust hinnati reaalsetel testturniiridel. Mõõtmistulemused näitavad, et HTTP-otspunktide p99-viiteaeg jäi äriloogika operatsioonidel alla 50 millisekundi, WebSocketi-tasandil ei registreeritud turniiri vältel ühtegi ootamatut süsteemiviga. Mängijate keskmine 5-palline kasutuskogemuse hinnang oli 4,4. Paberipõhise ja digitaalse korraldusviisi võrdlus näitas, et rakenduse kasutuselevõtt vähendab korraldajate kumulatiivset töökoormust ligikaudu 82% (652 minutilt 116 minutile) ühe turniiri kohta.

Loodud platvorm on tellijale üle antud ja kasutusvalmis järgmistel turniiridel.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 59 leheküljel, 6 peatükki, 22 joonist, 26 tabelit.

Abstract

Research into the digitalization of “What? Where? When?” team game tournament workflows and the design, implementation, and evaluation of a web-based platform prototype

This bachelor’s thesis analyses the operational bottlenecks of paper-based “What? Where? When?” trivia tournament management and presents a web-based prototype that digitises the manual workflow. The work was carried out in cooperation with MIS? KUS? MILLAL? NGO, one of Estonia’s most active tournament organisers since 1991.

The implemented system consists of a NestJS-based server process that manages the game state machine and real-time communication via Socket.IO, and an Expo / React Native client application that provides user interfaces for web browsers and iOS/Android devices from a single codebase. Persistent data is stored in PostgreSQL while transient game state is held in Redis. The system applies a backend-driven state pattern and a server-side timer to guarantee technical parity between teams.

System performance and usability were evaluated at real test tournaments. Measurements show that the p99 latency of business-logic HTTP endpoints stayed below 50 milliseconds; no unexpected server-side errors were registered during the tournament at the WebSocket level. The average user-experience rating from the post-game survey was 4.4 / 5. A comparison between paper and digital tournament management revealed that the digital solution reduces cumulative organiser workload by approximately 82% per tournament (from 652 to 116 person-minutes).

The resulting platform has been delivered to the client and is ready for production use at upcoming tournaments.

The thesis is in Estonian and contains 59 pages of text, 6 chapters, 22 figures, 26 tables.

Lühendite ja mõistete sõnastik

ACID	andmebaasitehingute neli omadust (Atomicity, Consistency, Isolation, Durability), mis tagavad andmete terviklikkuse
Apellatsioon	võistkonna ametlik protest mängujuhi hindamisotsuse vastu, mida käsitletakse menetluskorras
API	rakendusliides (Application Programming Interface), mille kaudu tarkvarakomponendid omavahel suhtlevad
Backend-driven state	arhitektuuriline lähenemine, kus äriloogika tõeallikat hoiab server ja edastab klientidele struktuurseid olekuobjekte
BDUI	serveripoolne kasutajaliidese juhtimine (Backend-Driven UI) — arhitektuurimuster, mille korral server kirjeldab ka kasutajaliidese komponentide paigutuse ja stiili
CORS	brauseri turvamehhanism (Cross-Origin Resource Sharing), mis reguleerib päringute lubamist eri päritoluga domeenide vahel
Docker Compose	tööriist mitmest konteinerist koosnevate rakenduste defineerimiseks ja käivitamiseks ühe konfiguratsioonifaili kaudu
Edetabel	võistkondade pingerida, mis arvestab nii õigete vastuste arvu (score) kui ka küsimuste kaalupõhist reitingut
Expo	platvorm ja tööriistastik React Native rakenduste arendamiseks, mis võimaldab jagada koodi veebi ja mobiili vahel
GA4	Google Analytics 4 — Google'i veebianalüüsi platvormi versioon
GDPR	Euroopa Liidu isikuandmete kaitse üldmäärus (General Data Protection Regulation)
Grafana	avatud lähtekoodiga tööriist mõõdikute ja logide visualiseerimiseks armatuurlaudade kujul
Hetzner	Saksamaa pilveteenuse pakkuja, mille Helsinki andmekeskust kasutatakse süsteemi majutamiseks
HTTP	veebipõhise andmevahetuse rakenduskihi protokoll (Hypertext Transfer Protocol)
IDE	integreeritud arenduskeskkond (Integrated Development Environment)
JSON	kerge struktureeritud andmevahetusvorming (JavaScript Object Notation)
JSONB	PostgreSQL andmetüüp JSON-andmete binaarseks salvestamiseks
JWT	kompaktne digiallkirjastatud märgend kasutaja identiteedi ja õiguste edastamiseks (JSON Web Token)

pg_advisory_lock	PostgreSQL-i nõuandev lukustusmehhanism, mida kasutatakse rakendustasandi samaaegsuse kontrollimiseks
Mixpanel	tootekasutuse analüüsi platvorm, mida kasutatakse klientrakenduse kasutajategevuste mõõtmiseks
MTÜ	mittetulundusühing
Mängija	võistkonna liige, kes liitub mänguga pääsukoodi alusel ja esitab vastuseid
Mängujuht	turniiri korraldav administraator, kes loob mängu, esitab küsimusi ja hindab vastuseid
NestJS	TypeScripti-põhine moodulipõhine serveriraamistik, mis kasutab dekoraatoritel põhinevat sõltuvuste süstimist
Node.js	JavaScripti käituskeskkond serveripoolse koodi käivitamiseks
Olekumasin	lõpliku arvu olekute ja nende vaheliste üleminekutega formaalne mudel, mida kasutatakse mängu elutsükli ja küsimuse faaside haldamiseks
ORM	objektide ja relatsiooniliste andmebaasi tabelite vaheline vastendamine (Object-Relational Mapping)
PostgreSQL	avatud lähtekoodiga relatsiooniline andmebaasisüsteem, mida kasutatakse püsivate andmete salvestamiseks
Prisma	TypeScripti-põhine ORM-tööriist tüübiturvalise andmebaasipääsu ja migratsioonihalduse jaoks
Prometheus	avatud lähtekoodiga süsteem ajaridade kujul mõõdikute kogumiseks ja jälgimiseks
PWA	brauseripõhine rakendus (Progressive Web Application), mis pakub natiivse rakenduse sarnast kasutuskogemust
Pääsukood	neljakohaline kood, mille alusel mängija liitub konkreetse mänguga anonüümselt
React Native	raamistik mitmel platvormil töötavate rakenduste arendamiseks ühisest TypeScripti/JavaScripti koodibaasist
Redis	mälusisene võtme-väärtus andmebaas, mida kasutatakse vahemäluna ja lühiajalise mänguseisundi hoidmiseks
Reiting	edetabeli teisene näitaja: õigete vastuste kaalude summa, kus iga küsimuse kaal sõltub raskusastmest
REST	arhitektuuristiil veebiteenuste loomiseks HTTP-protokolli kaudu (Representational State Transfer)
RFC	internetistandardite ja -spetsifikatsioonide dokumendisari (Request for Comments)
SaaS	pilvepõhine tarkvarateenuse ärimudel (Software as a Service)
Skoor	edetabeli esmane näitaja: võistkonna õigesti vastatud küsimuste koguarv
Socket.IO	WebSocket-protokollil põhinev teek, mis lisab tubade, automaatse taasühendamise ja tagasilanguse-transpordi tuge
SQL	relatsiooniliste andmebaaside päringukeel (Structured Query Language)

SSE	protokoll ühesuunaliseks andmevooks serverist kliendile (Server-Sent Events)
TS	TypeScript — staatiliselt tüübitud JavaScripti laiendkeel
Tuba (room)	Socket.IO loogiline kanal, mis isoleerib ühe mängu sündmusi paralleelselt toimuvatest teistest mängudest
UI	kasutajaliides (User Interface)
URL	veebiressursi unikaalne aadress (Uniform Resource Locator)
Voor	turniiri struktuurne osa, mis koondab ühe temaatilise küsimuste komplekti
VPS	virtuaalne privaatserver (Virtual Private Server)
Võistkond	mängijate rühm, mis esitab ühise vastuse iga küsimuse kohta
WebSocket	kahesuunaline püsiühenduse protokoll kliendi ja serveri vahel reaalajas andmevahetuseks (RFC 6455)
Žürii	paberipõhises töövoos vastuseid käsitsi tabelitesse kandev korraldusmeeskond

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Intellektuaalsed meeskonnamängud, mille tuntuimaks esindajaks on «Mis? Kus? Millal?» formaat, on oluline mitteformaalse õppe ja intellektuaalse vaba aja veetmise viis, arendades osalejate analüütilist mõtlemist ning meeskonnatööd. Kuigi taolised mängud on Eestis ja rahvusvaheliselt äärmiselt populaarsed, on nende korralduslik pool jäänud valdavalt paberipõhiseks. Suuremahuliste turniiride läbiviimine nõuab märkimisväärset inimressurssi ning detailset logistilist planeerimist, mis tekitab korraldajatele suure administratiivse koormuse.

Enne käesoleva bakalaureusetööga alustamist mälumängukorraldajatega suhtlemisel selgus, et praegune töövoog tugineb suurele hulgale vabatahtlikele ehk «pääsukestele», kes koguvad vastuseid paberlipikutel, ning žüriile, kes peab need käsitsi tabelitesse kandma. See manuaalne protsess on vigadele vastuvõtlik, ei taga võistkondadele tehnilist pariteeti ning muudab protestide ja apellatsioonide lahendamise läbipaistmatuks. Tuntumad olemasolevad digitaalsed viktoriiniplatvormid ei ole aga antud probleemi lahendamiseks sobivad, kuna need on enamasti suunatud valikvastustele ega toeta mälumängu formaadile omast lühikeste vabatekstiliste vastuste käsitsi hindamist ja mängujuhi detailset kontrolli.

Käesoleva bakalaureusetöö peamine eesmärk on luua veebipõhise mälumängu-platvormi prototüüp, mis vähendab paberipõhise turniirikorraldusega kaasnevat inimressurssi ja võrdsustab tehnilised tingimused võistkondade vahel. Töö on tehtud koostöös tellijaga MIS? KUS? MILLAL? MTÜ-ga, mis on üks Eesti aktiivsemaid mälumänguturniiride korraldajaid alates 1991. aastast [1]. Tellija oli kogu arendustsüklis aktiivne osapool: pakkudes turniiripraktikast tulenevat domeeniteadmist, valideerides Figma-prototüüpe ning andes tagasisidet kolmel reaalsel testturniiril.

Peamise eesmärgi saavutamiseks püstitati järgmised alaeesmärgid:

- Kaardistada paberipõhise «Mis? Kus? Millal?» turniirikorralduse pudelikaelad ja tuletada nendest süsteemi nõuded (peatükk 2).

- Analüüsida olemasolevate digitaalsete viktoriiniplatvormide sobivust mälumängu spetsiifika jaoks (alapeatükk 2.3).
- Kavandada arhitektuur, mis toetab reaalajas toimuvat sünkroonset andmevahetust ja tagab võistkondadele tehnilise pariteedi (peatükk 3).
- Realiseerida prototüüp, mis katab võistkonna ja mängujuhi tuumik-töövood (peatükk 4).
- Hinnata loodud platvormi jõudlust, kasutatavust ja mõõdetavat mõju paberipõhisele korraldusele reaalsel testturniiril (peatükk 5).

Töö lõpptulemusena valmib tsentraliseeritud prototüüp, mille tehnilised valikud — React Native [2] kliendi-pool, Node.js [3] ja NestJS [4] serveri-pool, PostgreSQL [5] püsivaks salvestuseks ja WebSocket reaalajas-kanaliks — moodustavad ühtse TypeScripti-põhise ökosüsteemi. Süsteemi mõju mõõdetakse paberipõhise turniiriga võrdluses ning ootuste järgi väheneb korraldajate kumulatiivne inimtöökoormus turniiri kohta märkimisväärselt.

Bakalaureusetöö on jaotatud kuueks peatükiks. Teises peatükis analüüsitakse probleemi olemust ja olemasolevat manuaalset töövoogu, antakse ülevaade sarnastest lahendustest ning defineeritakse loodava süsteemi nõuded. Kolmas peatükk keskendub lahenduse väljatöötamisel kasutatud metoodikale ja süsteemi arhitektuurile. Neljandas peatükis antakse ülevaade rakenduse prototüübi tehnilisest realiseerimisest. Viiendas peatükis esitatakse tulemused ja analüüs ning töö viimases, kuuendas peatükis tehakse saavutatud tulemustest kokkuvõte.

Käesolev bakalaureusetöö ja sellega seotud rakendus on valminud kahe üliõpilase — Daniel Tjulinovi ja Alexander Smirnovi — ühistööna. Autorite individuaalne panus arendusse ja töö kirjalikku ossa on kirjeldatud alapeatükis 5.6.

2 Taust ja analüüs

Käesolevas peatükis antakse põhjalik ülevaade mälumänguturniiride praegusest korraldus-likust olukorrast, kaardistatakse olemasoleva käsitsi hallatava töövoo kriitilised probleemid ning esitatakse analüüs turul pakutavate alternatiivide kohta. Peatüki lõpus defineeritakse uue süsteemi loomiseks vajalikud funktsionaalsed ja mittefunktsionaalsed nõuded, mis põhinevad eelnevalt kaardistatud vajadustel.

2.1 Praeguse protsessi analüüs

Mälumängu «Mis? Kus? Millal?» traditsiooniline läbiviimine kooliturniiridel põhineb peamiselt paberpõhistel ja käsitsi sooritatavatel tegevustel, mis järgivad Rahvusvahelise Klubiühingute Liidu (MAIIK) kehtestatud spordikoodeksit [6].

Käesolevas alapeatükis esitatud kvantitatiivsed andmed põhinevad töö autorite poolt 13. veebruaril 2026 läbi viidud vaatlusel reaalselt toimunud kooliturniiri käigus. Autorid viibisid turniiri ajal kohal ja fikseerisid stopperiga iga küsimuse puhul eraldi neli ajavahemikku: küsimuse lugemise kestus, mõtlemis- ja vastamisaeg, vastuste füüsilise kogumise kestus ning vastuse häälega teatamise aeg. Vaatlusobjektiks oli 13 võistkonnaga turniir, mis koosnes kolmest voorust (kokku 36 küsimust). Kogutud mõõteandmed kanti hiljem tabelarvutusprogrammi ning kasutati ka alapeatükis 5.3 esitatud võrdlusanalüüsis paberipõhise ja digipõhise mängu kestuste kõrvutamiseks.

Praegune manuaalne töövoog koosneb neljast peamisest etapist:

- **Ettevalmistus:** Võistkondade registreerimine toimub staatiliste tabelarvutusprogrammide või paberankeetide kaudu, mis muudab andmete haldamise ja hilisema statistika loomise keeruliseks. Suureks logistiliseks väljakutseks on turniiri teenindava personali leidmine, mille suurus sõltub otseselt osalevate võistkondade arvust. Keskmisel turniiril osaleb tavaliselt 10–20 võistkonda, suuremate puhul kuni 30; väga erandlikel juhtudel ulatub osalejate arv 40–60 võistkonnani, mil on vaja kokku

panna lausa 15–20 liikmest koosnev vabatahtlike (nn «pääsukeste») ja žüriiliikmete meeskond, kuid sellise mahuga turniire korraldatakse harva. Lisaks nõuab paberipõhine formaat spetsiaalsete vastuslipikute füüsilist ettevalmistamist ja trükkimist, kuhu on eelnevalt märgitud voorud, küsimuste numbrid ja iga osaleva võistkonna nimi, mis on iga turniiri eel märkimisväärselt aja- ja ressursikulukas protsess.

- **Mängu kulg:** Mängujuht esitab küsimuse sünkroonselt kõigile saalis viibivatele võistkondadele. Pärast küsimuse esitamist algab rangelt piiratud aruteluaeg, milleks on ette nähtud täpselt 60 sekundit. Sellele järgneb lühike, 10 sekundi pikkune lisaviivitus vastuse lõplikuks fikseerimiseks paberile. Selline range ajaraamistik loob väga intensiivse ja katkematu mängurütmi, kus igasugused korralduslikud viivitused järgnevates etappides häirivad tugevalt turniiri sujuvust.
- **Vastuste kogumine:** Võistkonnad kirjutavad oma lõplikud vastused paberilehtedele. Saalis liikuvad vabatahtlikud ehk «pääsukesed» peavad lipikud laudade vahel joostes kokku korjama ja füüsiliselt žüriile toimetama. Turniiride reaalajas läbiviidud mõõtmised näitavad, et vastuste füüsiliseks kogumiseks kulub iga küsimuse järel keskmiselt 40 sekundit, kusjuures viivituste korral ulatub see aeg ligi minutini (kuni 56 sekundit). See etapp on kriitiline, kuna rikub võistkondade vahelist tehnilist pariteeti — «pääsukesed» ei suuda jõuda kõikide laudadeni täpselt samal sekundil, andes kaugematele laudadele ebaõiglaselt lisaaega vastuse viimistlemiseks.
- **Hindamine ja tabeli pidamine:** Žürii peab esitatud vastuseid kontrollima käsitsi ja sisestama tulemused lokaalsesse Exceli tabelisse. See on äärmiselt veaohtlik ja koormav etapp, kuna isegi keskmise suurusega turniiril tuleb iga küsimuse järel lühikese aja jooksul töödelda kümneid paberlipikuid. Suuremate turniiride puhul ajakulu ja vigade tekkimise tõenäosus paratamatult mitmekordistub. Kuna žürii ei jõua manuaalse andmetöötluse tõttu vastuseid alati jooksvalt mängutempoga sünkroonis hinnata, tekib vajadus tehniliste pauside järele. Turniiride reaalajas läbiviidud mõõtmiste põhjal tekivad isegi suhteliselt väikese, 13 võistkonnaga turniiri puhul voorude vahele ligikaudu 2-minutilised pausid tulemuste kokkuvõtmiseks ja järelehindamiseks, millele lisandub mängu lõpus umbes 10-minutiline paus lõppjärjestuse arvutamiseks. Kokku tekitab käsitsi tabeli pidamine turniiri jooksul 10–14 minutit puhast ooteaega, mis venitab kogu sündmuse kestust. Lisaks puudub paberpõhise hindamise korral võimalus tagada reaalajas läbipaistvat protestide haldust

ja statistika edastamist publikule.

2.2 Probleemi olemus

Käsitsi läbiviidava protsessi analüüsimisel ilmnevad selged pudelikaelad, mille peamisteks murekohtadeks on ebaefektiivsus, suur ressursikulu ja läbipaistmatus. Peamised tuvastatud kitsaskohad manuaalses töövoolus on järgmised:

- **Suur personalivajadus ja inimressursi kulu:** Turniiri läbiviimiseks on vaja suurt hulka abilisi, et tagada vastuste operatiivne kogumine laudade vahel. Standardne nõue on tagada ligikaudu üks vabatahtlik iga viie meeskonna kohta. Turniiride reaalajas läbiviidud mõõtmised näitavad, et isegi väikese, 13 võistkonnaga turniiri teenindamiseks kulus 5-liikmelisel meeskonnal (kaks vastuste kogujat ja kolm žüriiliiget) turniiri peale kokku ligikaudu 11 tundi ehk 652 minutit reaalset tööaega. Suuremate, üle 30 võistkonnaga turniiride puhul nõuab see juba kuni 15–20 vabatahtliku ja žüriiliikme kaasamist, mis on korraldajatele keeruline logistiline väljakutse.
- **Ajakulu ja tehnilise pariteedi rikkumine:** Vastuste füüsiline kogumine laudade vahel on aeglane. Andmeanalüüsi põhjal kulub ühe küsimuse vastuste kogumiseks keskmiselt 40 sekundit, ulatudes viivituste korral isegi 56 sekundini. See rikub otseselt võistkondade vahelist tehnilist pariteeti, kuna vastuste kogujad ei suuda kõikide laudadeni jõuda täpselt samal sekundil, andes kaugematele laudadele ebaõiglaselt lisaaega. Samuti tekitab käsitsi punktiarvestus pikki viivitusi — näiteks kulub voorude vahel tulemuste arvutamiseks sageli 10–14 minutit pause, mis venitab kogu sündmuse kestust.
- **Andmete kadumise risk ja inimfaktorist tingitud vead:** Paberkandjal vastused võivad sünkroonse ja kiire kogumise käigus laudade vahel kaduda või segamini minna. Žürii peab lühikese aja jooksul sisestama sadade lipikute andmed käsitsi tabelarvutusprogrammi (nt Excel), mis on äärmiselt veaohtlik protsess. Lisaks teeb paberipõhine formaat apellatsioonide lahendamise läbipaistmatuks — vaidlusi lahendatakse sageli mitteametlikult ja tagantjärele on algandmeid raske objektiivselt verifitseerida.
- **Reaalaja tagasiside ja statistika puudumine:** Osalejad ei näe oma vastuste staatust (kas žürii luges vastuse õigeks või valeks) ega jooksvat edetabelit enne, kui žürii

on kõik andmed käsitsi tabelisse kandnud ja tulemused ametlikult välja kuulutanud. Samuti puudub publikul dünaamiline ülevaade turniiri kulgemisest. Kogu varasem statistika ja mängutulemused on paberarhiivides hajutatud. Peale sisestusvigade riski tähendab lokaalsete arvutustabelite kasutamine seda, et manuaalne meetod ei võimalda pidada mängijate ja võistkondade ühtset ajaloolist statistikat (näiteks linna või riigi mastaabis), kuna andmed jäävad lukustatuks erinevate korraldajate eraldiseisvatesse failidesse.

2.3 Olemasolevate lahenduste võrdlus

Eelmises alapeatükis kaardistatud manuaalse töövoo pudelikaelade ja läbipaistmatuse likvideerimiseks on loomulik samm pöörduda digitaalsete platvormide poole. Digitaalsel turul leidubki mitmeid populaarseid reaalajas mängitavaid viktoriiniplatvorme, nagu Kahoot [7] ja Wayground [8], publiku kaasamise ja esitlustööriistu, nagu Mentimeter [9] ja Slido [10], staatilisi andmekogumisvahendeid, nagu Google Forms [11], ning ka spetsiifilisi mälumängudele suunatud tehnoloogilisi ökosüsteeme. Nende lahenduste lähem analüüs aga kinnitab, et valmiskujul pakutav ei kata täielikult «Mis? Kus? Millal?» kooliturniiride spetsiifilisi vajadusi, mistõttu jäävad mitmed kriitilised probleemid lahendamata. Olemasolevate lahenduste puudused antud töö kontekstis võib jagada nelja peamisesse kategooriasse:

Üldotstarbeliste platvormide fookus automaatsele hindamisele. Tuntumad hariduslikud platvormid ja interaktiivsed esitlustööriistad on loodud meelelahutusliku tempo või publiku tagasiside kogumise eesmärgil, keskendudes valikvastustega küsimustele ja automatiseeritud punktiarvestusele. Vaadeldav mälumängu formaat nõuab aga lühikesi vabatekstilisi vastuseid, mis võivad sisaldada sünonüüme, trükivigu või mitmetimõistetavusi. Nimetatud platvormidel puudub liides, mis võimaldaks mängujuhil sünkroonselt laekuvaid kümneid tekstivastuseid ühel ekraanil mugavalt läbi vaadata ning neid operatiivselt ja käsitsi õigeks, valeks või vaieldavaks märkida.

Staatiliste andmekogumisvahendite asünkroonsus. Sageli kaalutakse turniiride läbiviimiseks tasuta ankeedilahendusi, nagu näiteks Google Forms. Nende suurimaks puuduseks on aga asünkroonsus ja sünkroniseeritud reaalaja funktsionaalsuse puudumine. Need platvormid ei võimalda rakendada tsentraalselt juhitud 60 sekundi aruteluaja ja 10 sekundi

vastamise taimerit, mis lukustaks vastuste esitamise automaatselt aja lõppemisel. Samuti puudub mängujuhil reaalajas automaatselt uuenev ülevaade laekuvatest vastustest.

Mängujuhi piiratud kontroll ja protestide süsteemi puudumine. «Mis? Kus? Millal?» formaat tugineb rangetele ajalistele piirangutele ja õigluse tagamisele. Nii interaktiivsed rakendused kui ka staatilised vormid pakuvad mängujuhile piiratud kontrolli, mitte võimaldades konfigureerida formaadile omast spetsiifilist kahefaasilist taimerit. Üheks kriitilisimaks puuduseks kõikide eelnevalt mainitud platvormide puhul on sisseehitatud tagasisideahela puudumine: meeskondadel ei ole võimalik esitada struktureeritud apellatsioone otsese seosega konkreetse küsimuse ja nende antud vastuse vahel.

Spetsiifiliste mälumängulahenduste killustatus. Tuleb märkida, et professionaalsel mälumänguturul eksisteerib ka spetsiifilisi tehnoloogilisi lahendusi. Telegrami ja Discordi botid on aga suunatud pigem asünkroonsele treeningule või individuaalsetele harrastusmängudele. Detailsemad veebiplatvormid (nagu Open-Quiz [12]) toetavad küll klassikalisi mälumängureglemente, kuid on sageli suunatud mastaapsetele rahvusvahelistele sünkroonturniiridele, nõudes korraldajatelt spetsiifilist tehnilist ettevalmistust — näiteks abiprogrammide (nagu ChgkSuite) kasutamist JSON-formaadis küsimustefailide genereerimiseks. Selliste platvormide arhitektuur ei paku piisavalt kergekaalulist ja intuitiivset kogemust kohalike kooliturniiride vabatahtlikele korraldajatele ja õpilastele.

Eeltoodud nelja kategooriasse jagatud puuduste konkreetsemaks võrdluseks on Lisas 2 (Joonis 15) süstematiseeritud kõige enam levinud platvormide ja käesoleva töö raames välja töötatud prototüübi tugi «Mis? Kus? Millal?» formaadile omastele kriitilistele kriteeriumitele. Võrdlus kinnitab, et ükski analüüsitud platvorm ei kata samaaegselt kõiki nelja eelnevalt kirjeldatud probleemikategooriat — eelkõige vabatekstiliste vastuste käsitsi valideerimist, struktureeritud apellatsioonide menetlust ning kahefaasilist sünkroonset taimerit.

Eelnevate puudujääkide analüüsist järeldub, et ükski turul olev valmislahendus ei suuda täielikult ja valutult asendada senist paberipõhist logistikat. Seetõttu on antud bakalaureusetöö raames põhjendatud spetsiifilise erilahenduse loomine, mis ühendab endas sünkroniseeritud vastuste kogumise, käsitsi valideerimise funktsionaalsuse ja struktureeritud protestide haldamise mugavas kasutajaliideses.

2.4 Süsteemi nõuded

Loodava erilahenduse edukaks realiseerimiseks koostati mälumängu korraldajatega peetud arutelude ja eelneva analüüsi põhjal detailne nõuete spetsifikatsioon. Eesmärgipärase ja töökindla tarkvara loomiseks jagati süsteemile esitatavad ootused kaheks: funktsionaalseteks ja mittefunktsionaalseteks nõueteks.

2.4.1 Funktsionaalsed nõuded

Funktsionaalsed nõuded defineerivad konkreetsed teenused ja käitumismustrid, mida platvorm peab pakkuma kahele peamisele osapoolele: võistkondadele ning mängujuhile koos žüriiga.

Võistkondade moodul (kliendirakendus): Kasutajaliides peab võimaldama võistkondadel mugavalt liituda spetsiifilise mängutoaga, kasutades selleks unikaalset PIN-koodi ja võistkonna nime. Kliendirakenduse keskseks elemendiks on vastuse sisestamise liides, mis sisaldab mängujuhi seadmega sünkroniseeritud dünaamilist taimerit. Ranges ajaraamistikus (näiteks 60 sekundit aruteluks ja 10 sekundit esitamiseks) peab süsteem vastuse pärast aja lõppemist automaatselt lukustama ja esitama, välistades sellega ajapiirangu ületamise. Mängu vältel peab rakendus andma meeskonnale reaalajas ülevaate mängu faasidest (arutelu, vastuse saatmine, ooterežiim), esitatud vastuste staatusest ja jooksva edetabeli seisust.

Mängujuhi ja žürii moodul (halduspaneel): Mängujuhil peab olema täielik kontroll turniiri sisu, küsimuste, voorude ja mänguvoolu üle. Reaalajas läbiviimise käigus peab mängujuht saama käivitada ja peatada taimerit ning otsustada, millist informatsiooni ja millal (nt jooksev edetabel, õiged vastused) mängijatele ning publikule kuvatakse. Žürii koormuse vähendamiseks on kriitilise tähtsusega tsentraalse hindamislaua olemasolu. Sinna peavad laekuma sünkroonselt kõikide võistkondade vastused, mida žürii saab ühe nupuvajutusega märkida kas «õigeks», «valeks» või «vaieldavaks». Seejuures peab süsteem võimaldama ka apellatsioonide kiiret haldamist otseses seoses vaidlusaluse küsimusega. Suuremate turniiride puhul peab halduspaneel toetama mitme kasutaja paralleelset tööd, et turniiri läbiviimise käigus saaksid erinevad korraldusmeeskonna liikmed samaaegselt täita oma ülesandeid üksteist segamata.

2.4.2 Mittefunktsionaalsed nõuded

Mittefunktsionaalsed nõuded on aluseks süsteemi stabiilsusele, kiirusele ja kasutusmugavusele, tagades turniiride tõrgeteta läbiviimise ka intensiivse andmevahetuse tingimustes.

- **Sünkroniseerimine ja madal latentsus:** Eelnevalt kirjeldatud tehnilise pariteedi tagamiseks peab andmete edastus mängujuhi ja kõikide võistkondade vahel toimuma absoluutses reaalajas. Viiteaeg seadmete vahel ei tohi ületada ühte sekundit. Selle saavutamiseks rakendatakse kahepoolset andmesidet võimaldavat WebSocket tehnoloogiat.
- **Andmete säilivus:** Platvorm peab olema vastupidav ootamatutele tehnilistele tõrgetele. Süsteem peab tagama, et võistkondade sisestatud vastused, varasemalt saadetud tulemused ja jooksev skoor säiliksid andmebaasis ka siis, kui kasutaja võrguühendus ajutiselt katkeb või rakenduse lehte kogemata värskendatakse.
- **Skaleeritavus ja jõudlus:** Arvestades suurturniiride mahtu ja intensiivsust, peab arhitektuur olema loodud varuga. Süsteemi peamiseks koormusühikuks on võistkonna seadme-ühendus: iga võistkond kasutab turniiril ühte aktiivset rakenduse-instantsi (üks osaleja sisestab vastuse võistkonna nimel). Platvorm peab suutma stabiilselt toetada vähemalt 100 võistkonda samaaegselt ühe turniiri raames, ilma et süsteemi reageerimiskiiruses esineks märgatavat langust. Mainitud 100 võistkonna sihtväärtus on tellijaga teadlikult seatud turvavaruga tulevasi kasutusjuhtude jaoks (näiteks klubidevahelised või rahvusvahelised turniirid); tellija käesolevas turniiripraktikas jääb tüüpiline maht 10–30 võistkonna piiresse, erandjuhtudel kuni 60 (vt 2.1). Saalis viibivate kasutajate koguarv ($\approx$500 inimest 5-liikmeliste võistkondade puhul) ei mõjuta otseselt serveri koormust, kuna avalik vaade publikule on planeeritud edasiarenduses (vt 5.5).
- **Ristplatvormsus ja ergonoomika (UX):** Rakendus peab pakkuma adaptiivset kasutajaliidest, mis on veatult kasutatav nii kaasaegsetes veebibrauserites, tahvelarvutites kui ka mobiilsetes seadmetes (iOS/Android). Kasutajakogemus peab olema intuitiivne, et vältida pingelises mänguolukorras inimfaktorist tingitud tehnilisi eksimusi.
- **Mitmekeelsus:** Kuna turniire korraldatakse sageli mitmekeelses keskkonnas, peab süsteem toetama lokaalsuse haldamist ning võimaldama kiiret keelevahetust (esialgu

eesti, vene ja inglise keele vahel).

- **Õiguslik vastavus ja andmekaitse:** Rakendus peab vastama Eesti Vabariigi ja Euroopa Liidu kehtivale seadusandlusele. Arendusprotsessis järgitakse isikuandmete kaitse üldmääruse (GDPR) põhimõtteid, tagades kasutajate andmete turvalise hoiustamise, andmete töötlemise minimaalsuse ning osalejate privaatsuse kaitse.

3 Metoodika ja arhitektuur

Käesolevas peatükis kirjeldatakse süsteemi tehnoloogilist baasi, valitud raamistike põhjendusi ning lahenduse arhitektuurilist ülesehitust. Eesmärk on luua tehniline ülevaade, mis toetab reaalajas toimuvat andmevahetust ning tagab süsteemi stabiilsuse suure hulga samaaegsete kasutajate korral.

3.1 Tehnoloogiavalikud

Süsteemi arendamisel lähtuti vajadusest luua reaalajas reageeriv ja mitmel platvormil toimiv lahendus, mis ühtlasi võimaldaks bakalaureusetöö ajaraamis kiiret iteratiivset prototüüpimist. Valitud tehnoloogiad moodustavad ühtse ökosüsteemi, mis lubab koodi taaskasutust kliendi ja serveri vahel ning tagab tõhusa andme- ja seisundihalduse. Igas alapeatükis on kirjeldatud nii valitud lahendust kui ka kaalutud alternatiive ja nende kõrvalejätmise põhjuseid.

3.1.1 Klientrakenduse tehnoloogiad

Klientrakenduse arendamiseks valiti React Native [2] koos Expo platvormiga [13], mis võimaldavad jagada ühte TypeScripti koodibaasi veebibrauseri ja iOS-/Android-i natiivsete rakenduste vahel. Valikut ajendab kaks mälumängu turniiri kontekstist tulenevat nõuet: liitumine ei tohi eeldada eelnevat installeerimist (mille katab Expo veebiversioon), kuid samal ajal peab säilima võimalus kasutada hiljem seadme natiivseid võimekusi.

Alternatiividena kaaluti kahte natiivset rakendust (Swift ja Kotlin), Flutterit [14] ning brauseripõhist PWA-d. Kahe natiivse koodibaasi paralleelne arendamine oleks bakalaureusetöö ajaraamis ebaproportsionaalne, kuna iga muudatus tuleks realiseerida kahekordselt. Flutter põhineb Darti keelel [15] ega võimalda andmemudelite tüüpide jagamist TypeScripti-põhise serveriga, mis sunniks samad andmevahetuslepingud defineerima kahes erinevas keeles. Brauseripõhine PWA oleks installeerimisvajaduse kõrvaldanud, kuid piiraks juurdepääsu seadme võimekustele, mille olulisus on välja toodud allpool. React Native ja Expo

kombinatsioon ühendab nende variantide tugevused: ühtse TS-koodibaasi, brauseripõhise tarbimise ilma installeerimiseta ning juurdepääsu natiivsele platvormile. Sarnaste cross-platform-lähenemiste süstemaatiline võrdlus, mis sisaldab nii brauseripõhiste PWA-de kui natiivsete pakettide jõudluse- ja võimekuse-mõõtmist, on esitatud Biørn-Hanseni jt töös [16].

Mälumängu kontekstis on edaspidi olulised järgmised natiivsed funktsionaalsused, mille usaldusväärsus brauseris on piiratud:

- ekraani aktiivsena hoidmine ühe küsimuse kuni minuti pikkuse arutelu jaoks (Web Screen Wake Lock API tugi iOS Safaris on ebaühtlane [17], natiivse rakenduse kaudu on lahendus platvormiülene);
- haptiline ja heliline tagasiside taimeri lõpu lähenemise kohta, mille brauseripoolsed API-d on rangete kasutaja-eelduste taga;
- süsteemsed teavitused turniiri faaside ja apellatsioonide kohta ka taustal töötamise korral — iOS-i veebiteavitused on saadaval alates versioonist 16.4, kuid eeldavad PWA eelnevat avalehele lisamist [18], mis pole turniiri kontekstis realistlik;
- püsiv WebSocketi-ühendus taustal, mille mobiilsed brauserid sageli sulgevad.

Käesolev prototüüp ei kasuta veel ühtegi nimetatud natiivsetest võimekustest — turniiri tuumikfunktsionaalsuse realiseerimine oli prioriteet, kuid React Native ja Expo platvormi valik hoiab tee neile lahti, ilma et selleks oleks vaja klientrakenduse arhitektuurimuudatust. Üksikasjalikum natiivse versiooni töövoog on kirjeldatud alapeatükis 5.5.

3.1.2 Serveri tehnoloogiad

Serveri poolel on kasutusel Node.js käituskeskkond [3] ja NestJS raamistik [4]. NestJS valiti selle moodulipõhise ülesehituse, dekoraatoritel põhineva sõltuvuste süstimise (ingl *Dependency Injection*) ja TypeScripti tugeva toe tõttu. Erinevalt madalama tasemega Node.js raamistikest (Express [19], Fastify [20]), mis ei kehtesta rakenduse struktuurile rangeid nõudeid, pakub NestJS terviklikku tuge nii REST-liidese kui WebSocket-põhise sündmusvahetuse jaoks ühe ja sama mooduli- ja dekoraatorisüsteemi raames — seega pole vaja kahe paradigma jaoks eraldi abstraktsioonikihte. Server-poolse keelena kaaluti ka Java Spring Booti [21], kuid otsustavaks sai võimalus jagada andmemudeleid ja

sündmuselepingute tüüpe kliendi ja serveri vahel ühises TypeScripti pinus — see vähendab integratsioonivigu ja lühendab arendustsüklit. Täiendavaks teguriks oli arendusmeeskonna eelnev kogemus TypeScripti ökosüsteemiga, mis lühendas õppimiskõverat.

Andmebaasiga suhtlemiseks kasutatakse Prisma ORM-i [22], mis tagab tüübiturvalise päringukirjutamise, automaatse skeemi tuletamise ja deklaratiivse migratsioonihalduse. Alternatiividena kaaluti TypeORM-i, mis jäi piiratud arendusaktiivsuse ja nõrgemate tüübigarantiide tõttu kõrvale, ning vahetut pg-draiverit, mis oleks nõudnud märkimisväärset hulka korduvkoodi iga päringu jaoks.

3.1.3 Andmebaasilahendus

Püsivate andmete salvestamiseks valiti PostgreSQL relatsiooniline andmebaasisüsteem [5]. Süsteemi andmemudel on selgepiiriliselt struktureeritud — mängud, voorud, küsimused, kategooriad, võistkonnad, osalejad ja vastused on omavahel rangete seostega ühendatud — mistõttu relatsiooniline lähenemine sobib paremini kui dokumendipõhine (nt MongoDB), kus seoste kontroll oleks rakenduse-poolse koodi vastutus. Lisaks nõuab hindamise auditjälje säilitamine ACID-vastavust, mis dokumendipõhistes süsteemides eeldab täiendavaid kompromisse.

Konkreetse relatsioonilise süsteemi valikul kaaluti SQLite-i, MySQL-i ja PostgreSQL-i. SQLite jäi kõrvale ühe kirjutamissoone tõttu, mis ei sobi mitme samaaegse mängujuhi ja kümnete võistkondade teenindamiseks. MySQL-iga võrreldes pakub PostgreSQL töö jaoks olulisi täiendavaid mehhanisme — nõuandvad lukud (pg_advisory_lock), RETURNING-klauslid ja rikkalikum tüübisüsteem (sh JSONB mängijate tagasiside salvestamiseks), mida käesolevas töös on rakendatud andmete terviklikkuse ja samaaegsuse haldamise mehhanismidena.

3.1.4 Vahemälu ja seansiseisund

Reaalajas reageerimist nõudvate andmete jaoks on kasutusel Redis vahemälusüsteem [23]. Redises hoitakse mängu lühiajalist jooksvat seisundit — käimasoleva küsimuse identifikaatorit, faasi olekut, taimeri lõpetamise ajatemplit ja seotud abimuutujaid. Alternatiividena kaaluti seisundi hoidmist ainult serveriprotsessi mälus või PostgreSQL-is sagedase pärimisega. Esimene jätaks seisundi serveri taaskäivitusel kaduma, mis turniiri

ajal pole vastuvõetav; teine tähendaks märgatavat lisakoormust kettapõhisele andmebaasile, kuna sekundi täpsusega uuenevat infot kirjutataks püsivasse mällu. Redis pakub vahepealse lahenduse: kiire mälu-tasemel juurdepääs koos järjepidevusega serveri taaskäivituste vahel.

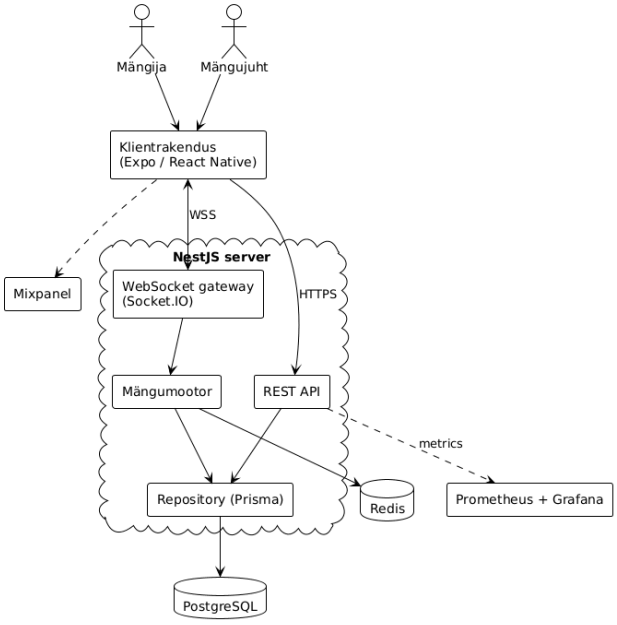
3.1.5 Reaalajasuhtlus

Serveri ja klientide vaheliseks reaalajas andmevahetuseks kasutatakse Socket.IO teeki [24], mis põhineb WebSocket-protokollil (RFC 6455 [25]). Alternatiividena kaaluti pikka küsitlust (ingl *long polling*), serverist saadetavaid sündmusi (ingl *Server-Sent Events*, SSE) [26] ning vahetut WebSocketi kasutamist ilma kõrgema taseme teegita. Pikk küsitlus on ressursimahukas pidevate uute HTTP-ühenduste loomise tõttu ja halvasti sobiv reaalajas mängu jaoks. SSE on ühepoolne ega toeta vastuste saatmist serverile, mis on käesolevas süsteemis vajalik. Vahetu WebSocketi kasutamine annaks väikseima andmesidekoormuse, kuid eeldaks omakirjutatud loogikat automaatse taasühendamise, sõnumite mänguruumipõhise marsruutimise ja vanemate brauserite tagasilanguse jaoks.

Socket.IO pakub neid funktsionaalsusi valmislahendusena: automaatset taasühendamist, tubade (ingl *rooms*) kontseptsiooni mänguruumide isoleerimiseks ning agnostilist transporti, mis vajadusel langeb tagasi pikale küsitlusele. Tubade mehhanism on kriitilise tähtsusega funktsionaalse nõude täitmiseks, et ühe mängu sündmused ei lekiks paralleelselt toimuvasse teise mängu. WebSocket-kanali kaudu saab server edastada sündmusi kõigile osalejatele samaaegselt, tagades võistkondadele võrdsed tingimused ajakriitilistes faasides.

3.2 Süsteemi arhitektuur

Süsteem on üles ehitatud moodulipõhise kihilise arhitektuuri põhimõtetel, kus rollid ja vastutusalad on selgelt jaotatud loogiliste moodulite vahel. Selline ülesehitus võimaldab arendada, testida ja edasi arendada süsteemi osi üksteisest sõltumatult, samal ajal kui kogu mängu loogika koondub ühte serveriprotsessi, mis tagab andmete järjepidevuse. Joonisel 1 on esitatud süsteemi komponendidiagramm, mis kujutab kliendirakenduse, serveriprotsessi ning välise andmebaasi- ja vahemälusüsteemi vahelist suhtlust.



Joonis 1. Süsteemi komponendidiagramm.

3.2.1 Üldine ülesehitus

Süsteem koosneb neljast komponendist: keskselt paiknev NestJS-põhine serveriprotsess, ühe koodibaasiga klientrakendus, mis pakub erinevaid vaateid sõltuvalt kasutaja rollist, PostgreSQL andmebaas püsivate andmete jaoks ning Redis vahemälusüsteem reaalajas mänguseisundi jaoks. Klientrakendus suhtleb serveriga REST-liidese kaudu seadistustegevuste tarvis ja WebSocketi-protokolli kaudu reaalajas sündmuste vahetamiseks. Server haldab andmete püsivust läbi Prisma ORM-i ning vahetab vahemällu kantud seisundiandmeid Redisega ioredis-i kliendi vahendusel.

Klientrakendusel eristatakse kahte peamist kasutajarolli: mängija (võistkonna liige) ja mängujuht (administraator). Mõlemad rollid suhtlevad sama serveriga, kuid neile avatakse erinevad funktsionaalsed moodulid, andmevood ja kasutajaliidese marsruudid. Selline ülesehitus võimaldab hoida turniiri loogikat tsentraalselt ühes tõeallikas, tagades muu hulgas, et mängijad ei näe informatsiooni (nt õigeid vastuseid või teiste võistkondade hindamise hetkeseisu) enne selleks ettenähtud aega.

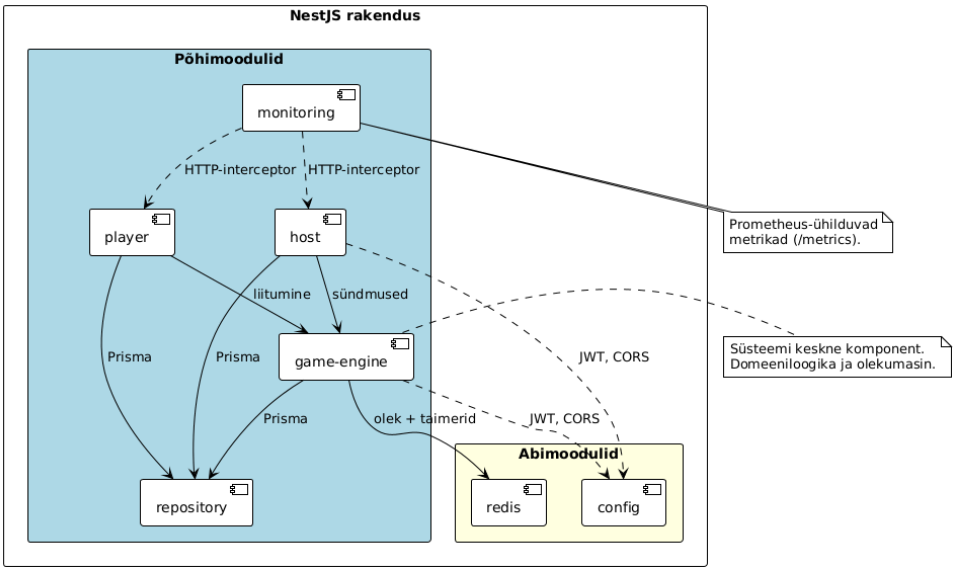
3.2.2 Serveri moodulistruktuur

Serveriprotsess on jaotatud loogilisteks NestJS mooduliteks, mis on rühmitatud rolli- ja vastutusala põhiselt. Peamised moodulid on:

- host — mängujuhi REST-liidesed mängude haldamiseks (loomine, redigeerimine, eksport) ning autentimine ja autoriseerimine JWT-märgendite kaudu;
- player — mängija avalikud REST-liidesed (pääsukoodi kontroll, võistkondade nimekiri) ning anonüümse liitumise tugi;
- game-engine — mängu elutsükli ja olekute üleminekute haldamine, faaside ja taimerite juhtimine ning kõigi reaalajaks vajalike WebSocketi-sündmuste edastamine;
- repository — andmete püsivuskiht, mis kapseldab Prisma päringud ning teostab valideerimise ja samaaegsuse kontrolli;
- monitoring — Prometheus-ühilduvate jõudluse metrikate kogumine, mille üksikasjad on kirjeldatud alapeatükis 3.2.5.

Mängumootor (game-engine) on süsteemi keskne komponent, mis ei sõltu konkreetsest kasutajaliidese realisatsioonist ega edastusprotokollist — ta töötab puhtalt domeeniobjektidel ja olekuüleminekutel. See võimaldab moodulit testida üksikult ning teoreetiliselt asendada kasutajaliidese tehnoloogia ilma tuumloogikat puudutamata.

Lisaks toetavad nimetatud peamooduleid kaks abimoodulit: redis (Redise kliendi konfigureerimine ja jagamine kõikide moodulite vahel) ja config (jagatud JWT-konfiguratsioon ning CORS-i päritoluseaded, mis hoiab ära seadete kõrvalekaldumise erinevates moodulites). Joonisel 2 on kujutatud moodulite vahelisi sõltuvusi.



Joonis 2. Serveri moodulite sõltuvused.

Mängumootori ühendusvaba ülesehitus on aluseks ka võimalikule edaspidisele üleminekule mikroteenusarhitektuurile, mida käsitletakse alapeatükis 5.5.

3.2.3 REST API vs WebSocketi sündmused

Suhtlus kliendi ja serveri vahel toimub hübriidselt: kasutatakse paralleelselt nii REST-liidest kui WebSocketi sündmustepõhist suhtlust, kuid kummalgi on selgelt määratletud roll.

Klientrakenduse ja serveri vahelise vastutuse jaotuse kirjeldamise alused panid paika REST-stiili kirjeldused [27], mille edasiarendusena on praktikas levinud kolmetasandiline taksonoomia:

- kliendipoolne juhtimine — server tagastab puhtad andmed (nt küsimuste loendi), kogu kuvamis- ja olekuloogika elab kliendis;
- serveripoolne oleku juhtimine (ingl *backend-driven state*) — server hoiab äriloogika tõeallikat ja edastab struktuurseid olekuobjekte, kuid kuvamise üksikasjad jäävad kliendile;
- serveripoolne kasutajaliidese juhtimine (ingl *backend-driven UI*, BDUI) — server kirjeldab ka kasutajaliidese komponentide paigutuse ja stiili.

Käesolevas süsteemis valiti serveripoolne oleku juhtimine kõikide ajakriitiliste voogude jaoks (mängu olekumasin, taimer, edetabel). Server edastab klientidele struktuurseid andmeobjekte mängu hetkeseisu kohta — aktiivne faas, järelejäänud sekundid, käimasoleva küsimuse identifikaator, edetabel ning klient otsustab ise, kuidas neid kasutajaliideses kuvada. Selline lähenemine toimib käesolevas töös läbiva arhitektuurilise põhimõttena: alapeatükkides korduvalt põhjendatakse, miks äriloogika otsustamine jäi serverisse, mitte ei viidud kliendile.

Alternatiivina kaaluti BDUI-d, kus server kirjeldaks ka kasutajaliidese komponentide paigutuse. Selline lähenemine oleks lihtsustanud klientrakendust, kuid sidunud serveri tarbetult tihedalt kasutajaliidese detailidega ning muutnud iga UI-muudatuse keeruliseks (eeldaks nii kliendi kui serveri samaaegset uuendust). Kliendipoolne juhtimine omakorda oleks lubanud kahel sakil ühe ja sama mängu kohta erinevat olekut näha, mis on turniirikeskkonnas vastuvõetamatu. Backend-driven state lähenemine ühendab nende kahe äärmuse tugevused.

3.2.4 Autentimine ja autoriseerimine

Süsteemis on kasutusel kaks erinevat autentimismudelit, mis tulenevad mängujuhi ja mängija rollide erinevatest vajadustest.

Mängujuhid tuvastatakse JSON Web Token (JWT) märgenditega [28]. Märgend väljastatakse pärast e-posti ja parooli kontrollimist ning saadetakse iga järgneva REST-päringu Authorization-päises ja iga WebSocket-ühenduse handshake.auth.token-väljas. Mängujuhi rolle (HOST, ADMIN, SCORER) kontrollib täiendav RolesGuard. Selline mehhanism võimaldab säilitada serveri olemuselt seansita (stateless), kuna märgend sisaldab kasutaja identiteeti ja rolle.

Mängijad liituvad süsteemiga anonüümselt, sisestades neljakohalise pääsukoodi. Kasutajatelt ei nõuta registreerimist ega sisselogimist, mis vähendab liitumiskünnist ja on kooskõlas turniiri lühiajalise iseloomuga. Kasutajakonto-põhise autentimise asemel rakendatakse sokli ja osaleja sidumist: pärast edukat liitumist seotakse võistkonna identifikaator konkreetse Socket.IO-ühendusega ning iga järgnev mängija sündmus valideeritakse selle sidumise alusel.

3.2.5 Vaadeldavus ja telemeetria

Kuna süsteem teenindab paljusid samaaegseid mängijaid ja võistkondi reaalajas, on töökindluse hindamiseks ja kitsaskohtade tuvastamiseks vajalik kogu süsteemi käitumine pidevalt mõõdetav. Käesolevas töös on rakendatud Prometheus-põhine metrikate kogumise pinu [29], mis hõlmab kolme tasandi mõõtmist:

- Rakendustasand. Serveripool on instrumenteeritud prom-client teegiga ning katab kolme kategooria mõõdikut. HTTP-päringute metrikad kogutakse globaalse HttpMetricsInterceptor-i kaudu, mis hõlmab kõiki REST-otspunkte automaatselt. WebSocketi-tasandi metrikad on kogutud Socket.IO gateway tasandil: aktiivsete ühenduste arv, vastuvõetud ja saadetud sündmuste loendurid sündmuste kaupa, käitleja kestus histogrammina ning vigade loendur. Lisaks kogutakse prom-client standardmetrikate komplekt — Node.js heap, garbage collector, event-loop lag, aktiivsete handle'ite arv, mis võimaldavad jälgida käituskeskkonna seisundit.
- Andmekihi tasand. PostgreSQL ja Redis on monitooritavad postgres-exporter ja redis-

exporter teenuste kaudu, mis pakuvad andmebaasi sisemisi metrikad (ühenduste arv, päringute viiteaeg, mälukasutus, võtmete arv) Prometheusele tarbitavas vormingus.

- Süsteemitasand. Tööriist node-exporter kogub serverihosti metrikad — protsessori, mälu, võrgu ja kettainfo, mis on olulised koormuskatsete tõlgendamiseks.
- Kogutud andmed visualiseeritakse Grafana dashboardide abil. Töö raames on koostatud kuus dashboardi: süsteemi üldine seisund, HTTP-päringute jaotus otspunktide kaupa, WebSocketi metrikad, Postgresi sisemised näitajad, Redise näitajad ning serveri host-tasandi näitajad. Telemeetriastack käivitub eraldi konfiguratsiooniga docker compose kaudu ning saab andmeid Prometheusest, mille korjatihedus on 5 sekundit.

Alternatiividena kaaluti SaaS-tüüpi monitooringuteenuseid (Datadog, New Relic) ning OpenTelemetry kollektoriga lahendust. SaaS jäi kõrvale käesoleva töö eelarveraamis tekkida võiva sõltuvuse tõttu välisest teenusepakkujast. OpenTelemetry on perspektiivne valik, mis võimaldaks ühtlustada metrikate, jälgede ja logide kogumise üheainsa kollektori taha, kuid eeldas hetkel rohkem konfiguratsioonitööd kui Prometheusel põhinev minimaalne seadistus; üleminek OpenTelemetry-le on planeeritud edasiarendusena.

3.2.6 Klientrakenduse telemeetria

Serveripoolse mõõtmise kõrval kogub klientrakendus oma kasutusandmed Mixpaneli teenuse [30] kaudu. Klient on instrumenteeritud nii, et iga oluline kasutajategevus (avalehe nuppude vajutamine, võistkonna valimine, vastuse esitamine, mängujuhi hindamisotsus, taimeri pausimine, ekraanide vahetamine) saadetakse Mixpaneli kasutusanalüüsi-serverisse koos kontekstuaalsete omadustega — rakenduse versioon, platvorm (web, ios, android), kasutajaroll, mängu identifikaator, parasjagu nähtud ekraani nimi. Iga sündmuse kogumiseks kasutatakse partii-päringu-mehhanismi, mis kogub kuni 20 sündmust kuni 1,5 sekundi jooksul ühte päringusse — see vähendab võrgukoormust mängu ajal, kus kümned sündmused saabuvad sekundi täpsusega. Andmete kogumise serveri-otspunktiks on valitud Mixpaneli Euroopa Liidu instants, mis on oluline andmekaitse-vastavuse seisukohast: kasutajate käitumisandmed ei lahku Euroopa Majanduspiirkonnast. Anonüümne identifikaator hoitakse natiivsel platvormil turvalises mälus, veebis aga seansi-mälus, mis välistab üheainsa identifikaatori jagamise mitme avatud saki vahel. Sündmuste täielik nimekiri on toodud klientrakenduse repositooriumis ja dubleeritud Lisas 3. Mixpaneli valikul kaaluti

alternatiividena Google Analyticsit (GA4), Amplitude'i ja PostHogi. GA4 jäi kõrvale, kuna selle andmemudel on optimeeritud veebianalüüsi jaoks ning eeldaks rakenduse kohandatud sündmuste tuletamist; Amplitude pakub sarnast funktsionaalsust, kuid Euroopa-instants oli valiku ajal saadaval ainult ettevõtte-paketis; PostHog (avatud lähtekoodiga, ise-hostitav alternatiiv) on perspektiivne, kuid eeldaks lisaks olemasolevale infrastruktuurile ühte täiendavat haldatavat teenust.

3.2.7 Operatiivkeskkond

Kogu süsteem — rakendusserver, PostgreSQL, Redis ning telemeetriapinu — käivitatakse Hetzneri pilve VPS-instantsis, mille andmekeskus paikneb Soomes (Helsinki). Hetzner valiti kolme kriteeriumi alusel: hinna ja jõudluse suhe (võrreldav konfiguratsioon on AWS-il ja Google Cloudis prototüübimaht-arvestuses 3–5 korda kallim), andmekeskuse asukoht Eesti kasutajatele lähedases regioonis (latentsus Tallinnast Soomesse jääb tüüpiliselt alla 10 ms), ning teenusepakkuja vastavus Euroopa Liidu andmekaitse seadusandlusele (GDPR), mis on oluline, kuna süsteem salvestab mängijate tagasisidet ja mängujuhtide e-postiaadresse. Kaaluti ka mitmeid alternatiive: serverivabad lahendused (AWS Lambda, Cloudflare Workers) jäid kõrvale, kuna käesolev süsteem hoiab pikaajalisi WebSocket-ühendusi, mis ei sobi serverivabade funktsioonide lühikese tööaja-mudeliga ning eeldaks alternatiivse väljapesakanali (ingl *sticky sessions*) seadistamist; juhitud platvormid-kui-teenus (Heroku, Render) jäid kõrvale prototüübifaasi eelarveraamis. Kogu rakenduspinu on koondatud docker compose konfiguratsioonifaili, mis võimaldab arenduskeskkonna ja tootmiskeskkonna ühtset käivitamist ühe käsuga.

Pideva integreerimise ja paigalduse (ingl *Continuous Integration / Continuous Deployment*, CI/CD) jaoks on seadistatud GitHub Actions automaatne töövoog, mis käivitub iga koodimuudatuse korral peaharusse. Töövoog koosneb kahest järjestikusest etapist: esimeses sammus käivitatakse staatiline analüüs (TypeScripti tüübikontroll, ESLint), automaattestid (NestJS rakenduse üksusetestid Jest-i abil) ning kontrollitakse, et Prisma andmemudel oleks sünkroonis migratsioonifailidega; teises sammus ehitatakse rakenduse Dockeri tõmmis, lükatakse see Hetzneri serverisse ning käivitatakse uuesti docker compose abil, ilma teenuse käsitsi taaskäivituseta. Selline lähenemine tagab, et iga peaharusse jõudev muudatus on automaatselt valideeritud ning et tootmiskeskkonda jõudvad ainult need versioonid, mis on edukalt läbinud koodikvaliteedi kontrolli. CI/CD-protsess kahandab ka inimliku eksimuse

riski paigalduses, kuna tootmiskeskkonna seadistamise sammud on kõik kirjeldatud koodina (ingl *infrastructure as code*) versioonihalduses.

3.3 Andmemudel ja seisundi haldamine

Käesolevas peatükis kirjeldatakse süsteemi andmemudelit — nii püsivat relatsioonilist andmemudelit, mis kajastub PostgreSQL andmebaasi skeemis, kui ka mängu lühiajalist olekut, mida hallatakse Redises ja serveriprotsessi mälus.

Süsteemi püsivat andmemudelit kirjeldab esitatud olemite-seoste diagramm (ingl *Entity-Relationship Diagram*, ERD) ning skeem on toodud Lisas 4.

3.3.1 Põhitabelite ja seoste kirjeldus

Andmemudel on jaotatud kuueks loogiliseks rühmaks. Järgnevalt kirjeldatakse iga rühma keskseid tabeleid ja nende vahelisi seoseid.

Tabel users salvestab mängujuhtide andmed (e-post, parooli kontrollsumma, loomisaeg) ning viitab tabelile roles. Kasutusel on kolm rolli — HOST, ADMIN ja SCORER —, millest prototüübi tuumikvood toetuvad HOST-rollile; ADMIN ja SCORER on andmemudelis ette valmistatud süsteemi haldustoiminguteks ja hindamise delegeerimiseks. Selline ülesehitus võimaldab piirata juurdepääsu rolli põhiselt. Mängijaid eraldi kasutajatabelisse ei salvestata, kuna nende seos süsteemiga on lühiajaline ja anonüümne.

Mängu seadistus. Mängu peamine olem on *games*, mis sisaldab nii üldist konfiguratsiooni (nimi, kuupäev, vaikimisi mõtlemis- ja vastamisaeg), režiimilippe (show_answer, show_leaderboard, can_appeal) kui ka jooksvat olekut (status, modified_at). Igal mängul on üks korraldav mängujuht. Mäng jaguneb voorudeks, mille puhul on kehtestatud unikaalsuse piirang (game_id, round_number). Iga voor sisaldab küsimusi, millel on samuti unikaalne järjekord (round_id, question_number) ning kus iga küsimusele saab seada eraldi mõtlemis- ja vastamisaja, mis kirjutab üle mängu vaikeväärtused. Kategooriad on mängujuhi kontoga seotud korduvkasutatavad ressursid (nt «Vanemad», «Algajad»); nende seostamine konkreetse mänguga toimub vahetabeli category_game_relations kaudu, mis võimaldab sama kategooriat kasutada paljudes mängudes.

Võistkonnad ja mänguosalemine. Tabel *teams* esindab mängujuhi kontoga seotud korduvkasutatavaid võistkondi. Konkreetse mänguga seotud osalus on modelleeritud eraldi tabelis game_participants, kus iga rida kujutab endast ühte kombinatsiooni «võistkond × kategooria» konkreetses mängus. Tabelil on unikaalne piirang (game_id, team_id), mis välistab sama võistkonna mitmekordse osalemise samas mängus. Tabeli game_participants veerg socket_id salvestab aktiivse WebSocket-ühenduse identifikaatori, mis on aluseks sokli ja osaleja sidumise kontrollile. Veerg is_available näitab, kas võistkonna koht on hetkel vaba liitumiseks. Selline kombineeritud lähenemine — püsivate andmete (võistkond, kategooria) ja jooksva oleku (sokli sidumeed, saadavus) hoidmine ühes tabelis — on kompromiss, mis vähendab päringute hulka mängu liidese kuvamisel, kuid eeldab pärast mängu lõppu jooksvate väljade tühjendamist.

Vastused ja hindamine. Iga vastus salvestatakse tabelisse *answers* ning seda iseloomustab unikaalne piirang (game_participant_id, question_id), mis tagab, et üks võistkond saab esitada igale küsimusele täpselt ühe vastuse — kordusesitused kirjutatakse üle. Vastuse hindamise olek viitab eraldi tabelile answer_statuses, mis sisaldab kolme põhistaatust (UNSET, CORRECT, INCORRECT) ja täiendava staatuse DISPUTABLE apellatsiooniga seotud vastuste märgistamiseks. Veerg late_by_seconds jäädvustab, kui palju aega oli vastamise hetkel ületatud — see võimaldab eristada hilinenud, kuid akna piires saadetud vastuseid täielikult tagasilükatutest. Iga hindamise muudatus salvestatakse tabelisse answer_status_history, mis säilitab nii vana kui uue staatuse, muudatuse autori ja ajatempli — see audit-jälg võimaldab vajadusel rekonstrueerida ühe vastuse hindamise kulgu.

Apellatsioonid on andmemudelis ette valmistatud apellatsioonide menetluseks; nende kasutuselevõtt jäi prototüübi skoobist välja.

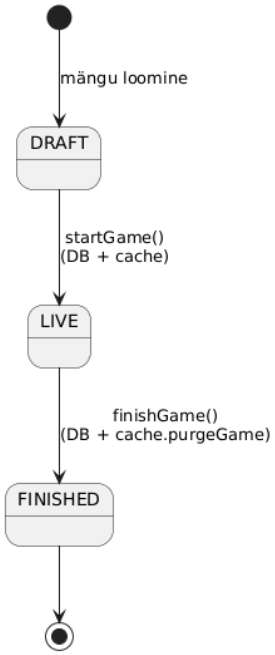
Mängijate tagasiside. Tabel player_app_feedback salvestab mängijate poolt edastatud anonüümset rakenduse tagasisidet JSON-vormingus väljal payload. Seos mängu ja osalejaga on valikuline, mis võimaldab tagasisidet säilitada ka pärast mängu või osaluse kustutamist analüüsi otstarbel.

3.3.2 Mängu olekumasin

Mängu seisundit hallatakse kahetasandilise olekumasinaga. Esimene tase kirjeldab mängu elutsüklit, mis kujutab kogu turniiri progressi, ning teine tase kirjeldab käimasoleva küsi-

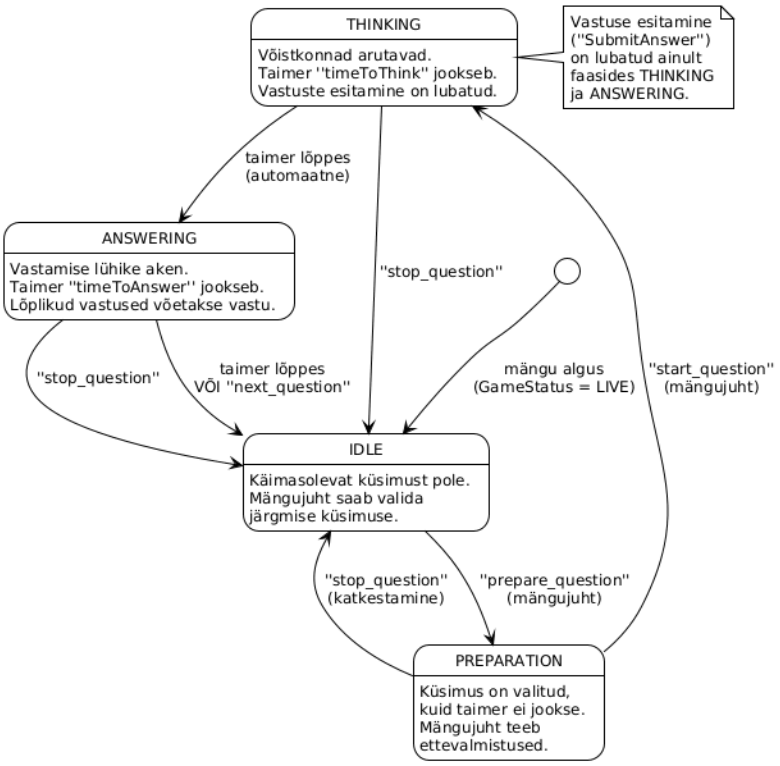
muse faasi, mille hierarhilise jaotuse kontseptsioon põhineb Hareli statechart-formalismil [31].

Mängu elutsükli olekumasin koosneb kolmest olekust: DRAFT (mäng on koostamisel, küsimusi ja kategooriaid saab redigeerida), LIVE (mäng on alanud ja jooksvalt mängitakse) ning FINISHED (mäng on lõppenud, andmed on muutmatud). Üleminekud on ühesuunalised ja toimuvad mängujuhi käsuga. Tagasipöördumine LIVE olekust DRAFT olekusse on välistatud, et tagada andmete järjepidevus ja vältida võistkondade vahepeal salvestatud vastuste kaotamist. Olekumasinat illustreerib Joonis 3.



Joonis 3. Mängu elutsükli olekumasin.

Küsimuse faasi olekumasin on aktiivne ainult mängu LIVE olekus ning koosneb neljast faasist: IDLE (käimasolevat küsimust ei ole, mängujuht saab valida järgmise), PREPARATION (küsimus on valitud, kuid taimer veel ei jookse — mängujuht teeb ettevalmistused), THINKING (võistkonnad arutavad küsimust, taimer jookseb) ning ANSWERING (vastamise lühike aken, mille jooksul on lubatud saata lõplik vastus). Üleminekud käivitatakse mängujuhi sündmustega ja taimeri-põhiste sündmustega. Lisaks on lubatud käsitsi jätkamine. Faasi olekumasinat illustreerib Joonis 4.



Joonis 4. Küsimuse faasi olekumasin.

Olekumasina range struktureerimine on aluseks ka serveripoolsele valideerimisele. Näiteks vastust saab esitada ainult faasides THINKING ja ANSWERING, mängu hinnata võib ainult pärast ANSWERING faasi lõppemist, ning apellatsioone saab algatada ainult juhul, kui vastav lipp on mängul lubatud. Kogu üleminekuloogika on koondatud game-engine moodulisse, mis on ainus sisemise oleku muutuste allikas.

3.3.3 Püsiva oleku ja vahemälu jaotus

Süsteemi olekut hoitakse kolmel tasandil, igaühel oma rolliga, mille esitab Tabel 1. Selline jaotus vähendab koormust andmebaasile, säilitades samal ajal andmete püsivuse ja taastatavuse.

Tabel 1. Püsiva oleku ja vahemälu jaotus.

Tasand	**Mida hoitakse**	**Põhjendus**
PostgreSQL	Mängu seadistus (games, rounds, questions, categories); osalejad (game_participants); vastused ja nende staatuste ajalugu (answers, answer_status_history); apellatsioonid (disputes); kasutajad ja rollid (users, roles). Veerg games.status kajastab mängu jooksvat elutsükliolekut, kuid see on samaaegselt vahemälustatud Redises kiireks kättesaamiseks.	Andmed peavad üle elama serveri taaskäivitused ning säilitama auditeeritava ajaloo.
Redis	Käimasoleva küsimuse faas (game:{id}:phase); mängu staatus (game:{id}:status); aktiivse küsimuse andmed (game:{id}:activeQuestionData); faasi lõppemise ajatempel (game:{id}:phaseDeadlineTs); pausi kogusekundid (game:{id}:pausedSeconds); küsimuse tähtaeg koos TTL-iga (question:{id}:deadlineTs).	Andmed muutuvad sekundi täpsusega ja neid päritakse sageli. Kettapõhine kirjutamine raiskaks ressurssi; in-memory kaotaks andmed taaskäivitusel.
InMemory	setInterval-l põhinevad faasi taimerid (activeTimers); sokli sündmuste tagasikutsefunktsioonid (tickCallbacks, phaseChangeCallbacks).	Funktsiooniviited eksisteerivad ainult konkreetses Node.js protsessis ja viitavad otse aktiivsele Socket.IO seansile. Nende viimine välisesse vahemällu eeldaks Redise Pub/Sub-i.

Selline kolmetasandiline jaotus tähendab, et mängu jooksva oleku osa läheb serveri taaskäivitusel kaotsi (in-memory taimerid), kuid kriitiline osa (faas, deadline, status) säilib Redises ja on taastatav.

4 Rakenduse prototüübi realiseerimine

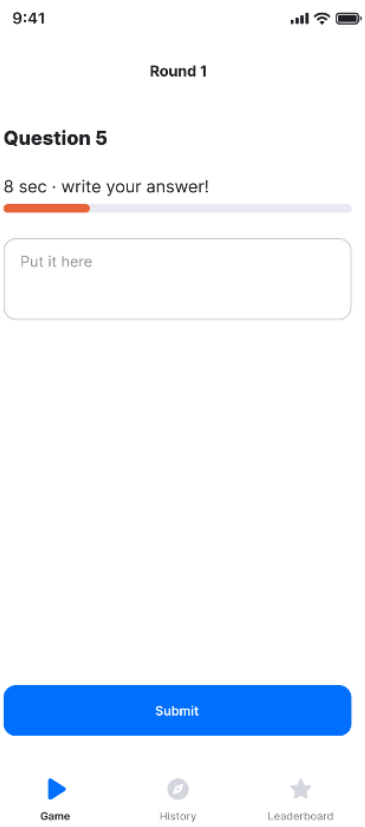
Käesolev peatükk kirjeldab prototüübi kasutaja vaatest ja realiseerimise vaatest.

4.1 Kasutajaliidese kujundamise põhimõtted

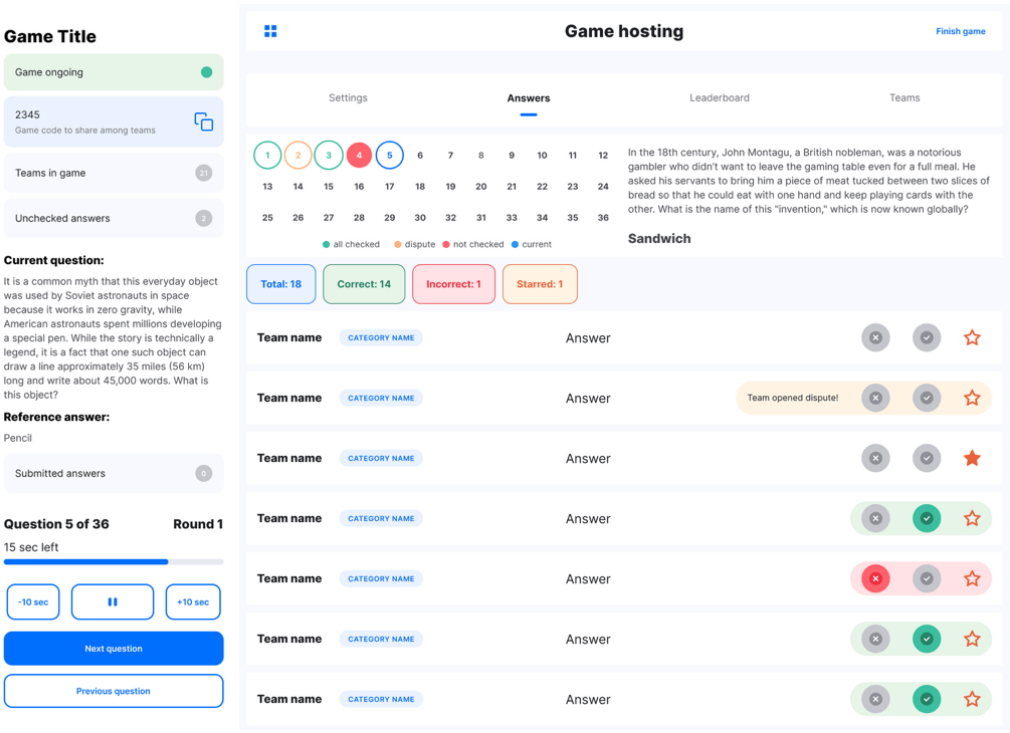
Käesolevas alapeatükis kirjeldatakse kasutajaliidese kujundamise lähenemist kahest aspektist: disainiprotsessi ja koostööd tellijaga. Alapeatüki eesmärk on tutvustada metodoloogilist konteksti, milles järgnevad realiseerimise alapeatükid tekkisid.

4.1.1 Disainiprotsess ja koostöö tellijaga

Süsteemi arendamise esimene etapp oli kasutajavoogude kavandamine Figma-prototüübis [32]. Tellijaga toimusid prototüübi kallal regulaarsed iteratsioonivoorud Figma kommenteerimisrežiimis, mille tulemusena saavutati enne koodi-arenduse algust kasutajaliidese põhimõtteline kokkulepe. Visuaalne keel — värvipalett, tüpograafia ja kontrast — on koondatud Expo süsteemse teema mehhanismi alla, mis võimaldab tulevikus üle minna kas hele- või tumeteemale ühe konfiguratsioonifaili muutmisega. Joonisel 5 ja Joonisel 6 on toodud vastavalt mängija ja mängujuhi vaate Figma-prototüübid.



Joonis 5. Mängija aktiivne mängu ekraan.



Joonis 6. Mängujuhi aktiivse mängu vastuste ekraan.

4.1.2 Klientrakenduse jagamine kahe rolli vahel ühes koodibaasis

Klientrakenduse koodibaasis eristatakse kahte kasutajarolli, kuid mõlemad rollid jagavad ühte projekti, ühte sõltuvuste komplekti ja ühte ehitusprotsessi. See on saavutatud Expo Routeri failipõhise marsruutimise (ingl *file-based routing*) kaudu, kus kausta struktuur määrab rakenduse navigatsioonipuu.

Rakenduse marsruudid on jaotatud kahte rühma:

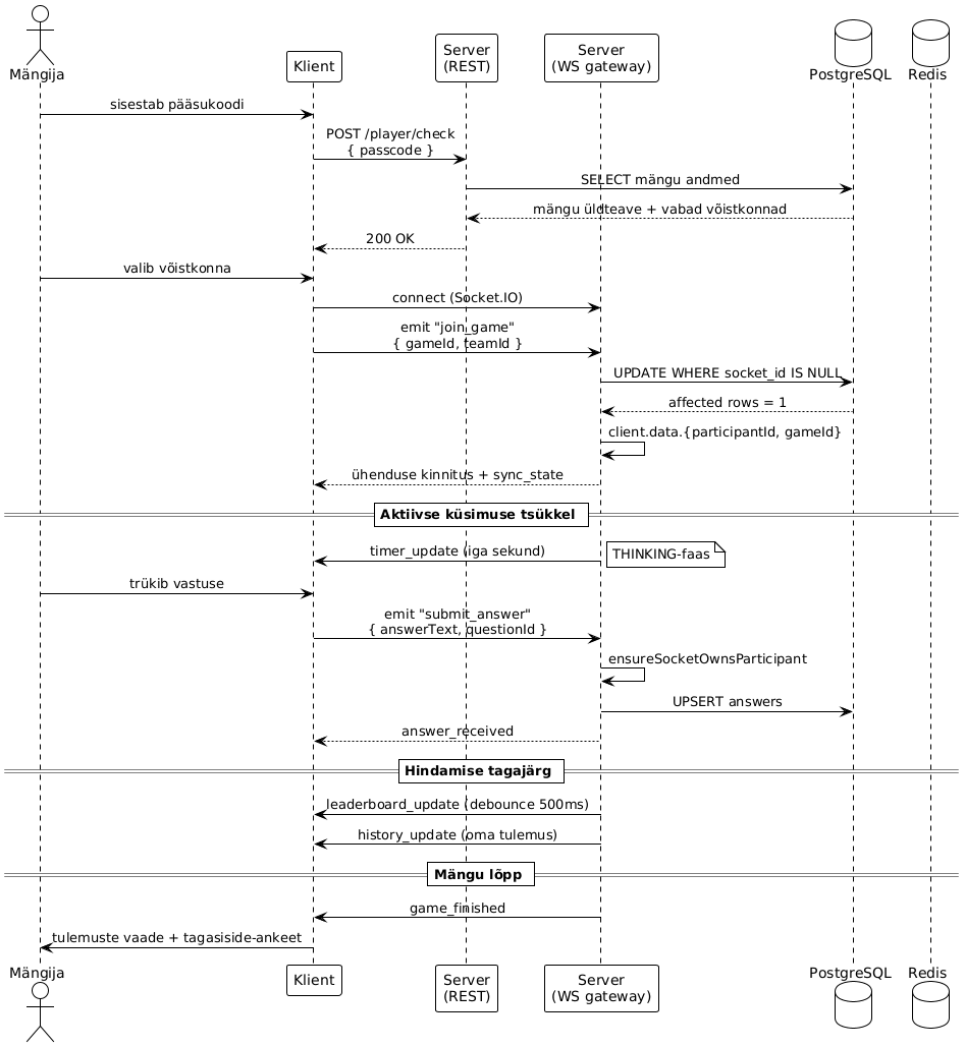
- (player)/. . . — mängija ekraanid (pääsukoodi sisestamine, võistkonna valik, küsimuse vaade, tulemuste vaade). Need ekraanid on optimeeritud puuteseadmetel ühe käega kasutamiseks; navigatsiooniriba on minimaalne või täielikult peidetud.
- (host)/. . . — mängujuhi ekraanid (mängude haldus, redaktor, käimasoleva mängu juhtpaneel, vastuste hindamine). Need ekraanid eeldavad suuremat kuvapinda ja kasutavad sakkide-paneelide paigutust.

Mõlema rühma jagatud komponendid (näiteks taimer, võistkonna nimekiri, edetabel), API-kliendi vood, tüübi-definitsioonid ning Socket.IO sündmuste lepingud asuvad ühtses jagatud kataloogis. See vähendab koodi dubleerimist ja tagab, et serveripoolse muudatuse tulemusena uuendub mõlema rolli klient ühtaegu.

Marsruutimise loogiline pool — kuhu kasutaja peale liitumist suunatakse — toimub kahel viisil. Mängujuht logib sisse sisselogimise vormiga ning sissevõetud JWT-märgendi alusel suunatakse (host)/. . . puusse. Mängija sisestab pääsukoodi avalehel; pärast edukat REST-päringut suunatakse ta (player)/select-team ekraanile, kus algab WebSocketi-põhine suhtlus.

4.2 Mängija liides

Joonisel 7 on esitatud mängija tüüpilise positiivse stsenaariumi järgnevus — alates pääsukoodi sisestamisest kuni mängu lõpu tulemuste vaateni, mis annab terviklikku ülevaate alapeatükkides 4.2.1–4.2.4 kirjeldatavatest töövoogudest.



Joonis 7. Mängija tüüpilise positiivse stsenaariumi järgnevus.

4.2.1 Mänguga liitumine pääsukoodi alusel

Liitumismehhanismi valikul kaaluti kolme lähenemist: pikka URL-i, QR-koodi ja lühikest numbrilist koodi.

Pikk URL eeldaks osalejatelt seose loomist välistest allikatest (e-post, sõnumirakendus, projektorile kuvatav lühilink), mis turniiri kogunemise tingimustes — kus mängijad peavad alustama samaaegselt — on ebaotstarbekas. QR-kood lihtsustaks liitumist seadmega, millel on kaamera, kuid eeldaks projektorit pääsukoodi kuvamiseks ning ühtmoodi seadme orientatsiooni, mida ei saa kõigil osalejatel garanteerida. Lühike numbriline kood on suuliselt edastatav, projektorile selgesti kuvatav ning kiiresti sisestatav.

Numbrilise koodi pikkuse valikul lähtuti tööstuspraktikast: sarnaseid koode kasutavad nii

Kahoot [7] (6–7 numbrit), Slido [10] (5–6 numbrit) kui Mentimeter [9] (6–8 numbrit). Käesolevas töös valiti neljakohaline kood, mis pakub 10 000 unikaalset kombinatsiooni. Süsteemi mittefunktsionaalsetes nõuetes määratletud samaaegsete aktiivsete turniiride hulk (kuni mõnikümmend, vt peatükk 2.4) jätab unikaalsuse jaoks rohkem kui piisava varu. Koodi unikaalsus tagatakse PostgreSQL-i nõuandvate lukkudega ja UNIQUE-piiranguga andmebaasi tasandil (vt alapeatükk 4.5.3); kollisiooni võimalus on välistatud isegi kahe samaaegse mängu loomise korral.

Lisaks valiti mängijate jaoks registreerimisvaba liitumine. Registreerimine oleks lisanud liitumiskünnise (e-posti kontroll, parooli loomine), mis turniiri lühiajalises kontekstis (osaleja kasutab rakendust paari tunni jooksul) ei oleks põhjendatud. Anonüümsus tagab ka, et mängija isikuandmed ei kogune süsteemi tarbetult. Turvakontroll, mis välistab teise võistkonna nimel sündmuste saatmise, toimub sokli ja osaleja sidumise kaudu.

4.2.2 Võistkonna valik ja koha atomaarne hõivamine

Pärast pääsukoodi sisestamist näeb mängija nimekirja saadaolevatest võistkondadest koos nende kategooriatega. Iga võistkond on tähistatud kas saadavaks (is_available = true ja socket_id = null) või hõivatuks. Mängija valib enda võistkonna ja kategooria ning saadab serverile WebSocket-sündmuse JoinGame. Eduka liitumise korral salvestab server võistkonna kohta sokli sidumee ning seob kliendi WebSocket-toaga (Socket.IO room).

Võistkonna koha hõivamise puhul on kriitiline tagada, et kaks mängijat ei saaks samaaegselt hõivata sama võistkonda. Kaaluti kahte lähenemist: eelkontroll-koos-uuendusega ja tingimuslik atomaarne uuendus. Esimene lähenemine — kontrollida is_available ja seejärel kirjutada socket_id — on rasside seisukohast haavatav, kuna kahe samaaegse päringu vahele võib jõuda teine kirjutamine. Teine lähenemine kasutab ühte SQL-lauset, mis on esitatud Joonisel 8:

```
UPDATE game_participants
SET socket_id = $1, is_available = false
WHERE id = $2 AND socket_id IS NULL;
```

Joonis 8. Atomaarne SQL-uuendus võistkonna koha hõivamiseks.

PostgreSQL tagab sellise lause atomaarsuse rea-tasemel; mõjutatud ridade arv on usaldusväärne indikaator, kas hõivamine õnnestus. Kui *affected rows* = 0, tagastab server kliendile veasõnumi ning mängija saab valida teise võistkonna. Selline lähenemine kõrvaldab vajaduse pessimistliku lukustuse järele ja säilitab teiste osalejate kogemuse mittehäirituna.

4.2.3 Vastuse esitamine ja dünaamiline taimer

Aktiivse küsimuse jooksul kuvatakse mängijale küsimuse number, faasi nimetus (mõtlemine või vastamine) ning järelejäänud sekundite arv. Vastuse esitamiseks on tekstiväli ja saatmisnupp; vastust saab muuta ja uuesti saata kuni ANSWERING faasi lõpuni. Iga esitus saadetakse WebSocket-sündmusena, mis kannab kaasas vastuse teksti, küsimuse identifikaatori ja kliendi-poolse esitamise ajatempli.

Taimeri kuvamisel kaaluti kahte lähenemist: kohalik kliendipoolne loendus ühekordse algusajatempli põhjal või serveripoolne taimer, kus iga sekundi kohta saadetakse kõikidele osalejatele sündmus timer_update. Kohalik taimer on võrgukoormuselt tõhusam, kuid tugineb seadme süsteemikellale, mis võib seadmete vahel mitme sekundi võrra lahkneda — see seab võistkonnad ebavõrdsesse olukorda. Seetõttu valiti serveripoolne taimer ühisalusena. Ühe küsimuse jooksul edastatakse ühte mänguruumi tubasse kuni 70 tikki, mis infrastruktuuri jaoks ei ole märkimisväärne koormus.

Realisatsioonis asendati esialgne setInterval-põhine tikilahendus deadline-põhise hübriidiga: faasi alguses arvutatakse lõpetamise ajatempel phaseDeadlineTs ja kirjutatakse Redisesse, ning iga tikk arvutab järelejäänud aja avaldisena phaseDeadlineTs − Date.now(). Selline ümber-arvestus eraldab kuvamissageduse ajaarvestuse usaldusväärsusest — kui tikk hilineb (näiteks edetabeli arvutamise hetkel), parandab järgmine tikk vahepealse erinevuse ümber, mitte ei lükka kogu ülejäänud aega edasi. Sama referentsajatempel on aluseks ka vastuse saabumisel hilinemissekundite arvutamiseks. Kliendi ühenduse katkemise korral saadab klient taasühendumisel automaatse JoinGame-päringu, millele server vastab oleku hetkeseisuga (faas, järelejäänud sekundid, aktiivse küsimuse identifikaator), ning kuva sünkroniseerub aktiivse seisundiga ilma kasutaja sekkumiseta.

4.2.4 Tulemuste vaade ja tagasiside

Mängu lõpetamisel edastab server kõikidele osalejatele sündmuse game_finished, misjärel suunatakse mängija tulemuste vaatesse. Vaates kuvatakse võistkonna lõpptulemus (skoor, koht edetabelis), võistkonna kõigi vastuste kokkuvõte ning järgmise sammuna anonüümne tagasiside-ankeet. Tagasiside salvestatakse tabelisse player_app_feedback JSON-vormingus, mis võimaldab edasiarenduse käigus küsimuste hulka muuta ilma andmemudelit uuendamata.

4.3 Mängujuhi paneel

4.3.1 Mängu seadistamine: kategooriad, võistkonnad, voorud, küsimused

Mängu seadistuse haldamine toimub mängujuhi paneeli redaktorivaates, mis on jaotatud sakkideks: üldsätted, kategooriad, võistkonnad, voorud ja küsimused. Mängujuht saab seadistust mugavalt muuta enne mängu algust, kuid pärast mängu staatuse üleminekut olekusse LIVE lukustub osa väljadest, et vältida käimasoleva turniiri muutmist.

Mängu seadistuse salvestamise puhul kaaluti kahte lähenemist: inkrementaalseid uuendusi olemi kaupa (iga kategooria, voor, küsimus eraldi PATCH-päringuga) ja terve mängu täielikku salvestamist (üks PUT-päring kogu mängu seadistusega). Esimene lähenemine vähendab võrgukoormust üksiku muudatuse korral, kuid raskendab kliendi-poolset olekuhaldust ning nõuab keerukaid sünkroniseerimisreegleid juhul, kui mitu muudatust on järjestikused. Teine lähenemine, terve mängu täielik salvestamine, on suurema andmesidemahuga, kuid lihtsustab klienti kordades. Kuna mängu seadistuse muutmine ei ole sageduselt kriitiline (mängujuht teeb seda turniiri-eelses ettevalmistuses, mitte aktiivses kasutuses), valiti teine lähenemine.

Samaaegselt avatud mängu redigeerimise vältimiseks rakendatakse optimistlikku samaaegsuse kontrolli veerul games.modified_at. Klient saadab kaasa eelnevalt loetud modified_at-ajatempli; server võrdleb selle andmebaasis salvestatuga ja kirjutab uue versiooni vaid juhul, kui ajatemplid kattuvad. Konflikti korral tagastatakse veasõnum koos andmebaasi-poolse värskemate andmetega ning klient palub kasutajal otsustada, kas need üle kirjutada.

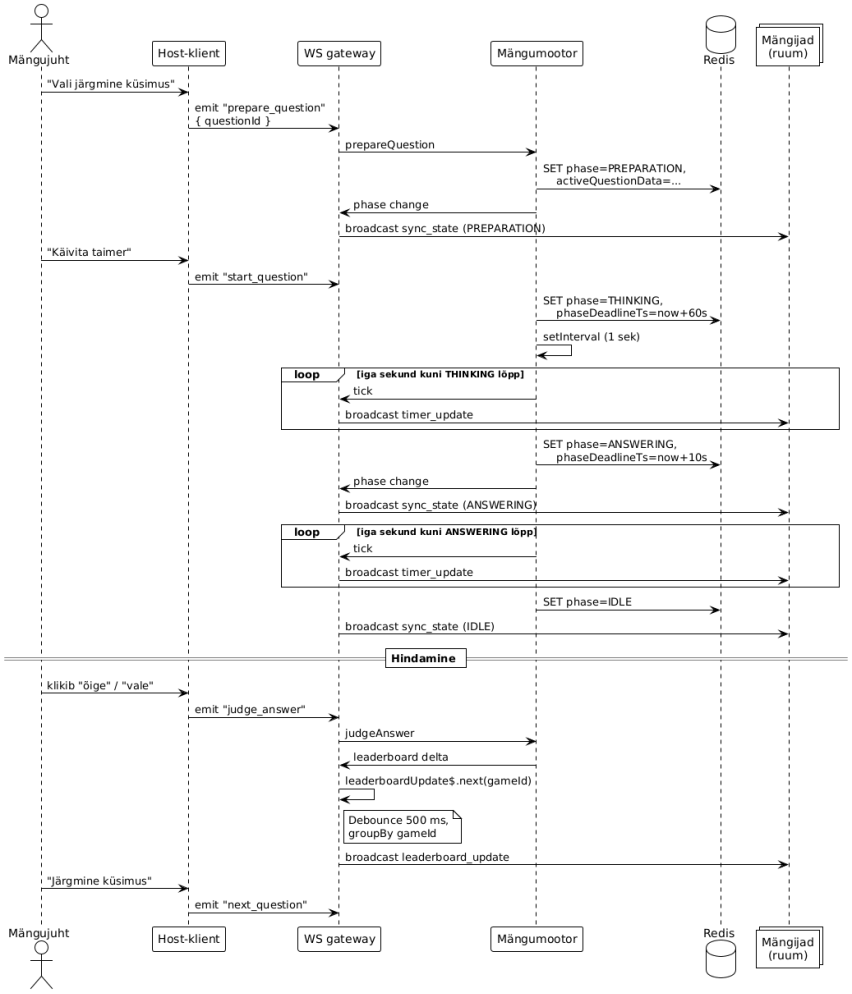
4.3.2 Mängu elutsükli juhtimine

Mängu kulgu juhitakse rea WebSocketi sündmustega, mille kaudu mängujuht annab käsklusi serveri olekumasinale ja mille kaudu server teavitab kõiki ruumi kliente uuest seisundist. Tabelis 2 on kokku võetud peamised sündmused, nende suunad ja olekumõjud. Sündmuste täielik loend ja vastav REST-otspunktide loetelu on toodud Lisas 5.

Tabel 2. Mängu elutsükli peamised WebSocketi sündmused.

Otstarve	**Sündmus**	**Suund**	**Olekumõju**
Mängu käivitamine	start_game	Klient → Server	DRAFT → LIVE
Mõtlemisaja algus	start_question	Klient → Server	PREPARATION → THINKING
Järgmine küsimus	next_question	Klient → Server	* → IDLE
Mängu lõpetamine	finish_game	Klient → Server	LIVE → FINISHED
Edetabeli värskendus	leaderboard_update	Server → Klient	(broadcast)

Olekumasin asub serveri poolel ja toimib ühtse tõeallikana — server valideerib iga ülemineku, lükkab tagasi keelatud üleminekud ning saadab kõigile mänguruumi klientidele uue seisundi. Joonisel 9 on toodud ühe küsimuse tüüpilise tsükli sündmustepõhine järgnevus.



Joonis 9. Küsimuse tüüpilise tsükli sündmustepõhine järgnevus.

4.3.3 Vastuste valideerimine ja apellatsioonide menetlus

Pärast ANSWERING-faasi lõppu kuvatakse mängujuhi vaates kõikide võistkondade vastused antud küsimusele. Iga vastuse jaoks valib mängujuht ühe kahest staatusest: õige (CORRECT) või vale (INCORRECT). Hindamise käsklus saadetakse sündmusena judge_answer koos vastuse identifikaatori ja otsusega.

Iga hindamise muudatus salvestatakse tabelisse answer_status_history, kus säilib vana ja uue staatuse paar, hindamise teostanud kasutaja identifikaator ning ajatempel. See audit-jälg on aluseks mängujuhi otsuste rekonstrueerimisele ja võimalikule apellatsioonide menetlusele edasiarendustes (vt alapeatükk 5.5). Lisaks tagab repository-kiht, et juba lõplikult hinnatud vastust ei kirjutata kogemata uuesti üle (näiteks juhul, kui mängija saadab uuesti vastuse pärast hindamise lõppu).

4.4 Edetabel ja reaalaja sünkroniseerimine

4.4.1 Skoorimisalgoritm

Edetabeli järjestus põhineb kahel näitajal:

- esmane (score) — võistkonna õigesti vastatud küsimuste arv;
- teisene (rating) — kaalutud summa, kus iga õige vastuse kaal sõltub sellest, kui paljud teised aktiivsed võistkonnad sellele küsimusele õigesti vastasid.

Küsimuse kaal arvutatakse järgnevalt:

$$\text{kaal}(q) = \text{aktiivsete\_võistkondade\_arv} - \text{õigete\_vastuste\_arv}(q) + 1 \qquad (4.1)$$

Mida vähem võistkondi küsimusele õigesti vastas, seda suurem on selle kaal. Reiting arvutatakse iga võistkonna jaoks tema õigete vastuste kaalude summana; kui ükski võistkond ei vastanud küsimusele õigesti, võrdub selle kaal aktiivsete võistkondade koguarvuga.

Aktiivseteks võistkondadeks loetakse need, kellel on vähemalt üks vastus salvestatud või kellel on aktiivne WebSocket-ühendus. Järjestamine toimub leksikograafiliselt: esmalt võrreldakse võistkondade score-väärtused (suurem on parem); võrdsete skooride korral võrreldakse rating-väärtused. Sellisel kaheliit-skeemil on peamine eelis — võrdsete skooride korral arvestatakse küsimuste keerukust reitingu kaudu, mis on klassikalise «Mis? Kus? Millal?» formaadi standard [6].

4.4.2 Edetabeli arvutamine

Iga vastuse hindamise järel uuendatakse mängu sisemine edetabel ning edastatakse see kõikidele mänguruumi osalejatele WebSocket-sündmusena leaderboard_update. Lähtekäigul saadeti uuendus iga hindamissündmuse järel, kuid see osutus ebaefektiivseks, kui mängujuht hindab järjestikku mitut vastust kiires tempos — server arvutas ja edastas edetabelit liiga sageli, mis koormas nii andmebaasi kui võrku.

Optimeerimiseks kasutatakse RxJS reaktiivse vooga lähenemist: kõik hindamissündmused suunatakse ühe Subject-i kaudu vooluga, mis grupeeritakse mängu identifikaatori järgi, ning seejärel rakendatakse igale rühmale eraldi summutusmehhanism (ingl *debounce*) aeg 500 ms. See tagab, et samale mängule edetabeli arvutamise päring tehakse korraga

maksimaalselt iga 500 millisekundi tagant, samal ajal kui paralleelselt toimuvate mängude omad ei häiri üksteist.

Konkreetne 500-millisekundine väärtus valiti kahe vastandliku piirangu vahepealse kompromissina. Madalam künnis (näiteks 100 ms) ei summutaks sageli järjestikku saabuvaid hindamisi: testturniiri Tabel 3 põhjal oli minimaalne mõõdetud intervall kahe järjestikuse hindamisotsuse vahel 233 ms ja umbes 7,6% intervallidest jäid alla 500 ms. Kõrgem künnis (näiteks 1500 ms) muudaks edetabeli värskendamise mängijatele märgatavalt aeglaseks, kuna mediaan (p50) intervall hindamisotsuste vahel on 1246 ms, mis tähendab, et 1500 ms aknaga summutaks summutusmehhanism ka selliseid värskendusi, mis tegelikult ei kuulu samasse kiiresse hindamiste seeriasse. Valitud 500 ms pakub keskteed: see eemaldab ressursse-raiskavad duplikaadid kõige kiiremate hindamiste seeriate korral (kus kaks hindamist saabuvad lähemalt kui inimese tüüpiline lihtsa reaktsiooni latentsus $\sim$235–250 ms [33]), kuid ei mõjuta mängijate tajuvat värskendussagedust juhul, kui mängujuht kaalub iga hindamist eraldi.

Tabel 3. Hindamisotsuste vaheliste intervallide jaotus testturniiril.

Statistik	**Väärtus (ms)**
Minimaalne	233
Mediaan (p50)	1 246
Keskmine	1 824
90. protsentiil	3 782
99. protsentiil	4 527
Maksimum	4 934

Edetabeli enda arvutamine toimub serveri poolel, kus kombineeritakse Prisma agregaatfunktsioone kaalu arvutamiseks. Lähenemise alternatiivina kaaluti kliendipoolset arvutamist, mis oleks vähendanud serveri töökoormust, kuid tähendanud, et iga klient peab saama kõikide võistkondade kõik vastused — see oleks võimaldanud osalejatel näha hindamise hetkeseisu enne, kui mängujuht otsustab need näidata, ja oleks vastuolus jaotatud teabe põhimõtetega.

4.5 Töökindlus ja aus mäng

Käesolevas alapeatükis kirjeldatakse meetmeid, mis tagavad süsteemi töökindluse ja võistkondade võrdsed tingimused. Tähelepanu keskmes ei ole klassikaline pettuse ennetamine (näiteks välise abi kasutamine küsimusele vastamisel) — selline kontroll on füüsilise turniiri kontekstis korraldajate vastutuses ja oli tellijaga teadlikult skoobist välja jäetud. Süsteemi turvaline käitumine keskendub tehniliste kõrvalekallete vältimisele.

4.5.1 Pääsuõigused ja seansi sidumine

Mängija puhul rakendatakse kasutajakonto-põhise autentimise asemel sokli ja osaleja sidumist. Eduka liitumise hetkel salvestatakse osaleja ja mängu identifikaatorid konkreetse Socket.IO-ühenduse server-poolsesse seansi-objekti ning sokli identifikaator kirjutatakse atomaarse SQL-lausega tabelisse game_participants. Identifikaatorit hoitakse andmebaasis, mitte vahemälus, et see oleks samas transaktsioonilises tsoonis nagu osaleja kirje ise — see lubab kasutada tingimuslikku mustrit ühe SQL-lausena.

Iga järgneva mängija sündmuse korral kontrollivad gateway abifunktsioonid, et saatva sokli seansi-objektis registreeritud osaleja ja mängu identifikaatorid vastavad sündmuse parameetritele. Kontroll toimub serveriprotsessi mälus aktiivse seansi tasandil, mitte andmebaasi pärimisega — see on piisav, sest sokli füüsiline objekt on sama ja seansi-objekt ei ole klientrakenduse poolt muudetav. Kontrolli läbikukkumise korral lükatakse sündmus tagasi erindi sõnumiga, mis on filtreeritud kasutajale kuvamiseks.

4.5.2 Andmete terviklikkus ja vigade käsitlemine

Andmete terviklikkuse tagamine põhineb kolmel mehhanismil, millest igaüks on rakendatud tingimustes, kus see on kõige tõhusam.

Nõuandvad lukud (ingl *advisory locks*) on kasutusel pääsukoodi eraldamise lühikese kriitilise lõigu ajal. Pääsukoodi genereerimisel võtab server lukku spetsiifilise võtmega, genereerib uue koodi, kontrollib selle unikaalsust andmebaasis ning vabastab lukku tehingu lõppedes. Lukk ei blokeeri muid tehinguid, vaid välistab vaid teiste pääsukoodi-genereerivate päringute samaaegse kulgemise — see on eelistatud lähenemine UNIQUE-piirangu poolt tekitatava veaolukorra ja korduva päringutsükli ees.

Tingimuslik atomaarne uuendus on kasutusel võistkonna koha hõivamisel (vt 4.2.2). Üks SQL-lause kasutab rea-tasemel atomaarsust: kahe samaaegse päringu vahel ei jää «akna», mille jooksul mõlemad saaksid vana väärtust kontrollida ja seejärel mõlemad kirjutada — täpselt üks päring õnnestub, teine tagastab nulli mõjutatud ridade arvu. Sama mehhanismi kaudu salvestab repository-kiht ka vastuse staatuse muudatusi nii, et juba lõplikult hinnatud vastust ei kirjutata kogemata uuesti üle.

Optimistlik samaaegsuse kontroll (ingl *optimistic concurrency control*) on kasutusel mängu seadistuse muutmisel versiooni-välja modified_at põhjal — kui kaks mängujuhti üritavad sama mängu üheaegselt salvestada, tagastab server konfliktiteate koos värskemate andmetega. Vigade käsitlemine on koondatud globaalsesse erindite filtrisse, mille tehnilised üksikasjad on kirjeldatud peatükis 4.5.3.

4.5.3 Vastuste ajalised piirid

Vastuse esitamise sündmuse SubmitAnswer korral teostab server-meetod processAnswer järgmised valideerimised:

- Mängu hetkeline GameStatus peab olema LIVE;
- Mängu hetkeline GamePhase peab olema THINKING või ANSWERING;
- Sõnumi questionId peab kattuma serveri vahemälus salvestatud aktiivse küsimuse identifikaatoriga;
- Sündmuse saabumise ajatempel võrreldakse Redises hoitava phaseDeadlineTs-ga; kui aken on ületatud, kuid faas on veel ANSWERING, salvestatakse vastus koos late_by_seconds-väärtusega; kui faas on juba lõppenud (üleminek IDLE-le toimunud), vastus lükatakse tagasi.

Iga eelneva tingimuse rikkumise korral lükatakse vastus tagasi WsException-iga ning tabelisse answers ei kirjutata midagi. Selline kontroll välistab olukorra, kus klient saadab pärast next_question-sündmust hilinenult vastuse eelmisele küsimusele, või proovib esitada vastust mängu lõppemise järel.

5 Tulemused ja analüüs

Käesolevas peatükis esitatud mõõtmistulemused põhinevad kolmel reaalsel turniiril ja ühel võrdleval paberipõhise korralduse vaatlusel. Tabelis 4 on koondatud iga turniiri parameetrid ja andmete kasutamise eesmärk töö raames.

Tabel 4. Töös kasutatud turniiride ülevaade.

Turniir	**Kuupäev**	**Võistkondi**	**Korralduse formaat**	**Andmete kasutamine**
T1	13. veebruar 2026	13	Paberipõhine	Manuaalse töövoo baasandmed
T2	13. märts 2026	14	Hübriid (kaks vooru paberil, üks vooruvalt rakenduses)	Tagasiside kogunemine ja idee valideerimine
T3	2. aprill 2026	14	Täismahus rakenduses	Tagasiside kogunemine ja täismahus rakenduse esmane reaalvalideerimine
T4	29. aprill 2026	15	Täismahus rakenduses	Stabiilsuse, jõudluse ja kasutatavuse mõõtmine

5.1 Jõudluse ja skaleeritavuse hindamine

Loodud süsteemi jõudluse, stabiilsuse ja skaleeritavuse hindamiseks viidi läbi üks praktiline katse päris mängijatega (T4 turniir), mis andis objektiivse pildi süsteemi käitumisest. Reaalajas läbiviidud katsetuse ajal koguti serveri telemeetriat, et analüüsida päringute viiteaegu, WebSocketi sündmuste sagedust ning andmekihi koormust.

5.1.1 HTTP-päringute viiteaeg

Tabelis 5 on koondatud kuue informatiivsema HTTP-otspunkti viiteaja kvantiilid T4 testturniirilt. Mõõtmised teostati prom-clienti HttpMetricsInterceptor-i kaudu (vt 3.2.5) ühe minuti lõikega ning agregeeriti ja visualiseeriti Grafanas.

Tabel 5. HTTP-otspunktide viiteaja kvantiilid T4 testturniiril.

Otspunkt	**Klass**	**Tippmäär (päringut/min)**	**p95 (ms)**	**p99 (ms)**	**Keskmine (ms)**
POST /host/login	autentimine	1,09	400	450	∼200
POST /host/game/save	suur kirjutusoperatsioon	4,44	500	500	∼200
POST /host/game/get	hosti lugemispäring	2,18	50	50	∼20
POST /player/check-game	kiire järskoormuse lugemispäring	182	25	50	14
POST /player/feedback	mängija kirjutuspäring	8,73	25	25	∼15
GET /player/feedback-form	staatiline konfiguratsioon	10,9	5	6	0,3

Mõõdetud tulemustes eristub kolm gruppi otspunkte. Kõige nõudlikum on mängu seadistuse salvestamine (POST /host/game/save): andmemahukas kirjutusoperatsioon, mille üks päring kannab kogu mängu struktuuri (voorud, küsimused, võistkonnad, kategooriad). Mõõdetud p99 = 500 ms saavutati esmakordsel salvestusel, kus Prisma ORM täidab mitut INSERT-tehingut järjestikku; korduvate salvestuste puhul (p99 $\approx$ 250 ms) domineerivad UPDATE-päringud koos optimistliku samaaegsuse kontrolliga (vt 4.3.1). Mängujuhi autentimine (POST /host/login, p99 $\approx$ 450 ms) on samas suurusjärgus, kuna kasutusel on tahtlikult arvutuslikult kallis bcrypt-algoritm — see on ühekordne operatsioon turniiri alguses ega mõjuta mängu jooksvat dünaamikat.

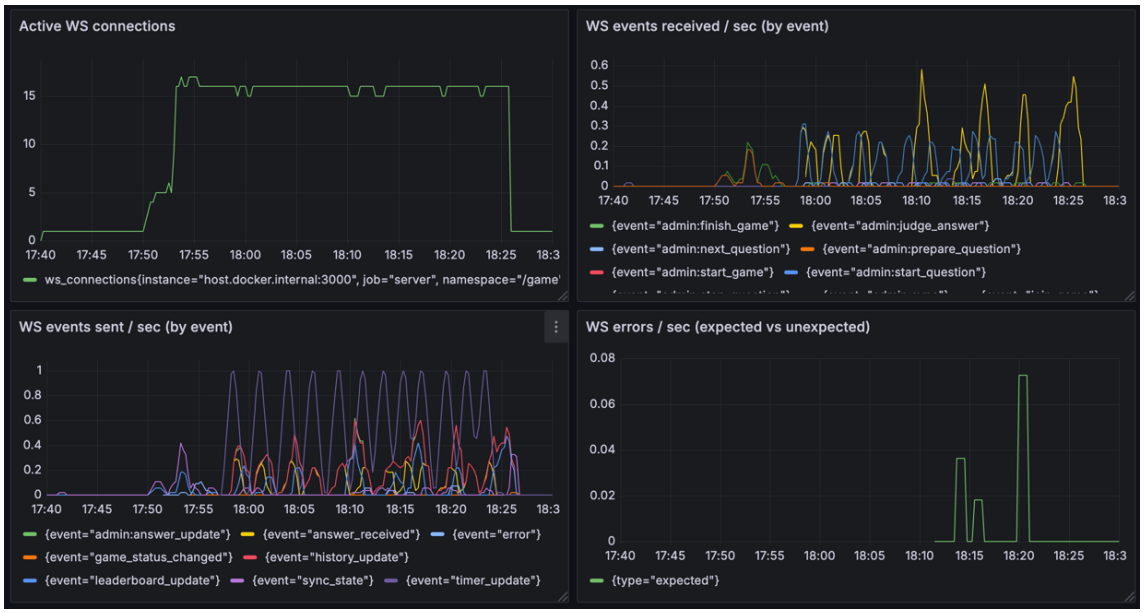
Reaalajas kriitilisem on pääsukoodi kontroll (POST /player/check-game), mis talub järsku liitumiskoormust: kõikide võistkondade samaaegne liitumine umbes 17:51 paiku tekitas tipptasemel 182 päringut minutis, kuid p99-viiteaeg jäi 50 ms ja keskmine 14 ms — see on kahekümnekordse varuga alla mittefunktsionaalsetes nõuetes (vt 2.4) seatud ühe sekundi piiri. Ülejäänud äriloogika otspunktide (POST /host/game/get, POST /player/feedback) viiteaeg jäi samuti p99 alla 50 ms. Staatiline GET /player/feedback-form (p99 = 6 ms) toimib süsteemi baasjoonena, mis kirjeldab puhast HTTP- ja NestJS-konteineri viiteaega ilma andmekihi panuseta.

Edetabeli HTTP-otspunkti (GET /player/game/:gameId/leaderboard) kasutati turniiri ajal

praktiliselt mitte: edetabeli reaalajas levitamine toimub WebSocketi sündmusena (vt 4.4.2), HTTP-otspunkt jääb varukanalina (ingl *fallback*) mängu järgseks tulemuste vaateks. Selline duaalne lähenemine vähendab serveri koormust ilma funktsionaalsust kaotamata.

5.1.2 WebSocketi sündmuste rütm

WebSocket on süsteemi peamine reaalajas andmevahetuskanal — mängu jooksul liigub selle kaudu kogu olekuteave (taimer, faasi-üleminekud, hindamisotsused, edetabeli värskendused). Joonisel 10 on esitatud nelja võtmemõõdiku käitumine testturniiri vältel: aktiivsete ühenduste arv, vastuvõetud sündmuste määr sündmuste lõikes, saadetud sündmuste määr sündmuste lõikes ning vigade määr eraldatuna ootuspärasteks ja ootamatuteks.



Joonis 10. WebSocketi koondnäitajad T4 testturniiri vältel.

Aktiivsete ühenduste arv. Mõõdetud tipparv 16 samaaegset ühendust saavutati turniiri esimese viie minuti jooksul ning püsis stabiilsena kuni viimase küsimuse lõpuni (umbes 30 minutit). Ühenduste arv vastab osalejate koguarvule (mängujuht ja võistkondade seadmed). Stabiilne plato ilma jõnksudeta tähendab, et ükski ühendus ei katkenud serveriga töö jooksul ootamatult — Socket.IO automaatne taasühendamise mehhanism (vt 3.1.5) ei pidanud sekkuma.

Vastuvõetud sündmused. Kõige sagedasem mängujuhi käsklus on admin:judge_answer — vastuse hindamise otsus —, mille tipptasemel saavutati 0,5–0,6 sündmust sekundis

(st kuni 6 hindamist 10 sekundi jooksul ühe küsimuse arutelu lõpus). Mängu seisundi muutmise käsklused ilmuvad joonisel üksikute kõrgete piikidena, mis vastavad mängu loogiliste etappide üleminekutele. See sündmuste struktuur kinnitab arhitektuurset eeldust, et hindamine on protsessuaalselt kõige intensiivsem etapp ning peamine optimeerimise sihtmärk.

Saadetud sündmused. Tipptasemel saadab server umbes 1,0 sündmust sekundis. Pidevat fooni tekitab timer_update-sündmus, mis edastab järelejäänud sekundeid kõikidele võistkondadele iga sekundi tagant. Hindamise faasis lisanduvad answer_update (mängujuhi vaatesse), history_update (võistkonna vaatesse oma vastuste seisuga) ja leaderboard_update (kogu ruumile). Märkimisväärne tähelepanek: kui admin:judge_answer saabub serverisse tipphetkel 0,6 sündmust sekundis, siis sellele vastav leaderboard_update saadetakse oluliselt madalama sagedusega — see kinnitab, et alapeatükis 4.4.2 kirjeldatud 500 ms summutusmehhanism summutab kiired hindamiste seeriad ootuspäraselt ega tekita iga hindamise järel ümberarvutuse ja levitamise koormust.

Vigade määr ja töökindlus. Vigade paneel eristab kahte tüüpi sündmusi: ootuspärased ja ootamatud. Ootuspärased vead on WsException-tüüpi erindid, mille filter WsExceptionFilter (vt 4.5.3) tõlgendab kontrollitud äriloogika reegli rikkumiseks — näiteks vastuse esitamise katse väljaspool lubatud faasi. Ootamatud vead esindavad serveripoolseid sisemisi tõrkeid (programmiloogika erindid, andmebaasi-tõrked).

Mõõdetud tulemused näitavad, et turniiri jooksul registreeriti kokku $\sim$10–15 ootuspärast veasündmust (peamiselt 18:14–18:20 vahemikus, kus tipptasemel ulatus määr 0,075 sündmust sekundis) ning mitte ühtegi ootamatut viga. Ootuspäraste vigade suurim osa langes hindamise kõrghetkele, mis viitab sellele, et tegemist oli kas mängijate katsetega esitada vastust juba lõppenud faasis või mängujuhi kahekordsete klikkidega. Ootamatute vigade puudumine kogu turniiri vältel kinnitab, et nii serveripoolne valideerimisloogika kui andmekiht töötasid tõrgeteta tegeliku kasutusjuhu koormuse all. Tabel 6 võtab kokku WebSocketi mõõdetud koondnäitajad.

Tabel 6. WebSocketi mõõdetud koondnäitajad T4 testturniiril.

Näitaja	**Väärtus**
Aktiivseid ühendusi (tipphetkel)	16

Näitaja	Väärtus
Aktiivseid ühendusi (keskmine plato ajal)	16
Aktiivse plato kestus	∼30 minutit
Vastuvõetud sündmuste tipphetk (admin:judge_answer)	∼0,6 sündmust/s
Saadetud sündmuste tipphetk	∼1,0 sündmust/s
Tüüpilisem saadetav sündmus	timer_update (pidev foon)
Ootuspäraseid vigu (kokku)	∼10–15
Ootamatuid vigu (kokku)	0
Serveripoolseid ühenduse katkemisi	0

WebSocketi-tasandi mõõtmistulemused kinnitavad, et arhitektuurilised valikud — Socket.IO transpordina, summutusmehhanism edetabeli edastuse puhul ja kontrollitud erindite filter — toimivad reaalse kasutuse koormuse all ootuspäraselt. Ootamatute vigade täielik puudumine ja stabiilne ühenduste plato annavad alust väita, et süsteem on käesoleva mahuga turniiride läbiviimiseks usaldusväärne.

5.1.3 Andmekihtide koormus

Andmekihtide ja serveri operatsioonisüsteemi koormus mõõdeti samaaegselt rakendustasandi telemeetria kõrval samal turniiril (T4). Redis teenindas tipphetkel umbes 6,5 käsku sekundis kahe ühenduvuse klientidega (server + redis-exporter), mälukasutus jäi 1,3 MiB juurde, evictions oli null ning aegunud võtmete sagedus vastas TTL-mehhanismile küsimuse-tähtaegade võtmete puhastamiseks. PostgreSQL kasutas tipphetkel 7 ühendust (vaikimisi 100-st), tegi 6–10 kinnistust sekundis ilma tagasivõtmisteta, registreeris null tupikseisu kogu turniiri vältel ning teenindas päringuid umbes 100% puhvritabamise määraga — see seletab ka HTTP-otspunktide madalad p99-viiteajad. Hosti protsessor töötas 6–7% tasemel, Node.js heap kasutas 28–44 MiB ning sündmustsükli viiteaeg jäi enamiku ajast alla 5 ms.

Mõõdetud koormusprofiil — 7% CPU kasutust 16 samaaegse ühenduse korral — võimaldab püstitada hüpoteesi süsteemi suurema mahu kohta, kuid mitte seda kvantitatiivselt tõestada. Reaalajas süsteemide jõudlus ei skaleeru tüüpiliselt lineaarselt üle suurusjärgu: kasvavate ühenduste arvu korral lisanduvad mittelineaarsed komponendid (operatsioonisüsteemi pistikupesade haldus, sündmustsükli viiteaeg sündmuste hulga kasvades, andmebaasi-

ühenduste basseini küllastumine, võrgupõhise multipleksimise üldkulu). Mõõdetud 7% CPU kasutust 16 ühenduse korral näitab, et käesoleva mahu juures on süsteem CPU-poolelt madala koormusega ning protsessor ei ole pudelikael; siiski ei luba see üksik mõõtmispunkt teha kindlat järeldust süsteemi käitumise kohta 100 või 500 ühenduse juures. Selle väite tõestamiseks on vajalik sünteetiline koormuskatse Apache JMeteri või sarnase tööriistaga, mille käigus mõõdetakse jõudlust mitmel tasemel (näiteks 25, 50, 100, 200 ühendust) ning tuvastatakse esimese kvalitatiivse halvenemise (p99-viiteaeg ületab 1 sekundi) tekkimise kohas. Selline mõõtmine jäi käesoleva töö skoobist välja ning on planeeritud järgmise iteratsiooni jaoks (vt 5.5).

5.2 Kasutatavuse analüüs

Kasutatavuse hindamiseks kombineeriti kolme meetodit. Niinimetatud trianguleeritud lähenemise loogika tugineb Nielseni klassikalisele kasutatavusanalüüsi (ingl *usability-engineering*) metoodikale [34]: ühelegi üksikule mõõtmismeetodile ei tohi anda eksklusiivset autoriteeti, kuna iga meetod katab teiste nõrkused — objektiivne käitumistelemeetria näitab «kuidas», kuid mitte «miks»; lõputagasiside ankeet annab subjektiivse hinnangu, mis pole alati kooskõlas tegeliku käitumisega; otsene vaatlus annab kontekstuaalse seletuse, kuid ei skaleeru. Käesolevas töös kasutati järgmiseid komponente:

Klientrakenduse telemeetria. Mixpaneli kaudu (vt 3.2.6) koguti turniiri T4 ajal klientrakenduse sündmuste telemeetria. Need andmed annavad objektiivse pildi sellest, milliseid funktsioone kasutati, kui kaua igal ekraanil veedeti ja kus tekkisid tüüpilised vigade mustrid.

Lõputagasiside ankeet. Pärast mängu lõppu palus rakendus mängijatel täita anonüümse 5-pallilise tagasiside ankeedi (vt 4.2.4). Vastuse andis 12 osalejat 15-st (80%).

Tagasiside-vorm koostati neljast kasutuskogemuse mõõtmest lähtudes — liitumiskogemus, sünkroniseerimine, vastuste esitamine ja meelelaadne kvaliteet — ning iga aspekt on esindatud sümmeetriliselt nii positiivse kui negatiivse sektsiooni kaheksa seas. Negatiivsesse sektsiooni on sihilikult lisatud prototüübi skoobist välja jäetud funktsioonid (apellatsioonid, õigete vastuste järelvaade, süsteemsed teavitused), et koguda objektiivset andmestikku edasiarenduse prioriteetide jaoks.

5.2.1 Kasutajate käitumise tähelepanekud

Mixpaneli kaudu (vt 3.2.6) registreeriti turniiri T4 vältel kõikide süsteemis defineeritud sündmuste — kokku 31 erinevat tüüpi — täielik telemeetria-jälg. Telemeetria peamine roll käesolevas alapeatükis on alapeatükkides 3.1–4.5 põhjendatud arhitektuuriliste valikute empiiriline kontrollimine: iga järgnev tähelepanek seob konkreetse mõõdetud suuruse ühe disainiotsusega. Tabel 7 esitab koondtulemused, mida järgnev tekst tõlgendab.

Tabel 7. Telemeetria koondtulemused turniiril T4 ($n = 15$ võistkonda).

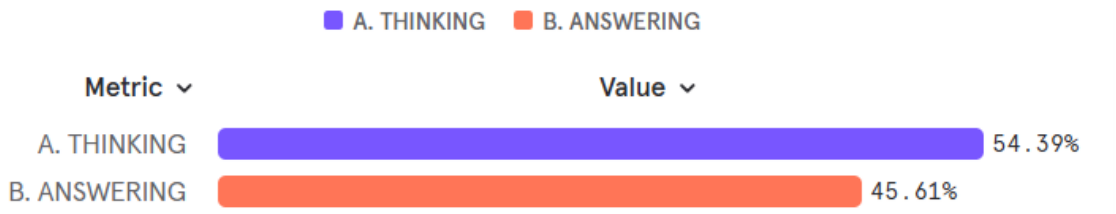
Näitaja	**Väärtus**
Esmaesitused / korrigeeritud (is_resubmit = true)	135 / 36
THINKING- / ANSWERING-faasis fikseeritud	93 (54%) / 78 (46%)
Mediaan- ja keskmine aeg küsimusest vastuseni	47 s / 57 s
Mediaan-aeg App Opened → Team Selected	1,9 min
Mängujuhi ümberhindamised	16 / 171 (9%)
Taimeri peatamisi / taastamisi / korrigeerimisi	5 / 4 / 2
Lühiajalised soklikatkestused ($\leq$ 2 s, automaatne)	16
Käsitsi taasliitumisi (kasutaja sekkumisega)	0

Liitumise voog. Mediaan-aeg rakenduse avamisest võistkonna valimiseni oli 1,9 minutit, hõlmates pääsukoodi sisestamise, nimekirja sirvimise ja koha hõivamise. See tulemus näitab, et alapeatükis 4.2.1 kirjeldatud minimaalse sisendiga liitumine (numbriline pääsukood ilma kasutajakontota) ja alapeatükis 4.2.2 kirjeldatud ühe puudutusega koha hõivamine võimaldavad esmakordsel kasutajal jõuda mängu alguspunkti kiiresti ja intuitiivselt, minimaalsete juhistega või ilma nendeta.

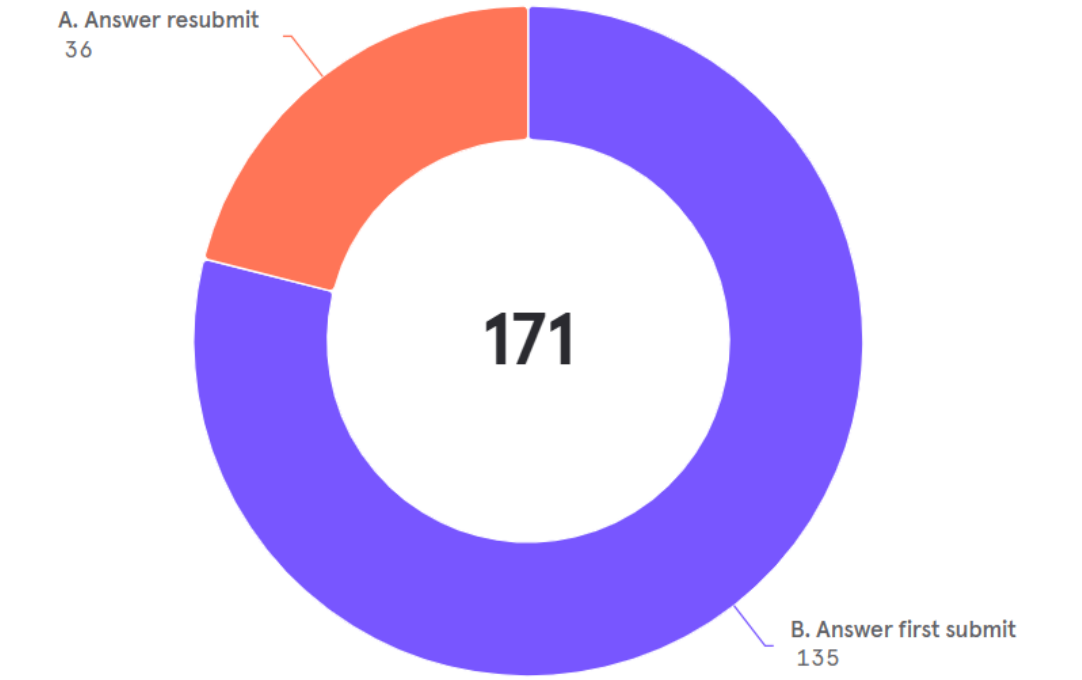
Vastuste fikseerimise muster. Vastustest 54% saabusid THINKING-faasi jooksul ja 46% viimase 10-sekundilise ANSWERING-akna sees (vt Joonis 11). Pea pool vastustest fikseeritakse seega kahefaasilise taimeri teises aknas — see kinnitab alapeatükis 4.2.3 kirjeldatud disainivaliku õigustatust. Üheastmelise taimeri korral, kus vastuste vastuvõtmine katkestatakse THINKING-faasi lõpus, kaotaks süsteem 46% praegustest fikseeritud vastustest või vajaksid neid eraldi käsitleda apellatsioonimenetluses.

Kogu 171 vastusest 36 (21%) läbisid vähemalt ühe iteratsiooni enne lõplikku esitamist (vt Joonis 12). Iga viies fikseeritud vastus läbis seega korrigeerimise — see näitab, et vastuse

muutmise võimalus pole mugavusfunktsioon, vaid tegelik osa võistkonna kollektiivse otsuse kujunemisest. Olemasolevatel platvormidel (vt 2.3), kus vastus on lõplik kohe esitamisel, oleksid need 21% kas vananenud versioonis lukus või vajaksid eraldi apellatsioonimenetlust.



Joonis 11. Vastuste jaotus mängufaaside lõikes (T4, $n = 171$).

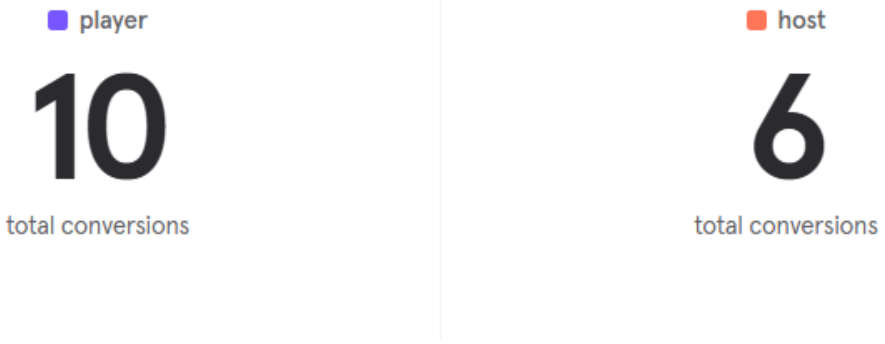


Joonis 12. Vastuste esitamise iteratsioon (T4, $n = 171$).

Mängujuhi tegevus. Mängujuht hindas kõik 171 laekunud vastust; 9% neist (16) said uue hinde, mis ilmneb samale answer_id-le vastava korduva Host Answer Judged sündmuse põhjal. Need ümberhindamised koondusid turniiri esimesse poolde, mil hindamistempo polnud veel sisse elatud, ning on kvantitatiivne kinnitus mängujuhi õppimiskõverale (vt 5.4.1). Sarnane muster ilmnes taimeri haldamisel: 5 peatamist ja 4 taastamist ning 2 ajalist korrigeerimist koondusid valdavalt turniiri esimesse poolde, mil mängujuht

alles harjus süsteemiga. Turniiri teises pooles kulges taimeri haldamine juba vigadeta, mis kinnitab, et algkonfiguratsioon (60 + 10 sekundit) sobib formaadi nõuetele ning õppimiskõver on lühike.

Soklite stabiilsus. Mixpaneli funnel-analüüs (Socket Disconnected → Socket Connected) tuvastas 16 lühiajalist soklikatkestust — 10 mängija poolt ja 6 mängujuhi poolt (vt Joonis 13). Kõik taastusid Socket.IO automaatse taasühendamise mehhanismi kaudu, mille käigus serveripoolne sync_state-mehhanism (vt 4.5.3) taastas kliendi täismahus oleku — mänguringi-faasi, taimeri jääk-aja, esitatud vastuste seisu ja edetabeli — ilma kasutaja sekkumiseta. Olulisim järeldus on aga see, mida need 16 sündmust ei põhjustanud: mitte ükski osaleja ei märganud ühtegi katkestust ega pidanud rakendust käsitsi taaskäivitama; mängijate ankeedis ja tellija tagasisides ei mainitud ühtegi ühenduvuse-kaebust. See on empiiriline kinnitus alapeatükis 3.2.3 põhjendatud serveripoolse oleku-juhtimise põhimõttele: kuna kliendil ei ole iseseisvat olekut, on iga võrgu hetketõrge süsteemile triviaalselt taastatav ja kasutajale läbipaistev.



Joonis 13. Lühiajalised soklikatkestused rollide lõikes (T4, $n = 16$).

5.2.2 Kasutajate tagasiside

Tagasiside-skoor ja valikvastused. Mängijate keskmine 5-palline hinnang oli 4,4 ja mediaan 5; jaotuse kohaselt andis 7 vastajat 12-st kõrgeima hinnangu, 3 vastajat hinnangu 4 ning 2 vastajat hinnangu 3. Kuus vastajat 12-st ei valinud ühtegi negatiivse sektsiooni valikut. Tabelis 8 on esitatud valikute jaotus mõlema sektsiooni kohta.

Tabel 8. Valikvastusega küsitluse tulemuste jaotus.

Sektsioon	**Valik**	**Vastuseid**	**%**
Mis meeldis?	Kiire reageerimine	10	83
	Lihtne liitumine	9	75
	Hästi nähtav taimer	8	67
	Vastuse muutmine enne aja lõppu	7	58
	Intuitiivne kasutajaliides	6	50
	Reaalajas edetabel	5	42
	Vastuste ajalugu	2	17
Mida saaks parandada?	Õigete vastuste järelvaade	3	25
	Teavitused	2	17
	Heli/vibratsioon	2	17
	Apellatsioonide võimalus	1	8
	Võistkonna leidmine	1	8
	Faasi selgus	1	8
	Tume teema	0	0
	Suurem teksti suurus	0	0
	Ühenduse probleemid	0	0

Positiivse sektsiooni tipp-neli kattub täpselt arhitektuurivalikutega, mille saavutamiseks tehti eraldi otsuseid: serveripoolne taimer, deadline-põhine vastamise aken ja summutusmehhanism. Negatiivse sektsiooni tipp-kolm — õigete vastuste järelvaade ja süsteemsed teavitused, heli- ja vibromärguanded — kattub prototüübi skoobist välja jäänud funktsioonidega. Asjaolu, et kasutajad ei tundnud puudust ühestki juba realiseeritud funktsioonist, kinnitab arendusplaani prioriteete. Vabakirjelduse väljal lisas neli vastajat täpsustusi, mis olid kõik kooskõlas valitud vastustega.

5.2.3 Tellija tagasiside ja tehtud parandused

Pärast turniire T2 (hübriid) ja T3 (täismahus rakenduses, vt Tabel 4) koguti mängukorraldajatelt tagasisidet, mille põhjal optimeeriti süsteemi reaalajas haldamise võimekust ja andmete terviklikkust. Tellija soovid jaotati koheselt realiseeritud täiendusteks ning tulevikuarendusteks.

Realiseeritud täiendused ja parandused:

- **Detailne statistika ja seire:** Mängujuhi paneelile lisati reaalajas loendurid liitunud võistkondade (sh *online*/*offline* staatus), kategooriate kaupa rühmitamise ning laekunud vastuste jaotuse (õiged vs valed) kohta. See võimaldab korraldajatel operatiivselt jälgida mängu kulgu ja tuvastada tehnilisi tõrkeid (nt mängija ühenduse katkemine).
- **Kasutajaliidese (UX) parandused:** Mobiilseadmetel eemaldati nuppudelt tekstivaliku (*text selection*) võimalus, mis varem takistas kiireid toiminguid. Lisaks loodi mängujuhi paneelist lihtsustatud vaade nutitelefonidele, mis on mõeldud spetsiaalselt küsimuste lugejale taimeri ja faaside juhtimiseks, võimaldades faaside juhtimise ja vastuste hindamise füüsilist eraldamist kahe korraldaja vahel.
- **Andmete terviklikkus ja kontroll:** Välistati vastuste hindamine enne vastamisaja (*ANSWERING* faas) lõppu, et ennetada olukordi, kus tulemusi hinnatakse ajal, mil mängijad saavad veel vastust muuta.
- **Ühilduvus pärandsüsteemidega:** Lisati lõpptulemuste eksport .xlsx failina. Kuna antud bakalaureusetöö raames loodud rakendust kasutati ainult hooaja viimastel etappidel, ei olnud tsentraliseeritud andmehoidla loomine kogu turniiriaasta koondtabeli jaoks otstarbekas. Ekspordifunktsioon tagas kriitilise tagasiühilduvuse, võimaldades korraldajatel integreerida uue platvormi tulemused vaevata oma senisesse, aasta algusest peetud tabelipõhisesse (Exceli) punktiarvestussüsteemi.

Tulevikuplaanidesse siirdatud soovid:

- **Heli- ja vibratsioonisignaalid:** Taimeriga seotud märguanded jäid prototüübi skoobist välja veebibrauserite tehniliste piirangute tõttu (nõuavad natiivset rakendust).
- **Manuaalsed möödapääsud:** Punktide käsitsi lisamine või mängija eest vastuste sisestamine lükati edasi, et säilitada andmemudeli atomaarsus ja vältida auditi-jälje konflikte prototüübifaasis.

5.3 Digiülemineku mõju

Loodud lahenduse tegeliku väärtuse hindamiseks võrreldi traditsioonilise paberipõhise turniirikorralduse ja uue digitaalse korralduse ressursikulu. Võrdluse aluseks on kaks võrreldava mahu, kuid mitte identsete tingimustega turniiri: paberipõhine baasturniir T1 (13 võistkonda) ja digitaalne valideerimisturniir T3 (14 võistkonda); vt Tabel 4 peatüki 5 alguses. Selleks logiti esmalt paberipõhise mängu ajakulu vaatluse teel ning seejärel

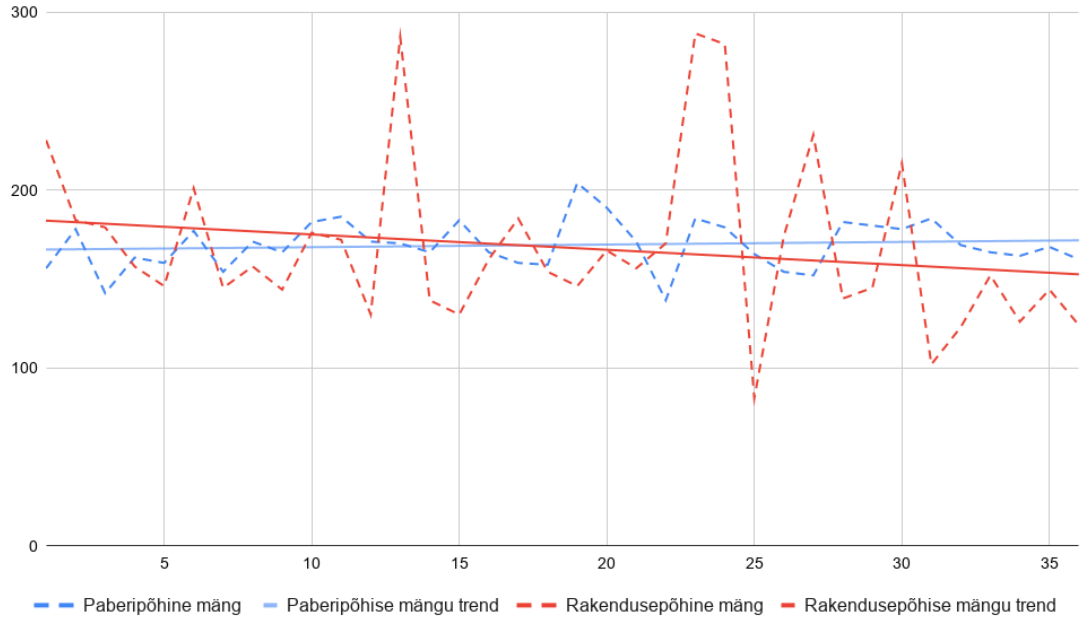
kõrvutati tulemusi rakenduse abil läbi viidud sarnase mahuga turniiriga. Ühe võistkonna erinevus ei mõjuta protsessuaalseid näitajaid (vastuste kogumise kestus, voorudevahelised pausid, žürii kogu ajakulu), küll aga tuleb absoluutsete väärtuste tõlgendamisel see asjaolu meeles pidada (vt 5.4 alapeatükis esitatud piirangud).

5.3.1 Mõõdetud parameetrid

Manuaalse töövoo baasandmete kogumiseks mõõdeti reaalajas 13 osaleva võistkonnaga turniiri, mis koosnes kolmest voorust ja 36 küsimusest. Iga küsimuse juures fikseeriti küsimuse lugemisele, arutelule ja vastamisele, vastuste kogumisele ning hindamisele kulunud aeg. Samuti mõõdeti voorude vahelised ja mängu lõpus tekkinud tehnilised pausid, mis kulusid žüriil tulemuste kokkuarvutamiseks. Digitaalse läbiviimise puhul mõõdeti samaväärseid parameetreid platvormi telemeetria ja vaatluse teel.

5.3.2 Võrdluse tulemused

Digiülemineku tulemuste analüüsimisel ilmnes ootuspärane ajaline võit, kuid platvormi kõige märgilisem mõju seisnes hoopis inimressursi vajaduse radikaalses vähenemises. Joonisel 14 on esitatud võrdlev graafik küsimuste läbiviimise ajakulu kohta koos lineaarsete trendjoontega. Trendjooned paljastavad kvalitatiivse erinevuse, mida keskmised ei kajasta: paberipõhise turniiri ajakulu trend on kerge tõusuga, digitaalse turniiri oma aga langev. Paberipõhise formaadi aeglustumise põhjuseks on žürii kumulatiivne koormus — iga vooru järel suureneb käsitsi sisestatavate paberlipikute hulk ja kasvab kognitiivne väsimus, mistõttu vastuste kogumine ja hindamine sünkroonis mängutempoga muutub turniiri lõpu poole keerukamaks (vt 2.2). Digitaalse formaadi puhul ilmneb vastupidine muster — mängujuht harjub liidesega turniiri käigus, mida kinnitab alapeatükis 5.2.1 mõõdetud ümberhindamiste ja taimeri-korrigeerimiste koondumine turniiri esimesse poolde.



Joonis 14. Küsimuste läbiviimise ajakulu dünaamika paberipõhises ja rakendusepõhises formaadis koos lineaarsete trendjoontega.

Ajalise kokkuhoiu vaatenurgast oli digitaalse platvormi mõju esimeses testis tagasihoidlik: küsimuste läbiviimisele kulus paberipõhisel mängul kokku 101,47 minutit ning digitaalsel 100,62 minutit — vahe 51 sekundit on mõõtmisvea piirides ja statistiliselt ebaoluline. See tulemus on ootuspärane: tegemist oli mängujuhi esimese kogemusega uues süsteemis, mistõttu liidese kasutamise harjumatus neutraliseeris vastuste kogumise automatiseerimisest tuleneva potentsiaalse ajasäästu. Seevastu voorudevahelised tehnilised pausid — kaks 2-minutilist pausi voorude vahel ja 10-minutiline paus lõpus tabelite käsitsi kokkulöömiseks — kadusid täielikult, andes 14-minutilise netovõidu. Turniiri kogukestus lühenes seega 130 minutilt ligikaudu 116 minutile. Kuid absoluutne ajaline kokkuhoid on vaid murdosa tegelikust efektiivsuse kasvust — peamine väärtus joonistub välja kumulatiivses inimtöökoormuses.

Digiülemineku peamine väärtus joonistub välja kumulatiivse inimtööaja võrdluses, kui arvestada kogu turniiri teenindamisele kulunud aega. Manuaalselt läbi viidud turniiri kogukestus, mis hõlmas lisaks 101-minutilisele mänguajale ka registreerimist ja tulemuste väljakuulutamist, oli 130,47 minutit. Selle haldamiseks oli vaja 5-liikmelist meeskonda: kaks vastuseid koguvat abilist (ehk «pääsukest») ja kolm žüriiliiget. Paberipõhise turniiri

kogu inimtöökoormuseks kujunes seega 652,33 minutit (5 inimest $\times$ 130,47 minutit).

Seevastu rakenduse abil läbi viidud turniiril kadus vajadus nii füüsiliste vastuste kogujate kui ka mitmeliikmelise žürii järele. Kogu mängu loogika, faaside vahetamise ja vastuste hindamise suutis operatiivselt hallata vaid üks inimene. Kuna tehnilised pausid kadusid, lühenes digitaalse turniiri kogukestus (koos ettevalmistuse ja kokkuvõtetega) hinnanguliselt 116 minutini. Ühe mängujuhi panus kogu sündmuse vältel tähendab seega kogu süsteemi inimtöökoormuseks vaid ligikaudu 116 minutit.

Tulemus avaldub kolmes mõõtmes:

- Korraldajate arvu vähenemine: 5-liikmelise meeskonna (kaks vastuste kogujat ja kolm žüriiliiget) asemel piisab ühest mängujuhist (vähenemine -80% koosseisuliselt).
- Turniiri kogukestus: 130 minutilt 116 minutile (-11%, ligikaudu 14-minutiline netoekonoomia tehniliste pauside elimineerimisest; küsimuste läbiviimise aeg jäi esimeses testis praktiliselt samaks).
- Kumulatiivne inimtöökoormus: 5 inimest $\times$ 130,47 minutit $\approx$ 652 minutit langes 1 inimene $\times$ 116 minutit = 116 minutile (-82%).

See vabastab korraldajad suurest logistilisest koormast vabatahtlike leidmisel ning tõestab, et loodud digitaalne lahendus muudab turniiride korraldamise märkimisväärselt jätkusuutlikumaks ja skaleeritavamaks (vt Tabel 9).

Tabel 9. Paberipõhise ja digitaalse turniirikorralduse võrdlus.

Näitaja	**Paberipõhine (T1)**	**Digitaalne (T3)**	**Muutus**
Küsimuste läbiviimise aeg	101,47 min	100,62 min	-51 s (-1%)
Voorudevahelised pausid	14 min	0 min	-14 min
Turniiri kogukestus	$\sim$130 min	$\sim$116 min	-11%
Korraldajate arv	5	1	-80%
Kumulatiivne inimtöökoormus	652 min	116 min	-82%

5.3.3 Laiem sotsiaalne ja praktiline väärtus

Eelpool mõõdetud inimtöökoormuse vähenemine (-82%, vt Tabel 9) ei piirdu üksnes ühe konkreetse turniiri kontekstiga — uurimuse sotsiaalne ja praktiline väärtus ulatub

kohalikust turniirikorraldusest kaugemale. Kuigi käesoleva töö raames loodud lahendus on vahetult valideeritud koostöös tellijaga MIS? KUS? MILLAL? MTÜ-ga, panustab digitaliseeritud töövoog laiemalt mitteformaalse hariduse ja intellektuaalse vaba aja veetmise viiside demokratiseerimisse. Madaldades organisatoorset ja logistilist barjääri, võimaldab loodud platvorm intellektuaalsete meeskonnamängude läbiviimist piiratud ressurssidega keskkondades — näiteks maapiirkondade koolides või huvikeskustes, kus mitmeliikmelise korraldusmeeskonna ja žürii kaasamine on füüsiliselt või majanduslikult raskendatud. Seeläbi toetab töö noorte analüütilise mõtlemise ja meeskonnatöö arendamist laiemas ühiskondlikus mastaabis, muutes arendava tegevuse kättesaadavamaks. Selle potentsiaali kvantitatiivne hindamine teistsugustes organisatsioonilistes kontekstides eeldab eraldi valideerimist (vt 5.4.1).

5.4 Eesmärkide saavutamise hindamine

Käesolevas alapeatükis hinnatakse, mil määral prototüüp täidab 2.4 alapeatükis sõnastatud funktsionaalseid ja mittefunktsionaalseid nõudeid. Hinnang esitatakse kahes tabelis (Tabel 10 ja Tabel 11) koos viidetega tõendusmaterjalile töö asjakohastes peatükkides.

Tabel 10. Funktsionaalsete nõuete täidetus.

Moodul	**Nõue**	**Olek**	**Tõendus**
Võistkondade moodul	PIN-koodi alusel liitumine	Täidetud	4.2.1, 5.1.1
Võistkondade moodul	Võistkonna valik koos koha hõivamisega	Täidetud	4.2.2
Võistkondade moodul	Vastuse sisestamine sünkroniseeritud taimeriga	Täidetud	4.2.3, 5.1.2
Võistkondade moodul	Oma vastuste ajalugu koos verdiktidega	Täidetud	4.2.4
Võistkondade moodul	Reaalajas edetabel	Täidetud	4.4.1–4.4.2, 5.1.2
Mängujuhi paneel	Mängu seadistamine (kategooriad, voorud, küsimused, võistkonnad)	Täidetud	4.3.1, 5.1.1
Mängujuhi paneel	Mängu elutsükli juhtimine (käivitamine, paus, järgmine küsimus, lõpetamine)	Täidetud	4.3.2
Mängujuhi paneel	Vastuste reaalajas hindamine	Täidetud	4.3.3, 5.1.2
Mängujuhi paneel	Apellatsioonide menetlus	Osaliselt täidetud	3.3.2, 4.3.3, 5.5

Moodul	Nõue	Olek	Tõendus
Mängujuhi paneel	Täiendavad rollid	Osaliselt täidetud	3.3.2, 5.5

Funktsionaalsete nõuete osas on prototüübis täielikult realiseeritud kõik kaks tuumikrolli — võistkonna ja mängujuhi liidesed —, mis kataksid mälumängu turniiri täisvoo alates liitumisest kuni lõpliku edetabelini. Osaliselt täidetud staatuses on apellatsioonide menetlus ja täiendavad mängujuhi rollid (ADMIN, SCORER): nende andmemudelid on ette valmistatud (vt 3.3.2), kuid kasutajaliideseid ei ole prototüübis realiseeritud. Mitme kasutaja paralleelne töö halduspaneelis on sellegipoolest võimalik ühe jagatud konto kaudu. Selline skoobi piiramine sai tellijaga teadlikult kokku lepitud, et keskenduda turniiri tuumikfunktsionaalsuse kvaliteetsele realiseerimisele.

Tabel 11. Mittefunktsionaalsete nõuete täidetus.

Nõue	Sihtväärtus	Mõõdetud ehk saavutatud	Olek	Tõendus
Latentsus seadmete vahel	< 1 s	HTTP p99 $\leq$ 50 ms (äriloogika), WebSocket sündmused < 1 s	Täidetud	5.1.1, 5.1.2
Andmete säilivus tehniliste tõrgete korral	täielik	PostgreSQL ACID + Redise taastuv olek + audit-jälg	Täidetud	3.3.3, 4.5.2
Skaleeritavus — võistkonnad	≥ 100 samaaegselt	mõõdetud tüüpilisel turniirimahul (T4, 15 võistkonda); küllastumise märke ei täheldatud	Mõõdetud reaalmahul	5.1.3
Ristplatvormsus	Web, iOS, Android	Expo veebiversioon töötab; natiivpakendi ehitus testitud	Täidetud	3.1.1, 4.1.2
Mitmekeelsus	et / ru / en	i18n raamistik, kõik kasutajaliidese tekstid lokaliseeritud	Täidetud	4.1.2
GDPR-vastavus	EL territoorium	Hetzneri Soome andmekeskus ja Mixpaneli EU-instants	Täidetud	3.2.6

Mittefunktsionaalsetes nõuetes täidetakse latentsuse, andmete säilivuse, ristplatvormsuse, mitmekeelsuse ja andmekaitse nõuded mõõdetud või kontrollitud kujul. Skaleeritavuse nõue — sihtväärtusega vähemalt 100 võistkonda samaaegselt — nõuab eraldi tõlgendust. Sihtväärtus seati alapeatükis 2.4 teadliku turvavaruga, suunatuna tulevastele potentsiaal-

setele kasutusjuhtudele (näiteks klubidevahelised või rahvusvahelised turniirid); tellija käesolevas turniiripraktikas osaleb keskmiselt 10–20 ning suuremate turniiride puhul kuni 30 võistkonda, väga erandlikel juhtudel kuni 60. Käesoleva prototüübi raames on süsteemi käitumine empiiriliselt valideeritud reaalse turniirimahu juures (T4, 15 võistkonda), mis langeb tellija tüüpilise kasutusjuhu raamesse, ning ühelgi mõõdetud kihil ei täheldatud küllastumise märke ega ootamatuid süsteemivigu (vt 5.1.3). Sihtväärtuse 100 võistkonda formaalne kvantitatiivne kinnitamine eeldab sünteetilist koormuskatset astmeliselt (näiteks Apache JMeteriga 25, 50, 100, 200 ühendust), kuna reaalajas süsteemide jõudlus ei skaleeru lineaarselt — kasvavate ühenduste arvu korral lisanduvad mittelineaarsed komponendid (vt 5.1.3). Selline koormuskatse jäi käesoleva töö skoobist välja ning on planeeritud järgmise iteratsiooni jaoks (vt 5.5). Arhitektuurselt on suurema mahu tugi ette valmistatud: mängumootori ühendusvaba ülesehitus (vt 3.2.2) lubab game-engine'i edaspidi eraldada eraldi teenuseks, mis on horisontaalse skaleerimise eeldus.

Töö peamine eesmärk — luua digitaalne alternatiiv paberipõhisele turniirikorraldusele, mis vähendab korraldajate töökoormust ja tagab võistkondadele võrdsed tingimused — on saavutatud, mida kinnitavad nii mõõdetud jõudluse näitajad (vt 5.1) kui digiülemineku mõõdetud mõju (vt 5.3) ja tellija positiivne hinnang (vt 5.2.3).

5.4.1 Tulemuste piirangud ja valiidsuse ohud

Käesoleva töö valideerimise kvantitatiivsete tulemuste tõlgendamisel tuleb arvesse võtta järgmisi piiranguid:

Andmestiku maht. Mõõtmised pärinevad kolmest turniirist (T2 hübriid, T3 ja T4 täismahus rakendusepõhised, vt peatükk 5 algust). Statistilise üldistatavuse tagamiseks oleks vaja korrata mõõtmist mitmel turniiril, sealhulgas erinevatel mahtudel (40+ võistkonda) ja erinevatel korraldusliku tugevusega kohtades. Sünteetilise koormuskatse läbiviimine 100+ samaaegse võistkonnaga jäi prototüübi skoobist välja ning on planeeritud järgmise iteratsiooni jaoks (vt 5.5).

Vaatleja efekt. Mängijad ja korraldajad olid teadlikud, et katsetatakse uut platvormi, mis võis suurendada nende tähelepanelikkust kasutatavuse-vastuolude märkamisel ning mõjutada lõpuankeedi vastuseid positiivse poole.

Autorite topelt-roll. Töö autorid olid samaaegselt nii rakenduse arendajad kui ka tulemuste analüüsijad, mis võib põhjustada eelarvamusi tagasiside tõlgendamises. Seda riski leevendati objektiivsete telemeetria-andmete (Mixpanel, Prometheus) ja subjektiivsete ankeedivastuste ristkontrolliga ning tellija sõltumatu sisendi kasutamisega.

Konkreetne organisatsiooniline kontekst. Tulemused on saadud koostöös ühe Eesti mälumänguklubiga, mille turniirikorraldus võib erineda teiste organisatsioonide praktikatest (näiteks rahvusvahelised mastaapsed sünkroonturniirid, vt 2.3). Üldistamine sellesse konteksti eeldaks täiendavat valideerimist.

Paberipõhise ja digitaalse turniiri võrreldavus. Paberipõhise vaatluse turniir (T1, 13 võistkonda) ja digitaalne valideerimisturniir (T3, 14 võistkonda) ei olnud identsed mahu osas; see võib mõjutada absoluutse ajakulu võrdlust, kuid ei mõjuta proportsioonilisi näitajaid (inimtöökoormuse 82% vähenemine).

Mängujuhi õppimiskõver. Digitaalse turniiri ajamõõtmised pärinevad süsteemi esimesest tootmiskasutusest, kus mängujuht ei olnud liidese kasutamisega harjunud. Küsimuste läbiviimise otsene ajaline erinevus paberipõhisega jäi sellest tulenevalt minimaalseks (51 sekundit 36 küsimuse peale, vt Tabel 9). Kogenud mängujuhi puhul on oodata kiiremat vastuste hindamist ja faaside vahetamist, mis võib anda täiendava ajasäästu vastuste kogumise automatiseerimisest. Selle hüpoteesi kontrollimiseks on vaja korduskatset kogenud kasutajaga.

5.5 Edasised arendusvõimalused

Käesolev prototüüp katab turniiri tuumikfunktsionaalsuse, kuid mitmed laiendused on vajalikud platvormi jätkuvaks arendamiseks tootmistasemele:

Apellatsioonide menetlus. Andmemudel ja serveripoolsed sündmused on ette valmistatud; puudub mängujuhi ja mängija UI-poolne realisatsioon. Edasiarenduse käigus on plaanis lisada apellatsiooni-aken kindla aja jooksul pärast vastuse hindamist ning selle menetlemise liides mängujuhile.

Täiendavad mängujuhi rollid. ADMIN-roll võimaldab turniiri korraldajatel pääseda süsteemi haldusvaadetele; SCORER-roll võimaldab hindamise delegeerimist eraldi kasutajatele, mis

on suuremate turniiride puhul kriitilise tähtsusega.

Natiivse rakenduse väljaandmine. Praegune Expo veebiversioon katab põhilist kasutust, kuid natiivne pakend (iOS App Store, Google Play) annaks juurdepääsu seadme haptilisele tagasisidele, süsteemsetele teavitustele ja taustal töötavale WebSocketi-ühendusele (vt 3.1.1).

Mikroteenusarhitektuur. Mängumootori ühendusvaba ülesehitus lubab tulevikus eraldada game-engine eraldi teenuseks, mis kommunikeerib host- ja player-teenustega Redise Pub/Sub-i kaudu. See on vajalik horisontaalse skaleerimise jaoks, kui samaaegsete turniiride arv ületab ühe Node.js protsessi talluvuse.

Koormustestimine ja jõudluse profileerimine. Sünteetiline koormuskatse Apache JMeteriga astmeliselt mitmel tasemel (25, 50, 100, 200 ühendust) on vajalik, et tuvastada esimese kvalitatiivse halvenemise (p99-viiteaeg ületab 1 sekundi) tekkimise koht ning kinnitada mittefunktsionaalse skaleeritavuse nõude (vt 2.4) täidetus 100 võistkonna sihtväärtuse juures.

Avalik vaade publikule. Suure ekraani vaade, mis kuvab projektoril aktiivse küsimuse, jooksva edetabeli ja turniiri faasi. See ei kuulunud käesoleva prototüübi skoopi, kuna fookus oli mängija- ja mängujuhi-tuumiku stabiliseerimisel.

5.6 Tiimitöö

Käesolev projekt ja bakalaureusetöö valmisid kahe üliõpilase, Daniel Tjulinovi ja Alexander Smirnovi, tihedas koostöös. Tööülesanded jaotati vastavalt arendusetappidele ja autorite spetsialiseerumisele, tagades süsteemi tervikliku valmimise.

Daniel Tjulinovi panus:

- Kasutajaliidese kavandamine: Projekteeris rakenduse esialgsed kasutusvood ning koostas mängija liidese ekraanide disaini.
- Taustarakendus ja andmehaldus: Kavandas süsteemi üldarhitektuuri ning realiseeris serveripoolse tuumikloogika, andmebaasimudeli ja vahemälulahenduse.
- Telemeetria ja monitooring: Seadistas täies mahus Prometheus ja Grafana seirekeskkonna ning teostas süsteemi jõudluse ja skaleeritavuse analüüsi.

- Klientrakendus: Alustas eesrakenduse arendusprotsessi ning vastutas mängija vaadete lõpliku viimistlemise ja vigade paranduse eest.
- Kommunikatsioon: Koordineeris suhtlust tellija MIS? KUS? MILLAL? MTÜ-ga.
- Kirjalik osa: Koostas bakalaureusetöö 3. peatüki («Metoodika ja arhitektuur») ning 4. peatüki («Rakenduse prototüübi realiseerimine»).
- CI/CD: Vastutas pideva integratsiooni ja paigalduse protsesside seadistamise eest.

Alexander Smirnovi panus:

- Kasutajaliidese kavandamine: Realiseeris ja viis lõpuni mängujuhi haldusliidese detailse disaini.
- Klientrakendus: Jätkas eesrakenduse arendust, keskendudes peamiselt mängujuhi paneeli keeruka funktsionaalsuse ja haldusvaadete realiseerimisele.
- Analüütika: integreeris Mixpaneli analüütikateenuse sündmuste kogumiseks.
- Taustarakendus: Realiseeris serveripoolse rakenduse esmase versiooni.
- Infrastruktuur: Seadistas serveri infrastruktuuri rakenduste majutamiseks.
- Kirjalik osa: Koostas töö 1. peatüki («Sissejuhatus»), 2. peatüki («Taust ja analüüs»).

Mõlemad autorid osalesid ühiselt süsteemi nõuete kaardistamisel, UI/UX kontseptsioonide läbimõtlemisel, koodi ülevaatustel ning jooksvates tehnilistes aruteludes. Ühiselt viidi läbi ka reaalajas toimunud testturniirid, süsteemi valideerimine praktilistes tingimustes ning kogutud andmete tõlgendamine. Samuti kirjutati ühistööna bakalaureusetöö 5. peatükk («Tulemused ja analüüs») ja 6. peatükk («Kokkuvõte»).

Projektis osalejate tööaja jaotus ja individuaalne kronometraaž on esitatud Lisas 6.

6 Kokkuvõte

Käesoleva bakalaureusetöö raames realiseeriti veebipõhine mälumängu-platvorm, mis asendab paberipõhise «Mis? Kus? Millal?» turniirikorralduse digitaalse, reaalajas toimiva lahendusega [35]. Süsteem koosneb NestJS-põhisest serveriprotsessist, mis haldab mängu olekumasinat ja reaalajas suhtlust Socket.IO kaudu, ning Expo / React Native klientrakendusest, mis pakub ühest koodibaasist kasutajaliideseid veebibrauseritele ja iOS-/Android-seadmetele. Püsivate andmete jaoks kasutatakse PostgreSQL andmebaasi, lühiajalist mänguseisundit hoitakse Redises.

Töö käigus analüüsiti paberipõhise turniirikorralduse pudelikaelad, koostati funktsionaalsete ja mittefunktsionaalsete nõuete spetsifikatsioon ning realiseeriti prototüüp, mille jõudlust, kasutatavust ja mõju mõõdeti reaalsetel testturniiridel. HTTP-otspunktide äriloogika-päringute p99-viiteaeg jäi alla 50 millisekundi; WebSocketi-tasandil ei registreeritud kogu turniiri vältel ühtegi ootamatut süsteemiviga. Mängijate keskmine 5-palline hinnang oli 4,4 ja tellija kinnitas, et lahendus vähendab oluliselt korraldajate töökoormust ning muudab võistkondade vahelised tingimused võrdsemaks. Paberipõhise ja digitaalse korraldusviisi võrdlus näitas, et rakendus elimineerib voorudevahelised tehnilised pausid täielikult ning vähendab kumulatiivset inimtöökoormust 5-liikmeliselt meeskonnalt 1 mängujuhile, andes $\sim$82% kokkuhoiu turniiri kohta.

Loodud platvorm on tellijale MIS? KUS? MILLAL? MTÜ-le üle antud ja kasutusvalmis järgmistel reaalturniiridel.

Kasutatud kirjandus

[1] MIS? KUS? MILLAL? MTÜ. *Tallinna Mälumänguklubi (MTÜ) Mis? Kus? Millal?* 2011. URL: https://mkmestonia.blogspot.com/ (vaadatud 03.05.2026).

[2] Meta Platforms. *React Native Documentation*. 2026. URL: https://reactnative.dev/docs/getting-started (vaadatud 08.04.2026).

[3] OpenJS Foundation. *Node.js Documentation*. URL: https://nodejs.org/docs/ (vaadatud 01.05.2026).

[4] Kamil Mysliwiec. *NestJS — A progressive Node.js framework*. URL: https://docs.nestjs.com/ (vaadatud 01.05.2026).

[5] The PostgreSQL Global Development Group. *PostgreSQL 16 Documentation*. URL: https://www.postgresql.org/docs/16/index.html (vaadatud 01.05.2026).

[6] Rahvusvaheline Klubiühingute Liit (MAIIK). *Mälumänguspordi reeglid (MAIIK koodeks)*. URL: https://rating.chgk.gg/rules/ (vaadatud 03.05.2026).

[7] Kahoot! AS. *What is Kahoot!?* URL: https://kahoot.com/what-is-kahoot/ (vaadatud 01.05.2026).

[8] Quizizz Inc. (Wayground). *Wayground*. URL: https://wayground.com/?lng=en (vaadatud 03.05.2026).

[9] Mentimeter AB. *Mentimeter*. URL: https://www.mentimeter.com/ (vaadatud 01.05.2026).

[10] Cisco Systems, Inc. *Explore Slido*. URL: https://www.slido.com/product (vaadatud 01.05.2026).

[11] Google LLC. *Google Forms*. URL: https://workspace.google.com/products/forms/ (vaadatud 03.05.2026).

[12] Sergey Sabirov. *Welcome to the Open Quiz!* URL: https://www.open-quiz.com/ (vaadatud 03.05.2026).

[13] Expo. *Expo Documentation*. URL: https://docs.expo.dev/ (vaadatud 01.05.2026).

[14] Google LLC. *Flutter documentation*. URL: https://docs.flutter.dev/ (vaadatud 01.05.2026).

[15] Google. *Dart programming language*. URL: https://dart.dev/ (vaadatud 01.05.2026).

[16] Andreas Biørn-Hansen, Tim A. Majchrzak ja Tor-Morten Grønli. *Progressive Web Apps: The Possible Web-Native Unifier for Mobile Development*. 2017. URL: https://www.scitepress.org/papers/2017/63537/63537.pdf (vaadatud 01.05.2026).

[17] MDN Web Docs. *Screen Wake Lock API*. Juuli 2025. URL: https://developer.mozilla.org/en-US/docs/Web/API/Screen_Wake_Lock_API (vaadatud 01.05.2026).

[18] Brady Eidson ja Jen Simmons. *Web Push for Web Apps on iOS and iPadOS*. Veebr 2023. URL: https://webkit.org/blog/13878/web-push-for-web-apps-on-ios-and-ipados/ (vaadatud 01.05.2026).

[19] OpenJS Foundation. *Express — Node.js web application framework*. URL: https://expressjs.com/ (vaadatud 01.05.2026).

[20] OpenJS Foundation. *Fastify: Fast and low overhead web framework, for Node.js*. URL: https://fastify.dev/ (vaadatud 01.05.2026).

[21] VMware Tanzu. *Spring Boot Documentation*. URL: https://spring.io/projects/spring-boot (vaadatud 01.05.2026).

[22] Prisma Data Inc. *Introduction to Prisma*. URL: https://www.prisma.io/docs (vaadatud 01.05.2026).

[23] Redis Ltd. *Redis Documentation*. URL: https://redis.io/docs/ (vaadatud 01.05.2026).

[24] Socket.IO. *Socket.IO Documentation*. Märts 2026. URL: https://socket.io/docs/v4/ (vaadatud 01.05.2026).

[25] Ian Fette ja Alexey Melnikov. *The WebSocket Protocol*. Google, Inc. Dets 2011. URL: https://www.rfc-editor.org/rfc/rfc6455.txt (vaadatud 01.05.2026).

[26] WHATWG. *HTML Living Standard — Server-sent events*. Apr 2026. URL: https://html.spec.whatwg.org/multipage/server-sent-events.html (vaadatud 01.05.2026).

[27] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm (vaadatud 15.05.2026).

[28] Michael Jones, John Bradley ja Nat Sakimura. *JSON Web Token (JWT)*. Mai 2015. URL: https://www.rfc-editor.org/rfc/rfc7519.html (vaadatud 01.05.2026).

[29] The Prometheus Authors. *Prometheus Documentation*. URL: https://prometheus.io/docs/introduction/overview/ (vaadatud 01.05.2026).

[30] Mixpanel. *Mixpanel Documentation*. URL: https://docs.mixpanel.com/docs/what-is-mixpanel (vaadatud 03.05.2026).

[31] David Harel. *Statecharts: A visual formalism for complex systems*. Juuni 1987. URL: https://www.sciencedirect.com/science/article/pii/0167642387900359 (vaadatud 01.05.2026).

[32] Figma Inc. *Figma documentation*. URL: https://help.figma.com/ (vaadatud 01.05.2026).

[33] A. Jain *et al. A comparative study of visual and auditory reaction times on the basis of gender and physical activity levels of medical first year students*. Aug 2015. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC4456887/ (vaadatud 03.05.2026).

[34] Jakob Nielsen. *Usability Engineering*. Boston: Academic Press, 1993.

[35] *Lõputöö tulemusel realiseeritud mälumängu platvorm*. 2026. URL: https://what-where-when.eu/.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Meie, Alexander Smirnov ja Daniel Tjulinov

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "«Mis? Kus? Millal?» võistkondlike mänguturniiride töövoogude digitaliseerimise uurimine ja veebipõhise platvormi prototüübi kavandamine, realiseerimine ning hindamine", mille juhendaja on Siim Rebane
    1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
    1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Oleme teadlikud, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autoritele.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

18.05.2026

[1] Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Olemasolevate mälumängu- ja viktoriiniplatvormide funktsionaalne võrdlus

Olemasolevate digiplatvormide funktsionaalsuse võrdlus «Mis? Kus? Millal?» kooliturniiride kontekstis. Kriteeriumid on rühmitatud nelja kategooriasse vastavalt peatükis 2.3 kirjeldatud puudujääkidele. Hindamine põhineb platvormide avalikul dokumentatsioonil seisuga november 2025. Tähistused: Jah — kriteerium on täielikult toetatud; Osaline — toetatud piirangutega või lisatööriistade abil; Ei — kriteerium ei ole toetatud.

Kriteerium	Kahoot!	Wayground	Mentimeter /	Google	Open-Quiz	ChGK
A. Vastuste käsitlemine ja valideerimine						
Lühikesed vabatekstilised vastused (avatud küsimus)	Osaline	Osaline	Jah	Jah	Jah	Jah
Mängujuhi käsitsi valideerimine reaalajas (õige/vale/vaieldav)	Ei	Ei	Ei	Ei	Jah	Osaline
Sünonüümide ja trükivigade tolerantsi konfigureerimine	Osaline	Osaline	Ei	Ei	Jah	Ei
Struktureeritud apellatsioonide / protestide menetlemine	Ei	Ei	Ei	Ei	Ei	Ei
B. Reaalaja sünkroniseerimine ja ajalised piirangud						
Kahefaasiline taimer (60 s arutelu + 10 s vastus) automaatse lukust	Ei	Ei	Ei	Ei	Jah	Osaline
Tsentraalselt juhitud sünkroonne küsimuse esitamine kõigile tiimide	Jah	Jah	Jah	Jah	Ei	Osaline
Reaalajas uuenev edetabel	Jah	Jah	Jah	Jah	Jah	Ei
Vastuste sünkroonne kogumine ja samaaegne avalikustamine	Osaline	Osaline	Osaline	Osaline	Ei	Osaline
C. Mängujuhi tööriistad ja korraldajakogemus						
Mängu elutsükli juhtimine (paus, voor edasi, küsimuse kordamine)	Osaline	Osaline	Osaline	Osaline	Ei	Jah
Voorude ja kategooriate struktureeritud konfigureerimine	Osaline	Osaline	Ei	Ei	Jah	Osaline
Madal sisenemisbarjäär korraldajale (ei vaja eraldi tööriistu)	Jah	Jah	Jah	Jah	Ei	Ei
Mitme operaatori (žürii + juhi) paralleelne töö ühe turniiri kallal	Osaline	Osaline	Ei	Ei	Jah	Ei
Ajalooline statistika ja sessioonide aruanded	Jah	Jah	Osaline	Jah	Jah	Ei
D. Kasutuselevõtu tingimused ja küpsus						
Mobiilne ligipääs ilma rakenduse paigaldamiseta (BYOD)	Jah	Jah	Jah	Jah	Osaline	Jah
Tasuta kasutus 60+ tiimi puhul	Ei	Ei	Ei	Jah	Jah	Jah
Eestikeelne kasutajaliides	Jah	Jah	Jah	Jah	Osaline	Ei
Eelvalmistatud küsimusepank / mallid mälumängu vormingus	Ei	Ei	Ei	Ei	Jah	Jah
Toote küpsus ja pikaajaline tugi	Jah	Jah	Jah	Jah	Osaline	Osaline
LMS-integratsioon (Moodle, Google Classroom)	Jah	Jah	Osaline	Jah	Ei	Ei

Joonis 15. Olemasolevate digiplatvormide funktsionaalsuse võrdlus «Mis? Kus? Millal?» kooliturniiride kontekstis.

Lisa 3 – Mixpanel sündmuste loend

Käesolevas lisas on esitatud klientrakenduse poolt Mixpaneli teenusesse saadetavate sündmuste täielik nimekiri seisuga turniir T4. Loend on jaotatud kolmeteistkümneks rühmaks, mis vastavad rakenduse funktsionaalsetele osadele: vaikimisi konteksti omadused (Tabel 12), rakenduse elutsükkel ja navigatsioon (Tabel 13), avalehe sündmused (Tabel 14), mängujuhi autentimine (Tabel 15) ja mängude haldamine (Tabel 16 ja Tabel 17), mängu jooksva juhtimise sündmused (Tabel 18), serveri tõuke-sündmuste vaatlused (Tabel 19), mängija töövoog (Tabel 20), tulemuste ekraan (Tabel 21), tagasiside vorm (Tabel 22) ja tänan-ekraan (Tabel 23) ning sokli elutsükkel (Tabel 24). Sündmuste ja omaduste nimetused on jäetud ingliskeelseteks, kuna need vastavad otse klientrakenduse lähtekoodis kasutatavatele identifikaatoritele.

Tabel 12. Super-omadused: iga sündmuse külge kinnitatud vaikimisi kontekst.

Võti	**Kus seatakse**	**Tähendus**
`app_version`, `platform`, `locale`, `environment`	`mixpanel.init`	Standard system properties
`role`	After login/signup, on host-admin screen, on player socket connect	`host` / `player`
`host_id`	After login/signup	Host id
`session_present`	After init / login / logout	Whether a host session is active
`game_id`	On host admin (`/game/[gameId]`) and in player game	Game id
`team_id`	After player socket connect	Team id
`host_screen`	On host admin	`create_game` / `game_admin`
`is_new`	On host admin	Creating a new game vs editing

Tabel 13. Rakenduse elutsükli ja navigatsiooni sündmused.

Sündmus	Millal	Võtmeomadused
`App Opened`	After `mixpanel.init()` on app startup	`has_session`, `init_time_ms`, `distinct_id`
`Session Restored`	On startup, when a stored host session is found	`host_id`, `role`
`Session Cleared`	After host logout	—
`Host Session Expired`	API returned 401 while loading the games list	`response_time_ms`
`Screen Viewed`	Any navigation (`mixpanel.screen` in `_layout`)	`screen_name`, `route`, `role`

Tabel 14. Avalehe sündmused.

Sündmus	Millal	Omadused
`Home CTA Clicked`	Click on a button on the home screen	`cta` ∈ `join_game` / `feedback`

Tabel 15. Mängujuhi autentimise sündmused.

Sündmus	Millal	Omadused
`Host Login Submitted`	Click on “Login”	`email_domain`
`Host Login Succeeded`	Successful login	`host_id`, `role`, `response_time_ms`
`Host Login Failed`	Login error	`email_domain`, `error_message`, `status`, `response_time_ms`
`Host Login Show Password Toggled`	Toggle password visibility	`visible`
`Host Login Forgot Password Clicked`	Click on “Forgot password?”	`email_domain`
`Host Login Register Link Clicked`	Navigation to signup	—
`Host Signup Submitted`	Click on “Sign up”	`email_domain`, `agreed_terms`, `name_filled`, `password_length`, `passwords_match`
`Host Signup Succeeded`	Successful signup	`host_id`, `role`, `response_time_ms`
`Host Signup Failed`	Signup error	`email_domain`, `error_message`, `status`, `response_time_ms`

Sündmus	Millal	Omadused
`Host Signup Show Password Toggled`	Visibility of password / confirm field	`field`, `visible`
`Host Signup Terms Toggled`	Agreement checkbox	`agreed`
`Host Signup Terms Link Clicked`	Click on T&C / Privacy link	`link` ∈ `terms_and_conditions` / `privacy_policy`
`Host Logout Clicked`	"Log out" button on the setup screen	—

Tabel 16. Mängujuhi mängude nimekirja sündmused.

Sündmus	Millal	Omadused
`Host Games List Viewed`	After the list is loaded	`result` ∈ `success/fail`, `items_count`, `response_time_ms`, `error_message`, `status`
`Host Game Create Started`	"Create game" button	—
`Host Game Opened`	Click on a game card	`game_id`

Tabel 17. Mängujuhi mängu redaktori sündmused.

Sündmus	Millal	Omadused
`Host Editor Loaded`	Existing game loaded successfully	`game_id`, `version`, `rounds_count`, `response_time_ms`
`Host Editor Load Failed`	Load error	`game_id`, `error_message`, `status`, `response_time_ms`
`Host Editor Date Changed`	Event date changed	`game_id`, `is_new`, `has_value`
`Host Editor Setting Changed`	Any toggle/value in the Settings section	`setting_key`, `value`, `previous_value`, `value_type`
`Host Editor Round Added`	Round added	`game_id`, `is_new`, `round_number`, `total_rounds`
`Host Editor Round Removed`	Round removed	`game_id`, `round_id`, `round_number`, `questions_in_round`, `was_persisted`

Sündmus	Millal	Omadused
Host Editor Round Selected	Round selected in the editor	round_id, round_number
Host Editor Question Added	Question added	round_id, round_number, question_number, total_questions_in_round
Host Editor Question Removed	Question removed	round_id, question_id, question_number, was_persisted
Host Editor Question Selected	Question selected in the editor	round_id, question_id, question_number
Host Editor Category Added	Category added	is_new, name_length, has_description
Host Editor Category Updated	Category updated	category_id, name_changed, description_changed
Host Editor Category Removed	Category removed	category_id, was_persisted
Host Editor Team Added	Team added	category_id, name_length, code_length
Host Editor Team Updated	Team updated	team_id, name_changed, code_changed, category_changed, category_id
Host Editor Team Removed	Team removed	team_id, was_persisted
Host Editor Team Category Selected	Picking a category chip when adding/editing a team	category_id, previous_category_id, is_editing
Host Game Create Submitted	Create form submit	—
Host Game Create Succeeded	Game created	game_id
Host Game Create Failed	Create error	error_message, status
Host Game Saved Submitted	“Save” button in the editor	game_id, rounds_count, teams_count, categories_count, questions_count, deleted_*_count
Host Game Saved Succeeded	Saved	game_id, version, response_time_ms

Sündmus	Millal	Omadused
Host Game Saved Failed	Save error	game_id, error_message, status, response_time_ms

Tabel 18. Mängujuhi mängu jooksva juhtimise sündmused.

Sündmus	Millal	Omadused
Host Game Mounted	/game/[gameId] screen opened	game_id, is_new
Host Game Loaded	When editor.loaded resolves	game_id, version, rounds_count, questions_count, teams_count, categories_count, status
Host Game Back Clicked	Back button in the NavBar	game_id, is_new, active_tab, game_status
Host Tab Viewed	Tab change (Settings/Answers/Leaderboard/Teams)	game_id, tab, tab_from, is_new
Host Admin Sync Emitted	Emitting admin:sync after socket connect	game_id
Host Game Start Clicked	"Start game" button	game_id
Host Question Prepared	prepare_question command (incl. Restart and Prev)	game_id, question_id
Host Question Started	start_question command	game_id, question_id
Host Next Question	"Next" button	game_id
Host Prev Question Clicked	"< Prev" button	game_id, active_question_id, current_index, total_questions, has_prev
Host Active Question Changed	activeQuestionId changed (from broadcast)	from_question_id, to_question_id, to_question_number
Host Timer Resumed	resume_timer command	game_id
Host Timer Paused	pause_timer command	game_id
Host Timer Toggle Clicked	Click on the play/pause button in the sidebar	current_action $\in$ start_question/resume_timer/pause_timer, phase, is_paused, time_left_s

Sündmus	Millal	Omadused
Host Adjust Time Clicked	±10 sec buttons	direction, delta_s, phase, time_left_s, active_question_id
Host Time Adjusted	adjust_time emit (internal command)	game_id, delta_s
Host Question Stopped	“Stop question”	game_id
Host Game Finished	“Finish game”	game_id
Host Answer Judged	Verdict on a team’s answer	game_id, answer_id, verdict
Host Code Copy Clicked	Game passcode copy	has_passcode, game_status
Host Answers Question Selected	Question circle on the Answers Dashboard	question_id, question_number, round_id, from_question_id, is_active
Host Game Export Started	XLSX export started	game_id, format, teams_count
Host Game Export Succeeded	Export completed	game_id, size_bytes, response_time_ms
Host Game Export Failed	Export error	game_id, error_message, status, response_time_ms

Tabel 19. Serveri tõuke-sündmuste vaatlused kliendi poolt.

Sündmus	Millal	Omadused
Game Status Changed	Server broadcasted a status change	game_id, from, to
Game Phase Changed	Server broadcasted a phase change	game_id, from, to, active_question_id, active_question_number
Timer Paused Broadcast	timer_paused broadcast	game_id, phase, time_left_s, active_question_id
Timer Resumed Broadcast	timer_resumed broadcast	game_id, phase, time_left_s, active_question_id

Tabel 20. Mängija töövoo sündmused.

Sündmus	**Millal**	**Omadused**
`Player Join Mounted`	Code-entry screen opened	—
`Player Join Code Entered`	OTP filled (4 digits)	`code_length`, `input_method` ∈ `paste/type`, `has_error_before`
`Player Join Code Submitted`	Code check start/success/fail	`code`, `result` ∈ `pending/success/fail`, `attempt`, `previous_failed_attempts/failed_attempts`, `response_time_ms`, `teams_count`, `game_id`, `error_message`
`Player Join Back Clicked`	"Back" button	`digits_entered`, `attempts`, `failed_attempts`
`Player Select Team Mounted`	Team-selection screen opened	`game_id`, `code`, `teams_count`, `available_teams_count`
`Player Team Selected`	Tap on an available team	`game_id`, `team_id`, `team_name`, `is_available`
`Player Team Taken Pressed`	Tap on a taken team	`game_id`, `team_id`, `team_name`
`Player Select Team Back Clicked`	"Back" button	`game_id`, `had_selection`
`Player Entered Game`	Continue → navigation into the game	`game_id`, `team_id`
`Player Game Mounted`	Game screen opened	`game_id`, `team_id`, `team_name`
`Player Tab Changed`	Bottom tab change	`game_id`, `team_id`, `tab_from`, `tab_to`, `game_phase`, `game_status`
`Player Join Game Emitted`	`join_game` emit after `connect`	`game_id`, `team_id`
`Player Game Phase Observed`	Phase changed via `sync_state/timer_update`	`from`, `to`, `active_question_id`, `seconds`
`Player Game Status Observed`	Game status changed	`from`, `to`, `source`
`Player Game Active Question Observed`	Active question changed	`from_question_id`, `to_question_id`, `to_question_number`
`Player Answer Input Focused`	First focus on the answer input for the current question	`question_number`, `phase`, `time_left_s`, `has_existing_answer`

Sündmus	Millal	Omadused
Player Answer Submit Clicked	Click on "Submit"/"Resubmit"	question_number, phase, time_left_s, answer_length, is_resubmit
Player Answer Submitted	submit_answer emit	game_id, team_id, participant_id, question_id, question_number, phase, time_left_s, answer_length
Player Answer Ack	Server confirmed answer_received	game_id, team_id, participant_id, question_id, latency_ms
Player History Updated	history_update received	count
Player Leaderboard Updated	leaderboard_update received	count
Player Mini Widget Pressed	Tap on the mini widget (collapsed timer)	game_id, team_id, phase, time_left_s, from_tab
Player Game Finished Viewed	gameStatus === FINISHED observed in PlayTab	participant_id, history_count
Player Results Redirect	Auto-redirect from /game to /game-results	game_id, team_id, participant_id, reason ∈ game_finished / join_blocked_finished, has_stored_participant
Player Feedback Clicked	"Give feedback" button on the results screen	source ∈ game_results, game_id, participant_id, teams_count, my_score
Player Socket Error	Socket error event	game_id, team_id, error_message

Tabel 21. Mängu tulemuste ekraani sündmused.

Sündmus	Millal	Omadused
Player Game Results Mounted	/game-results screen opened	game_id, participant_id, has_participant_id
Player Game Results Loaded	Leaderboard fetch succeeded	game_id, teams_count, response_time_ms
Player Game Results Load Failed	Leaderboard fetch failed (or no gameId)	game_id, reason, error_message, response_time_ms

Sündmus	Millal	Omadused
Player Game Results My Team	Current participant resolved on the leaderboard	game_id, participant_id, my_score, my_rating, category_id, teams_in_category, my_rank
Player Game Results Back Clicked	"Back" button at the bottom of the screen	teams_count, had_my_team, feedback_visible

Tabel 22. Mängija tagasiside vormi sündmused.

Sündmus	Millal	Omadused
Player Feedback Mounted	Feedback screen opened	from_home, game_id, participant_id, has_game_context
Player Feedback Form Loaded	GET /player/feedback-form succeeded and parsed	sections_count, chips_count, response_time_ms
Player Feedback Form Load Failed	Fetch error or parse failure	reason ∈ parse_failed / fetch_failed, error_message, response_time_ms
Player Feedback Rating Selected	Star tap (1–5)	rating, previous_rating
Player Feedback Chip Toggled	Chip in a dynamic section toggled	section_key, chip_key, selected, section_total_after
Player Feedback Comment Started	First non-empty character typed in the comment	—
Player Feedback Submit Clicked	"Submit" tap (passes validation pre-submit)	rating, sections_with_selection, total_chips_selected, comment_length, from_home, has_game_context
Player Feedback Submit Blocked	Validation prevents submit	reason ∈ rating_required / missing_game_context
Player Feedback Submitted	POST /player/feedback succeeded	rating, sections_with_selection, total_chips_selected, comment_length, from_home, has_game_context, response_time_ms
Player Feedback Submit Failed	POST /player/feedback failed	error_message, from_home, has_game_context, response_time_ms

Sündmus	Millal	Omadused
`Player Feedback Skipped`	Primary button pressed without any input (Skip / Return Home)	`from_home`, `has_game_context`

Tabel 23. Tänan-ekraani sündmused.

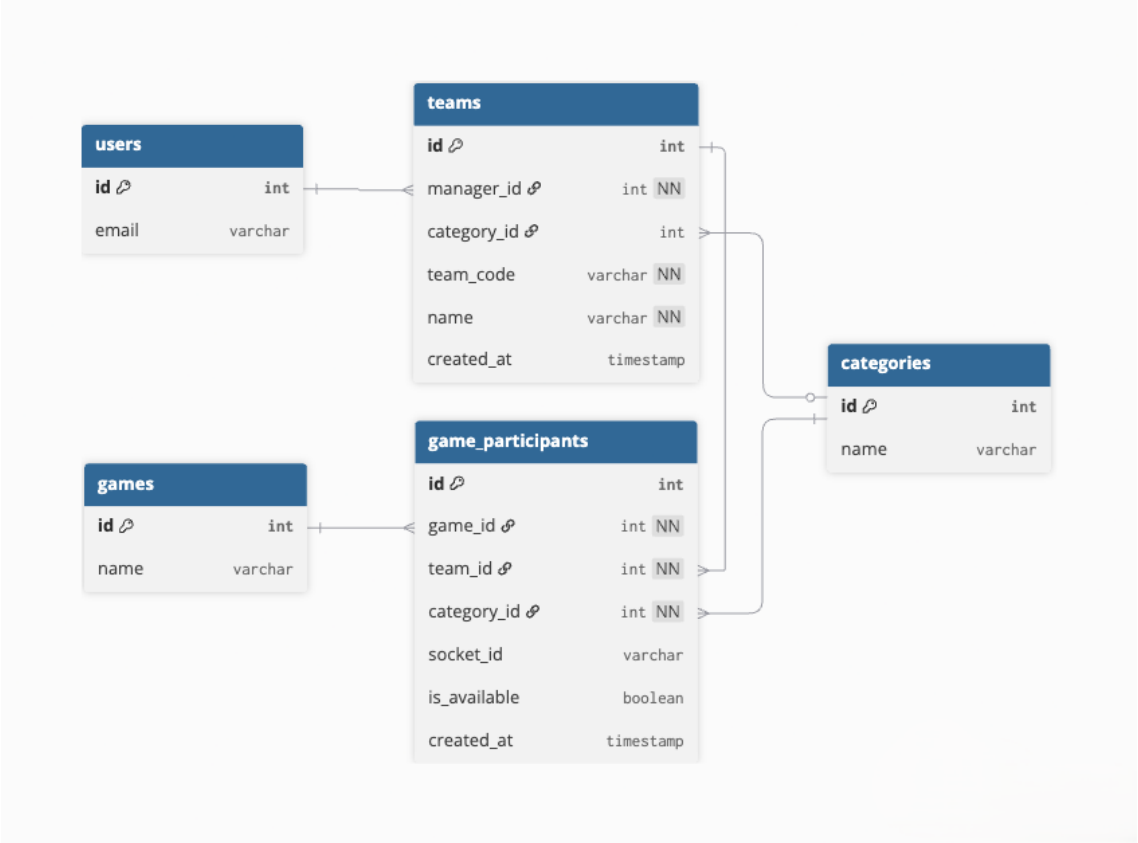
Sündmus	Millal	Omadused
`Player Thank You Mounted`	`/thank-you` screen opened	—
`Player Thank You Return Home Clicked`	“Return to Home” button	—

Tabel 24. Sokli elutsükli sündmused.

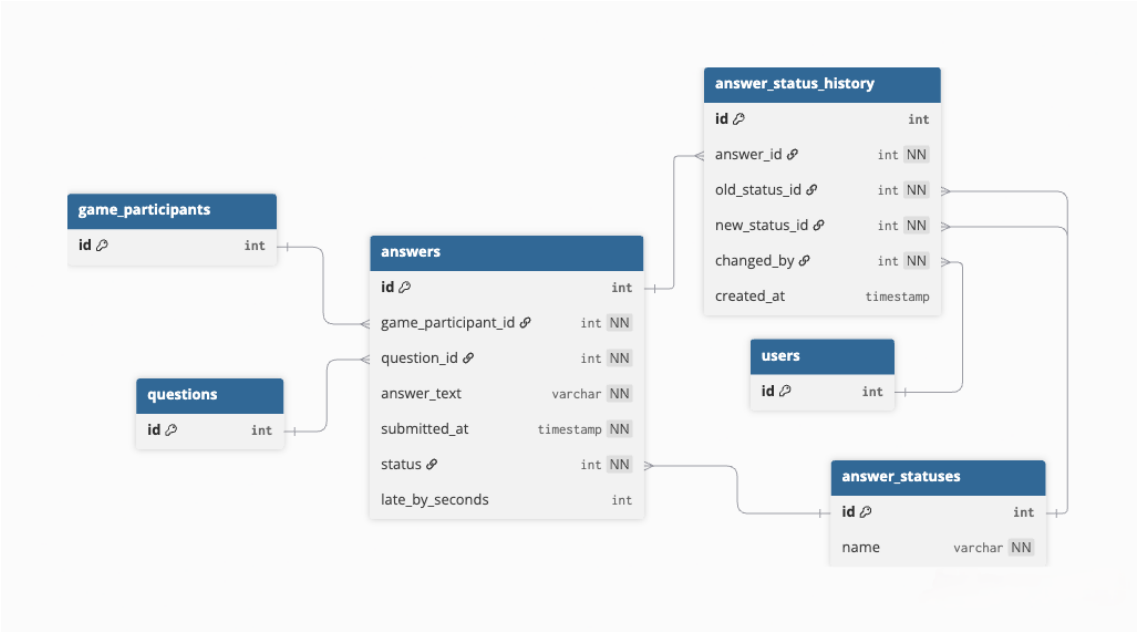
Sündmus	Millal	Omadused
`Socket Connected`	Any namespace	`namespace`, `role`
`Socket Connect Error`	Connection error	`namespace`, `error_message`
`Socket Disconnected`	Socket lost / closed	`namespace`, `role`

Lisa 4 – Andmemudeli olemite-seoste diagramm

Kuna täismahus diagramm sisaldab üle viieteistkümne tabeli ning selle põhitekstis esitamine raskendaks loetavust, on käesoleva lisa raames esitatud üldistatud, kolmeks teemaks rühmitatud kuva: Joonis 16 mängijad ja võistkonnad, Joonis 17 vastused ja hindamine ning Joonis 18 mängu seadistus. Apellatsioonide ja audit-tabelid on andmemudelis olemas, kuid nende täismahus realiseerimine jäi prototüübi skoobist välja, seetõttu jäeti need üldistatud diagrammilt välja.



Joonis 16. Andmemudeli olemite-seoste diagramm: mängijad ja võistkonnad.



Joonis 17. Andmemudeli olemite-seoste diagramm: vastused ja hindamine.

Joonis 18. Andmemudeli olemite-seoste diagramm: mängu seadistus.

Lisa 5 – WebSocket sündmuste ja REST otspunktide loend

Tabel 25. WebSocket sündmused.

Sündmus	**Suund**	**Otstarve**	**Olekumõju**
sync_history	klient → server	võistkonna ajaloo päri-mine	loeb ajaloo; vastus his-tory_update
sync_leaderboard	klient → server	edetabeli pärimine ühe-le kliendile	loeb edetabeli; vastus le-aderboard_update
admin:stop_question	admin → server (JWT)	aktiivse küsimuse pea-tamine	stopQuestion — mängu faas/küsimus
admin:next_question	admin → server (JWT)	järgmise küsimuse tsükli käivitamine	järgmine küsimus; timer_-update ruumi
admin:sync	admin → server (JWT)	admini sünk: tuba + täielik olek	liitub game_* ja game_*_-admins; sync_state klien-dile
admin:start_game	admin → server (JWT)	mängu LIVE-ks viimi-ne	startGame; game_status_-changed ruumi
join_game	mängija → server	võistkonnaga ruumi lii-tumine	osaleja + tuba; sync_sta-te mängijale ja adminitele; vea korral error
admin:prepare_ques-tion	admin → server (JWT)	küsimuse ettevalmistus	prepareQuestion; timer_-update ruumi
admin:start_question	admin → server (JWT)	küsimuse tsükli käivi-tamine	startQuestionCycle
player:submit_answer	mängija → server	vastuse salvestamine	processAnswer; kinnitus answer_received; admini-le admin:answer_update
admin:judge_answer	admin → server (JWT)	vastuse õige/vale otsus	judgeAnswer; adminile uuendus; võistkonnale his-tory_update; võib käivita-da edetabeli uuenduse

Sündmus	Suund	Otstarve	Olekumõju
player:dispute	mängija → server	vaidlustus	raiseDispute; adminile admin:answer_update + admin:new_dispute; ruumile leaderboard_update
admin:pause_timer	admin → server (JWT)	taimeri paus	pauseTimer; ruumile timer_paused
admin:resume_timer	admin → server (JWT)	taimeri jätkamine	resumeTimer; ruumile timer_resumed
admin:adjust_time	admin → server (JWT)	sekundite lisamine/eemaldamine	adjustTime
admin:finish_game	admin → server (JWT)	mängu lõpetamine	finishGame; game_status_changed ruumi
history_update	server → mängija (isiklik / ruum)	ajaloo uuendus pärast kohtuotsust	ainult klientide olek (UI)
leaderboard_update	server → klient(id)	edetabel	UI; summutatud uuendused ka gateway afterInit
answer_received	server → mängija	vastuse vastuvõtmise kinnitus	UI
admin:answer_update	server → adminite tuba	vastuse objekti tõuke	UI adminil
admin:new_dispute	server → adminite tuba	uue vaidlustuse teade	UI
sync_state	server → klient(id)	faas, taimer, aktiivne küsimus, osalejad	UI; osaliselt pärast disconnect
game_status_changed	server → ruum	LIVE / FINISHED jms	UI
timer_update	server → ruum	sekundid, faas, aktiivne küsimus	UI
timer_paused	server → ruum	taimer peatatud	UI
timer_resumed	server → ruum	taimer jälle käimas	UI
error	server → klient	veateade (käitleja või WsExceptionsFilter)	ei muuda mängu loogikat ise; kasutajale selgitus

Tabel 26. HTTP-otspunktid.

Meetod + tee	Juurdepääs	Otstarve	Olekumõju
GET /health	avalik	elutsükli / LB health	ei

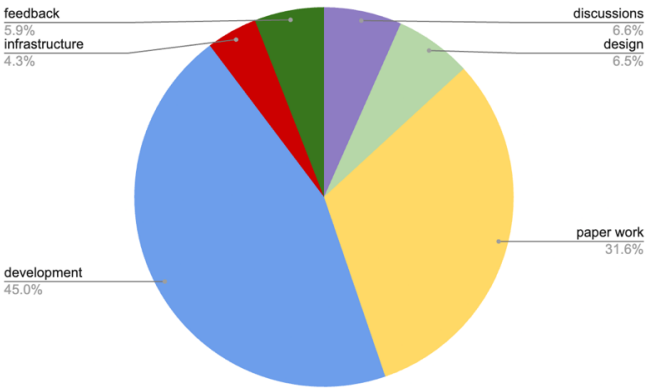
Meetod + tee	**Juurdepääs**	**Otstarve**	**Olekumõju**
GET /metrics	avalik	Prometheuse mõõdikute kraapimine	loeb mõõdikuid
POST /host/login	avalik	hosti JWT	sessioon/token
POST /host/register	avalik	uus host	kasutaja kirje
POST /host/passdrop	avalik	parooli taastus / “passdrop”	auth teenus
POST /host/games	JWT + roll HOST/ADMIN	mängude nimekiri	loeb
POST /host/games/create	JWT + roll	uue mängu loomine	uus mäng
POST /host/game/get	JWT + roll	ühe mängu detailid	loeb
POST /host/game/save	JWT + roll	mängu struktuuri salvestamine	versioon, küsimused, võistkonnad jne
POST /host/game/exportgame	JWT + roll	XLSX eksport	faili genereerimine
POST /player/check-game	avalik	pääsukoodi kontroll, võistkondade saadavus	loeb
GET /player/game/:gameId/leaderboard	avalik	edetabel HTTP kaudu	loeb
GET /player/feedback-form	avalik	tagasiside vormi skeem	staatiline konfig
POST /player/feedback	avalik	tagasiside saatmine	salvestab tagasiside

Lisa 6 – Meeskonnaliikmete ajakulu

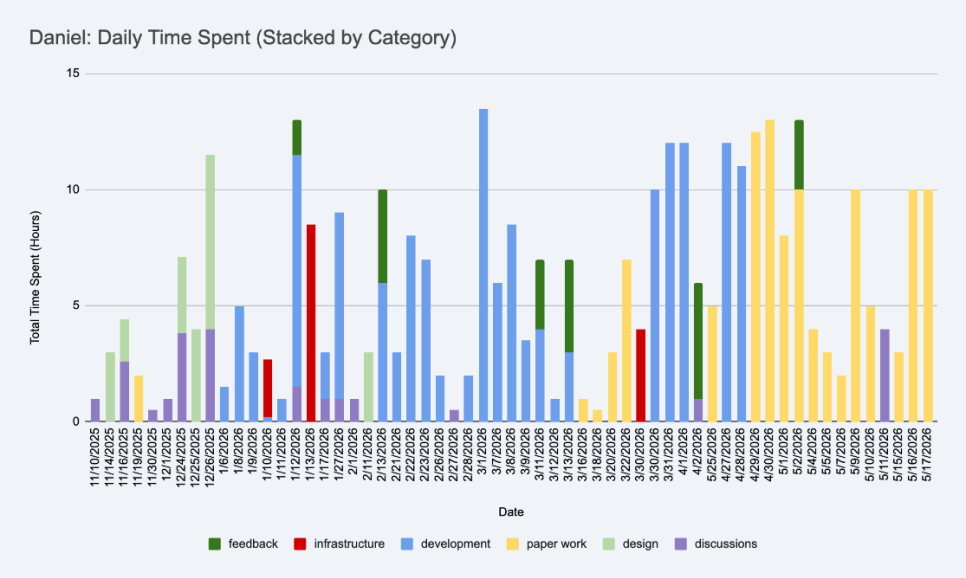
Esitatud andmed on aktuaalsed seisuga 17.05.2026.

Danieli üldine tööaja maht projekti jooksul oli **345.2 tundi**. Tööaja jaotus tundide ja protsentide lõikes on järgmine (vt Joonis 19 ja Joonis 20):

- Tarkvaraarendus (ingl *development*) — **155.2 tundi** (45.0%)
- Dokumentatsioon (ingl *paper work*) — **109.0 tundi** (31.6%)
- Arutelud ja koosolekud (ingl *discussions*) — **22.9 tundi** (6.6%)
- Disain ja planeerimine (ingl *design*) — **22.6 tundi** (6.5%)
- Tagasiside kogumine (ingl *feedback*) — **20.5 tundi** (5.9%)
- Infrastruktuur (ingl *infrastructure*) — **15.0 tundi** (4.3%)



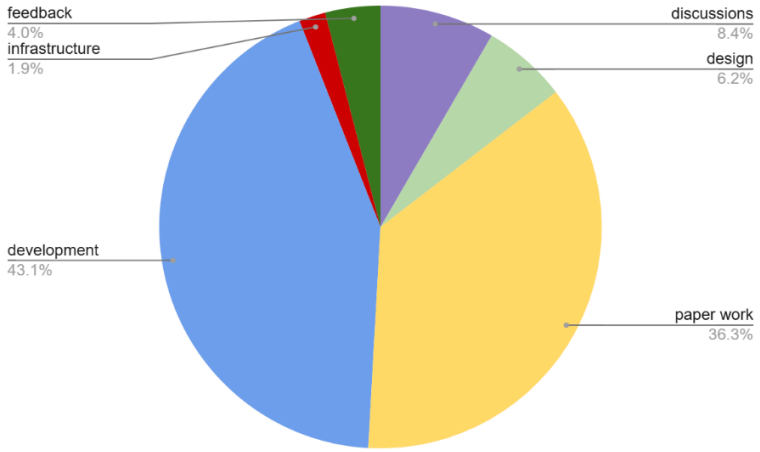
Joonis 19. Daniel: tööaja protsentuaalne jaotus.



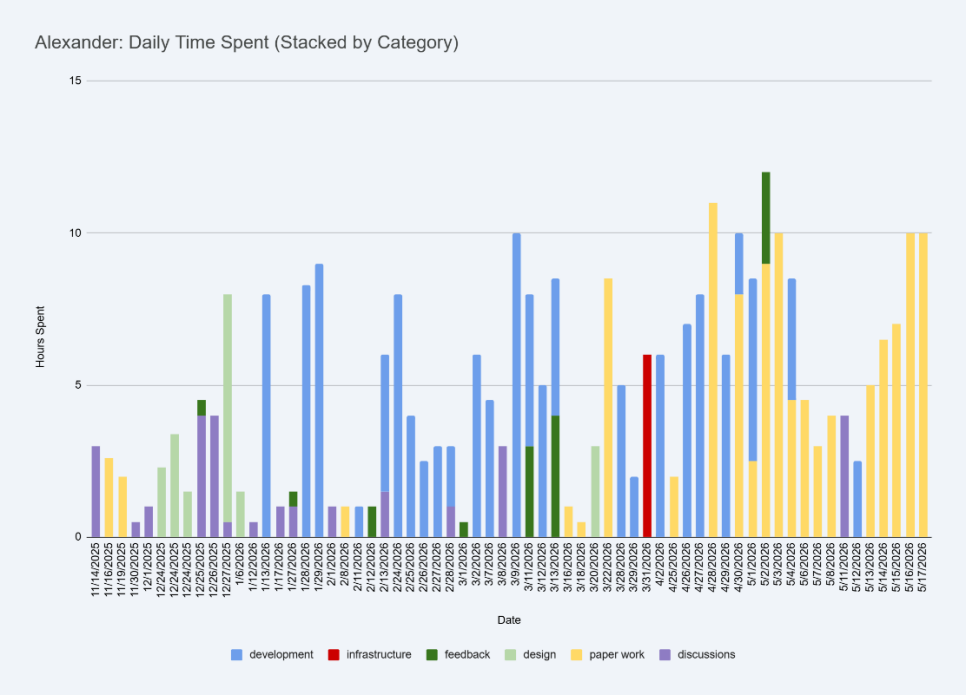
Joonis 20. Daniel: igapäevane ajakulu.

Alexandri üldine tööaja maht projekti jooksul oli **310.1 tundi**. Tööaja jaotus tundide ja protsentide lõikes on järgmine (vt Joonis 21 ja Joonis 22):

- Tarkvaraarendus (ingl *development*) — **133.8 tundi** (43.1%)
- Dokumentatsioon (ingl *paper work*) — **112.6 tundi** (36.3%)
- Arutelud ja koosolekud (ingl *discussions*) — **26.0 tundi** (8.4%)
- Disain ja planeerimine (ingl *design*) — **19.2 tundi** (6.2%)
- Tagasiside kogumine (ingl *feedback*) — **12.5 tundi** (4.0%)
- Infrastruktuur (ingl *infrastructure*) — **6.0 tundi** (1.9%)



Joonis 21. Alexander: tööaja protsentuaalne jaotus.



Joonis 22. Alexander: igapäevane ajakulu.