TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kätlin Rajamäe 213318IADB

Meeldetuletuste genereerimine konfiguratsiooni lõppkuupäeva lähenemise puhul

Bakalaureusetöö

Juhendaja: Kristiina Hakk
PhD

Tallinn 2025

Autorideklaratsioon

Kinnitan, et olen 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.

Autor: Kätlin Rajamäe

06.01.2025

Annotatsioon

Käesolev bakalaureusetöö keskendub Playtechis arendatava *Datalink*-süsteemi laiendamisele läbi lisafunktsionaalsuse, pakkudes lahendust, mis parandab kasutajakogemust ja lihtsustab konfiguratsioonide haldamist. Töö peamiseks eesmärgiks on välja töötada konfiguratsioonide meeldetuletuste süsteem, mis teavitab kasutajaid e-kirja teel lähenevatest konfiguratsioonide lõppkuupäevadest. Lahendus hõlmab uut töötüüpi "*EmailReminders*", mis genereerib automaatselt teavitused vastavalt kasutaja määratud seadistustele.

Töö käigus kaardistatakse probleem, analüüsitakse nõudeid, määratletakse süsteemi arhitektuur ning viiakse läbi arendus- ja testimisprotsess. Arendustöö tulemusena lisatakse *Datalink*-süsteemi kasutajaliidesesse konfigureeritavad väljad, mis võimaldavad määrata meeldetuletuste ajastust ning valida funktsionaalsuse kasutamist. Arenduse käigus rakendatakse mitmeid funktsionaalseid ja mittefunktsionaalseid nõudeid, sealhulgas lahenduse töökindlus ja kasutusmugavus.

Töö tulemuseks on süsteem, mis vastab seatud nõuetele ning lihtsustab konfiguratsioonide haldamist, seeläbi vähendades aega, mis kulub aegunud konfiguratsioonide taastamisele tagantjärele. Uus funktsionaalsus suurendab süsteemi usaldusväärsust ning pakub kasu nii Playtechi töötajatele kui ka nimetatud ettevõtte klientidele.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 37 leheküljel, 7 peatükki, 11 joonist, 4 tabelit.

Abstract

Generating Reminders for Approaching Configuration End Dates

Playtech, an international leader in gambling software development, relies on its Datalink system for data transformation and regulatory reporting. This bachelor thesis focuses on enhancing the Datalink system by implementing a new notification feature to improve the management of expiring configurations. The new functionality provides timely alerts to users of the system, providing proactive action to prevent possible disruptions and ensure system reliability.

The feature development begins with identifying user requirements through interviews and analyzing the system's limitations. The solution includes an update that allows users to enable receiving email reminders and set custom notification periods for configuration expirations. A backend task, EmailReminders, runs daily, if not configured differently, to identify expiring configurations and sends HTML email alerts containing needed details and a link to configuration.

Extensive testing in both local and integrated environments confirms the feature's reliability, usability, and performance. Current results show that the functionality effectively streamlines configuration management, reduces downtime risks, and saves time for internal teams. While the feature is currently limited to internal users, its broader implementation is expected to enhance client satisfaction and strengthen the Datalink system's and therefore Playtech's overall value.

This thesis demonstrates how targeted improvements can optimize workflows and lay the foundation for future advancements, such as integration with additional notification systems or analytics tools.

The thesis is in Estonian and contains 37 pages of text, 7 chapters, 11 figures, 4 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakenduste programmeerimisliides, mis võimaldab erinevatel tarkvararakendustel omavahel suhelda
CSS	*Cascading Style Sheets*, programmeerimiskeel kujundamiseks ja struktureerimiseks
Datalink	Bakalaureusetöös arendatav süsteem andmete transformeerimiseks
DI	*Dependency Injection*, Java Spring teegi funktsionaalsus – sõltuvuste süstimine
Garbage collector	Java funktsionaalsus teostada automaatset mäluhaldust
HEML	*Human-Email Markup Language,* avatud lähtekoodiga märgistuskeel e-kirjade loomiseks
HTML	*HyperText Markup Language,* veebilehtede loomiseks vajalik märgendkeel
HTTP	*HyperText Transfer Protocol*, suhtlusprotokoll veebibrauserite ja serverite vahel andmete vahetamiseks
IDE	*Integrated Development environment,* tarkvararakendus, mis pakub arendajatele terviklikku tööriistakomplekti programmeerimiseks
IEEE	*Institute of Electrical and Electronics Engineers*
IMS	*Information Management System*, ettevõtte Playtech osakond, kuhu kuulub bakalaureusetöö autor
IoC	*Inversion of Control,* Java Spring teegi funktsionaalsus – kontrolli ümberpööramine
JVM	*Java Virtual Machine*, virtuaalne masin, mis käivitab Java baitkoode.
MJML	*Mailjet Markup Language*, avatud lähtekoodiga märgistuskeel e-kirjade loomiseks
OOP	*Object Oriented Programming, o*bjektorienteeritud programmeerimine
Spring Boot	Tööriist Springi projektide kiiremaks arendamiseks, pakkudes automaatset konfiguratsiooni ja sisseehitatud serverit

Spring Core	Springi raamistiku südamik, mis pakub IoC ja DI tuge komponentide ja sõltuvuste haldamiseks
Spring Data	Moodul, mis lihtsustab andmebaasiga suhtlemist
Spring MVC	Moodul veebirakenduse loomiseks, toetades mudel-vaade-kontroller (MVC) arhitektuuri
Spring Security	Moodul autentimise ja autoriseerimise haldamiseks, pakkudes tuge turvalisuse integreerimiseks rakendustesse
SQL	*SQL Structured Query Language,* keel andmebaasiga suhtlemiseks
SRS	*Software Requirements Specification*, dokument, mis määrab süsteemi eeldatava käitumise
URL	*Uniform Resource Locator,* veebilehe aadress
UTC	*Coordinated Universal Time*, nullmeridiaanile vastav vööndiaeg
WORA	*„Write Once, Run Anywhere“*, Java tunnuslause. Üks kord kompileeritud koodi saab käivitada igas seadmes, milles on olemas JVM.
XML	*Extensible Markup Language*, märgistuskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Playtech, rahvusvaheline tarkvaraarenduse ettevõte, on juba üle kahe aastakümne olnud juhtiv jõud hasartmängutarkvara valdkonnas. Asutatud 1999. aastal Eestis, on ettevõte arendanud välja laia valiku tooteid, alates veebikasiinodest kuni spordiennustuste ja bingosüsteemideni. Selle edu aluseks on mitmekülgsed arendusmeeskonnad, kes loovad ja täiustavad süsteeme, mis vastavad rahvusvahelistele regulatsioonidele ja klientide ootustele. [1]

Käesolev bakalaureusetöö keskendub Playtechis arendatava Datalink-süsteemi täiustamisele. Antud süsteem, mis vastutab andmete transformeerimise ja edastamise eest regulatsioonipõhistele domeenidele, on ettevõtte klientide ja töötajate jaoks kriitilise tähtsusega.

Bakalaureusetöö eesmärk on luua funktsionaalsus, mis lihtsustab konfiguratsioonide haldamist ja parandab kasutajakogemust. Meeldetuletuste funktsiooni arendamine, mis teavitab kasutajaid lähenevatest tähtaegadest, aitab tõhusalt ennetada probleemide teket ja suurendab süsteemi usaldusväärsust. See lahendus on oluline mitte ainult klientide rahulolu parandamiseks, vaid ka ettevõtte sisemiste tööprotsesside sujuvamaks muutmiseks. Lõputöö läbiviimisel järgitakse metoodilist lähenemist, alates nõuete kaardistamisest kuni süsteemi testimise ja kasutajate tagasiside kogumiseni.

Käesolev bakalaureusetöö koosneb seitsmest peatükist. Teises peatükis tutvustatakse organisatsiooni, seletatakse lahti probleemi taust ja olemasolev *Datalink*-süsteem, töö eesmärk ning töö metoodika eesmärgi saavutamiseks. Kolmandas peatükis analüüsitakse loodavat lahendust, püstitatakse funktsionaalsed ja mittefunktsionaalsed nõuded, võrreldakse võimalikke tehnoloogiaid arenduseks ning selgitatakse meeldetuletuse e-kirja disaini. Töö neljandas peatükis kirjeldatakse lahenduse loomise protsessi ning eelviimases peatükis kirjeldatakse valminud tööd ja selle protsessi. Viimane peatükk võtab kokku saavutatud tulemused, funktsionaalsuse testimise ning võimalused edasiarenduseks.

2 Probleemi taust ja rakenduse eesmärk

Käesolevas peatükis tuuakse välja probleemi taust, aktuaalsus ja lähtetingimused. Selle põhjal püstitatakse töö eesmärk ja seejärel kirjeldatakse eesmärgi saavutamiseks kasutatavat metoodikat.

2.1 Organisatsiooni tutvustus

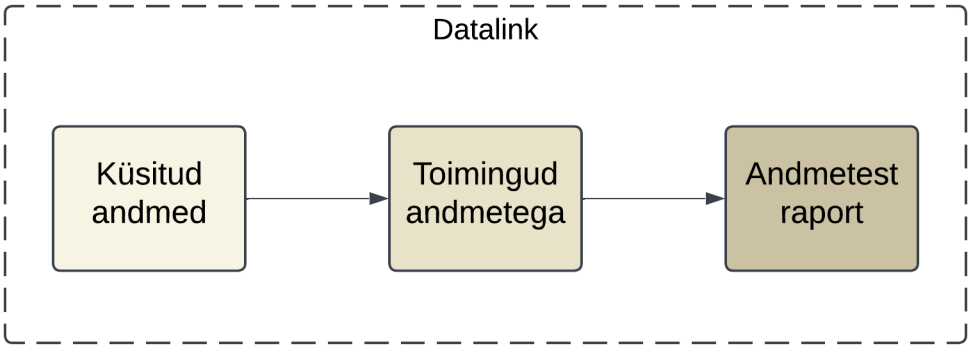
Playtech on rahvusvaheline tarkvaraarenduse ettevõte, mis on spetsialiseerunud hasartmängutarkvarale, sealhulgas veebikasiinodele, spordiennustustele, pokkerile, bingole ja muudele mänguteenustele. Ettevõte asutati aastal 1999 Eestis. Playtechil on kontoreid üle kogu maailma, kus töötab kokku üle 7000 töötaja, kellest üle 700 töötavad Eestis. [1]

Eestis on Playtech esindatud oma suurima arenduskeskusega, mis avati 2001. aastal Tartus ja hiljem laiendati Tallinnasse. Playtech on jagatud tiimideks, kelle vastutusala hõlmab erinevaid valdkondi arendusprotsessis. Bakalaureusetöö autor on antud firmaga seotud, töötades seal tarkvaraarendajana üheksaliikmelises tiimis. Antud bakalaureusetöö on juurdearendus tiimi IMS (*Information Management System)* hallatava toote *Datalink* kõikide regulatsioonide jaoks.

2.2 Süsteemi arhitektuur

Datalink on süsteem, mis küsib ühest kohast andmeid, transformeerib need ning selle tulemus edastatakse raportite näol. Vastav lihtsustatud protsess on kujutatud Joonis 1. Lihtsustatud *Datalink* käitumine Raportite genereerimiseks luuakse sellele konfiguratsioon, kus saab täpsustada just konkreetsele vajadusele sobivaid parameetreid, mille hulka kuuluvad näiteks koht kust andmeid loetakse, mis formaadis raport genereeritakse, mis ajatsoonis jne. Konfiguratsiooni loomisega luuakse uus töö, mis omakorda genereerib raporteid. Raportid genereeritakse vastavalt täpsustatud parameetritele, raporti tüübile ning valitud regulatsioonile. Samuti on ajaperiood, mille vältel andmeid transformeeritakse ja raporteid luuakse, märgitud konfiguratsioonis.

Regulatsioonid on enamasti riigi või osariigipõhised, näiteks Portugal ja Poola, kuid nad võivad olla ka midagi muud, näiteks *ReportViewer* või *General*. Iga konfiguratsioon loob regulatsiooni- ehk domeenipõhise uue töö, mis transformeerib andmeid vajaminevalt. Regulatsioon ja domeen on selle lõputöö raames mõistetavad samamoodi. Erinevatel domeenidel võivad olla erinevad töö tüübid, millele saab konfiguratsiooni luua. Töö tüüpideks on näiteks *GlobalCancellation*, mis tühistab kõik konfiguratsioonis sätestatud vahemikus olevad andmed, või *GlobalReplacement*, mis vastupidiselt tühistamisele hoopis asendab antud vahemikus olevad andmed.



Joonis 1. Lihtsustatud *Datalink* käitumine.

Domeenispetsiifiliste töötüüpidele lisaks on siiani vaid üks töö, mis on olemas igas regulatsioonis. Käesolevas bakalaureusetöös lisatakse samasugune töötüüp, mida saab käivitada igas domeenis. Domeenipõhised lahendused on jagatud eraldi domeenipõhistesse moodulitesse, kuid üldine lahendus läheb *datalink_worker_common* moodulisse, mis teeb ta kättesaadavaks kõikidele regulatsioonitüüpidele.

2.3 Taust

Nii testijad, *DevOps* (*Development Operations*) insenerid, *Datalink*-i kliendid ning teised, kellel on antud programmis palju konfiguratsioone loodud, seisavad silmitsi probleemiga, kuidas neil järge pidada. Kasutajakogemust parendaks võimalus saada meeldetuletusi peatselt lõppeva konfiguratsiooni kohta, et seejärel uute parameetritega uus koostada. Kasutajatele, kes on autoriks paljudele töödele, oleks abiks loodav arendus, mis teavitab töö autorit e-kirjaga, mille sisus on välja toodud konfiguratsiooni nimi, id, konfiguratsiooni looja ning algus- ja lõppkuupäev, lisaks sisaldab e-kiri ka linki

konfiguratsiooni muutmisele *Datalink* veebilehe kasutajaliidesele, kus saab mugavalt uue luua või olemasolevat muuta.

Taoline lisafunktsionaalsus vähendaks arusaamatust, miks enam raporteid ei looda ning ennetab probleeme ja nende tagajärgedega tegelemist.

2.4 Eesmärk

Antud töö eesmärk on lihtsustada *Datalink* projekti kasutamist nii Playtechi klientide kui ka ettevõtte töötajate jaoks. Eesmärk on pakkuda kasutajatele võimalust valida meeldetuletuste saamine e-kirja teel ning samuti saab valida päevade arvu, kui palju enne konfiguratsiooni aegumist meeldetuletus saadetakse. Antud arendus võiks tekitada klientidele parema kogemuse programmi kasutades ning seeläbi võiksid nad jääda lojaalsemateks kasutajateks süsteemile ja ettevõttele. Samuti vähendaks see testijate ajakulu, mõistmaks miks järsku enam mingeid raporteid ei saadeta kui konfiguratsioon vahepeal aegunud on. Vähem aega kuluks probleemi tagajärgede väljauurimisele, kuna võimalik on probleeme ennetada.

2.5 Metoodika

Esiteks kirjeldatakse lahti probleem ja probleemi taust, misjärel saab püstitada konkreetsed eesmärgid. Kui eesmärgid paigas on, tuleb välja selgitada nõuded, millele arendus vastama peab. Kokkuleppele tuleb jõuda e-kirja disaini, konfigureeritavate parameetrite ning süsteemi toimimise osas. Nõuete seadmisel võetakse arvesse *Datalink*-i seatavaid piiranguid kui ka oma kogemust. Koos nõuete ja eeldatava disainiga tuleb koostada tarkvara nõuete kirjelduse dokument, kus on peale nimetatud elementide ka joonis protsessi toimumisest. Süsteemi mõistmiseks on hea luua skeem – kuidas toimub kasutaja ja programmi vaheline suhtlus. Nõuete põhjal saab analüüsida võimalikke variante eesmärgi saavutamiseks. Analüüsist peab selguma kõige optimaalsem viis, kuidas lahendus *Datalink* süsteemis luua ja mis tehnoloogiaid selleks kasutada.

Süsteemi valmimisel testitakse arendust lokaalselt, kasutades lokaalset kasutajaliidest, lokaalset andmebaasi ning võlts meiliserverit. Kui arendus lokaalselt toimib, lisatakse funktsionaalsus üldisesse projekti, et testida seda ka sealses keskkonnas. Süsteemi

valmimisel hinnatakse tulemusi ja kontrollitakse, kas määratud nõuded said täidetud. Kui süsteem vastab nõuetele, saab planeerida edaspidiseid arendusi.

3 Probleemi analüüs

Käesolevas peatükis sõnastatakse nii funktsionaalsed kui ka mittefunktsionaalsed nõuded arendusele, samuti tehakse valik nii taga- kui ka eesrakenduse tehnoloogiatele ning kirjeldatakse saadetava meeldetuletuse e-kirja disaini.

3.1 Nõuded arendusele

Nõuded on aluseks nii tarkvaraarendusele kui ka projektijuhtimisele, seetõttu peaksid mõlemad osapooled enne arenduse alustamist läbi mõtlema rakenduse nõuded [2]. See aitab kiirendada arendusprotsessi tehes kindlaks, kuidas süsteem käituma peab. Nõuete definitsioone on mitmeid ning ei saa eeldada, et kõik osapooled neid alati samamoodi mõistavad, selleks on oluline nõuded eelnevalt defineerida. Siin töös mõistetakse nõudeid kui täpsustust sellest, mida tuleks arendada. Nad on kirjeldused sellest, kuidas süsteem või selle osa käituma peaks, nad võivad olla ka piiranguks süsteemi arenduse protsessile. [3]

Enne arendusprotsessi alustamist luuakse dokument, mis kogub kokku arenduse nõuded. Dokumenti kutsutakse kui *software requirements specification*, edaspidi SRS. Nimetatud dokumendis on kirjeldatud arendatava süsteemi eeldatav käitumine. SRS on tähtis dokument arendusprotsessi jaoks ning seda kasutavad erinevad grupid ja erinevatel eesmärkidel – nimelt klient, et teaks mida oodata, tarkvara arendajad, kelle jaoks on oluline dokumendist lugeda mida ja kuidas luua ning ka testijad, kes selle alusel teavad mida ja kuidas testida [4]. IEEE (Institute of Electrical and Electronics Engineers) standard 830 *Recommended Practice for Software Requirements Specification* on standard, kus on antud nõuete spetsifikatsiooni struktuuri ülevaade [5]. Tuginedes sellele standardile, peaks SRS koosnema kolmest põhisektsioonist: sissejuhatus, ülevaade ning nõuete loend.

Nõuete määramisel on arvesse võetud olemasoleva projekti seatavaid piiranguid, funktsionaalust kasutama hakkavate kasutajate eelistusi ning autori kogemust samalaadsete arendustega. Kasutajate eelistuste määramiseks on valitud ettevõttesisesed töötajad, kes kasutavad aktiivselt olemasolevat programmi. Eelistused kaardistatakse mittestruktureeritud intervjuu käigus. Programm arendatakse lisana *Datalink* projektile, mistõttu määrab osa nõudeid olemasolev süsteem. Kirja pannakse funktsionaalsed ja

mittefunktsionaalsed nõuded, millele süsteem toetuma peaks. Mõlemad on esitatud Tabel 1.

Tabel 1. Funktsionaalsed ja mittefunktsionaalsed nõuded.

Funktsionaalsed nõuded	**Mittefunktsionaalsed nõuded**
F1. Konfiguratsiooni loomise kasutajaliidesele tuleb lisada muudetav väli nimega *enable_notifications*	MF1. Kättesaadavus - Lahendus peab olema kättesaadav erinevates keskkondades üheaegselt
F2. Peab olema töö, mis kontrollib olemasolevaid konfiguratsioone ja koostab meeldetuletusi	MF2. Hooldatavus - keskmine süsteemi taastamise aeg pärast süsteemi riket ei tohib olla rohkem kui 30 minutit päevas
F3. Kuna erinevatel domeenidel on erinevad andmebaasid, peab töö neid kontrollima eraldi ja koostama domeenispetsiifilisi meeldetuletusi	MF3. Jõudlus - töö peab suutma töödelda ühe päevast andmemahtu 5-minutilise reageerimisajaga
F4. Kui meeldetuletuse väli on valitud konfiguratsioonile, peab töö kontrollima konfiguratsiooni lõppkuupäeva	MF4. Kasutatavus - Väljund peab olema arusaadav ja kergesti mõistetav
F4.1. Vaikimisi saadetakse meeldetuletus 30 päeva enne lõppkuupäeva	MF5. Sõltumatus - Lahendus ei tohi olla operatsioonisüsteemi poolt mõjutatav
F4.1.1. Päevade arv peab olema konfigureeritav	
F4.2. Lõppkuupäeva puududes meeldetuletusi ei koostata	
F5. Loodav töö peab käivituma kord päevas	
F6. Meeldetuletus saadetakse vaikimisi konfiguratsiooni loojale	
F7. Meeldetuletuse sisu tuleb lisada otse kõigile adressaatidele saadetavasse meili	

3.1.1 Funktsionaalsed nõuded

Funktsionaalsed nõuded on kõige olulisemad arendajale, kes ideed realiseerima hakkab. Funktsionaalseid nõudeid saab pidada funktsioonideks, mida programmeerimismeeskond tarkvara arendamisel realiseerib, nad määravad kindlaks, mida tarkvara tegema peab. [6]

Funktsionaalne nõue F1 nõuab, et uue konfiguratsiooni loomisel oleks kasutajaliidesel uus väli nimega *enable notifications*, selle tõeseks määrates saadetakse kasutajale meeldetuletused peagi aeguva konfiguratsiooni asjus e-kirjadena. F2 järgi eeldatakse, et

lahendus toimib eraldiseisva tööna, mis kontrollib olemasolevaid konfiguratsioone ning koostab vajadusel meeldetuletused. Nõudes F3 on välja toodud, et kuna domeenid kasutavad erinevaid andmebaase, peavad meeldetuletused olema domeenispetsiifilised, see tähendab, et igas domeenis peab olema meeldetuletuste jaoks eraldiseisev töö. F4, F4.1, F4.1.1 ja F4.2 lähevad omavahel kokku, nimelt kui F1-s välja toodud *enable notifications* väli on valitud, kontrollib töö konfiguratsiooni lõppkuupäeva ning koostatakse meeldetuletus. Meeldetuletus saadetakse vaikimisi 30 päeva enne lõppkuupäeva, kuid nõue F4.1.1 nõuab, et päevade arv peab olema ka konfigureeritav. Päevade arvu saab samuti sätestada konfiguratsiooni luues kasutajaliideselt ning väli ilmub nähtavale vaid siis, kui *enable notifications* väli on määratud tõeseks. Kui konfiguratsioonil ei ole määratud lõppkuupäeva, ei koostata meeldetuletust. Loodud töö meeldetuletuste kontrollimiseks ja saatmiseks peab nõude F5 puhul käivituma kord päevas. Funktsionaalne nõue F6 nõuab, et meeldetuletus saadetakse alati konfiguratsiooni looja e-maili aadressile.

3.1.2 Mittefunktsionaalsed nõuded

Mittefunktsionaalsed nõuded viitavad omadustele, mis esindavad tarkvaralahenduse kvaliteedioskusi. Mittefunktsionaalsete nõuete üleskirjutamine ei ole kohustuslik, kuna arendamine on võimalik ka ainuüksi funktsionaalsete nõuete alusel. Siiski mittefunktsionaalsete nõuete olemasolul saab üheselt mõista mudeli töökindluse ootusi ning tagada, et mudel toimiks parimal võimalikul viisil, arvesse võtmata jõudlusparameetrite ootusi. [6]

Mittefunktsionaalne nõude MF1 paneb paika süsteemi kättesaadavuse eelduse – kuna programmi kasutavad kliendid mitmes keskkonnas, on tähtis, et see oleks kättesaadav üheaegselt erinevates keskkondades. MF2 sätestab, et kui peaks toimuma süsteemirike, ei tohi taastamine võtta rohkem aega kui 30 minutit, et takistada minimaalselt süsteemi kasutamist. MF3 määrab süsteemi jõudluse eesmärgi – kuni ühe päeva jagu andmete töötlemine peab võtma aega vähem kui 5 minutit, et vältida kasutajate ootamist süsteemi järel ning et mitte blokeerida liialt teisi protsesse, mis samaaegselt jooksevad. Nõue MF4 nõuab kliendi kasutatavuse väljundi kerget mõistetavust, antud funktsionaalsus kaotaks mõtte kui väljundit on keeruline mõista. Nõue MF5 on põhjus, miks töö just niiviisi eraldi töö tüübina planeeritud on, ehk lahendus ei tohi olla operatsioonisüsteemi poolt

mõjutatav, see välistab töö teostamise andmebaasi skriptidena ning nõuab et töö peab olema süsteemi osa.

3.2 Rakenduses kasutatava tehnoloogia valik

Süsteemi loomiseks tuleb valida töövahendid, mis sobivad konkreetse süsteemi jaoks ning sobiks ka olemasolevasse *Datalink* projekti.

3.2.1 Tagarakenduse tehnoloogia valik

Tagarakendus koosneb tihti programmeerimiskeelest ning raamistikust või teegist, mis täiustavad teineteist ning mõlemal on nii positiivsed kui ka negatiivsed küljed. Järgnevalt on välja toodud kahe programmeerimiskeele võrdlus, mille käigus analüüsitakse keelte sobivust antud funktsionaalsuse jaoks. Keelte loetelu valik on tehtud vastavalt nende populaarsusele ja kasutusele aastal 2024, kus kõige populaarsemateks kujunesid Python ja Java [7].

Java

Java on populaarne ja üldotstarbeline objektorienteeritud (OOP) programmeerimiskeel, mida kasutatakse laialdaselt töölauarakenduste, veebisüsteemide, mobiilirakenduste ja isegi suurtöötlussüsteemide loomiseks. Tänud OOP põhimõtetele, pakub programmeerimiskeel omadusi nagu koodi taaskasutus, modulaarne disain ning hea turvalisus. Põhiliseks Java tunnuslauseks on „*Write Once, Run Anywhere*“ (WORA), mis tähendab, et üks kord kompileeritud Java rakendust saab käivitada igas seadmes, kui seal on olemas JVM (*Java Virtual Machine*). [8]

Java puudusteks on kiirus – see on küll mitmetest keeltest kiirem, kuid kuna Java töötab JVM-i kaudu, lisab see abstraktsioonikihi ning teeb ta aeglasemaks kui näiteks C ja C++. Lisaks kasutab Java võrdlemisi palju mälu tänu JVM-le ja automaatsele prügikoristusele ehk *garbage collector*-le, mis võib tekitada probleeme ressursipiirangutega keskkondades. Java kood võib olla võrreldes programmeerimiskeelega Python kompleksne ning keeruline lugeda, kuna see kasutab pikemaid ja keerulisemaid lauseid. [8], [9]

Java boonuseks on juba mainitud platvormisõltumatus – Java võimaldab ühe rakenduse kasutamist mitmel operatsioonisüsteemil kasutades JVM-i. Tänu *pointer*'te puudumisele

on Java keel ka turvalisem – pole võimalik kasutada massiivi mittekuuluvaid indekseid, see tekitab *ArrayIndexOutOfBound* erindi. Lisaks, operatsioonisüsteemist sõltumatus tagab samuti turvalisuse, hoides ära potentsiaalselt kahjuliku koodi otsest juurdepääsu süsteemiressurssidele. Java on robustne keel – see on arendatud kontrollima vigu nii vara kui võimalik, keele teevad robustseks prügikoristus (*garbage collection*), erandite käsitlemine ning mälu jaotamine. Java oluliseks positiivseks küljeks on selle suur hulk erinevaid raamistikke, millest populaarsemateks on Spring, Hibernate ja Akka. [8] [9]

Python

Python osutus aastal 2024 kõige populaarsemaks programmeerimiskeeleks [9]. See on üldotstarbeline, kõrgetasemeline ja dünaamilise tüübiga programmeerimiskeel, mida iseloomustavad lihtne süntaks ja loetavus. Selle töötas välja Guido van Rossum 1991. aastal ning keel on laialdaselt kasutusel nii algajate kui ka kogenud arendajate seas. Python sobib mitmesugusteks rakendusteks, sealhulgas veebiarenduseks, andmeteaduseks, tehisintellekti loomiseks, skriptimiseks ja automatiseerimiseks. Python on interpreteeritav programmeerimiskeel, mis tähendab, et selle kompileerimise asemel masinkoodi, käivitatakse see hoopis interpretaatori poolt. See tähendab, et lähtekood loetakse ja käivitatakse ridahaaval interpretaatori abil. [10], [11]

Keele positiivne pool on kergesti loetav süntaks, mis muudab selle õppimise ja kasutamise lihtsaks isegi algajatele, kuna keerulisi ülesandeid on võimalik lahendada väheste koodiridadega. Pythoni plussiks on veel selle platvormiülesus – selle kood töötab erinevates operatsioonisüsteemides ilma muudatusteta, Python järgib sarnasel Java-le WORA meetodit. Pythoni kasutamine on populaarne isiklikes, töövälistes projektides, kuna see on tasuta ja avatud lähtekoodiga. [10], [11]

Pythoni negatiivseks küljeks on selle suurem mälutarbimine võrreldes mõne madalama taseme programmeerimiskeelega. Tänu interpretaatori kasutusele on keel aeglasem võrreldes Java-ga. Kuna see on dünaamiliselt kirjutatud keel, ei pea ühtegi muutujat deklareerima, mistõttu täisarvu salvestav muutuja võib hiljem hoopis stringi salvestada. See võib hiljem programmi käivitamisel vigu või ootamatut käitumist põhjustada. Lisaks, vead tulevad välja alles programmi käivitades, mis teeb selle testimise keerukaks. Suures projektis võib keele piiranguks olla mitme lõime toe puudumine ning samuti ka selle aeglus. [10], [11]

Kokkuvõte

Järgnevalt on võrreldud Java ja Pythoni omadusi Tabel 2. Programmeerimiskeelte Java ja Python võrdlus „„

Tabel 2. Programmeerimiskeelte Java ja Python võrdlus [8], [9], [10], [11].

Omadus	**Python**	**Java**
Keel	Interpreteeritav, dünaamiliselt tüübitud keel	Kompileeritav, staatiliselt tüübitud keel
Kiirus	Aeglasem, tõlgitud olemuse tõttu	Kiirem kui Python
Süntaks	Lihtne ja loetav, sobib algajatele	Detailsem, nõuab rohkem koodiridu
Platvormiülesus	Platvormiülene, kui seadmesse on paigaldatud Python	Platvormiülene, kui seadmesse on paigaldatud JVM
Vead ja nende tuvastus	Vead ilmnevad programmi käivitades, võib põhjustada ootamatusi	Staatiline tüübi kontroll kompileerimise abil aitab vigu varakult tuvastada
Turvalisus	Vähem turvaelemente, kuna keel on dünaamiline	Turvalisem tänu staatilisele tüübitusele
Mitme lõime tugi	Piiratud, *Global Interpreter Lock* piirab mitme lõime paralleelset töötamist ühes protsessis	Hea mitme lõime tugi, sobib hästi paralleeltöötluseks ja suurtöötlussüsteemideks
Mälukasutus	Suurem mälutarbimine tänu interpretaatorile ja automaatsele prügikoristusele	Mälukasutus on paremini optimeeritud kui Pythonis, kuid on siiski märkimisväärne JVM-i ja automaatse prügikoristuse tõttu

Tabel 2. Programmeerimiskeelte Java ja Python võrdlus „„ on võrreldud Java ja Python programmeerimiskeelt, mille kokkuvõtteks võib öelda, et mõlemad on mitmekülgsed ja populaarsed programmeerimiskeeled, millel on selged tugevused ja ka piirangud. Python paistab silma oma lihtsa süntaksi ja kasutajasõbralikkuse poolest, mis teeb selle sobivaks kiireks prototüüpimiseks, andmeteaduseks, tehisintellekti arendamiseks ja automatiseerimiseks. Java on aga tugev ettevõtte tasemel arendustes, kus on vaja keerukate süsteemide töökindlust, turvalisust ja skaleeritavust. Nimetatud põhjustel sobib antud probleemi lahendamise jaoks paremini Java programmeerimiskeel.

Lisaks on programmeerimiskeele valikul määravaks ka olemasolevas programmis kasutatud programmeerimiskeel. Kuna eelneva analüüsi põhjal selgus, et Java sobib paremini käesoleva projekti teostamiseks ning Datalink põhineb Java programmeerimiskeelel, otsustatakse lisafunktsionaalsuse arendamisel kasutada Javat.

Järgnevalt võrreldakse kahte raamistikku, mis Java-t komplementeeriksid antud probleemi lahendamisel.

Spring

Java-Spring on raamistik, mida kasutatakse Java põhiste rakenduste arendusel. Selle lihtsus, kiirus ja produktiivsus on selle teinud maailmas üheks populaarseimaks raamistikuks. Spring toetab modulaarset arhitektuuri, võimaldades arendajatel kasutada ainult vajalikke komponente, nagu *Spring Core*, *Spring MVC*, *Spring Boot*, *Spring Data* ja *Spring Security*. Springi eesmärk on lihtsustada rakenduste arendamist, võimaldades arendajatel keskenduda oma rakendusele, samas kui Spring haldab taustprotsesse. Antud raamistik kasutab lõimepõhist samaaegsusmudelit, kus lõime kasutatakse taotluste käsitlemiseks ja ülesannete täitmiseks. Lisaks, selle asünkroonne suhtlus tugineb traditsioonilistele lähenemisviisidele, nagu lõimestamine või Java CompletableFuture. [12], [13]

Springi puuduseks on selle komplekssus, eelneva kokkupuute puudumisel on raamistiku selgeks tegemine pikem protsess, just uute programmeerimismeetodite tõttu nagu IoC (*Inversion of Control*) ja DI (*Dependency Injection*). Springil on palju sarnaseid mehhanisme nii, et arendaja peab nende vahel valima. Arendajale võib see segadust tekitada ning vale otsuse puhul võib see tekitada ettevõtte tasemel olulisi probleeme, nagu näiteks programm jääb aeglaseks. [12], [13]

Küll aga on Springil ka palju eeliseid, mis teeb sellest maailmas populaarseima raamistiku. Spring on kiire, nii projekti käivitades kui ka isegi uut Springi projekti luues – kasutades *Spring Initializr* tehnoloogiat, võib see toimuda sekunditega. Springil on suur kogukond ulatusliku dokumentatsiooni, õpetuste ja kolmandate osapoolte teekidega. Seda kasutatakse tööstuses laialdaselt ja sellel on elav ökosüsteem. Töö muudavad lihtsamaks IoC ja DI tehnoloogiad, mis pakuvad suure valiku funktsionaalsuseid. IoC ehk kontrolli ümberpööramine, kus programmi voogu ei juhi mitte komponent ise, vaid väline raamistik või konteiner, see võimaldab objektidel saada oma sõltuvusi välisest süsteemist,

selle asemel et neid ise luua, lihtsustades seeläbi koodi hooldatavust ja testitavust. DI ehk sõltuvuste süstimine võimaldab klasse iseseisvamaks muuta, et neid tulevikus taaskasutada saaks. Springi positiivseks küljeks on ka selle modulaarne korraldatus – pakette ja klasse on väga palju, kuid kasutada saab vaid neid, mida vaja ning ülejäänuid saab ignoreerida. [12], [13]

Akka

Akka on tööriistakomplekt samaaegsete, hajutatud ja tõrketaluvate rakenduste loomiseks. Akka sobib eriti hästi süsteemidele, mis nõuavad kõrget jõudlust ja usaldusväärsust isegi osaliste rikete korral. Akka põhineb näitlejamudelil, mis võimaldab jagada süsteemi väiksemateks, paremini juhitavateks osadeks (näitlejateks), mis suudavad üksteisega asünkroonselt suhelda. Näitlejad on sõltumatud, ühe lõimega protsessid, mis saavad sõnumeid vastu võtta ja neile sõnumitele vastuseks toiminguid teha. [13], [14]

Sarnaselt Java-Spring teegiga, on ka Akka ja selle näitlejamudel suhteliselt kompleksne ja uuele kasutajale eelneva paralleelprogrammeerimise kogemuse puudumisel keeruline hoomata. Suuremate süsteemide seadistamine võib olla aeganõudev ja nõuab põhjalikku arusaama. Miinuseks on veel selle piiratud Java tugi – kuigi Akka on saadaval nii Java kui ka Scala jaoks, on see algelt loodud siiski Scala jaoks, mis võib põhjustada keerukusi puhtalt Java-põhistes projektides. Lisaks, kui Akka projektid ei ole korralikult seadistatud ja hooldatud, võib see tekitada projektis tõrkeid ja probleeme. [13], [14]

Küll aga on Akka projektidel hea skaleeritavus. Akka disain võimaldab kergesti skaleerida rakendusi, rakenduse töömahu suurenedes ei ole vaja keerulisi ja kalleid koordineerimismehhanisme. Tänu näitlejatele võimaldab Akka arendada rakendusi, mis kasutavad sünkroniseeritud koodi, vältides tihedaid sidemeid, muutes koodi kergesti testitavaks ja hooldatavaks. Akkal on hea rikete haldus – see võimaldab süsteemil rikke puhul iseseisvalt taastuda ning tööd jätkata. Positiivseks on selle puhul veel võimalus seda integreerida muude populaarsete teekidega, nagu Akka HTTP (HyperText Transfer Protocol). Akka soodustab komponentide vahelist asünkroonset suhtlust, mis võimaldab mitteblokeerivaid ja samaaegseid süsteeme. See pakub asünkroonsete toimingute käsitlemiseks võimsaid abstraktsioone, nagu *Future* ja *stream*. [13], [14]

Kokkuvõte

Nii Spring kui ka Akka komplementeeriks Javat antud probleemi lahendamisel. Tabelis 3 on välja toodud kokkuvõte nende tööriistade tehnoloogiate positiivsetest ja negatiivsetest külgedest.

Tabel 3. Springi ja Akka võrdlus Javaga integreerimisel. [12], [13], [14]

Omadus	**Spring**	**Akka**
Arhitektuur	Lõimepõhine arhitektuur; kasutatakse lõimesid ja *CompletableFuture* klassi asünkroonseks suhtluseks.	Näitlejamudel (*Actor Model*); komponendid on väiksemad sõltumatud näitlejad, mis suhtlevad asünkroonselt sõnumite kaudu.
Kompleksus	Keeruline algajale, eriti IoC ja DI kasutuse tõttu, kuid dokumentatsioon ja kogukond on suured ja toetavad.	Kompleksne uutele kasutajatele, eriti Java kontekstis, kuna on algselt loodud Scala jaoks; suure süsteemi seadistamine võib olla keeruline.
Jõudlus ja skaleeritavus	Hea jõudlus, kuid raskem hallata väga suurt samaaegsust (kui võrrelda näitlejamudeliga).	Väga hästi skaleeritav; sobib süsteemidele, mis nõuavad samaaegsust ja jõudlust.
Veataluvus ja stabiilsus	Tugineb traditsioonilistele vigade käsitlemise meetoditele (*try-catch*, erindite haldus).	Sisseehitatud vigade haldus; näitlejad võimaldavad rikete puhul süsteemil taastuda ilma kogu protsessi peatamata.
Modulaarsus	Modulaarne, võimaldab kasutada ainult vajalikke komponente, näiteks *Spring Boot, MVC,* või *Security.*	Näitlejapõhine arhitektuur soodustab komponentide autonoomsust ja vähendab tihedaid sidemeid.
Integreerimine	Tugev ökosüsteem; integreerimine paljude tööriistadega nagu *Spring Data, Security* ja *Boot*.	Hea integreerimine teiste Akka teekidega, näiteks Akka HTTP; piiratud Java-põhistes projektides.
Sihtrakendused	Sobib hästi veebirakendustele, äriloogikale ja mikroteenustele.	Sobib hajutatud süsteemidele, reaalajas töötlemisele ja kõrget jõudlust nõudvatele rakendustele.
Arenduskiirus	*Spring Initializr* abil saab projektid kiiresti käima panna; kogukonna ja tööriistade toel on arendus produktiivne.	Keerukam seadistamine võib aeglustada arenduse algfaasi, eriti ilma Scala kogemuseta.
Kogukond ja dokumentatsioon	Suur ja küps kogukond; ulatuslik dokumentatsioon ja õpetused.	Väiksem kogukond kui Springil, kuid Scala ökosüsteemis hästi toetatud.

Võrreldes mõlemaid tööriistu, skaleerib Springi samaaegsusmudel vähem tõhusalt ja sellel võib olla rohkem üldkulusid kui Akkas kasutatava näitlejamudeliga. Lisaks, Springi lähenemine asünkroonse suhtluse puhul ei pruugi olla nii tõhus kui on Akka näitlejapõhine mudel. Samas on Springil väga heal tasemel dokumentatsioon ning kolmandate osapoolte teekide tugi koos aktiivse kogukonnaga, Akkal vastupidiselt on kogukond väiksem ning Java spetsiifiline dokumentatsioon puudulikum.

Kokkuvõttes erinevad Akka ja Spring oma samaaegsusmudelite, jaotus- ja tõrketaluvuse võimaluste, asünkroonse suhtluse, reaktiivse programmeerimise toe, teiste tehnoloogiatega integratsiooni taseme ning nende koosluste ja ökosüsteemide suuruse poolest. Olenevalt rakenduse nõuetest saavad arendajad valida Akka näitlejamudeli samaaegsete ja hajutatud süsteemide jaoks või Springi tervikliku raamistiku vahel ettevõtete rakenduste loomiseks.

Kuna käesoleva probleemi lahendamisel on oluline programmi hea jõudlus, samaaegsus ja hea skaleeritavus, sobiks Java kõrvale kõige paremini Akka tööriistakomplekt. Lisaks Akka oluline eelis on selle sisseehitatud vigade haldus – näitlejad võimaldavad rikete puhul süsteemil taastuda ilma kogu protsessi peatamata, mis on tähtis mahuka rakenduse puhul hea kasutajakogemuse pakkumiseks.

3.2.2 Rakenduse kasutajaliidese valik

Käesoleva töö arenduse üks osa on tagarakenduse arendus – kus toimub loogika ning suurem osa tööst, kuid teine osa on e-kirja disaini loomine. Saadetavas e-kirjas peab olema kindel info, mis pärineb kindla töö konfiguratsioonist. Info on igal konfiguratsioonil erinev, seega e-kirja põhi peaks olema dünaamiliselt muudetav. Eelneval põhjusel ning kuna kiri koosneb ka piltidest, oleks hea saata kiri välja HTML (*HyperText Markup Language*) alusel, mille genereerimiseks on erinevaid variante. Saab kas otse HTML malli luua või selle genereerimiseks kasutada mingit raamistikku. Emaili raamistikud on kollektsioonid ettevalmistatud HTML ja CSS (*Cascading Style Sheets*) komponentidest, mida emaili arendajad kirjade loomiseks kasutavad [15]. Järgnevalt tuuakse välja positiivsed ja negatiivsed küljed e-kirja malli koostamiseks, kirjutades otse HTML-is või kasutada populaarseid raamistike selle genereerimiseks.

HTML

HTML emailil on spetsiaalne vorming, pildid, lingid ja muud aspektid, mida tavalistes e-kirjades ei sisaldu. Kuna arendaja ei tea, millises brauseris saadetud kiri avatakse, on sellisel kirjal hea omadus enda suurust kohandada vastavalt ekraani suurusele. [16]

Positiivseid omadusi, mille alusel oleks hea kirjutada otse HTML keeles mall, on mitu. Arendaja käes on kogu kontroll - võimalik on üksikasjalik kohandamine iga malli aspekti jaoks. Otse kirjutades puudub sõltuvus raamistikest, lisaks välistab see vajaduse kompileerimise või muude täiendavate tööriistade järele. Teeke kasutamata võimaldab mall olla paindlik – võimalik on kasutada kohandatud skripte, kui disain seda nõuab. [16]

Miinuseks on selle hooldamisele kuluv aeg – muudatused ja uuendused võivad muutuda tüütuks, eriti korduvate elementide puhul. Mallis olevad käsitsi lisatud stiilid ning võimalikud ühilduvuse muudatused võivad põhjustada vigu, seega tõrgete oht on suurem. Otse HTML-is malli kirjutada võib olla aeganõudev, kuna see vajab täiendavat pingutust, et tagada ühilduvus alati uuenevate meiliklientide vahel. Sellisel viisil puuduvad raamistike abstraktsioonid, mis lihtsustavad reageerivat disaini ja CSS-i sisestust. [16]

HEML

HEML (*Human-Email Markup Language*) lihtsustab e-maili mallide jaoks HTML-i loomise protsessi, kasutades semantilist märgistust, mis kompileerub emailiga ühilduvaks HTML-ks. Dünaamilise HTML-i loomiseks kasutab HEML kohatäitjat, mis tagarakenduses väärtusega asendatakse, näiteks *{{name}}*. Iga HEML element on ekvivalentne eelnevalt määratud HTML elemendiga. Renderdamise käigus leiab mootor kõik HEML-i elemendid ning teisendab need HTML-ks. [17], [18]

HEML kasutab semantilist märgistust – selle intuitiivne süntaks hoiab ära vigu ja parandab koodi loetavust, muutes mallid kergelt mõistetavaks ja hooldatavaks. Keelel on automaatne kompileerimine, mis tähendab, et see tegeleb CSS integreerimisega suurematele meiliklientidele ja reageeriva paigutuse kohandamisega. Positiivseks on veel keele kasutajasõbralikkus, uuel e-kirja arendajal on seda lihtne hoomata. Lisaks, see on väga sarnane HTML-iga, kasutades osati samu nimesid siltidel, näiteks <*body*> ja <*button*>. [17], [18]

Kuigi raamistikul on hea meilikliendiga ühilduvus ja reageeriv disain, võib selle väiksem kogukond ja keele piiratud funktsioonid osutuda piiranguks väga kohandatud ja keerukate

e-kirjade kujunduste jaoks. Võrreldes teiste raamistikega pakub HEML vähem pistikprogramme ja tööriistu. [17], [18]

MJML

MJML (Mailjet Markup Language) on HTML-il põhinev avatud lähtekoodiga keelespetsiifikatsioon ja tööriistakomplekt, mis on loodud e-posti mallide lihtsa ja optimaalse loomise ja haldamise jaoks. Antud keel kasutab semantilist süntaksit, mis muudab selle kasutamise lihtsaks ja arusaadavaks ning selle avar standardkomponentide valik vähendab arendusaega ning emaili koodihulk jääb samuti väiksemaks. MJML-il on avatud lähtekood ning see genereerib dünaamilise HTML aluse, mis on kooskõlas HTML tavadega. MJML dokumentatsiooni kohaselt hoitakse keelt kursis emailikliendi värskendustega, mis tähendab, et arendaja ei pea seda ise tegema. Nii püsib e-kirja põhi alati uute spetsifikatsioonide ja nõuetega kooskõlas. MJML võimaldab kirjutada lihtsat ja abstraktset koodi, mis tõlgitakse automaatselt erinevate e-posti klientide jaoks optimeeritud HTML-i. Selle eesmärk on lahendada probleem, et e-posti klientide tugi HTML-i standarditele on sageli ebajärjepidev. [19], [20]

MJML keele kasuks räägivad paljud asjaolud. Tänu selle vaikimisi reaktiivsusele, kuvatakse e-kirjad kõikides seadmetes probleemideta, ilma täiendavate seadistusteta. Raamistikul on ulatuslik komponentide valik, mille hulgas on nii karussellid, nupud kui ka menüüd. Lisaks põhineb see komponendipõhisel struktuuril, kiireks arendamiseks kasutab see kohandatavaid silte, nagu <*mj-section*> ja <*mj-text*>. Keelel on tugev kogukonna tugi ja regulaarsed värskendused tagavad pideva ühilduvuse alati uuenevate emaili klientide standarditega. Sarnaselt nagu HEML, kasutab MJML samuti semantilist märgistust. [19], [20]

Miinuseks MJML keele puhul, on selle sõltuvus kompileerimisest – kasutamiseks valmis HTML-i loomiseks on vaja eraldi sammu selle genereerimiseks, lisasamm võib mõne töövoo keerukamaks muuta. Väga spetsiifilise disaini puhul võib jääda puudu disainielementidest. Väga edasijõudnud disain võib nõuda kompileeritud väljundi manuaalset muutmist. [19], [20]

Kokkuvõte

Tabel 4. HTML, MJML ja HEML võrdlus e.kirja malli loomiseks. [16], [17], [18], [19], [20]

Omadus	**HTML**	**MJML**	**HEML**
Eesmärk	Standardne veebilehtede märgistuskeel, mida kasutatakse ka emailide loomiseks.	Spetsiaalne e-posti mallide keel, mis kompileeritakse standardseks HTML-ks.	Avatud lähtekoodiga e-posti mallide koostamise raamistik, keskendub lihtsusele ja mugavusele.
Süntaksi keerukus	Vajab keerukat käsitsi kirjutamist ja kohandamist; iga e-posti kliendi ühilduvuse kontroll ise.	Lihtsustatud süntaks; abstraheerib keerulise HTML-i ja teeb selle hõlpsamini loetavaks.	Lihtne ja intuitiivne süntaks, mis keskendub kasutaja tootlikkusele ja aja kokkuhoiule.
Ühilduvus emailiklientidega	Vajab spetsiifilist koodi ja stiile, et toetada kõiki emailikliente nagu Outlook ja Gmail.	MJML konverteerib koodi automaatselt ühilduvaks kõigi emailiklientidega.	Samuti fokusseeritud ühilduvusele, kuid pakub vähem detaile ja pistikprogramme kui MJML.
Arenduse kiirus	Võtab aega, eriti algajale, kuna nõuab käsitsi stiilide ja tabelite kohandamist.	Kiirem arendus tänu deklaratiivsele ja lihtsale süntaksile; automaatne kompileerimine HTML-ks.	Väga kiire ja sobiv arendajale, kes soovib lühikest õppimiskõverat ja tõhusust.
Kohandamine ja paindlikkus	Väga paindlik, kuna arendaja saab täielikult kontrollida iga koodi elementi.	Piiratud võrreldes puhta HTML-iga, kuid pakub piisavat paindlikkust tavaliste emailimallide jaoks.	Piiratud võimalused keerukamate ja väga spetsiifiliste kohanduste jaoks.
Stiilide haldus	Vajalik kasutada *inline*-stiile ja tabeleid, mis on töömahukas.	Automaatne *inline*-stiilide haldus.	Automaatne *inline*-stiilide ja koodistruktuuri optimeerimine.
Dokumentatsioon ja kogukond	Lai dokumentatsioon ja suur kogukond veebiarenduse kontekstis.	Tugev kogukond ja dokumentatsioon, suunatud just e-posti mallide arendusele.	Väiksem kogukond ja dokumentatsioon kui MJML-il, kuid arenev ökosüsteem.
Sobivus	Kogenud arendajale„ kes vajab täielikku kontrolli ja paindlikkust.	Neile, kes vajavad kiiret ja lihtsat lahendust ilma HTML-i detailidesse süvenemata.	Neile, kes otsivad lihtsat ja modernset lahendust emailimallide jaoks.

Tabel 4. HTML, MJML ja HEML võrdlus e.kirja malli analüüsitakse HTML, MJML ja HEML tehnoloogiaid e-kirja malli koostamise jaoks.

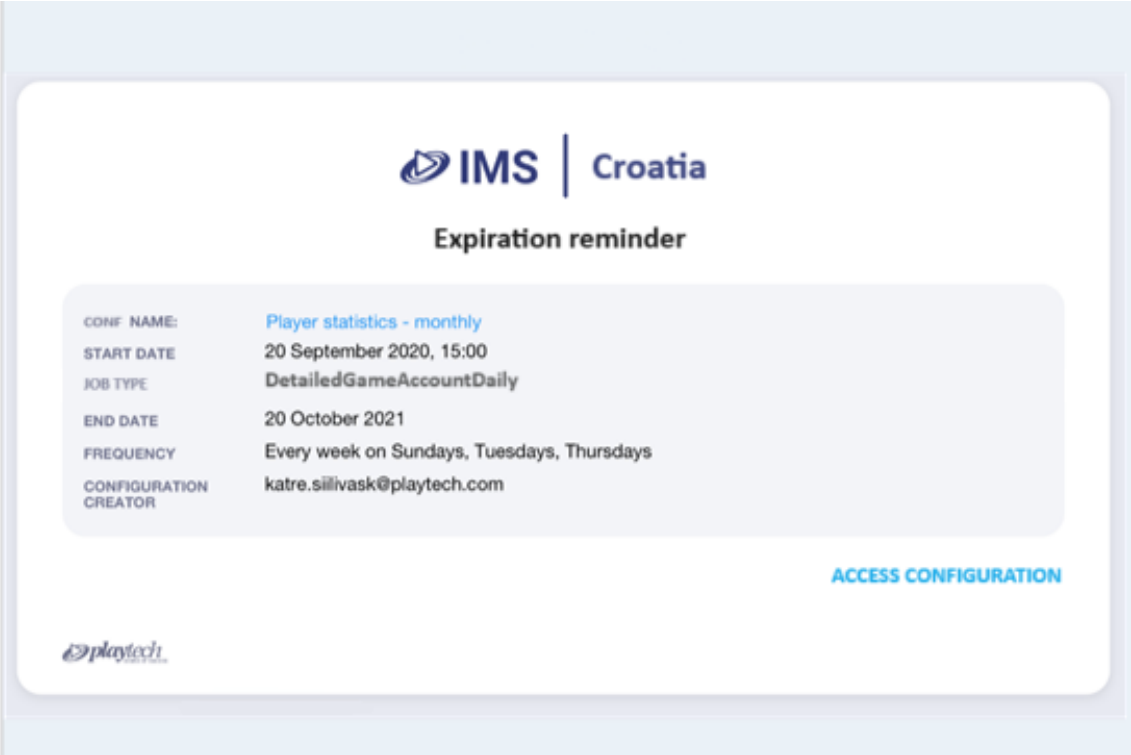
HTML, MJML ja HEML pakuvad erinevaid lähenemisi e-posti mallide koostamiseks. HTML on universaalne, kuid keerukas ja aeganõudev, nõudes palju käsitsi tööd ja detailset tähelepanu emailiklientide ühilduvusele. Võttes arvesse, et arendaja ei tea millist emaili brauserit klient e-kirjade avamiseks kasutab, peab ta kindlaks tegema, et koostatud disain näeks korrektne igal platvormil välja, selle eest aitavad vastutada raamistikud, mis ongi loodud seda probleemi lahendama. HEML pakub algajatele lihtsust ja kiiret arendusprotsessi, kuid selle piiratud paindlikkus võib osutuda keerukamate projektide puhul takistuseks. MJML pak lihtsustatud süntaksit ja kiiret arendust, samas võimaldades automaatset ühilduvust kõikide emailiklientidega. [16], [17], [18], [19], [20]

MJML-i valimine oleks parim valik, kuna see tasakaalustab kasutusmugavuse, paindlikkuse ja emailiklientide toetuse, muutes selle heaks tehnoloogiaks käesoleva probleemi lahendamisel.

3.3 E-kirja disain

Peatükis 3.1 koostatud SRS dokumenti lisati peale sissejuhatuse, nõuete ja ülevaate ka loodava e-kirja prototüüp, kuhu on määratud vajalikud komponendid ning info, mida meeldetuletuse e-kiri sisaldama peaks. Kuna e-kirja eesmärk on edastada infot, peavad andmed olema esitatud selgelt ja arusaadavalt. Infot ei tohiks olla liiga palju ega ka liiga vähe.

On loodud rakenduses Figma töö autori poolt. See on tasuta tööriist, mida kasutatakse laialdaselt kasutajaliideste, veebilehtede ja rakenduste disainimiseks ja prototüüpimiseks. Kasutamise teeb mugavamaks erinevate väliste komponentide ja teekide olemasolu, mis võimaldavad kasutada juba valmis komponente See aitab oluliselt aega säästa ning paremini toodet visualiseerida. E-kirja eeldatavat disaini näeb Joonis 2. E-kirja oodatav disain.



Joonis 2. E-kirja oodatav disain.

Kirja ülaosas vasakul on IMS logo, mis näitab, et antud teenust pakub *Information Management System* osakond Playtechis. Selle kõrval paremal on rasvases tekstis ning eelneva logoga samas toonis vastava domeeni nimi, mille konfiguratsioonile meeldetuletust saadetakse. Mõlema vahel on püstine musta värvi kriips, mille eesmärk on elemente eraldada. Järgnevalt on logo ja domeeni nime all e-kirja pealkiri mustas tekstis, mis on keskele joondatud, pealkirjaks on alati *Expiration reminder*, eesti keeles aegumise meeldetuletus.

E-kirja põhiosas on informatsioon aeguva konfiguratsiooni kohta. Esimeseks väljaks on konfiguratsiooni nimi helesinises tekstis. Järgnevalt tuleb selle algusaeg, mis on ühtlasi aeg, millal esimene raport genereeriti selle jaoks. Kolmas väli sisaldab infot töötüübi kohta. Järgmine element näitab mis kuupäeval konfiguratsioon lõpeb ning ühtlasi mis on viimase raporti lõppaeg. Edasi tuleb töö kordumissagedus, mis on toodud välja päevade nimedega konkreetsel juhul. Viimane väli on konfiguratsiooni looja emaili aadress, kuna mõne töö puhul võib tekkida küsimus, kes selle koostas. Kõik nimetatud väljad on kasti sees ning kast on täidetud helehalli värviga.

Kirja allosas paremal pool on helesinises toonis tekst, mida saab klikkida. Sellele peale vajutades suunab URL edasi *Datalink* süsteemi konfiguratsiooni info juurde, kus näeb

peale eelnevalt väljatoodud väljade ka kõike muud. Enne konfiguratsiooni avamist kontrollitakse kasutaja õigusi selle nägemiseks keskkonda sisselogimise teel. Kõige all vasakul on Playtechi logo.

4 Lahenduse loomine

Käesolevas peatükis kirjeldatakse arenduse käiku *Datalink* projektis. Arendus viidi läbi kahes osas, millest esimene keskendub tagarakenduse arendusele ning teine e-kirja malli loomisele. Arenduse käigus järgiti kaasaegseid tarkvaraarenduse praktikaid, keskenduses koodi modulaarsusele, efektiivsele ressursikasutusele ning koodi taaskasutatavusele.

4.1 Arendus *Datalink* projektis

Nõuete kohaselt ei tohi meeldetuletuste genereerimine olla operatsioonisüsteemi poolt mõjutatav ning lahendus peab olema mitmes keskkonnas saadaval samaaegselt. Nõuete põhjal selgus, et hea viis meeldetuletuste genereerimiseks on luua uus töötüüp, mille nimeks saab *EmailReminders*. See on parim viis, kuna selline lahendus on üldine, olemas igal regulatsioonil ning on osa projektist. Funktsionaalsus arendati *Datalink* projekti lisana, kasutades Java programmeerimiskeelt ning Akka raamistikku. Integreeritud arenduskeskkonnana (IDE) kasutati IntelliJ IDEA arendustarkvara. IntelliJ IDEA on populaarne integreeritud arenduskeskkond Java programmeerimiskeele jaoks. Antud IDE on tuntud oma funktsioonide poolest, nagu nutikas koodi lõpuleviimine, tõrkeotsing käigu pealt, refaktoreerimistööriistad ja versioonihaldussüsteemide, nagu Git, sisseehitatud tugi. Selle ulatuslik pistikprogrammide ökosüsteem suurendab veelgi selle võimalusi, võimaldades arendajatel kohandada keskkonda oma konkreetsetele vajadustele. [21]

Järgnevad alampeatükid toovad välja mõned olulisemad programmikoodi funktsioonid.

4.1.1 Uue töö loomine

Uue töö loomine peab toimuma domeenis ühel korral.

Töö käivitamiseks süsteemis on vajalik sellele luua deskriptor, mis töö välja kutsub. Deskriptor on kontsept, mida kasutatakse mitmetes raamistikes töö ajastamiseks ja käivitamiseks, näiteks raamistik Quartz, mis on avatud lähtekoodiga raamistik tööde ajastamiseks [22].

Kuigi deskriptori rakendamine ja kasutus võib erineda, sisaldab see tavaliselt töö metaandmeid ja juhiseid, nagu näiteks [23]:

- Töö ID: töö kordumatu tunnus.

- Ülesande üksikasjad: informatsioon konkreetse toimingu või protsessi kohta, mida töö tegema peaks.
- Prioriteet: näitaja töö tähtsuse jaoks või täitmise järjekord.
- Olekuteave: töö praegune olek, näiteks ootel, lõpetatud või nurjunud.
- Sõltuvused: viited teistele töödele, mis tulevad enne selle töö käivitamist lõpetada.

Datalink programmis kasutatakse töö deskriptoreid tööde määratlemiseks ja konfigureerimiseks Akka klastri jagamise raamistikus. Antud deskriptori atribuutideks on:

- Töö tüüp – täpsustab töö tüübi.
- Olemi tüübi võti – määrab töökäskude jaoks olemi tüübi võtme.
- Klastri jagamise sätted – klastri jagamise konfiguratsioonisätted.

Töö spetsiifiline loogika kirjutati klassi *EmailRemindersTask* ning Joonis 3. *EmailRemindersTask-i* käivitav meetod.Joonis 3. *EmailRemindersTask-i* käivitav meetod. on kujutatud meetod, mis käivitatakse esimesena ülesannet alustades.

Kuna projekt on suur ning kasutusel on mitu lõime korraga, on oluline logida, mis tööd hetkel toimuvad, selle tõttu algab joonisel 3 kujutatud meetod informatiivse logiga, kus on kirjas töö ID ning ülesande nimi. Järgnevalt luuakse olemasolevatest atribuutidest uus sõnastik, kuhu lisatakse juurde regulatsioonitüübi nimi, seda läheb vaja järgmises meetodis. Edasi luuakse ühendus andmebaasiga kasutades Java *Supplier* liidest. Nimetatud liides on saadaval *java.util.function* paketis ning seda kasutatakse laisa initsialiseerimise jaoks, kus väärtus arvutatakse alles siis, kui väärtust vaja läheb [24]. Andmebaasist küsitakse kõik sobivad konfiguratsioonid, millele on vaja meeldetuletus koostada ning lisatakse need nimekirja. Andmebaasist küsimine on kirjeldatud peatükis 4.1.2. Konfiguratsioonidest koostatud nimekirja hakatakse ükshaaval lugema ning igal lugemisel luuakse emaili klient, millele on külge lisatud loetud töö ja selle konfiguratsiooni ID-d, töö atribuudid, emaili serveri omadused ning lisaks ka emaili aadress, kuhu meeldetuletus saata on vaja. Viimase kahe välja väärtus on määratud vaadeldavas konfiguratsioonis, kust need ka loetakse. Kui e-kirja saatmisel visatakse hoopis erind, logib süsteem vastava teavituse koos töö ID-ga ning käivitab meetodi fail, mille puhul kasutajale meeldetuletust ei saadeta ning tööle jääb staatuseks *ReminderFailed*. Töö lõpetamist ja staatuseid on pikemalt kirjeldatud peatükis 4.1.4.

```
protected void executeTask(ExecuteDynamicTask cmd) throws Exception {
  log.info("{} executing email reminders task", cmd.getJobId());
  RegulationType regulationType = cmd.getRegulationType();
  Map<String, String> attrs = new HashMap<>(cmd.getAttrs());
  attrs.put("regulationType", regulationType.getName().toLowerCase());

  DbSessionSupplier sessionSupplier = dbSessionSupplier(regulationType);
  List<ScheduledJobConfStateEntity> jobConfStateEntityList;

  try (DbSession dbSession = sessionSupplier.get()) {
    ScheduledJobConfStateMapper scheduledJobConfStateMapper =
dbSession.getMapper(ScheduledJobConfStateMapper.class);
    jobConfStateEntityList =
scheduledJobConfStateMapper.getExpiringConfigurations();
  }

  for (ScheduledJobConfStateEntity scheduledJobConfStateEntity :
jobConfStateEntityList) {
    datalinkEmailClient(attrs, scheduledJobConfStateEntity)
            .whenComplete(new CompletionConsumer<>() {
              @Override
              public Consumer<DatalinkEmailClient> resultHandler() {
                return emailClient -> sendEmail(emailClient, attrs, cmd,
scheduledJobConfStateEntity);
              }

              @Override
              public Consumer<Throwable> errorHandler() {
                return throwable -> {
                  log.error("{} failed to send reminder", jobId, throwable);
                  fail(cmd.getReplyTo(), throwable);
                };
              }
          });
  }
}
```

Joonis 3. *EmailRemindersTask-i* käivitav meetod.

4.1.2 Andmebaasiga suhtlus

Datalinki süsteemis kasutatakse andmebaasiga suhtlemiseks Mapperit, mis võimaldab SQL-päringute (*Structured Query Language*) ja Java-objektide vahel lihtsat ja efektiivset ühendust. ScheduledJobConfStateMapper on selleks otstarbeks loodud liides, mis

määratleb SQL-päringud ja tulemuste kaardistamise meetodid, pakkudes tugevat andmesidekihti süsteemi erinevate osade vahel. Selle rakendamiseks kasutatakse MyBatis raamistikku, mis vähendab vajadust käsitsi kirjutada ulatuslikku andmebaasikoodi. [25]

Käesoleva arenduse jaoks on kasutusel päring, mis on nähtav järgneval Joonis 4. Päring leidmaks meeldetuletust vajavate konfiguratsioonide jaoks. Alloleva päringu väljakutsumine on nähtaval Joonis 34, mille alusel lisatakse järjekorda kõik meeldetuletust vajavad konfiguratsioonid.

```
@Select("SELECT * " +
        "FROM scheduled_job_conf_state " +
        "WHERE json_value(state, '$.job_conf.enable-notifications') = 'true'
" +
        "AND json_value(state, '$.scheduling_enabled') = 'true' " +
        "AND (CAST(json_value(state,
'$.schedule_settings.recurrence_range.end_time') AS DATETIME) - " +
        "INTERVAL CAST(json_value(state, '$.job_conf. days_before_reminder ')
AS INT) DAY) " +
        "> CAST(#{startTime} AS DATETIME) " +
        "AND (CAST(json_value(state,
'$.schedule_settings.recurrence_range.end_time') AS DATETIME) - " +
        "INTERVAL CAST(json_value(state, '$.job_conf.days_before_reminder) AS
INT) DAY) " +
        "<= CAST(#{endTime} AS DATETIME)")
@Results(value = {
        @Result(property = "persistenceId", column = "persistence_id"),
        @Result(property = "sequenceNr", column = "sequence_number"),
        @Result(property = "state", column = "state")
})
List<ScheduledJobConfStateEntity> getExpiringConfigurations
(@Param("startTime") String startTime, @Param("endTime") String endTime);
```

Joonis 4. Päring leidmaks meeldetuletust vajavate konfiguratsioonide jaoks.

Antud meetod *getExpiringConfigurations*() on loodud selleks, et pärida kõik konfiguratsioonid, mille puhul teavituste saatmine on lubatud ning mis vastavad kindlale ajavahemikule. Selle päringu kõige keerukam osa oli loogika välja mõtlemine ning selle kirja panemine SQL käsuna.

@Select annotatsioon: SQL-päring määrab ära, et andmebaasist valitakse tabelist *scheduled_job_conf_state* kõik read, kus:

- *state* väljalt tuvastatakse, et *job_conf.enable-notifications* väärtus on tõene,
- *scheduling_enabled* on lubatud (tõene).

- teatud ajaperiood jääb etteantud algus- ja lõpuaegade vahemikku (alguskuupäev ei ole kaasa arvatud, lõppkuupäev on).
    - Algus- ja lõpuajad on parameetritena kaasa antud, need tulevad *EmailReminders* konfiguratsiooni poolt järjepidevalt loodavatelt töödelt.
- JSON-andmete töötlemine: Kuna *state* veerg sisaldab keerukaid JSON-struktuure, kasutatakse päringus *json_value()* funktsioone, et saada kätte spetsiifilisi atribuute, nagu *days_before_reminder* ja *recurrence_range.end_time*.
- Ajavahemiku kontroll: Päring kasutab SQL-i *CAST* ja *INTERVAL* funktsioone, et teisendada kuupäevad ja määrata ajavahemikud dünaamiliselt.
- *@Results* annotatsioon: See määrab, kuidas päringu tulemused kaardistatakse Java objektiks *ScheduledJobConfStateEntity*.
- Parameetrid: Meetod võtab sisendina kaks parameetrit (startTime ja endTime), mis määravad ajavahemiku, mille põhjal filtreeritakse konfiguratsioone.

Selle Mapperi kasutamine võimaldab rakendusel suhelda andmebaasiga lihtsal ja struktureeritud viisil. Lisaks tagab Mapperi arhitektuur, et andmete sünkroniseerimine Java-koodi ja SQL-päringute vahel toimub sujuvalt ja vigadeta. Selline lähenemine vähendab manuaalse SQL-koodi kirjutamise vajadust, tõstab arendamise efektiivsust ning muudab andmebaasipäringute haldamise paindlikuks ja hooldatavaks.

4.1.3 Emaili klient

Suurem osa arendatava funktsionaalsuse loogikast toimub Emaili kliendi klassis nimega *DatalinkEmailClient*. Klient loeb kõigepealt projektis asuvast HTML failist e-kirja malli ning esimesena asendab kõik muutujad vajalike väärtustega. Selles sammus muudetakse järgnevad väärtused:

- {configuration-name} -> konfiguratsioonist küsitud nimi
- {ims-logo-cid} -> teekond IMS logo pildini projektis
- {regulation-display-name} -> töö atribuutidest loetakse regulatsioonitüübi nimi
- {start-date} -> konfiguratsioonist küsitud selle algusaeg koos ajavööndiga
- {end-date} -> konfiguratsioonist küsitud selle lõpuaeg koos ajavööndiga
- {job-type} -> konfiguratsioonist küsitud tüübi nimetus
- {frequency} -> konfiguratsioonist küsitud ajastusreeglid, mis on sõnastatud inimloetavaks tekstiks
- {configuration-creator} -> konfiguratsioonist küsitud selle looja

- {pt-logo-cid} -> teekond Playtechi logo pildini projektis
- {configuration-link} -> koostatud URL, mis viib konfiguratsiooni muutmisele kasutajaliideses

Koos vajaliku informatsiooniga on e-kirja mall loodud. Edasi luuakse e-kirja objekt, mille klass on kujutatud joonisel 5.

```
public static class Email {
	String fromEmail;
	String toEmail;
	String subject;
	String html;
}
```

Joonis 5. Emaili klass.

Saatja emailiaadress on alati sama ning saaja aadress leitakse eraldi päringu vastusest. HTML on eelmisest sammust saadud tulemus koos asendatud väärtustega.

Sõnumi saatmiseks kasutatakse Jakarta Mail paketti, mis on standardne Java API e-maili saatmiseks ja vastuvõtmiseks. See on eriti kasulik, kui on vaja integreerida e-maili saatmise funktsionaalsus Java-põhiste rakendustega. Paketti kasutatakse sõnumi loomisel, millele lisatakse saatjaks Joonis 5. Emaili klass kujutatud Email klassist loodud objektilt küsitud saatja emailiaadress. Saaja ning pealkiri küsitakse samuti eelmisest objektist ning lisatakse sõnumi külge. Hiljem lisatakse sellele ka muudetud HTML kood koos logode ja vajalike väärtustega.

4.1.4 Töö lõpetamine

Et oleks näha kuidas ning millal töö lõppeb, peab see näitama veebirakenduses mingit informatiivset staatust. Staatus ei tohi olla liigselt pikk ega ka liigselt lühike, see peab erineva olemasolevatest muude tööde staatustest, kuna loogika on erinev. Töö nime järgi, milleks on *EmailReminder*, oleks hea mõte lõpetada töö staatusega *ReminderSent*. Kuna üks töö võib saata mitmeid meeldetuletusi, võiks staatusel olla lisas veel kasulik informatsioon nagu konfiguratsiooni ID ning emaili aadress, kuhu meeldetuletus saadeti. Sel juhul on võimalik selgeks teha kellele ning mis meeldetuletus saadeti. Kui ühtegi meeldetuletust välja ei pea saatma, peaks staatus olema erinev ning sarnaselt informatiivne. Sarnaselt eelmisele staatusele oleks mõistlik staatuseks lisada *NoRemindersSent*, mis annab selgelt mõista et meeldetuletusi saata vaja ei olnud.

Kui tekib meeldetuletuse saatmisel mingi erind, pannakse töö staatuseks *ReminderFailed*, mille lisas on kirjeldatud vea põhjus ning konfiguratsiooni ID. Täpsem viga logitakse ka süsteemis. Sellise staatusega lõppedes, lisatakse tööle ajapiiranguga 30 minutiline periood, peale mida toimub uuesti proovimine. Manuaalselt on võimalik ka varem läbi kukkunud tööd uuesti käivitada, vajutades vastavalt nupule, milleks on kasutajaliideses *Run now*.

4.2 E-kirja põhja loomine

E-kirja loomiseks kasutati MJML tööriistakomplekti. MJML on HTML-il põhinev avatud lähtekoodiga keelespetsifikatsioon, mis on loodud e-posti mallide lihtsa ja optimaalse loomise ja haldamise jaoks. Kasutades elemente nagu <mj-section>, <mj-text>, ja <mj-button>, luuakse mall, mida saab hõlpsasti kohandada erinevate vajaduste jaoks. MJML-i eeliseks on ka see, et loodud mall teisendatakse automaatselt universaalselt ühilduvaks HTML-iks.

Peaosa: <mj-head>

Malli alustav <mj-head> plokk määrab e-kirja metaandmed, nimelt eelvaate teksti. <mj-preview> sisaldab teksti: „*Expiration reminder for '{report-name}'*“, kus *{report-name}* asendatakse dünaamiliselt vastava raporti nimega. Eelvaate tekst kuvatakse postkastis pealkirja kõrval või all. Kirja avades ei ole see tekst kasutajale nähtaval. Kirja peaosa on nähtaval joonisel 6.

```
<mj-head>
    <mj-preview>
        Expiration Reminder for '{report-name}'
    </mj-preview>
</mj-head>
```

Joonis 6. E-kirja *head.*

Kereosa: <mj-body>

<mj-body> plokk sisaldab e-kirja põhisisu.

<mj-wrapper> pakub üldist raamistikku ja võimaldab lisada täiendavat vormindust, näiteks täiendavad äärised ja sisu paigutamine. Käesolevas mallis ümbritseb see kogu sisu, lisades välised äärised (*padding*) üles, alla ja külgedele.

<mj-section> jaotab e-kirja sisu horisontaalseteks plokkideks, kus iga plokk võib sisaldada mitmeid elemente.

Esimest sektsiooni kasutatakse logo ja regulatsiooni nime jaoks. Need on grupeeritud kokku <mj-group> abil. Logo jaoks kaustatakse asetäitjat, mis viitab logo manusele, mida hoiustatakse projektis. Dünaamiline väärtus {*regulation-display-name*} kuvab regulatsiooni nime. Teksti suurus on määratud (24px), värviks tumelilla (#2E346F) ja tekstistiiliks Arial.

Edasi on kirjutatud teate pealkiri tekstina. Tekst „*Expiration Reminder for '{report-name}'*“ kuvatakse keskel, rõhutades raporti aegumise teadet rasvases tekstis. Dünaamiline väärtus {report-name} asendatakse hiljem raporti nimega. Pealkirja teksti hõlmav koodilõik on nähtaval joonisel 7.

```
  <mj-section padding="0" padding-bottom="10px">
      <!—Email name-->
      <mj-column>
      <mj-text align="center" font-size="18px" font-weight="700" font-
family="Arial" line-height="16px">
            Expiration Reminder for '{configuration-name}'
      </mj-text>
      </mj-column>
  </mj-section>
```

Joonis 7. Emaili pealkirja sektsioon.

Järgmisena esitatakse tabeliformaadis detailid (<mj-table>). Tabel kuvab seotud informatsiooni, milleks on nimetatud järjekorras konfiguratsiooni nimi, alguskuupäev, töötüüp, lõpukuupäev, sagedus ning töö looja. Reas esimene veerg on hall tekst, rasvases kirjas ja väikese fondiga (11px), samas kui teine veerg sisaldab dünaamilisi väärtusi, mis asendatakse hiljem õige väärtusega. Kõige esimeses reas konfiguratsiooni nimi on helesinises toonis, mis on sama toon kirja ülaosas paikneva IMS pildiga. Tabeli algus ning lõpp on nähtaval joonisel 8.

```
<mj-section border-radius="18px" background-color="#F2F3F7" padding="14px
16px">
  <!--Card with info rows-->
  <mj-column>
    <mj-table padding="0" font-family="Arial" line-height="16px" font-
size="14px">
      <tr>
        <td style="color: #6F728C; font-weight: 700; vertical-align: top;
padding: 7px 4px 4px 4px; width: 25%; font-size: 11px; line-height:
12px">CONF NAME</td>
        <td style="vertical-align: top; padding: 4px">{report-name}</td>
      </tr>
      ...
      <tr>
        <td style="color: #6F728C; font-weight: 700; vertical-align: top;
padding: 7px 4px 4px 4px; width: 25%; font-size: 11px; line-height:
12px">CONFIGURATION CREATOR</td>
        <td style="vertical-align: top; padding: 4px">{configuration-
creator}</td>
      </tr>
    </mj-table>
  </mj-column>
</mj-section>
```

Joonis 8. Tabel koos konfiguratsiooni infoga.

Eelviimane element kirjas on tegevusnupp, mis loodi <mj-button> abil. Sinine (värvikood: #2194f3) nupp viitab lingile {configuration-link}, mis võimaldab kasutajal otse vajaliku konfiguratsiooni juurde pääseda. Nupul on ümardatud servad (border-radius: 18px) ja tekstiks „Access configuration“. Sinine värv on sama, mis oli kasutusel konfiguratsiooni nimel ning IMS logol.

Viimane sektsioon kirjas on jalus. Seal asub teine logo, mis kuulub Playtechile. Selle asukoht on vasakul ja selle suurus on väike (74px x 16px).

5 Valminud lahenduse kirjeldus

Käesolevas peatükis tutvustatakse valminud lahendust ning selle töövoogu jooniste ja skeemide abil.

5.1 Ülevaade lahendusest

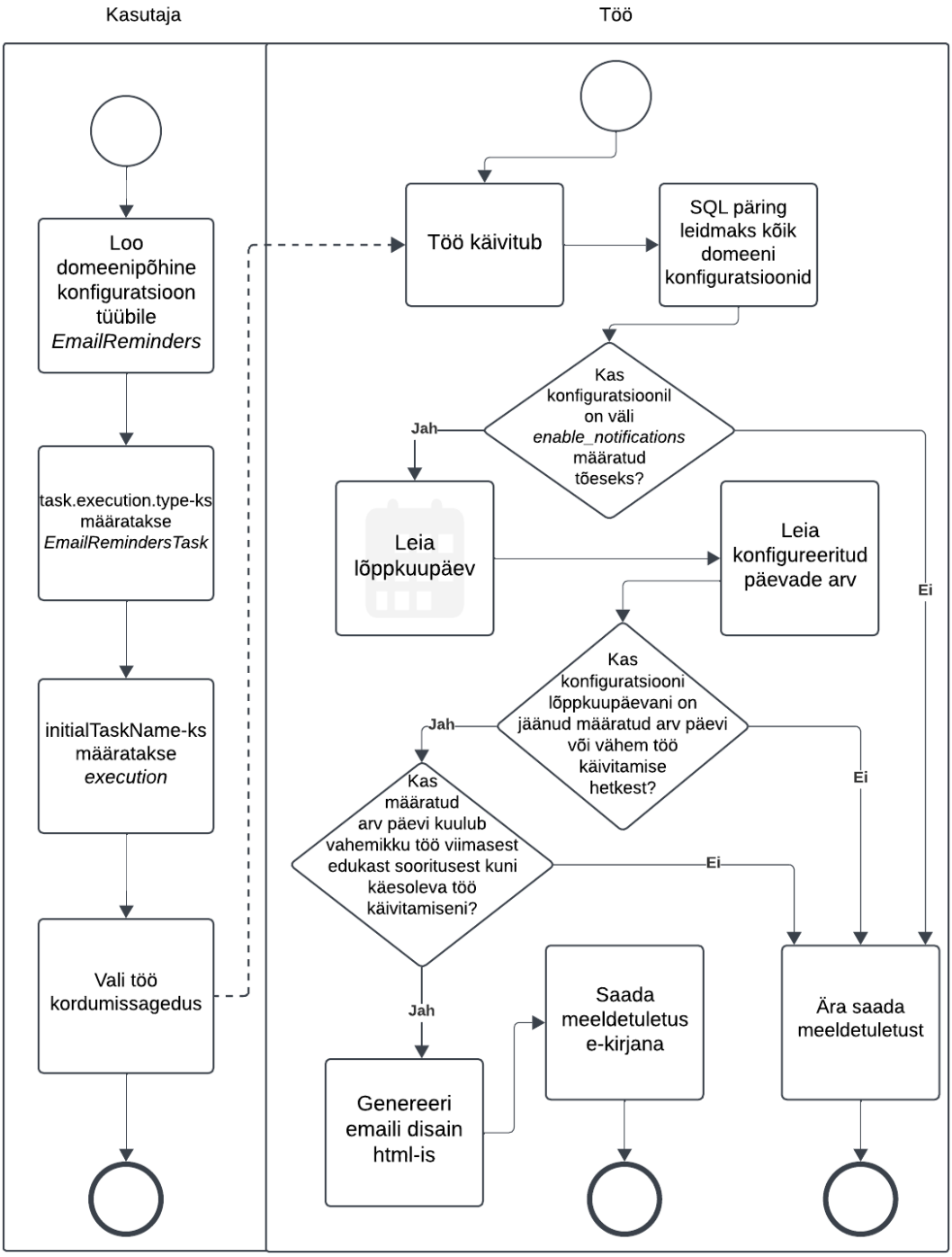
Töö käigus täiendas autor *Datalink* rakendust ning lisas sellele uue funktsionaalsuse luua *EmailReminders* töö ja määrata sellele kordumissagedus. Määratud sageduse põhjal saadab antud töö kliendile meeldetuletusi tema peagi aeguvast konfiguratsioonist. Joonis 9. EmailReminders töö loomine ja selle voog9 on näidatud töö *EmailReminders* loomine ning protsessi liikumine.

Protsess algab kasutajapoolse konfiguratsiooni loomisega töö tüübile *EmailReminders*. Et töö käivituks õigesti, tuleb väli *task.execution.type* jätta automaatselt täidetuks väärtusega *EmailRemindersTask*. See annab programmile käsu alustada protsessi just sellest klassist. Samamoodi on järgmise väljaga *initialTaskName*, mis peaks jääma ka automaatselt täidetuks väärtusega *execution*. Järgnevalt tuleb valida töö käivitumise sagedus, ehk kui sageli saadetakse meeldetuletusi. Kuna antud konfiguratsiooni peaks looma domeenile vaid ühel korral, on soovitatav kordumissagedus üks kord päevas, iga päev. Sagedust saab ka valida kasutades *cron* avaldist, läbi mille saab rohkem ajastust täpsustada.

Programmi enda poolt on esimeseks sammuks töö käivitumine eelnevalt konfiguratsioonis määratud ajastuse kohaselt. Töö koostab SQL-päringu leidmaks kõik konfiguratsioonid domeeni kohta. Päringuga leitakse kõik tööd, millel:

- väli *enable_notifications* on määratud tõeseks
- lõppkuupäevani on jäänud määratud arv päevi või vähem töö käivitamise hetkest
- määratud arv päevi kuulub vahemikku töö viimasest edukast sooritamisest kuni käesoleva käivitamiseni.

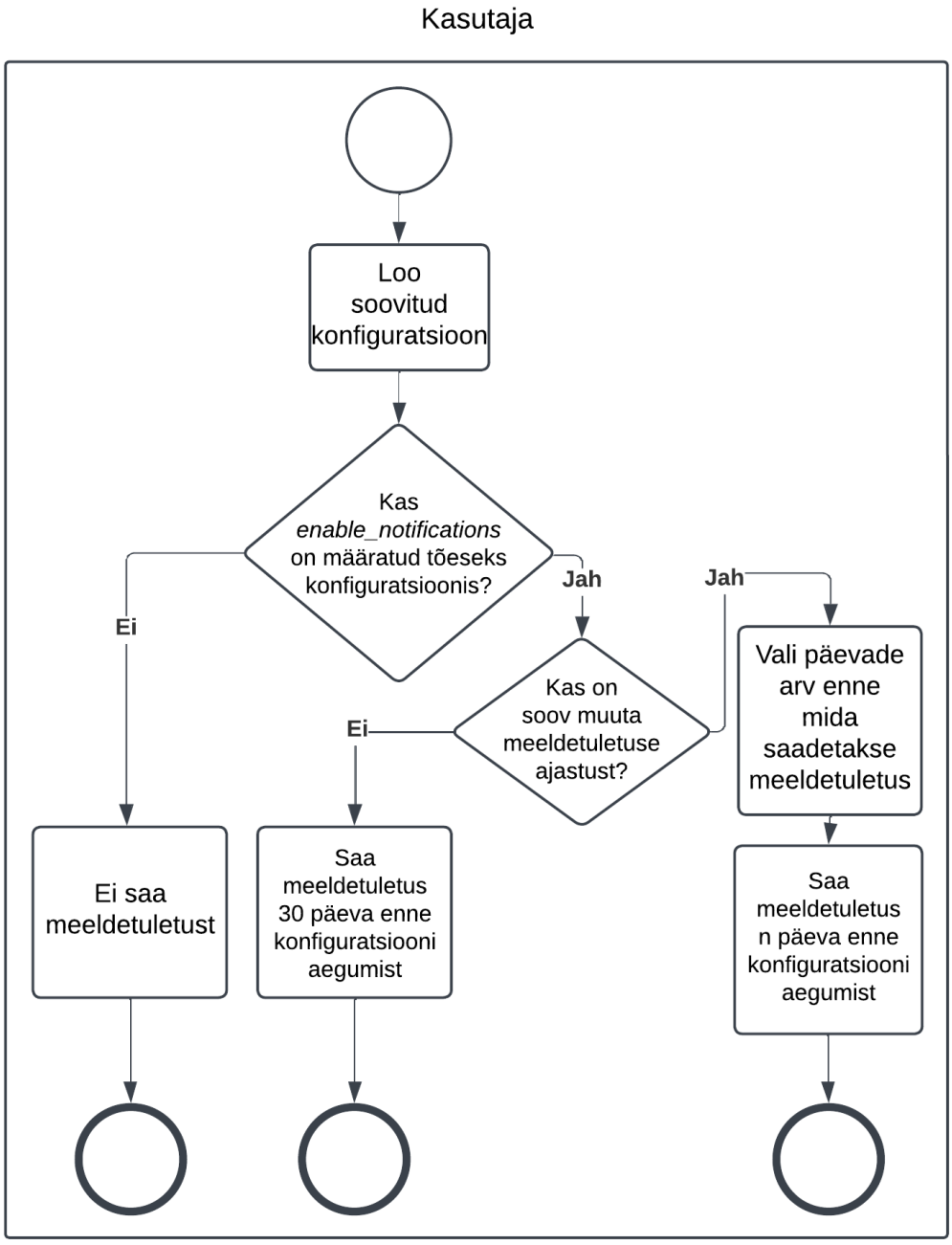
Kui mõni neist tingimustest ei ole rahuldatud, siis meeldetuletust ei koostata. Kõikidele leitud töödele genereeritakse eraldi e-kiri HTML koodis, täidetakse spetsiifiliste väärtustega ning saadetakse töö autorile e-kiri konfiguratsiooni aegumise kohta.



Joonis 9. EmailReminders töö loomine ja selle voog.

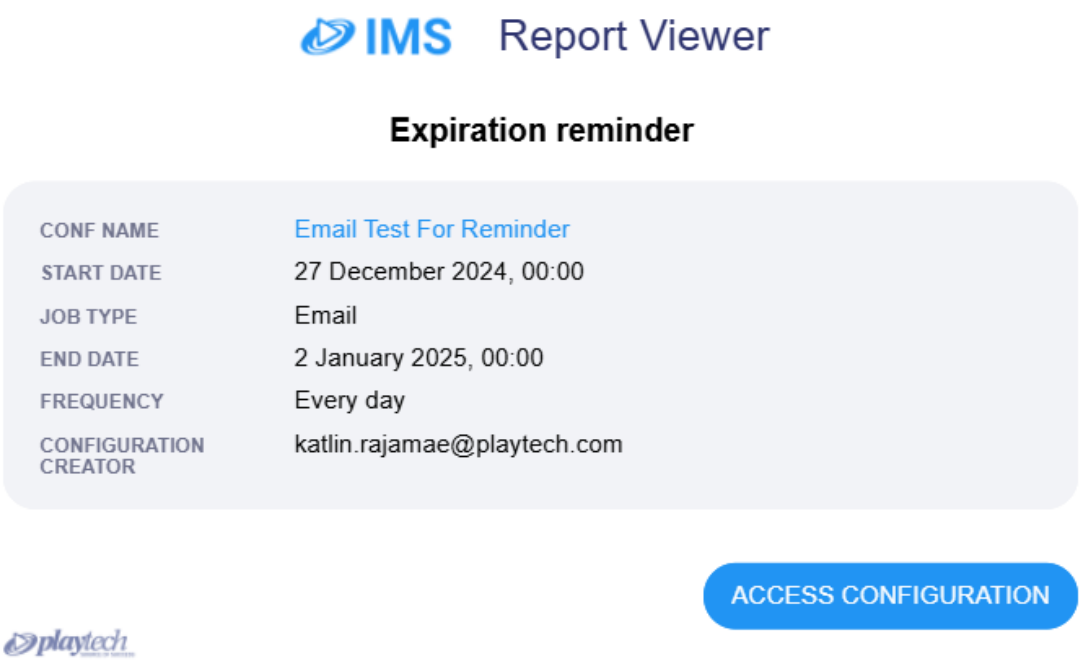
Joonisel 10 on välja toodud skeem uuendatud konfiguratsiooni loomisest, kus saab valida meeldetuletuste saamise. Lisatud on uus väli nimega *enable_notifications*, kui see on tõeseks määratud, tekib nähtavale uus väli *days_before_reminder*, kuhu saab lisada päevade arv täisarvudes, mis määrab kui mitu päeva enne töö aegumist meeldetuletus genereeritakse ja saadetakse. Automaatselt on välja väärtus 30, mis tähendab, et

meeldetuletus saadetakse 30 päeva enne aegumist, kuid saab ka valida n päevade arvu. Kasutajale ei saadeta meeldetuletust kui väli *enable_notifications* on märgitud mitte tõeseks.



Joonis 10. Konfiguratsiooni loomine ja meeldetuletused

Valminud lahenduse välja saadetav meeldetuletuse disain on nähtav jooniselJoonis 11. Valminud meeldetuletuse disain Käesolev e-kiri on avatud kasutades Outlook platvormi, erinevatel platvormidel võib esineda minimaalseid erisusi.



Joonis 11. Valminud meeldetuletuse disain

6 Tulemused

Käesoleva lõputöö käigus valmis e-kirjade genereerimise süsteem, läbi mille saadetakse peagi aeguvate konfiguratsioonide loojatele selle kohta meeldetuletus. Süsteem täidab kõik seatud funktsionaalsed ja mittefunktsionaalsed nõuded. Kuigi funktsionaalsus ei ole veel avalik ettevõtte klientidele, on see kasuks tulnud töötajatele ettevõttesiseselt, lihtsustades loodud konfiguratsioonidel järje pidamist ning vähendades aega tegelemaks aegunud konfiguratsioonide taastamisega tagantjärgi. Uue funktsionaalsuse klientideni tuues, aitab see tõsta Playtechi väärtust ning klientide rahulolu.

6.1 Testimine

Testimine on tarkvaraarenduse oluline komponent. Testimine tagab, et rakendused töötavad korrektselt kõikides stsenaariumites. Testimise tulemus kinnitab, et rakendus vastab etteantud nõuetele. [26]

Selleks, et kindlaks teha lisafunktsionaalsuse eesmärgipärane toimimine ning vigade puudumine, teostati sellele testimine. Antud lahendusele eraldi teste ei kirjutatud vaid kontrolliti manuaalselt. Testimisel oli oluline õige arvutuskäik, millal meeldetuletus saadetakse, nii tavapärases olukorras – 30 päeva enne aegumist kui ka kasutaja poolt määratud päevade hulk enne aegumist. Lisaks kontrolliti olukorda kus samal päeval peab mitmele tööle meeldetuletuse saatma, ning olukorda kus pole ühtegi meeldetuletust saata vaja. Oluline oli ka andmete õigsus, millele samuti pöörati tähelepanu, et ükski väli ei jääks tühjaks ning et neisse valed väärtused ei satuks.

Testimisel loodi mitmed stsenaariumid, mille käigus kõik saadetud meeldetuletused olid õiged ning õigete väärtustega. Saatmata ei jäänud ühtegi e-kirja ning meeldetuletuste puudumisel ei saadetud ka midagi üleliigset.

6.2 Edasiarenduse võimalused

Loodud lahenduse edasiarenduseks näeb autor kahte võimalust. Järgmiseks oluliseks funktsionaalsuseks oleks otse e-kirjast konfiguratsiooni lõpp-kuupäeva muutmine. See aitaks kasutajal veelgi aega säästa, kaotades ära vajaduse veebilehele minna ning seal vajalik muudatus teha. Selle lähenemise puhul aga on vaja mõelda turvalisusele – kuidagi

peab tagama kasutaja autentsuse ning ka rakenduse poolt peab teada olema kes muudatuse tegi ning kas kasutajal selleks õigust on.

Teine edasiarenduse võimalus oleks selliste juhtude jaoks, kus konfiguratsiooni looja ning selle reaalne saaja on erinevad. Sellisel puhul saadetakse meeldetuletus mõlema kasutaja emailile. Tööde puhul, kus autor saab valida teise kasutaja, kellele raport saadetakse, võib raporti saajale jääda segaseks, miks enam raporteid ei saadeta. Antud kasutajale oleks ülevaate saamiseks kasulik samuti meeldetuletus saada peatse lõppemise puhul. Nii saaks kasutaja kas ise konfiguratsiooni muuta või õiguste puudumisel anda teada autorile, et muudatus on vajalik.

7 Kokkuvõte

Käesolev töö keskendub ettevõtte Playtech *Datalink*-süsteemi täiustamisele, pakkudes lahendusi, mis parandavad nii kasutajakogemust kui ka süsteemi tõhusust. *Datalink* on oluline tööriist andmete transformeerimiseks ja edastamiseks vastavalt regulatsioonidele, hõlmates domeenispetsiifilisi nõudeid.

Töös keskendutakse uue meeldetuletuste funktsionaalsuse arendamisele, mis teavitab kasutajaid lähenevatest konfiguratsioonide aegumistähtaegadest. Süsteem loodi lisana olemasolevasse *Datalink* programmi. Kasutajatele tekkis võimalus valida meeldetuletuste saamine e-kirja teel ning samuti saab valida päevade arvu, kui palju enne konfiguratsiooni aegumist meeldetuletus saadetakse.

Lõputöö käigus läbiviidud analüüsist selgus, et funktsionaalsust oleks mõistlik rakendada eraldi tööna, mis käivitub iga päev, kui just ei ole muudmoodi sätestatud. Töö käivitudes otsib see üles kõik konfiguratsioonid, millele oleks vaja vaadeldava päeva jooksul meeldetuletus saata ning genereerib ja saadab e-kirja selle autorile. Lahenduse loomiseks valiti Java programmeerimiskeel ning seda toetama Akka raamistik, kuna need sobisid süsteemi nõuete täitmiseks kõige paremini. E-kirja puhul määrati selle malli disain ning valiti kõige optimaalsem märgistuskeel selle loomiseks, mis vastaks kõikide emaili brauserite nõuetele. Valituks osutus MJML.

Bakalaureusetöö eesmärgid said täidetud. Uus funktsionaalsus loob lisaväärtust nii klientidele, kelle rahulolu ja lojaalsus süsteemile tõusevad, kui ka ettevõtte töötajatele, vähendades manuaalseid tööprotsesse. Testimise ja kasutajate tagasiside kogumise tulemusena hinnati arenduse edukust, mis toob kaasa süsteemi jätkusuutlikuma arengu ja efektiivsema kasutuse igapäevastes tööprotsessides. See projekt on näide, kuidas tarkvaralahenduste abil saab suurendada tõhusust ja tugevdada kliendisuhteid rahvusvahelises ettevõttes.

8 Kasutatud kirjandus

[1] „Estonia - Playtech People,“ [Võrgumaterjal]. Available: https://www.playtechpeople.com/country/estonia/. [Kasutatud 3 12 2024].

[2] K. Wiegers ja J. Beatty, Software Requirements, Microsoft Press, 2013.

[3] I. Sommerville ja P. Sawyer, Requirements Engineering: A Good Practice Guide, Wiley, 1997.

[4] T. Rolandsson, „A Study on Software Requirements Specifications - some examples (Bachelor's thesis, Lund University),“ Lund University Publications, Lund, 2010.

[5] „IEEE Recommended Practice for Software Requirements Specifications,“ *IEEE Std 830-1998,* 1998.

[6] V. Fulber-Garcia, „Requirements: Functional vs. Non-functional,“ 2024. [Võrgumaterjal]. Available: https://www.baeldung.com/cs/requirements-functional-vs-non-functional. [Kasutatud 3 12 2024].

[7] „The Top Programming Languages 2024,“ IEEE Spectrum, [Võrgumaterjal]. Available: https://spectrum.ieee.org/top-programming-languages-2024. [Kasutatud 27 12 2024].

[8] „Introduction to Java,“ 19 11 2024. [Võrgumaterjal]. Available: https://www.geeksforgeeks.org/introduction-to-java/. [Kasutatud 03 12 2024].

[9] „Difference between Python and Java,“ GeeksforGeeks, [Võrgumaterjal]. Available: https://www.geeksforgeeks.org/difference-between-python-and-java/. [Kasutatud 27 12 2024].

[10] „Python,“ [Võrgumaterjal]. Available: https://www.python.org/. [Kasutatud 27 12 2024].

[11] „Advantages of Python | Disadvantages of Python,“ Python Geeks, [Võrgumaterjal]. Available: https://pythongeeks.org/advantages-disadvantages-of-python/. [Kasutatud 27 12 2024].

[12] „Java Spring Pros and Cons - Javatpoint,“ [Võrgumaterjal]. Available: https://www.javatpoint.com/java-spring-pros-and-cons. [Kasutatud 03 12 2024].

[13] „Akka vs Spring: What are the differences?,“ stackshare, [Võrgumaterjal]. Available: https://stackshare.io/stackups/akka-vs-spring. [Kasutatud 29 12 2024].

[14] „Akka for Beginners: Benefits and Best Practices,“ Poespas Blog, 4 5 2024. [Võrgumaterjal]. Available: https://blog.poespas.me/posts/2024/05/05/akka-for-beginners-benefits-and-best-practices/. [Kasutatud 3 12 2024].

[15] C. Slater, „Love it or Leave It? Email Developers Weigh In On Email Frameworks,“ Litmus, 6 12 2022. [Võrgumaterjal]. Available: https://www.litmus.com/blog/email-framework-pros-cons. [Kasutatud 3 12 2024].

[16] Indeed Editorial Team, „How To Create an HTML Email (With Tips),“ 16 10 2023. [Võrgumaterjal]. Available: https://www.indeed.com/career-advice/career-development/how-to-create-html-email. [Kasutatud 3 12 2024].

[17] „<heml>,“ [Võrgumaterjal]. Available: https://heml.io/docs/getting-started/overview. [Kasutatud 3 12 2024].

[18] „Difference between MJML and HEML ?,“ GitHub, [Võrgumaterjal]. Available: https://github.com/SparkPost/heml/issues/12. [Kasutatud 29 12 2024].

[19] „mjml,“ mjml, [Võrgumaterjal]. Available: https://documentation.mjml.io/. [Kasutatud 3 12 2024].

[20] P. Mioni, „Choosing a Responsive Email Framework: MJML vs. Foundation for Emails,“ CSS-Tricks, 23 04 2018. [Võrgumaterjal]. Available: https://css-tricks.com/choosing-a-responsive-email-framework%e2%80%8amjml-vs-foundation-for-emails/. [Kasutatud 03 12 2024].

[21] „The Leading Java and Kotlin IDE,“ JetBrains, [Võrgumaterjal]. Available: https://www.jetbrains.com/idea/. [Kasutatud 7 12 2024].

[22] „Quartz Scheduler,“ Quartz, [Võrgumaterjal]. Available: https://www.quartz-scheduler.org/. [Kasutatud 09 12 2024].

[23] „Job scheduler,“ Wikipedia, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Job_scheduler. [Kasutatud 6 1 2024].

[24] „Interface Supplier<T>,“ Oracle, [Võrgumaterjal]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html. [Kasutatud 09 12 2024].

[25] MyBatis, [Võrgumaterjal]. Available: https://mybatis.org/mybatis-3/. [Kasutatud 2 1 2025].

[26] E. Whiting ja S. Datta, „Design and Development of a Technology-Agnostic NFR Testing Framework: Introducing the framework and discussing the future of load testing in Agile software development,“ Association for Computing Machinery, New York, 2022.

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

Mina, Kätlin Rajamäe

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Meeldetuletuste genereerimine konfiguratsiooni lõppkuupäeva lähenemise puhul“, mille juhendaja on Kristiina Hakk
   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. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

06.01.2025

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.