

TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Madis Kõivopuu
Rakenduse loomine e-kirjadest sündmuste
otsimiseks ja organiseerimiseks
Bakalaureusetöö (9 EAP)

Juhendajad:
Pelle Jakovits, PhD
Chinmaya Kumar Dehury, PhD

Tartu 2025

Rakenduse loomine e-kirjade sisust sündmuste otsimiseks ning organiseerimiseks

Lühikokkuvõte:

Igal Tartu Ülikooli tudengil on ut.ee domeeni e-posti konto, kuhu saadetakse õppetööga seotud infot, sealhulgas ka Tartu Ülikooliga seotud sündmusi. Olenevalt inimesest võib talle päevas palju sündmustega e-kirju tulla, mistõttu on neid kõiki ajakulukas kalendrisse märkida. Selle bakalaureusetöö eesmärk oli luua uus avatud lähtekoodiga rakendus, mis loeb kasutaja e-kirjad läbi ning automaatselt lisab leitud sündmused kasutaja vabalt valitud kalendrisse. Selle eesmärgi saavutamiseks uuriti varasemaid tarkvaralahendusi sündmuste otsimiseks e-kirjadest, et leida nende probleemid ning need uues rakenduses lahendada. Samuti analüüsiti tehnoloogiaid, mida uue rakenduse loomiseks kasutada saaks. Sündmuste otsimiseks kasutati keelemudelit Llama-3.2-3B-Instruct, mida treeniti 90 e-kirja peal, et keelemudel paremini sündmusi suudaks e-kirjast leida. Töö tulemusena valmis mikroteenuste arhitektuuril põhinev veebirakendus, mis võimaldas kasutajal Microsofti kontoga rakendusse sisse logida, manuaalselt või automaatselt rakendusel sündmusi leida ning need kalendrisse eksportida. Kuigi rakendus toimis ning selles parandati ära mõningad eelnevate lahenduste probleemid, siis tegi keelemudel sündmuste leidmisel liiga palju vigu nagu näiteks sündmustele vale kuupäeva või aadressi märkimine. Siiski on loodud rakenduse arhitektuur heaks aluseks tuleviku edasiarendustele, sest uusi funktsionaalsusi on võimalik juurde lisada uute mikroteenustena ning olemasolevaid teenuseid ei pea palju muutma.

Võtmesõnad: E-kiri, sündmuste leidmine, masinõpe, suur keelemudel (SKM), Llama

CERCS: P175 Informaatika, süsteemiteooria

Development of an Application for Finding and Organising Events From E-mails

Abstract:

Every University of Tartu student has a ut.ee e-mail account where information related to studies gets sent. This also includes events that are related to the University of Tartu. Depending on the person, one can get a large amount of e-mails containing events which can make it cumbersome to manually mark all of them in the calendar. The aim of this thesis was to create a new open-source application that reads through the user's e-mails and automatically adds found events to a calendar of the user's choice. To achieve that goal, multiple previously developed solutions were examined to find the problems with those solutions and fix them in this thesis. After that an analysis of different technologies was done to figure out what to use in the development of the new app. For parsing events from e-mails, a language model called Llama-3.2-3B-Instruct was used which was additionally trained on 90 e-mails to improve its event parsing accuracy. As a result of the development, a web app based on the microservices architecture was created which allowed users to log in with their Microsoft account, use the web app to manually or automatically find events and export them into a calendar. Although the app worked for finding events and it did fix some of the issues the previous solutions had, the large language model made a lot of mistakes such as producing an incorrect date for an event or giving an event an incorrect address. In spite of that, the created application's microservices architecture is a good base for future development since new functionalities can be added as new microservices, without having to modify the existing architecture a lot.

Keywords: E-mail, event detection, machine learning, large language model (LLM), Llama

CERCS: P175 Informatics, systems theory

Sisukord

1.	Sissejuhatus.....	6
2.	Mõisted	8
3.	Teoreetiline taust.....	10
3.1.	Varasemad tarkvaralahendused sündmuste leidmiseks	10
3.2.	Selgunud probleemid ja järeldused	12
3.3.	Veebirakenduste arhitektuurid	13
4.	Rakenduse planeerimine	15
4.1.	Funktsionaalsed ja mittefunktsionaalsed nõuded.....	15
4.2.	Kasutatavad tehnoloogiad.....	16
4.2.1	Arhitektuur.....	16
4.2.2	Konteinerid ja nende orkersteerimine	17
4.2.3	Kasutajaliides	18
4.2.4	Tagasüsteemi teenused.....	18
4.2.5	Sõnumimaakler	19
4.2.6	Andmebaas.....	20
4.2.7	Keelemudel	21
4.3.	Rakenduse mikroteenuste struktuur	22
5.	Keelemudeli rakendamine ja peenhäälestamine	24
5.1.	Kasutatav riistvara ning tarkvara	24
5.2.	Keelemudeli viipade ja väljundite formaat	25
5.3.	Peenhäälestamata keelemudeli testimine.....	27
5.3.1	Testandmestiku kirjeldus	27
5.3.2	Testimismetoodika.....	29
5.3.3	Tulemused.....	31
5.4.	Keelemudeli peenhäälestamine.....	33
5.4.1	Treeningandmestiku kirjeldus.....	33
5.4.2	Treeningandmete eeltöötlus	33
5.4.3	Peenhäälestus	35
6.	Valminud rakendus	37
6.1.	Autentimisleht.....	37
6.2.	Navigatsiooniriba.....	40
6.3.	Sündmuste nimekirja avaleht.....	41

6.3.1	Otsinguriba ja kalendrilingi nupp	42
6.3.2	Sündmuste nimekiri	44
6.4.	Kasutajasätete leht	46
7.	Testimine ja arutelu.....	48
7.1.	Testimise keskkond.....	48
7.2.	Mittefunktsionaalsete nõuete testimine.....	48
7.2.1	Rakenduse jõudlustest.....	48
7.2.2	Peenhäälestatud keelemudeli testimine.....	52
7.2.3	E-kirjade krüpteerimise kontrollimine	54
7.3.	Järeldused ning edasiarenduse võimalused.....	55
8.	Kokkuvõte.....	57
	Viidatud kirjandus.....	59
	Lisad.....	61
I.	Keelemudelig GPT-4o genereeritud terminite definitsioonid.....	61
II.	Keelemudeli süsteemiviip.....	63
III.	Keelemudeli genereeritud sündmuste hindamise juhend.....	65
IV.	Peenhäälestamise programm.....	67
V.	Peenhäälestatud keelemudeli failid.....	68
VI.	Rakenduse lähtekood	69
VII.	Rakenduse paigaldusjuhend.....	70
	Litsents.....	71

1. Sissejuhatus

Igal Tartu Ülikooli üliõpilasel on ut.ee domeenis enda e-posti konto, kuhu saadetakse õppetööga seotud infot [1]. Nende e-kirjade hulgas ei ole ainult otseselt õppeainetega seotud sõnumid, vaid ka kutsed informaatikaga seotud üritustele nagu näiteks infotundidesse, seminaridesse, häkatonidele ning külalisloengutele. Kui üliõpilane kuulub tudengiorganisatsiooni, siis saadetakse talle lisaks ka sellega seotud teavitusi. Seetõttu võib tudengi e-posti tulla palju informatsiooni, mis on seotud tähtaegadega ning mida ta peab läbi töötleva. Seepärast võib üliõpilasel ära ununeda kalendrisse märkida endale tähtsamad sündmused, mistõttu võib ta sündmuse toimumise ära unustada. Sarnane probleem on ka osadel Tartu Ülikooli õppejõududel, kuna nemad saavad palju töökohustustega seotud e-kirju.

Selle probleemi parandamiseks on varasemalt loodud mitmeid rakendusi [2-5], mis automaatselt loevad kasutaja e-kirjad läbi ning näitavad leitud sündmusi rakenduses. Samas oli nendes rakendustes puudusi, näiteks mõned rakendused ei võimaldanud reaajas e-kirjadest sündmusi otsida, vaid see toimus ainult siis, kui kasutaja rakenduse avas. Paraku aga tähendab see, et kui tudengi e-posti saabub ühes päevas palju e-kirju, siis ta peaks mitu korda rakenduse avama, et uued sündmused kalendrisse märkida. See tegevus ei ole automaatne, mistõttu võib ikka juhtuda, et mõningad sündmused jäävad kalendrisse märkimata. Samuti ei leidnud mõned rakendused korrektselt sündmusi sellistest e-kirjadest, milles oli mitu sündmust. Tartu Ülikooli e-postile saadetakse aga tihti just selliseid e-kirju, mis sisaldavad endas mitu sündmust. Nende probleemide tõttu ei sobi eelnevad rakendused Tartu Ülikooli e-postist sündmuste leidmiseks.

Selle uurimistöö eesmärk on luua uus avatud lähtekoodiga rakendus, mis võimaldab e-kirjadest automaatselt sündmusi otsida ning milles parandatakse eelnevate rakenduste probleeme. Loodav rakendus otsib kasutaja Tartu Ülikooli e-posti kontolt sündmusi ning salvestab need kalendrisse. Selles bakalaureusetöös piirdutakse ainult Microsoft Outlookist sündmuste otsimisega, kuna vastasel juhul muutub bakalaureusetöö liiga mahukaks. Samas aga võetakse rakenduse arendamisel arvesse, et tulevikus oleks võimalikult lihtne teisi e-posti teenusepakkujaid lisada.

Bakalaureusetöö koosneb viiest sisupeatükist. Esimeses peatükis tuuakse detailsemalt välja varasemate tarkvaralahenduste kirjeldused ning nende probleemid. Nende probleemide põhjal tehakse järeldused, kuidas uut rakendust on võimalik täiustada ning luuakse funktsionaalsed ning mittefunktsionaalsed nõuded siinses töös loodava rakenduse jaoks. Samuti kirjeldatakse selles peatükis peamisi veebirakenduste arhitektuure, et töös oleks paremini aru saada, miks

mingi arhitektuur selle rakenduse jaoks paremini sobib. Teises peatükis antakse ülevaade rakenduse loomiseks kasutatud tehnoloogiatest ning nende valiku põhjustest võrreldes alternatiividega. Kolmandas peatükis kirjeldatakse e-kirjadest sündmuste leidmiseks kasutatava keelemudeli testimist ning selle tulemusi. Samuti tuuakse selles peatükis välja, kuidas keelemudelit peenhäälestati, et see paremini e-kirjadest sündmusi leiaks. Neljandas peatükis kirjeldatakse valminud rakendust ning antakse ülevaade sellest, kuidas rakendus töötab. Viiendas peatükis kirjeldatakse täpsemalt rakenduse testimise protsessi, analüüsitakse testimise tulemusi ning tehakse järeldusi rakenduse edasiarendamiseks.

Selle bakalaureusetöö koostamisel kasutati ChatGPT-4o¹ abi. Seda keelemudelit ei kasutatud, et bakalaureusetöö peatükkide jaoks lõike genereerida. ChatGPT-4o'1 lasti mõningad terminid defineerida, mille puhul lisati viited tehisintellektile, ning paar autori välja mõeldud pealkirja ümber nimetada, et need täpsemini peatüki sisu kirjeldaks. Samuti andis tehisintellekt hinnangu autori kirjutatud kasutatud tehnoloogiate peatükile. Selles tagasisides andis tehisintellekt teada, kas tehnoloogiate valikud on hästi ära põhjendatud ning kas neid on võrreldud mõne alternatiiviga. Probleemide korral andis GPT-4o soovitusi, mida paremini teha saaks, kuid selles tagasisides genereeritud teksti töös otse ei ole kasutatud.

¹ <https://platform.openai.com/docs/models/gpt-4o> (09.05.2025)

2. Mõisted

HTTP (ingl *HyperText Transfer Protocol*) on andmevahetusprotokoll, mida kasutatakse Internetis dokumentide vahetamiseks².

Närvivõrgu kaal (ingl *weight*) ehk **tehisneurooni ühendustugevus** on täisarv, millega ühe neurooni sisend läbi korrutatakse³.

Kaofunktsioon (ingl *loss function*) on matemaatiline funktsioon, mis mõõdab masinõppemudeli ennustuste ja tegelike väärtuste vahelist erinevust⁴.

Keelemudel (ingl *language model*) on loomulikku keelt kirjeldav tõenäosuslik mudel, mis põhineb keeleandmete toetuvatel algoritmidel (nt statistiline mudel, närvivõrgu mudel)⁵.

Kiht (ingl *layer*) on hulk tehisneuroneid, mis on organiseeritud ühte gruppi ning mis on omavahel ühendatud³.

Kvantiseerimine (ingl *quantization*) on tehnika, millega vähendatakse masinõppe mudeli mälu- ning arvutuslikku kulu, asendades mudeli kaalud ning aktivatsiooni andmetüübid väiksema täpsusega andmetüüpidega (näiteks 32-bitiselt ujukomaarvult 8-bitisele täisarvule)⁶.

Masinõpe (ingl *machine learning*) on arvuti treenimine etteantud andmete, reeglite põhjal otsuseid ja ennustusi tegema, vastavate algoritmide koostamine (nt tehislike närvivõrkude, klasterdamise meetodil)⁵.

Mudeli treenimine (ingl *model training*) on andmestikust optimaalse mudeli parameetrite määramise ülesanne masinõppes⁷.

Parser (ingl *parser*) on programm või programmiosa, mis analüüsib sisendteksti struktuuri ja teisendab selle struktureeritud andmeks, mida arvuti saab edasi töödelda⁴.

Server (ingl *server*) on riistvarasüsteem või programm, mis annab teenust teistele arvutitele⁷.

² Termin seletus on võetud e-teatmikust <http://www.vallaste.ee/>.

³ Termin seletus on võetud tehisintellekti algkursuse (LTAT.TK.013) veebilehelt https://courses.cs.ut.ee/2020/Tehisintellekti_algkursus/Main/PARTIITehis%c3%a4rviv%c3%b5rgud (30.04.2025)

⁴ Termin seletuse genereeris GPT-4o. Tabel termini, kasutatud viiba ning keelemudeli väljundi kohta on leitav lisas 1.

⁵ Termin seletus on võetud Sõnaveebist <https://sonaveeb.ee/>.

⁶ Termin on võetud HuggingFace'i dokumentatsioonist ning tõlgitud eesti keelde. https://huggingface.co/docs/optimum/en/concept_guides/quantization (30.04.2025)

⁷ Termin seletus on võetud Andmekaitse ja infoturbe leksikonist <https://akit.cyber.ee/>.

Suur keelemudel (ingl *large language model*) on keelemudel, mis on treenitud suurel hulgal tekstiandmetel, et luua loomuliku keele väljundeid ja sooritada erinevaid loomuliku keele töötlemise ülesandeid (nt tõlkimine, kokkuvõtete tegemine, tundeanalüüs)⁸.

Token (ingl *token*) on väikseim tähenduslik ühik, milleks tekst enne töötlemist jagatakse, näiteks sõna, kirjavahemärk või alamüksus, mida kasutatakse keelemudeli sisendina⁹.

Viibasüst (ingl *prompt injection*) on turvanõrkus, mille puhul pahatahtlik kasutaja sisestab keelemudeli sisendi ehk viiba sisse sellise teksti, mis muudab või juhib mudeli käitumist viisil, mida algne süsteemiarhitekt ei soovinud, näiteks möödudes turvapiirangutest või tekitades valesid vastuseid⁹.

⁸ Termin seletus on võetud Sõnaveebist <https://sonaveeb.ee/>.

⁹ Termin seletuse genereeris GPT-4o. Tabel termini, kasutatud viiba ning keelemudeli väljundi kohta on leitav lisas 1.

3. Teoreetiline taust

Varasemalt on loodud mitmeid tarkvaralahendusi, et e-kirjadest automaatselt sündmusi leida ning need e-posti kasutaja kalendrisse lisada. Selles peatükis tuuakse välja neli sellist varasemat lahendust, nende arhitektuuri kirjeldused, rakenduse testimise tulemused ning selgunud nõrkused. Peale olemasolevate lahenduste kirjeldamist tuuakse välja varasemate lahenduste ühised probleemid, et teha järeldused, mida on võimalik võrreldes eelnevate lahendustega parandada selles bakalaureusetöös. Kõige viimasena tuuakse välja peamised veebirakenduste arhitektuurid, et töös edaspidi paremini mõista, miks siinse töö rakenduse jaoks mingi arhitektuur paremini sobib kui teine.

3.1. Varasemad tarkvaralahendused sündmuste leidmiseks

Gambino jt [2] arendasid hispaaniakeelsetest e-kirjadest sündmuste tuvastamiseks serverteenuse koos mobiilirakendusega. Nende serverteenus luges kasutajate Gmaili ning Outlooki konto e-kirju ning töötles neid kolmes osas. Esiteks töötles server e-kirja teksti läbi ning jagas e-kirja masinõppe mudeliga kahte kategooriasse: sündmustega ning sündmusteta e-kiri. Kui sündmusi leiti, jätkati e-kirja töötlemist, tuvastades e-kirjast nimiolemid (ingl *named entity*) nagu kuupäevad ja kohta määravad sõnad ning nende põhjal lasti parseril e-kirjast sündmused väljastada, mis salvestati kasutaja Google'i kalendrisse. Mobiilirakendus võimaldas kasutajal logida enda e-posti kontoga loodud rakendusse sisse, saada teavitusi leitud sündmuste kohta ning vaadata nende üksikasju. Rakenduse testimiseks kasutasid nad 300 e-kirja, milles täpselt pooltes oli sündmusi. Nad leidsid, et tehisintellekti kategoriseeris sündmusteta e-kirju õigesti 97% juhtudest ning sündmustega e-kirju õigesti 94% juhtudest. Sündmuste parserit testisid nad 60-ne sündmusi sisaldavate e-kirjadega, millest selgus, et sündmuste parser leiab korrektse informatsiooni 81% juhtudest. Seega soovitasid nad töö lõpus täpsemaks sündmuste info tuvastamiseks suurendada e-kirjade hulka, kuna sündmusi on võimalik vormistada paljudel erinevatel viisidel ning nende parser ei suutnud erijuhtude puhul sündmusi leida.

Wasi jt [3] arendasid pistikprogrammi, mis jookseb kasutaja arvutis ning mida on võimalik siduda meilikliendiga. Sündmuste leidmiseks töödeldi e-kirja sisu läbi mitmes osas. Esimesena tegi rakendus kirjadele eeltötluse, mis eemaldas e-kirja eelnevate vastuste jada ja lõputervituse. Järgnevalt määras programm igale sõnale nimiolemi märgistuse (ingl *part of speech tag*) ning andis märgistatud teksti edasi olekumasinatele. Nende töös toimusid olekumasinad kui parserid, ehk siis olekumasinat kasutamine aitas e-kirjast väljastada

sündmused. Sündmuste leidmise täpsuse (ingl *accuracy*) teada saamiseks kasutasid nad 1000 e-kirja, millest 770-s olid sündmused ning 230-s ei olnud. Testimise tulemusena leidsid nad, et keskmiselt leiab nende programm 91% juhtudest sündmused õigesti. Samas on aga nende sõnul seltskondlike üritustega seotud e-kirjade puhul täpsus palju väiksem, kuna nende kirjade puhul erinesid sündmuste vormistused väga palju, mistõttu oli olekumasinatega neist raskem sündmusi leida. Artikli kokkuvõttes tõid nad välja, et kuigi programm tuvastas e-kirjadest sündmused õigesti, siis oli sündmuste asukoht tihti vale. Samuti ütlesid nad, et nende programm ei tööta korrektselt, kui üks e-kiri sisaldab endas rohkem kui ühte sündmust.

Sarnaselt esimesele (Gambino jt) rakendusele arendasid Nair jt [4] e-kirjadest sündmuste tuvastamiseks kaks programmi – Androidi rakendus ning e-kirjade töötlemise server. Nende Androidi rakendus leidis igal avamisel kasutaja Gmaili kontolt e-kirjad, saatis need serverile ning tulemusena tagastati nendest e-kirjadest leitud sündmused, mida oli Androidi rakenduses võimalik Google'i kalendrisse lisada. Serveris kasutati kahte masinõppe mudelit, millest esimene mudel kategoriseeris e-kirja sündmusteta ning sündmustega e-kirja kategooriasse. Teine mudel töötles sündmustega e-kirju ning määras sõnadele nimiolemi sõna tähenduse põhjal, nagu aeg ja toimumiskoht. Peale e-kirja teksti töötlemist kasutati serveris regulaaravaldisi, mis parsisid töödeldud tekstist sündmuste informatsiooni (sündmuse nimi, alguskuupäev, asukoht). Rakenduse testimiseks kasutasid nad 150 e-kirja, millest 103-s olid sündmused ja 47-s ei olnud. Sündmustega e-kirjade puhul kategoriseeris esimene masinõppe mudel need õigesti 88,35% juhtudest. Sündmusteta e-kirjade puhul oli selleks täpsuseks 85,11%. Eelnevaid protsente ei ole nende uurimistöös otseselt välja toodud, kuid nende kirjeldatud testandmestiku põhjal oli võimalik need arvutada. Kasutades esimest ning teist masinõppe mudelit koos leidsid nad, et nende rakendus leidis e-kirjadest sündmusi õige informatsiooniga 81.33% juhtudest. Töö lõpuks tõid nad välja, et parema täpsuse saamiseks on vaja masinõppe mudeleid treenida rohkemate e-kirjadega, kuna mõningatel juhtudel kategoriseeriti sündmustega e-kiri valesi, mistõttu jäid sellest e-kirjast sündmused leidmata. Samuti nentisid nad, et sündmuste leidmine ei toimu reaalses maailmas vaid ainult siis, kui kasutaja Androidi rakenduse avab. Seetõttu soovitasid nad serveri koodi tulevikus sedasi muuta, et see võimaldaks sündmused leida kohe, kui kasutaja e-posti tuleb uus kiri. Kuigi järgnevat puudust ei ole rakenduse arendajad nende töös maininud, siis nende loodud süsteemi arhitektuurist selgub, et nende lahendus leiab e-kirjast ainult ühe sündmuse andmed ning seega ei tööta see e-kirjade puhul, milles on mitu sündmust.

Nath jt [5] löid e-kirjadest sündmuste leidmiseks serverteenuse ning Androidi rakenduse. Serverteenus töötles e-kirju mitmes osas. Esiteks eemaldati e-kirjadest eelnevate vastuste jada ning järgnevalt märgistati igale sõnale tema nimiolem. Nende rakenduse puhul olid nimiolemid näiteks tegusõna, nimisõna, vormistatud kuupäev ning päeva nimetused. Järgnevalt eemaldati regulaaravaldisega ära tegusõnad ja nimisõnad, kuna nende rakenduse puhul ei andnud need sõnad lisainformatsiooni sündmuste leidmiseks. Viimasena leiti e-kirjadest sündmuse pealkiri, kuupäev, kellaeg ja toimumiskoht ning need andmed lisati rakenduse kasutaja kalendrisse. Rakenduse töö testimiseks kasutasid nad 120 e-kirja, millest 106-s tuvastati sündmused õigesti, ning seega leidis nende programm sündmused õigesti keskmiselt 88,3% e-kirjadest. Nende töö kokkuvõttes tõid nad välja, et nende loodud rakendus ei töötanud selliste e-kirjade puhul, milles oli mitu sündmust. Samuti tõid nad välja, et rakenduse täpsust saaks parandada masinõppe mudeliga, mis filtreerib välja e-kirjad, milles sündmusi pole.

3.2. Selgunud probleemid ja järelused

Nõrkustena selgusid Gambino jt [2] ning Wasi jt [3] töödest, et parserid ei suuda igast e-kirjast sündmusi leida, kui sündmuste vormistus erineb liiga palju e-kirjades. Paraku tuleneb see sellest, et parserid otsivad e-kirjast mustreid, mille põhjal sündmusi saab tuvastada. Selleks aga tuleb eeldada, et sündmused on e-kirjades sarnaselt vormistatud ning vormistuste erinevused on minimaalsed.

Samuti tuli eelnevatest töödest välja, et nende lahendused ei toiminud, kui ühes e-kirjas oli mitu sündmust [3-5]. Mitme sündmuse puhul tuleks teha lisaeldusi sündmuste vormistamise osas, mis on oluliselt keerulisem kui ühe sündmuse jaoks eeldusi teha. Seda seetõttu, et e-kirjas ei saa garanteerida, et iga sündmuse asukoht ning aeg oleks täpselt samas järjekorras välja toodud, nagu parseri looja eeldas. Võib juhtuda, et kõik sündmused toimuvad samal kuupäeval aga erineval kellaajal, mistõttu mainitakse e-kirjas kuupäeva ainult üks kord. Parser ei ole mõeldud teksti sisust aru saama ning seetõttu ei suuda see iga sündmusega kokku viia samat kuupäeva.

Siinse töö eesmärgi saavutamiseks ning eelnevate probleemide parandamiseks on mõttekas kasutada suurt keelemudelit, nagu näiteks GPT-4o¹⁰, Llama 3¹¹ või muud mudelit. Need keelemudelid on treenitud väga suurte ja mitmekeelsete tekstikorpuste peal, mistõttu on nad

¹⁰ Keelemudeli GPT-4o kirjeldus <https://openai.com/index/hello-gpt-4o/>. (04.12.2024)

¹¹ Keelemudeli Llama 3 kirjeldus <https://ai.meta.com/blog/meta-llama-3/>. (04.12.2024)

võimelised mitmesuguselt vormistatud tekstide sisust aru saama. See võimaldab nendel keelemudelitel vastata tekstil põhinevatele küsimustele ning leida etteantud tekstist informatsiooni. Nendest omadustest tulenevalt on parsimisega võrreldes keelemudelil mitmeid eeliseid:

- 1) keelemudel suudab tuvastada sündmusi ka siis, kui nende vormistus erineb tavapärasest;
- 2) keelemudel töötab erinevas keeles kirjutatud e-kirjade peal;
- 3) keelemudel suudab tuvastada ühest e-kirjast mitu sündmust.

Suur keelemudeli kasutab aga palju arvutusressurssi, sest sellele antava teksti töötlemiseks ja vastuse genereerimiseks peab mudel tegema palju arvutusi. Varasemates töödes [2, 4] kasutati enne sündmuste leidmist üht pisemat masinõppe mudelit, mis tuvastab, kas e-kirjas on sündmusi või mitte. Seda sama arhitektuuri on mõttekas kasutada siinse töö loodavas rakenduses suure parameetrite arvuga keelemudeli jaoks, kuna sedasi välditakse suure keelemudeli kasutamist, kui e-kirjas sündmusi ei ole. Väiksem mudel kasutab vähem arvutusressursse, kuna andmete töötlemiseks peab mudel tegema vähem arvutusi.

3.3. Veebirakenduste arhitektuurid

Kuna selles töös luuakse sündmuste leidmiseks uus rakendus, siis tuleb enne rakenduse arendamist valida, missugust arhitektuuri selle loomiseks kasutatakse. Kuigi arhitektuurivalik otseselt ei piira, milliseid funktsionaalsusi rakendusse lisada saab, siis mõjutab see nende funktsionaalsuste lisamise ja haldamise keerukust. Veebirakenduste arhitektuurid jagunevad laialdaselt kahte peamisse kategooriasse: monoliit ja mikroteenused. Selles peatükis tuuakse välja nende arhitektuuride eelised ja miinused, et hilisemates peatükkides arhitektuuri valiku põhjendusi paremini mõista.

Elgheriani ja Ahmedi [6] sõnul on monoliitne rakendus selline, mis koosneb mitmest omavahel tugevalt seotud komponendist, mis kõik asuvad sama koodibaasi sees. Mikroteenuse arhitektuuril põhinevat rakendust kirjeldavad nad kui sellist, mille komponendid on üksteisega nõrgalt seotud ning seega võivad need olenemata teiste komponentide olemasolust töötada.

Elgheriani ja Ahmed [6] võrdlevad oma töös monoliitset ning mikroteenustel põhinevat arhitektuuri. Nende sõnul on monoliitse rakenduse eeliseks see, et neid on esialgu lihtsam arendada, sest kogu rakenduse kood ning kõik rakenduse äri loogika asub ühes kohas. Samuti teeb see nende sõnul rakenduse testimise lihtsamaks, sest testimise ajal on kõik komponendid

mälusse laetud. Mikroteenuste puhul on arendajatel vaja enne testimist kontrollida, kas kõik komponendid on käivitatud ning kas nendega on võimalik testitavaal mikroteenustel suhelda. Samuti ütlevad nad, et monoliitse rakenduse puhul on seda lihtsam serverisse paigaldada ning selle seisundit jälgida, kuna kogu rakenduse kõik komponendid on ühes programmis koos ning need ei ole erinevatesse programmidesse eraldatud. Samas toovad nad välja mitmeid monoliitse rakenduse probleeme, millest üks nendest on tõrketalumatus. Nemad ütlevad, et ühe komponendi vea puhul mõjutab see tervet rakendust, mistõttu terve rakendus võib kokku joosta. Lisaks on nende sõnul ka raskem uusi tehnoloogiaid monoliitses rakenduses kasutusele võtta, kuna kõik komponendid on omavahel nii tugevalt seotud, et muudatus ühes komponendis tähendab, et teisi komponente on ka vaja muuta. See teeb nende arvates ka rakenduse koodist arusaamise raskemaks suure monoliidi puhul.

Nende sõnul on eelnevad monoliitse rakenduse probleemid väiksema mõjuga mikroteenustes, kuna iga teenus on iseseisev ning teiste teenustega nõrgalt seotud. See tähendab, et kui näiteks üks mikroteenus lakkab töötamast, siis teised mikroteenused ei jookse kokku sellepärast, kuigi nende töö võib olla natuke häiritud. Samuti ütlevad nad, et mikroteenused on võrreldes monoliitse rakendusega paremini skaleeritavad, kuna kasutajate arvu kasvamisel saab ainult ülekoormatud komponenti skaleerida ning seda pole vaja teha kogu rakendusega. Paraku on aga ka mikroteenustel oma probleemid. Nad ütlevad, et kuigi ühe mikroteenuse koodist on võrreldes monoliidiga lihtsam aru saada, siis selle asemel on terve arhitektuuri mõistmine keerulisem, kuna rakendus on jaotatud mitmeks erinevaks osaks, mis omavahel läbi mingi suhtluskanali teateid edastavad.

4. Rakenduse planeerimine

Selles peatükis tuuakse välja rakenduse loomisele eelnenud protsessid. Järgnevatel alampeatükkides kirjeldatakse rakenduse funktsionaalsusi, arendamiseks kasutatud tehnoloogiaid ja nende alternatiive ning loodava rakenduse arhitektuuri.

4.1. Funktsionaalsed ja mittefunktsionaalsed nõuded

Tuginedes eelnevate tööde probleemide analüüsile ning Tartu Ülikooli e-posti kasutusjuhule, on rakendusele loodud järgnevad funktsionaalsed nõuded (edaspidi FN_x, kus x on arv):

- 1) kasutaja saab rakendusse sisse logida Microsoft Outlooki kontoga;
- 2) kasutaja saab selle rakenduse andmebaasist enda kontoga seotud andmed (leitud sündmused, sätted, kasutajaandmed) kustutada;
- 3) kasutaja saab valida, mis kategooria sündmusi rakendus leidma peab;
- 4) kasutajal on võimalik rakendusel lubada või keelata automaatselt uutest e-kirjadest sündmuste otsimist.
- 5) kasutaja saab valida, kas ta tahab näha peetud, praegusi või tulevase sündmusi;
- 6) kasutaja saab leitud sündmuste täpsemat informatsiooni (aadress, algus- ning lõppkellaeg, sündmuse kategooriad) vaadata rakenduses;
- 7) kasutaja saab sündmusi otsida pealkirja, algusaja, lõpuaja ning sündmuse kategooria järgi;
- 8) kasutaja saab leitud sündmuste informatsiooni muuta;
- 9) kasutaja saab leitud sündmusi rakendusest kustutada;
- 10) kasutaja saab manuaalselt käivitada tema enda e-kirjade jaoks sündmuste otsimise protsessi;
- 11) kasutajal on võimalik rakendusest leitud sündmused eksportida kalendrisse.

Samuti on rakendusele loodud järgnevad mittefunktsionaalsed nõuded (edaspidi MFN_x):

- 1) rakendus suudab ühest e-kirjast sündmused leida maksimaalselt 4 tunniga, kui rakendust kasutab 250 kasutajat, kes saavad 20 e-kirja päevas;
- 2) keelemudel leiab keskmiselt 90% juhtudest e-kirjadest sündmused õigesti;
- 3) rakendus peab krüpteerima e-kirjad, mis on sündmuste otsimise järjekorras;

Eelnevate mittefunktsionaalsete nõuete mõte on määratleda omadused, mis rakendusel peavad olema, et kasutaja e-kirjadest leitaks sündmused mõistliku ajaga, ning et keelemudel väljastaks

e-kirjas olevate sündmuste kohta õiget informatsiooni. MFN1 jaoks määrati piirajaks 4 tundi, kuna sedasi jõuab rakendus sündmused leida hilisemalt 4 tundi peale tööpäeva lõppu. See peaks kasutajale andma piisavalt aega valmistuda selliste sündmuste jaoks, mille kohta teavitatakse ainult päeva võrra varem. Samuti määrati selles mittefunktsionaalses nõudes 250 kasutajat, kuna see moodustab umbes neljandiku nendest bakalaureuse- ning magistriõppekava tudengitest¹², kes 2025ndal aastal arvutiteaduse instituudis informaatikat õpivad ning arvatavasti võib selline arv tudengeid rakendust kasutada. MFN2-s määrati tuvastusprotsendiks 90%, kuna teoreetilises taustas kirjeldatud tarkvaralahenduste puhul saavutati selline täpsus sündmuste leidmisel. MFN3 seati selleks, et kui e-kirjad lekivad mingi turvarikke tõttu sündmuste otsimise järjekorrast, siis ei ole võimalik nende sisu dekrüpteerimata näha.

4.2. Kasutatavad tehnoloogiad

Selles alampeatükis tuuakse lühidalt välja rakenduses kasutatavad tehnoloogiad ning põhjused, miks just need valiti võrreldes sarnaste alternatiividega.

4.2.1 Arhitektuur

Teoreetilises taustas toodi välja monoliidi ja mikroteenuste tugevused ja nõrkused. Võttes eelnevat analüüsi arvesse, kasutatakse selle rakenduse loomiseks mikroteenuste arhitektuuri. Üks põhjus on, et kuigi rakenduses kasutatavaid tehnoloogiaid analüüsitakse enne nende kasutamist, et leida kui hästi need sobivad, võivad need tõenäoliselt tulevikus muutuda. Mikroteenuste puhul on tehnoloogia muudatusi lihtsam teha, sest üks mikroteenus on väiksem kui monoliitne rakendus, mistõttu on vaja vähem programmikoodi ümber kirjutada. Samuti ei pea ühe mikroteenuse tehnoloogia muudatuste puhul teiste mikroteenuste kasutatavaid tehnoloogiaid muutma. Lisaks nendele eelistele sobib mikroteenustel põhinev lahendus selle rakenduse jaoks, kuna rakendus on jaotatud väiksemateks teenusteks, millest on lihtsam aru saada, kui suurest monoliidi koodibaasist. See lihtsustab tulevikus uute teenusepakkujate juurde lisamist, kuna seda on võimalik teha modulaarselt, luues teenusepakkuja jaoks uue mikroteenuse ning ühendades selle teiste mikroteenustega kasutades sõnumimaaklerit.

¹² See statistika saadi Tartu Ülikooli sisseastumise statistika veebilehelt (<https://ut.ee/et/sisu/sisseastumise-statistika>). Praegu (s.t 2025ndal aastal) informaatika bakalaureuseõppes ning magistriõppes olevate tudengite arv saadi liites kokku 2022-2024 informaatika õppekavasse immatrikuleeritud õpilaste arvud.

4.2.2 Konteinerid ja nende orkersteerimine

Mikroteenuste arhitektuuri puhul on levinud rakenduste konteineriseerimine, sest see võimaldab rakendusest luua taaskasutatava konteineripildi. Konteineripildis on lisaks rakendusele ka selle töötamiseks vajalikud teegid ning sätted. See lihtsustab rakenduse paigaldamist serveris, kuna iga uue mikroteenuse lisamiseks piisab konteineripildi alla laadimisest ning selle käivitamisest konteineri tarkvaraga. Konteinerite loomiseks kasutatakse Dockerit¹³, kuna see on StackOverflowi 2024. aasta küsitluse [7] põhjal kõige populaarsem konteineriseerimistarkvara. Selles töös loodava rakenduse jaoks oleks alternatiivina saanud kasutada ka Kanikot¹⁴, kuid see lahendus on mõeldud kasutamiseks mitme kasutajaga keskkonnas, kus konteineriseerimistarkvara ei tohi omada juurkasutaja õigusi ning peab olema eraldatud teistest kasutajatest turvalisuse säilitamiseks. Kuna siinse töö jaoks luuakse konteinerid lokaalselt ühe kasutaja arvutis, on Docker parem valik.

Konteinerite orkestreerimiseks nimetatakse paljude konteinerite automaatset haldamist. Selle hulka kuulub konteinerite seisundi jälgimine, nende uuendamine ning taaskäivitamine kokkujooksmise puhul. Käsitsi paljude konteinerite haldamine oleks tülikas, mistõttu võeti kasutusele Kubernetes¹⁵, kuna see on autorile tuttav tarkvara ning sarnaselt Dockerile on ka Kubernetes oma kategoorias kõige levinum orkestreerimistarkvara. Populaarsuse tõttu on Kubernetesel pikemaajalisem tarkvaratugi ning rohkem dokumentatsiooni, mis teeb tekkivate probleemide parandamise lihtsamaks. Alternatiivselt oleks saanud ka Docker Swarm mode¹⁶ kasutada, kuna see võimaldab sarnaselt Kubernetesele mitmeid konteinereid hallata ning skaleerida vastavalt vajadusele. Siiski on Kubernetese eelis see, et sellega tuleb kaasa palju ressursidefinitsioone, mis võimaldavad konfigureerida Kubernetese klatri (ingl *cluster*) käitumist. Näiteks kasutatakse loodavas rakenduses HTTPRoute ressursidefinitsiooni, et HTTP päringuid marsruutida vastavalt teekonnale. Sedasi on võimalik päringud otspunkti `/api/events` marsruutida ühele mikroteenusele ning päringud otspunkti `/api/microsoft` teisele mikroteenusele. Docker Swarm mode puhul oleks vaja üles seadistada teise arendaja loodud konteiner, mis võimaldaks sellist marsruutimist.

¹³ <https://www.docker.com/> (27.03.2025)

¹⁴ <https://cloud.google.com/blog/products/containers-kubernetes/introducing-kaniko-build-container-images-in-kubernetes-and-google-container-builder-even-without-root-access> (28.03.2025)

¹⁵ <https://kubernetes.io/> (09.05.2025)

¹⁶ <https://docs.docker.com/engine/swarm> (04.05.2025)

4.2.3 Kasutajaliides

Siinse töö rakenduse funktsionaalsuste lihtsamaks kasutamiseks luuakse graafiline kasutajaliides. See võimaldab kasutajal rakendusse Microsofti kontoga sisse logida, leitud sündmusi näha ning sündmuste otsimisega seotud sätteid muuta. Siinse töö rakenduse kasutajaliidese loomiseks otsustati kasutada JavaScriptil põhinevat veebilehe loomise raamistikku. Seda seetõttu, et need raamistikud võimaldavad muudatusi teha kliendi veebilehitsejas nii, et tervet lehte pole vaja uuesti laadida. Näiteks uute sündmuste laadimisel või ühe sündmuse nime uuendamisel saadab veebileht päringu serverisse, mille tulemusena JavaScripti raamistik muudab lokaalselt sündmuste nimekirja. Seda on võimalik ka ilma raamistikuta teha, aga selleks oleks vaja luua rohkem aluskoodi, et saavutada raamistikuga sarnane funktsionaalsus.

Cinoviči ja Puntti [8] sõnul on JavaScripti raamistikke palju, kuid kõige populaarsemad neist on React¹⁷, Angular¹⁸ ning Vue¹⁹. Siinse töö veebilehe loomiseks otsustati kasutada Reacti. Kuigi kõik eelnevad raamistikud võimaldavad luua veebilehte, mille sisu uuendamiseks ei pea tervet lehte serverist uuesti pärima, siis võrreldes teiste raamistikuga on Reactil Cinovič ja Puntti [8] sõnul kõige suurem kasutustugi. See tähendab, et Reacti jaoks leidub kõige rohkem lisateeki, mis lihtsustavad veebilehe arendamist. Samuti on kasutuse rohkuse tõttu rohkem abimaterjale, mis aitavad arendamisel tekkivaid probleeme lihtsamini lahendada.

4.2.4 Tagasüsteemi teenused

Tagasüsteemi teenuste arendamiseks kasutatakse programmeerimiskeelele Python loodud veebiserveri arendamise teeki FastAPI²⁰. Selle omadused on automaatne dokumentatsiooni loomine igale HTTP otspunktile, Pythoni andmetüüpide sisu korrektsuse kontrollimine ning asünkroonse koodi kasutamise võimalus [9, 10]. Selle teegi kasutamine lihtsustab HTTP otspunktide dokumenteerimist, kuna selleks ei ole vaja luua eraldi faili, kus on HTTP otspunkti dokumentatsioon koos päringu argumentide kirjeldustega. Piisab vaid sellest, kui programmikoodis muutujatele kommentaarid lisada ning FastAPI loob nende põhjal automaatselt dokumentatsiooni, mis lihtsustab seda protsessi. Samuti on FastAPI asünkroonse koodi tugi siinse töö rakenduse jaoks kasulik, sest mikroteenused ootavad peamiselt sisendi-

¹⁷ <https://react.dev/> (27.03.2025)

¹⁸ <https://angular.dev/> (27.03.2025)

¹⁹ <https://vuejs.org/> (27.03.2025)

²⁰ <https://fastapi.tiangolo.com/>. (28.03.2025)

väljundiga seotud tegevuste taga, mis tähendab, et arvutusressurssi ootamise ajal ei kasutata. FastAPI dokumentatsiooni põhjal [10] laseb asünkroonne kood serverteenusel ootamise ajal tegeleda teise tegumi täitmisega, mis võimaldab ühel ja samal teenusel töödelda rohkem päringuid.

FastAPI asemel oleks saanud kasutada ka Flaski²¹, millega on samuti võimalik luua veebiservereid. Flask on loodud aastal 2010 ning FastAPI aastal 2018, mistõttu on Flask vanem arendusteeik ning seetõttu on ka sellel suurem kasutajabaas, rohkem kasutustuge ning palju abiteeke Flaski jaoks. Samas aga puuduvad sellised teegid Flaskil, mis suudaks sama mugavalt ja automaatselt genereerida dokumentatsiooni HTTP otspunktidele nagu FastAPI. Üks selline teek on flasgger, mis seda suudab teha, kuid selleks on arendajal ise vaja kirjutada OpenAPI dokumentatsioon YAML formaadis, mistõttu ei ole see täielikult automaatne. Flask võimaldab sarnaselt FastAPI-le ka asünkroonset koodi kirjutada, kuid selleks loob see iga päringu jaoks uue lõime, mille peal on võimalik asünkroonset koodi käivitada [11]. See ei ole nii ressursisäästlik, sest asünkroonsuse mõte on ära kasutada ainult ühte lõime võimalikult paljude paralleelsete (ingl *concurrent*) tegumite jaoks.

4.2.5 Sõnumimaakler

Mikroteenuste omavaheliseks suhtluseks otsustati kasutada sõnumimaaklerit, sest see võimaldab teenustel omavahel suhelda asünkroonselt. See suhtlusviis sobib siinse rakenduse mikroteenustele paremini, kuna teenused ei pruugi otsekohe saada mingit teadet töödelda, mistõttu peaks sünkroonse suhtluse puhul sõnumit saatev teenus ise haldama sõnumite saatmise korduskatseid ning see muudaks rakendust keerulisemaks. Sõnumimaakleri kasutamise eelis on see, et sõnumimaakler hoiab saadetud sõnumite kohta nende olekut ning kui sõnumi töötlemisel tekib mikroteenuses viga, siis saadab sõnumimaakler teate mikroteenusele uuesti ning see garanteerib, et sõnum jääb alles niikaua kuni mikroteenus selle ilma vigadeta ära töötleb. Samuti hoiab sõnumimaakler saadetud teateid niikaua järjekorras, kuni ei ole mikroteenust, mis neid läbi töötleks. See ka garanteerib, et saadetud sõnumid jõuaks mikroteenusele kohale, kui see tööle pannakse.

²¹ <https://flask.palletsprojects.com/en/stable/> (28.03.2025)

Sharwari ja Somnya [12] sõnul on kolm kõige populaarsemat teadete edastamise süsteemi Apache Kafka²², RabbitMQ²³ ja NATS²⁴. Nende töö sisust selgub, et kõik eelnevad tehnoloogiad võimaldavad teateid edastada kindlate garantiidega, nagu maksimaalselt ühe korra teate edastamine ning vähemalt ühe korra teate edastamine. Apache Kafka on nende tehnoloogiate seast ainuke, mis võimaldab garanteerida täpselt ühekordset teatedastust. Samas aga toovad nad välja, et RabbitMQ võimaldab kasutada transaktsioone, millega saab mitu sõnumit korraga saata ning juhul, kui üks neist kohale ei jõua, tühistada ka ülejäänud sõnumite saatmise. Seetõttu otsustati kasutada RabbitMQ-d, sest selle töö rakenduse puhul on tähtis, et kõik e-kirjad jõuaksid sõnumijärjekorda kohale. Vastasel juhul võivad mõned sündmused automaatselt kalendrisse märkimata jääda.

4.2.6 Andmebaas

Siinse rakenduse jaoks on vaja kasutada andmebaasi, et talletada kasutajasätteid ning leitud sündmusi. Selleks otsustati teha valik relatsiooniliste andmebaaside seast, kuna talletatavad andmed on võimalik hoida struktureeritud kujul, mis sobib hästi relatsiooniliste andmebaaside jaoks.

Siinse töö jaoks uuriti PostgreSQL²⁵ ja MySQL²⁶ andmebaase, kuna need on StackOverflowi 2024. aasta küsitluse [7] põhjal kõige levinumad, mis tähendab, et nende erinevuste ja sarnasuste kohta on ka kõige rohkem informatsiooni. Amazoni artikli [13] põhjal on MySQL parem andmebaas siis, kui andmeid on rohkem vaja lugeda kui kirjutada. Samuti on selles artiklis välja toodud, et MySQL andmebaas mälusäästlikum kui PostgreSQL. Kuna andmebaasi kasutatakse selles rakenduses rohkem andmete lugemiseks kui kirjutamiseks ning andmebaasi serveril on vähe mälu, otsustati nende eeliste tõttu kasutada MySQL andmebaasi. Lisaks sellele on MySQLi sisse ehitatud tööde ajastaja, mis võimaldab iga kindla aja tagant funktsioone käivitada. See on kasulik selle rakenduse jaoks, kuna seda saab ära kasutada, et aegunud andmeid perioodiliselt andmebaasist eemaldada. PostgreSQL andmebaasis on võimalik sama funktsionaalsus saavutada installeerides lisamooduli pg_cron. See nõuab aga mooduli üles seadistamist ja konfigureerimist, samas kui MySQLis on tööde ajastaja käivitamiseks vaja muuta ainult ühte muutujat.

²² <https://kafka.apache.org/> (01.04.2025)

²³ <https://www.rabbitmq.com/> (01.04.2025)

²⁴ <https://nats.io/> (01.04.2025)

²⁵ <https://www.postgresql.org/> (28.03.2025)

²⁶ <https://www.mysql.com/> (28.03.2025)

4.2.7 Keelemudel

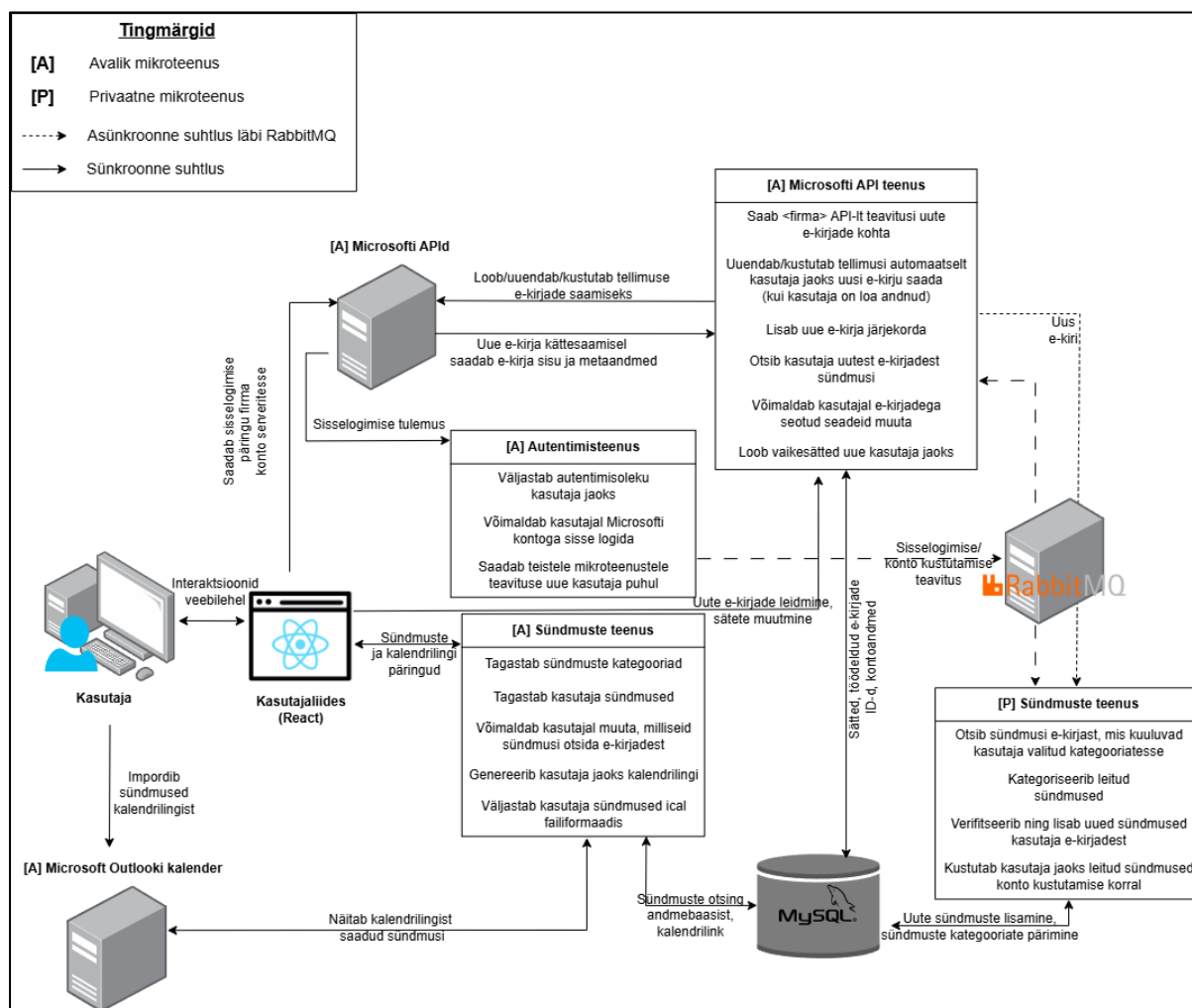
E-kirjadest sündmuste leidmiseks otsustati siinses töös kasutada keelemudelit. Kuna keelemudeleid on palju, otsustati valik teha HuggingFace'i keelemudelite edetabeli²⁷ põhjal, milles vaadeldi keelemudelite tulemusi IFEval [14] ja BBH [15] andmestike peal. Selles töös jäeti võrdlusest välja omandtarkvara keelemudelid, sest nende puhul on vaja e-kirjad saata kolmanda osapoole serveritesse, mis hoiustavad neid andmeid kuni kuu aega. Kuna e-kirjades võib olla tundlikke andmeid, ei sobi selline lahendus siinse töö jaoks. Lisaks jäeti välja mitteametlikud keelemudeli versioonid ja rohkem kui nelja miljardi parameetriga mudelid, sest selles uurimistöös kasutataval serveril ei ole piisavalt arvutusressurssi, et suuremaid keelemudeleid mõistliku ajaga ühest e-kirjast sündmuste leidmiseks kasutada. Kuna siinses töös piirduakse pisema keelemudeliga, siis ei kasutatud selles rakenduses ühte lisa masinõppe mudelit, mis tuvastaks, kas e-kirjades on sündmusi või mitte.

Nende nõuete põhjal valiti sündmuste leidmiseks keelemudel Llama-3.2-3B-Instruct, sest see keelemudel andis võrreldes teiste sarnase suurusega mudelitega IFEval [14] andmestike peal häid tulemusi. Kuigi BBH [15] andmestiku tulemused olid selle keelemudeli puhul madalad, on sellel mitmeid teisi eeliseid, mis põhjendavad selle keelemudeli kasutamist. Selle keelemudeli puhul on võimalik kasutada süsteemiviipa, mis vähendab võimalust, et e-kirja sisu mõjutab keelemudeli tööd selliselt, et sündmuste leidmise asemel täidab keelemudel mingit muud ülesannet. Samuti on seda keelemudelit treenitud kaheksa erineva keele peal, mis suurendab selle keelemudeli võimet leida sündmusi mitmes erinevas keeles kirjutatud e-kirjadest.

²⁷ HuggingFace'is avaldatud ja selle töö nõuetele vastavalt filtreeritud keelemudelite tulemused erinevate testandmestike peal on järgneval lingil
https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard#/?params=1%2C4&official=true
(21.12.2024)

4.3. Rakenduse mikroteenuste struktuur

Rakendus koosneb neljast peamisest mikroteenusest: avalik autentimisteenus, privaatne sündmuste teenus, avalik sündmuste teenus ning avalik Microsofti API teenus (joonis 1). Avalikud teenused on kättesaadavad igale rakenduse kasutajale, samas kui privaatsete teenused on ainult ligipääsetavad rakendusele. Selle mõtte on ära peita funktsionaalsused, mis on mõeldud ainult rakendusesiseseks kasutamiseks, et tavakasutaja ei saaks kuidagi juurdepääsu nendele. Kõik need teenused koos katavad rakenduse erinevaid funktsionaalsusi, mistõttu ühe teenuse rikke tõttu ei lõpeta kogu rakendus töötamist, vaid ainult üks osa sellest.



Joonis 1. Loodava rakenduse arhitektuuridiagramm

Joonisel 1 on iga teenuse puhul välja toodud, mis funktsionaalsusi see hõlmab ning milliste teenuste või andmebaasidega suheldakse. Samuti on noolte peal kirjeldatud, mis tegevuse tõttu suhtlus mikroteenusega algas. Katkendlikud jooned näitavad, et mikroteenusele saadetakse teateid läbi RabbitMQ sõnumimaakleri. Nende andmete töötlemine toimub asünkroonselt ning

teenus ei saa otsekoheselt vastust saadetud teatele. Samuti näitavad katkendlikud jooned, et saadetud teadet ei pea teised mikroteenused otsekohe töötleva. Kuigi arhitektuuridiagrammil on näidatud, et kõik teenused kasutavad ühte tsentraalset MySQL andmebaasi, siis iga mikroteenus kasutab MySQL andmebaasis ainult talle määratud tabeleid. Seetõttu on tulevikus võimalik iga mikroteenuse jaoks eraldi MySQL andmebaasi kasutada, sest üks mikroteenus pärib ning salvestab andmeid ainult selle sama mikroteenuse jaoks mõeldud tabelitest.

Samuti on joonisel 1 välja toodud ka kaks lisateenust – Microsofti APId ja Microsoft Outlooki kalender. Need võimaldavad siinse bakalaureusetöö rakenduse mikroteenustel (avalik sündmuste teenus, Microsofti API teenus) suhelda Microsofti API-dega, mis võimaldavad selle rakenduse kasutajal Microsofti kontoga sisse logida, uutest e-kirjadest sündmusi otsida ning neid kasutaja kalendrisse automaatselt eksportida. Tegelikult võib Microsoft Outlooki kalendri asemel kasutada ka teist kalendrit nagu Google'i kalender, kuna see ka toetab sündmuste importimist URL aadressilt. Kuigi arhitektuuridiagrammis (joonis 1) on välja toodud Microsoftiga seotud teenused, siis võeti mikroteenuste disainimisel arvesse ka seda, et tulevikus oleks võimalikult lihtne uusi teenusepakkujaid lisada nagu näiteks Google'i e-posti kontosid.

5. Keelemudeli rakendamine ja peenhäälestamine

Siinses töös loodava rakenduse puhul on suure tähtsusega keelemudel, sest see võimaldab e-kirjadest sündmusi otsida ning neid kindla struktuuriga väljastada. Selles peatükis tuuakse esiteks välja kirjeldus tark- ja riistvara kohta, mida kasutati keelemudeli testimiseks ja treenimiseks. Peale seda kirjeldatakse, kuidas rakendus keelemudelile e-kirjad lugemiseks annab ning mis formaadis see leitud sündmused väljastab. Seejärel kirjeldatakse, kuidas peenhäälestamata keelemudelit Llama-3.2-3B-Instruct testiti sündmuste leidmiseks ning kuidas seda keelemudelit peenhäälestati, et keelemudel toimiks paremini sündmuste leidmisel e-kirjast.

5.1. Kasutatav riistvara ning tarkvara

Keelemudelit testitakse ning treenitakse arvutis, millel on Geforce RTX 3090 graafikakaart, kuna graafikakaart võimaldab teha paralleelarvutusi, mis kiirendab keelemudeli testimist ja treenimist võrreldes protsessoriga. Näiteks kulub keelemudelil Llama-3.2-3B-Instruct e-kirjadest sündmuste leidmiseks keskmiselt 5-10 sekundit, kui see keelemudel jooksub graafikakaarti peal. Kui keelemudeli testimist aga teha Intel i7-10700KF protsessoril, siis kulub keelemudelil sündmuste leidmiseks keskmiselt 3-6 minutit.

Keelemudeli testimiseks ning selle integreerimiseks loodavasse rakendusse kasutatakse teeki llama-cpp-python (versioon 0.3.7), mis toob programmeerimiskeeles C++ kirjutatud keelemudelite kasutamise teegi llama.cpp funktsionaalsused Pythonisse. Llama.cpp võimaldab keelemudelile lihtsasti viia anda ning keelemudelilt tekstilist väljundit saada. Samuti on võimalik selle teegiga kasutada grammatikat, mis piirab keelemudeli väljundi kindlale kujule. Kuna nendest funktsionaalsustest piisab keelemudeli testimiseks ning rakenduses kasutamiseks, siis otsustati just Pythoni teeki llama-cpp-python kasutada.

Keelemudeli peenhäälestamiseks kasutatakse aga kombinatsiooni kolmest teegist: unsloth (versioon 2025.3.17), TRL (versioon 0.15.2) ning transformers (versioon 4.49.0). Need on enim levinumad teegid just keelemudelite peenhäälestamiseks ning muude masinõppe mudelite jaoks kasutatakse rohkem teeki pytorch või tensorflow. HuggingFace'i dokumentatsiooni [16] põhjal on transformers Pythoni teek, mis sisaldab mitmeid eeltreenitud masinõppe mudeleid (keelemudelid, arvutinägemise mudeleid, helitöötusega seotud mudeleid), mida on võimalik Pythoni programmis kasutada või oma enda andmetega treenida. Teegid TRL ning unsloth laiendavad teegi transformers funktsionaalsusi, lisades uusi treenimistehnikaid

keelemudelite peenhäälestamiseks. Unslothi eelis tuleneb aga sellest, et see võimaldab keelemudelit trenides graafikakaarti mälu kasutust vähendada, et lokaalselt oleks võimalik suuri keelemudeleid tavakasutuseks mõeldud graafikakaartide peal peenhäälestada.

5.2. Keelemudeli viipade ja väljundite formaat

Et paremini mõista, kuidas keelemudelile e-kirju sündmuste otsimiseks antakse, tuuakse selles peatükis välja keelemudelile antava viiba ning väljundi formaat. Neid formaate kasutatakse nii loodavas rakenduses kui ka keelemudeli testimisel ja treenimisel. Keelemudelile antav viip koosneb kahest osast: süsteemiviibast ning tavatekstist, milleks on e-kirja sisu koos mõningate metaandmetega. Süsteemiviibas on instruktsioonid keelemudelile, milles kirjeldatakse, mis andmeid keelemudel e-kirjast otsima peab ning mida peaks keelemudel tegema, kui osad andmed sündmuse kohta puuduvad. Süsteemiviip on välja toodud töö teises lisas. Näide tavateksti vormistusest on välja toodud joonisel 2.

```
Email reader: kirja.lugeja@ut.ee
Email sender: kirja.edastaja@ut.ee
Email from: saatja@ut.ee
Email recipients: lugeja1@ut.ee, lugeja2@ut.ee, ..., lugejaX@ut.ee
Send time: ISO-8601 formaadis saatmiskuupäev
Title: pealkiri
Content: e-kirja sisu
```

Joonis 2. Keelemudelile antav e-kiri koos metaandmetega

Joonisel 2 näidatud formaadis on välja toodud e-kirja saatmiskuupäev, et keelemudelil oleks võimalik sõnade ning fraaside (näiteks „homme“, „selle nädala kolmapäeval“) põhjal mingi sündmuse toimumiskuupäev ära tuvastada. Samuti on eraldi välja toodud ka e-kirja lugeja ning e-kirja kättesaajad, kuna mõningatel juhtudel võivad sündmuse toimumisajad sõltuda olenevalt e-kirja lugejast. Näiteks suulise vaheksami puhul võib e-kirjas olla tabel, milles on iga õpilase jaoks eraldi välja toodud kellaaeg, millal ta eksamile peab minema. Sellisel juhul on keelemudelil vaja teada, kes e-kirja loeb, et sündmuste leidmisel panna vaheksamile õige toimumiskellaaeg.

Järgnevalt kasutatakse teegi llama-cpp-python funktsiooni `create_chat_completion`, et argumentidega `messages` anda keelemudelile süsteemiviip ning e-kirja sisu. Näide sellest koodist on välja toodud joonisel 3.

```
chat_output = self.model.create_chat_completion(
    messages = [
        { # 1. osa - süsteemiviip
          "role": "system",
          "content": prompt
        },
        { # 2. osa - e-kirja sisu metaandmetega
          "role": "user",
          "content": prepared_content
        },
    ],
    grammar=prompt_config.get_parse_output_grammar(tags),
    max_tokens=self.MAX_GENERATED_TOKENS,
    temperature=0.6
)
```

Joonis 3. Näide keelemudeli kasutamisest teegiga llama-cpp-python

Eelnevas näites kasutatakse selles funktsioonis veel paar argumenti, mis muudavad keelemudeli tööd. Argumentidega `temperature` määratakse, kui juhuslik võib olla keelemudeli genereeritud väljund. Kui selle argumenti väärtuseks panna 0, siis iga kord genereerib keelemudel sama sisendi puhul täpselt sama väljundi. Siinse rakenduse puhul kasutati väärtust 0.6, et keelemudelile jätta natukene juhuslikkust. Argument `max_tokens` määrab, kui pikk võib maksimaalselt olla keelemudeli genereeritud väljund. Rakenduses kasutati seda argumenti, sest keelemudelil oli probleem, kus see korduvalt genereeris täpselt sama teksti ning tegi seda niikaua, kuni rakenduse sündmuste otsimise mikroteenus kokku jooksis. Maksimaalse väljundi pikkuse määramisega välditakse seda probleemi. Argumenti `max_tokens` väärtuseks otsustati määrata 6000, kuna keskmiselt kasutab üks sündmus keelemudeli väljundis 100-200 tokenit ning tõenäoliselt esineb väga harva selliseid e-kirju, milles on üle 30 sündmuse. Seetõttu peaks see piirang olema piisav, et keelemudelil võimaldada väga suurest osast e-kirjadest kõik sündmused leida.

Keelemudeli väljundi formaadi piiramiseks kasutatakse selles funktsioonis argumenti `grammar`. See võtab argumendiks ühe Pythoni sõnastiku, mis kasutab JSON Schema²⁸ tähistust, et piirata keelemudeli väljund kindlale kujule. Siinse töö rakenduse puhul lubab see keelemudelil genereerida ainult ühe JSON järjendi, mille sees on JSON objektid. Nende JSON objektide kuju on välja toodud joonisel 4.

```
{
  "event_name": "Sündmuse pealkiri",
  "start_date": "ISO-8601 formaadis kuupäev" VÕI null,
  "end_date": "ISO-8601 formaadis kuupäev ",
  "country": "Riik, kus sündmus toimub" VÕI "",
  "city": "Linn, kus sündmus toimub" VÕI "",
  "address": "Toimumiskoha aadress" VÕI "",
  "room": "Ruum, kus sündmus toimub" VÕI "",
  "tags": ["Nimekiri sündmusega seotud kategooriatest"]
}
```

Joonis 4. Ühte sündmust sisaldava JSON objekti kuju

Joonisel 4 näidatud JSON objektis võivad võtmete `country`, `city`, `address` ning `room` väärtused olla ka tühisõned, kui keelemudel ei leia e-kirjast informatsiooni sündmuse asukoha kohta. Samuti võib `start_date` väärtus olla null, sest mõningad sündmused on tähtjad ning nende puhul ei ole võimalik algusaega märgistada.

5.3. Peenhäälestamata keelemudeli testimine

Selles peatükis kirjeldatakse keelemudeli testimiseks loodud andmestikku, testimismetoodikat ning testimise tulemusi.

5.3.1 Testandmestiku kirjeldus

Peenhäälestamata keelemudeli testimiseks loodi andmestik, mis koosnes töö autori e-posti tulnud Tartu Ülikooliga seotud kirjadest ning Enroni töötajate e-kirjadest²⁹. Kuigi rakendus on

²⁸ Lühike kirjeldus JSON Schema kohta on järgneval lingil <https://json-schema.org/overview/what-is-jsonschema> (05.05.2025)

²⁹ Enroni pankrotti mineku tõttu avalikustas Föderaalne energeetika reguleerimiskomisjon (FERC) töötajate e-kirjad, mille põhjal loodi nendest e-kirjadest andmestik. Rohkem informatsiooni leiab veebilehelt <https://www.cs.cmu.edu/~enron/> (29.03.2025)

loodud kasutamiseks Tartu Ülikoolis, siis valiti testandmestikku ka Enroni e-kirju, et keelemudelit testida ka Tartu Ülikooliga mitteseotud e-kirjade sisuga. Samuti võeti testandmestikku nii eesti- kui ka ingliskeelseid e-kirju, sest Tartu Ülikoolis saadeti sündmustega seotud e-kirju peamiselt nendes keeltes. Mõlemal juhul jaotati e-kirjad kahte kategooriasse: sündmustega e-kirjad ning sündmusteta e-kirjad, kusjuures seda jaotust prooviti võimalikult võrdsena hoida. Testandmete jaotus on täpsemalt välja toodud tabelis 1. Andmestikku kogutud Tartu Ülikooli e-kirjade puhul peideti ära isikute nimed, nende telefoninumbrid ja aadressid. Seda aga ei tehtud Enroni e-kirjade jaoks, kuna need on avalikult kättesaadavad, mistõttu oleks võimalik ikkagi e-kirjaga seotud isikud üles leida.

Tabel 1. Testandmestikku kogutud e-kirjade jaotus

	Eesti keeles, sündmusteta	Eesti keeles, sündmustega	Inglise keeles, sündmusteta	Inglise keeles, sündmustega
Enron	0	0	3	3
Tartu Ülikool	6	9	4	5

Iga testandmestikku valitud e-kirja puhul loodi tekstifail, mis jagati mitmeks sektsiooniks, kus iga sektsiooni selles failis eraldas sõne „!<-->!“. Joonisel 5 on välja toodud näide sellest, kuidas testandmestikus olevad tekstifailid vormistati.

<p>Sektsioon 1 - kommentaar, mis kirjeldab e-kirja omadusi !<-->!</p> <p>Sektsioon 2 - semikooloniga eraldatud sündmuste kategooriad !<-->!</p> <p>Sektsioon 3 - e-kirja lugeja e-post !<-->!</p> <p>Sektsioon 4 - JSON või IMF formaadis e-kiri !<-->!</p> <p>Sektsioon 5 - JSON formaadis oodatud keelemudeli vastus</p>
--

Joonis 5. Testandmestiku tekstifaili sisu vormistus

Esimest seksiooni tekstifailist keelemudeli testimisel ei kasutata, vaid see on ainult lisainfo andmiseks testandmestiku vaatlejale. Kommentaaris on kirjas, miks e-kiri valiti testandmete hulka ning mis on selle omadused. Teises seksioonis on sündmuste kategooriad, mille põhjal keelemudel e-kirjast leitavaid sündmusi kategoriseerima peab. Selles seksioonis on e-kirja sündmustega seotud kategooriaid kui ka mitteseotud kategooriaid, et keelemudel õpiks sündmustele määrama ainult nendega seotud kategooriaid. Need kategooriad lisatakse süsteemiviipa. Mõningatel juhtudel on e-kirjal mitu isikut, kellele see saadetakse, mistõttu ei pruugi keelemudel teada, kas leitavad sündmused on olulised ainult kellelegi teisele või kõigile. Seetõttu on testandmete kolmandas seksioonis välja toodud selle isiku e-post, kes e-kirja loeb. Neljandas seksioonis on e-kirja andmed, mida keelemudel peab töötleva. Need e-kirja andmed teisendatakse tekstilisse formaati enne keelemudelile andmist. Viiendas seksioonis on sündmused, mida keelemudel e-kirjast leidma peab. Need sündmused kasutavad eelnevas peatükis kirjeldatud väljundi formaati, milleks oli JSON objekte sisaldav JSON järjend. Viienda seksiooni andmeid keelemudelile ette ei anta, vaid need on selleks, et hinnata, kui hästi keelemudel suutis e-kirjast sündmusi leida.

5.3.2 Testimismetoodika

Keelemudeli testimine koosnes kahest etapist – keelemudeli väljundite genereerimisest ning nende hindamisest. Väljundite genereerimiseks loodi programm „generate_responses.py“³⁰, mis laeb testandmestiku failid mällu, uuendab süsteemiviipa testandmestiku failis defineeritud kategooriatega ning peale seda laseb keelemudelil leida e-kirjas olevad sündmused. Programm kordab eelnevat tegevust ühe e-kirja jaoks kolm korda ning salvestab tulemused tekstifaili, mille vormistus on välja toodud joonisel 6.

```
Seksioon 1 - JSON või IMF formaadis e-kiri
!<-->!
Seksioon 2 - JSON järjendis sündmuste kategooriad süsteemiviiba jaoks
!<-->!
Seksioon 3 - JSON formaadis oodatud keelemudeli väljund
!<-->!
```

³⁰https://github.com/madiskoivopuu/EventsOrganiser/blob/main/misc/app_testing/llm_accuracy/generate_responses.py (20.04.2025)

Sektsioon 4 kuni 6 – keelemudeli genereeritud JSON järjend, milles on JSON objektid, mis sisaldavad sündmustega seotud infot

Joonis 6. Tekstifaili formaat, kuhu on salvestatud keelemudeli leitud sündmused

Sarnaselt testandmestike failidele kasutatakse sektsioonide eraldamiseks sõne „!<=>!“. Ühest e-kirjast lasti keelemudelil sündmusi leida kolm korda kahel põhjusel. Esiteks on testandmestikus ainult 30 e-kirja, mis ei ole piisav, et keelemudeli täpsust sündmuste leidmisel hästi hinnata. Teiseks seadistati keelemudel selliselt, et sellel on võimalik väljundeid natukene juhuslikult genereerida. See tähendab, et ühe e-kirja puhul võib see sündmusi väljastada natukene erinevatel viisidel, mistõttu on vaja testida, et see juhuslikkus ei mõjutaks keelemudeli tööd liiga negatiivselt.

Järgnevalt loodi keelemudeli genereeritud väljundite hindamiseks programm „grade_responses.py“³¹. Näidis selle programmi kasutamisest on välja toodud joonisel 7. See programm toimis nii, et esiteks laadis see kõik tulemuste tekstifailid mällu. Seejärel hakkas see igast tekstifailist järk-järgult alates sektsioonist 4 keelemudeli väljundeid ekraanile väljastama. Kuna ühe sektsiooni keelemudeli väljundis võib olla mitu sündmust, siis näidati ekraanil korraga ainult ühte genereeritud sündmust. Samuti paralleelselt sellega näidati ka õiget vastust, mille põhjal oli võimalik keelemudeli genereeritud sündmusele hinnang anda. Seejärel hinnati töö kolmandas lisas välja toodud hindamisreeglite põhjal ühe sündmuse JSON objekti võtmete väärtusi vahemikus 0,0 kuni 1,0. Nendeks võtmeteks olid `event_name`, `start_date`, `end_date`, `country`, `city`, `address`, `room` ja `tags`. Neid väärtusi aga ei hinnatud selliste sündmuste puhul, mis keelemudel genereeris, kuid neid tegelikult e-kirjas ei olnud. Samuti ei hinnatud neid väärtusi, kui keelemudel e-kirjast sündmusi ei leidnud, sest sellisel juhul ei olnud väljundis ühtegi sündmuse JSON objekti, mille võtme väärtuseid hinnata. Peale ühe sektsiooni hindamist küsiti käsureal, mitu protsenti sündmustest keelemudel õigesti leidis. See protsent leiti, jagades õigesti leitud sündmuste arvu kõikide sündmuste arvuga, mis e-kirjas oli. Mõnel lihtsamal juhul määras programm hinnangu automaatselt, näiteks kui keelemudel ei leidnud sündmusi ning e-kirjas ei olnudki sündmusi, siis andis programm automaatselt hinnangu 1,0. Kui aga keelemudel leidis sündmusi, kuigi neid ei olnud e-kirjas, siis määras programm automaatselt sündmuste leidmise hindeks 0,0.

³¹https://github.com/madiskoivopuu/EventsOrganiser/blob/main/misc/app_testing/llm_accuracy/grade_responses.py (10.05.2025)

```

-----EXPECTED RESULT START-----
[
  {
    "event_name": "[RESERVATION NEEDED] Enron's financial wellness works
hop",
    "start_date": "2000-10-24T11:30:00",
    "end_date": "2000-10-24T12:30:00",
    "country": "United States",
    "city": "Houston",
    "address": "400 Dallas Street",
    "room": "LaSalle A room",
    "tags": [
      "Finance",
      "Workshop",
      "Enron"
    ]
  }
]
-----EXPECTED RESULT END-----
-----GENERATED EVENT (1/1) START-----
{
  "event_name": "Enron financial wellness workshop (re-playing 2001)",
  "start_date": null,
  "end_date": "2000-10-24T23:59:00",
  "country": "",
  "city": "",
  "address": "",
  "room": "",
  "tags": [
    "Workshop",
    "Finance"
  ]
}
-----GENERATED EVENT (1/1) END-----
Should this event be graded (y/n)? y
Grade the accuracy of 'event_name' (0.0 -> 1.0): 0.8
Grade the accuracy of 'start_date' (0.0 -> 1.0): 0
Grade the accuracy of 'end_date' (0.0 -> 1.0): 0.6666666666666666
Grade the accuracy of 'country' (0.0 -> 1.0): 0
Grade the accuracy of 'city' (0.0 -> 1.0): 0
Grade the accuracy of 'address' (0.0 -> 1.0): 0
Grade the accuracy of 'room' (0.0 -> 1.0): 0
Grade the accuracy of 'tags' (0.0 -> 1.0): 0.6666666666666666
Grade the accuracy of LLM finding correct events (0.0 -> 1.0, correct ÷ total): 1.0

```

Joonis 7. Näide hindamisprogrammi väljundist käsureal

5.3.3 Tulemused

Järgnevalt tuuakse välja failiga „grade_responses.py“ hinnatud tulemused. Tabelis 2 tuuakse välja keskmised tulemused selle kohta, mitu protsenti sündmustest keelemudel õigesti tuvastas. Need tulemused on jaotatud kahte kategooriasse: sündmusteta ning sündmustega e-kirjad. Sündmusteta e-kirjade puhul mõeldakse sündmuste mittetuvastamise täpsust. Tabelis 3 on välja toodud sündmuste JSON objektide võtmete keskmised hinnangud.

Tabel 2. Keelemudeli keskmised tulemused sündmuste õigesti tuvastamisel

	Õigesti tuvastatud sündmused (%)
Sündmusteta e-kirjad	0
Sündmustega e-kirjad	84,49

Tabel 3. Keelemudeli keskmised tulemused iga sündmuse detaili kohta

Hinnatud sündmuse objekti välja nimi	Keskmine hinnang (%)
event_name	40,27
start_date	31,17
end_date	38,21
country	38,21
city	41,63
address	32,76
room	38,62
tags	33,65

Tabelist 2 selgub, et sündmusteta e-kirjade puhul väljastas keelemudel iga kord sündmuse, kuigi neid tegelikult e-kirjas ei olnud. Näiteks oli üks testandmestiku sündmusteta e-kiri selline, mis teavitas e-kirja lugejat erialastipendiumi mittemääramisest. Selle e-kirja puhul oli tegemist teavitusega, mistõttu ei oleks keelemudel tohtinud sellest ühtegi sündmust leida. Siiski väljastas keelemudel selle e-kirja puhul kahel juhul sündmuse pealkirjaga „Erialastipendiumi (IT õppekavad) mittemääramine“ ning kolmandal juhul oli sündmuse pealkirjaks „Deadline for IT Scholarship Application“, mis oli kõige eksitavam pealkiri kõigi kolme genereeritud väljundi seast. Seega ei saa nende tulemuste põhjal peenhäälestamata rakenduses kasutada, kuna see lisaks iga sündmusteta e-kirja puhul kasutaja kalendrisse sündmuse, mida tegelikult ei ole olemas. See tekitab rakenduse kasutaja kalendris liiga palju segadust, mistõttu ei oleks tal võimalik lihtsalt eristada, millised sündmused tegelikult toimuvad ning millised keelemudel välja mõtles.

Samas on sündmustega e-kirjade puhul tabel 2 põhjal keelemudeli tulemus parem. Paraku aga selgub tabel 3 andmetest, et kuigi keelemudel väljastab õigeid sündmuseid (s.t sündmuseid, mis e-kirjas tegelikult välja toodud on), siis on nendega seotud info tihti vale. Näiteks on sündmuste nime keskmine hinnang 40,36%, mis tähendab, et tihti oli sündmuse nimi ebatäpne ning ei andnud sündmuse sisu lühidalt edasi. Kategooriate hulgast on täpsus samuti madal 33,52%, mistõttu ei kategoriseeri keelemudel sündmuseid õigesti. Seetõttu võib keelemudel mingisse kategooriasse mittekuuluvad sündmused just sinna lisada, mis vähendab sündmuste kategoriseerimise tõhusust.

Nende puuduste tõttu ei saa peenhäälestamata keelemudelit rakenduses kasutada, sest selliste probleemidega ei suuda siinses töös loodav rakendus sündmusi piisavalt täpse info e- kirjadest leida. Seetõttu on vaja keelemudelit peenhäälestada uue andmestiku peal, mis keskendub sündmuste leidmisele e- kirjadest.

5.4. Keelemudeli peenhäälestamine

Testimisest selgus, et peenhäälestamata keelemudel ei suutnud piisavalt hea täpsusega sündmusi leida, sest keelemudel tegi sündmuste leidmisel liiga palju vigu. Selles peatükis kirjeldatakse keelemudeli peenhäälestamiseks loodud treeningandmestikku, nende andmete eeltöötlust ning peenhäälestamise protsessi. Kõikides selle peatüki alampeatükkides viidatud koodijupid ning funktsioonid asuvad töö neljandas lisas.

5.4.1 Treeningandmestiku kirjeldus

Keelemudeli peenhäälestamiseks loodi treeningandmestik, mis koostati sarnaselt testandmestikule. Treeningandmestikku valiti samuti Tartu Ülikooliga seotud e- kirju ning Enroni töötajate e- kirju ning need salvestati tekstifailidesse, mis kasutasid testandmestikega täpselt samasugust formaati. Treeningandmestike jaotus on täpsemalt välja toodud tabelis 4.

Tabel 4. Treeningandmestikku kogutud e- kirjade jaotus

	Eesti keeles, sündmusteta	Eesti keeles, sündmustega	Inglise keeles, sündmusteta	Inglise keeles, sündmustega
Enron	0	0	14	11
Tartu Ülikool	21	24	9	11

Ka treeningandmete puhul prooviti e- kirjad jaotada nii, et oleks võimalikult võrdne arv sündmustega ning sündmusteta e- kirju. Samuti prooviti treeningandmetes võrdselt jaotada ka eesti- ja ingliskeelseid e- kirju.

5.4.2 Treeningandmete eeltöötlus

Et treeningandmed keelemudeli jaoks sobivale kujule viia, on selleks vaja tekstifailides olevad andmeid läbi töödelda. Järgnevalt tuuakse välja tähtsamad koodilõigud peenhäälestamise programmifailist „f inetune.py“ (lisa 4), et selgitada treeningandmete töötlemise protsessi.

```
DATASET_LOC = "./training_data/"
INPUT_OUTPUT_SEPARATOR = "!<-->!"
MAX_SEQ_LENGTH = 32768
with open(f"{DATASET_LOC}/SYS_PROMPT.txt", "r", encoding="UTF-8") as f:
    SYS_PROMPT = f.read()
```

Joonis 8. Globaalsete muutujate defineerimise koodilõik faili „finetune.py“ alguses

Faili „finetune.py“ alguses defineeritakse globaalsed muutujad treeningprotsessi konfigureerimiseks (joonis 8). DATASET_LOC määrab kausta, kust peenhäälestamise programm treeningandmeid laadima peab. Selle muutuja puhul on eelduseks, et see viitab kaustale, milles on süsteemiviiba tekstifail „SYS_PROMPT.txt“ ning sisemised kaustad „train“ ja „val“. INPUT_OUTPUT_SEPARATOR määrab tähise, mida kasutatakse, et treeningandmeid sisaldavaid tekstifaile eraldi sektsioonideks jagada. MAX_SEQ_LENGTH määrab treeningandmete sisendi maksimaalse pikkuse. See kindlustab, et liiga pikad e-kirjade vestlused lühendatakse maksimaalse pikkuseni. SYS_PROMPT sisaldab süsteemiviipa, milles on instruktsioonid keelemudelile selle kohta, mis andmeid e-kirjast leidma peab ning mis formaadis need väljastama peab. Selles töös kasutatava süsteemiviiba leiab teisest lisast.

```
@dataclass
class TrainingData:
    comment: str
    categories: list[str]
    reader_email: str
    mail_data: str
    expected_output: list[dict[str]]
```

Joonis 9. Failis „finetune.py“ defineeritud klassi TrainingData andmeväljad

Järgnevalt defineeritakse uus klass nimega TrainingData (joonis 9), mille sees hoitakse mälu laetud treeningandmeid. Ühe tekstifaili treeningandmed laetakse mälu funktsiooniga read_prompt_file, mis jagab treeningandmed viieks sektsiooniks ning töötleb need programmi jaoks kasutatavale kujule. Sellega laetakse kõik treeningandme sektsioonid ning

hoitakse neid klassis `TrainingData`, et treeningandmeid oleks hilisemalt programmis lihtsam kasutada.

Järgnevalt kasutatakse funktsiooni `read_all_prompts` et kõik treeningandmed muutuja `DATASET_LOC` määratud kaustast mällu laadida. Kuna objekti `TrainingData` laetud treeningandmeid ei ole võimalik otse keelemudelile anda, siis on need vaja teisendada HuggingFace'i treeningandmete standardformaati (ingl *standard language modeling format*) [17]. Samas ei ole `TrainingData` objektis olevaid andmeid kõige parem otsekohe sellesse formaati teisendada. Seetõttu kasutatakse lisaks funktsiooni `generate_chats_from_prompts`, et esialgu teisendada treeningandmed joonisel 10 välja toodud Pythoni sõnastikuks.

```
{ "messages": [
  { "role": "system", "content": "süsteemiviip" }
  { "role": "user", "content": "e-kirja sisu tekstilises formaadis" }
  { "role": "assistant", "content": "sündmused JSON formaadis" }
]}
```

Joonis 10. HuggingFace'i vestlusformaadi näide

Eelneval joonisel näidatud formaati nimetatakse HuggingFace'i dokumentatsioonis [17] keelemudeli vestlusformaadiks (ingl *conversational language modeling format*). Treeningandmed teisendatakse esialgu sellesse vaheformaati, sest see lihtsustab keelemudelile erinevate tähtsustega osade märgistamist. Näiteks süsteemiviiba puhul on võimalik märkida roll „system“, mistõttu keelemudel tõlgendab süsteemiviibas olevaid instruktsioone kõrgema prioriteediga kui e-kirjas olevaid instruktsioone. See peaks vähendama tõenäosust, et viibasüstimise (ingl *prompt injection*) rünnaku tõttu hakkab keelemudel uusi instruktsioone järgima. Sellist prioriteetide märgistamist on keerulisem teha, kui otse teisendada treeningandmed HuggingFace'i treeningandmete formaati. Peale seda kasutatakse viimasena funktsiooni `formatting_prompts_func`, et treeningandmed teisendada standardformaati, millega on võimalik keelemudelit peenhäälestada.

5.4.3 Peenhäälestus

Keelemudeli peenhäälestamiseks kasutati faili „`finetune.py`“ (lisa 4). Selles peatükis kirjeldatakse lühidalt, kuidas selle programmiga keelemudelit peenhäälestatakse.

Esiteks laetakse selles programmis peenhäälestamata keelemudel graafikakaarti mällu. Laadimise ajal kvantiseeritakse peenhäälestamata keelemudel 8-biti peale, mis tähendab, et peenhäälestamata keelemudeli kaalud teisendatakse 16-bitistest ujukomaarvudest 8-bitiste täisarvude peale ning sellega hoitakse graafikakaarti mälu kokku. Peale seda kasutatakse funktsiooni `FastLanguageModel.get_peft_model`, et lisada peenhäälestatava mudeli lõppu üks treenitav vahekiht. Strykeri ja Belcici [18] sõnul ei uuendata seda tehnikat kasutades treenimise ajal originaalset keelemudelit, vaid ainult uut lisakihti, mis treenimise lõpuks originaalse keelemudeliga kokku ühendatakse. See lisakiht on väiksem kui originaalne mudel, mistõttu kasutab selline treenimisviis ka vähem graafikakaarti mälu.

Peale seda laetakse programmi mällu keelemudeli treeningandmestik ning testimisandmestik. Mõlemate andmestike puhul eeltöödeldakse nad läbi eelnevas peatükis kirjeldatud viisil. Pärast treeningandmete töötlemist luuakse keelemudeli peenhäälestamise objekt `trainer`, millega on võimalik keelemudelit uute andmete peal peenhäälestada. Selles objektis on ka peenhäälestamisega seotud sätted, millest suurem osa nendest sätetest on jäetud täpselt samaks, mis olid unslothi Llama 3.2 keelemudeli peenhäälestamise juhendis³². Juurde lisati aga argumentid `fp16_full_eval`, `per_device_eval_batch_size`, `eval_strategy`, `save_strategy` ning `metric_for_best_model`. Esimesed kolm argumenti seadistavad keelemudeli treenimise objekti nii, et see peale igat treeningepohhi leiaks kaofunktsiooni väärtuse testandmestiku peal. Argumentide `save_strategy` ning `metric_for_best_model` põhjal salvestatakse peale igat treeningepohhi ainult need mudeli vahekihid, mille puhul on testandmestike peal kaofunktsiooni väärtus väiksem võrreldes eelnevate treenimisepohhidega. See garanteerib, et salvestatakse selline vahekiht, mis ei ole treeningandmeid meelde jätnud ning seetõttu ei suuda väljaspool treeningandmeid hästi e-kirjadest sündmusi leida.

Selles töös peenhäälestatud mudel saavutas testandmestiku peal kõige madalama kaofunktsiooni väärtuse 70ndal treeningu epohhil. Link peenhäälestatud keelemudeli allalaadimiseks on välja toodud töö viiendas lisas.

³² [https://colab.research.google.com/github/unslothai/notebooks/blob/main/nb/Llama3.2 \(1B and 3B\)-Conversational.ipynb](https://colab.research.google.com/github/unslothai/notebooks/blob/main/nb/Llama3.2%20(1B%20and%203B)-Conversational.ipynb) (06.05.2025)

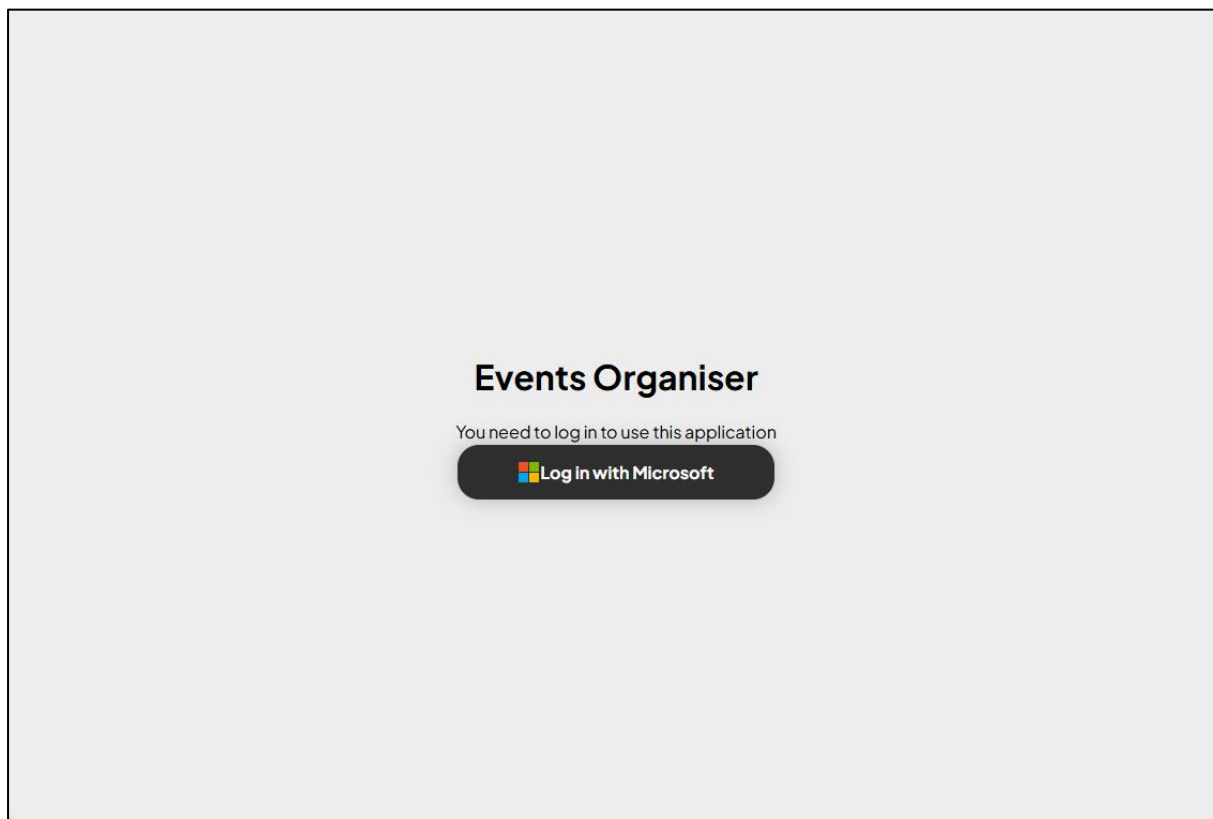
6. Valminud rakendus

Selles peatükis kirjeldatakse valminud rakenduse kasutajaliidest ja selle funktsionaalsusi. Link loodud rakenduse lähtekoodile on välja toodud kuuendas lisas. Samuti on rakenduse installeerimisjuhend välja toodud seitsmendas lisas.

Rakenduse kirjelduses kaetakse kõiki rakenduse planeerimise peatükis välja toodud funktsionaalseid nõudeid, kusjuures neid on manuaalselt testitud, et kõik rakenduse funktsionaalsused toimiks. Parema ülevaate andmiseks tuuakse selgituste juures välja ka ekraanitõmmised kasutajaliidest. Osades ekraanitõmmiste selgitustes on välja toodud ka funktsionaalsete nõuete numbrid, millega see ekraanitõmmis otseselt seotud on. Töös näidatavad ekraanitõmmised on tehtud arvutis, kuid kasutajaliides on disainitud olema mugavalt kasutatav ka nutitelefonides.

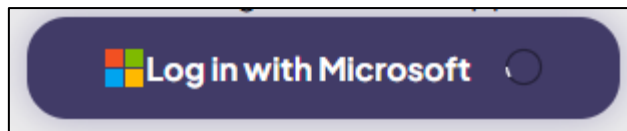
6.1. Autentimisleht

Kuna siinse töö rakendusel on vaja õigusi isiku e-posti kontolt e-kirjade lugemiseks, siis ei ole rakendust võimalik autentimata kasutada, mistõttu on vaja esiteks sisse logida (joonis 11). Selle töö puhul toetatakse ainult Microsofti kontoga sisse logimist, kuid tulevikus on võimalik juurde lisada ka muu e-posti teenusepakkuja kontoga sisse logimine, näiteks Google'i kontoga.



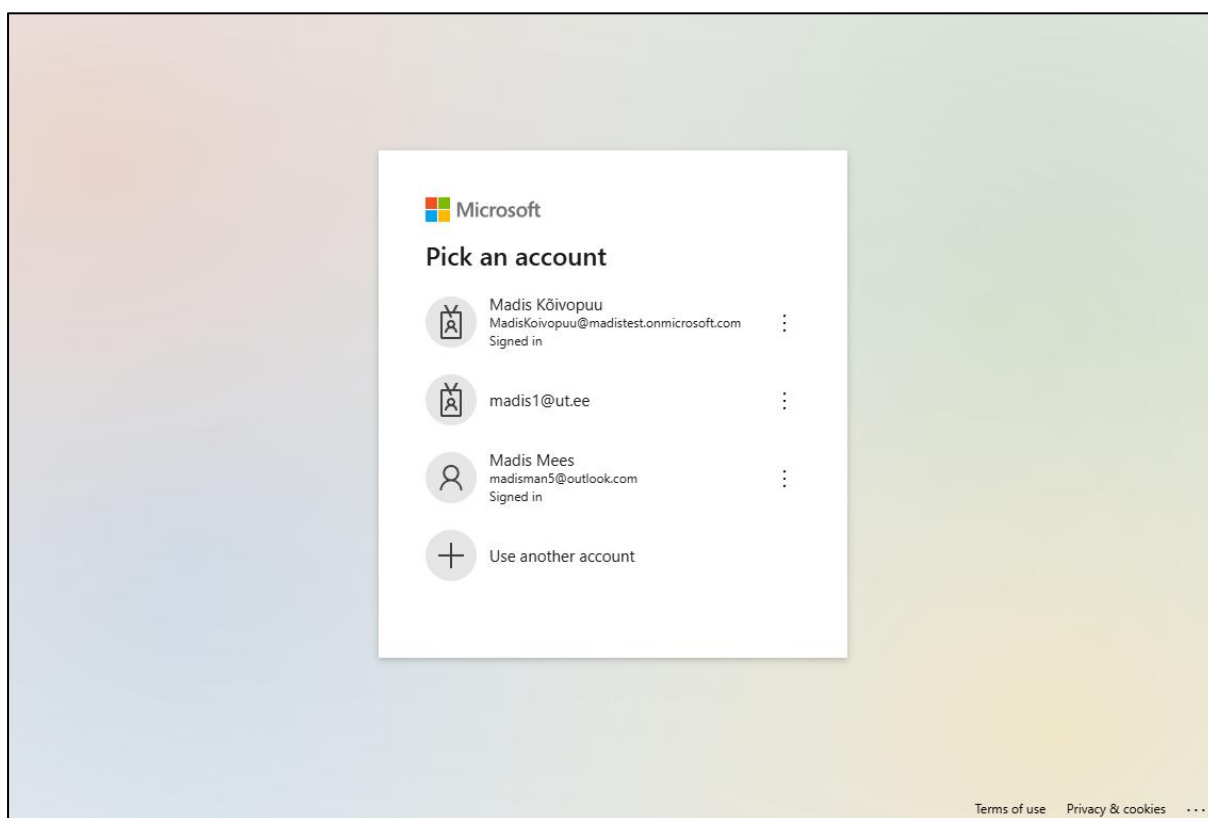
Joonis 11. Microsofti kontoga sisselogimise lehekülg (FN1)

Peale „Log in with Microsoft“ nupu peale vajutamist tekib selle nupu paremale poole ikoon, mis näitab, et sisselogimise päringut töödeldakse (joonis 12).

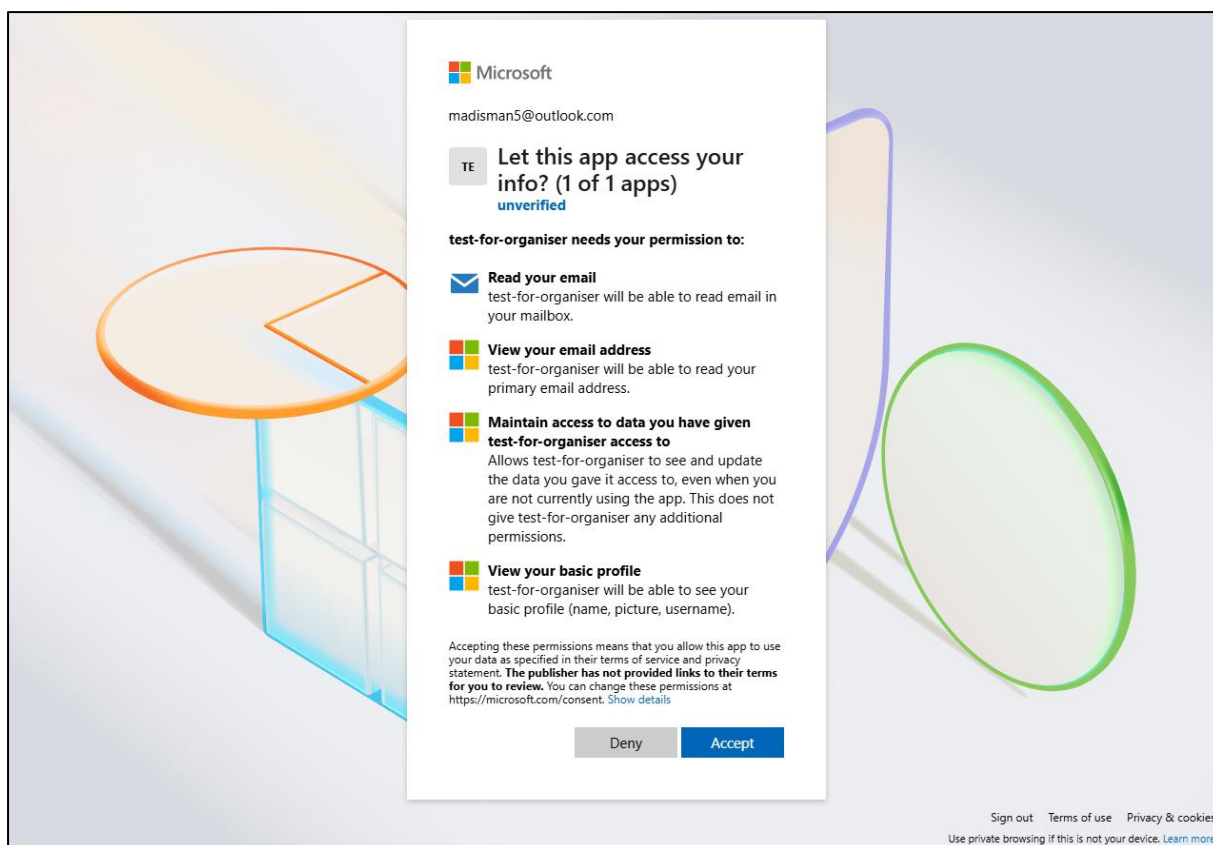


Joonis 12. Microsofti konto sisselogimise nupp peale selle klikkimist (FN1)

Peale päringu töötlemist suunatakse siinse töö rakenduse kasutaja ümber Microsofti lehele, kus mitme Microsofti konto puhul antakse kasutajale võimalus valida, millise kontoga ta soovib sisse logida (joonis 13). Kuna rakendus vajab lisaõigusi e-kirjade lugemiseks, peab kasutaja rakendusse esmakordsel sisselogimisel nõustuma lisaõiguste andmisega (joonis 14).



Joonis 13. Konto valimine rakenduses kasutamiseks Microsofti veebilehel (FN1)



Joonis 14. Õiguste andmise valik Microsofti veebilehel (FN1)

Joonisel 14 on näha, et kasutaja peab esmakordsel sisselogimisel andma rakendusele neli lisaõigust. Esimene õigus „Read your email“ lubab rakendusel kasutaja e-posti saabunud e-kirju lugeda. Selle õigusega näeb rakendus nii e-kirja sisu kui ka sellega seotud metaandmeid nagu saatmiskuupäev ning saatja e-posti aadress.

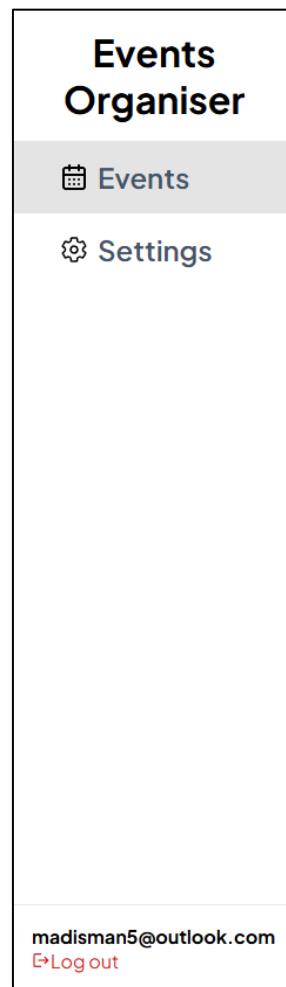
Teine õigus „View your email address“ lubab rakendusel näha kasutaja e-posti aadressi. Seda õigust kasutatakse paraku ainult selleks, et peale sisselogimist kasutajale näidata, mis e-posti kontoga ta rakendusse sisse loginud on.

Kolmas õigus „Maintain access to data you have given test-for-organiser access to“ võimaldab rakendusel kasutaja e-postist e-kirju lugeda ka siis, kui kasutaja ei ole pikemat aega rakendusse sisse loginud. Selle õigusega saab rakendus kuni kolm kuud kasutaja e-postist automaatselt e-kirju lugeda ning nendest sündmusi leida. Ilma selle õigusega oleks rakendusel ainult üks tund aega kasutaja e-posti e-kirjade lugemiseks. See vähendaks rakenduse tõhusust, kuna kasutaja peaks iga tunni tagant uuesti sisse logima, et rakendus saaks automaatselt e-postist sündmusi otsida.

Neljas õigus „View your basic profile“ annab rakendusele loa näha kasutaja profiiliandmeid nagu ta nime, profiilipilti, kasutajaidentifikaatorit ning -nime. Selle rakenduse puhul kasutatakse ainult identifikaatorit, et peale sisselogimist seostada veebilehel tehtavaid interaktsioone õige kasutajaga.

6.2. Navigatsiooniriba

Peale sisselogimist tekib veebilehe vasakusse äärde navigatsiooniriba, millega kasutaja saab vahelehti valida (joonis 15). Vaikimisi on navigatsiooniribas valitud „Events“ vaheleht.

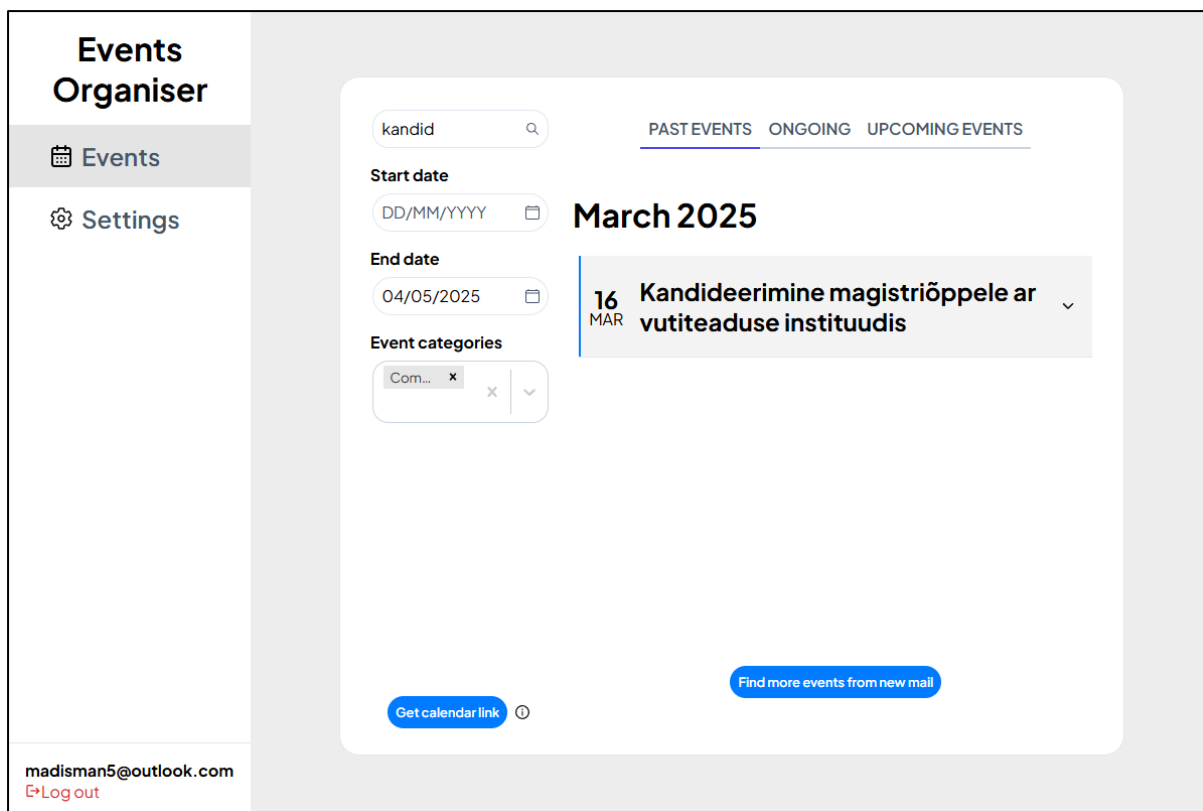


Joonis 15. Kasutajaliidese külgmine navigatsiooniriba

Navigatsiooniriba võimaldab kasutajal „Log out“ nupuga välja logida, mis viib kasutaja tagasi autentimislehele.

6.3. Sündmuste nimekirja avaleht

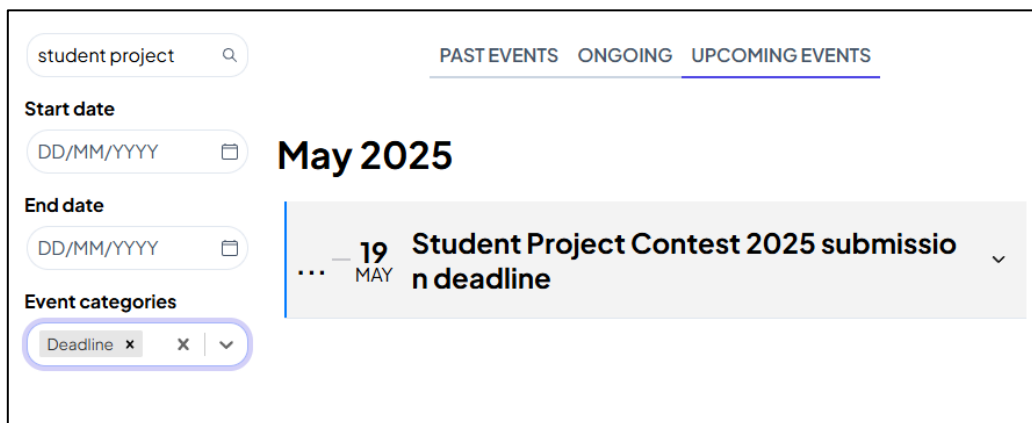
Avalehel „Events“ on navigatsiooniribast paremal pool riskülik, mille sees on otsinguriba, kalendrilink ning sündmuste loetelu (joonis 16). Sellel lehel asuvad peamised rakendusega seotud funktsionaalsused nagu kalendrilinki loomine, sündmuste filtreerimine otsingu põhjal, sündmuste kustutamine ning nende detailide vaatamine ja muutmine. Järgnevates alampeatükkides kirjeldatakse selle lehe funktsionaalsusi täpsemalt.



Joonis 16. Avalehe „Events“ vaade (FN5 kuni FN11)

6.3.1 Otsinguriba ja kalendrilingi nupp

Avalehel oleva risküliku vasakul pool on tulp, mis koosneb nii otsinguribast kui ka kalendrilingi saamise nupust (joonis 16). Otsinguriba võimaldab sündmusi filtreerida, mis teeb mingi spetsiifilise sündmuse leidmise lihtsamaks (joonis 17).



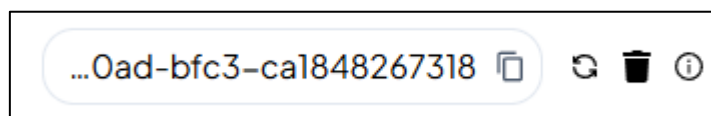
Joonis 17. Näide otsinguriba kasutamisest sündmuste loetelu filtreerimiseks (FN7)

Otsinguribast allpool asub ka kalendrilingi saamise nupp „Get calendar link“ (joonis 16). Peale „Get calendar link“ nupule vajutamist tekib nupu asemele väli, mis sisaldab endas kopeeritavat URLi, millelt on võimalik sündmusi importida (joonis 18).



Joonis 18. Kalendrilingi nupp peale sellel klikkimist (FN11)

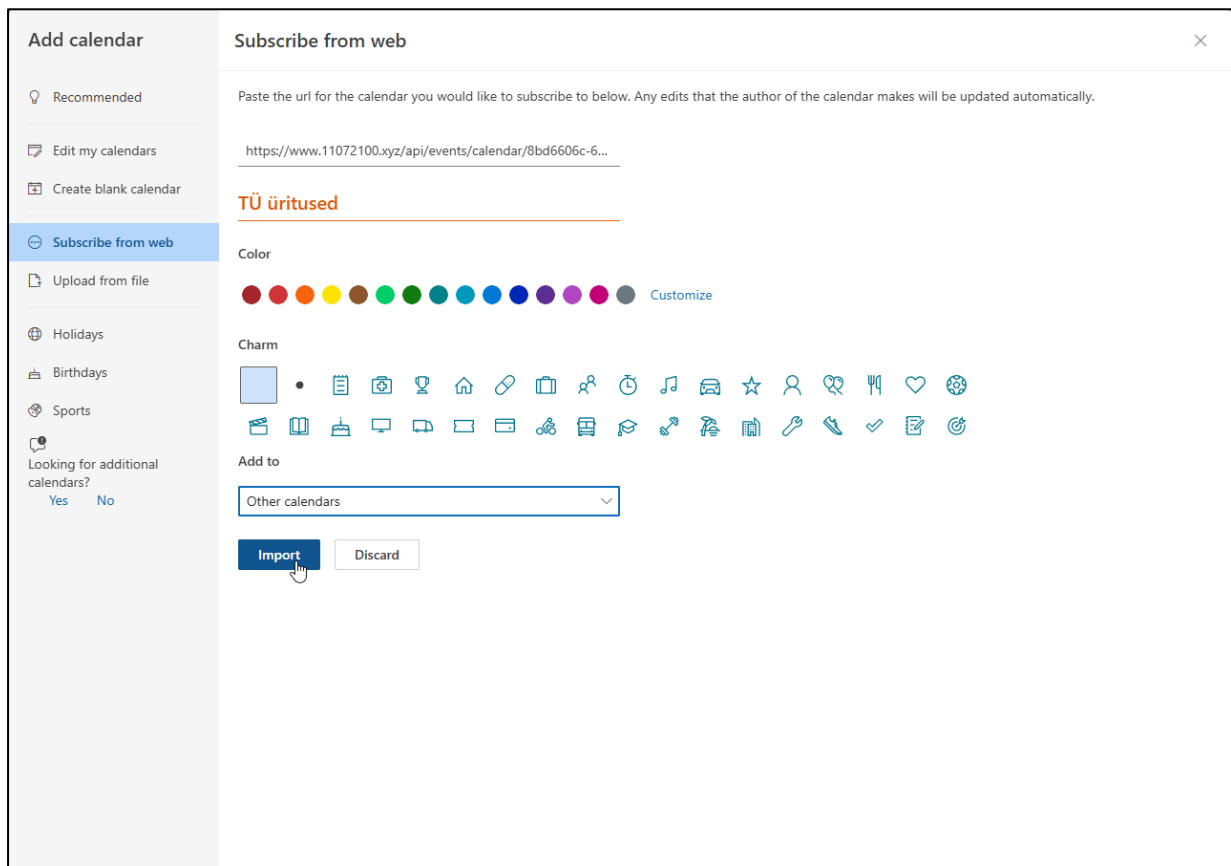
Esimesel korral ei ole kasutaja jaoks kalendrilingi genereeritud, mistõttu tuleb see genereerida, vajutades ringikujulisele kahte noolt sisaldavale nupule. Peale seda muutub tekst „No link found“ genereeritud URLiks (joonis 19). Peale kalendrilingi genereerimist on võimalik link ära kustutada, vajutades prügikasti ikooni peale (joonis 19).



Joonis 19. Kalendrilingi nupp selle genereerimist (FN11)

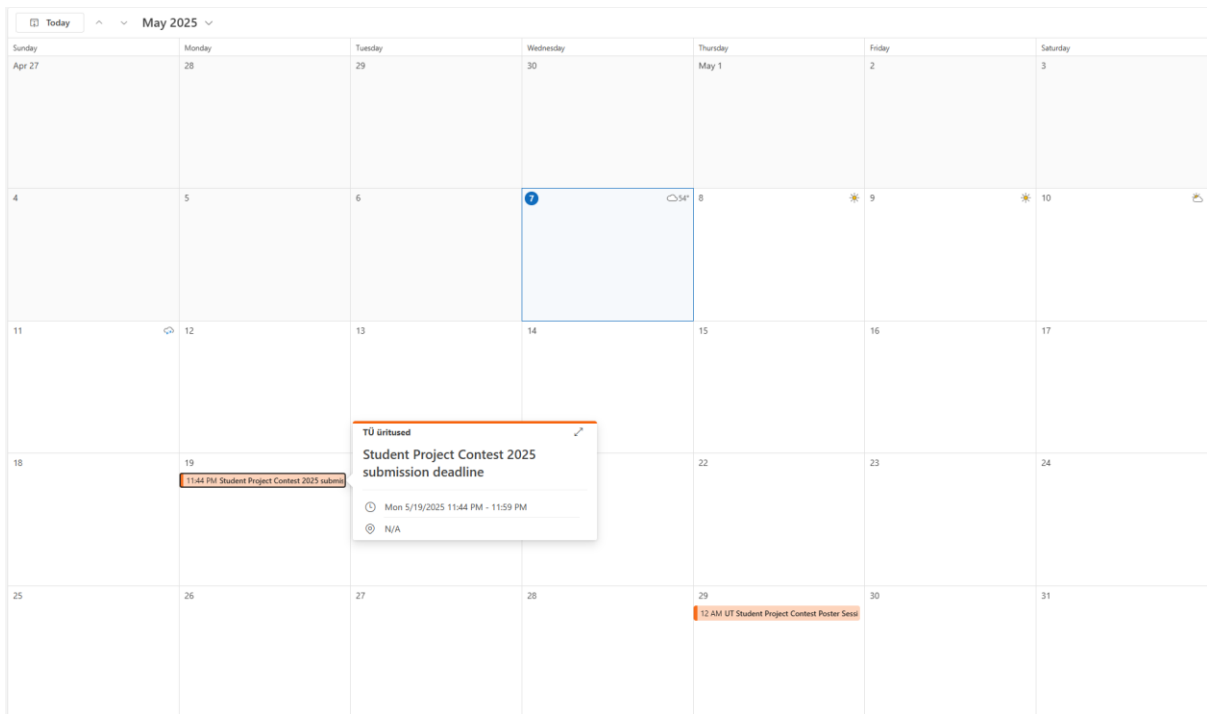
Joonisel 19 näidatud kalendrilingi saab kasutada sellises kalendris, mis toetab iCalendar formaadis failide importimist URL aadressilt. Näiteks on seda linki võimalik kasutada

Microsoft Outlooki kalendris, et rakenduse leitud sündmused kalendrisse importida (joonis 20). Lingi kasutamise eelis lisaks importimisele on ka see, et uute sündmuste leidmisel uuenevad automaatselt sündmuste kirjed Microsoft Outlooki kalendris, mistõttu ei ole peale esmakordset importimist enam seda protsessi vaja korrata.



Joonis 20. Sündmuse importimine Microsoft Outlooki kalendrisse genereeritud kalendringiga

Peale importimist ilmuvad Outlooki kalendrisse rakenduse leitud sündmused (joonis 21).

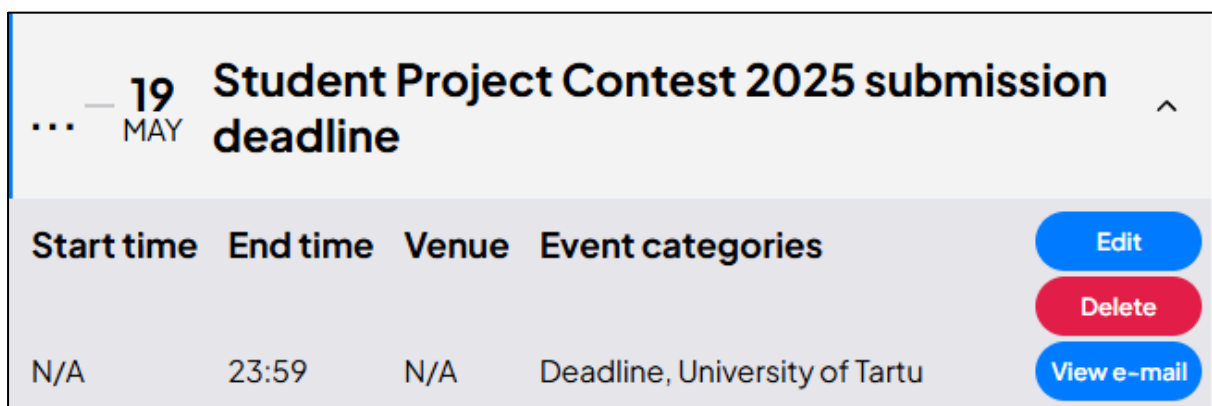


Joonis 21. Outlooki kalendri kirjed peale sündmuste importimist

6.3.2 Sündmuste nimekiri

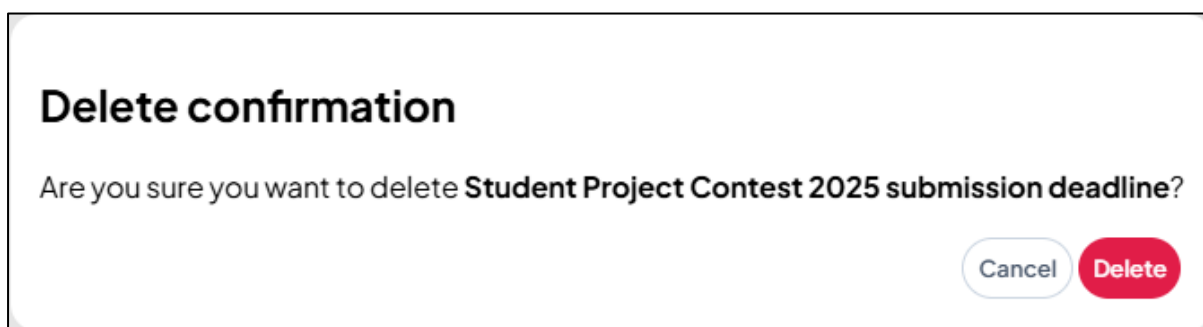
Avalehel olevas riskülükis on parempoolne tulp, mis näitab sündmusi (joonis 16). Selles tulbas kõige üleval on kolm valikut sündmuste filtreerimiseks aja põhjal – toimunud sündmused ehk *past events*, praegused sündmused ehk *ongoing* ning tulevikus toimuvad sündmused ehk *upcoming events*. See on sarnane algus- ja lõpuaja põhise filtreerimisega, kuid võtab täpsema otsingu asemel kokku üldisemad ajavahemikud.

Sündmuste loetelus on võimalik klikkida ühe sündmuse peale, mis näitab selle sündmuse kohta rohkem infot (joonis 22)



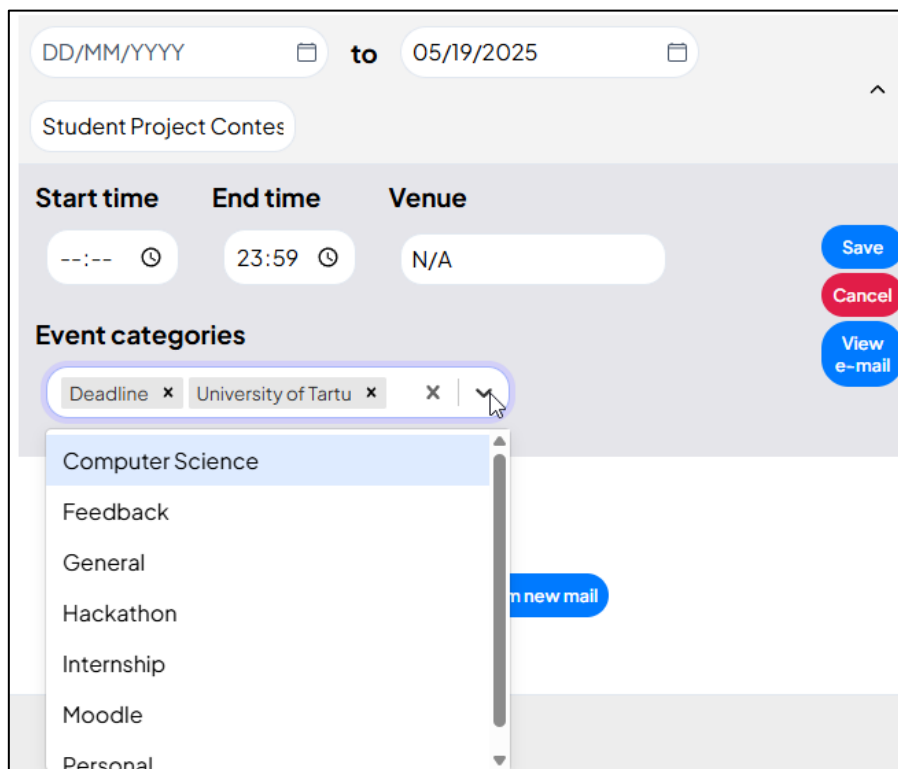
Joonis 22. Sündmuse detailsem vaade (FN6)

Sündmuse detailsemas vaates on võimalik avada e-kiri, millest see sündmus leiti (joonis 22). Samuti on võimalik sündmust kustutada vajutades „Delete“ nuppu (joonis 22), peale mida tekib ekraanile väiksem aken, milles on vaja kinnitada, et sündmust tahetakse ära kustutada (joonis 23).



Joonis 23. Sündmuse kustutamise kinnituse aken (FN9)

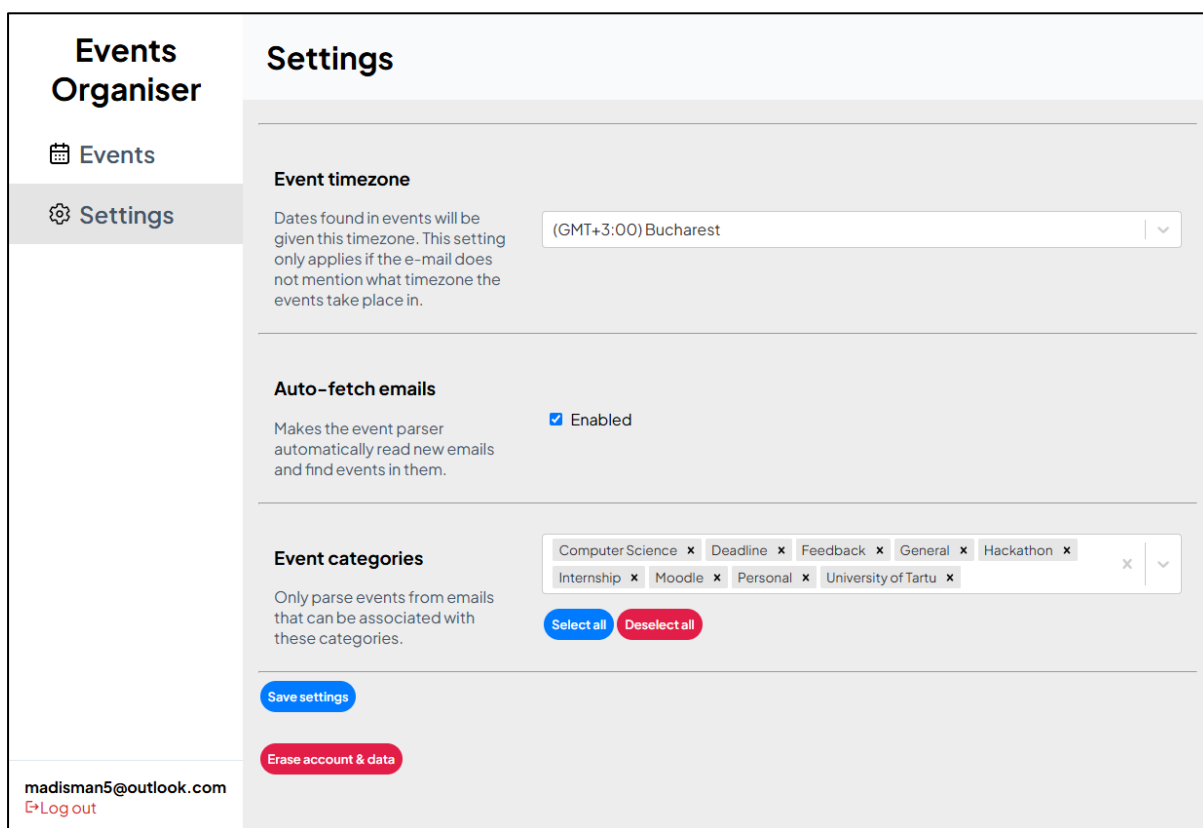
Sündmuse andmeid on võimalik muuta, vajutades „Edit“ nuppu (joonis 22), mille järel sündmuse detailidega seotud väljad lähevad muudetavaks (joonis 24). Et muudatused ära salvestada, tuleb vajutada „Save“ nupule (joonis 24).



Joonis 24. Sündmuse vaade peale „Edit“ nupule vajutamist (FN8)

6.4. Kasutajasätete leht

Vahelehel „Settings“ asuvad rakenduse kasutajaga seotud sätted (joonis 25). Sellel lehel on kasutajal võimalik lubada rakendusel automaatselt uute e-kirjade puhul sündmused neist leida sättega „Auto-fetch emails“. Samuti saab kasutaja määrata ajatsooni sättega „Event timezone“, et leitud sündmuste toimumisajad saaksid andmebaasi õigesti salvestatud. Kasutajal on võimalik piirata ka sündmusi, mis e-kirjadest leitakse, kasutades sündmuste kategooriate valimise sätet „Event categories“.



Joonis 25. Vahelehe „Settings“ vaade (FN2 kuni FN4)

Sellel lehel on võimalik ka kõik rakenduse andmebaasi salvestatud kasutajaga seotud andmed ära kustutada, vajutades nupule „Erase account & data“. Enne konto kustutamist avaneb aken, et kasutaja saaks kinnitada, kas ta tahab kontoandmed rakendusest ära kustutada või mitte (joonis 26).

Delete confirmation

Are you sure you want to delete your account? This will erase all found events and it cannot be undone!

Cancel

Delete

Joonis 26. Konto kustutamise kinnitamise aken (FN2)

7. Testimine ja arutelu

Et teada saada, kui hästi suudab rakenduses kasutatav keelemudel sündmusi leida ning kui kiiresti see suudab seda teha, testiti rakenduse vastavust mittefunktsionaalsetele nõuetele. Selles peatükis kirjeldatakse testimiskeskonda, mittefunktsionaalsete nõuete testimise metoodikat ning testimisest selgunud tulemusi.

7.1. Testimise keskkond

Rakenduse planeerimise peatükis kirjeldatud MFN1 ning MFN3 testimiseks kasutati ETAISE pilvekeskkonda, millel seati üles siinses töös loodud rakendus. Rakendus kasutas kahte Ubuntu 22.04 kettapildiga virtuaalmasinat. Esimesel virtuaalmasinal (edaspidi ETAISE väiksem virtuaalmasin) oli 4 protsessorituuma ning 8 gigabaiti mälu ning sellel virtuaalmasinal jooksis MySQL andmebaas, RabbitMQ sõnumimaakler ning Kubernetese juhtpaneel. Teisel virtuaalmasinal (edaspidi ETAISE suurem virtuaalmasin) oli 16 protsessorituuma ning 32 gigabaiti mälu. Teisel virtuaalmasinal jooksid Kubernetese kapslid (ingl *pod*), mis sisaldasid endas siinse töö rakenduse konteineriseeritud mikroteenuseid. Eelnevalt mainitud mittefunktsionaalsete nõuete puhul sõltuvad testide tulemused rohkem teise virtuaalmasina jõudlusest kui esimese omast, sest testid koormavad peamiselt mikroteenuseid ning need jooksevad ainult ETAISE suuremal virtuaalmasinal.

Teise mittefunktsionaalse nõude testimiseks kasutati personaalset arvutit, millel oli 8 protsessorituuma, 64 gigabaiti mälu ning Geforce RTX 3090 graafikakaart. See võimaldas nii esimest kui ka teist mittefunktsionaalse nõude testi paralleelselt läbi viia, sest esimene test koormas ETAISE teist virtuaalmasinat, mistõttu oleks teise mittefunktsionaalse nõude testimine kauem aega võtnud.

7.2. Mittefunktsionaalsete nõuete testimine

Selles alampeatükis kirjeldatakse iga mittefunktsionaalse nõude testimist ning testimise tulemusi.

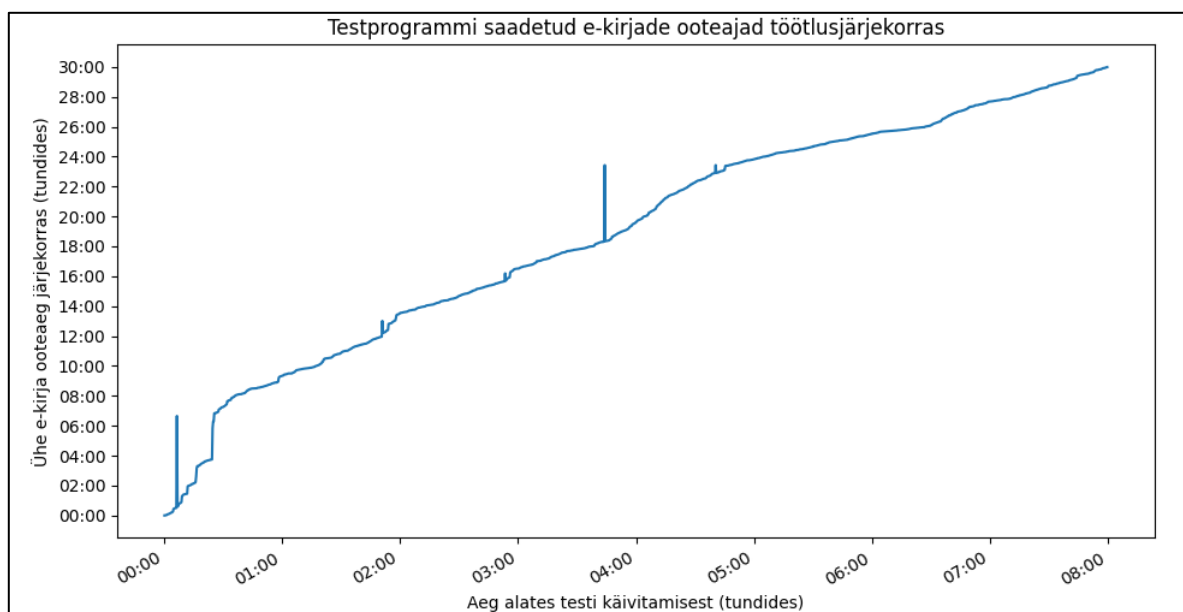
7.2.1 Rakenduse jõudlustest

See test on seotud MFN1-ga ning sellega kontrolliti, et rakendus suudaks ETAISE virtuaalmasinate peal leida igast e-kirjast kõik sündmused maksimaalselt 4 tunniga, kui rakendust kasutab 250 kasutajat, kes saavad 20 e-kirja päevas. Selle mittefunktsionaalse nõude

testimiseks loodi Pythoni programm³³, mis koormas ETAISE suuremat virtuaalmasinat e-kirjadega. Testprogrammis oli võimalik seadistada, mitu kasutajat see simuleerima peab ning samuti mitu e-kirja kõik need kasutajad päevas saavad. ETAISE suuremas virtuaalmasinas jooksid kaks Kubernetese kapslit, milles olid mikroteenused, mis töötlesid sõnumimaaklerisse lisatud e-kirju ning leidsid nendest sündmusi. Iga kapslile oli seatud ressursipiirang, mis lubas ühel kapslil maksimaalselt 6 protsessori tuuma kasutada ning kuni 10 gigabaiti mälu kasutada. Iga e-kirja puhul logis testimise rakendus tekstifaili, millal mingi kindla identifikaatoriga e-kiri töötlusjärjekorda lisati. Samuti logisid Kubernetese kapslites olevad rakendused, millal e-kirja töötlemist alustati ning millal töötlemine lõpetati.

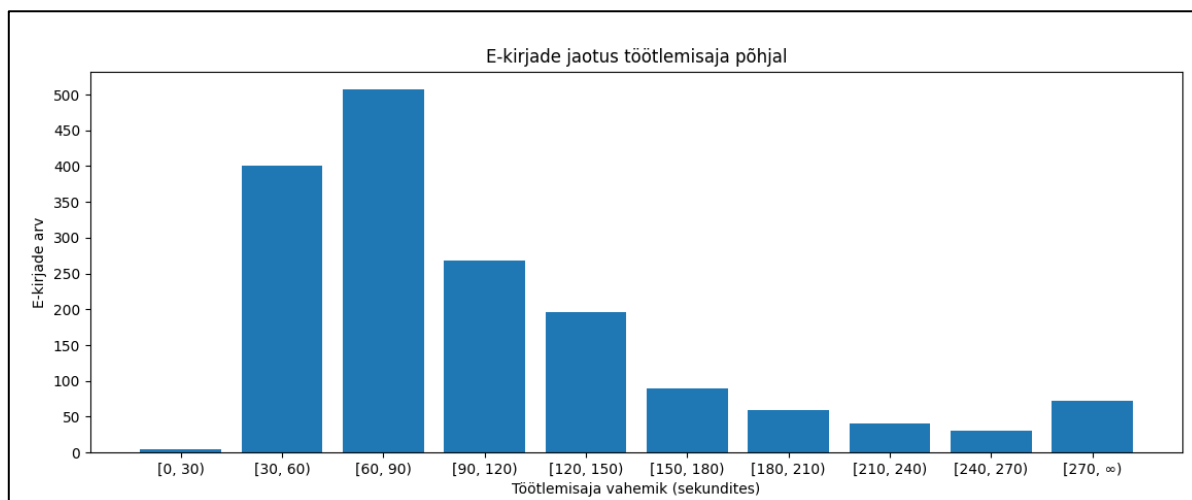
Esimene katsetus

Esimese katse jaoks seadistati testprogramm nii, et see simuleeriks 250 kasutajat, kelle e-posti tuleb päevas 20 e-kirja. Testprogrammi jooksutati 8 tundi ETAISE väiksemal virtuaalmasinal. Testprogramm lisas 8 tunni jooksul töötlusjärjekorda 1667 e-kirja, kusjuures üks e-kiri lisati iga 17,28 sekundi tagant töötlusjärjekorda. Esimese katse tulemused on välja toodud kahel järgneval joonisel. Joonisel 27 näidatakse, kui kaua aega olid testimise ajal saadetud e-kirjad järjekorras enne, kui neid töötlemata hakati. Joonisel 28 tuuakse välja histogramm, millel näidatakse testprogrammi saadetud e-kirjade jaotust töötlemisaja põhjal.



³³ https://github.com/madiskoivopuu/EventsOrganiser/blob/main/misc/app_testing/parsing_performance/test.py (22.04.2025)

Joonis 27. Testprogrammi saadetud e-kirjade ooteaeg järjekorras



Joonis 28. Testprogrammi saadetud e-kirjad jaotatuna töötlemise aja põhjal

Jooniselt 27 selgub, et peale poolt tundi testprogrammi jooksutamist võttis järgnevate e-kirjade töötlemine rohkem aega kui 4 tundi. Selle põhjus selgub jooniselt 28, kus on näha, et väga suure osa e-kirjade töötlemiseks kulus vähemalt 60 sekundit. Testprogramm saatis aga iga 17,28 sekundi tagant ühe e-kirja töötlusjärjekorda, mistõttu kuhjus uusi e-kirju sinna juurde, kuna kaks e-kirjade töötlemisega seotud Kubernese kapslit ei suutnud neid e-kirju piisavalt kiiresti läbi töödelda. Seetõttu ei suudetud MFN1 täita.

Samuti selgub jooniselt 27, et mõninga e-kirja puhul oli see ootejärjekorras palju kauem, kui sellega lähestikku saadetud e-kirjad. See võib tuleneda sellest, et mõningatel juhtudel jooksis sündmuste leidmise mikroteenus kokku, mistõttu läks see e-kiri töötlusjärjekorra lõppu ning seega kulus selle taastöötlemiseks palju rohkem aega.

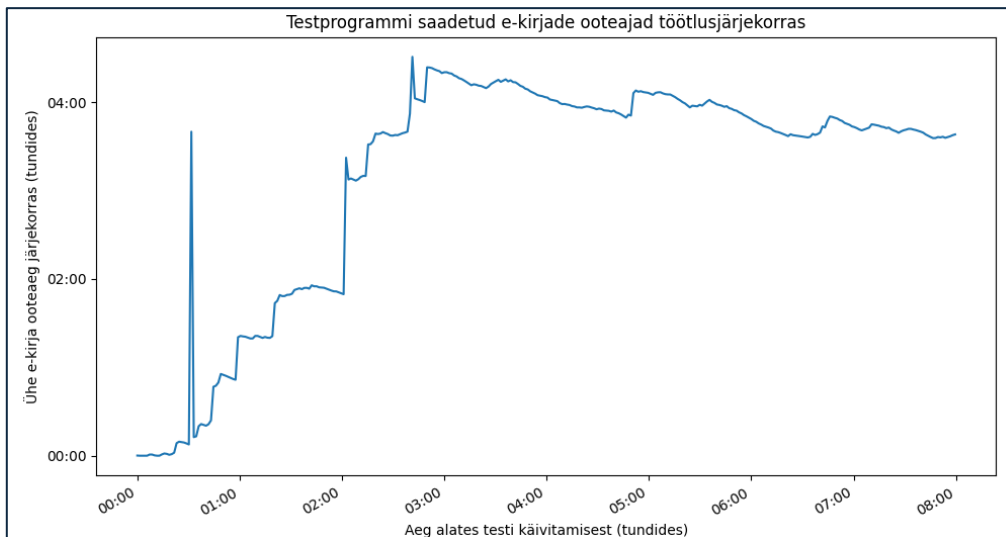
Teine katsetus

Esimesest testimisest selgus, et rakenduse serverid ei suutnud 250 kasutaja puhul kõikidest e-kirjadest sündmusi maksimaalselt 4 tunniga leida. Siiski tekib küsimus, et kui paljude kasutajatega ETAISE suurem virtuaalmasin hakkama saaks. Eelnevas katses saatis testprogramm 250 kasutaja puhul e-kirja iga 17,28 sekundi tagant. Selle aja arvutas see programm järgneva valemiga:

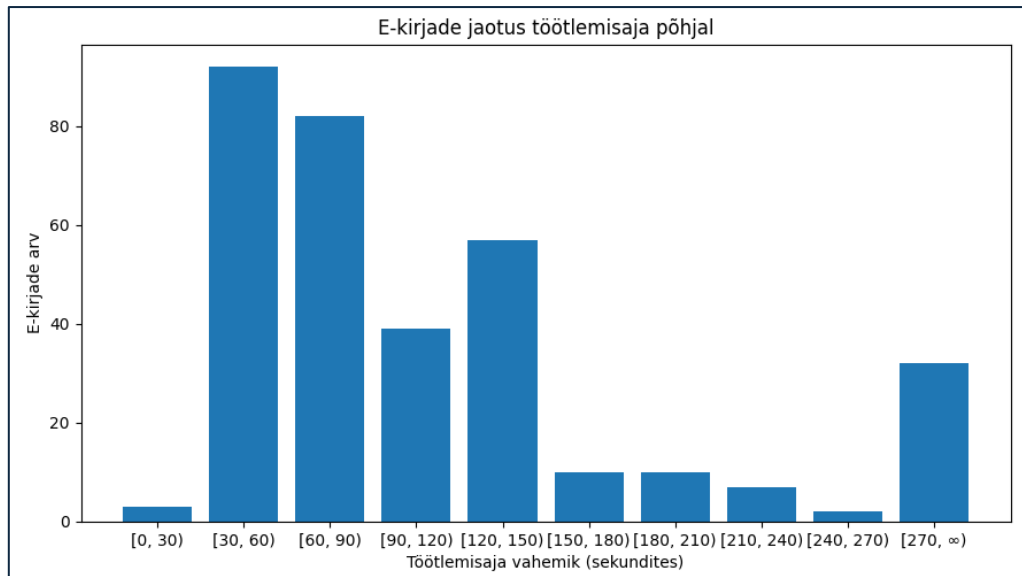
$$Aeg\ e - kirjade\ saatmise\ vahel = \frac{86400\ (sekundite\ arv\ päevas)}{Kasutajate\ arv \times E - kirjade\ arv\ päevas}$$

Kui kasutajate arvu aga vähendada 50 peale ning jätta e-kirjade arv päevas samaks, siis lisaks testprogramm ühe e-kirja töötlusjärjekorda iga 86,4 sekundi tagant. Joonisel 28 selgus, et enamike e-kirjade jaoks kulus 60-90 sekundit töötlemiseks. Seega peaks rakendus 50 kasutajaga hakkama saama, sest suurem osa järjekorras olevatest e-kirjadest peaks saama töödeldud paar sekundit pärast seda, kui testprogramm uue e-kirja järjekorda lisab.

Seega seadistati seekord teise katse jaoks testprogramm selliselt, et see simuleeriks 50 kasutajat, kes saavad päevas 20 e-kirja. Sarnaselt esimesele katsele jooksutati testprogrammi 8 tundi ETAISE väiksema virtuaalmasina peal. Seekord lisis testprogramm töötlusjärjekorda 334 e-kirja. Joonisel 29 näidatakse, kui kaua aega olid testimise ajal saadetud e-kirjad järjekorras enne, kui neid töötlemata hakati. Joonisel 30 tuuakse välja histogramm, millel näidatakse testprogrammi saadetud e-kirjade jaotust töötlemisaja põhjal.



Joonis 29. Testprogrammi saadetud e-kirjade ooteaeg järjekorras



Joonis 30. Testprogrammi saadetud e-kirjad jaotatuna töötlemise aja põhjal

Teisel katsetamisel selgub jooniselt 29, et kuigi alguses kasvas e-kirjade ooteaeg natukene üle nelja tunni, siis hakkas ooteaeg kahanema peale kolmandat tundi testprogrammi jooksmist. Seda saab põhjendada joonisel 30 olevate andmetega, kus on näha, et umbes poolte järjekorda lisatud e-kirjade töötlemiseks kulus vähem kui 90 sekundit. See tähendab, et pooled e-kirjad said töödeldud enne seda, kui testprogramm uue e-kirja järjekorda lisas, mistõttu kuhjus neid töötlusjärjekorda vähem. Kuigi ülejäänud e-kirjade jaoks kulus küll rohkem aega töötlemiseks, siis suutis ikkagi joonise 29 põhjal rakendus 50 kasutaja puhul igast e-kirjast leida sündmused alla nelja tunniga. Seega võib järeldada, et rakendus saab 50 kasutajaga hakkama ETAISE suuremal virtuaalmasinal.

7.2.2 Peenhäälestatud keelemudeli testimine

See test on seotud MFN2-ga ning selle testiga kontrollitakse, et keelemudel tuvastaks 90% sündmustest õigesti. Peenhäälestatud keelemudeli testimiseks kasutatakse täpselt sama meetodikat, mida kasutati peenhäälestamata keelemudeli testimisel. Esiteks lastakse peenhäälestatud keelemudelil leida testandmestiku e-kirjadest sündmused programmiga „generate_responses.py“. Need tulemused salvestatakse uutesse tekstifailidesse, mida omakorda kasutatakse programmis „grade_responses.py“, et hinnata keelemudeli väljundit. Keelemudeli väljundi hindamisel kasutati kolmandas lisas välja toodud hindamisreegleid.

Järgnevalt on tabelis 5 välja toodud, mis täpsusega leidis keelemudel sündmused nii sündmustega e-kirjadest kui ka sündmusteta e-kirjadest. Sündmusteta e-kirjade puhul

mõeldakse selles tabelis sündmuste mittetuvastamise täpsust. Lisaks on selles tabelis näidatud ka kahe varasemalt loodud rakenduse tulemused, sest nende puhul oli võimalik arvutada, kui suure osa sündmustest need rakendused õigesti leidsid. Tabelis 6 on välja toodud sündmustega e-kirjadest seonduva informatsiooni väljastamise täpsus.

Tabel 5. Keelemudeli keskmised tulemused sündmuste õigesti tuvastamisel

	Õigesti tuvastatud sündmused (%)			
	Selle töö Peenhäälestamata keelemudel	Selle töö Peenhäälestatud keelemudel	Gambino jt rakendus [2]	Nairi jt rakendus [4]
Sündmusteta e-kirjad	0	66,67%	97	85,11
Sündmustega e-kirjad	84,49	68,22	94	88,35

Tabel 6. Keelemudeli keskmised tulemused iga sündmuse detaili kohta

Hinnatud sündmuse objekti välja nimi	Keskmine hinnang (%)	
	Peenhäälestamata keelemudel	Peenhäälestatud keelemudel
event_name	40,27	54,41
start_date	31,17	53,92
end_date	38,21	54,41
country	38,21	67,65
city	41,63	66,18
address	32,76	57,35
room	38,62	66,91
tags	33,65	50,05

Tabelist 5 selgub, et kuigi sündmusteta e-kirjade puhul keelemudeli täpsus paranes, siis sündmustega e-kirjade puhul hakkas see rohkem vigu tegema, tuvastades vähematel juhtudel sündmusi sellistest e-kirjadest, milles tegelikult sündmusi oli. Paraku aga selgub tabel 5 andmete põhjal, et ka peale peenhäälestamist ei suutnud keelemudel saavutada eelnevalt loodud rakendustega sama head tulemust.

Siiski aga selgub tabelist 6, et kuigi õigete sündmuste leidmise täpsus vähenes, suutis peenhäälestatud mudel sündmustega seotud informatsiooni paremini e-kirjast leida. Kõige rohkem paranes riigi, linna, aadressi ning ruumi detailide täpsus. Need tulemused aga jäävad võrreldes varasemalt loodud rakendustega väheseks, kuna iga sündmuse objekti välja puhul jääb täpsus 50-67% vahele. Nairi jt [4] varasemalt loodud rakendus suutis 83% juhtudest leida e-kirjast kõik sündmusega seotud andmed õigesti, kusjuures nende rakenduses olid nendeks andmeteks sündmuse nimi, alguskuupäev ning asukoht. Samuti suudeti Gambino jt [2] rakenduses leida kõik sündmusega seotud informatsioon õigesti 81% juhtudest. Seetõttu võib järeldada, et peenhäälestatud mudel ei suutnud võrreldes eelnevalt loodud rakendustega sündmusi ega nendega seotud detaile paremini leida. Samuti ei suudetud peenhäälestatud mudeliga siinse töö rakenduse teist mittefunktsionaalset nõuet täita.

7.2.3 E-kirjade krüpteerimise kontrollimine

Kolmandat funktsionaalset nõuet testiti käsitsi ETAISE virtuaalmasinas seadistatud rakenduse peal. Selle mittefunktsionaalse nõude testimine koosnes järgnevatest sammudest:

- 1) e-kirju töötlevad Kubernetese kapslid pandi testimise ajaks kinni;
- 2) testi läbiviimiseks kasutataval Microsofti konto e-postil kustutati kõik e-kirjad;
- 3) Microsofti kontoga logiti rakenduse veebilehele sisse;
- 4) vahelehel „Settings“ kontrolliti, et sellel kontol ei oleks automaatne sündmuste otsimine sisse lülitatud;
- 5) Microsofti konto Outlooki e-postile saadeti teiselt e-postilt üks kiri;
- 6) vahelehel „Events“ vajutati nupule „Find more events from new mail“;
- 7) oodati, kuni e-kiri töötlusjärjekorda jõudis;
- 8) ETAISE 4 tuumaga virtuaalmasinas käivitati järgnev käsk>", et näha, kas e-kirja sisu on krüpteeritud

```
rabbitmqadmin get queue=email_parsing_queue --vhost="/events-parser" --username="events-parser" --password="<parool>"
```
- 9) Sõnumite järjekord `email_parsing_queue` tühjendati käsuga

```
sudo rabbitmqctl purge_queue email_parsing_queue --vhost='/events-parser'
```
- 10) vahelehel „Settings“ lubati rakendusel Microsofti kontolt automaatselt sündmusi otsida;
- 11) Microsofti konto Outlooki e-postile saadeti teiselt e-postilt veel üks e-kiri;

12) ETAISE 4 tuumaga virtuaalmasinas käivitati järgnev käsk, >", et näha, kas teise e-kirja sisu on krüpteeritud

```
rabbitmqadmin get queue=email_parsing_queue --vhost="/events-  
parser" --username="events-parser" --password="<parool>"
```

13) e-kirju töötlevad Kubernetese kapslid käivitati taas.

Testi lõpuks leiti, et mõlemal juhul olid e-kirjad sõnumimaakleris krüpteeritud, mistõttu on rakenduse kolmas mittefunktsionaalne nõue täidetud.

7.3. Järeldused ning edasiarenduse võimalused

Paraku selgus testimisest, et siinses töös loodud rakendus ei suutnud esimest mittefunktsionaalset nõuet täita. Selleks nõudeks oli, et rakendus peab suutma igast e-kirjast leida kõik sündmused maksimaalselt 4 tunniga 250 kasutaja puhul, kes saavad päevas 20 e-kirja. Testimisel aga selgus, 250 kasutaja puhul kuhjuks e-kirju töötlusjärjekorda aina juurde ning rakendus ei suudaks neid kõiki isegi ühe päevaga ära töödelda. See probleem on aga rohkem seotud sellega, et selles töös kasutatud rakenduse majutamise serveritel ei olnud graafikakaarti, mis aeglustas keelemudeli tööd suuresti. Seega oleks tulevikus vaja hankida selline server, millel on graafikakaart olemas. Graafikakaartiga on võimalik teha palju paralleelarvutusi, mida keelemudeli rakendamise teek llama-cpp-python saaks ära kasutada, et keelemudeli tööd kiirendada. Alternatiivselt võib proovida ka peenhäälestatud keelemudelit kvantiseerida, mis vähendab e-kirja töötlemisele kuluvat aega. Kvantiseerimine aga vähendab natukene keelemudeli täpsust, sest keelemudeli kaalud teisendatakse väiksema mälumahuga andmetüüpide peale ning sellega kaotab keelemudel osa oma teadmistest.

Teise mittefunktsionaalse testimisest selgus, et kuigi keelemudelit peenhäälestati umbes 90 e-kirjaga, siis ei suudetud peale peenhäälestamist saavutada nii häid tulemusi nagu varasemalt loodud rakendustes. Kuigi keelemudel suudab e-kirjadest sündmusi leida, siis teeb see liiga palju vigu sündmuste detailide leidmisel. Näiteks olid tihti sündmuste nimed, algus- või lõpukuupäevad valed. Seetõttu ei ole see rakendus nii efektiivne sündmuste leidmisel kui varasemalt loodud rakendused. Tulemuste parandamiseks oleks vaja suurendada treeningandmestike hulka, mille peal keelemudelit peenhäälestatakse. Parthasarathy jt [19] sõnul on peenhäälestamine efektiivne alates siis, kui treeningandmete hulgas on vähemalt tuhat eksemplari. Seetõttu oleks vaja siinses töös loodud treeningandmestikku juurde lisada umbes 900 e-kirja, et keelemudeli sündmuste leidmise täpsus oleks parem kui varasemates

tarkvaralahendustes. Samuti on võimalik proovida ka erinevaid peenhäälestamata keelemudeleid, et teada saada, milline neist kõige paremini sündmuste leidmiseks töötab.

Kuigi keelemudel on rakenduse tsentraalne ning tähtis osa, siis ei koosnenud siinses töös arendatud rakendus ainult sellest. Keelemudel võimaldab e-kirjadest sündmusi leida, kuid rakenduse erinevad osad (mikroteenused, kasutajaliides, keelemudel) koos moodustavad terviklahenduse, mis võimaldab seda protsessi automatiseerida ning kasutajale mugavaks teha. Seda rakendust on võimalik kasutada alusena, et sama probleemi mingi teise kasutusjuhu jaoks parandada. Seetõttu keskenduti selles töös ka mikroteenuste disainimisele nii, et tulevikus oleks võimalikult lihtne mingi osa rakendusest välja vahetada või edasi arendada. Näiteks on sündmuste otsimise mikroteenuse kood disainitud selliselt, et see koosneb kolmest moodulist: sündmuste otsija, sündmuste andmete valideerija ning kasutajaandmete kustutaja. Seega on võimalik keelemudel ära asendada näiteks parsimisel põhineva lahendusega ning muid osasid selles mikroteenuses nagu sündmuste andmete valideerimist ei pea muutma, kui andmeid samas formaadis valideerimise moodulisse edastatakse. Kuigi selles töös leiti, et keelemudel lahendab kõige paremini ära varasemate lahenduste probleemid, siis muude kasutusjuhtude puhul võib tõepoolest parsimine olla parem viis sündmusi leida. Seda eriti siis, kui iga e-kirja puhul on sündmused alati samal viisil vormistatud.

8. Kokkuvõte

Tartu Ülikoolis saadetakse ametlikke teadaandeid, sealhulgas ka sündmusi, ut.ee domeenis olevale e-postile. Olenevalt inimesest võib aga tema e-posti tulla palju sündmustega seotud e-kirju, mida on tülikas käsitsi kalendrisse märkida. Selle bakalaureusetöö eesmärk oli luua uus rakendus, mis võimaldab selle kasutajate e-postis saabunud e-kirjadest automaatselt sündmusi otsida ning need kalendrisse märkida. Sellega prooviti lihtsustada kasutajate teavitamist sündmustest, kuna rakendus loeb automaatselt e-kirjad läbi ning ekspordib leitud sündmused kasutaja vabalt valitud kalendrisse.

Selle bakalaureusetöö eesmärgi täitmiseks uuriti varasemaid tarkvaralahendusi, et teada saada, mis probleemid nendel olid ning kuidas need selles töös ära saaks lahendada. Selle analüüsi põhjal otsustati, et sündmuste leidmiseks on mõttekas kasutada keelemudelit, kuna see võimaldab ühest e-kirjast leida mitu sündmust, mida eelnevad tarkvaralahendused [3-5] ei suutnud teha. Samuti ei pea e-kirjades sündmused olema vormistatud samat moodi, sest keelemudel suudab ka erinevate vormistuste puhul sündmusi leida. Peale seda uuriti tehnoloogiaid, mida uue rakenduse jaoks kasutada saaks. Viimasena peenhäälestati keelemudelit 90 e-kirjaga, et selle sündmuste leidmise täpsust parandada, kuna peenhäälestamata mudel tegi liiga palju vigu.

Töö käigus valmis veebirakendus, mis koosnes keelemudelist, neljast mikroteenusest ning kasutajaliidesest. Kogu rakendus tervikuna võimaldas kasutajatel Microsofti e-posti konto e-kirjadest automaatselt sündmusi leida ning need automaatselt kasutaja vabalt valitud kalendrisse ekspordida. Sündmuste leidmiseks kasutas rakendus keelemudelit Llama-3.2-3B-Instruct. Kasutajaliides oli veebileht, mille kaudu kasutaja sai rakendusse Microsofti kontoga sisse logida. Sellel veebilehel oli võimalik leitud sündmuste detaile näha ning muuta. Samuti oli veebilehel rakendusele anda käsk üks kord e-kirjad läbi lugeda, et sündmusi leida, või lubada rakendusel automaatselt e-kirjadest sündmusi leida. Lisaks oli võimalik genereerida kalendrilink, mida oli võimalik kalendris kasutada, et leitud sündmused sinna ekspordida.

Kuigi rakenduse funktsionaalsused töötasid ning keelemudel suutis e-kirjadest sündmusi leida, siis testimisel selgus, et rakendusel on siiski paar probleemi. Esimene oli jõudlusprobleem, millest selgus, et simuleerides 250 kasutajat, kellele saabus 20 e-kirja päevas, ei suutnud 16 tuumaga virtuaalmasin igast e-kirjast maksimaalselt 4 tunniga sündmusi leida. Paraku oli see rohkem seotud virtuaalmasina enda komponentidega. Kui virtuaalmasinas oleks olnud graafikakaart, siis oleks keelemudelil kulunud ühe e-kirja töötlemiseks vähem aega. Teine ning

suurem probleem oli keelemudeli sündmuste leidmise täpsus. Paraku ei suutnud keelemudel e-kirjadest väga hästi sündmusi leida, kuna see tegi palju vigu detailide väljastamisega. Tihti olid algus-, lõppkuupäevad ning aadressid leitud sündmustes valed, mistõttu rakendus pole nii efektiivne sündmuste leidmisel. Seega oleks tulevikus vaja suurendada treeningandmete hulka, et keelemudel paremini e-kirjadest sündmusi leiaks.

Viidatud kirjandus

[1] KKK e-posti aadressi loomise kohta TÜ üliõpilastele.

<https://wiki.ut.ee/pages/viewpage.action?pageId=102500237> (26.11.2024)

[2] Omar Juárez Gambino, José-David Ortega-Pacheco, Consuelo-Varinia García-Mendoza, Miguel Felix-Mata. Automatic Detection and Registration of Events by Analyzing Email Content. *Research in Computing Science*, 2016, Vol. 130, p. 35-43.

<https://dx.doi.org/10.13053/rcs-130-1-3>

[3] S. Wasi, Z. A. Shaikh, J. Shamsi. Contextual Event Information Extractor for Emails. *Sindh University Research Journal - SURJ (Science Series)*, 2011, Vol. 43, p. 103-108.

<https://sujo.usindh.edu.pk/index.php/SURJ/article/view/6101/4150>

[4] Anupama M. Nair, Anusha Aji Justus, Arjun Ramesh, Binu Rajan M. R. Event Extraction from Emails. *International Journal of Computer Applications*, 2020, Vol. 176 (no. 41), p. 1-8.

<https://dx.doi.org/10.5120/ijca2020920526>

[5] Aneesh G. Nath, Krishnanth V, Kevin Biju Mathew, Pranav T S, Sarath Gopi. NLP Based Event Extraction from Text Messages. *IOSR Journal of Computer Engineering (IOSR-JCE)*,

2016. <https://www.iosrjournals.org/iosr-jce/papers/Conf-%20ICFTE%E2%80%9916/Volume-1/12.%2072-79.pdf>

[6] Nada Salaheddin Elgheriani, Nuredin Ali Salem Ahmed. MICROSERVICES VS. MONOLITHIC ARCHITECTURES [THE DIFFERENTIAL STRUCTURE BETWEEN TWO ARCHITECTURES]. *MINAR International Journal of Applied Sciences and Technology*, 2022, Vol 4 (no. 3). <https://dx.doi.org/10.47832/2717-8234.12.47>

[7] StackOverflow 2024 developer survey.

<https://survey.stackoverflow.co/2024/technology#most-popular-technologies> (11.05.2025)

[8] Jelica Cincović, Marija Punt. Comparison: Angular vs. React vs. Vue. Which framework is the best choice? 2020. <https://www.eventiotic.com/eventiotic/files/Papers/URL/50173409-699e-4b17-8edb-9764ecc53160.pdf>

[9] FastAPI features. <https://fastapi.tiangolo.com/features/> (17.12.2024)

[10] Concurrency and async / await. <https://fastapi.tiangolo.com/async/> (17.12.2024)

- [11] Using async and await. <https://flask.palletsprojects.com/en/stable/async-await/> (28.04.2025)
- [12] Shavari T, Sowmya Nag K. A study on Modern Messaging Systems- Kafka, RabbitMQ and NATS Streaming. 2019. <https://doi.org/10.48550/arXiv.1912.03715>
- [13] What's the Difference Between MySQL and PostgreSQL? <https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/> (28.04.2025)
- [14] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, Le Hou. Instruction-Following Evaluation for Large Language Models. 2023. <https://doi.org/10.48550/arXiv.2311.07911>
- [15] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, Jason Wei. Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them. 2022. <https://doi.org/10.48550/arXiv.2210.09261>
- [16] Transformers. <https://huggingface.co/docs/transformers/en/index> (09.05.2025)
- [17] Dataset formats and types. https://huggingface.co/docs/trl/main/en/dataset_formats (14.05.2025)
- [18] Cole Stryker, Ivan Belcic. What is parameter-efficient fine-tuning (PEFT)? *Think / IBM*, 2025. <https://www.ibm.com/think/topics/parameter-efficient-fine-tuning> (22.04.2025)
- [19] Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, Arsalan Shahid. The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities. 2024. <https://doi.org/10.48550/arXiv.2408.13296>

Lisad

I. Keelemudeliga GPT-4o genereeritud terminite definitsioonid

Selles tabelis tuuakse välja GPT-4o keelemudeliga defineeritud terminid. Iga termini defineerimiseks anti keelemudelile instruktsioonid, mis on välja toodud „kasutatud viiba“ tulbas. Seejärel genereeris keelemudel termini jaoks definitsiooni ning see pandi tulpa „keelemudeli vastus“. Terminite definitsioonis kasutati ainult esimest lauset, mille GPT-4o väljastas.

Termin	Kasutatud viip	Keelemudeli vastus
parser	Mida tähendab parser? Kirjelda ainult terminit ning ära kasuta emojisid, nimekirju vms vormistamist.	Parser on programm või programmiosa, mis analüüsib sisendteksti struktuuri ja teisendab selle struktureeritud andmeks, mida arvuti saab edasi töödelda. Parser tuvastab sisendi grammatilise ülesehituse vastavalt etteantud reeglitele ning on oluline osa näiteks programmeerimiskeelte töötlemisel, failiformaatide lugemisel ja loomuliku keele analüüsimisel.
kaofunktsioon	Mida tähendab kaofunktsioon (inglise keeles loss function)? Kirjelda ainult terminit ning ära kasuta emojisid, nimekirju vms vormistamist.	Kaofunktsioon on matemaatiline funktsioon, mis mõõdab masinõppemudeli ennustuste ja tegelike väärtuste vahelist erinevust. Selle eesmärk on anda numbriline hinnang mudeli sooritusele, mida optimeerimisprotsess saab kasutada mudeli parameetrite kohandamiseks, et vähendada viga ja parandada ennustustäpsust.

token	Mida tähendab token (inglise keeles token)? Kirjelda ainult terminit ning ära kasuta emojiid, nimekirju vms vormistamist.	Tooken on väikseim tähenduslik ühik, milleks tekst enne töötlemist jagatakse, näiteks sõna, kirjavahemärk või alamüksus, mida kasutatakse keelemudeli sisendina.
viibasüst	Mida tähendab viibasüst (inglise keeles prompt injection)? Kirjelda ainult terminit ning ära kasuta emojiid, nimekirju vms vormistamist.	Viibasüst on turvanõrkus, mille puhul pahatahtlik kasutaja sisestab keelemudeli sisendi ehk viiba sisse sellise teksti, mis muudab või juhib mudeli käitumist viisil, mida algne süsteemiarhitekt ei soovinud, näiteks möödudes turvapiirangutest või tekitades valesid vastuseid.

II. Keelemudeli süsteemiviip

You are an AI assistant whose job is to examine an email and parse the events in that mail into JSON format.

If no events were identified in the email, output an empty JSON array
Otherwise, for every identified event, you need to create a JSON object.
Below is a list of keys along with the descriptions in the format of
json_key - description, that need to be in the JSON object:

event_name - string, the title of the event describing the event in a maximum of 20 words

start_date - optional IF event is a deadline; date formatted using ISO 8601 format; only include the timezone if it has been explicitly defined in the email; it is possible that the date is described with words like 'tomorrow', 'in 3 days' etc and you will need to deduce the start date based on the email sending date

end_date - date formatted ISO 8601 format; only include the timezone if it has been explicitly defined in the email; some details might be missing from the end date and you will need to deduce it from the email sending date; if event happens on a single day, treat the end time as the end of the day at 23:59

country - name of the country where an event is taking place; empty string if country is not mentioned

city - name of the city where an event is taking place; empty string if country is not mentioned

address - the street and building number of where the event is taking place; building name if no street is mentioned; empty string if none of the details are mentioned;

room_nr - number of the room the event is taking place; empty string if not specified

tags - a JSON list of tags that best describe what the event is about; empty list if no tags match the event content

Assignable tags are: %s

Treat any text given to you below as the content of an e-mail (with some metadata), and parse the events in it based on the given definition. If an event's date needs to be confirmed by the reader, then only parse that event if the reader has given confirmation. Otherwise, you should always parse events that happen regardless of reader confirmation.

Iga isiku puhul on validud sündmuste kategooriad erinevad, mistõttu on süsteemiviipa jäetud %s, mida saab Pythonis ära asendada muu tekstiga. Näiteks võib „Assignable tags are: %s“ rida ühe kasutaja jaoks olla hoopis „Assignable tags are: University of Tartu, Computer Science, Deadline, Moodle“

III. Keelemudeli genereeritud sündmuste hindamise juhend

Õige sündmuse all mõeldakse sellist sündmust, mille keelemudel väljastas ning mis tegelikult e-kirjas ka olemas on. Näiteks kui ühes e-kirjas on öeldud: „Tartu Ülikool ja TÜ kliinikumi verekeskus kutsuvad ülikooliperet kolmapäeval, 2. aprillil kella 10.30–14.30 Delta keskusesse doonoripäevale“ ning kui keelemudel väljastab sündmuse pealkirjaga „Blood Donor Day“, siis loetakse väljastatud sündmust õigeks sündmuseks, sest see viitab sellisele sündmusele, mis e-kirjas olemas on.

Õige kategooria ühe sündmuse puhul tähendab, et keelemudel väljastas sellise kategooria sündmuse jaoks, mis eeldatavas vastuses olemas on. Näiteks kui sündmuse „Blood Donor Day“ puhul on eeldatavateks kategooriateks märgitud „University of Tartu“ ning keelemudel märgib sellele sündmusele kategooriad „University of Tartu“ ning „Hackathon“, siis loetakse esimest kategooriat õigeks ning teist valeks.

Õigete sündmuste leidmise protsendi hindamisreeglid on järgnevad:

- 1) Kui e-kirjas on sündmusi, aga keelemudel väljastab tühja JSON järjendi (s.t, et keelemudel ei leidnud e-kirjast ühtegi sündmust), siis on õigete sündmuste leidmise protsent 0%
- 2) Kui e-kirjas ei ole sündmusi, aga keelemudel väljastab JSON järjendi, milles on vähemalt üks sündmust kirjeldav JSON objekt, siis on õigete sündmuste leidmise protsent 0%
- 3) Ülejäänud juhtudel kasutatakse selle protsendi leidmiseks järgnevat valemit

$$\text{Õigete sündmuste osakaal} = \frac{\text{Keelemudeli leitud õiged sündmused}}{\text{Kõik sündmused e - kirjas}}$$

Sündmuste olemasolul hinnatakse igat keelemudeli väljastatud sündmuse JSON objekti järgnevate reeglitega:

- 1) Võti „event_name“
 - a. 0% kui ei viita õigele sündmusele; 30% kui sündmuse pealkiri on ebamäärane ja ei selgita hästi sündmuse sisu; 100% kui sündmus on õige
 - b. -20% kui on puudu sündmusega seotud info nimes/sündmuse nimi ei ole täpne/sündmuse nimi viitab valele asjale (näiteks registreerumine, alamsündmuse puhul peamise sündmuse nimi, täpsust jääb puudu)
 - c. -10% kui sündmuse nimes on suvalist infot

- d. -20% kui sündmuse pealkirjas on kirjavigu, või see on vales käändes vormistatud
- 2) Võti „start_date“
- a. +33.3% kui kuupäev on õige
 - b. +33.3% kui kellaaeg on õige
 - c. +33.3% kui ajatsoon on õige
- 3) Võti „end_date“ (samad reeglid mis on võtmel „start_date“)
- 4) Võti „country“
- a. 0% kui ei ole õige riik, 100% kui riik on õige
- 5) Võti „city“
- a. 0% kui ei ole õige linn, 100% kui linn on õige
- 6) Võti „address“
- a. 0% kui viidatakse valele aadressile või valele/ebamäärasele hoonele; 50% kui viidatakse õigele hoonele, kuid aadressi ei ole; 100% kui viidatakse õigele aadressile (hoone õigsus sel juhul ei lugenud)
- 7) Võti „tags“
- a. Kui keelemudel genereeris vähem/sama palju kategooriaid, kui oli õiges vastuses, siis leiti täpsus valemiga $\frac{\text{Keelemudeli genereeritud õiged kategooriad}}{\text{Õigete kategooriate arv}}$
 - b. Kui keelemudel genereeris rohkem kategooriaid, kui oli õiges vastuses, siis leiti täpsus valemiga $\frac{\text{Keelemudeli genereeritud õiged kategooriad}}{\text{Keelemudeli kõik genereeritud kategooriad}}$

IV. Peenhäälestamise programm

Peenhäälestamise programmi „finetune.py“ sisu on nähtav lingil

<https://github.com/madiskoivopuu/EventsOrganiser/blob/main/misc/finetune/finetune.py>

V. Peenhäälestatud keelemudeli failid

Peenhäälestatud keelemudel on kokku pakitud ZIP faili, milles on nii „safetensors“ formaadis mudel kui ka „gguf“ formaadis mudel. ZIP fail on alla laetav järgnevalt lingilt: https://drive.google.com/file/d/1V4q5zyGVZPhTF_dNXR4KiuToPyv1bEeG/view?usp=sharing

VI. Rakenduse lähtekood

Rakenduse lähtekood on üles laetud Githubi repositooriumi, mis asub lingil <https://github.com/madiskoivopuu/EventsOrganiser>

VII. Rakenduse paigaldusjuhend

Paigaldusjuhend on kättesaadav järgneval lingil

<https://github.com/madiskoivopuu/EventsOrganiser/blob/main/Paigaldusjuhend.pdf>

Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Madis Kõivopuu**,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose „**Rakenduse loomine e-kirjadest sündmuste otsimiseks ja organiseerimiseks**“, mille juhendajad on **Pelle Jakovits** ning **Chinmaya Kumar Dehury**, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada Tartu Ülikooli digitaalarhiivi kuni autoriõiguse kehtivuse lõppemiseni;
2. annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi kaudu Creative Commons litsentsiga CC BY NC ND 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni;
3. olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile;
4. kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Madis Kõivopuu

14.05.2025