TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Rasmus Meos

Õppimist toetav veebirakendus kitarrimängijatele

Bakalaureusetöö (9 EAP)

Juhendaja: Sven Aller

Tartu 2025

Õppimist toetav veebirakendus kitarrimängijatele

Lühikokkuvõte:

Bakalaureusetöö eesmärk oli luua veebirakendus, mis aitab kitarrimängijatel leida teoseid, mis sobivad nende oskustasemega. Kasutaja saab sisestada artisti nime, mille põhjal kuvatakse valik teoseid koos sobivusprotsendiga – see arvutatakse lähtudes akordidest, mida kasutaja on eelnevalt märkinud kui juba õpitud. Töös antakse ülevaade muusikateooria olulisematest mõistetest, sealhulgas akordide ehitusest, transponeerimisest ja lihtsustamisest. Samuti käsitletakse veebikoorimise tehnikaid, mida kasutati akordilehtede hankimiseks, ning tutvustatakse rakenduse realiseerimiseks valitud tehnoloogiaid ja süsteemi arhitektuuri. Kirjeldatakse ka akorditöötluseks loodud algoritme, sealhulgas helistiku tuvastamise, akordide transponeerimise ja lihtsustamise loogikat. Töö lõpus esitatakse mõõtmistulemused, kasutajate tagasiside ja võimalikud edasiarendusvõimalused.

Võtmesõnad:

Muusika, akordid, veebirakendus, transpositsioon, veebikoorimine

CERCS: P170 Arvutiteadus, arvanalüüs, süsteemid, kontrol

An Educational Web App for Guitar Players

Abstract:

The aim of this Bachelor's thesis was to develop a web application that helps guitar players find songs that match their skill level. The user can enter the name of an artist, after which the system displays a list of songs along with a suitability percentage—calculated based on the chords the user has previously marked as learned. The thesis provides an overview of key music theory concepts, including chord structure, transposition, and simplification. It also discusses web scraping techniques used to collect chord sheets, and introduces the technologies and system architecture selected for implementing the application. The thesis further describes the custom-built chord processing algorithms, including key detection, chord transposition, and simplification logic. The final chapters present measurement results, user feedback, and potential opportunities for further development.

Keywords:

Music, chords, web application, transposition, web scraping

CERCS: P170 Computer science, numerical analysis, systems, control

Sisukord

Sissejuhatus

Kitarrimängu populaarsus nii harrastajate kui ka professionaalide seas on püsinud stabiilsena läbi aastakümnete. Õppijate hulk, kes kasutavad veebipõhiseid vahendeid akordide harjutamiseks ja repertuaari avastamiseks, on üha kasvamas. Samal ajal jääb paljudele alustavatele mängijatele takistuseks keerukate akordide valdamatus või sobiva helistiku leidmine, mis vastaks nende vokaalsele või tehnilisele võimekusele. Nendest vajadustest lähtudes on lõputöö eesmärk luua isikupärastatud veebirakendus, mis toetab mängijat akordide tuvastamisel, transponeerimisel, lihtsustamisel ning teoste valikul vastavalt kasutaja oskustele.

Töö keskendub teoreetiliste teadmiste, näiteks akordide struktuuri, helistike ülesehituse ja veebikoorimise põhimõtete rakendamisele tarkvaraarenduse kontekstis. Lõputöö tulemusena valmis veebirakendus, mis võimaldab kasutajal sisestada artisti nime, mille alusel kraabitakse akordilehti, töödeldakse need automaatselt ning hinnatakse, kuivõrd hästi need sobivad kasutaja poolt märgitud tuttavate akordidega. Töö sisaldab ka muusikateoreetilist tausta, sh akorditüübid ja transpositsioon, mis on otseselt seotud loodava süsteemi funktsionaalsusega.

Kuna varasemad rakendused ei toeta muusikaliste teoste filtreerimist mängija oskuste põhjal, pakub loodud veebirakendus sellele lahendust. Samuti toetab töö laiemat eesmärki – muuta kitarrimängu õppimine interaktiivsemaks ja aktraktiivsemaks.

Uuritav probleem seisneb selles, kuidas akordipõhine repertuaari soovitussüsteem võiks sobituda erineva tasemega kitarrimängijate vajadustega. Töö raames püstitati uurimisküsimus: *kuidas saab automaatne helistiku tuvastus ja akordide analüüs aidata isikupärastada kitarrimängijale soovitatud lugusid?* Vastuse leidmiseks viidi läbi veebirakenduse arendusprotsess, mille käigus ühendati muusikateooria, andmetekorje veebist ja algoritmiline töötlus.

Töö koosneb neljast peatükist. Esimeses peatükis käsitletakse muusikateooria aluseid, sh akordide ehitust, transponeerimist ja akordide lihtsustamist. Teine peatükk tutvustab veebikoorimise põhimõtteid, ajalugu, tööriistu ning juriidilisi ja tehnilisi kaalutlusi. Kolmas peatükk keskendub loodud veebirakenduse arendusele, tehnoloogiatele, arhitektuurile ja algoritmidele. Neljas peatükk esitab mõõtmistulemused, kasutajate tagasiside ning toob välja edasised arendusvõimalused.

1. Muusikateooria

1.1 Akordi ehitus ja heliread

Akordiks, muusikateoorias, nimetatakse vähemalt kolme erineva fikseeritud helikõrguse üheaegset kõlamist. See tähendab, et helikõrguste vahe peab olema määratud kindlate intervallidega. Intervalliks nimetatakse kahe helikõrguse korraga kõlamist [1]. Selleks, et määratleda intervalli vahemik, kasutatakse näiteks mõõtühikuid toon ja pooltoon. Pooltoon määrab vähima helikõrguse erinevuse kahe noodi vahel [26].

Kolmkõlad on akordide lihtsaimad vormid ning omavad olulist rolli muusikateoorias. Kolmkõla kujuneb kolmest noodist, mis paiknevad üksteisest tertsi kaugusel. Põhinoot annab akordile nime, samas kui tertsid määravad akordi üldise olemuse – on see kas mažoorkolmkõla (duurid), minoorkolmkõla (mollid), vähendatud kolmkõla või suurendatud kolmkõla [1].

Järgnevalt on toodud erinevate kolmkõlade ülesehitus:

- Mažoorkolmkõla moodustub põhinoodist, millele järgneb suur terts (4 pooltooni) ja seejärel väike terts (3 pooltooni). Näiteks C-duur koosneb nootidest C, E ja G.
- Minoorkolmkõla moodustub põhinoodist, millele järgneb väike terts ja seejärel suur terts. Näiteks C-moll koosneb nootidest C, D# ja G.
- Vähendatud kolmkõla koosneb põhinoodist ja kahest järjestikusest väiksest tertsist. Näiteks Cdim koosneb nootidest C, D# ja F#.
- Suurendatud kolmkõla koosneb põhinoodist ja kahest järjestikusest suurest tertsist. Näiteks Caug koosneb nootidest C, E ja G#.

Helireaks nimetatakse järjestatud helikõrguste kogumit mingi helisüsteemi piires, järgides kindlaid intervalle. Selles töös keskendutakse peamiselt diatoonilistele heliridadele – see tähendab heliridadele, mis koosnevad seitsmest erinevast noodist ning järgivad kindlat mustrit täis- ja pooltoonide järjestamisel. Diatoonilisteks heliridadeks loetakse nii loomulike mažoorseid kui ka minoorseid heliridu. Näiteks C-duur heliredel järgib mustrit toon, toon, pooltoon, toon, toon, toon, pooltoon. Mažoorseid heliridu harmoneerides tekib kindel akordimuster: kolm mažoorakordi (I, IV, V), kolm minoorakordi (ii, iii, vi) ning üks vähendatud kolmkõla akord (vii°). Need moodustavad paljude tuntud akordide järgnevuse

(ingl *chord progression*) aluse, näiteks I-IV-V, mis kõlab kuulsa The Beatlesi loo "Let It Be" harmoonias [2].

1.2 Transponeerimine

Transponeerimine võimaldab teose helistikku tõsta või langetada nii, et mängija saab kasutada mugavamaid võtteid ja laulja püsib sobivas hääleulatuses.

Transponeerimise aluseks peetakse intervalli, sest iga akord tuleb teisendada üles- või allapoole täpselt sama hulga pooltoone, säilitades sealjuures akordi liigi; seega peab kitarrist arvestama nii nihke suurust kui ka seda, et akordi kuju säiliks [23].

Praktikas aitab segadust ära hoida relatiivne astmetähistus: kui akordide järgnevus kirjutatakse rooma numbritega, saab sama skeemi kohe teisaldada mistahes helistikku, sest numbrid jäävad püsima ka siis, kui noodinimed muutuvad [24]. Kapo (ingl *capo*) kasutamine põhineb samal loogikal: kui seade kinnitatakse kitarrikaelal näiteks kolmandale astmele, kõlab avatud D-duur kolmkõla tegelikult F-duur kolmkõlana, mistõttu peab mängija sellega arvestama, et korrektset helistikku säilitada [24].

Keerukamates teostes esineb ka modulatsioone, kus kogu akordide järjestus kogu teost transponeeritakse järjest kõrgemale, et luua dramaatiline kulminatsioon. Üheks näiteks on lugu My Girl, mille refrään tõuseb kahe pooltooni võrra ja säilitab sama akordimustri uues võtmes [25]. Kitarril tähendab see akordide kollektiivset nihutamist vastava pooltoonide arvu võrra ning on hõlpsasti järgitav, kui alg-/sihthelistiku suhe on teada [25].

Levinud vead tulenevad peaasjalikult intervalli valesti arvestamisest või kapo asetamise unustamisest, mis viib kogu ansambli nt pooltooni võrra lahku; samuti kiputakse muutma akordi kuju, kui transponeeritakse ainult põhinoot. [23].

Õigesti rakendatuna annab transponeerimine kitarristile paindlikkuse kohandada lugusid eri häälekõrgustele, vältida ebamugavaid barre-akorde ja värskendada harmoonilist kõlapilti.

1.3 Akordide lihtsustamine

Algajatele kitarrimängijatele on akordide lihtsustamine hädavajalik. Tehniliselt keerukad akordid, nt „klassikaline" F või variatsioonid Dmaj7-st, võivad õppimisel ja harjutamisel olla esialgu liialt vaevarikkad. Selliste akordide lihtsustamine (näiteks asendades Dmaj7 lihtsa D-duur akordiga) muudab nende mängimise kergemaks, kuigi peab teadvustama, et sellisel juhul ei peegelda laulu üldine harmooniline alus enam esialgset. Nagu Jon Green on märkinud, jagavad paljud akordid ühist põhistruktuuri – kolme põhinoodi kooslust. See võimaldab kasutada osalisi akordivorme või lihtsustatud variante, mis kõlavad enamikus muusikalistes kontekstides siiski terviklikult [3].

Charlotte Adams soovitab keerulisemate akordide puhul välja jätta näiteks akordi seitsmenda või üheksanda astme, et lugude mängimine ladusamini läheks, säilitades samal ajal harmoonilise üldpildi [4]. Näiteks võib D9 asemel mängida lihtsalt D-duuri, mis hoiab alles laulu olemuse, aga vähendab tehnilist keerukust.

Lihtsustades keerukaid akordivorme, püüab loodav veebirakendus suurendada repertuaari valikut, mida algajad kitarrimängijad saavad harjutada. Näiteks kui kasutaja täiendab enda oskustepagasit akordiga D (ent mitte selle keerukamate variantidega), lisab süsteem repertuaari ka palad, kus esinevad Dmaj7 või D9, kuid lihtsustatud kujul. Tuleb siiski nentida, et liigse lihtsustamisega hoogu minemine võib teatud žanrites (näiteks džässis) viia oluliste nüansside kadumiseni.

1.4 Teoreetiliste teadmiste roll kitarrimängimise arengul

Teoreetilised teadmised muusikast aitavad täiendada kitarrimängija oskusi ja laiendada muusikalist silmaringi. Need pakuvad kitarrientusiastile suurepärast võimalust rikastada oma repertuaari uute tehnikate ja stiilidega. Algajate puhul ei pruugi need teadmised aga alati olla esmatähtsad – sageli alustatakse põhiliste oskuste omandamisest, liikumata kohe keerulise teooria juurde. Selline järkjärguline lähenemine võib kitarrimängijat innustada, sest saavutatud esmane kompetents loob kindlustunde ja motiveerib edasi õppima.

Järgnevad kaks lõiku põhinevad autori kogemustel:

Tüüpiline õpiteekond võib alata lihtsamate akordivormide ja kergemate laulude mängimisega, keskendudes esmalt rütmimustritele (ingl *strumming patterns*), mis loovad tugeva vundamendi edasiseks arenguks. Järgmisena omandatakse *power chords*, et rokilugudele kaasa mängida, ja pentatoonika, mis võimaldab lihtsamaid soolopartiisid esitada. Kui baasteadmised on omandatud, hakatakse nt ka kapot kasutama, et kohandada lugusid vastavalt hääleulatusele ja mänguoskustele. Edasi katsetatakse erinevate häälestustega, et saavutada võimalikult autentne kõlapilt.

Oskuste arenedes hakatakse huvi tundma ka keerukamate kontseptsioonide vastu, nagu näiteks transponeerimine ja arpedžod. Arpedžo-võte, kus akordi noodid mängitakse järjestikku, lisavad kitarrimängule meloodilist dünaamikat ja võimaldavad loovamat väljendust.

Sarnane sammsammuline lähenemine ilmneb Kyle'i džässiakordide õppimise metoodikas, mis rõhutab, et õppimist tuleb lihtsustada, ja etappideks jaotada, vältides algajate ülekoormamist [5]. Autori kogemusega kooskõlas näitab Kyle'i meetod, kuidas väikeste sammudega saab saavutada sügavama arusaamise akordidest ja nende variatsioonidest.

Teoreetiliste teadmiste – nagu helistike struktuur, akordide ülesehitus ja transpositsioon – rakendamine praktilises kontekstis võimaldab kohandada repertuaari mängija oskustele vastavaks. Keerukate akordide lihtsustamine ja lugude transponeerimine sobivasse helistikku muudab muusika mängimise ja õppimise kättesaadavamaks ning toetab mängija arengut.

Akordide tundmine aitab kitarrimängijal paremini mõista muusika struktuuri, kohandada repertuaari oma oskuste tasemele ning arendada muusikalist väljendusvõimet. Selles peatükis käsitleti akordide ülesehitust, diatooniliste heliridade struktuuri, transpositsiooni põhimõtteid ja akordide lihtsustamist. Lisaks kirjeldati, kuidas teoreetilised teadmised toetavad järkjärgulist arengut kitarrimängu õppimisel. Järgmises peatükis kirjeldatakse andmete kogumise metoodikat veebilehtedelt, mida loodav veebirakendus rakendab.

2. Veebikoorimine

Veebikoorimiseks (ingl *web scraping*) nimetatakse protsessi, mille eesmärk on veebilehelt andmeid koguda. Veebikoorimist sooritavad enamasti autonoomsed programmid (ingl *bot*) ehk robotid, mis saadavad selleks päringu veebilehte hoiustavale serverile ning analüüsivad vastuseks saadud HTML-i dokumenti, et sealt sobivaid andmeid välja võtta [6].

2.1 Veebikoorimise ajalugu ja kasutus

Veebikoorimine kujunes välja koos veebi loomisega 1989. aastal. Kuigi esialgu oli veebilehtede URL-ide kogumine ja kategoriseerimine pelgalt veebi administraatorite pärusmaa, arenesid esimesed veebikoorijad välja juba 1993. aastal. Nendeks olid World Wide Web Wanderer, mille eesmärk oli indekseerida uusi veebilehtesid, ning JumpStation, mis toimis esimese otsingumootorina [7, 8]. 2000. aastate alguses hakkasid esimesed veebilehed teenusena pakkuma ka avalikke API-sid, mis võimaldasid andmete ametlikku ülekandmist. Selle uue tehnoloogia teerajajad olid Salesforce ja eBay [7]. API-de kasutuselevõtt vähendas mõnevõrra vajadust veebikoorijate järgi.

Veebikoorimine on kasutusel paljudes valdkondades. Peamiselt rakendatakse seda hinnakontrolliks ja -võrdluseks; näiteks külastab veebikoorija mitmeid e-poodide veebilehti, et koguda igaühelt tootekirjeldusi ja hindasid [6, 8]. Samuti kasutatakse veebikoorimist turu-uuringutes ja ärianalüütikas, näiteks foorumite ja sotsiaalmeedia andmete analüüsimiseks [6, 9]. Lisaks saab veebikoorijaid rakendada kinnisvaraturu jälgimiseks, ilmaandmete kogumiseks, ja otsingumootorite indekseerimiseks [9].

2.2 Veebikoorimise kategooriad

Veebikoorimiseks leidub mitmeid tehnikaid ja tööriistu. Üheks peamiseks kategooriaks on HTML-i parsimiseks mõeldud teegid. Nagu nimigi viitab, parsitakse nende abil alla laetud HTML-faile, et eraldada vajalikud andmed. Ühed populaarsemad Pythoni teegid on Beautiful Soup ja lxml, mis konstrueerivad analüüsitavast HTML/XML-faili struktuurist parsimispuu [10]. Beautiful Soup on kõrgema taseme teek, mis on arendatud parserite (nt lxml või html.parser) peale. See tuleb toime ka halvasti vormindatud HTML-iga ning sisaldab

kasulikke meetodeid, et HTML-puud läbida ja sellest andmeid otsida [11]. Seevastu lxml on madalama taseme parser, mis on kirjutatud C keeles, olles seega kiire, mälusäästlik ja tõhus. Seetõttu on lxml etem töötlemaks suuri või keerulisi veebilehti [10].

Teine, laialt levinud kategooria on veebikoorimise raamistikud. Üks tuntumaid raamistikke on Scrapy, mis hõlmab endas allalaadimise, parsimise ja koorimise funktsionaalsust. Ta suudab saata samaaegselt saabunud päringuid asünkroonselt, käsitseda korduskatseid ja lehekülje numeratsiooni (ingl *pagination*), ning tegutseda veebiämblikuna (ingl *web crawling*). Scrapy on seega sobiv suuremahuliste või keeruliste projektide realiseerimiseks, kuid seetõttu ka pikema õpiajaga [12]. Seevastu eelmainitud teeki, Beautiful Soupi, ei saa pidada täisväärtuslikuks veebiämblikuks, sest ei suuda ise päringuid saata [11].

Järgmise kategooria alla käivad erinevad automatiseeritud brauseri tööriistad. Need on eelkõige mõeldud töötlemaks dünaamilisi veebilehti, mis kasutavad suurel määral JavaScripti. Selenium ja Puppeteer on ühed populaarsemad brauseri tööriistad, toimides graafilise liideseta brauseritena ja võimaldades neil renderdada veebilehti ning nendega interakteeruda – nagu teeks seda harilik brauser. Puppeteer on kõige efektiivsem Chromium-baasil loodud brauseritega, samas kui Selenium toetab erinevaid brausereid [13].

API-põhine koorimine on samuti levinud kategooria. Üldjuhul, kui veebilehed pakuvad ametlikku API teenust, siis on selle kasutus soovitatav. Sel juhul ei pea HTML faili parsima ning koorija saab saata päringu API otspunktile, mille vastuseks saadakse struktueeritud andmed kas JSON või XML formaadis. Selline toimimine on tihtipeale efektiivsem, sest andmed on eelvormindatud ning tõrgete tekkimine ebatõenäolisem. Siiski võivad API pakkujad jagada vaid piiratud informatsiooni, mida oleks võimalik veebikoorimise teel enam hankida. Parimaks lähenemiseks peetakse siiski API olemasolul selle kasutamist, sest muutused veebilehe struktuuris võivad põhjustada veebikoorija soovitud töövoo lakkamise [14].

2.3 Eetilised ja juriidilised kaalutlused

Veebikoorimise puhul tuleb arvestada mitmete kirjutatud ja kirjutamata reeglitega, mis reguleerivad selle lubatavust ja sobivust. Eetiline veebikoorimine algab veebilehe *robots.txt* faili ja kasutustingimuste järgimisest. Tekstifail *robots.txt* on avalikult kättesaadav fail, mida veebilehe omanikud kasutavad, ning mille järgi peaksid veebikoorijaid juhinduma [6]. Seal on kirjas, milliseid lehti võib koorida ja millistest lehtedest peaks hoiduma. Kuigi failis

märgitud reeglistik ei ole õiguslikult siduv ning seal kirjeldatust kinnipidamine on vabatahtlik, peetakse siiski sündsaks juhiseid järgida. Samuti tuleks enne veebikoorimist kurssi viia ennast kasutustingimustega, sest leidub veebilehti, kes seavad selgesõnalised piirangud või keelavad igasuguse veebikoorimise [15].

Veebikoorimise seaduslikkus on tihti kompleksne ning sõltub suuresti vastavast jurisdiktsioonist. Üldiselt ei ole avalike andmete koorimine illegaalne, kuid teatud tegevused võivad esile kutsuda õiguslike meetmete võtmise. Näiteks Ameerika Ühendriikides esitatakse enim kaebusi veebikoorimise vastu, kui esineb autoriõiguse rikkumisi (kraabitakse autoriõigusega kaitstud materjale), kui on rikutud Computer Fraud and Abuse Act (CFAA) seadust (andmeid on kraabitud volitatud juurdepääsuta), või kui on toimunud vallasvarasse loata sekkumine (nt serveri ülekoormamine) [16]. Kohtuotsuseid on mõnikord langetatud andmevaldajate kasuks: 2000. aastate algul oli menetlusel mitu kohtujuhtumit, kus lennufirmad esitasid kaebuse lennupileti hindade kogujate vastu, sest viimased koorisid lennuandmeid. ÕKuid õigusmaastik selles vallas on sestsaadik edasi arenenud. Märkimisväärne kohtuasi leidis aset 2022. aastal LinkedIni ja hiQ Labsi vahel, kus Ameerika Ühendriikide apellatsioonikohus kinnitas, et avalikult kättesaadavate andmete koorimine ei lähe vastuollu CFAA-s sätestatuga. See tähendab, et kui andmetele eksisteerib ligipääs ilma sisselogimiseta, siis on veebikoorimine lubatud [18]. Kuigi kohtuotsus oli positiivse tulemusega eelkõige ajakirjanike ja arhivaaride jaoks, ei saa eeldada, et sellest tuleneb *carte blanche* – seadused seoses autoriõiguste või privaatsusega on endiselt kehtivad. Euroopa Liidus võib omavoliline veebikoorimine siiski vastuollu minna Isikuandmete kaitse üldmäärusega (ingl *General Data Protection Regulation* ehk GDPR); eriti kui eesmärgiks on isikuandmete kogumine või kui arvestatav osa andmebaasi andmestikust ekstraheeritakse [19].

Vastutustundlik veebikoorija seega jälgib, et sihikule võetud veebilehtedele põhjustatud potentsiaalne kahju oleks minimaalne ning veendub, et austatakse andmete omandiõigust. Suuremahuline veebikoorimine võib oma sagedaste päringute tõttu põhjustada serverite ülekoormust, mis vähendab teenuse kvaliteeti teiste klientide jaoks. Seetõttu soovitatakse rakendada erinevad piirangud: viivitused kahe järjestikuse päringu vahel ning koorimisest hoidumine veebiliikluse tippajal. Samuti on kasulik seada juhuslikud pausid programmitöös, et matkida inimbrausimist, vältides sellega *anti-bot* süsteemide käivitumist [20]. Lisaks tasub anda veebilehe administraatoritele võimalus ennast identifitseerida, et nad teaksid, kes nende saiti külastab. Seda saab teha edastatavasse päringusse päise *User-Agent* lisamisega. Selleks,

et veebikoorimine oleks eetiline ja stabiilne, tasub teenuste olemasolul kasutada ametlikke andmevooge või API-sid [21].

2.4 Tehnilised keerukused ja lahendused

Üheks peamiseks takistuseks efektiivsel veebikoorimisel on asjaolu, et paljud moodsad veebilehed on dünaamilised – need esitavad pärast esialgse HTML-i laadimist kliendipoolselt JavaScripti abil lisapäringuid, et täiendada lehe funktsionaalsust, näiteks interaktiivsete kaartide või sotsiaalmeedia sisu laadimise näol. Veebikoorija, mis kasutab kõigest lihtsaid HTTP-päringuid, võib vastuseks saada peaaegu tühja või puuduliku lehe, sest veebilehe sisu genereeritakse skriptide abil. Selleks, et modernseid veebilehti tõhusalt koorida, peetakse tulemuslikuks resultatiivseks lahenduseks graafilise liideseta brauserite kasutamist. Need suudavad renderdada täieliku veebilehe, võimaldades veebikoorijal ligipääsu uuendatud DOM-ile, mis sisaldab terviklikku sisu [14]. Siiski leidub ka alternatiivne meetod – veebilehe esitatud päringute pöördprojekteerimine (ingl *reverse engineering*). Kuna teinekord on veebilehe päring kolmanda osapoole andmetele esitatud XHR abil või *fetch*-funktsiooni kutse tagastab JSON-formaadis vastuse, saab veebikoorija otse nendele otspunktidele päringu esitada, leides selleks peidetud API.

Teiseks laialt levinud takistuseks on IP-aadresside blokeerimine ning kiiruse piiramine (ingl *rate limiting*). Veebilehed on tihti kasutusele võtnud meetmed, et jälgida veebiliiklust ja leida koorimisele viitavaid mustreid. Lühikese aja jooksul esitatud suur kogus päringuid, mis pärinevad samalt IP-aadressilt, käivitavad tihtipeale veebikoorimise vastased meetmed ning blokeerivad saatja IP-aadressi või paluvad lahendada CAPTCHA-sid. Üldlevinud vastumeetmeks on puhverserverite kasutamine, mis võimaldavad päringuid jaotada mitmete IP-aadresside vahel, võimaldades selle läbi hoiduda lihtsamatest IP-põhistest blokeerimistest. Paljud veebikoorijad on kasutusele võtnud puhverserverite kogumid (ingl *proxy pools*) või rotateeruvad puhverserveri teenused nii, et iga esitatud päring saabuks sihtveebisaidile erinevalt IP-aadressilt; eelistatult ka erinevatest geograafilistest asupaikadest. Viivituste implementeerimine ja *crawl-delay* direktiivist kinnipidamine *robots.txt* failis, on samuti soovituslik, sest ennetab serveri ülekoormamist. Nende meetmete kasutusele võtmine pole mitte ainult eetiline, vaid ka praktiline, sest vähendab veebikoorimise tuvastamise tõenäosust [22].

Keerulisemad veebikoorimisvastased süsteemid rakendavad rohkem meetmeid kui pelgalt IP-põhine blokeerimine – nad moodustavad seansi analüüsimisel kliendist sõrmejälje (ingl *fingerprinting*). Selle käigus otsitakse mustreid, mis vihjavad automatiseeritud interaktsioonile. Samuti kontrollitakse, kas kliendil jookseb JavaScript, kas ta omab teatud küpsiseid, või kas teostab inimesele omaseid liigutusi (hiire liikumine ja klaviatuuri sisendid). Selle vastu saab rakendada mitmeid eelmainitud meetmeid nagu graafikaliideseta brauserite kasutamine õigete sätetega (JavaScripti kasutamine ja küpsiste aktsepteerimine). Lisaks on kasulik modifitseerida veebilehele saadetavat päringut, nimelt eelnimetatud *User-Agent* päist, andes sellele juhusliku sobiva sõne väärtuse. Paljud veebikoorijad (nt Pythoni teek urllib või curl) saadavad vaikimisi *User-Agent* päise, mille väärtust on sihtveebilehtedel lihtne tuvastada ning seeläbi päringust keelduda. Seetõttu kasutatakse kogumit modernsete brauserite *User-Agent* sõne väärtustest, mida perioodiliselt muutes on veebikoorijal võimalik paremini võrguliiklusesse sulanduda. Kasutatakse teisigi abinõusid nagu *Referer* päise lisamine, CAPTCHA-de lahendamine või nendest möödumine, ning vajadusel isegi osapoolte vahelise suhtluse võltsimine. CAPTCHA-de lahendamine kolmanda osapoole teenuste osapoolte poolt on samuti levinud meede, kus neid reaalajas lahendavad kas inimesed või AI-lahendused. Tegelikkuses kasutabki edukas veebikoorija erinevaid võtteid, ja veebikoorijate ning vastaspoole vahel on käimas aktiivne “võidurelvastumine”, sest igale *anti-bot* meetmele leitakse varem või hiljem vastumeede [22].

3. Veebirakenduse Guitar App arendus

See peatükk keskendub loodud veebirakenduse arenduse protsessile, tehnoloogilistele valikutele ning ülesehitusele. Esmalt antakse ülevaade sarnastest olemasolevatest lahendustest, et tuua välja, milliseid funktsionaalsusi need pakuvad ja millised puudujäägid õigustasid uue rakenduse loomist. Seejärel käsitletakse kasutatud tööriistu ja raamistikke, koos valikute põhjendustega. Edasi tutvustatakse projekti struktuuri ning varajasi katsetusi ja keerukusi, mis ilmnesid arenduse esimestes etappides. Peatüki lõpetab põhjalik kirjeldus veebirakenduse arhitektuurist, mis selgitab, kuidas erinevad moodulid omavahel suhtlevad ning kuidas toimub andmete liikumine kasutaja päringust lõpliku vaate kuvamiseni.

3.1 Sarnased rakendused

Selles uurimistöös loodava veebirakenduse arenduse esimeseks etapiks oli välja selgitada, millised lahendused juba eksisteerivad ning milliseid funktsionaalsuseid need pakuvad. Järgnevalt tuuakse välja tuntumad veebirakendused, mis võimaldavad kitarriakordide õppimist ja lugudele kaasa mängimist.

Ultimate Guitar pakub ulatuslikku kogumikku kasutajate poolt üles laetud tabulatuuridest. Rakendus toetab erinevate akordimustrite kuvamist, lugude transponeerimist ja salvestamist ning kasutajaks registreerimisel õppevideoga tutvumist. Lisaks annab veebirakendus loole kategoorilise keerukuse hinnangu ning võimaldab akorde filtreerida kas diees või bemoll notatsiooni alusel. Ultimate Guitar ei paku võimalust salvestada juba õpitud akorde ega saada nende põhjal sobivuse hinnanguid otsitud lugudele. Samuti puudub võimalus akorde lihtsustada.

Chordify on veebirakendus, mis võimaldab kasutajatel artisti või loo järgi YouTube'i videot otsida ning sellega interaktiivselt kaasa mängida. Rakendus tuvastab automaatselt loos esinevad akordid ja helistiku ning kuvab neid kasutajale reaalajas. Samuti võimaldab Chordify lugu transponeerida, lihtsustada, tempot muuta ning kapoga kasutada, kuid paljud nendest lisasätetest on tasulised. Veebirakendus ei paku võimalust salvestada juba õpitud akorde ega nende põhjal sobivuse hinnanguid saada, kuid sellist isikupärastatud lähenemist ei ole ilmselgelt rakenduse sihtotstarbeks ette nähtud.

Songsterr on veebirakendus, mis võimaldab kasutajal sobivat lugu otsida üle miljoni tabulatuuri seast. Seejärel saab kasutaja koos tabulatuuriga interaktiivselt YouTube'i videole

(või akordilehel iseseisvalt) kaasa mängida. Rakendus võimaldab loo tabulatuuri lehel tempot ja helikõrgust muuta. Valdav enamus funktsionaalsustest on siiski tasulised, sealhulgas tabulatuuriga katkematult kaasa mängimine. Rakendus ei võimalda salvestada juba õpitud akorde ega nende põhjal otsitavate lugude sobivuse hinnanguid saada.
Kuigi need veebirakendused pakuvad kasutajatele mitmekesist rakendust akordide õppimiseks ja lugudele kaasa mängimiseks, ei keskendu ükski neist kasutaja individuaalsele oskustasemele.

3.2 Veebirakenduse kirjeldus ja nõuded

Loodav veebirakendus on mõeldud kitarriõppijatele, et võimaldada neil otsida teoseid, milles leidub akorde, mida nad juba mängida oskavad. Kasutaja saab sisestada huvipakkuva artisti nime ning näha loeteluna selle artisti laule, mille akordilehed on veebilehelt azchords.com leitud ning seejärel analüüsitud. Iga loetelus oleva repertuaari jaoks kuvatakse, kui hästi sobitub see õppuri oskustasemega ning kasutajal on võimalik teose akordilehel transponeerida helistikku, lihtsustada akorde või märkida lugu lemmikute hulka. Samuti võimaldab veebirakendus igale repertuaarile lisatud YouTube'i videole kaasa mängida.
Veebirakendus eeldab kasutajaks registreerumist ja võimaldab eraldi lehtedel hallata tuttavaid akorde ja lemmikuks märgitud repertuaare.
Järgnevates loeteludes on välja toodud veebirakenduse nõuded.

Serveripoolne töötlus peab tagama järgmise:

1. Rakendus peab suutma töökindlalt akordilehti artisti/bändi kohta veebilehelt azchords.com koorida.
2. Akordilehe põhjal tuleb tuvastada kõik akordid ning analüüsida iga akordi esinemissagedust ning märkida iga akordi paiknevus akordilehel.
3. Rakendus peab suutma tuvastada repertuaari helistikku.
4. Rakendus peab võimaldama akordilehe transponeerimist ning akordide lihtsustamist, säilitades akordilehe vormingu.
5. Rakendus peab arvutama kasutajale tuttavate akordide põhjal repertuaari sobivuse protsendi ning kuvama akordid, mida oleks vaja juurde õppida.
6. Artisti otsingu päringule vastamiseks ei tohiks kuluda rohkem kui 15 sekundit.

Kasutajaliidese kaudu peab olema võimalik:

1. Luua konto, logida sisse ning enda profiili hallata.
2. Valida tuntud akorde erinevatest kategooriatest (nt duur, moll, dominantseptakord).
3. Sisestada artisti või bändi nime ning saada vastuseks valik artisti/bändi repertuaaridest.
4. Iga repertuaari kohta kuvada pealkiri, artisti nimi, sobivuse protsent ning nimekiri akordidest, mis tuleks kasutajal ära õppida, et sobivus oleks 100%. Lisaks peab olema võimalus repertuaar lemmikute hulka märkida.
5. Repertuaari lehel kuvada kasutajale pealkiri, tuvastatud helistik, lisatud YouTube'i video ning akordileht. Lisaks peab olema kasutajal võimalik repertuaari transponeerida, akorde lihtsustada ning repertuaar lemmikute hulka märkida.
6. Vaadata eraldi lehte „Minu lemmikud", kus kuvatakse loetelu lemmikuks märgitud repertuaaridest.

Need nõuded seavad raamistiku nii rakenduse funktsionaalsusele kui ka kasutuskogemusele. Järgnev alapeatükk kirjeldab tehnoloogilisi valikuid, mille abil seed eesmärgid saavutati.

3.3 Tehnoloogilised valikud

Järgmiseks etapiks veebirakenduse arendamisel oli sobivate tehnoloogiate valik, mis oleksid jõudluselt efektiivsed, kergesti rakendatavad ning kooskõlas plaanitavate funktsionaalsuste ja arhitektuuriga. Järgnevalt on toodud valitud tehnoloogiad koos nende valiku põhjendustega ning kaalutud alternatiividega.

Veebirakenduse serveripoolseks programmeerimiskeeleks valiti Python, sest see oli autorile varasemalt tuttav ning pakkus toetust rakenduses plaanitud funktsionaalsuste realiseerimiseks, kasutades selleks osaliselt kolmanda osapoole teeke. Näitena saab tuua beautifulsoup4, mis on mõeldud veebikoorimiseks, ning pychord, mida kasutatakse akordide transponeerimiseks. Alternatiivina kaaluti JavaScripti kasutamist koos Node.js käitussüsteemiga (*runtime environment*), kuid Python osutus sobivamaks tänu oma kasutusmugavusele ning planeeritud veebiraamistikule.

Veebirakenduse loomiseks valiti veebiraamistik Flask, mis on Pythoni kergekaaluline mikroraamistik. Flaski kasuks otsus põhines suuresti selle seadistamise lihtsusel ning

omadusel, et Flask võimaldab kiiret prototüüpimist, mis annab võimaluse vajadusel rakenduse arhitektuuri kohendada. Alternatiivina vaadati veel Pythoni veebiraamistikke Django ja FastAPI. Kuigi Django on võimas ning paljude sisseehitatud komponentidega, ei olnud selle kasutuselevõtt plaanitava veebirakenduse mastaapi ja keerukust hinnates sobilik. FastAPI veebiraamistik on modernne ja hea jõudlusega, kuid lõplik valik langes siiski Flaski kasuks tänu olemasolevale arenduskogemusele.

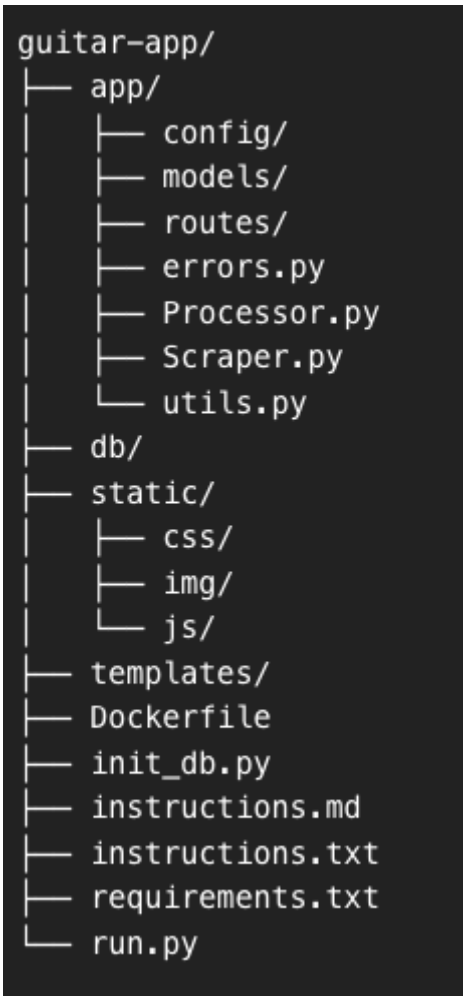
Andmete salvestamiseks ja hoiustamiseks valiti SQLite, mis on failipõhine ja kergekaaluline andmebaas. Selle suurim eelis on lihtne seadistus – SQLite ei nõua eraldi andmebaasiserverit, mis muudab arenduse ja testimise oluliselt mugavamaks. SQLite sobib hästi väiksematele rakendustele, kus andmemaht ja kasutajate arv on piiratud. Alternatiivina kaaluti PostgreSQL integreerimist, kuid selle kasutamine oleks veebirakenduse ressursimahukamaks teinud ning potentsiaalsete kasutajate arvu arvestades kaheldava lisaväärtusega.

Kasutajaliidese (*frontend*) loomiseks valiti Pythoni mallimootor (*template engine*) nimega Jinja, mille abil HTML-i dokumendid kasutajani toimetada. Minimaalse ning dünaamilise funktsionaalsuse tagas JavaScript, mis võimaldas veebirakenduse hoida lihtsana ning vähendada sõltuvusi. Alternatiivina kaaluti Vue.js rakendamist, kuid selle kasutuselevõtt oleks suurendanud keerukust ilma suuremat kasu toomata.

Valitud tehnoloogiate eesmärk oli tagada arenduse efektiivsus ning rakenduse minimaalsus.

3.4 Projekti struktuur

Veebirakenduse lähtekood on organiseeritud loogiliselt jaotatud kaustadesse ning moodulitesse. Alloleval joonisel on kujutatud projekti failistruktuur.



„Joonis 1. Veebirakenduse struktuur”

Järgnevalt on toodud järgmiste kaustade ja moodulite otstarbed:

1. „app” – Sisaldab kogu veebirakenduse talitlusloogikat (*business logic*). Alammoodulid „config”, „models” ja “routes” sisaldavad endas vastavalt andmebaasiga ühendumist, andmebaasiga suhtlemist ning marsuutimismalle. Klassifailid „Processor.py” ja „Scraper.py” tegelevad akordilehtede töötlemise ja koorimisega. Fail „utils.py” sisaldab abifunktsioone andmestruktuuride teisendamiseks ning „errors.py” pakub veahaldust.

2. „db” – Sisaldab SQLite andmebaasifaili, mis lisatakse käitusaegselt. Kaust võimaldab andmete salvestamise ilma eraldi andmebaasiserverita.
3. „static” – Staatilised ressursid, mis saadetakse otse brauserile. Alamkaustad sisaldavad CSS-faile, pilte ning JavaScript-faile.
4. „templates” – HTML-mallid, mida Flask kasutab Jinja2 abil serveripoolsel ilmestusel (*rendering*). Siia kuuluvad nii vahelehed (*partials*) kui ka brauserile esitatavad vaated (*views*).
5. „Dockerfile” – Kirjeldab juhiseid veebirakenduse konteineriseerimiseks.
6. „init_db.py” – Skript, millega initsialiseeritakse andmebaas ja selles olevad tabelid. Lisaks sisestatakse kasutaja poolt valitavad akordid.
7. „instructions.md” ja „instructions.txt” – Identse sisuga paigaldus- ja kasutusjuhend.
8. „requirements.txt” – Fail, kus loetletud kõik veebirakenduse sõltuvused Pythoni keskkonna jaoks.
9. „run.py” – Veebirakenduse sisestuspunkt (*entry point*). See käivitab Flaski serveri ning määratleb, kuidas päringuid töödeldakse.

Sellise struktuuri eesmärk on hoida veebirakenduse komponentide vahel selget piiri ning võimaldada rakenduse hõlpsat laiendust ja hallatavust.

3.5 Akorditöötluse algoritmide kirjeldus

Veebirakenduse oluline osa on kooritud akordilehtede töötlus ja muusikaline analüüs. Järgnev alapeatükk kirjeldab algoritme, mis tuvastavad teose helistiku, transponeerivad helistikku ning lihtsustavad akorde.

3.5.1 Helistiku tuvastamine

Helistiku tuvastamine on oluline samm akorditöötluse protsessis, kuna see võimaldab hinnata repertuaari muusikalist struktuuri ning on abiks sihipärasel (helistikust helistikku) transponeerimisel. Kuna internetist kooritud akordilehtedel puudub sageli märge helistiku kohta, on vajalik seda automaatselt akordide esinemise põhjal tuvastada. Alljärgnev kirjeldus annab ülevaate, kuidas see algoritm üles ehitati.

Protsess koosneb järgmistest sammudest:

1. Akordide eraldamine akordilehe tekstist – akordileht jagatakse ridade kaupa ning regulaarse avaldise abil sobitatakse ridu, mis sisaldavad vaid akordile omaseid mustreid. Tuvastatud akordidest koostatakse esinemiste sagedustabel ning lisaks salvestatakse iga akordi positsioonid tekstis.
2. Enharmooniline teisendus – et tagada järjepidev notatsioon ning vältida tähistusviiside mitmekesisusest tulenevaid üleliigsuseid, teisendatakse kõik bemollid nende ekvivalentseteks dieesideks (nt B ♭ → A#).
3. Akordide standardiseerimine – iga akord teisendatakse kujule, mis sobituks diatoonilise helirea akordidega. Selleks valitakse iga akordi põhiheli (ingl *root*) ja akordi kuju (ingl *quality*), kuid liigitus säilitatakse ainult juhul, kui akord sisaldab minoorkolmkõla või vähendatud kolmkõla. Näiteks teisendatakse Cmaj7 ja C7 mõlemad kujule C, samas kui Gm7-st saab Gm ning Bdim jääb muutmata. Selline lähenemine tagab, et analüüs keskendub akordi põhistruktuurile ning võimaldab seda võrrelda diatoonilistes heliridades esinevate akordidega.
4. Kõige tõenäolisema helistiku tuvastamine – iga loomuliku mažoorse ja minoorse helirea jaoks arvutatakse kaalutud skoor. Kaalutud skoor saadakse, korrutades iga repertuaaris esineva standardiseeritud akordi esinemissageduste summa vastava helireas esineva akordi tähtsuse koefitsiendiga (nt toonika 1.5, dominant 1.3 jne). Protsessi korratakse iga helirea jaoks, saades arvuline näitaja, mis väljendab selle sobivust repertuaaris esinevate akordide põhjal. Määratud läve järgi lisatakse kõrgete skooridega helistikud kandidaatidena lõplikku nimekirja.
5. Lõpliku otsuse tegemine – kui kandidaate on mitu, rakendatakse täiendavat analüüsi, et teha lõplik valik. Otsustamisel arvestatakse järgmisi tunnuseid:
    a. kas repertuaari esimene või viimane akord vastab kandidaathelistiku toonikale;
    b. kas repertuaaris esineb sellele helistikule tüüpilisi akordide järjestusi (nt I-IV-V või I–V–vi–IV);

       Esimese/viimase akordi näitaja hindamisel arvestatakse repertuaarist leitud unikaalsete akordide arvuga, et vältida ülehindamist (nt kui repertuaaris esinebki vaid 2 unikaalset akordi).

Selle protsessi tulemusena valitakse helistik, mis vastab repertuaaris esinevatele akordisagedustele ja -järjestustele kõige paremini.

3.5.2 Transponeerimine ja akordide lihtsustamine

Akordide töötlemise üks olulisemaid eesmärke on kohandada repertuaari vastavalt kasutaja vajadustele – näiteks vähendada tehnilist keerukust või võimaldada transponeerimist helistikku, mis sisaldaks suurema hulga kasutajale tuttavaid akorde. Selleks on veebirakendusse lisatud kaks funktsionaalsust: akordide transponeerimine ja lihtsustamine. Mõlemad toimingud muudavad akordilehe sisu, säilitades selle üldise vormingu ja loetavuse.

Transponeerimine teisendab kõik repertuaaris olevad akordid ühest helistikust teise. Protsessi käigus:

1. arvutatakse algse ja soovitud helistiku vahe pooltoonides;
2. iga akord teisendatakse vastavalt selle intervallile uude helistikku.
3. transponeeritud akordi pikkust võrreldakse algse akordi pikkusega ning säilitatakse tekstis joondus, lisades vajadusel tühikuid või eemaldades liigseid tähemärke.

Kuna tekstina esitatud akordilehel on tähemärkide asukoht oluline, transponeeritakse akordid paremalt-vasakule-järjestuses – see väldib olukorda, kus varajasem asendus mõjutab järgnevaid positsioone tekstis; olgu selleks järgmine akord või laulusõna.

Transponeerimisel kasutatakse teadlikult ainult diees-tähistust (nt A# mitte B ♭). Näiteks olukorras, kus D-duur transponeeritakse ühe pooltooni võrra allapoole on levinum akord D ♭ -duur, kuid veebirakendus kuvab selle C#-duurina. Selline otsus lihtsustab akordide töötlemist ja sisemist loogikat, vältides olukordi, kus samakõlalisi akorde tähistatakse mitmel erineval viisil.

Akordide lihtsustamine aitab muuta repertuaari mängimise kergemaks, teisendades keerukad akordid. Lihtsustatakse septakordid (va dominantseptakordid), suurendatud ja vähendatud kolmkõlad ning *suspended*- ja *added*-akordid, mida alustavad mängijad sageli ei oska või ei soovi kasutada. Lihtsustamise käigus tuvastatakse iga akordi põhiheli ning selle lihtsustatud kuju määratakse järgmiste reeglite alusel:

- Kui akord on dominantseptakord (nt B7, G7), siis see jäetakse muutmata, sest sellel on oluline harmooniline funktsioon ja näiteks on algajal lihtsam mängida B7-akordi kui barree-võtet nõudvat B-duuri.

- Kui akord on minoorkolmkõla või selle laiend (nt Am, Am7, Am9), siis jäetakse alles vaid minoorkolmkõla.

- Kõik muud akordid (nt Cmaj7, Dsus4, Fadd9, Aaug) lihtsustatakse kujule, mis sisaldab vaid põhiheli abil koostatud kolmkõla.

  Lihtsustatud akordid sisestatakse akordilehele, säilitades võimalikult sarnase vormingu ja joondatuse, et tagada visuaalne loetavus ja mängitavus.

Kokkuvõttes moodustavad helistiku tuvastamine, transponeerimine ja akordide lihtsustamine akorditöötluse tuuma, mis võimaldab veebirakendusel kohandada repertuaare vastavalt kasutaja oskustasemele ja eelistustele. Need algoritmid tagavad, et kooritud akordilehed ei jää staatiliseks esitluseks, vaid muutuvad interaktiivseks ja õppimist toetavaks tööriistaks. Tulemuseks on isikupärastatud ning tehniliselt mängitav repertuaar, mis võimaldab kasutajal oma oskusi järk-järgult arendada.

3.6 Varajane arendus ning ilmnenud väljakutsed

Veebirakenduse arendamine algas mitmete testskriptide ja eksperimenteerivate lahendustega, et hinnata, millised meetodid sobiksid kõige paremini projekti vajadustega. Varases faasis keskenduti peamiselt artistide ja akordilehtede töökindlale leidmisele.
Esialgne katse kasutada DuckDuckGo otsingumootorit artisti lehekülgede leidmiseks ebaõnnestus kiiresti. Kuigi päringud suudeti esitada edukalt, kasutades juhuslikult vahetuvaid User-Agent päiseid, ilmnes peagi probleem usaldusväärse vastuse saamisel. Nimelt hakkas DuckDuckGo juba vähem kui kümne päringuga tagastama viipasid, kus paluti lahendada CAPTCHA. See muutis automatiseerimise sobimatuks ning seega tuli leida teine lähenemismeetod. Katse näitas, kui oluliseks osutub ligipääsu stabiilsus andmete koorimiseks.
Sellele järgnes Serper.dev teenuse kasutuselevõtt, mis võimaldas töökindlalt päringuid Google'i otsingumootorile saata, kasutades nende API-teenust. Serper.dev osutus tunduvalt tõhusamaks, pakkudes järjepidevalt struktueeritud vastuseid. Kuigi teenuse tasuta paketi sisaldas vaid 2500 päringut, ei ole mahtu siiani ületatud.

Veebikoorimise loogika realiseeriti algselt eraldi skriptina, mille valmimisel see „Scraper.py" klassiks teisendati. Algselt oli väljastpoolt kutsutav üks kapselmeetod (ingl *wrapper method*), mille ülesanne oli sisendina võtta artisti nimi ning tagastada artistide lehelt leitud esimese loo akordileht. Arenduse edenedes muutus lahendus aga granulaarsemaks – töövoog jagati etappideks: artisti lehekülje leidmine, kõikide lugude tuvastamine ja juhuslikult valitud akordilehtede koorimine ning töötlemine. Selle tulemusel vähenes ka tõrgete tõenäosus ja paranes testitavus.

Veebilehelt azchords.com andmete hankimisel ilmnes täiendav tehniline tõrge. Kuigi sait kasutas HTTPS-protokolli ning brauser tõlgendas sertifikaati kehtivana, ei vastanud see tegelikult domeenile. Pythoni requests teek esitas veateate, kuna TLS-sertifikaadi domeeniks oli hoopis www.allmusicals.com. Sellest tulenevalt pidi Pythonis seadistama päringute esitamisel kätluse (ingl *handshake*) mittekontrollimise. Hoiatuste vältimiseks keelati urllib3 teegi abil vastavad teated. Kuigi sellist lahendust ei peeta üldjuhul turvaliseks, osutus see antud olukorras vajalikuks.

Paralleelselt koorimisloogika arendusega testiti ka akordilehtede töötlemise algoritme. Selleks loodi esmalt eraldiseisev testskript, milles katsetati akordide ja helistiku tuvastamist ning transponeerimist. Enamik loogikast koondati hiljem eraldi „Processor.py" klassi, mille eesmärk on akorditekstide analüüs ja teisendus. Klass kujunes välja iteratiivselt ning näiteks transponeerimisloogika tuli hiljem ümber kirjutada, et lahendada akordide pikkuste erinevustest tingitud vorminduse probleemid.

Lisaks koorimisloogikale loodi skript „utils.py" funktsionaalsuse testimiseks, mis sisaldas meetodeid spetsiifilise Python andmestruktuuri teisendamiseks JSON-formaati ja tagasi. Samuti katsetati skripti abil andmebaasi loomist ja suhtlust.

Kuigi lõviosa töövoost ja veebirakenduse funktsionaalsustest kujunes välja hilisemas etapis, aitasid varajased katsed selgelt määratleda usaldusväärse andmeallika ning luua aluse hilisema arhitektuuri jaoks.

3.7 Projekti arhitektuur

Veebirakenduse arhitektuur toetub moodulipõhisele ülesehitusele, mille eesmärk on tagada selge töövoog ning komponentide isoleeritus. Selles alapeatükis kirjeldatakse rakenduse põhilist kasutusstsenaariumi – artisti otsingut, akordilehtede töötlemist ning tulemuste kuvamist. Lisaks selgitatakse põhimoodulite rolli ning nende omavahelisest koostööd.

Kui kasutaja on edukalt konto loonud ning sisse logitud, suunatakse ta avalehele („index.html"), kus kuvatakse otsinguvorm. Kasutaja sisestab artisti või bändi nime ning vajutab „Otsi". Selle tulemusena saadetakse POST-päring marsruudile /search, millele vastab Flaski vaatefunktsioon (ingl *Flask view function*), teostades järgmised sammud:
esmalt kasutatakse „Scraper.py" klassi, et leida artisti lehekülg AZChords portaalist, tuginedes selleks Serper.dev API-teenusele. Kui artist pole veel andmebaasi talletatud, kooritakse lehelt kõikide artisti lugude lingid ning salvestatakse need SQLite-andmebaasi koos artisti tabeli võõrvõtmega.
Järgmisena valitakse juhuslikult ülimalt 7 lugu, mida kuvatakse kasutajale. Kui mõnda valikusse sattunud lugu pole veel eelnevalt töödeldud, edastatakse see „Processor.py" klassile. Selle ülesanneteks on tekstist regex'i abil akordide tuvastamine, bemollide teisendamine dieesideks ning akordide standardiseerimine helistiku tuvastamiseks. Töödeldud lugu salvestatakse koos akordilehel esinevate akordide positsioonidega andmebaasi. Kui mõne loo puhul tuvastatakse viga (nt puuduvad akordid), eemaldatakse see kuvatavate lugude loendist ning eemaldatakse kirje andmebaasist.
Kui kõikide valitud lugude töötlemine on lõppenud, lisatakse iga tulemuse juurde sobivushinnang, mis arvutatakse kasutaja poolt varem märgitud tuttavate akordide alusel. Kui kasutaja on määranud teatud akordid juba õpituks (vastavas vaates), leitakse kattuvus loole vastavate akordide vahel. Tulemused kuvatakse „results.html" malliga, kus iga lugu esitatakse eraldiseisva elemendina. Iga loo kohta on näha pealkiri, artisti nimi, sobivuse protsent ning loetelu akordidest, mis tuleks ära õppida. Lisaks on võimalik märkida lood lemmikuks.
Kui kasutaja klikib mõnele loole, tehakse GET-päring marsuudile /song, mille kaudu laaditakse selle konkreetse loo vaade. Loo vaates kuvatakse pealkiri, tuvastatud helistik ning manustatakse YouTube'i video, kui selle leidmine Serper.dev API-teenuse kaudu õnnestub. Kui video URL on andmebaasi juba salvestatud, kasutatakse eksisteerivat linki. Kasutaja saab akordilehe peal mitmesuguseid toiminguid sooritada: märkida lugu lemmikuks, transponeerida helistikku või lihtsustada akorde. Mõlemad akordide teisendamise operatsioonid toimuvad asünkroonselt – JavaScript saadab POST-päringu vastavale Flaski marsruudile ning uuendab DOM-i ilma lehte uuesti laadimata.
„Scraper.py" ja „Processor.py" moodustavad arhitektuuri tuuma. Esimene on vastutav välise sisu hankimise eest ning teine kogu akorditöötluse eest.
Kõik serveripoolsed vaated on kirjutatud Flaski marsruutidena ja tegelevad peamiselt andmebaasipäringute, failide töötlemise ja vastuste ettevalmistamisega. HTML-lehed on

koostatud Jinja2 abil, mille aluseks on „base.html”, mis sisaldab ka navigeerimisriba. JavaScripti roll on säilitada lehe dünaamilisus – akordimuudatused ja lemmikuks märkimised toimuvad kasutajale sujuvalt ja viivitusteta.

Kuigi põhiline arhitektuur keskendub artisti otsingule ja akordilehtede töötlemisele, on loodud ka mitmeid lisafunktsionaalsusi. Kasutajad saavad määrata endale tuttavaid akorde, mille alusel hinnatakse iga loo sobivust. Samuti saab lugusid lemmikuks märkida ja neid hiljem eraldi vaates sirvida. Lisaks võimaldab YouTube’i video manustamine kasutajal loole reaalajas kaasa mängida.

Seega on rakenduse arhitektuur üles ehitatud nii, et kõik moodulid täidaksid kitsast ja selgelt määratletud ülesannet. Kliendi- ja serveripoolne koostöö toimub standardiseeritud API-päringute kaudu ning rakenduse erinevad kihid – vaated, mallid, andmebaasimudelid ja töötlusloogika – on hoitud teineteisest sõltumatuna. See võimaldab lihtsamat testimist ning muudab tulevikus uute funktsionaalsuste lisamise kergemaks.

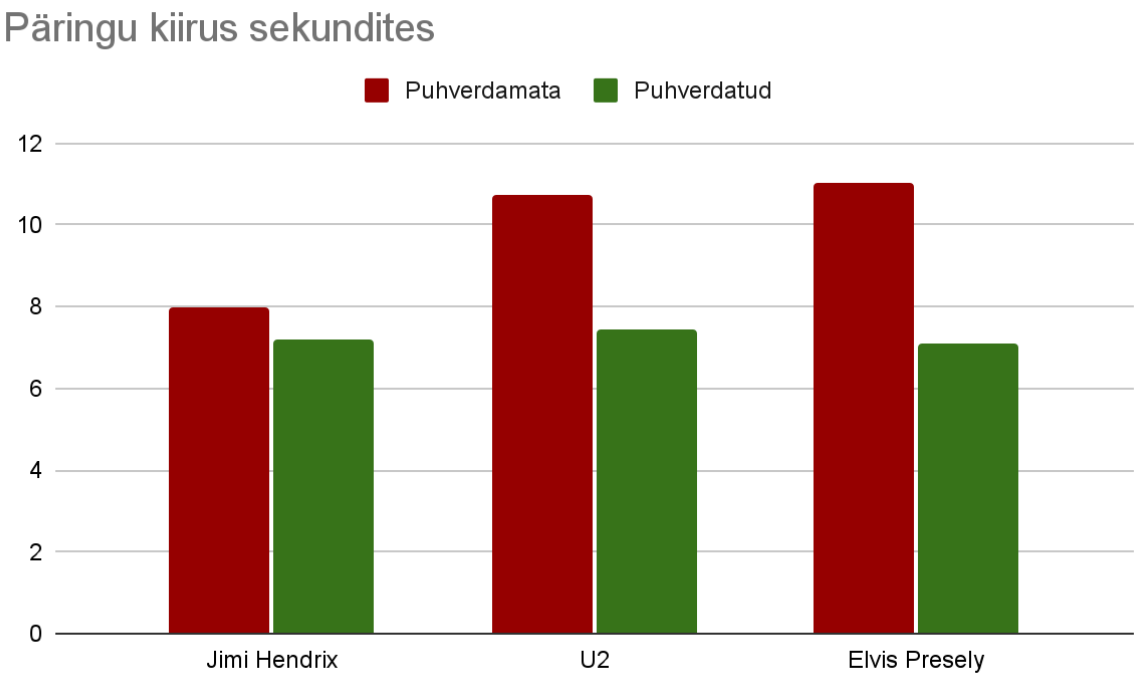
4. Tulemused

Selles peatükis antakse hinnang loodud veebirakendusele. Esmalt mõõdetakse päringute kiirust, seejärel käsitletakse kasutajate tagasisidet ning lõpetuseks esitatakse võimalikud edasiarendusvõimalused. Peatüki eesmärk on anda ülevaade rakenduse toimivusest ja arengupotentsiaalist.

4.1 Mõõtmistulemused

Kuna tegemist on veebirakendusega, mille riistvaranõuded on minimaalsed ning kogu töö toimub kergekaalulises serverikeskkonnas, keskenduti jõudluse hindamisel just artistiotsingu ja akordilehtede koorimise kiirusele. Ehkki saab mõõta ka muid ressursikasutuse näitajaid, peeti oluliseks hinnata päringute tegelikku vastamiskiirust. Enamik päringu kestusest kulub Serper.dev API-teenuse abil AZChords veebilehelt info hankimisele, samal ajal kui akordide töötlemine toimub sekundi murdosa jooksul.

Selleks viidi läbi päringute kiiruse mõõtmine kolme erineva artistiga, kelle tabulatuuride arv AZChords andmebaasis varieerus. Iga artisti kohta mõõdeti päringu kestvust kahes olukorras: kõigepealt tehti päring artisti kohta, keda andmebaasis veel ei eksisteerinud ning teisena korrati sama päringut, mis oli juba puhverdatud.



„Joonis 2. Päringu kiirus sekundites”

Joonisel „Päringu kiirus sekundites” on näha, et esimesel artistil – Jimi Hendrixil, kellel oli AZChords veebilehel saadaval vaid üks akordilehtede lehekülg – oli puhverdatud ja puhverdamata ajaline vahe kõige väiksem. U2, kellel oli AZChords veebilehel kolm akordilehtede lehekülge, oli erinevus selgemini märgatav. Kõige suurem vahe ilmnes Elvis Presley puhul, kelle AZChords profiil sisaldas kuut lehekülge ja ligikaudu 430 lugu, olles seeläbi ka testide kõige ressursinõudlikum juhtum.

Mõõtmistulemused näitavad, et veebirakendus saab kasu andmebaasi talletatud kirjetest ning juba salvestatud artisti puhul on päringule vastuse saatmine keskmiselt 2,67 sekundit kiirem.

4.2 Kasutajate tagasiside

Veebirakendus anti testimiseks väikesele sihtrühmale, mis koosnes kitarrimängutaustaga kasutajatest. Tagasiside kogumine toimus vabas vormis, kus osalejatel paluti tutvuda rakenduse funktsionaalsusega ning edastada tähelepanekuid kasutajakogemuse ja soovitavate täienduste osas.

Üldine hinnang rakendusele oli positiivne: kasutajad tõid esile, et rakendus täidab oma peamise eesmärgi – aidata leida kitarril mängitavaid lugusid, võttes arvesse kasutaja poolt valitud akorde – efektiivselt ja intuitiivselt. Kasutajaliidest peeti lihtsasti mõistetavaks ning visuaalse sobivuse protsendi kuvamine andis väärtuslikku indikatsiooni loo mängitavuse kohta.

Samuti tehti mitmeid konstruktiivseid ettepanekuid. Mõned kasutajad soovisid, et otsingutulemustes oleks võimalik tulemusi sorteerida sobivuse alusel, et tuua kõige sobivamad lood oleksid kohe silme ees. Lisaks sooviti võimalust kuvada sõrmestust akordide jaoks, mida kasutaja veel ei tunne, et õppimisprotsessi paremini suunata. Üks kasutaja tundis huvi võimaluse vastu otsida lisaks artistile ka konkreetseid lugusid. Märkimisväärne oli ühe kasutaja esitatud idee luua AI-põhine soovitussüsteem, mis pakuks kasutajale uusi lugusid eelnevate otsingute põhjal.

Tagasiside kinnitas, et rakenduse põhifunktsioonid töötavad ootuspäraselt ning seal välja toodud soovitused ja ideed andsid hea ülevaate, kuidas kasutajakogemust saaks veelgi parandada.

4.3 Võimalused edasiarenduseks

Veebirakenduse arenduse käigus kujunes välja selge arhitektuuriline baas, millele saab tulevikus hõlpsasti lisada uusi funktsionaalsusi. Testijate tagasiside ning enda hinnangulise analüüsi põhjal on võimalik välja tuua mitmeid arendusvõimalusi.

Esiteks võiks täiendada otsingutulemuste lehte, võimaldades kasutajal sorteerida erinevate kriteeriumite alusel – näiteks sobivusprotsendi, loo pealkirja või kuupäeva järgi, mil andmebaasikirje loodi. Samuti oleks võimalik pakkuda valikut, kas kuvada kõik akordid või ainult need, mida kasutaja veel ei oska. See lisaks paindlikkust ja suurendaks õppimise eesmärgil loodud rakenduse väärtust.

Lisaks kaaluti täiendada kasutajaliidest nii, et iga akordi juurde oleks võimalik kuvada ka visuaalne sõrmestuse skeem, mis on levinud mitmetes kitarriõppe keskkondades. See nõuaks kas kolmanda osapoole andmestiku integreerimist või akordipiltide genereerimist olemasoleva teegi abil.

Tehnilise poole pealt on planeeritud veebirakenduse juurutamine tööserverisse, kasutades selleks FreeBSD jaili-põhist lahendust. Sobiv keskkond on selleks juba loodud ning rakenduse näiteks Dockeri abil konteineriseerimine võimaldaks sujuvat üleviimist. Avalik juurdepääs annaks rohkem kasutajaid ja laiendaks testimisvõimalusi.

Pikemas perspektiivis võib kaaluda ka soovitussüsteemi loomist, mis lähtuks kasutaja varasematest otsingutest ja lemmikuks märgitud lugudest. See võimaldaks soovitada sarnaseid lugusid, mis võiksid kasutaja oskustasemele ja eelistustele vastata. See tähendaks kasutaja käitumise analüüsi ning sobivuse prognoosimist, mida oleks võimalik rakendada näiteks lihtsustatud masinõppe mudelite abil.

Kuigi praegune veebirakendus täidab oma esmase eesmärgi, on arhitektuuriline ülesehitus piisavalt paindlik, et võimaldada funktsionaalsuse täiendamist nii sisulises kui tehnilises plaanis.

Kokkuvõte

Töös töötati välja veebirakendus, mis võimaldab kitarrimängijatel otsida repertuaare, mille akordid vastavad nende oskustasemele. Rakenduse eesmärgiks oli ühendada muusikateoreetilised põhimõtted praktilise õpikeskkonnaga, et muuta lugude õppimine efektiivsemaks ja isikupärasemaks. Selleks tuvastati kooritud akordilehtedelt helistikud, võimaldati akordide transponeerimist ja lihtsustamist ning hinnati iga loo sobivust kasutaja poolt märgitud tuttavate akordide põhjal.

Töös anti ülevaade akordide ehitusest, helireadest, transpositsiooni ja lihtsustamise põhimõtetest ning nende praktilisest rakendamisest kitarrimängu kontekstis. Lisaks käsitleti veebikoorimise meetodeid, selle eetilisi ja tehnilisi aspekte ning tutvustati loodud veebirakenduse ülesehitust ja arhitektuuri. Rõhku pandi ka helistiku tuvastamise ja akordide töötlemise algoritmidele, mille toimimine võimaldas repertuaare kasutajapõhiselt kohandada.

Rakenduse funktsionaalsus testiti reaalse kasutajagrupi abil ning mõõtmistulemused kinnitasid, et süsteem suudab artistiotsingule vastata piisava kiirusega. Kasutajate tagasiside põhjal toodi välja mitmeid potentsiaalseid täiustusi, sealhulgas akordide sorteerimine, visuaalne sõrmestus ning soovitussüsteemi loomine.

Töös saavutati püstitatud eesmärk ja loodi toimiv lahendus, mis võimaldab muusikateooria rakendamist digitaalses kontekstis ning toetab erineva tasemega õppureid. Edasiarendusena nähakse võimalust süsteemi täiendada dünaamilise soovitussüsteemiga ning kasutajaliidest mitmekesistavate funktsioonidega.

Viidatud kirjandus

[1] Kotta K. Kooskõla. Akord. Muusikateooria õpik. Tallinn: Eesti Muusika- ja Teatriakadeemia. 2019.
https://mt.ema.edu.ee/muusika-elementaarteooria/i-7-kooskola-akord/ (15.12.2024)

[2] Happy Bluesman. Understanding the basics of guitar chord theory.
https://happybluesman.com/guitar-chord-theory-basics/ (15.12.2024)

[3] Green J. Guitar Chords Simplified. 2022.
https://discover.hubpages.com/entertainment/Guitar-lesson-Problem-Chords (15.12.2024)

[4] Adams C. How to Simplify a Song.
https://guitar-instruction-video.com/how-to-simplify-a-song/ (15.12.2024)

[5] Kyle. Guitar Chords Simplified.
https://www.jazzguitarguide.com/guitar-chords-simplified/ (15.12.2024)

[6] Imperva. Web Scraping.
https://www.imperva.com/learn/application-security/web-scraping-attack (15.04.2025)

[7] Crawlbase. Web Scraping – The Comprehensive Guide for 2025, Web Scraping For Beginners, 2024. https://crawlbase.com/blog/web-scraping-the-comprehensive-guide (15.04.2025)

[8] Alexander M. What is Web Scraping?. 2022.
https://www.scrapingbee.com/blog/what-is-web-scraping (15.04.2025)

[9] Research Data Services. An Introduction to Web Scraping for Research. 2019.
https://researchdata.wisc.edu/news/an-introduction-to-web-scraping-for-research (18.04.2025)

[10] Web Scraping FYI. beautifulsoup vs lxml.
https://webscraping.fyi/lib/compare/python-beautifulsoup-vs-python-lxml (18.04.2025)

[11] Pavlovskytė E. Scrapy vs. Beautiful Soup: A Comparison of Web Scraping Tools. 2023.
https://oxylabs.io/blog/scrapy-vs-beautifulsoup (15.04.2025)

[12] Proxyway. Scrapy vs Beautiful Soup vs Selenium – Which One to Use?. 2021.
https://proxyway.com/guides/scrapy-vs-beautiful-soup-vs-selenium (15.04.2025).

[13] Jedud K. Choosing Between Puppeteer vs. Selenium for Web Scraping, Zyte Blog, 2024.
https://www.zyte.com/blog/puppeteer-vs-selenium-the-web-scraping-tools/ (15.04.2025).

[14] Pelakauskas A. Web Scraping vs API: Which to Choose in 2025. 2025. https://oxylabs.io/blog/api-vs-web-scraping (15.04.2025).

[15] Roy A. Read and Respect Robots.txt File. 2024. https://www.promptcloud.com/blog/how-to-read-and-respect-robots-file/ (16.04.2025).

[16] Hirschey J. Symbiotic Relationships: Pragmatic Acceptance of Data Scraping, Berkeley Technology Law Journal, Vol. 29, 2014. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2419167.

[17] Imperva, Detecting and Blocking Site Scraping Attacks. 2016. https://www.imperva.com/docs/WP_Detecting_and_Blocking_Site_Scraping_Attacks.pdf (18.04.2025).

[18] Whittaker Z. Web scraping is legal, US appeals court reaffirms. 2022. https://techcrunch.com/2022/04/18/web-scraping-legal-court/ (18.04.2025).

[19] Bereczki T, Liber Á. The state of web scraping in the EU. 2024. https://iapp.org/news/a/the-state-of-web-scraping-in-the-eu (18.04.2025).

[20] Montanari A. Web-Scraping and Analyzing Song Lyrics. 2021. https://amontanari.altervista.org/webscraping-lyric/ (18.04.2025).

[21] FindDataLab. The Ultimate Guide To Ethical Web Scraping | Web Scraping Ethics – How To Not Get Blocked. https://finddatalab.com/ethicalscraping (18.04.2025).

[22] Krukowski I. Web Scraping Without Getting Blocked (2025 Solutions). 2024. https://www.scrapingbee.com/blog/web-scraping-without-getting-blocked/ (18.04.2025).

23] Feezell M. Transposition. 2011. https://learnmusictheory.net/PDFs/pdffiles/01-03-04-Transposition.pdf (15.05.2025). Learn Music Theory

[24] Boettcher J. *How To Transpose Chords With a Guitar Capo.* 2010. https://playguitar.com/how-to-transpose-chords/ (15.05.2025).

[25] Serna D. *How to Transpose a Chord Progression on the Guitar.* 2022. https://www.dummies.com/article/academics-the-arts/music/instruments/guitar/how-to-transpose-a-chord-progression-on-the-guitar-155165/ (15.05.2025). Dummies

[26] Heino Eller Music College. Tooniline suurus. https://eller.tmk.ee/oppematerjalid/intervallid/tooniline_suurus.html (15.05.2025).

Lisad

1. Github repositooriumi link:

   https://github.com/RasmusMeos/guitar-app