Jesse Arola

Päätemusiikkisoitin modaalisella käyttöliittymällä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

4.11.2024

Tiivistelmä

Tekijä:	Jesse Arola
Otsikko:	Päätemusiikkisoitin modaalisella käyttöliittymällä
Sivumäärä:	29 sivua
Aika:	4.11.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	FM Simo Silander

Opinnäytetyössä suunnitellaan ja toteutetaan pääasiallisesti vain näppäimistöllä ohjattava, pelkällä komentopäätteellä toimiva musiikkisoitin. Työn idea syntyi oppilaan omasta tarpeesta saada täysin näppäimistövetoinen musiikkisoitin, joka toimii pääosin samoilla komennoilla ja käyttöliittymäparadigmalla kuin useat muut samaan paradigmaan pohjaavat ohjelmat. Tämän lisäksi käydään läpi, miksi lähtökohdaksi valittiin pelkällä näppäimistöllä toimiva ohjelma, mitä hyötyä ja haittaa siitä on ja miten samanlainen käyttöliittymä on toteutettu muissa erilaisissa sovelluksissa.

Työssä käytetään toiminnallisuudeltaan valmista MPD-musiikkisoittimen palvelinohjelmaa, jotta voidaan keskittyä projektin kannalta olennaisempaan asiakasohjelmaan. Ohjelman väliaikainen nimi on Vimu.

Työn lopputuloksena saatiin musiikkisoitin sen perustavanlaatuisilla toiminnoilla, eli soittolistan muokkaaminen, soitettavan kappaleen valinta ja musiikin haku tietokannasta jonosoittolistaan.

Avainsanat: pääte, modaalinen, näppäimistövetoinen

Abstract

Author:	Jesse Arola
Title:	Terminal Music Player with Modal User Interface
Number of Pages:	29 pages
Date:	4 November 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Software Engineering
Supervisors:	Simo Silander, M.Sc.

In the study a keyboard driven, modal terminal user interface music player is designed and built. The idea for the software rose from the author's own need to have an entirely keyboard driven music player that functions with mainly the same commands as with similar software using the same paradigm. Additionally the thesis covers the reasoning as to why the keyboard driven paradigm was chosen, what its benefits and disadvantages are and how it is implemented in other similar software.

The software built uses the ready-made, feature-rich MPD back end music listening and organizing software to enable concentrating entirely on building the client software. The software goes under the name vimu.

As a result a music player program was made with the rudimentary features of playlist editing, choosing which song to play and searching the database for songs to add in to the queue playlist.

Keywords: terminal, modal, keyboard-driven

Sisällys

Lyhenteet

IRC: *Internet Relay Chat*. Protokolla reaaliaikaiseen pikaviestintään.

MPD: *Music Player Daemon*. Palvelinohjelmisto musiikin soittamiseen ja musiikkikirjaston hallintaan.

ssh: *Secure Shell*. Tietoturvallinen etäyhteys.

TUI: *Terminal User Interface*. Käyttöliittymä komentopäätteellä.

1 Johdanto

Tämän insinöörityön tarkoituksena oli luoda musiikkisoitin sellaiselle tietokoneen tehokäyttäjälle, joka suosii näppäimistön käyttöä ja mahdollisimman vähän hiiren käyttöä. Työn aihe valittiin, koska aloitushetkellä vastaavaa valmista vaihtoehtoa ei ollut, joten sellainen täytyi rakentaa itse. Päätteellä toimivia musiikkisoittimia on olemassa, mutta ei sellaista, joka toteuttaa halutun tavan käsitellä näppäinkomentoja. Projektissa toteutettavan ohjelman nimi tulee olemaan väliaikaisesti Vimu.

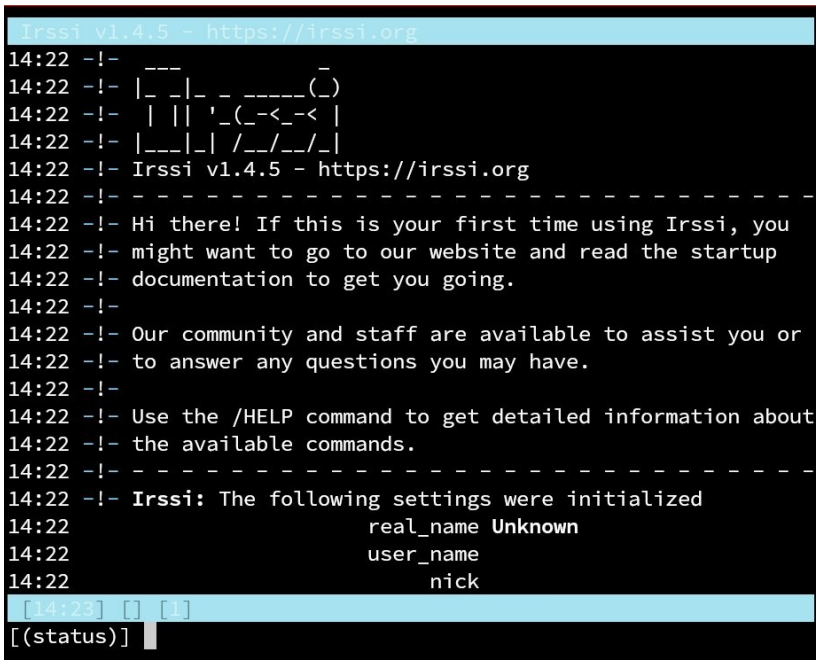
Projektin tavoitteena oli myös laajentaa musiikkikirjaston navigoinnin ja manipuloinnin mahdollisuuksia verrattuna vaihtoehtoisiin päätemusiikkisovelluksiin. Sen sijaan että navigoidaan ylös ja alas listassa yksi rivi kerrallaan, voitaisiin ensin antaa kerroinnumero, ja sen jälkeen painaa ylös tai alas, jolloin liikutaan kertoimen määräämän verran. Sen sijaan että suurta soittolistaa tarvitsisi järjestellä yksi kappale kerrallaan siirtäen ylös tai alas, voisi Vimussa merkata soittolista kappaleita valituiksi ja siirtää sen jälkeen toiseen kohtaan listassa.

Työssä selitetään ensin, miten haluttu näppäinkomentojen käsittelytapa toimii ja miten Vimu toteuttaa sen verrattuna muihin pääteohjelmiin. Tämän jälkeen käydään läpi työn suunnittelu ja toteutus, jonka jälkeen lopetetaan jatkokehitysmahdollisuuksilla ja yhteenvedolla.

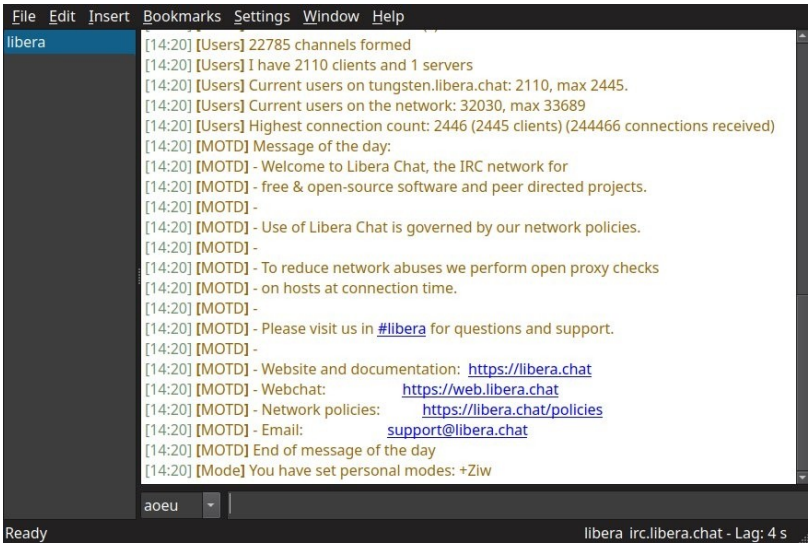
2 Työn lähtökohdat

Normaalissa tekstieditorissa kuten esimerkiksi Notepadissä tarve esittää käyttäjälle informaatiota ei ole kovin suuri. Notepadin tarkoitus on nimensä mukaisesti mahdollistaa nopeat muistiinpanot. Tekstinkäsittelyn monimutkaistuessa tarve tekstin ja sivun tyylin muotoilulle kuitenkin kasvaa ja tasapainottelu käyttöliittymän selkeyden ja ominaisuuksien esilletuomisen kanssa alkaa tuoda omat ongelmansa. Tehokäyttäjälle hiiripainotteinen käyttöliittymä kasvaa enemmänkin apupyöräksi ja näppäinyhdistelmien muistaminen valikoiden klikkailemisen sijaan voi hyödyttää käyttäjää pitkällä aikavälillä. Tässä osiossa käydään läpi käyttöliittymän suunnitteluun liittyvien päätösten motivaatiosta.

Toinen hyvä esimerkki näiden käyttöliittymäparadigman välillä on IRC-asiakasohjelmat. Minimalistinen käyttöliittymä ei joudu graafisessakaan käyttöliittymässä tarjoamaan kovin paljon, mutta keskusteluohjelmat ovat pohjimmiltaan näppäimistövetoisia, ja komentopääteohjelman oppiminen voi pidemmällä aikavälillä koitua mukavammaksi käyttää. Kuva 1 esittää IRC-asiakasohjelmaa graafisella käyttöliittymällä ja kuva 2 vain näppäimistöllä ohjattavaa, päätteellä toimivaa IRC-asiakasohjelmaa.



Kuva 1: Kuvankaappaus Irssi-ohjelmasta. Ohjelma näyttää nykyisen kanavan tiedot ylärivillä, nykyisen kanavan tai muuta informaatiota isolla keskialueella, avointen kanavien välilehdet toisiksi alimmalla rivillä ja alimmalla rivillä aktiivisen kanavan nimen ja tekstinsyötön.



Kuva 2: Kuvankaappaus Konversation IRC -ohjelmasta. Ohjelma tarjoaa kaikki sen ominaisuudet hiirellä klikattavassa käyttöliittymässä.

Neovim ja sen käyttöliittymä

Vimun käyttöliittymä pohjaa paljolti Vin, siitä nykyaikaisemman Neovim-tekstinkäsittelyohjelman ja etenkin samaan käyttöliittymäparadigmaan nojaavan Ranger-tiedostonhallintaohjelman käyttöliittymän suunnittelun ydinfilosofiaan. Tästä syystä on syytä tarkastella ensin Neovimin käyttöliittymää ja sen jälkeen verrata sitä tämän työn toteutukseen seuraavassa osiossa.

Neovimin käyttöliittymän kannalta neljä tärkeintä peruspilaria ovat:

- lähtökohtaisesti vain näppäimistöllä ohjattava
- kieli haluttujen toimintojen suorittamiseksi
- modaalinen käyttöliittymä
- käyttöliittymä lähtökohtaisesti pääte-emulaattorissa.

Näppäimistövetoisuus on koko ohjelman perusta. Kun käyttöliittymän suunnittelussa ei tarvitse enää ottaa huomioon, miten käyttäjä löytää tarvittavat ominaisuudet suureen valikkovalikoiman sisältä, voidaan keskittyä vain kaikkein olennaisimpaan. Kun ohjelman perusolettamus on, että käyttäjä muistaa käyttöön tarvittavat näppäinyhdistelmät eli kielen, voidaan käyttöliittymä siivota lähes kokonaan ylimääräisestä.

Kielellä tarkoitetaan pikanäppäimiin vertautuvia komentoja. Tekstinkäsittelyohjelmissa hyvin harva vähänkään kokeneempi käyttäjä klikkaa Tiedosto ja Tallenna tallentaakseen uusimmat muutokset, koska pikanäppäimet ctrl+s toteuttaa saman paljon nopeammin ja tulee ajan myötä täysin selkärangasta sitä edes sen enempää ajattelematta. Neovim vie tämän vain pidemmälle, nimittäin koko käyttöliittymän tasolle.

Normal-moodissa, eli tekstin manipulointimoodissa esimerkiksi diw poistaa kursorin alla olevan sanan, ja dd poistaa kokonaisen rivin. Erilaisten komentojen oppiminen ajan mittaan luo käyttäjän ja ohjelman välille kielen, jonka avulla oh-

jataan ohjelmaa niin nopeasti kuin käyttäjä kykenee ajattelemaan. Kun välissä on kieli, jolla ohjata ohjelmaa, asiat ikään kuin "vain tapahtuu". [1.]

Neovimissä on modaalinen käyttöliittymä, jonka tarkoitus on eriyttää ohjelman toiminnot pääasiallisesti neljään eri moodiin. Moodit määrittelevät, mitä näppäimistön eri näppäimet tekevät, ja moodien valitsemiseen on omat näppäimensä. Neovimin neljä pääasiallista moodia ovat:

- normal tekstin manipulointiin
- visual tekstin valitsemiseen
- insert tekstin kirjoittamiseen
- kaksoispistekomennot, sisäisten ohjelmoitavien funktioiden suorittamiseen ja asetusten muokkaamiseen.

Neovimin tuoma modaalinen käyttöliittymä tekee siitä moniulotteisen: tekstin valitseminen, muokkaaminen ja siihen tekstin suoraan kirjoittaminen erotetaan omiin moodeihinsa. Mikä voi päältäpäin vaikuttaa turhalta kompleksisuuden lisäämiseltä käytännössä, lisää työn tehokkuutta ja sen ergonomiaa paljon suurempaa oppimiskäyrää vastaan. Normal on pääasiallinen moodi ja missä Neovimin kieli nousee esiin. Insert on tekstin kirjoittamista varten, kuten perinteisessä tekstinkäsittelyohjelmassa. Visual on kuin Normal-moodi mutta nimenomaan tekstin valitsemista varten.

Käyttöliittymän luominen pääte-emulaattorin rajoitusten puitteissa ei ole sinänsä pakollista, mutta pakottaa käyttöliittymän suunnittelijan panostamaan kolmeen edellä mainittuun peruspilariin, koska asioita ei voi vain piilottaa hiirellä klikattaviin, usein hyvin syviin valikoihin. Toinen hyöty liittyy sen siirrettävyyteen. Toisin kuin graafiset käyttöliittymät, komentokehotekäyttöliittymät ovat vain tekstiä ja siten niitä voi hyödyntää esimerkiksi pelkän ssh-yhteyden avulla.

Pelkän näppäimistön kanssa käytettävä käyttöliittymä onkin tietoinen valinta tehdä inkrementaalisesti ylimääräistä työtä lyhyemmällä aikavälillä ohjelmiston käytön opiskeluun, jotta voidaan nauttia sen hyödyistä pitemmällä aikavälillä. Se

ei sovellu kaikkiin tarkoituksiin, mutta se on sovellettavissa yllättävän monen tyyppiseen ohjelmaan. Mitä enemmän ohjelmaa käyttää päivittäisesti, sitä enemmän siitä on mahdollista hyötyä.

3 Vimun toiminnallinen määrittely

Vimu omaksuu vi:n käyttöliittymäparadigman vähän omilla painotuksillaan ja on paljon lähempänä Ranger-tiedostonhallintaohjelmaa.

3.1 Moodit

Toisin kuin Neovim, Vimun moodit jakautuvat kolmeen:

Normal-moodi käsittää operaatiot kuten kappaleen poistamisen soittolistalta ja välilehden vaihtamisen. Tekstinkäsittely ei ole melkein missään roolissa Vimussa, toisinkuin Neovimissä, joten Normal- ja Visual-moodit yhdistetään ja valjastetaan liikkumiseen ja muun muassa soittolistojen manipulointiin.

Insert-moodia tullaa käyttämään lähinnä esimerkiksi hakukentissä ja metadatan editoinnissa. Kuten Neovimissä, päärooli on Normal-moodilla.

Kaksoispistekomennoilla voidaan suorittaa mahdollisia ohjelman sisäisiä komentoja ja muokata asetuksia.

3.2 Oletuspikanäppäimet

Vi-oletuspikanäppäimet ovat käytännössä standardisoituneet, ja Vimu käyttää niitä myös oletusarvoina. Taulukko 1 esittää Vimun oletuspikanäppäimet Normal-moodissa. Ne ovat lähestulkoon samat kuin Neovimissä ja Rangerissä.

Taulukko 1: Vimun oletuspikanäppäimet.

h	Vasemmalle siirtyminen
j	Alas siirtyminen
k	Ylös siirtyminen
l	Oikealle siirtyminen
CTRL+d	Puolen ruudullisen verran rivejä alas
CTRL+u	Puolen ruudullisen verran rivejä ylös
g	Ylinpään riviin siirtyminen
G	Alimpaan riviin siirtyminen
välilyönti	Rivin valinta
d	Valittujen rivien poistaminen ja ohjelman sisäiseen leikepöytään kopioiminen
[n][muu komento]	Kaikkiin komentoihin voi etuliittää numeron ja se toistetaan [n] kertaa

3.3 Teknologiat

3.3.1 Rust lyhyesti

Vimun pääasialliseksi kieleksi harkittiin Pythonia ja Rustia. Rustiin päädyttiin sen tuoman lisähaasteen, yleisen suosion ja mielenkiintoisten uusien konseptien vuoksi.

Rust on verrattain nuori ohjelmointikieli, jonka perimmäiset painopisteet ovat samanaikaisuus ilman kilpailutilanteita ja muistin hallinta ilman muistinsiivousmekanismia. Kummatkin tavoitteet tulee myös saavuttaa ilman negatiivista vaikutusta suorituskykyyn. [2.]

Rustin syntaksi muistuttaa paljon C++-kieltä [3], mutta toiminnallisuudessaan se ottaa paljon vaikutteita myös funktionaalisista kielistä muun muassa sulkeumien, pattern match, enum ja iteraattoreiden muodossa. Kaikki arvot ovat oletusarvoisesti vakioita, mutta ne voidaan erikseen merkata muuttujiksi mut-avainsanalla. [4.]

Rustin turvallinen muistinhallinta perustuu sen borrow check -mekanismiin, joka takaa koodin käännösvaiheessa, että [5]:

1. jokaisella arvolla on omistaja
2. omistajia on jokaisella arvolla vain yksi
3. arvo pudotetaan ohjelmasta pois, kun arvolle määritelty elinkaari päättyy.

Esimerkkikoodi 1 esittää sekä omistajuuden siirron funktiokutsussa että vakion elinkaaren päättymisen.

```
fn main() {
     let nimi = String::from("Jesse");
     hello(nimi); // hello-funktio ottaa vakion "nimi" omistukseensa.
     println!("{}", nimi);   // Aiheuttaa virheen koska vakio
                             // on jo pudotettu hello-funktiossa.
}

fn hello(nimi: String) {
     println!("Hello {}!", nimi);
} // nimi vakion elinkaari päättyy kaarisulkuun
  // ja se pudotetaan ohjelmasta.
```

Esimerkkikoodi 1: Kommentoitu koodi kuvaamaan omistajuuden siirtoa ja vakion tai muuttujan elinkaarta.

Koodin kokoonpanoon ja Vimun hallintaan käytetään pääasiallisesti Cargo-ohjelmaa, joka Vimua luodessa alustaa Vimun metatiedot ja oletuksena Git-versionhallinnan. Metatiedoista kaikkein tärkein on cargo.toml, johon merkitään perustiedot versionumerosta projektin riippuvuuksiin.

Rustin koodikirjastoja kutsutaan crateiksi (laatikko), ja sen ekosysteemi tarjoaa myös keskitetyn paikan näiden riippuvuuksien hallintaan crates.io-repositorion muodossa. Normaalisti riippuvuudet merkataan manuaalisesti [dependencies]-kohdan alle cargo.toml-tiedostoon, mutta uudemmissa Cargo-versioissa lisääminen onnistuu myös komennolla `cargo add [kirjaston nimi]`. Kun kirjasto on lisätty riippuvuudeksi, seuraavan koonnin yhteydessä Cargo myös lataa uudet riippuvuudet automaattisesti. Esimerkkikoodi 2 esittää lyhyen esimerkin cargo.toml-tiedoston sisällöstä. [6.]

Cargo tarjoaa paljon muitakin työkaluja kuten projektin julkaisemisen crates.io-repositorioon ja projektiohjelman käynnistämisen.

```
[package]
name = "TUIprojekti"
version = "0.1.0"
edition = "2021"

[dependencies]
ratatui = { version = "0.28.1", features = ["unstable-widget-ref"] }
```

Esimerkkikoodi 2: Lyhyt esimerkki cargo.toml-tiedostosta ja sen syntaksista valitessa riippuvuuden versiota ja ominaisuuksia.

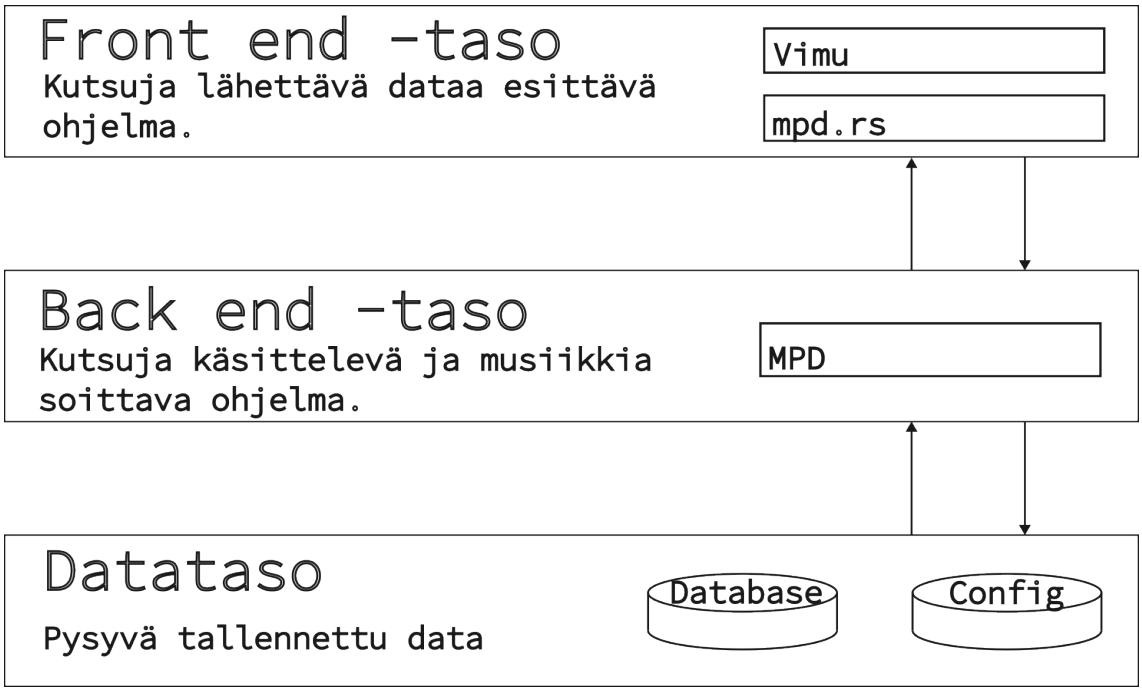
Rust vaatii itseltään vakautta ilman pysähtyneisyyttä, mikä käytännössä tarkoittaa jatkuvasti kehittyvää kieltä takaisinpäin yhteensopivana. Kun yhteensopivuutta rikkovia muutoksia tehdään, puhutaan Rustin eri editioista. Editiot merkataan vuosiluvulla ja tässä työssä käytetään editiota 2021. [7.]

Rust voidaan asentaa rustup-ohjelmalla ja tällä samalla ohjelmalla voidaan myös valita toolchain, eli tietty versio Rustista. Yleisimmin sitä käytetään vaihtamiseen vakaan ja kokeellisen Rust-version välillä.

3.3.2 MPD ja sen Rust-kirjasto

Vimun kehittämisessä haluttiin keskittyä nimenomaan käyttöliittymän rakentamiseen ja mikäli mahdollista, tekniset ratkaisut musiikin soittamiseen ja organisointiin voitaisiin delegoida valmiille sitä varten suunnitellulle ohjelmalle. MPD on client/server-arkkitehtuuriin perustuva musiikkisovellus. MPD itsessään hoitaa musiikin soittamisen ja musiikkitietokannan ylläpitämisen. Jotta sitä voitaisiin ohjata, tarvitaan client-sovellus. [8.]

MPD on taustaprosessi, johon otetaan yhteyttä TCP-yhteyden avulla. Sitä voidaan käyttää verkon yli, mutta pääasiallisesti sitä käytetään lokaalisti omalla koneella, johon TCP-protokkolla tarjoaa front end -sovelluksille rajapinnan. Kuva 3 kuvastaa MPD-ohjelman suhdetta sitä käyttäviin client-ohjelmiin ja sen pysyvään tallennustietoon arkkitehtuurikaaviona. [9.]



Kuva 3: MPD-ohjelman arkkitehtuurikaavio.

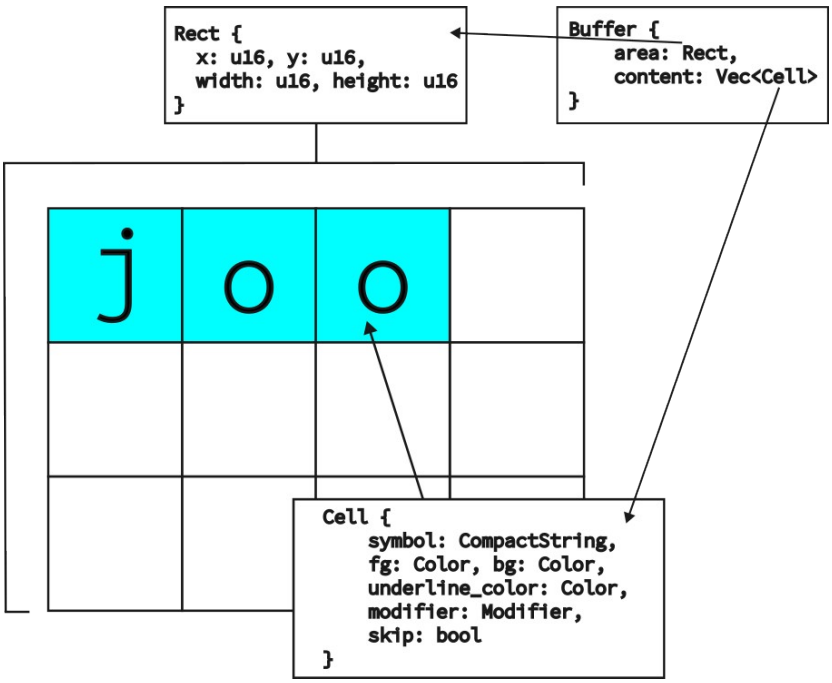
Kappaleiden soittamisen lisäksi MPD pitää tietokantaa kappaleista käyttäjän asettamasta hakemistosta ja tarjoaa toiminnallisuutta hakusanoilla etsimiseen. Lisäksi se tarjoaa toiminnallisuuden soittolistojen luontiin ja tallentamiseen.

3.3.3 Ratatui

Vimun on tarkoitus toimia kokonaan komentopäätteellä ja käyttöliittymän rakentaminen komentopäätteelle vaatii siihen erikoistuneen kirjaston. Ratatui tarjoaa Rust-ohjelmointikielelle oman TUI-kirjaston ja vaihtoehdon vanhemmille C/C++-kielillä kirjoitetuille kirjastoille kuten ncurses.

Ratatui antaa paljon painoarvoa siirrettävyydelle eri käyttöjärjestelmien välille, ja siten ei itsessään tarjoa toiminnallisuutta komentopäätteelle piirtämiseen, vaan merkistön jäsentelyyn, järjestelyyn ja sen jälkeen puskuriin asettamiseen. Tämä puskuri annetaan erilliselle kirjastolle, joka piirtää sen komentopäätteelle. Ratatui on suunniteltu näitä kirjastoja varten agnostisesti ja niitä on tarjolla kolme eri vaihtoehtoa, joista tässä työssä käytetään Crossterm-kirjastoa. [10.]

Piirtämistä varten merkit, muuntimet ja niiden tyylit asetellaan Buffer-rakenteen Celliin eli solukokoelmaan. Buffer sisältää myös Rect-taulukon, joka kuvaa päätteen tai sen osan koordinaatistoa. Buffer annetaan piirtokirjastolle, joka piirtää sen ruudulle. Kuva 4 kuvastaa Buffer-rakenteen Rect- ja Cell -kenttiä ja miten sen Cell-rakenteiden tiedot yhdistyvät Rect-rakenteeseen. [11.]



Kuva 4: Buffer-rakenne ja sen Rect ja Cell kentät.

Ratatui koostuu widgeteistä. Widgetit ovat valmiita palasia koodia tuomaan tietynlaista toiminnallisuutta valmiissa, helposti uudelleenkäytettävässä paketissa [12]. Esimerkkejä ovat Paragraph, joka tarjoaa vaihtoehtoja tekstin tyylitykseen, maksimipituuteen ja sen asetteluun. Toinen esimerkki on List, joka tarjoaa listatoteutuksen jostain yhteensopivasta kokoelmasta, minkä jälkeen listasta voi valita yhden. List-widgetti luo pohjan tälle työlle, ja sitä muokataan merkittävästi Vimun vaatimuksiin soveltuvammaksi.

4 Suunnittelu

4.1 Välilehdet

Ohjelman eri toimintokokonaisuudet tullaan jakamaan eri välilehtiin:

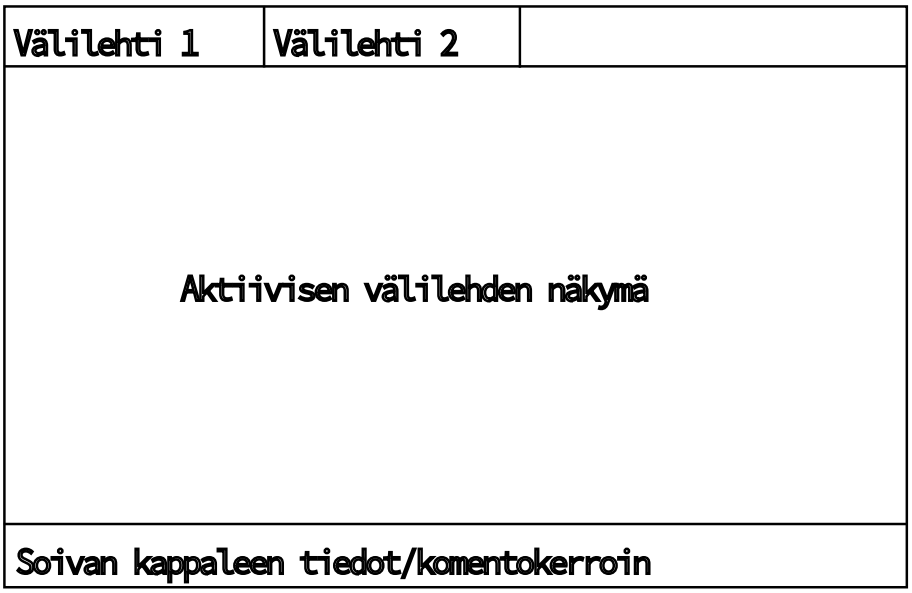
Ensimmäinen välilehti on nykyinen aktiivinen soittojono ja pääasiallinen soittolista, johon voi tuoda kappaleita osasta muista välilehdistä.

Toinen välilehti on kappaleiden hakuvälilehti, joka koostuu hakukentästä ja sen alla olevasta listasta. Hakukenttään voidaan syöttää artistin nimi, minkä jälkeen hakutulos näyttää kaikki kyseisen artistin kappaleet tietokannasta.

Oletuksena välilehtien vaihto tapahtuu painamalla Tab oikealla olevaan välilehteen ja Tab+Shift vasemmalla olevaan.

4.2 Asettelu

Ohjelmassa on kolme osiota: välilehtirivi, välilehden näkymä ja yleistä informaatiota esittävä tilarivi. Välilehtirivi pysyy muuttumattomana ohjelman ajon aikana, mutta välilehtinäkymän asetelma voi muuttua välilehden tarpeiden mukaan. Alin tilarivi näyttää oletuksena nykyisen soivan kappaleen tiedot, mutta komentokerrointa lisätessä se näyttää nykyisen kertoimen lukumäärän. Komentokerroin on numero, joka voidaan antaa ennen komentoa toistamaan sitä numeron luvun verran. Kuva 5 kuvastaa ohjelman eri osioiden asettelua, niin kuin ne näkyvät päätteellä.

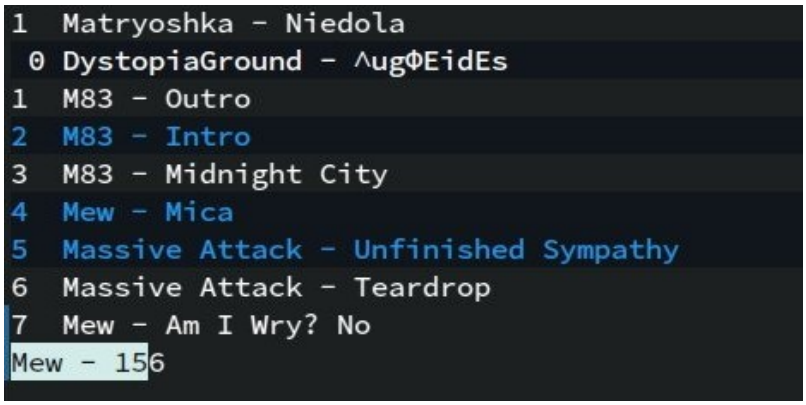


Kuva 5: Ohjelman pääasialliset kolme osiota.

Asettelua varten käytetään Ratatuin Buffer-rakenteen Rect-rakennetta, jota voidaan pilkkoa vapaasti eri osioiksi. Osioinnin jälkeen ohjelman eri Widgetit piirretään eri osioille.

4.3 Värit ja rivinumerot

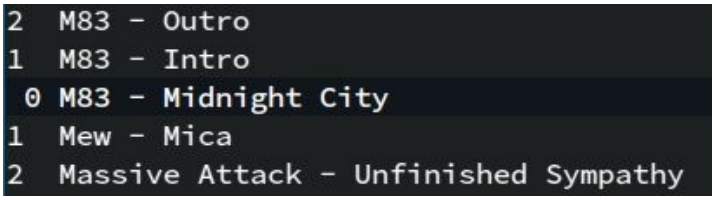
Jokaiseen välilehden listamuotoinen data esitetään samoin tavoin, käytännössä kahdessa osassa. Listan vasemmanpuoleinen osa esittää rivinumerot omassa sarakkeessa ja kappaleen tiedot omassa sarakkeessaan. Kuva 6 näyttää kuvankaappauksen avulla Vimun värivalintoja korostetuille ja valituille linjoille.



Kuva 6: Ylin rivi tummennetulla taustalla on tämän hetken korostettu rivi ja sinisellä tekstillä olevat valittuja rivejä. Alimpana oleva rivi taas esittää soivaa kappaletta ja sen edistymistä taustavärin avulla.

Ohjelman väreissä käytetään väriteemoja ja niiden käänteisarvoja esittämään valintoja ja muuta dataa. Valitut rivit korostetaan piirtämällä ne eri väreillä. Alimmainen tilannerivi esittää nykyisen kappaleen nimen ja ajan lisäksi nykyisen tilan edistymispalkin muodossa värittäen rivin taustaväriä vasemmalta oikealle kappaleen kuluneen ajan verran.

Rivinumerot näytetään aina vasemmalla puolella, ja korostettu rivi on aina sisennetty nolla. Kuva 7 näyttää ruudunkaappauksella rivinumerointia soittolistassa.



Kuva 7: Relatiiviset rivinumerot. Numerot piirretään aina vasemmalle puolelle.

5 Toteutus

5.1 Projektin alustus

Ratatuin ydin ei itsessään pakota projektille rakennetta, vain työkalut ja palaset widgettien muodossa. Kirjaston kehittäjät tarjoavat kuitenkin erillisen, vaihtoehtoisen oletusrakenteen, jonka päälle myös Vimu on rakennettu. [13.]

Normaalin Cargo-pohjan alustamisen sijaan käytetään Ratatuin kehittäjien tarjoamaa Async-pohjaa. Pohjan generointia varten asennetaan ensin cargo-generate-ohjelma. Esimerkkikoodi 3 esittää cargo-generate-ohjelman asennuskomennon.

```
> cargo install cargo-generate
```

Esimerkkikoodi 3: Cargo kokoaa ja asentaa cargo-generate-ohjelman käyttäjätasolle.

Tämän jälkeen projekti voidaan generoida. Esimerkkikoodi 4 esittää projektin generointiprosessin.

```
> cargo generate ratatui/templates
⚠️   Favorite `ratatui/templates` not found in config, using it as a
git repository: https://github.com/ratatui/templates.git
? 🤷   Which sub-template should be expanded? ›
 simple
❯ simple-async
 component

🤷   Project Name: vimu
```

Esimerkkikoodi 4: Projektin väliaikaiseksi nimeksi valitaan `vimu`, jonka jälkeen pohja generoituu `vimu`-alihakemistoon.

Generointi lisää tarvittavat perustavanlaatuiset riippuvuudet cargo.toml-metatietoihin, mutta sitä täytyy vielä muokata ja siihen täytyy lisätä riippuvuuksia. Esimerkkikoodi 5 esittää generoidun cargo.toml-metatietotiedoston sisällön.

```
[package]
name = "vimu"
version = "0.1.0"
authors = ["jarr"]
license = "AGPL-3.0-only"
edition = "2021"

[dependencies]
crossterm = { version = "0.27.0", features = ["event-stream"] }
futures = "0.3.30"
mpd = "0.1.0"
tokio = { version = "1.39.3", features = ["full"] }
```

Esimerkkikoodi 5: Generoitu cargo.toml-metatietotiedosto projektille.

Versionumeroinnissa on tarkoituksena tulevaisuudessa käyttää semanttista versionumerointia, jolloin vasemmanpuolinen on merkittävin, ja se muuttuu, kun esiintyy rikkovia muutoksia. Keskimmäinen minor tuo uusia ominaisuuksia, ja oikeanpuolimmainen paikkaavia versioita varten. [14.]

Lisenssiksi valittiin AGPL-3.0 takaamaan vapaata lähdekoodia, mikäli Vimu julkaistaan vapaaseen levitykseen.

Rustin 2024 editiota ei ollut vielä julkaistu tätä työtä tehdessä, joten siihen valittiin viimeisin vakaa editio 2021.

5.2 Asynkronisen projektirungon läpikäynti

Ratatui ei itsessään määritä projektin rakennetta, mutta projektin kehittäjät ovat luoneet sille kaksi eri esimerkkirunkoa uutta ohjelmaa varten, synkronisen ja asynkronisen. Synkronisessa kaikki operaatiot tapahtuvat määritellyssä järjestyksessä, mutta asynkronisessa funktiokutsujen päättymistä voidaan jäädä odottamaan samalla, kun muu ohjelma jatkaa pysähtymättä.

Vimun on tarpeellista tehdä automaattisia kutsuja TCP-protokollan avulla, jotta saadaan esimerkiksi nykyisen kappaleen kulunut aika piirrettyä ruudulle. Tätä varten valittiin Ratatui-projektin tarjoama Simple Async -runko uudelle ohjelmalle. Taulukko 2 esittää projektipohjaan generoitujen tiedostojen tarkoitukset ja kuva 8 Ratatui-prosessin tärkeimmät vaiheet.

Taulukko 2: Scr-, eli lähdekoodihakemistosta lähtien, tärkeimpien tiedostojen sisältö lyhyesti selitettyinä

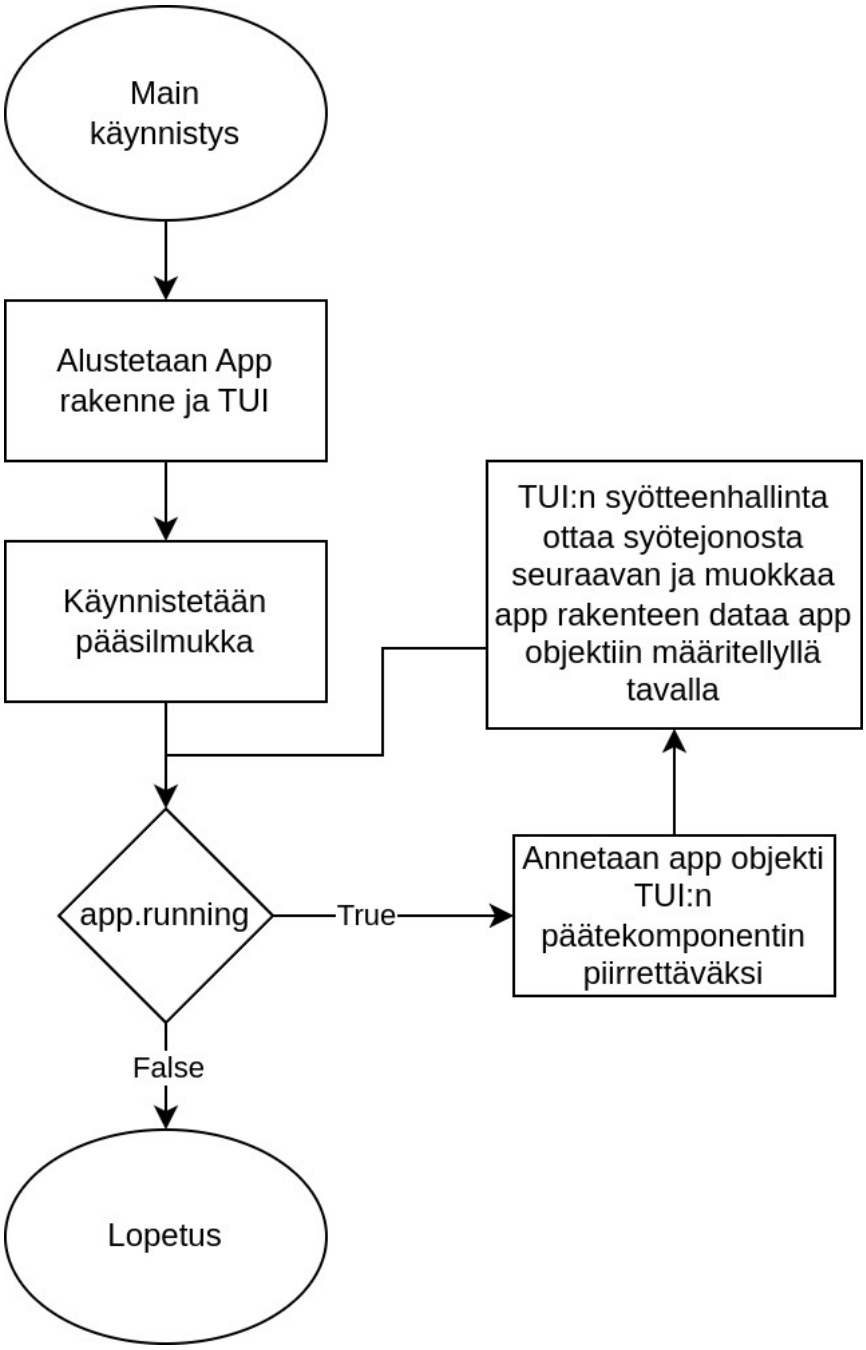
widgets/	Sisältää Vimua varten muokatut ja luodut widgetit.
app.rs	Vimun pääasiallinen rakenne sen alustamisen, tilan ja syötteen hallintaan.
event.rs	Komentopäätteen tapahtumien hallintaan tarkoitettu `struct`.
handler.rs	Syötetapahtumien `struct` joka määrittää mitä ohjelmassa tapahtuu käyttäjän syötteestä esimerkiksi näppäimistön tai hiiren syötteestä.
lib.rs	Ohjelman eri osien lataamiseen.
main.rs	Koko ohjelman alustava sen käynnistävä funktio.
tui.rs	Sisältää vakiotekstit ohjelmiston alustamiseen, lopettamiseen, piirtämiseen ja resetointiin komentopäätteellä käyttäen Vimuun valittua `crossterm`-kirjastoa. Sisältää myös vakiot käyttöliittymän komponenttien sijoitteluun.

ui.rs	Käyttöliittymän alustamiseen ja sen eri osa-alueiden sijoitteluun tarvittavat vakiot.

Lisäksi ohjelmaan on lisätty tiedostoja asetusten ja MPD:n hallintaan. Taulukko 3 esittää projektipohjaan lisätyt, Vimua varten luodut tiedostot ja niiden tarkoitukset lyhyesti selitettynä.

Taulukko 3: Src-hakemistoon lisätyt, Vimua varten luodut tiedostot.

app_config.rs	Rakenne Vimun käyttäjäkonfiguraation lataamiseen erillisestä tiedosta ja sen jälkeen konfiguraatiovakioiden käyttöön itse ohjelmassa
mpdw.rs	MPD-ohjelmistoon käytettävän `mpd-rs` kirjaston käärivä rakenne. Luo Vimua varten usein käytettyjä funktioita yhteen paikkaan.



Kuva 8: Ratatui-projektin rakenteen pääpiirteet.

5.3 Rivinumerointi

Pelkän näppäimistön kanssa toimiessa on tärkeää mahdollistaa jokaisen rivin numerointi helpottamaan ruudulla liikkumista. Vimuun suunnitellaan kaksi vaihtoehtoa:

Suhteellisessa numeroinnissa sen hetkinen aktiivinen rivi on aina numeroitu nollaksi, ja siitä ylös- ja alaspäin numeroitu suhteessa aktiiviseen riviin. Rivinumerot kasvavat aktiivisesta rivistä kumpaankin suuntaan absoluuttisin luvuin. Tämän avulla voidaan käyttää esimerkiksi pikanäppäintä [n][k], jotta voidaan siirtää aktiivinen rivi suoraan [n] määrä rivejä ylöspäin. Suhteellinen numerointi on ohjelman oletusasetus ja ainoa vaadittu numerointimuoto.

Absoluuttisessa numeroinnissa rivinumerot näytetään siinä järjestyksessä kuin ne ovat kokoelmassa, ylhäältä alaspäin kasvaen.

5.4 Widgetit

Osa Ratatuin Widgeteistä ovat niin perustavanlaatuisia, että niitä tulee käytettyä väistämättä. List-widget luo kuitenkin pohjan ohjelman sisällön esittämiseen. Kaikki sisältö – tässä vaiheessa musiikkikappaleet – esitetään rivimuodossa. Ratatuin oletustoteutus ei kuitenkaan tarjoa tähän työhön vaadittuja ominaisuuksia, joten rakenteita täytyi muokata.

Oletuksena List tarjoaa listan luonnin kokoelmasta ja yhden rivin valinnan, mutta Vimua varten tarvittiin mahdollisuus valita useampi kerralla. Lisäksi listaan täytyi lisätä suhteellinen rivinumerointi suunniteltuja navigointiominaisuuksia varten.

Ratatuin List-toteutusta lähdettiin muokkaamaan ensin kopioimalla List-widgettiin liittyvät tiedostot projektin omaan widget-hakemistoon. Taulukko 4 esittää kopioidut tiedostot ja niihin tehtyjen muutosten lyhyet kuvaukset.

Taulukko 4: Lista List-widgettiin kuuluvista tiedostoista ja niihin tehtyjen muutosten lyhyt kuvaus.

list.rs	Pääasiallinen rakenne, ei muutoksia.
item.rs	Listan esine, ei muutoksia.
rendering.rs	Implementoi Widget-piirteen tarvitsemat funktiot. Renderöintiä muokattu piirtämään rivinumerot ja valitut rivit pelkän kursoririvin sijaan.
list_state.rs	Vaihtoehtoinen tila listalle. Rakenteen alkuperäinen selected kenttä uudelleen nimettiin highlighted ja rakenteeseen lisättiin uusi kenttä selected.

Tiedostot kopioitiin alkuperäisestä Ratatui-repositoriosta sellaisenaan ja muokattiin alustavasti. Koodi-importit korjattiin ja nimet vaihdettiin uniikeiksi, minkä jälkeen itse muokkausprosessi voi alkaa. List-rakenne on jaettu neljään eri tiedostoon ja muutoksia tehtiin lähinnä renderöintiin ja tilan hallintaan. Koodiesimerkit 6 ja 7 esittävät muutoksia Ratatuin alkuperäiseen listan tilarakenteen toteutukseen. Rakenteen kenttien näkyvyyttä piti muuttaa, lisätä yksi uusi ja uudelleen nimetä yksi.

```
pub struct ListState {
     pub(crate) offset: usize,
     pub(crate) selected: Option<usize>,
}
```

Esimerkkikoodi 6: Alkuperäinen rakenne List-widgetin tilan hallintaan. Selected on tässä tarkoitettu yhden listan esineen valintaa varten.

```
pub struct VListState {
     pub offset: usize,
     pub highlighted: Option<usize>,
     pub selected: Vec<usize>,
}
```

Esimerkkikoodi 7: Muokattu ja uudelleennimetty rakenne. Näkyvyyttä on muokattu pakettitason julkisesta julkiseksi, vanha selected on uudelleennimetty highlightediksi ja uusi selected hallitsee valintakokoelmaa.

5.5 Syötteen hallinta

Ohjelman pääasialliset tilat ja sen alikomponenttien tilat sekä kaikki ohjelman eri toimintojen funktiot määritellään App-rakenteeseen. App-objekti annetaan syötettä hallitsevalle Handler-rakenteelle. Handler-rakenne kutsuu päätettä kontrolloivaa koodikirjastoa ja siten voidaan kytkeä eri näppäimet tiettyihin komentoihin. Koodiesimerkki 8 esittää app-objektin funktioita, joita voidaan kutsua esimerkiksi näppäimien painalluksen yhteydessä.

```
impl App {
     pub fn new() -> Self {
          Self::default()
     }
     pub fn change_volume(&mut self, volume: i8) {
          self.mpdw.conn.volume(
               self.mpdw.status.clone()
               .unwrap().volume + volume
          ).unwrap();
     }

     pub fn change_song(&mut self, queue_index: u32) {
          let _ = self.mpdw.conn.switch(queue_index);
     }
...
```

Esimerkkikoodi 8: Lyhennetty versio App-rakenteen funktioista. Ensimmäinen funktio luo uuden App-objektin, toinen vaihtaa äänenvoimakkuutta ja kolmas vaihtaa kappaletta jonosoittolistassa.

Näppäimistön syötetapahtumat määritellään KeyEvent-rakenteen KeyCode Enumissa, jota vasten kirjoitetaan match-lauseke. Oletuksena painallukset jätetään huomioimatta match-lausekkeen kaiken kattavalla alaviivalla ja erikseen määritellyt näppäimet sidotaan app-objektin funktioihin. Koodiesimerkki 9 esittää KeyEventtejä, eli näppäinten painalluksista tapahtuvia tapahtumia hallitseva match-lauseke, ja sen eri vaihtoehdoissa tapahtuvia funktiokutsuja.

```
match key_event.code {
     KeyCode::Enter => {
          app.change_song(app.highlighted as u32);
     }
     KeyCode::Char('q') => {
          app.quit();
     }
     KeyCode::Char('j') => {
          let mult = app.command_mult();
          app.move_down(mult);
          app.command_mult = None;
     },
     _ => {}
}
```

Esimerkkikoodi 9: Lyhennetyn KeyEventtejä hallitsevan match-lausekkeen neljä eri vaihtoehtoa. Enter vaihtaa kappaletta, q lopettaa ohjelman ja j liikkuu listassa alaspäin kertoinnumeron määräämän verran.

Ohjelmassa on kaksi eri syötettä hallitsevaa tilaa, Normal ja Insert. Normal-moodissa näppäimet hallitsevat linjojen välillä liikkumista ja Insert-moodissa voidaan syöttää tekstiä suoraan, tässä vaiheessa projektia vain hakukenttään.

5.6 Mpd

MDP:n, eli musiikkisoittimen back end -taustaohjelman hallintaa varten käytetty kirjasto on kiedottu omaan Mpdw-rakenteeseen, jotta yleisiä toimintoja voidaan hoitaa helpommin. Mpdw sisältää Mpd-kirjaston Client-objektin, joka pitää yhteyttä MPD-taustaprossiin. Mpdw sisältää myös ohjelman tilatietoja liittyen kappaledataan. Esimerkkikoodi 10 esittää yhtä Mpdw-rakenteeseen tehtyä funktiota laskemaan kappaleen edistymistä kuvaava desimaaliluku 0-1.

```
pub fn song_progress(&mut self) -> Option<f32> {
     match self.conn.status() {
          Ok(status) => {
               match status.time {
                    Some((played, total)) => {
                         Some((played.as_secs() as f32) /
                              (total.as_secs()) as f32)
                    },
                    None => None,
               }
          },
          Err(err) => {
               println!("{}", err);
               None
          },
     }
}
```

Esimerkkikoodi 10: Funktio Mpdw-rakenteessa kappaleen edistymisen prosenttiarvoa esittävälle desimaaliluvulle 0 – 1. Tämän luvun avulla voidaan esimerkiksi piirtää edistymispalkki.

5.7 Ongelmatilanteet

Merkkikoot ja erikoismerkit

Tekstin pituuden määrittäminen on usein tarpeellista asettelussa, mutta ei ollenkaan yksinkertaista. Komentopäätteessä käytetään normaalisti monospace-fontteja, jolloin jokainen merkki vie saman verran tilaa, mutta tämäkin pätee latinalaisiin aakkosiin eikä esimerkiksi kiinan- tai japaninkielisiin merkistöihin. Komentopääte koostuu tekstisoluista ja latinalaiset aakkoset vievät yhden solun verran tilaa, mutta ovat vain puolikkaan levyisiä. Kiinalaiset merkit taas ovat täysileveitä ja vievät 2 solua. Asia monimutkaistuu vielä enemmän emojien kohdalla. [15.]

Ottaen huomioon kappaleiden monikielisyys ja merkistön vaihtelevuus, oli Vimua varten tarpeellista lisätä tarkoitusta varten luotu kirjasto määrittämään tekstin pituus.

6 Lopputuloksen arviointi ja jatkokehitys

Ohjelman perusominaisuudet ovat valmiita, mutta moni ratkaisu suunnittelussa täytyy työstää uudestaan, nykyisiä ominaisuuksia on tarve laajentaa ja kokonaan uusia ominaisuuksia on tarve lisätä.

6.1 Laajat refaktoroinnit

Uuden kielen ja kirjaston tuomat haasteet aikarajoitusten puitteissa johtivat ratkaisuihin, jotka vaativat suurempia refaktorointeja ennen uusien ominaisuuksien tuomista ohjelmistoon. Nykyinen rakenne ei myöskään tue testausta tarpeeksi helposti, koska App-rakenteen kaikki funktiot muokkaavat itse rakennetta ottamatta parametrejä.

6.1.1 App-rakenteen vastuunjakoa eri osiin

Tällä hetkellä ohjelman päärakenne App hallitsee ohjelmaa liikaa ja sen kasvaessa siitä voi tulla hyvin vaikeasti hallittava. Eri funktioita tulisi lajitella omiin erillisiin tiedostoihin ja sen jälkeen importata päärakenteeseen

6.1.2 Tilan hallinnan keskittäminen

App-päärakenne hallitsee myös tilaa liian hajanaisesti. Rakennetta uudelleen suunnitellessa tavoitteena on ryhmittää eri tilaa määritteleviä kenttiä Struct- tai Enum-muotoon. Esimerkiksi Normal- ja Insert-tilojen hallintaan sopisi paremmin pelkän numeron sijaan InputMode Enum, mutta sekin vaatii refaktorointia ja muutoksia olemassa oleviin, Ratatuin tarjoamiin Widgetteihin.

6.2 Lisäominaisuudet

6.2.1 Haku

Tällä hetkellä haku onnistuu vain artistin nimen perusteella, mutta käyttäjän musiikkitietokannassa voi realistisesti olla satojatuhansia kappaleita. Rajaaminen vaatii hakuominaisuuden uudelleensuunnittelua kaksiosaiseksi, jolloin hakuvälilehdessä voi vaihtaa hakukenttänäkymän ja hakutulosnäkymän välillä.

6.2.2 Soittolistat

Tällä hetkellä ainoa hallittu soittolista on nykyinen soittojono. MPD ja sen Rust-kirjasto kuitenkin mahdollistavat soittolistojen tallentamisen, minkä myötä jonoon voi lisätä tai sen voi korvata kokonaan uudella listalla helposti.

6.3 Visuaaliset parannukset

Väreillä voitaisiin indikoida esimerkiksi nykyistä syötemoodia, mutta ennen sitä moodien hallinta pitää refaktoroida laajamittaisesti.

6.4 Konfiguraatio

Tällä hetkellä ohjelman toiminnot ovat kovakoodattuna. Mikäli ohjelma asetetaan julkiseksi, on tarpeellista mahdollistaa sen konfiguraatio jollain sopivalla konfiguraatiotiedostolla. Konfigurointi sisältäisi esimerkiksi kielen vaihdon, välilehtien uudelleenjärjestelyn ja näppäinkytkösten vaihtamisen.

6.5 Dokumentointi

Ajanpuutteessa dokumentointi jäi liian vähäiseksi, ja ennen kuin työtä jatketaan pidemmälle, on syytä dokumentoida jo tehtyä koodia. Lisäksi kopioitu ja muokattu listatoteutus vaatisi sekä dokumentaation muokkauksen että testien korjauksen.

7 Yhteenveto

Insinöörityön tarkoituksena oli luoda näppäimistövetoinen musiikkisoitin päätteelle modaalisella käyttöliittymällä.

Työ aloitettiin selittämällä, mikä on modaalinen käyttöliittymä ja miten se toimii muissa sovelluksissa. Työ määrittelee myös, miten ohjelman on tarkoitus toimia ja mitä teknologioita siihen käytettiin. Työ eteni ensin suunnitteluvaiheesta itse toteutukseen ja tuloksena saatiin toimiva musiikkisoitin sen pääasiallisilla toiminnallisuuksilla, eli musiikin soittaminen, listan muokkaaminen, määritellyllä navigointitavalla navigointi ja musiikin haku.

Työn edetessä kuitenkin oppi paljon ja monta asiaa täytyy kirjoittaa uusiksi, etenkin kun aletaan laajentamaan ohjelman ominaisuuksia ja tuomaan uusia ominaisuuksia.

Lähteet

1 Toomey, Chris. 2015. Mastering the Vim Language. Verkkoaineisto. Thoughtbot. <https://www.youtube.com/watch?v=wlR5gYd6um0> 29.5.2015. Katsottu 1.11.2023.

2 Rust Programming Language. 2024. Verkkoaineisto. Rust-Lang. <https://www.rust-lang.org> Luettu 01.10.2024.

3 Abdullahi, Munir Adavize. 2024. Rust Vs. Other Programming Languages: What Sets Rust Apart? Verkkoaineisto. Strapi.io. <https://strapi.io/blog/rust-vs-other-programming-languages-what-sets-rust-apart> 13.6.2024. Luettu 31.10.2024.

4 Functional Language Features: Iterators and Closures. 2024. Verkkoaineisto. The Rust Programming Language. <https://doc.rust-lang.org/book/ch13-00-functional-features.html> Luettu 7.10.2024.

5 References and Borrowing. 2024. Verkkoaineisto. The Rust Programming Language. <https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html> Luettu 7.10.2024.

6 Hello, Cargo! 2024. Verkkoaineisto. The Rust Programming Language. <https://doc.rust-lang.org/book/ch01-03-hello-cargo.html> Luettu 7.10.2024.

7 Appendix G - How Rust is Made and “Nightly Rust”. 2024. Verkkoaineisto. The Rust Programming Language. <https://doc.rust-lang.org/book/appendix-07-nightly-rust.html> Luettu 8.10.2024.

8 User’s Manual – Music Player Daemon 0.24~git documentation. 2021. Verkkoaineisto. Readthedocs.io. <https://mpd.readthedocs.io/en/latest/user.html#introduction> Luettu 1.10.2023.

9 User’s Manual – Music Player Daemon 0.24~git documentation. 2021. Verkkoaineisto. Readthedocs.io. <https://mpd.readthedocs.io/en/latest/protocol.html> Luettu 1.10.2023.

10 Comparisons of Backends | Ratatui. 2024. Verkkoaineisto. Ratatui. <https://ratatui.rs/concepts/backends/comparison/> Luettu 1.6.2024.

11 Rendering under the hood | Ratatui. 2024. Verkkoaineisto. Ratatui. <https://ratatui.rs/concepts/rendering/under-the-hood/> Luettu 1.6.2024.

12 Introduction to Widgets | Ratatui. 2024. Verkkoaineisto. Ratatui. <https://ratatui.rs/concepts/widgets/> Luettu 1.6.2024.

13 Templates for bootstrapping a Rust TUI application with Ratatui. 2024. Verkkoaineisto. Github. <https://github.com/ratatui/templates> Päivitetty 28.6.2024. Luettu 29.6.2024.

14 Semantic Versioning 2.0.0 | Semantic Versioning. 2023. Verkkoaineisto. Semver. <https://semver.org/> 19.12.2023. Luettu 10.10.2024.

15 Does Rust's String have a method that returns the number of characters rather than the number of bytes? 2017. Verkkoaineisto. Stackoverflow. <https://stackoverflow.com/questions/46290655/does-rusts-string-have-a-method-that-returns-the-number-of-characters-rather-th> Päivitetty 23.9.2023. Luettu 11.10.2024.