Aalto-yliopisto
27.4.2022

Kanban-sovellus

CS-C2120 OHJELMOINTISTUDIO 2: PROJEKTI

SAMULI YLI-SALOMÄKI (909509)

Sisällys

1. Henkilötiedot

Kanban-sovellus
Samuli Yli-Salomäki
909509
Tietotekniikka, 1. vuosi
27.4.2022

2. Yleiskuvaus

Olen luonut Kanban-sovelluksen, jota käytetään graafisella käyttöliittymällä. Sovelluksella käyttäjä voi luoda ja käyttää useita kanban-lautoja. Näkyvää lautaa voi vaihtaa valikosta. Laudoille käyttäjä voi antaa nimen sekä valita taustakuvan (kuva tai väri). Laudalla käyttäjä voi luoda listoja, joille voi määrittää reunan nimen ja värin. Näihin listoihin käyttäjä voi luoda kortteja. Korteissa on tekstiä ja niille voi asettaa reunojen ja tekstin värin, lisätä aikarajan, tarkistuslistoja sekä erilaisia liitteitä. Korteille voi myös asettaa tunnisteita, joiden perusteella niitä voi suodattaa. Korttia klikkaamalla siitä näytetään paljon lisää tietoja ja aiempia tietoja tarkemmin. Tällöin esimerkiksi tarkistuslistan kohtia voi raksia sekä liitteitä avata nappia painamalla. Lisäksi kaikkia edellä mainittuja lautojen, listojen sekä korttien ominaisuuksia voi muokata jälkeenpäin ja näitä kaikkia voidaan poistaa.

Laudalla kortteja voi siirtää listasta toiseen sekä listan sisällä klikkaamalla. Lisäksi listojen keskinäistä järjestystä pystyy muutamaan.

Jokaisella taululla on oma arkisto, jonne pystyy tallentamaan kortteja myöhempää käyttöä varten. Arkistosta voi palauttaa kortteja takaisin laudalle. Lisäksi ohjelma tukee korttien mallipohjia (template), joiden pohjalta kortteja voi luoda. Mistä tahansa kortista voidaan tehdä uusi mallipohja.

Kaikki lautojen tiedot voidaan tallentaa json-formaattiin perustuvaan tiedostoon. Tällä tiedostolla pystyy laudan avaamaan myöhemmin uudelleen. Tiedostonkäsittelyyn liittyvät mahdolliset IO-poikkeukset käsitellään ohjelman toimesta.

Työ on mielestäni toteutettu haastavalla vaikeusasteella, sillä se sisältää kaikki vaaditut ominaisuudet ja käyttökokemus on sujuva ja ohjelman käyttö intuitiivista.

3. Käyttöohje

Ohjelma käynnistetään kansiosta src/main/scala/kanban/ui löytyvästä **Main**-tiedostosta.

Tällöin aukeaa ohjelman graafinen käyttöliittymä, jolla ohjelmaa käytetään. Ohjelman käynnistyessä tulee ruutu, josta voi joko luoda ensimmäisen laudan tai avata laudan tiedostosta. Tämän jälkeen avautuu varsinainen lautanäkymä. Näkymän yläosassa on työkalupalkki, josta löytyy keskeisimmät napit (järjestyksessä): File-nappi tiedoston tallennusta ja avaamista varten, valikkonappi lautojen välillä vaihtamista varten, New Board sekä Edit Board -napit uuden laudan ja vanhan laudan muokkaamista varten, New List -nappi uuden listan luomiseen. Archive ja Templates -napit arkiston ja mallipohjien hallintaa varten sekä Manage Tags -nappi ja Filter-valikkonappi, joilla ensimmäisellä hallinnoidaan tunnisteita sekä jälkimmäisellä voidaan laittaa päälle suodatin, jolla vain tiettyjen tunnisteiden omaavat kortit ovat näkyvillä.

Työkalupalkin alla on varsinainen lauta. Siinä on listoja, joiden yläosassa on nappeja. Napilla New Card voidaan luoda uusi kortti ja painamalla napin oikean reunan nuolta avautuu mahdollisuus tuoda kortti arkistosta tai luoda kortti mallipohjaa käyttämällä. Napilla Edit voidaan muokata listaa, Delete-napilla poistaa lista (deaktivoituna, jos vain 1 lista) sekä nappia Move painamalla voi listan laitaa siirtotilaan. Tällöin jos klikkaa kahden listan välissä olevaa kohtaa, siirtyy lista siihen. Nappien alla on listan nimi, jonka alla ovat varsinaiset kortit.

Kortista näkyy tietoja riippuen mitä kaikkia ominaisuuksia kortilla on käytössä. Siinä näkyy kortin teksti sekä mahdollisesti aikaraja, kortin tehtävien edistysmistaso sekä tietoa liitteistä. Korttia klikkaamalla siitä tulee aktiivinen, jolloin tietoa näytetään enemmän ja tarkemmin sekä liitteitä voi avata. Näkyvillä on tällöin myös napit Edit, Delete ja Archive, joilla korttia voi muokata, sen poistaa ja arkistoida. Kun kortti on aktivoituna voi klikkaamalla tyhjää väliä korttien välissä tai ensimmäisen kortin yläpuolella / viimeisen kortin alapuolella siirtää kortin tähän väliin.

Näillä toiminnoilla voi laudasta tehdä halutunlaisen sekä lisäillä listoja ja niihin kortteja ja näin käyttää Kanban-lautaa sen tarkoitukseen.

4. Ohjelman rakenne

Varsinaisista ongelman osa-alueita kuvaavista luokista ylimpänä on luokka Kanban, jonka kautta päästään käsiksi kaikkeen tietoon. Se mallintaa yhtä käyttäjän istuntoa ja sillä on suoraan tiedossaan Kanban-laudat, jotka istunnossa on luotu sekä kaikki mahdolliset luodut tunnisteet ja mallipohjat. Sillä on monia get-metodeita, jotta etenkin edellä mainittuihin tietoihin päästään käsiksi. Sillä on lisäksi metodit uusien taulujen, tunnisteiden sekä mallipohjien luontiin ja poistoon. Lisäksi metodilla getAllCards saadaan kaikki mahdolliset kortit jokaisesta laudasta, jotta niille voidaan tarvittaessa suorittaa operaatioita, kuten metodi checkFiles. Tämä metodi varmistaa, että kaikki tiedostot löytyvät edelleen paikasta, jossa niiden pitäisi olla eikä niitä ole esimerkiksi siirretty.

Luontevasti Kanban-luokan alla on itse varsinaista lautaa mallintava luokka Board. Yhdellä Kanban-oliolla on oltava ainakin yksi Board-olio ja yksi Board-olio kuuluu tasan yhteen Kanban-olioon. Board-luokan oliolla on nimi, taustan väri, arkisto, joukko listoja sekä mahdollinen taustakuva. Jälleen luokalla on monia get-metodeja, jotta näihin tietoihin päästään käsiksi ja muutama set-metodi, jotta esimerkiksi nimeä ja taustakuvaa/väriä voidaan muuttaa. Lisäksi sillä on metodit moveCard ja moveColumn, joilla voidaan siirtää kortin sijaintia listoista toiseen sekä muuttaa listojen keskinäistä järjestystä. Luokalla on myös metodit, jotta uusia listoja voidaan luoda ja vanhoja poistaa.

Tämän luokan alla toimii laudan listaa mallintava luokka Column. Board-luokan tavoin yhdellä Board-oliolla on oltava ainakin yksi Column-olio ja yhteen Column-olioon kuuluu täsmälleen yhteen Board-olioon. Column-luokan oliolla on tiedossaan sen nimi, reunan väri sekä siinä olevat kortit. Aiempien tavoin luokalla on get-metodit, joiden kautta näihin tietoihin päästään käsiksi. Lisäksi sillä on metodit uusien korttien lisäämiseen ja vanhojen poistoon.

Column-luokan alla toimii varsinaista listan korttia mallintava luokka Card. Sillä on aiempia luokkia enemmän tietoa: korttioliolle on tallennettu kortin teksti, tekstin sekä reunan väri, tunnisteet, tarkistuslista sekä mahdollinen aikaraja ja mahdolliset liitteet (tiedosto, alikortti, URL). Luokka vaatii siis jälleen monia get-metodeja näitä tietoja varten. Lisäksi tarvitaan metodit tunnisteiden lisäämistä ja poistoa varten sekä kortin muokkausta varten. Kortilla on myös metodi toSub, jolla kortti voidaan muuttaa alikortiksi. Tämän toiminnasta ja luokan tarpeellisuudesta selitetään lisää alempana.

Ohjelman pääluokkien hierarkia on luonnollinen ja yksinkertainen. Yhdellä Kanban-oliolla on sisällään Board-olioita, Boardilla Columneja ja Columneilla Cardeja. Näin saadaan ohjelman pääongelma mallinnettua ja ylimmästä päästään käsiksi myös alimpaan. En siis näe näiden osalta mitään vaihtoehtoista ratkaisumallia.

Korteilla on tarkistuslista, jota mallinnetaan luokalla Checklist. Tämän luokan oliolla on tiedossaan osatehtävät, joista tarkistuslista koostuu. Tätä varten luokalla on muutama get-metodi tämän tiedon saamiseksi, mutta myös esimerkiksi prosentuaalisen edistymisen laskemiseksi. Metodeja on myös osatehtävien lisäämistä ja poistamista varten sekä yksittäisen osatehtävän totuusarvon, joka kuvaa onko tehtävä tehty, vaihtamista varten. Lisäksi luokalla on metodi, jolla määritetään, miten tarkistuslista esitetään merkkijonona.

Korteilla voi lisäksi olla aikaraja, jota kuvaa luokka Deadline. Tämän luokan olioiden tärkein tieto on aikarajan päivämäärä. Lisäksi sillä on status, joka kuvaa onko kortin tehtävä tehty, johon aikaraja liittyy. Tätä käytetään, mikäli kortilla ei ole osatehtäviä. Jälleen on get-metodeita, jotta näihin tietoihin päästään käsiksi sekä metodi toggleStatus, jolla totuusarvoa voidaan vaihtaa. Tarkistuslistan tapaan on määritetty, miten aikaraja esitetään merkkijonona. Tätä varten on muutama yksityinen metodi päivien laskentaan.

Näillä kahdella luokalla saattaisi olla vaihtoehtoisia toteutustapoja, jotka saattaisivat olla vielä yksinkertaisempia. En kuitenkaan usko, että mahdolliset vaihtoehtoiset ratkaisutavat olisivat merkittävästi erilaisia tai parempia.

Korteilla on ominaisuus, jolla niille voi laittaa liitteeksi toisen kortin. Olisikin ollut kaikista järkevintä, että Card-olion tietoihin tallennettaisiin viittaus toiseen Card-olioon, joka toimisi liitteenä ja näin asia oli suunnitelmassa tarkoitus hoitaakin. Toteutuksen aikana törmäsin ongelmaan, missä json-koodekit, joista kerrotaan myöhemmin enemmän, eivät jostain syystä hyväksyneet tätä toteutusta. En siis pystynyt muuttamaan Card-luokkaa json-formaattiin, jos osana tallennusta kortista tallennettaisiin toinen samaa Card-luokkaa oleva kortti. Ongelmaan olisi mitä luultavammin ollut jokin toinen ratkaisu, mutta en itse tätä onnistunut löytämään. Näin ollen jouduin luomaan luokan Subcard, joka perii Card-luokan. Se luodaan kortin ominaisuuksilla ja sillä on metodi, jolla saa Subcard-olion palautettua korttina. Luokka ei lisännyt mitään uutta eikä sille olisi pitänyt olla mitään tarvetta, mutta onnistuin kuitenkin sen avulla kiertämään circe-kirjaston luoman rajoitteen.

Käyttöliittymän keskiössä on olio Main, jossa oli lautaa kuvaavan päänäkymän koodi. Tämän tiedoston rivimäärä on melko suuri, mutta kaikkea koodia tarvittiin vain tämän yhden näkymän näyttämiseen, joten tiedostoa ei voinut jakaa osiin. Tiedosto sisältää pääosin käyttöliittymäkomponentteja ja yksityisiä muuttujia ja metodeja näiden komponenttien tuottamiseen eri tavoin jokaista korttia/listaa/lautaa varten. Oliolla on yksityiset metodit update ja fullUpdate, joista ensimmäinen hoitaa yleisen käyttöliittymän näkymän päivityksen muutosten yhteydessä ja jälkimmäinen muutaman muun asian päivityksestä tiedostosta ladatun istunnon yhteydessä. Lisäksi mukana on julkinen metodi drawAlert, jota myös myöhemmin mainitut dialogi-ikkunoiden oliot käyttävät varoitusikkunoiden luomiseen sekä muita yksityisiä metodeja, jotka esimerkiksi suorittavat esimerkiksi laskentaa komponenttien kokoa varten tai tuottavat taustoja ScalaFX:n luokkia käyttäen.

Käyttöliittymän alaisuudessa toimii myös FileHandler-luokka, joka huolehtii tiedoston lukemisesta sekä tiedostoon kirjoittamisesta. Näistä ensimmäiseen on metodi save, joka ensin avaa ikkunan, jolla käyttäjä valitsee tallennussijainnin sekä tiedoston nimen. Sitten metodi muuttaa luotuja json-koodekkeja käyttäen Kanban luokan (ja siten kaiken mitä sen alla on) json-formaattiin ja tallentaa sen tiedostoon. Poikkeukset käsitellään myös metodin toimesta ja mahdollisista virhetilanteista ilmoitetaan käyttäjälle. Jälkimmäisestä vastaa metodi load, joka avaa tiedostonvalintaikkunan ja lukee tiedoston sisällön. Tämän jälkeen luettu tieto yritetään jälleen json-koodekkeja käyttäen muuttaa takaisin Kanban-luokan olioksi, jota voidaan taas

normaalisti käyttää. Näissä vaiheissa jälleen metodi huolehtii mahdollisista poikkeuksista. Tiedostonkäsittely on merkittävässä osassa, joten sitä on järkevä kuvata luokalla ja en usko, että olisi ollut merkittävästi erilaista vaihtoehtoista ratkaisua.

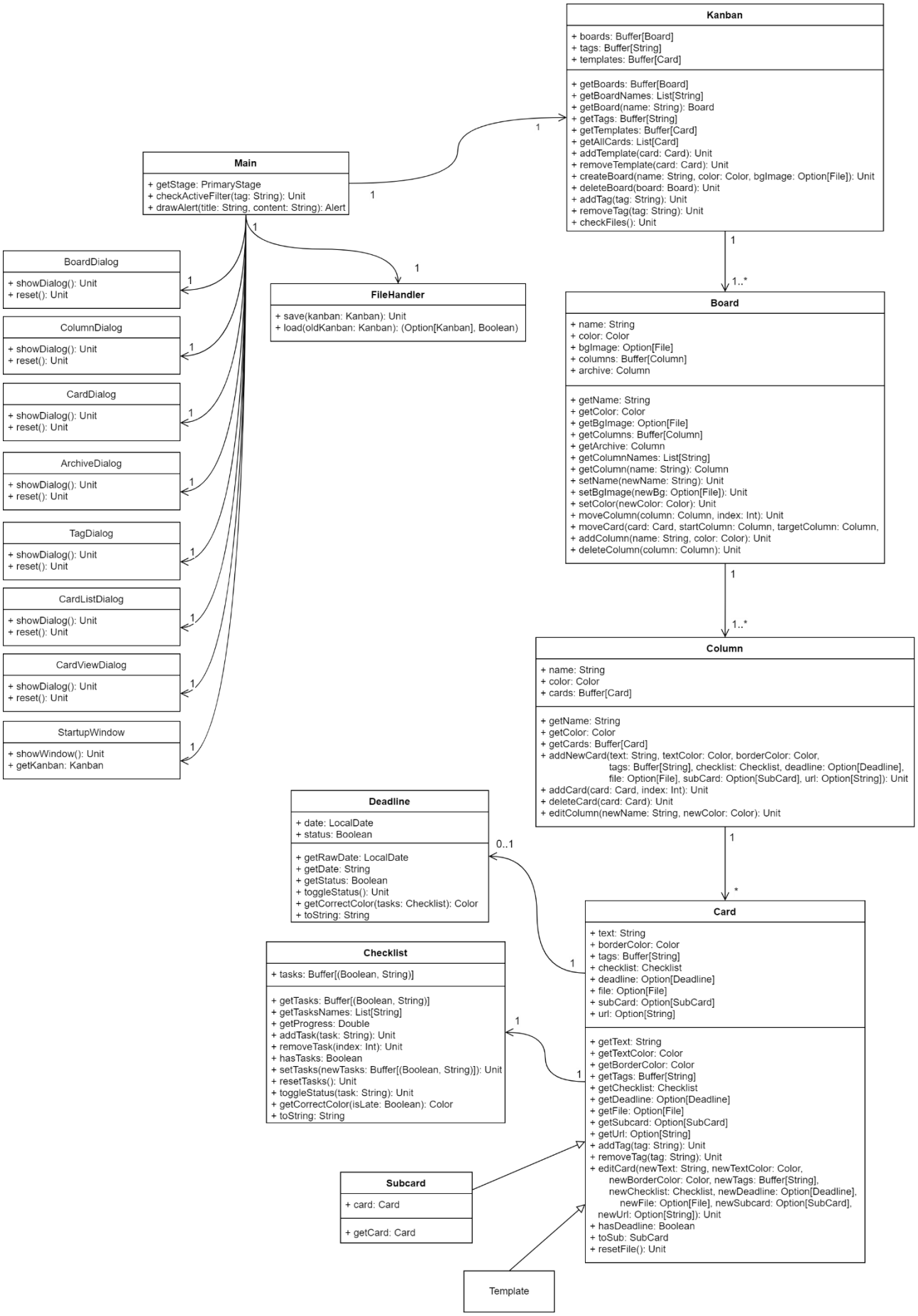
Ohjelmassa käytetään paljon dialogi-ikkunoita korttien/listojen/lautojen luomiseen ja muokkaamiseen sekä arkiston, mallipohjien ja tunnisteiden hallintaan. Kukin näistä ikkunoista on oma olionsa, jotta koodi saatiin hajautettua järkevästi. Alun perin oli tarkoitus, että näillä ikkunaluokilla olisi yliluokka, josta ne periytyisivät. Tämä kuitenkin aiheutti monin paikoin ajonaikaisia poikkeuksia, jotka johtivat ohjelman kaatumiseen ja joita en saanut korjattua. Näin ollen oliot sisältävät joiltain osin samoja ominaisuuksia, minkä takia yliluokan käyttö olisi voinut olla järkevää, mutta aiemmin mainituista syistä näin ei ole voitu tehdä. Main-olion tapaan nämä sisältävät monia käyttöliittymäkomponenttien tekoon liittyviä yksityisiä muuttujia ja metodeita. Niiltä löytyy julkinen metodi showDialog, jolla Main-tiedoston koodista saadaan kyseinen ikkuna näkyviin. Näillä on lisäksi metodi reset, jolla ikkuna alustetaan joko muokkauksen tapauksessa muokattavan kohteen tiedoilla tai uutta luotaessa tyhjillä tiedoilla. Lisäksi monissa on yksityinen metodi update, jolla ruudulle saadaan päivitettyä muokattuja tietoja.

Alkuperäisessä suunnitelmassa kortin liitteille oli oma luokkansa Attachment. Kuitenkin, koska liitteet olivat vain kortin yhteen muuttujaan tallennettu ominaisuus, ei oma luokka tuonut siihen mitään hyödyllistä. Olikin vain paljon yksinkertaisempaa, että jokainen liitetyyppi oli Option-muuttujaan tallennettuna, jolloin sillä joko oli liite tai ei ollut.

Arkisto oli myös alkuperäisessä suunnitelmassa oma luokkansa. Kuitenkin koska arkiston toiminta oli täsmälleen samanlainen kuin tavallisella listalla, voitiin ne kuvata vain niitä käyttämällä.

Alun perin tunnisteita oli tarkoitus kuvata omalla luokallaan Tag. Tämän hyötynä olisi ollut se, että kukin tunniste tietäisi itse missä kaikissa korteissa se on. Johtuen laudan tiedostoon tallentamiseen liittyvästä seikasta olisi tämä tieto menetetty, joten luokalle ei ollut enää mitään tarvetta. Näin ollen riitti, että jokainen tunniste oli vain merkkijono ja koska kortteja ei koskaan tulisi olemaan valtavaa määrää, ei tunnisteiden poistamisen korteista ole suorituskyvyn puolesta rasittavaa, vaikka joudutaankin käymään hieman suurempi määrä kortteja läpi.

Luokkarakennetta kuvaava UML-kaavio:


Kanban
+ boards: Buffer[Board]
+ tags: Buffer[String]
+ templates: Buffer[Card]
+ getBoards: Buffer[Board]
+ getBoardNames: List[String]
+ getBoard(name: String): Board
+ getTags: Buffer[String]
+ getTemplates: Buffer[Card]
+ getAllCards: List[Card]
+ addTemplate(card: Card): Unit
+ removeTemplate(card: Card): Unit
+ createBoard(name: String, color: Color, bgImage: Option[File]): Unit
+ deleteBoard(board: Board): Unit
+ addTag(tag: String): Unit
+ removeTag(tag: String): Unit
+ checkFiles(): Unit
Main
+ getStage: PrimaryStage
+ checkActiveFilter(tag: String): Unit
+ drawAlert(title: String, content: String): Alert
BoardDialog
+ showDialog(): Unit
+ reset(): Unit
ColumnDialog
+ showDialog(): Unit
+ reset(): Unit
CardDialog
+ showDialog(): Unit
+ reset(): Unit
ArchiveDialog
+ showDialog(): Unit
+ reset(): Unit
TagDialog
+ showDialog(): Unit
+ reset(): Unit
CardListDialog
+ showDialog(): Unit
+ reset(): Unit
CardViewDialog
+ showDialog(): Unit
+ reset(): Unit
StartupWindow
+ showWindow(): Unit
+ getKanban: Kanban
FileHandler
+ save(kanban: Kanban): Unit
+ load(oldKanban: Kanban): (Option[Kanban], Boolean)
Board
+ name: String
+ color: Color
+ bgImage: Option[File]
+ columns: Buffer[Column]
+ archive: Column
+ getName: String
+ getColor: Color
+ getBgImage: Option[File]
+ getColumns: Buffer[Column]
+ getArchive: Column
+ getColumnNames: List[String]
+ getColumn(name: String): Column
+ setName(newName: String): Unit
+ setBgImage(newBg: Option[File]): Unit
+ setColor(newColor: Color): Unit
+ moveColumn(column: Column, index: Int): Unit
+ moveCard(card: Card, startColumn: Column, targetColumn: Column,
+ addColumn(name: String, color: Color): Unit
+ deleteColumn(column: Column): Unit
Column
+ name: String
+ color: Color
+ cards: Buffer[Card]
+ getName: String
+ getColor: Color
+ getCards: Buffer[Card]
+ addNewCard(text: String, textColor: Color, borderColor: Color,
tags: Buffer[String], checklist: Checklist, deadline: Option[Deadline],
file: Option[File], subCard: Option[SubCard], url: Option[String]): Unit
+ addCard(card: Card, index: Int): Unit
+ deleteCard(card: Card): Unit
+ editColumn(newName: String, newColor: Color): Unit
Deadline
+ date: LocalDate
+ status: Boolean
+ getRawDate: LocalDate
+ getDate: String
+ getStatus: Boolean
+ toggleStatus(): Unit
+ getCorrectColor(tasks: Checklist): Color
+ toString: String
Checklist
+ tasks: Buffer[(Boolean, String)]
+ getTasks: Buffer[(Boolean, String)]
+ getTasksNames: List[String]
+ getProgress: Double
+ addTask(task: String): Unit
+ removeTask(index: Int): Unit
+ hasTasks: Boolean
+ setTasks(newTasks: Buffer[(Boolean, String)]): Unit
+ resetTasks(): Unit
+ toggleStatus(task: String): Unit
+ getCorrectColor(isLate: Boolean): Color
+ toString: String
Card
+ text: String
+ borderColor: Color
+ tags: Buffer[String]
+ checklist: Checklist
+ deadline: Option[Deadline]
+ file: Option[File]
+ subCard: Option[SubCard]
+ url: Option[String]
+ getText: String
+ getTextColor: Color
+ getBorderColor: Color
+ getTags: Buffer[String]
+ getChecklist: Checklist
+ getDeadline: Option[Deadline]
+ getFile: Option[File]
+ getSubcard: Option[SubCard]
+ getUrl: Option[String]
+ addTag(tag: String): Unit
+ removeTag(tag: String): Unit
+ editCard(newText: String, newTextColor: Color,
newBorderColor: Color, newTags: Buffer[String],
newChecklist: Checklist, newDeadline: Option[Deadline],
newFile: Option[File], newSubcard: Option[SubCard],
newUrl: Option[String]): Unit
+ hasDeadline: Boolean
+ toSub: SubCard
+ resetFile(): Unit
Subcard
+ card: Card
+ getCard: Card
Template
1
1..*
0..1
*

5. Algoritmit

Merkittävin algoritmi on käytössä korttien (ja listojen) siirtämisessä. Kun kortit piirretään käyttöliittymään, on niiden välissä tyhjiä aukkoja, jotka ovat oma käyttöliittymäkomponenttinsa. Näiden komponenttien ID:t tallennetaan tietorakenteeseen, jotta niiden järjestys tunnetaan. Kun kortti halutaan siirtää ja käyttäjä klikkaa jotain näistä aukoista, algoritmi selvittää mihin kohtaan kortti siirretään. Yksinkertaisimmassa tapauksessa, kun kortti siirretään toiseen listaan, algoritmi katsoo kuinka mones aukko, jota klikattiin, on ja poistaa kortin nykyisestä listasta ja laittaa sen uuteen listaan aukon järjestysnumeron indeksille. Tilanne on hankalampi, jos kortti siirretään saman listan sisällä. Tällöin korttia ei voi siirtää automaattisesti aukon järjestysnumeron perusteella. Esimerkiksi jos kortti on listassa ensimmäisenä ja käyttäjä klikkaa kortin alapuolella olevaa aukkoa eli aukkoa numero 2 (indeksinä 1). Tällöin, jos kortti poistetaan listasta ja sijoitetaan indeksille 1, onkin se väärässä paikassa, vaikka siirron ei olisi pitänyt muuttaa kortin paikkaa ollenkaan. Näin ollen on indeksistä vähennettävä yksi tilanteessa, jossa korttia yritetään siirtää oman listansa sisällä paikkaan, jossa ei oikeasti tapahtuisi muutosta. Tämä on huomioitava myös, jos korttia siirretään sen omassa listassa, johonkin sen yläpuolelle samasta syystä. Algoritmi siis laskee oikean paikan, johon kortti pitää sijoittaa.

Edellisessä algoritmissa listojen tapauksessa on viimeisen listan jälkeen mahdollisesti tultava leveydeltään suurempi väli. Näin on tehtävä, jotta tausta näkyy kokonaan ja käytössä onkin suhteellisen yksinkertainen algoritmi, joka määrittää tämän tarvitun leveyden. Algoritmi selvittää laudalla olevien listojen määrän ja niiden leveyden sekä niiden väliin jäävien aukkojen leveyden ja ikkunan senhetkisen leveyden perusteella kuinka paljon tyhjää tilaa jää. On huomioitava myös tilanne, jossa algoritmin perusteella ei tarvita yhtään tilaa, mutta tällöin on kuitenkin oltava hieman leveyttä, jotta saadaan luotua alue, jota käyttäjä voi klikata halutessaan siirtää listan sille paikalle.

Algoritmeja oli myös käytössä esimerkiksi korttien suodatuksessa ja tiedostonhallinnassa. Kun käyttäjä valitsi tunnisteen, jonka perusteella haluaa suodattaa kortteja, niin tällöin kyseinen tunnistemerkkijono lisättiin tietorakenteeseen. Sen jälkeen laudan näkymä päivitetään, jolloin jokaisen kortin kohdalla tarkistetaan, että se tulee näkyviin vain, jos sen tunnisteiden joukossa on tuo kyseinen tunniste. Algoritmi tukee myös usean tunnisteen perusteella suodattamista, jolloin korttien kohdalla varmistetaan, että kaikki suodatuksessa käytössä olevat tunnisteet löytyvät kortin tunnisteista.

Tiedostoon tallentaessa circe-kirjaston avulla tehdyt json-enkooderit muuttavat Kanban-luokan olion json-formaattiin. Koska tämä luokka pitää sisällään kaiken tallennettavan eli etenkin Board-oliot, jotka puolestaan pitävät sisällään Column-oliot ja ne puolestaan Card-oliot, niin näin koko luokkaa tallentuu json-formaatiksi. Tiedoston avaaminen on hieman monimutkaisempi prosessi. Mikäli tiedosto saadaan luettua, yritetään se muuttaa json-dekoodereilla takaisin Kanban-luokan olioksi. Jos tiedosto on kirjoitettu väärällä formaatilla, ei tätä voida tehdä, jolloin tulokseksi laitetaan vanha Kanban-olio. Mikäli tuloksena on tosiaan tämä vanha, tiedetään, että prosessi epäonnistui ja käyttäjälle ilmoitetaan asiasta.

Käytössä on myös algoritmi, jolla muodostetaan laudalla sen tauskuva. Ensin tarkistetaan, onko laudalle tallennettu kuvatiedostoa. Mikäli sitä ei ole, muodostetaan uusi Background-olio käyttämällä Board-olioon tallennettua väriä. Mikäli kuva löytyy, ladataan se tiedostosijainnista. Kuvan koolla on merkitystä ja algoritmi tutkii, mikäli kuvan korkeus on ikkunaa suurempi. Tällöin, jotta koko kuva näkyy, skaalataan se kuvan kuvasuhdetta käyttämällä ikkunan kokoiseksi. Mikäli kuva on pieni, toistetaan kuvaa monta kertaa taustassa, jotta resoluutiosta ei tarvitse joustaa. Lopullinen kuva muutetaan värin tavoin Background-olioksi.

Käyttöliittymän puolelta löytyy myös merkittävä algoritmi, joka vastaa käyttöliittymäkomponenttien muodostuksesta, minkä avulla koko laudan näkymä muodostetaan. Laudan tasolla luodaan vuorotellen lista

ja listojen väliin jäävä aukko. Jokaisen listan kohdalla listalle luodaan nappeja ja teksti listan nimeä varten, jonka jälkeen luodaan kortteja. Kortin tasolla jälleen vuorotellen luodaan kortti ja korttien väliin jäävä aukko. Kortin aktiivisuuden perusteella määräytyy sen reunan tyyli sekä koko ja esitettävät tiedot. Korttiin tallennettua tietoa käydään järjestelmällisesti läpi ja riippuen siitä, mikäli tieto löytyy, luodaan jokaista varten oma käyttöliittymäkomponentti, joka tulee näkyviin.

Edellä kuvattuun algoritmiin liittyy toinen algoritmi, joka suoritetaan aina kun käyttöliittymän näkymä halutaan päivittää. Sen alussa luodaan edellä kuvatun mukaisesti laudan käyttöliittymäkomponentit. Sen jälkeen työkalupalkin napeille ja valikoille päivitetään ajankohtaiset tiedot. Se tarkistaa myös onko jotain tiettyä asioita jo sallittu määrä ja tarvittaessa estää näistä uusien luomisen. Tämän peruspäivitysalgoritmin lisäksi istuntoa vaihdettaessa on tästä laajempi versio, jossa kaiken aiemmin mainitun lisäksi asetetaan laudoista ensimmäisen aktiiviseksi ja luodaan tälle oikea tausta sekä nollataan kaikkien muiden komponenttien aktiivisuus ja suodattimet.

Samankaltaisia algoritmeja käytetään lähes kaikissa dialogi-ikkunoissa, joissa niiden näkymä alustetaan saadun tiedon perusteella. Esimerkiksi vanhan kortin tapauksessa muokkausikkunaan tulee valmiiksi olemassa olevat tiedot, jotta niitä voi helposti muokata, ja uuden tapauksessa tekstikentät ovat tyhjiä ja muissa kohdissa perusarvot. Myös ikkunan tekstit ja joidenkin nappien näkyvyys/käyttömahdollisuus riippuvat parametreista.

6. Tietorakenteet

Valtaosassa tietorakenteissa on ohjelmassa käytössä muuttuvatilainen Buffer. Se on käytössä siis lähes kaikessa, missä pidetään kirjaa monista asioista, kuten Kanban-olion laudoista, Board-olion listoista, Column-olion korteista ja Card-olion tunnisteista. Käytännössä myös kaikki käyttöliittymässä tarvitut tietorakenteet ovat muotoa Buffer. Tämä johtuu siitä, että näissä kaikissa tilanteissa oli tarvetta sille, että tietorakennetta piti pystyä muuttamaan. Oli siis oltava mahdollisuus helposti lisätä ja poistaa asioita. Map-tietorakennetta ei voinut käyttää, sillä niiden keskinäisen järjestyksen oli pysyttävä samana.

Muuttuvatilaisen Buffer-tietorakenteen lisäksi olisi teoriassa voinut käyttää jotakin vastaavaa muuttumatonta tietorakennetta, jolloin lisättäessä ja poistettaessa olisi tilalle luotu aina uusi rakenne ja sijoittaa vanhan tilalle. Tämä ei kuitenkaan olisi lainkaan käytännöllistä ja veisi turhaan resursseja. Muuttuvatilainen Buffer oli siis käytännössä ainoa järkevä vaihtoehto.

Käyttöliittymässä on myös yhdessä kohtaa käytössä Map. Siinä avaimena toimii lista ja sen avulla päästään käsiksi Buffer-tietorakenteeseen, jossa on järjestyksessä korttien rakoina toimivat komponentit, joiden järjestystä käytetään korttien siirtämiseen. Tässä oli hyödyllistä, että juuri tietty lista pystyi toimimaan avaimena ja Map oli siten tietorakenteena käytännöllinen valinta.

Muutamassa tilanteessa on tietojen tallennukseen käytetty tietorakennetta List, joka on tilaltaan muuttumaton. Näin oli esimerkiksi käyttöliittymän kohdissa, joissa ComboBox-komponentille oli annettava nimet listassa. ScalaFX vaati, että listan tyyppinä oli oltava List, joten tästä syystä tätä tietorakennetta on käytetty, kun on ollut tarve antaa parametrina esimerkiksi lista tunnisteiden nimistä. Jokin muukin muuttumaton tietotyyppi, kuten Vector, olisi ollut toimiva, mutta juuri ScalaFX:n omien vaatimusten vuoksi oli tämä tietorakenne valittu. Näissä tilanteissa aina riitti, että tietorakenne oli tilaltaan muuttumaton, sillä listan joukkoon ei ollut tarvetta lisätä uusia elementtejä.

7. Tiedostot ja verkossa oleva tieto

Ohjelman istunto voidaan tallentaa .json-tiedostoon, jossa tieto on esitetty json-formaatissa. Tiedostoon voisi siten halutessaan käsin kirjoittaa kortteja, mutta tämä ei ole käytännöllistä ja lautojen muokkaus onkin suunniteltu tapahtuvan vain käyttöliittymän kautta. Oikeassa tallennustiedossa ei myöskään ole turhia välilyöntejä, sillä formaatti ei ole varsinaisesti tehty ihmisen luettavaksi.

Tiedoston rakenne etenee johdonmukaisesti ylimpänä olevan Kanban luokan tallennuksesta edeten sen lautoihin, lautojen listoihin ja listojen kortteihin. Näistä kaikista tallennetaan niiden tiedot, jotta ne voidaan palauttaa tiedostosta myöhemmin.

Tästä pätkästä on nähtävissä, miten tieto tallennetaan. Esimerkkiin on merkattu, mihin kohtaan tulisi esimerkiksi lisää lautoja, listoja sekä kortteja. Esimerkin vuoksi on näkyvissä kortti, missä ei ole melkein mitään tietoja, ja kortti, jossa on melkein kaikki mahdollinen tieto. Sekaan on myös kommentoitu //-alkuisilla merkinnöille lisää informaatiota ja esitysmuotoa on tiivistetty runsaasti tilan säästämiseksi. Näiden merkkien takia ei tiedostoa voi suoraan käyttää. Ohjelman mukana onkin toimitettu esimerkkitiedosto (*kanban_example.json*), johon on tehty valmiiksi lautoja ja joka esittelee näin ohjelman ja korttien ominaisuuksia.

```
{
  "boards" : [ // alussa istunnon laudat
    {
      "name" : "Lauta 1",
      "color" : { // väreistä tallennetaan kaikkien kolmen värin arvo
        "R" : 0.7019608020782471,
        "G" : 0.9019607901573181,
        "B" : 0.9019607901573181
      },
      "image" : null, // jos arvoa ei ole (pääosin Option-tapauksissa) niin null
      "columns" : [
        { // ensin listan omat tiedot
          "name" : "Lista 1",
          "color" : { // tilan säästämiseksi tästä eteenpäin väriarvot piilotettu
          },
          "cards" : [ // sitten listan kortit
            { // ensin kortti, jossa on tietona käytännössä vain teksti
              "text" : "Testikortti 1",
              "textColor" : {},
              "borderColor" : {},
              "tags" : [],// ei tunnisteita
              "checklist" : {
                "tasks" : [] // ei tehtäviä
              }, // mahdolliset liitetiedostot (Optioneita käyttämällä)
              "deadline" : null, "file" : null,
              "subcard" : null, "url" : null
            },
            { // tässä kortissa on käytössä lähes kaikki ominaisuudet
              "text" : "Testikortti 2",
              "textColor" : {},
              "borderColor" : {},
              "tags" : [ // kaksi tunnistetta listattuna
                "Aalto", "OS2"
              ],
              "checklist" : {
                "tasks" : [
                // osatehtävistä sekä suoritusta kuvaava true/false arvo sekä teksti
                   [ true, "Tee projekti" ],
```

```
                    [ false, "Palauta projekti" ]
                  ]
                },
                "deadline" : { // aikaraja, josta tallennettuna päivä ja status
                  "date" : "2022-04-27",
                  "status" : false // mikäli osatehtäviä on, niiden suoritus määrittää statuksen
                },
                "file" : {
                  "file" : "polku/dokumentti.pdf" // tiedostopolut on merkittynä absoluuttisena
polkuna, jotta ohjelma löytää ne käyttäjän koneelta
                },
                "subcard" : {
                  "card" : {} // liitekortin tiedot esitetty kuten aiemminkin kortille
                },
                "url" : "https://www.aalto.fi/fi"
              }
          ] // tähän voisi tulla perään lisää kortteja
          },
          {
            "name" : "Lista 2",
            "color" : {},
            "cards" : [] // tähän tulisi listan kortit
          }
          // tähän perään voisi tulla lisää listoja
        ],
        "archive" : { // jokaiseen lautaan kuuluu arkisto
          "name" : "Archive",
          "color" : {},
          "cards" : [] // tässä olisi arkiston kortit samalla tavalla kuten aiemminkin
        }
      },
      { // istunnon toinen lauta
        "name" : "Lauta 2",
        "color" : {},
        "image" : {
          "file" : "polku/kuva.png" // laudalla on taustakuva
        },
        "columns" : [
          {
            "name" : "Lista 3",
            "color" : {},
            "cards" : []
          }
        ],
        "archive" : {
          "name" : "Archive",
          "color" : {},
          "cards" : []
        }
      } // tässä voisi perässä olla lisää lautoja
    ],
    "tags" : [ // yleisesti listattuna kaikki istunnossa käytössä olevat tunnisteet (suodatusta varten)
      "Aalto", "OS2"
    ],
    "templates" : [
    ] // tässä olisi mallipohjat tallennettuna
}
```

Ohjelma ei lue tietoja verkosta.

8. Testaus

Kanban-sovelluksen luonteen vuoksi ohjelman testaus tapahtui pääosin käyttöliittymän kautta. Testasin ohjelman käyttöä erilaisin tavoin alkaen laudan normaalista käytöstä tavanomaisissa tilanteissa. Kun ohjelma vaikutti toimivan yleisesti kuten kuuluikin, aloin käyttämään ohjelmaa epäintuitiivisesti ja yritin toteuttaa tilanteita, joissa käyttäjä saattaisi tehdä jotain normaalista poikkeavaa. Tämän myötä löytyi bugeja, jotka näin saatiin korjattua. Aina kun kehitin uutta ominaisuutta, testasin sitä samanaikaisesti, jotta se toimisi odotetulla tavalla. Näin yhden ominaisuuden kehittämisen johti toimivaan lopputulokseen.

Ohjelmassa on suuri määrä erilaisia ominaisuuksia ja tämä aiheuttaa sen, että erilaisia käyttötilanteita on valtava määrä. Kaikkien mahdollisten tilanteiden testaaminen ja kaiken huomioonottaminen osoittautui siis melko hankalaksi. Pyrin kuitenkin mahdollisimman monta asiaa huomioimaan aikarajoitteiden puitteissa ja en siksi koe testaussuunnitelmassa olleen merkittäviä aukkoja. Kuitenkin olisi ollut luultavasti hyödyllisempää saada testaukseen enemmän järjestelmällisyyttä, jotta ominaisuuksien suuri lukumäärä ei olisi vaikuttanut niin paljoa.

Tiedosto-ominaisuuksien osalta suunnitelmassa oletettiin, että se olisi toteutettu eri tavalla, joten sen osalta suunnitelmasta oli poikettava ja testaus tehtävä toisella tavalla. Json-formaatin ja tekemieni koodekkien avulla luokkien formaattiin muuttaminen ja formaatin lukeminen tapahtui automaattisesti, joten sen osalta testattavaa ei ollut. Tehtäväksi jäi varmistaa, että koodekit toimivat täysin oikein ja kaikki data sekä tallennettiin että saatiin takaisin tiedostoa luettaessa. Tätä varten tein oman testausohjelman, jossa loin erilaisia lautoja vaihtelevilla ominaisuuksilla ja ensin varmistin, että muutettu json-tiedosto sisälsi kaiken tarvittavan tiedon. Tämän jälkeen muutin valmiita json-tiedostoja takaisin luokiksi ja varmistin, että ne sisälsivät kaiken alkuperäisen tiedon. Nämä testit ohjelma läpäisi.

Koska ohjelma perustui täysin käyttöliittymään ja varsinaisten logiikkaan liittyen luokkien toteutus oli suhteellisen yksinkertainen, ei tarvetta yksikkötesteille oikeastaan ollut.

9. Ohjelman tunnetut puutteet ja viat

Ohjelman suurimpana puutteena tehtävänannon suhteen pidän korttien siirtelyyn käytetyn drag-and-drop-ominaisuuden puuttumista. Nykyisellään korttien siirtely toteutetaan klikkaamalla. Mikäli projektia jatkettaisiin, voitaisiin tämä ominaisuus toteuttaa. Lisäajalla olisi mahdollista tutustua ja oppia enemmän käytössä olleesta käyttöliittymäkirjastosta ja näin voitaisiin löytää ratkaisuja, miten saataisiin tunnistettua paremmin raahauksen alku- ja loppupiste. Se tosin saattaisi vaatia, että käyttöliittymän rakenne olisi erilainen, sillä nykyisellään en itse onnistunut löytämään tapaa, jolla tällä rakenteella näitä asioita voitaisiin oikein havaita. Olisi siis mahdollista, että olisi tehtävä suuria aikaa vaativia rakenteellisia muutoksia ja esimerkiksi toteuttaa lauta siten, että kohteita (kuten kortteja) piirrettäisiin sinne koordinaatteihin perustuen ja näitä käsiteltäisiin ennemminkin siten, että piirrettäisiin suorakulmioita graafisesti ja näitä käytettäisiin kortteina. Näin voisi olla helpompi tämä ominaisuus toteuttaa.

Käyttöliittymän päivitykseen liittyy myös ongelma. Koska koko lauta päivitetään muutosten jälkeen, saattaa se aiheuttaa heikentymistä suorituskyvyssä, jos käyttäjä lisää merkittävän määrän asioita laudalle. Tämä voitaisiin korjata, että päivittäminen kohdistettaisiin tarkemmin vain siihen kohteeseen, jota muutos koskettaa, eikä koko lautaan. Tämä olisi suorituskyvyn puolesta paljon parempi.

Ohjelma sisältää ominaisuuden, jossa kortille voidaan tallentaa liitteeksi tiedosto tai nettisivun osoite. Ohjelma mahdollistaa näiden avaamisen laitteen oletusohjelmalla nappia painamalla. Omalla koneellani (Win10) ominaisuus toimii täysin, mutta huomasin, että eräällä Linux-laitteella napin klikkaaminen aiheuttaa ohjelman jäätymisen. En siis tiedä kuinka montaa laitetta ongelma koskettaa, mutta sen

mahdollisuus on valitettavasti olemassa. Ongelma huomattiin niin myöhään, ettei tilannetta enää ehtinyt korjata.

Lisäksi ominaisuuksien suuren lukumäärän sekä niihin liittyvien monien erilaisten käyttäjätoiminnan vuoksi on mahdollista, että ohjelma saattaa testauksesta huolimattaa sisältää jonkin bugin. Projektia jatkettaessa voitaisiin testausta laajentaa merkittävästi ja varmistaa, ettei mitään bugeja varmasti olisi.

10. Parhaimmat ja heikoimmat kohdat

Yleisesti ohjelman parhaimpana puolena pidän sen monipuolisuutta ja suurta ominaisuuksien määrää. Kortteja, ja kaikkea muutakin, voi kustomoida todella monipuolisesti. Kortin mahdolliset värivalinnat, monipuoliset liitemahdollisuudet sekä hyvin toimivat tarkistuslistat ja aikaraja, ovat esimerkkejä kortin monista ominaisuuksista. Ominaisuudet, kuten arkisto ja mallipohjat, mahdollistavat ohjelman tehokkaan käytön ja helpottavat käyttäjän toimintaa. Käyttöliittymä kuvaa tämän kaiken selkeästi ja käyttökokemus on sujuva. Kortit näkyvät kaikkialla kokonaisuudessaan eli esimerkiksi arkistoa hallinnoidessa voi kortteja selata, kuin ne olisivat normaalissa listassa. Lyhyesti sanottuna ohjelmasta löytyy kaikki tärkeimmät ominaisuudet, mitä Kanban-lautoja tekevä käyttäjä voisi haluta.

Toisena tärkeimpänä kohtana pidän ohjelman yleistä intuitiivisuutta ja koen, että sen käyttö on selkeää ja helposti ymmärrettävissä. Ohjelma estää väärien asioiden tekemisen esimerkiksi laittamalla napin pois käytöstä, jos nappia ei kuulu painaa. Lisäksi ohjelma kertoo virheilmoituksilla, jos jokin menee vikaan tai tekstikentissä kertoo punaisella virhevärillä, jos teksti, jota käyttäjä syöttää ei ole sallittu. Näin tapahtuu esimerkiksi, jos yrittää luoda kahta samannimistä tunnistetta.

Yhtenä heikkona kohtana pidän edellisessä kohdassa mainittua mahdollista suorituskyvyn heikentymistä, mikäli elementtejä luodaan suuria määriä. Vikaa on siinä, kuinka muutosten yhteydessä koko näkymä päivittyy eikä vain jokin tietty osa, jota muutos koskee. Tämä olisi voitu korjata, jos aikaa olisi ollut enemmän tekemällä päivittämisestä tarkempaa.

Toinen heikkous on mielestäni drag-and-drop-ominaisuuden puute, jonka koen olevan ainakin jollain tasolla merkityksellinen tällaisille ohjelmille ja olisi ollut hienoa saada se toteutettua. Edellisessä kohdassa on asiaa enemmän avattu, mutta lyhyesti sanottuna käyttöliittymäkirjaston parempi osaaminen olisi saattanut auttaa ratkaisun löytämisessä tai jos ohjelman rakenne ei ollut niin toimiva tätä ominaisuutta ajatellen, olisi asian voinut korjata uudella rakenteella.

11. Poikkeamat suunnitelmasta, toteutunut työjärjestys ja aikataulu

Ensimmäisen kahden viikon aikana minulla oli suunniteltua enemmän aikaa käytössäni ja toteutin projektia paljon pidemmälle kuin suunnitelman mukaan oli tarkoitus. Tänä aikana toteutin käyttöliittymän perustan ja luokat pääkomponenteille, kuten oli suunnitelmassakin, mutta nopeammassa ajassa. Ymmärsin käyttöliittymän tekemisen nopeammin kuin olin ajatellut ja sain perustan toimimaan lyhyemmässä ajassa. Tämän lisäksi toteutin jo korttien muokkaamisen, siirtämisen ja tunnisteet sekä arkiston. Käyttöliittymää rakentaessani oli luontevaa toteuttaa korttien toimintaa suunniteltua enemmän ja ymmärsin aiempia asioita tehdessäni, miten ne voisi toteuttaa, minkä takia poikkesin suunnitelmasta.

Seuraavan kahden viikon aikana oli alun perin tarkoitus toteuttaa korttien siirtämisen mahdollistava raahaaminen ja pudottaminen. Tähän mennessä olin tajunnut, että ohjelman syvempi rakenne oli sellainen, että korttien raahaus-pudotus-ominaisuuden toteuttaminen ei enää ollut yksinkertaista. Tämän ja monien kiireiden takia päädyin poikkeamaan suunnitelmasta ja aloin keskittymään moniin muihin vaadittuihin ominaisuuksiin, jotta ne saataisiin varmasti toteutettua ja korttien siirtämisen oli kuitenkin mahdollista

klikkaamalla. Korttien siirto-omaisuuden toteuttamiseen kului siihen vähemmän aikaan, koska se tehtiin helpommalla tavalla.

Tämän toisen kahden viikon aikana oli suunnitelmana toteuttaa tiedostoon kirjoitus sekä lukeminen. Aikataulua suunnitellessani ei minulla ollut tiedossa, että tulisin käyttämään json-formaattia tiedostoissa ja tämän takia arvioihin siihen kuluvan ajan väärin. Suunnitelman esittelyssä sain kuulla circe-kirjastosta, joka mahdollisti luokkien muuttamisen json-muotoon huomattavasti helpommin kuin olin luullut. Oli toteutettava omat koodekit, mutta se oli nopeampaa kuin aiempi toimintatapa, joten tiedosto-osuuden toteuttamiseen ei kulunut niin paljoa aikaa kuin suunnitelmassa.

Tämän lisäksi toteutin kortteihin lisää ominaisuuksia, jotka oli tarkoitus toteuttaa vasta myöhemmin. Näitä olivat esimerkiksi aikaraja (deadline) sekä tarkistuslista. Lisäksi toteutin myös laudan taustakuvaominaisuuden. Suunnitelmasta poikettiin siis jälleen hieman. Aika-arviot niihin liittyen olivat kuitenkin suhteellisen tarkkoja.

Kolmannen kahden viikon aikana oli suunnitelmassa tarkoituksena tehdä lisää ohjelman ominaisuuksia. Monet näistä oli jo toteutettu aiemmin, joten tänä aika käytin aikaa useimpien viimeisen ominaisuuksien toteuttamiseen. Näistä merkittävin oli mallipohjaominaisuus sekä käyttöliittymän päivittäminen tukemaan kaikkia uusia ominaisuuksia. Lisäksi json-koodekkeja oli muokattava, jotta uusien ominaisuuksien vaatima tieto saatiin tallennettua. Olin siis hieman aikataulua edellä, mutta itse ominaisuuksiin liittyvät aika-arviot osuivat suhteellisen oikeaan.

Viimeiset kaksi viikkoa käytin viimeistelyyn, bugien korjaamiseen ja tämän dokumentin kirjoittamiseen, kuten suunnitelmassa olikin.

Yleisesti suunnitelma piti suhteellisen hyvin paikkaansa. Merkittävin suunnitelmasta poikkeaminen oli se, että monia ominaisuuksia toteutettiin suunniteltua aikaisemmin saatavilla olevan ajan vuoksi sekä etenkin korttien ominaisuuksien kohdalla oli luontevampi toteuttaa niitä ennen muita asioita, jotta kokonaisuuden muodostuminen olisi luontevampaa. Ajankäyttöön liittyvät arviot olivat jokseenkin päteviä, mutta esimerkiksi tiedostoon liittyvät aika-arviot olivat vääriä, ja niihin kului järkevämmän toteutustavan vuoksi merkittävästi vähemmän aikaa. Yleisesti olin aikataululla jonkin verran edellä.

Opin etenkin sitä, että suunnitelmassa voi olla hankala hahmottaa toteutusjärjestystä ennen kuin projektia lähtee oikeasti tekemään. Näin oli varsinkin, kun käyttöliittymäprojekti ei ollut kovinkaan tuttu aihe ja näin ollen siihen liittyvä epävarmuus teki suunnittelusta haastavaa. Vasta projektia tehdessäni ymmärsin, että jonkin asian toteuttaminen olisi paljon järkevämpää tehdä aiemmin kuin olin suunnitellut. Yleisesti aika-arvioiden tekeminen on haastavaa ja eteen tulee aina kaikkea yllättävää, joka joko lisää merkittävästi käytetyn ajan määrää tai jonkin asian saakin tehtyä huomattavasti helpommin ja siten nopeammin. Seuraavia projekteja varten minulla on paljon parempi käsitys, miten asioita kannattaa suunnitella.

12. Kokonaisarvio lopputuloksesta

Tehtävänä oli luoda ohjelma, jolla voi luoda ja käyttää Kanban-lautoja monipuolisilla ominaisuuksilla ja koen tässä onnistuneeni. Ohjelmassa on käytännössä kaikki vaaditut ominaisuudet sekä käyttökokemus on mielestäni sujuva. Intuitiivinen ja miellyttävä ohjelman käyttö sekä runsas määrä ominaisuuksia ovat omasta mielestäni ohjelman parhaita puolia. Ominaisuudet toimivat odotusten mukaisesti ja laudan käyttäminen on helppoa ja mahdollistaa monimutkaistenkin lautojen teon.

Suurimpana puutteena pidän drag-and-drop-ominaisuuden puuttumista. Sen tekeminen olisi luultavasti ollut mahdollista pidemmän ajan kanssa, mikä olisi mahdollistanut laajemman ScalaFX-ymmärryksen hankkimisen ja siten toimivan ratkaisun keksiminen olisi tapahtunut helpommin. Myös mahdollinen

syvällisempi rakenteellinen ongelma saattoi tämän estää, mikä olisi korjaantunut suuremmilla rakenteellisilla muutoksilla, kuten dokumentissa on aiemmin tarkemmin selitetty. Toinen heikkous on myös mahdollinen suorituskyvyn aleneminen komponenttien määrän kasvaessa merkittävästi. Tämäkin johtuu paljolti ajan puutteesta, sillä nykyisellä rakenteella komponenttien tarkemman päivityksen toteuttaminen olisi vaatinut enemmän ajan käyttöä. Näiden suhteen ohjelman ratkaisumenetelmiä olisi mahdollisesti voinut tehdä paremmin.

Tulevaisuudessa ohjelmaa voisi etenkin parantaa korjaamalla nämä kaksi aiemmin mainittua asiaa. Lisäksi ohjelmaa voisi tietysti parantaa lisäämällä ominaisuuksien määrää. Koenkin, että uusia ominaisuuksia on ohjelmaan mahdollista lisätä suhteellisen helposti. Esimerkiksi korttien ominaisuuksien tapauksissa on ominaisuuteen liittyvät tiedot lisättävä Card-luokkaan, sitten tätä käyttäen se laitetaan näkyviin käyttöliittymään ja lisäksi on päivitettävä json-koodekit tukemaan uutta ominaisuutta. En koe, että uusien ominaisuuksien lisäämisestä voisi tehdä merkittävästi yksinkertaisempaa. Ohjelman eri osat on myös erotettu toisistaan etenkin käyttöliittymän ja muun logiikan suhteen, mikä tukee muutosten tekoa. Rakenne on siltä osin selkeä, että on ymmärrettävissä mitä osaa on muokattava, jotta haluttu muutos saadaan tehtyä.

Yleisesti pidän käytettyjä ratkaisumenetelmiä, tietorakenteita ja luokkajakoa toimivana. Etenkin tietorakenteiden suhteen aiemmin selitetyn mukaisesti en koe, että asiaa olisi voinut tehdä merkittävästi paremmin. Ohjelman luokkajako on myös suoraviivainen ja looginen, joten en näe siinäkään suuresti parannettavaa.

Kokonaisuutena koen, että tein hyvää työtä ja lopputulos vastaa osaamistasoani. Tein projektia koko työajan ainakin jollain tasolla ja huolehdin aikataulussa pysymisestä. Vaikka kiireitä paljon olikin, oli vain itsestä kiinni, pysyikö aikataulussa ja varmistinkin, ettei merkittäviä asioita jäänyt viime tippaan. Etenkin alussa sain pidettyä hyvää tahtia päällä ja olin todella tyytyväinen panokseeni. Myöhemmillä viikoilla kiireiden määrä kasvoi ja en aina pystynyt käyttämään projektiin niin paljoa aikaa kuin olisin halunnut, mutta kokonaisuutena aikataulussa pysyminen onnistui.

Jos nyt aloittaisin projektin alusta uudelleen, käyttäisin enemmän aikaa syvemmän tason rakenteen suunnitteluun. Näin voitaisiin varmistaa, etten törmäisi niihin ongelmiin, joita kohtasin. Varmistaisin, että kaikki mahdolliset ominaisuudet voitaisiin toteuttaa ilman suurta vaivaa tavalla, jolla olin ohjelmaa tekemässä. Sain myös paljon uutta kokemusta etenkin käyttöliittymän suhteen ja osaisin nyt aiemmin tietää parhaita toteutustapoja ilman, että asioita pitäisi ensin yrittää tehdä huonommalla tavalla. Tietäisin nyt myös eri osien järkevämmän toteutusjärjestyksen, jossa eri osat tukisivat toisiaan.

13. Viitteet

ScalaFX, API/docs
https://www.scalafx.org/docs/home/ (Website, Getting Started)
https://www.scalafx.org/api/8.0/index.html (API)

JavaFX API
https://docs.oracle.com/javase/8/javafx/api/toc.htm

Circe Json Library
https://circe.github.io/circe/
https://circe.github.io/circe/codecs/custom-codecs.html (Custom Codecs)

Java Time Libraries (java.time, java.time.format, java.time.tempral APIs, Java Platform SE 8)
https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html

https://docs.oracle.com/javase/8/docs/api/java/time/format/package-summary.html
https://docs.oracle.com/javase/8/docs/api/java/time/temporal/package-summary.html

Other Java Libraries (java.net.URI, java.io.File, java.awt.Desktop APIs)
https://docs.oracle.com/javase/7/docs/api/java/net/URI.html
https://docs.oracle.com/javase/7/docs/api/java/io/File.html
https://docs.oracle.com/javase/7/docs/api/java/awt/Desktop.html

ScalaFX Tutorial Series – Mark Lewis
https://www.youtube.com/playlist?list=PLKUBmwYhjbwSxOjw5tBt8Lfit39tP1jyd

Inspiration/other
https://en.wikipedia.org/wiki/Kanban_board
https://trello.com/

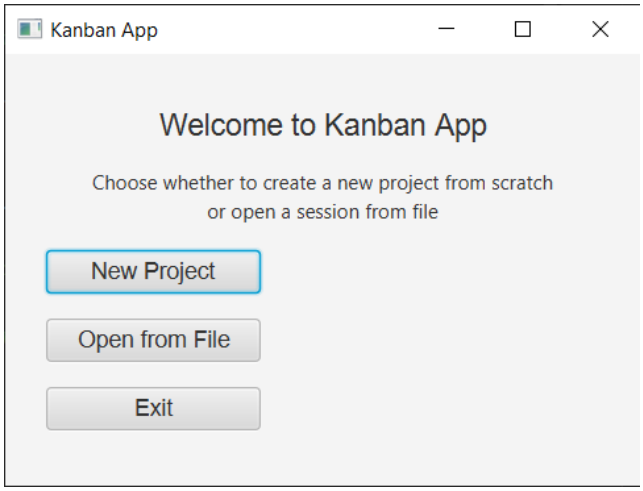
Projektin merkittävämpänä lähteenä toimi ScalaFX:n (ja joidenkin ominaisuuksien osalta JavaFX:n) API, jonka pohjalta sain tietoa, miten käyttöliittymäkomponentit toimivat ja miten niitä käytetään. ScalaFX:n nettisivujen Getting Started -osio opetti myös perusteista sekä etenkin Dialogi-ikkunoiden käyttöä. Mark Lewisin videotutoriaalisarja oli myös hyödyllinen erilaisten käyttöliittymäkomponenttien toiminnasta. Circe-kirjaston nettisivuilta löytyi hyvä sivu mmien json-koodekkien tekemisestä.

Käytin projektissa myös monia Javan kirjastostoja esimerkiksi aikaan ja päivään, tiedostoihin sekä laitteen hallintaan liittyen. Näiden osalta käytin kirjastojen omaa dokumentaatiota, josta sai hyvin tietoa niiden toiminnasta.

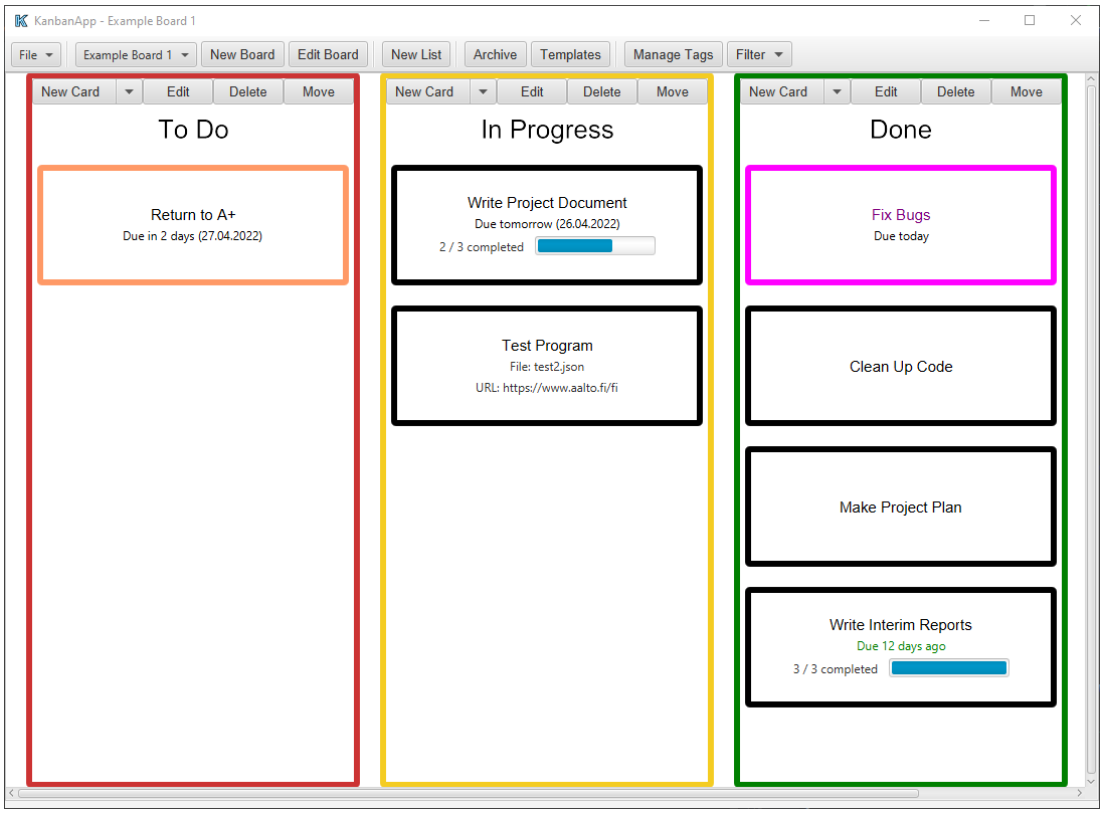
14. Liitteet

Projektin koko lähdekoodi on toimitettu tämän dokumentin mukana.

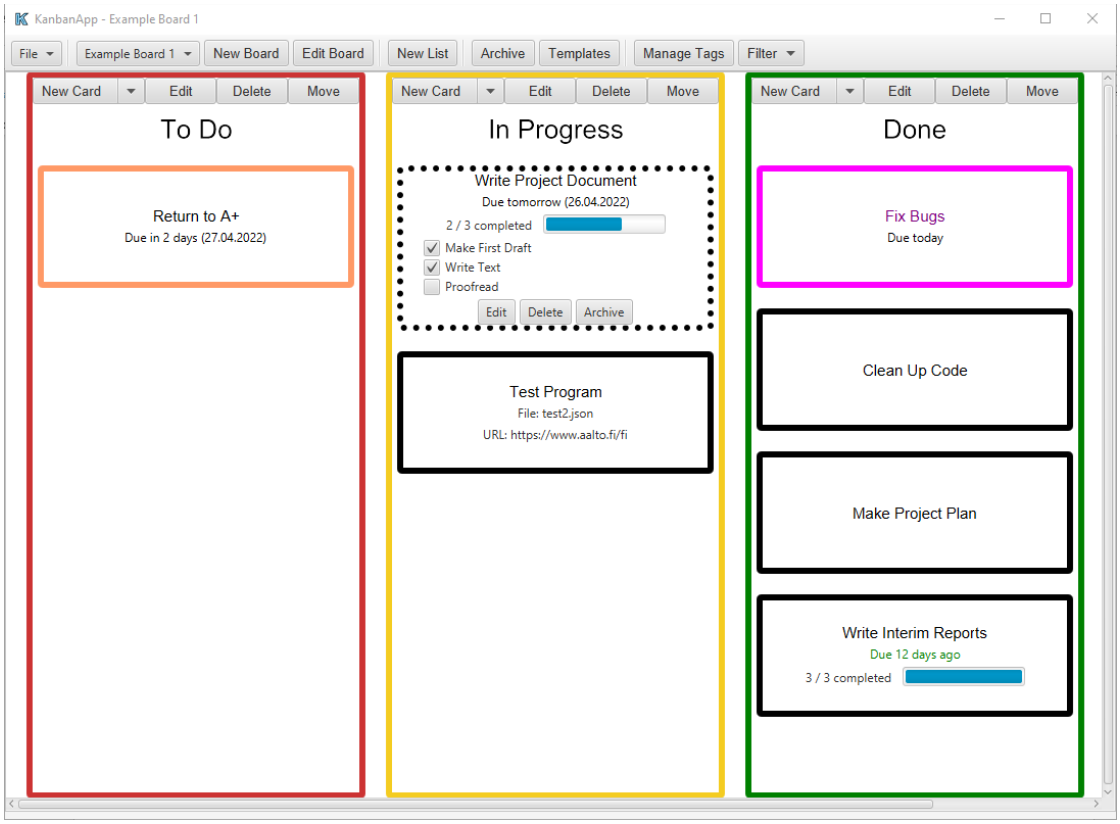
Ohessa vielä muutama ruutukaappaus ohjelmasta:



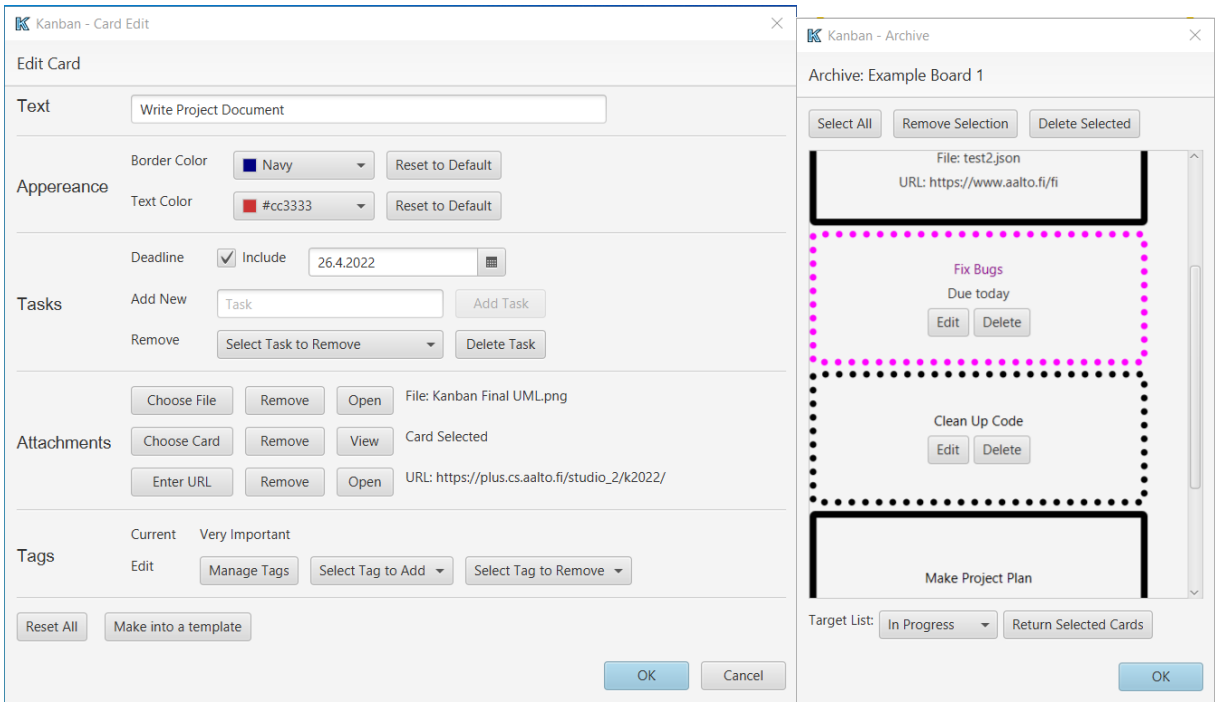
Tässä on kuvattuna ohjelman käynnistysikkuna. Siitä käyttäjä voi valita haluaako avata olemassa olevan projektin tiedosta, luoda kokonaan uuden vai poistua ohjelmasta.



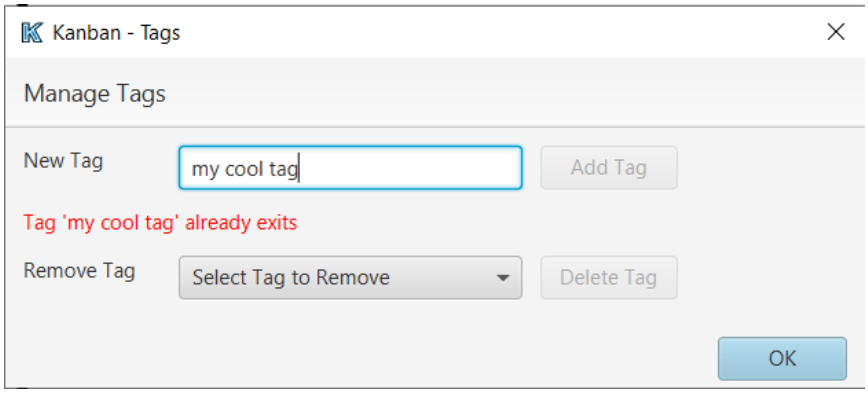
Tässä kuvassa voidaan nähdä ohjelman perusnäkymä. Siinä on luotu kortteja ja listoja käyttäen ohjelman erilaisia ominaisuuksia. Ruudun yläreunassa on nähtävissä ohjelman työkalupalkki, joka sisältää dokumentissa aiemmin kuvatut napit.



Tässä kuvassa nähdään, miten kortin näkymä laajenee, kun se klikataan aktiiviseksi.



Vasemmalla on nähtävissä kortin muokkausikkuna. Sen avulla kortin informaatiota voidaan muokata ja ottaa uusia ominaisuuksia käyttöön. Oikealla on kuva arkistosta. Siinä on listattuna arkistoon lisättyjä kortteja, joista kaksi on klikattu aktiiviseksi. Napilla ”Return Selected Cards” nämä kortit voidaan palauttaa laudalle valittuun listaan (kuvassa ”In Progress” -niminen lista).



Kuvassa on tunnisteiden hallintaikkuna. Nähtävissä on myös ikkunassa näkyvä virheilmoitus, kun käyttäjä yrittää tehdä jotakin väärää, tässä tapauksessa luoda tunnistetta, joka on jo olemassa.