Kirjasovelluksen

Kehittämisopas

Aku Laurila

akulau@student.uef.fi

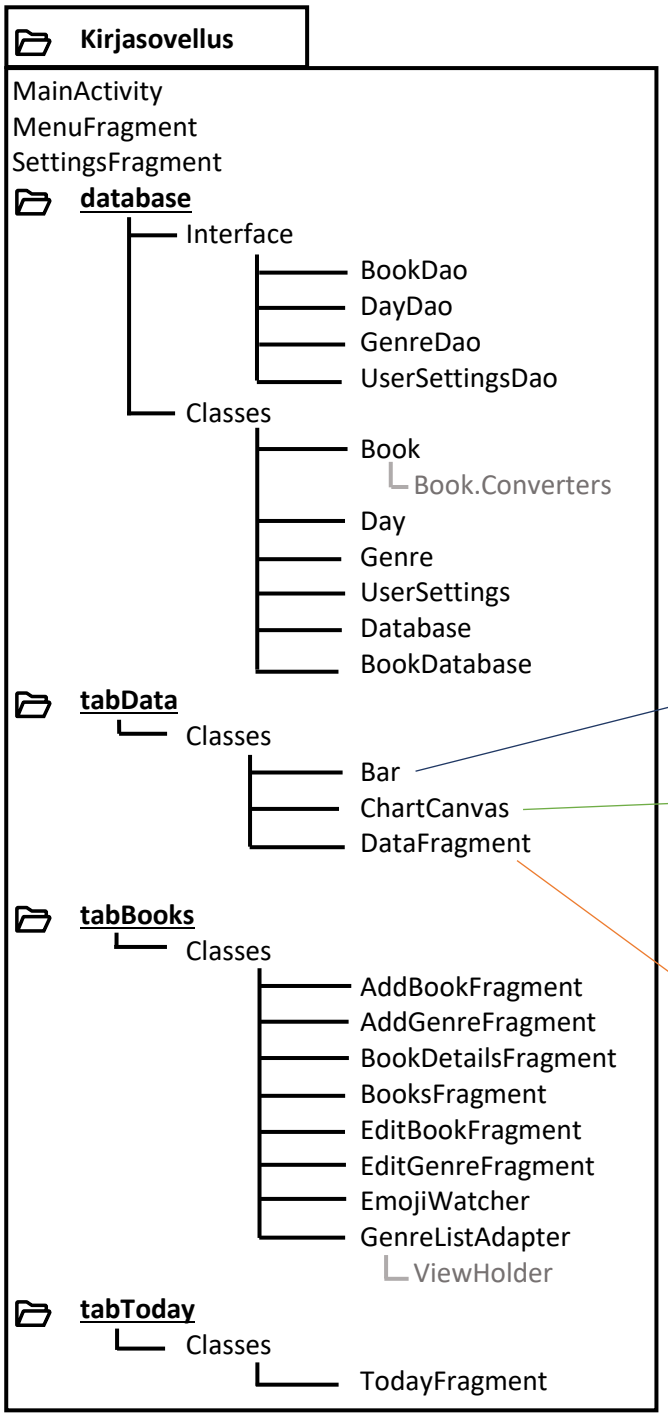
Sisällys

1. Sovelluksen osat 3
1.1 Jaottelu 3
1.2 Julkinen FragmentManager 4
1.3 Tietokanta 4
2. Kokonaisuuden rakenne ja osien kytkennät toisiinsa 5
2.1 Navigointi komponenttien välillä 5
2.2 Backstack 5
2.3 Tietokannan merkitys kokonaisuudessa 6
3. Ongelmia, haasteita ja ideoita 7
3.1 Tunnetut ongelmat 7
3.2 Kehittämishaasteet 7
3.3 Kehittämisideat 8

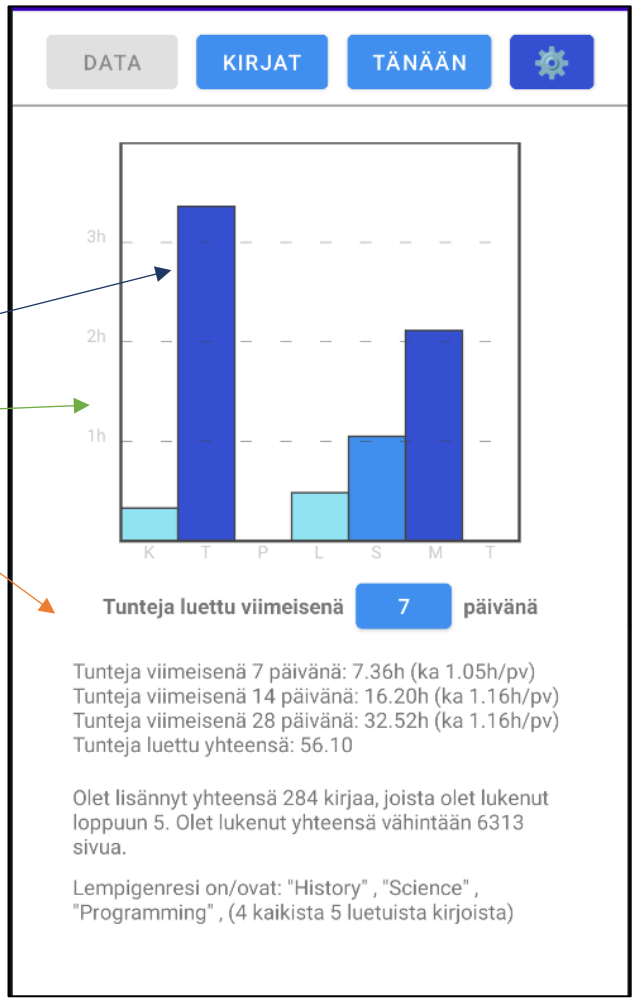
1. Sovelluksen osat

1.1 Jaottelu

Sovellus koostuu pääosin kahdestakymmenestäneljästä Java-luokasta ja kolmestatoista layout-tiedostosta. Näiden avulla sovellukseen on muodostettu yhteensä yhdeksän näkymää tai sivua (luokat, joiden nimet ovat muotoa `___Fragment.java`). Sovelluksessa on kolme (neljä, jos asetusnäkymä lasketaan mukaan) kappaletta ns. ”välilehtiä”, joihin sovelluksen toimintaa on jaettu kategorioittain. Välilehdet ovat ”Data”, ”Kirjat” ja ”Tänään”. välilehtien toiminta ja sisältö kuvastuu myös sovelluksen tiedostorakenteessa, jossa toiminnallisuuden toteuttavat Java-luokat on jaettu paketteihin välilehden mukaan.



Vasemmalla on puumainen esitys sovelluksen Java-luokkien tiedostorakenteesta. Alapuolella on näyttökuva sovelluksen käyttöliittymästä. Kuvista voidaan nähdä jaottelu, ja se, kuinka se helpottaa kokonaisuuden hahmottamista.



1.2 Julkinen FragmentManager

Sovelluksessa on vain yksi Activity. Activityn layout koostuu kahdesta päällekkäisestä `FragmentContainerView`:stä. Ylemmän `FragmentContainerView`:n tehtävänä on näyttää `MenuFragment`, jonka avulla sovelluksessa navigoidaan välilehtien ja asetussivun välillä. Alempi `FragmentContainerView` puolestaan näyttää aina relevantin näkymän sen mukaan, minkä syötteen käyttäjä on antanut.

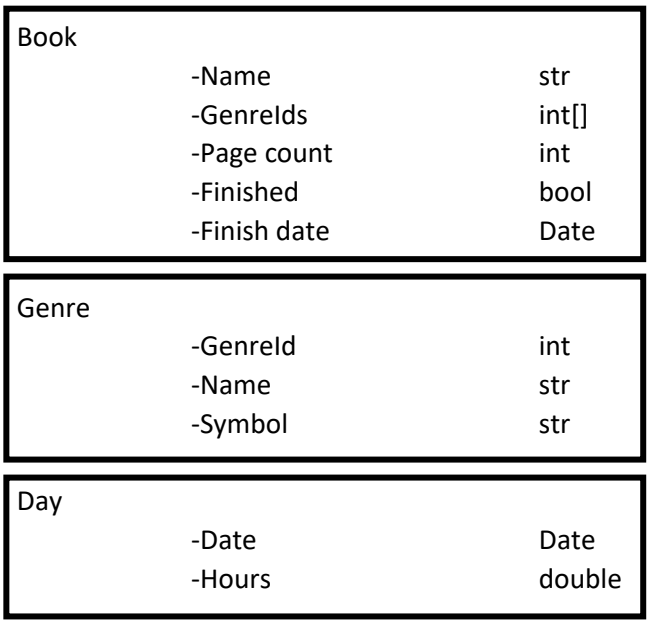
Sovelluksen käynnistyessä, `MainActivity`:n `onCreate()` -metodi alustaa sovelluksen käyttöön julkisen `FragmentManager`:in, jonka avulla vaihdetaan käyttöliittymän näkymää. `FragmentManager` ei koskaan korvaa yläreunan `MenuFragment`:iä toisella fragmentilla, vaan sisältö näytetään aina valikon alapuolella.

1.3 Tietokanta

`MainActivity`:n `onCreate()` -metodissa alustetaan myös `Room`-tietokanta sovelluksen datalle. Tietokanta koostuu neljästä entiteetistä: Kirja (`Book.java`), Genre (`Genre.java`), Päivä (`Day.java`) ja käyttäjän asetukset (`UserSettings.java`). Kirjoja, genrejä ja päiviä on mahdollista säilöä rajattomasti, mutta käyttäjän asetuksia on tarkoitus säilöä vain yksi kappale, id:llä 0.

Sovelluksen tietokantakutsut tehdään pääosin käyttöliittymäsäikeessä, mutta `BooksFragment.java` -luokan hakutoiminnallisuus on toteutettu asynkronisesti eri säikeessä. Tämä sen takia, että `BooksFragment`:in näkymä on todennäköisesti raskain ja useimmiten päivitettävä näkymä koko sovelluksessa. Tietokannassa on erikseen kyselyt taulukko- ja `LiveData` -muotoiselle palautukselle. `LiveData`:a voidaan tarkastella asynkronisesti `Observer`-rajapinnan avulla.

Testidatan generointiin ja tietokannan tyhjennykseen löytyy metodit `MainActivity`:stä (`generateTestData()` ja `nukeAllData()`). Sieltä löytyy myös metodi (`initializeDatabase()`), joka lisää tietokantaan viimeisen 28 päivän tyhjille päiville 0 tuntia. Tällöin data-välilehden pylväskaavio näyttää tyhjät tai ”puuttuvat” päivät oikein.

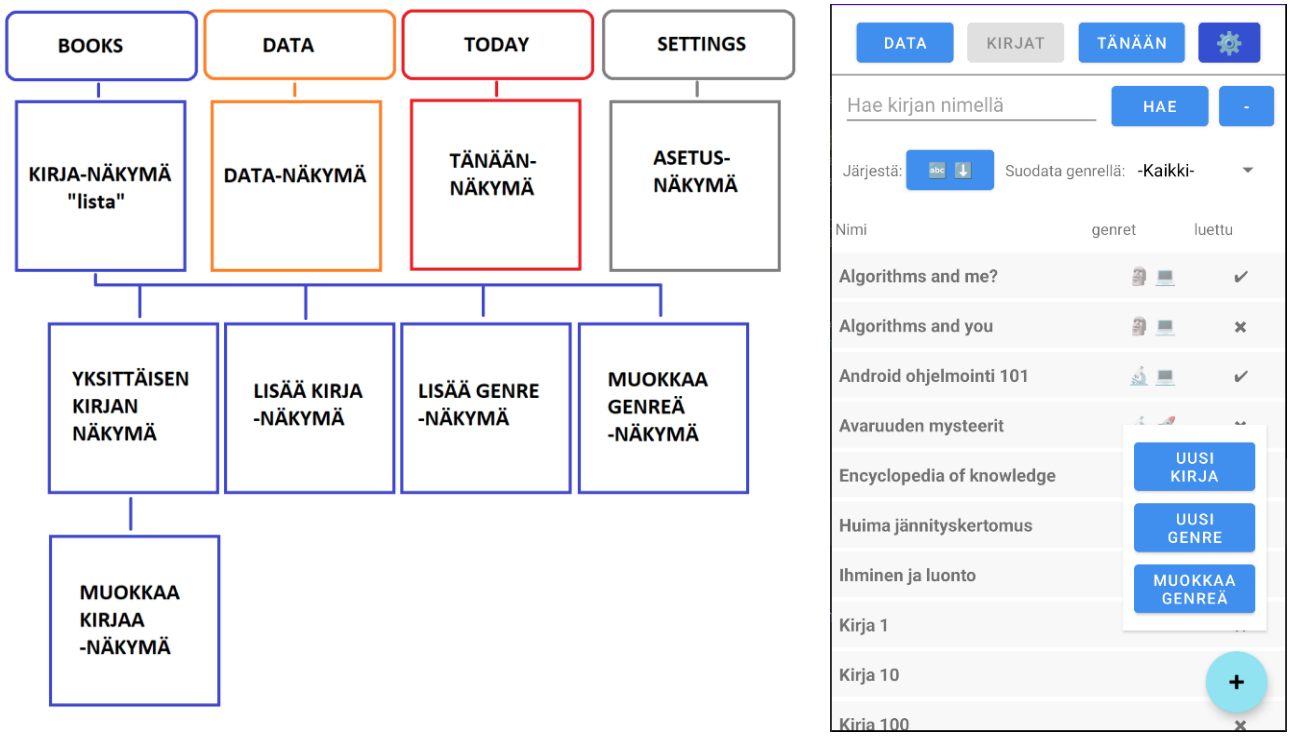


Tietokannan Entiteettien rakenne yksinkertaistettuna.

2. Kokonaisuuden rakenne ja osien kytkennät toisiinsa

2.1 Navigointi komponenttien välillä

Alla olevasta kaaviosta voidaan nähdä käyttöliittymäkomponenttien väliset suhteet. Kaavion ylimmällä rivillä olevat välilehdet ovat näkyvissä jokaisessa näkymässä. Niihin voidaan navigoida siis mistä tahansa. Kaavion oikealla puolella on näyttökaappaus ”Kirjat”-välilehdestä lisävalikot avattuina.



Kuten voidaan huomata, sovelluksen ominaisuuksiltaan laajin osa on ”Kirjat”-välilehti, jossa kirjalista-näkymän (`BooksFragment.java`) kautta voidaan siirtyä tarkastelemaan yksittäistä kirjaa, lisäämään uutta kirjaa, lisäämään uutta genreä tai muokkaamaan olemassa olevaa genreä. Tämän lisäksi käyttäjä voi tietenkin navigoida päävalikosta toiselle välilehdelle. Jotta näkymästä ei tulisi käyttäjälle liian sekavaa, on osa hakuasetuksista asetettu erillisen lisävalikon alle. Kirjan ja genren lisäys, sekä genren muokkaus on myös laitettu näytön oikeassa alareunassa sijaitsevan `FloatingActionButton`:in takaa löytyvään lisävalikkoon.

2.2 Backstack

Käyttäjän siirtyessä näkymästä toiseen, pitää `FragmentManager` yllä tietoa siitä, missä käyttäjä on liikkunut. Fragmentin vaihtuessa, lisätään se backstack:iin, jotta sinne voidaan palata esim. Androidin takaisin -napilla. Kirjasovelluksen `FragmentManager` pitää yllä backstack:iä, mutta tyhjentää sen kieltä vaihdettaessa kokonaan. Tällöin käyttäjä ei voi vahingossa palata ”väärän kieliseen” näkymään. Sovelluksessa palataan joskus myös automaattisesti takaisin backstack:issä. Esimerkiksi tallennettuaan kirjan muokkaukset, palaa sovellus automaattisesti muokatun kirjan yksilönäkymään.

2.3 Tietokannan merkitys kokonaisuudessa

Sovelluksensisäinen `Room` -tietokanta on luultavimmin koko sovelluksen keskeisin ja käytetyin komponentti. Jokainen sovelluksen välilehti, sivu tai näkymä joko lukee tai tallentaa tietoa tietokannasta/tietokantaan jollain tavalla. ”Data” -välilehti lukee ja analysoi tietokannasta haettua dataa. Se muodostaa datasta johtopäätöksiä ja muotoilee ne käyttäjälle luettavaan muotoon. ”Kirjat” -välilehdellä niin luetaan- kuin talletetaankin tietoa monessa muodossa. Dataa esiintyy sekä kirjoina, että genreinä, joskus molempina samanaikaisesti. ”Tänään” – ja ”Asetukset” -sivuilla dataa puolestaan pääsääntöisesti talletetaan.

Tietokannan ollessa näin suuressa osassa sovelluksen toimintaa, tulee kokonaisuuden olla selkeä ja yhtenäinen. Toiminnallisuutta on pyritty jakamaan eri näkymiin myös osaksi sen takia, että jokaisella näkymällä olisi selkeä rooli tietokannan suhteen. Esimerkiksi vaikkapa ”Tänään” -välilehden toiminnallisuus rajoittuu Tietokannan `Day` -taulun hallintaan nykyiseltä päivältä, ”Lisää Genre”-sivulla hallinnoidaan ainoastaan `Genre` -taulua, jne. ”Data” -välilehti on ainoa näkymä, johon dataa haetaan ja näytetään kaikista (paitsi `UserSettings`) tietokannan tauluista samanaikaisesti.

3. Ongelmia, haasteita ja ideoita

3.1 Tunnetut ongelmat

Suurin osa tietokantakutsuista tehdään käyttöliittymäsäikeessä. Ainoat asynkroniset kutsut tietokantaan tehdään BooksFragment:in search() -metodissa, sillä kyseisen näkymän haut ovat raskaammasta päästä. Sovelluksen toimintaa ja käyttöliittymää testaillessa ei tullut vastaan tilanteita, joissa käyttöliittymä olisi jäätynyt liian pitkäksi aikaa. Riskinä on kuitenkin, että tietokantakutsuissa voi mennä jotain pieleen ja sovellus jumittuu.

BooksFragment:in ”Suodata genrellä” -alasvetovalikko tunnistaa genret toisistaan nimen – eikä genreId:n perusteella. Tämä johtaa siihen, että kaksi samannimistä genreä tietokannassa hajottaa viimeisimmäksi lisätyllä rajaamisen. Ongelman voisi korjata ainakin kahdella tavalla: Joko rajoittaa käyttäjää niin, ettei tietokannassa voi olla useampaa kuin yhtä genreä samalla nimellä. Vaihtoehtoisesti rajauksen voisi muuttaa toimimaan genren genreId:llä nimen sijaan.

Joskus ilmenee ongelmia FragmentManager:in backstack:in toiminnassa. En ole aivan varma, mistä ongelmat johtuvat, mutta joskus takaisin -nappi ei toimi odotetusti. joskus tulee myös virhe, jonka mukaan FragmentManager:ia ei ole ”kiinnitetty”. Molempia ongelmia on kuitenkin vaikea toistaa tarkoituksella, enkä ole löytänyt ratkaisua. Ongelmat ovat onneksi olleet harvinaisia.

3.2 Kehittämishaasteet

Haasteita ilmeni eniten paikoissa, joissa niitä odotin olevan vähiten. Alla muutamia mieleen painautuneimpia haasteita.

BooksFragment:in `RecyclerView`-näkymä oli TODELLA hidas ja jumitti käyttöliittymää, kunnes tajusin ongelman johtuvan siitä, että se oli `ScrollView`-komponentin lapsena. Ilmeisesti tämä saa `RecyclerView`:in kutsumaan `onBindViewHolder()` -metodia tarpeettoman monta kertaa.

Ongelmia lokalisaation ja näytön kääntämisen kanssa. Jostain syystä en meinannut saada lokalisaatiota pysymään oikeana näyttöä käännettäessä. Kun sain sen vihdoin toimimaan, niin sovelluksen näkymä tahtoi aina nollautua takaisin alkunäkymään näyttöä käännettäessä. Tämän koitin korjata ottamalla talteen käyttäjän näkymän sen avautuessa ja näytön kääntyessä siirtymällä uudelleen talletettuun näkymään.

Tämä on ehkä enemmän dokumentaatioon liittyvä haaste, mutta JavaDoc:in generointi Android studiossa aiheutti päänvaivaa paljon enemmän, kuin olisin voinut odottaa. Aluksi JavaDoc ei löytänyt androidin JAR-paketin sisältöä, joten suurin osa importeista ei toiminut. Tämän korjaaminen oli ”helppoa”, sillä siihen riitti JAR-tiedoston importtaus `build.gradle`:ssa. Suurempi ongelma oli se, että kaikista yrityksistäni huolimatta, ei JavaDoc ymmärtänyt Androidin `Resource`-luokkaa (`import kirjasovellus.R`), jonka sain viikon kikkailun jälkeen kierrettyä niinkin helposti, kuin vaihtamalla importit muotoon ” `import kirjasovellus.*` ”.

Emojien erotteleminen merkkijonosta on vaikeampaa kuin luulin. Syynä tähän on, että yksi emoji voi olla ohjelmakoodissa jopa 3 tai 4 merkkiä pitkä, eivätkä käyttäjän syöttämien emojien pituuksia voi ennustaa. Keksin kuitenkin, että käyttäjän viimeisimmäksi syötetyn emojin voi teoriassa selvittää ottamalla merkkijonoista `uusi` ja `vanha` `uusi.subString(vanha.length, uusi.length)`, jolloin jäljelle jää vain merkit, jotka kuuluvat uuteen merkkijonoon (eli käytännössä emojiin). Varsinaisessa emojien tarkistuksessa auttoi Cory Kilger:n EmojiDetector.java: https://gist.github.com/cmkilger/b8f7dba3e76244a84e7e

3.3 Kehittämisideat

Alla on listattuna muutamia kehittämisideoita sovellukselle:

- Kaikki tietokantakutsut pois käyttöliittymäsäikeeltä.

- Parempi tapa syöttää genrelle emoji. Ehkä emoji -näppäimistö?

- Yleinen suorituskyky ja tehokkuus paremmaksi

- Monipuolisempia johtopäätöksiä datasta

- Enemmän käyttäjäasetuksia, kuten teema