Kanban App

Kimi Kuru 790695

Computer Science

2. Year

26.4.2022

General description

The app I built is a “Trello-like” Kanban application where users can create Kanban boards, add lists to the boards and add cards to the lists. The cards contain text and possible tags and time monitoring which shows elapsed hours from the card creation time. User can add as many boards, lists and cards as is needed. The cards can be: archived and brought back from archive, saved and loaded from files, deleted, and filtered based on tags. Lists and cards are drag-and-droppable.

The work was completed in intermediate difficulty.

User Interface

The app can be started from the file “kanbanApp.scala ” which is located in the “src\main\scala” folder along with the other packages.

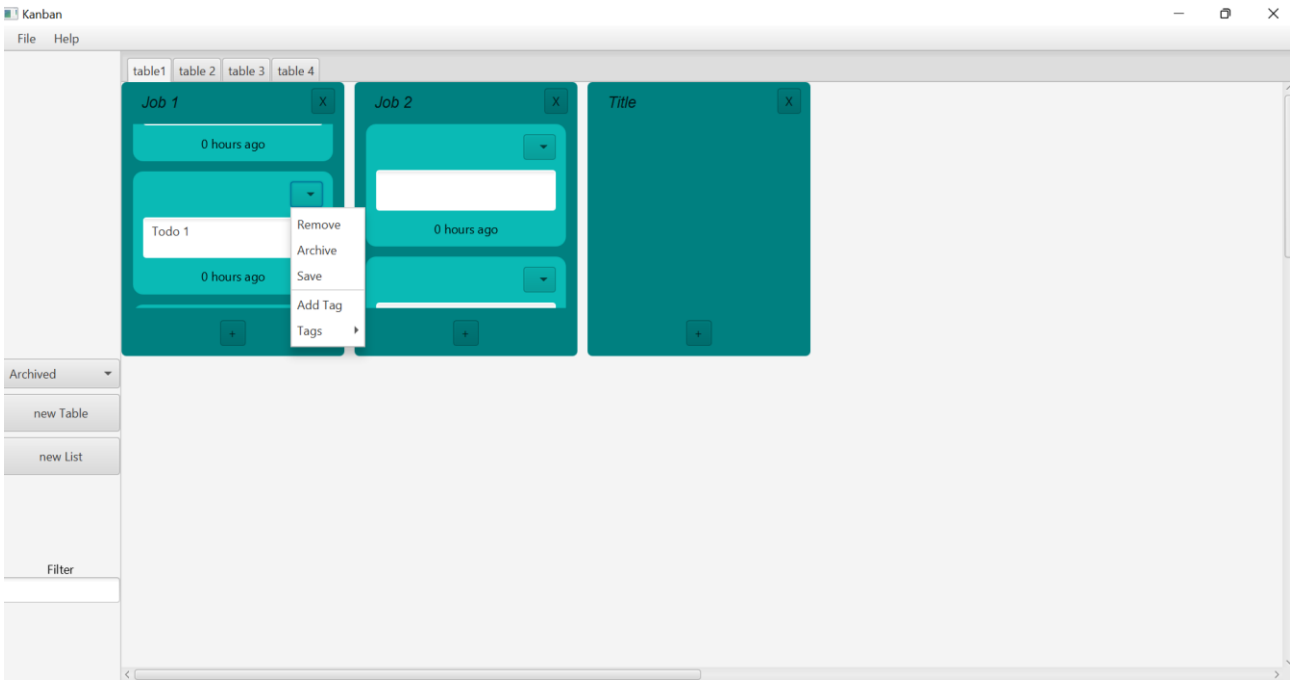


Figure 1. User interface.

Figure 1 shows the user interface. Through it we can add/remove tables, add/remove lists to active table (current tab), load cards from File -menu, filter cards by adding text to filter field, add/remove cards, archive and save cards through the cards dropdown menu, add tags through the cards dropdown menu, add text to cards and list title and also drag and drop lists and cards to other locations.

Program structure

The program structure is based on a Model-View-Service-Controller design pattern where models hold the data, service exposes the models and the business logic to controllers, controllers manipulate and receive the model data and send it to the views, and the views show the data to the user and expose functionality such as buttons. Model and service can be thought as the “backend” and view and controller the “frontend” or UI. This design was selected because JavaFx framework supports creating the UI with FXML and controllers. Also it is a good general pattern so it can help with reusability. The main point of creating a Service in addition to just MVC is to help separate the important business logic.

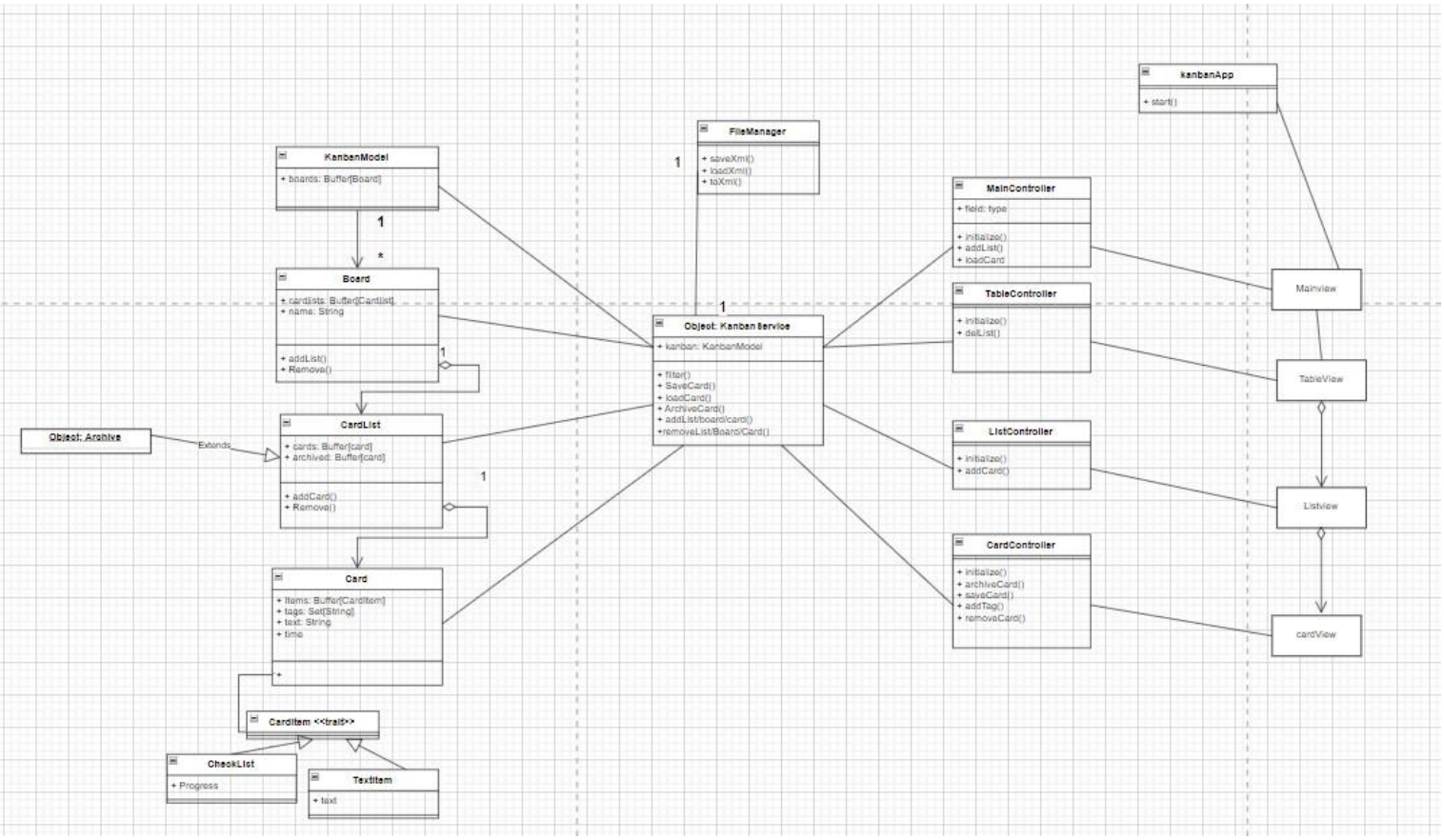


Figure 2 UML

For models we have classes KanbanModel, which holds all the Boards and can add or remove them; Board class which holds all the Lists and can add or remove them; CardList class which has all the cards and can add or remove them; Card class which holds the card text, tags and time; and also Archive which is an object of CardList class and used to store the archived cards.

For the service part we have an object KanbanService which implements the main methods used like: filter, which filters cards by tags (strings); saveCard method, which

saves the given card to path: loadCard for loading card from xml, ArchiveCard for archiving cards; and add/remove methods for creating boards/lists/cards. FileManager has methods for saving and loading cards to/from XML and converting a card object to XML.

The UI controllers all have initialize method, in which we add necessary starting values to the views and bind the models to the views (with listeners and such). All controllers also have methods which are invoked through e.g. by pressing a button from the view. MainController has methods to add Lists and tables. TableController (Board) can delete lists. ListController can add cards. CardController can archive, save and remove the card and add tags to the card.

The views are written in FXML to represent the UI of the app. The kanbanApp class starts the whole app by loading the MainView.

Algorithms

One main functionality of the program is saving and loading cards to XML files. This requires converting card model to XML and back. To turn a card to XML, we simply create necessary nodes for the XML and and add the convert the cards information there. For reading an XML we search the required nodes like text and time, and create a new card model out of those.

We also need to filter cards by tags. To filter, we go through the Kanbanmodels every board, and then every list and all of their cards, check whether the cards contain tags (strings) which contain the query string and return those cards.

In the UI side I wanted to create the Trello-like board, where the lists are added one after another. When we have multiple lists, and one list is removed somewhere from the middle, the cards collapse back, so that they are next to each other again (doesn’t leave a gap). This would seem easy enough, but as we used a UI component that laid out the lists according to coordinates rather than just by a list of children, this needed an algorithm to work out how to render these cards when modified. So, when an item is added to the board we calculate the next coordinates by a fixed number of columns (set to 10) and the amount of current items in the board. When an item is removed, we get the coordinate of the removed item, then go through the next coordinates after the removed one and at each coordinate we set the items one coordinate back to collapse the lists, and finally remove the last coordinate item. This seemed to be more efficient solution than just rendering the whole new list of items again or going through the whole list. We also needed to handle situations like changing the positions of the lists, so when an update event happened we made an algorithm to compare the updated model list and the UI side list items and switch places of the items that differed in those lists.

Data structures

For mutable data that needed to be mutated like the collections of boards, cardlists, cards and I used mutable buffer (scalafx observablebuffer), because the collections needed to be modified during runtime and I was familiar with it from the course. We also used a mutable set for the card tags as duplicates don't matter there. We used a mutable map to store the coordinates and javafx nodes in the board, because searching the nodes with coordinates with javafx methods wasn't particularly easy or efficient. We could have of course used immutable collections, but that would have required a bit different style to the programming and the app.

Files and internet access

The program reads and writes custom XML files presented in Figure 3.

Figure 3.

The XML must contain the cards text and the time it was created. Optionally one can add tags and card items. The program created XML also adds an id which is the card model that was saved. (the card items are not shown as I only implemented them to the backend at the start but didn't manage to create them to the UI because of lack of time).

Testing

The testing process was quite same as planned, with the exception that I didn’t make any actual testcases during development, just testing through the UI. Through UI I tested that all the required functionality worked and with different input. Also tested that the UI worked smoothly and didn’t slow down noticeably.

I created unit tests for all the methods of KanbanService. The models were not tested separately as their methods are directly used in the KanbanService methods. I tested that the collections of boards/lists/cards turn out right when adding and removing different kinds of inputs Also, methods for archiving were tested .The unit tests also included testing that the Services load and save methods worked correctly and that the filter method returned right results.

I also separately tested the Filemanager with couple test cases for the load and save methods and checked that with valid files it worked and that invalid files threw correct exceptions.

Known bugs and missing features

I haven’t noticed any bugs yet. I tried to make this project with difficulty as hard but didn’t have the time so some features from the hard difficulty level are not in the UI. All features of intermediate level are working.

3 best sides and 3 weaknesses

+ The overall structure of the model is good

+ All of the the requirements are filled and work nicely

+ The UI is pretty good and ok to use

-The code itself is partly messy especially in the initialize methods of controllers. This could be improved by e.g., adding this code as functions to Utils and importing from there.

- The program doesn’t in many cases take into consideration the possible problem situations like the max length of some strings in the UI etc.

- The code isn’t commented well and some of the variables are badly named

Deviations from plan, realized process and schedule

The project was created by firstly creating the backend functionality and then the frontend as planned. But much of the work was done during the last weeks as I ran into some issues with time and didn't have much time to do the project. Also, some parts were completely left out as I had to switch to intermediate level instead of the planned challenging. My time estimation was definitely too optimistic.

Final evaluation

The overall structure of the project is good and all the functionality is working nicely. I think the class structure is pretty good. However, some of the code is messy, badly documented and named. I also noticed during development that, KanbanService also exposes the model functionality somewhat badly. Also, there are some small deviations from the real MVSC design pattern like having connections between some of the controllers in a way which is not good practice. These could be made better simply by refactoring some of the code to simpler parts, adding comments and removing the hacky solutions with proper ones. The structure of the program is good for making extensions or changes especially if we were to implement something web based.

If I started over I would think of the solutions more beforehand and read more about e.g., the javafx framework before starting its implementation, as I noticed at some points I had done some things in a long and difficult way even though the framework already had a solution for it. The extra spent time, also ate up motivation for cleaning the code. I would also try to create better methods for the KanbanService, so that it would be easier to use.

References

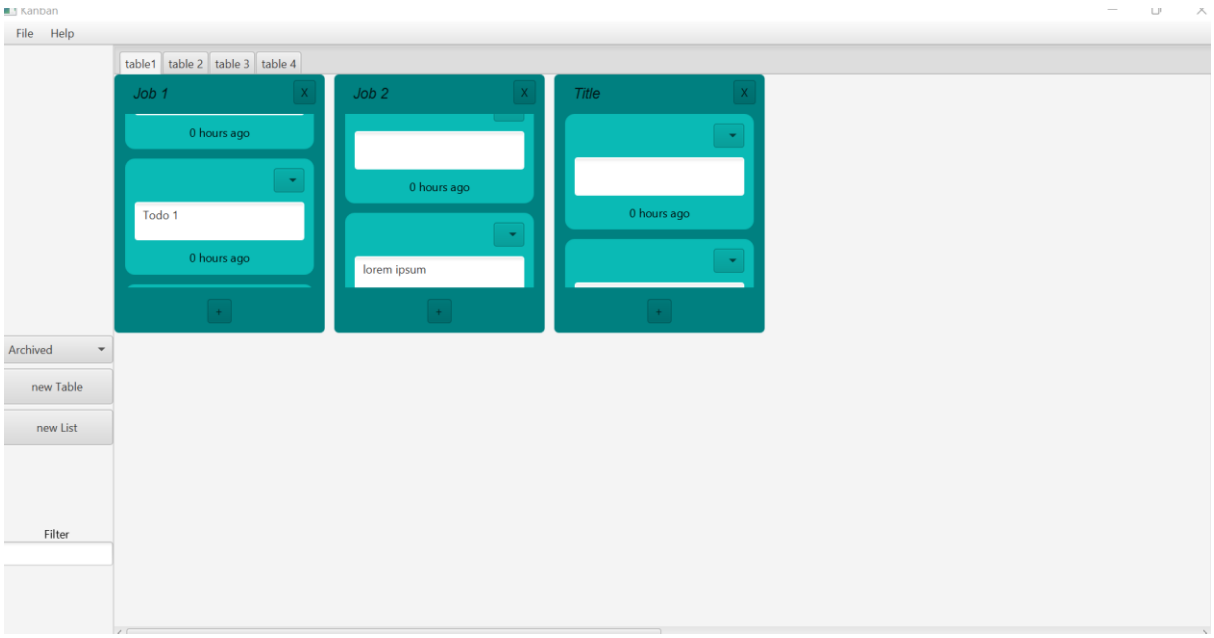
https://docs.oracle.com/javase/8/javafx/api/toc.htm

https://docs.scala-lang.org/

https://www.scalafx.org/

https://github.com/nscala-time/nscala-time

Appendix


Kanban
File
Help
table1
table 2
table 3
table 4
Job 1
0 hours ago
Todo 1
0 hours ago
Job 2
0 hours ago
lorem ipsum
Title
0 hours ago
Archived
new Table
new List
Filter

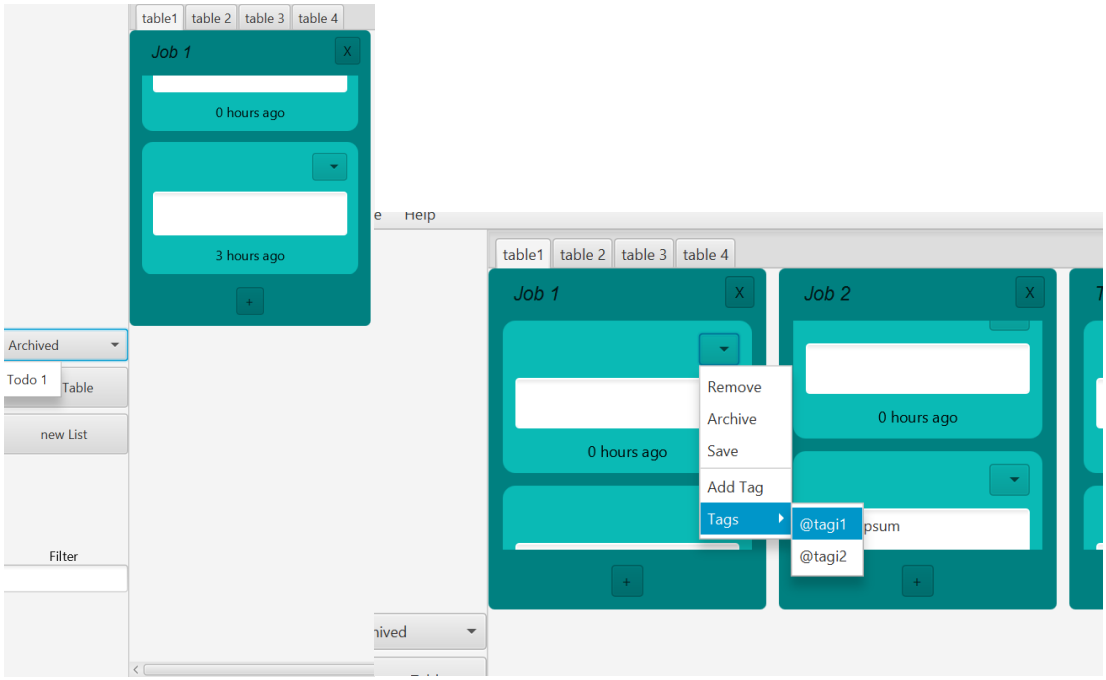


table1
table 2
table 3
table 4
Job 1
0 hours ago
3 hours ago
Archived
Todo 1
Table
new List
Filter
Help
table1
table 2
table 3
table 4
Job 1
0 hours ago
Remove
Archive
Save
Add Tag
Tags
@tagi1
@tagi2
Job 2
0 hours ago

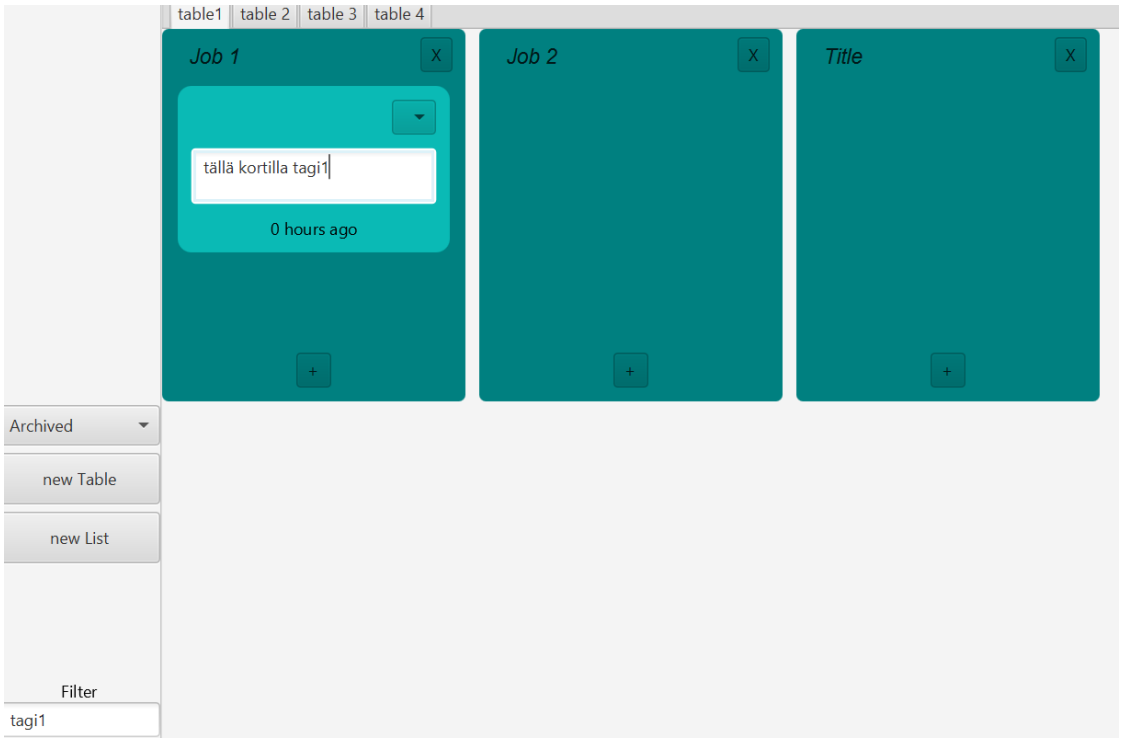
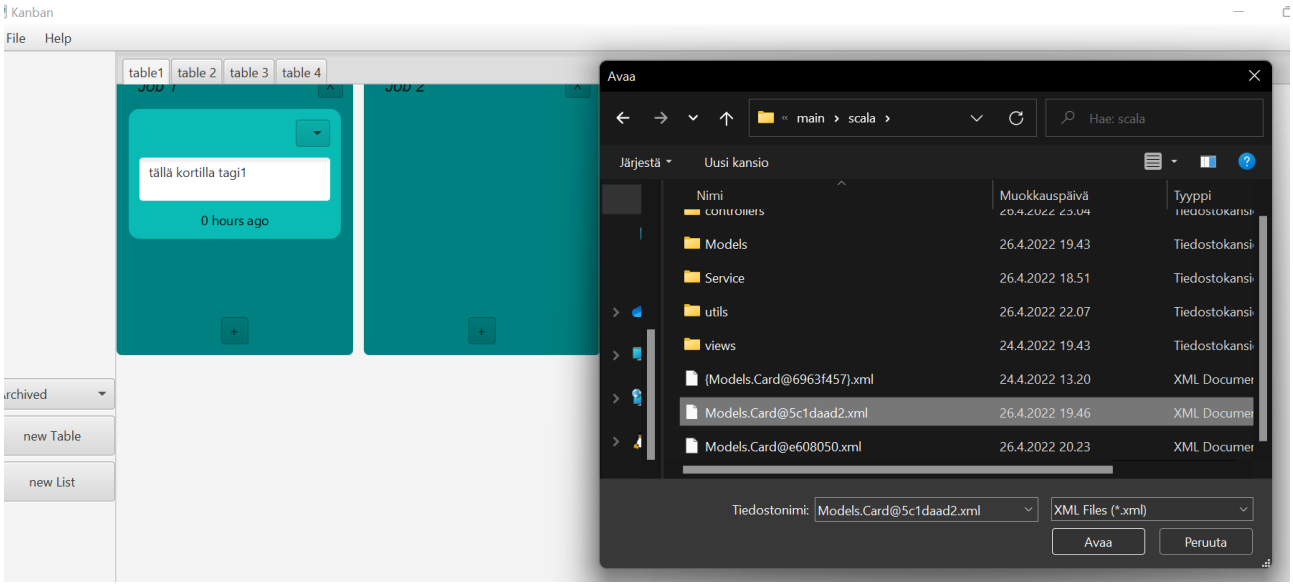


table1
table 2
table 3
table 4
Job 1
X
tällä kortilla tagi1
0 hours ago
Job 2
X
Title
X
Archived
new Table
new List
Filter
tagi1


Kanban
File
Help
table1
table 2
table 3
table 4
tällä kortilla tagi1
0 hours ago
new Table
new List
Avaa
main
scala
Hae: scala
Järjestä
Uusi kansio
Nimi
Muokkauspäivä
Tyyppi
Models
26.4.2022 19.43
Service
26.4.2022 18.51
utils
26.4.2022 22.07
views
24.4.2022 19.43
{Models.Card@6963f457}.xml
24.4.2022 13.20
Models.Card@5c1daad2.xml
26.4.2022 19.46
Models.Card@e608050.xml
26.4.2022 20.23
Tiedostonimi:
Models.Card@5c1daad2.xml
XML Files (*.xml)
Avaa
Peruuta