PROJECT DOCUMENT: KANBAN APP

Personal information

Title: Kanban Board

Student's name: Linh Ngo

Student number: 906502

Degree program: Aalto Bachelor's Programme in Science and Technology - Data Science

Year of studies: First year

Date: 28/04/2021

1. General description

The goal of the project is to create a Kanban board with a graphical user interface that allows users to create and use Kanban App, which helps them in managing their personal tasks and controlling the flow of their tasks.

The Kanban board has the basic structure of three columns representing the status of the task mentioned in each card: to-do, in progress, done.

Some of the function that users can do with the Kanban board is:

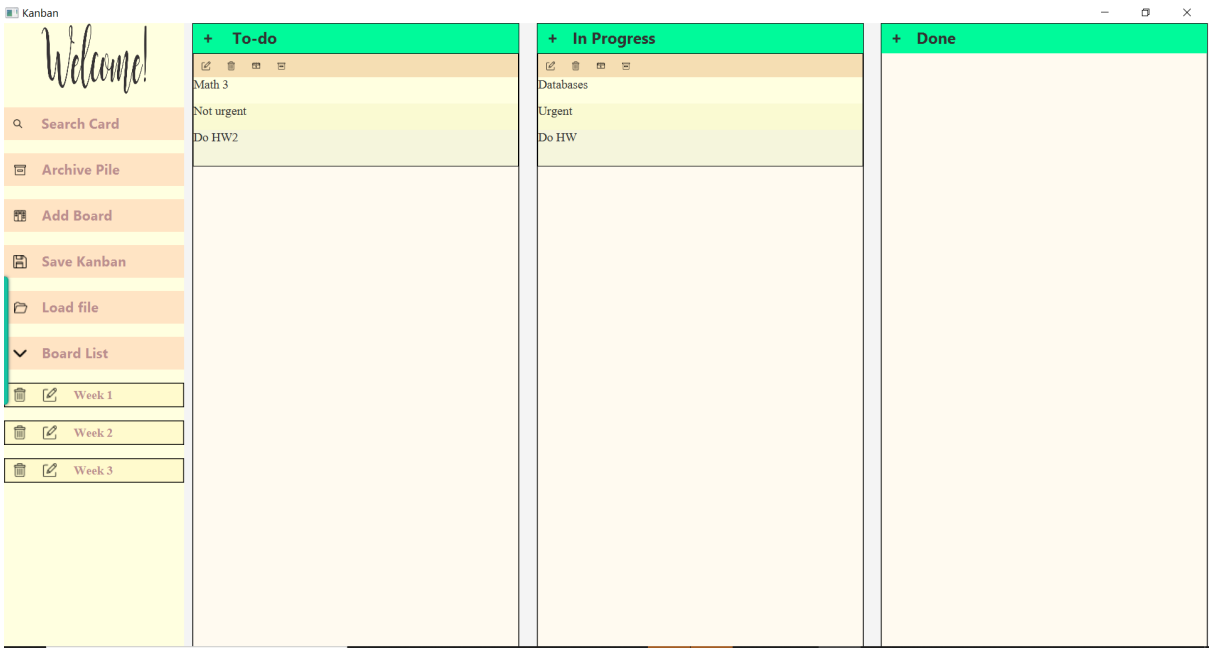
- Drag-and-drop cards between 3 columns in the graphical user interface.
- Add and remove cards to/from the board.
- Archive and return cards from an archive pile.

Each card includes main information such as card name, card tag which can be filtered based on tags, and text content. Each card also has a time tracking function in which a user can set a deadline for the task, and the card will show the number of days left to the deadline from the local date.

Card data is saved in and restored from external files. (human-readable format)

The program supports multiple boards for a user.

2. User interface



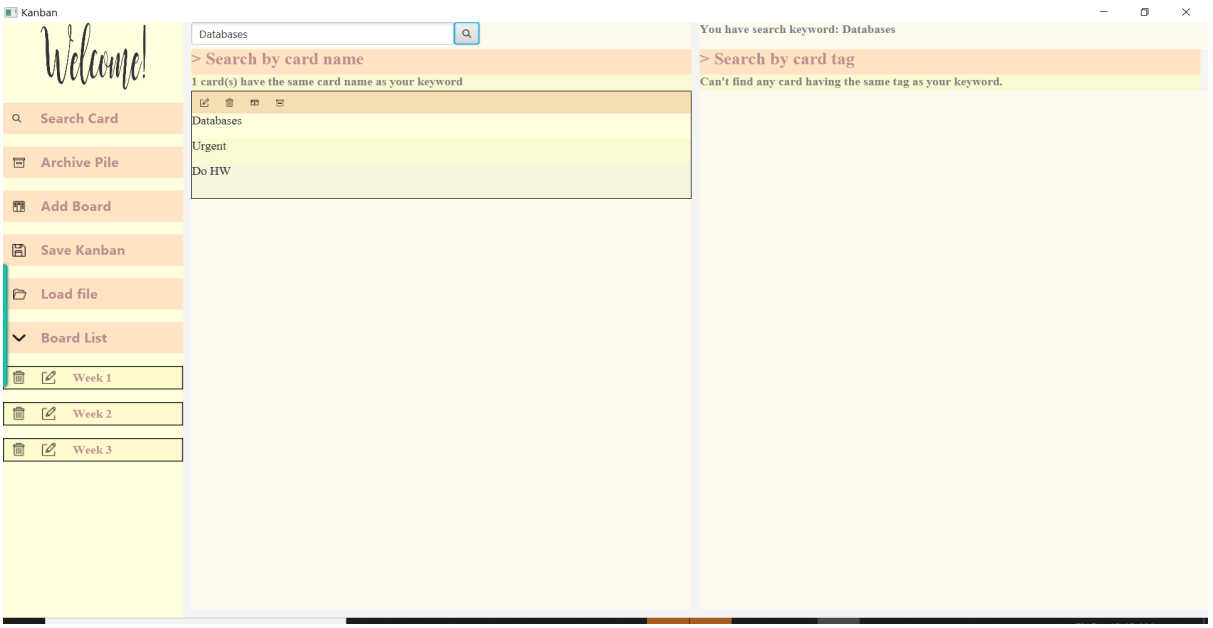
UI is separated into two different parts, with the Side Menu on the left and Main Window on the right.

The ***side menu*** is responsible for presenting all buttons which users can click on to perform certain actions and access the most basic and frequently used functionalities such as searching/filtering cards, getting access to the archive pile, adding boards to Kanban, saving Kanban to file, and loading recent file to Kanban.

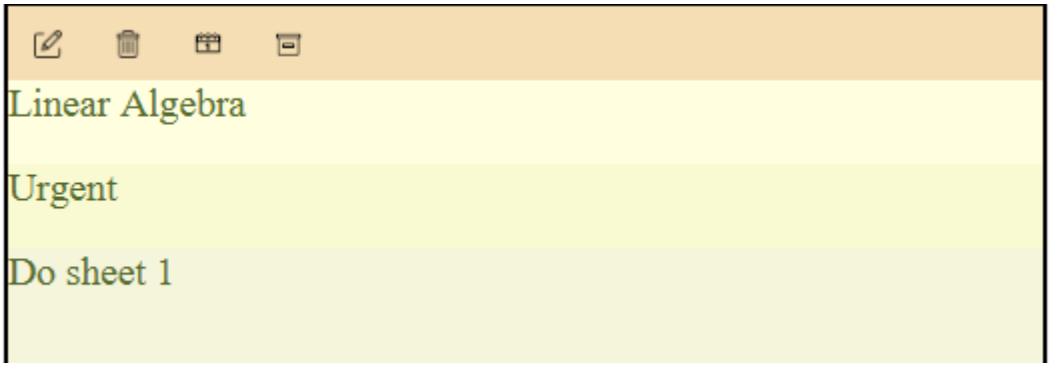
Below the Board List box are board boxes that represent boards. A user can click on the board button and the board will be shown on the right side.

The Main Root on the right is responsible for displaying different views in the program, including the search view, board views, and archive pile view. The view is shown when a user clicks on the button the related to that view.

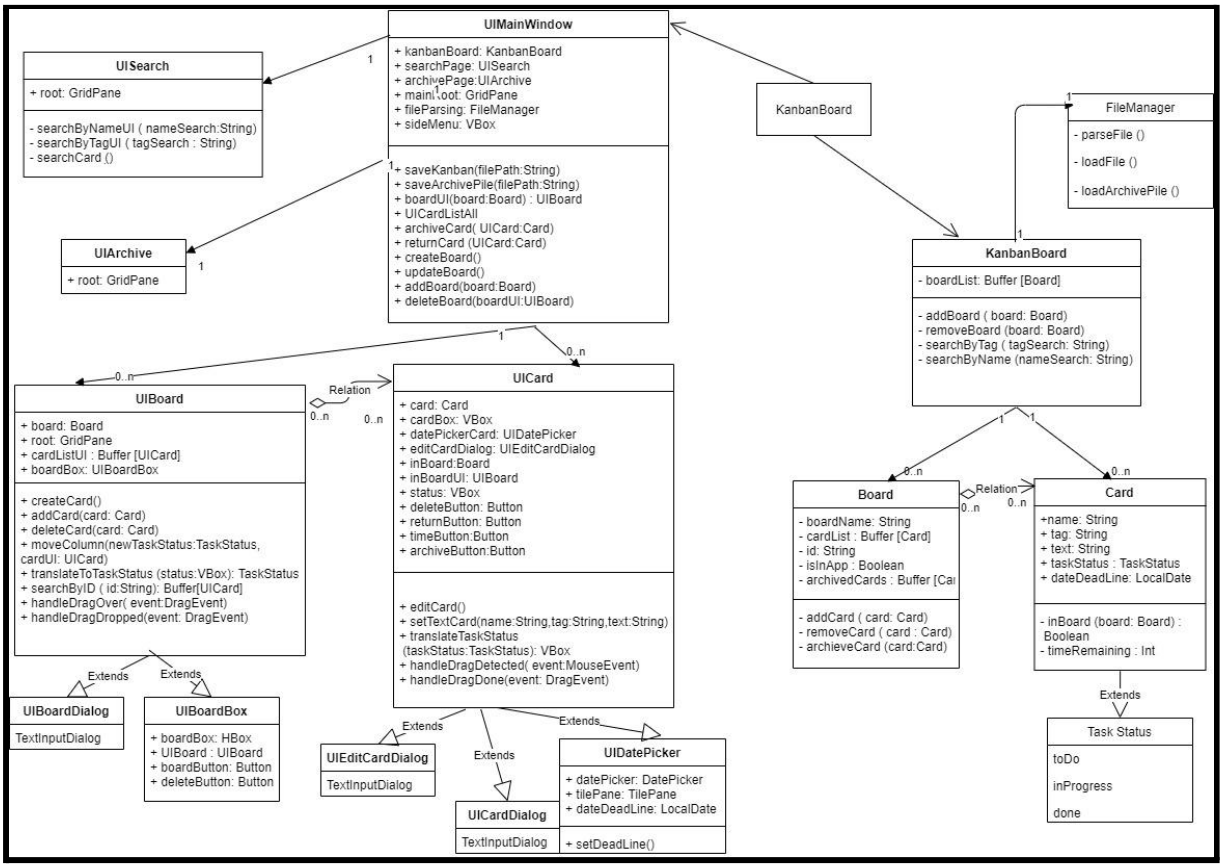
Here is an image of the search page view. Users can search for cards based on tag/name and the results are displayed in the boxes below.



Card objects can be dragged and dropped between three columns in the board views. The first icon box in the card displays an edit icon, a delete icon, a set deadline (date picker) icon, and an archive icon respectively. The next three rows display card name, card tag, and card text respectively.



3. Program structure



Link:

https://drive.google.com/file/d/1FcsPDQqCoJXqN07Q3aO6JbXgZoHEy-ub/view?usp=sharing

The program is separated into two sub-packages, package Kanban and package GUI, to improve the readability of the code.

Package Kanban represented on the right of the UML and under the path includes the implementations of object KanbanBoard, class Board, class Card, and class FileManager.

KanbanBoard stores all the boards and some core methods of adding/removing boards, searching for cards, and returning cards from an archive pile.

Board class stores all properties of cards in that board and some core methods of adding/removing cards from/to board and archiving cards.

Card class represents an instance of a card and its properties.

FileManager class allows the program to parse the data from the saved file and write data to KanbanBoard.

Package GUI represented on the left of the above UML represents sub-components UI used in this program.

UIMainScreen is the main object used to run this program.

UIMainWindow represents KanbanBoard, which stores the methods of adding/removing board, archiving/returning cards, and saving Kanban and its archive pile to file. It also extends ***UISearch*** and ***UIArchive***, which represent the search page and archive pile respectively.

UIBoard is the class that stores some methods related to cards and all Card instances. It also extends ***UIBoardDialog*** and ***UIBoardBox***, which are responsible for creating boards, showing board views, and deleting boards respectively.

UICard is the basic class representing an instance of a card in UI. This class extends ***UICardDialog***, ***UIEditCardDialog***, and ***UIDatePicker***, which are responsible for creating cards, editing cards, and picking the deadline date for a card.

The final UML structure is more complicated than the planned UML structure as more classes representing the GUI are added and some methods and properties are changed to improve the functionality of the program during the development phase.

5. Algorithms

The filtering/searching algorithm

This filtering method is used to search for cards based on their tag/name. This function was done by a recursive loop through all the cards in the KanbanBoard and return card that matches the input keyword. Another filtering function used in this program to find a specific instance that satisfies the specific condition is method “find”.

Drag and drop

This is one of the most frequently used functions of cards. Every card will have its unique cardID when created, and this cardID will be put into a drag board whenever the user clicks on the card and drags their mouse. This drag board is used for the drag-and-drop data transfer. When a user releases the card into board columns that are defaulted to accept cards, the board will search for cardID from a list of cards and then accepts the matching cards to be dropped into the board columns. The task status of the card will then be updated.

6. Data structures

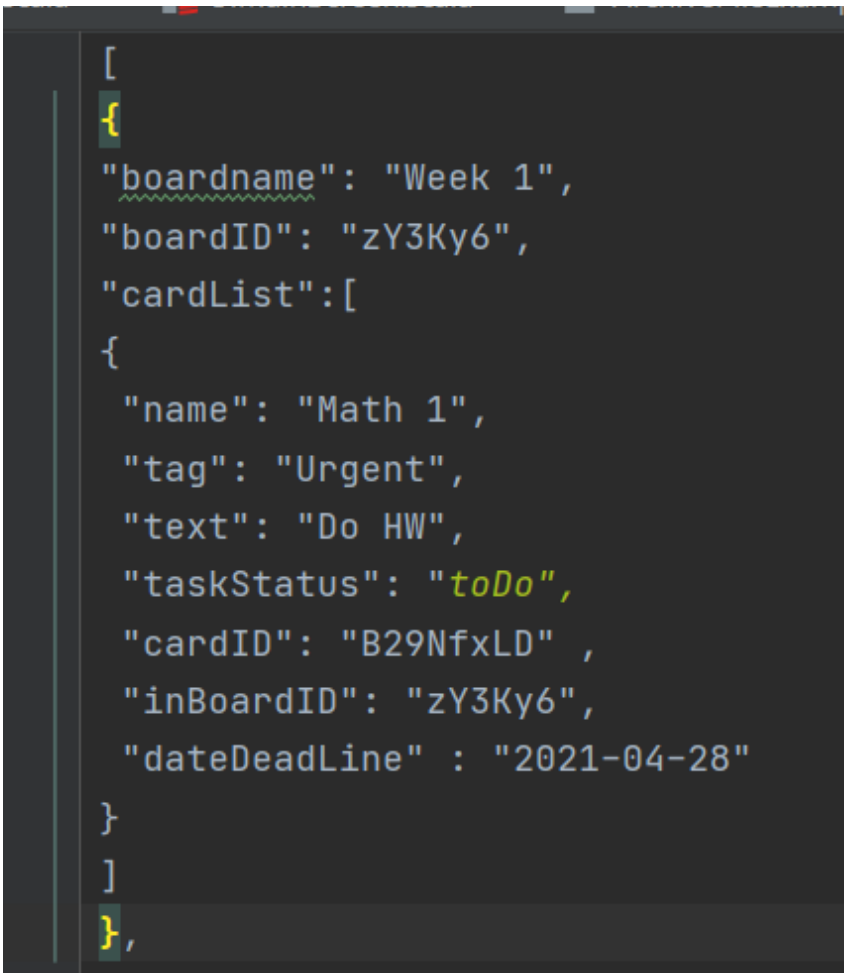
The program consists of a lot of Board and Card instances, so collection structures should be chosen to store them. Buffer collections were preferred in this program as these collections are mutable, so the state of the object can be updated when a user adds or removes board/card later on. I would say that Buffer worked well in this project.

No user-defined data structure is created in this project.

7. Files and Internet access

There are 2 separate data files used to read and write data from/to the Kanban in this project: one data file is for the Kanban Board itself, another data file stores the data of cards in an archive pile. The data is stored in JSON format into a text file ".txt" under the path "src/main/scala/Data". The JSON data is converted and parsed into a list of Board objects with nested data of a list of Card objects inside that board. The parsing of the data is implemented with the help of the Circe library to make the parsing code cleaner. I would say that using a well-known JSON format in this program is a good choice for this project, as JSON uses a map data structure that has key/value pairs that made it readable for nested data like the data in this program.

Here is an example of the JSON data used in this project. This example data contains Board objects with Card objects inside.



No Internet network access is required for this program.

8. Testing

The program includes unit testing under the path "src/test/scala/UnitTesting.scala". The test is made using the Scalatest library. It tests some core methods implementing the logic of the Kanban board such as add/remove cards to/from boards, add/remove boards to/from Kanban, archive/return cards from archive pile, filter/search cards based on card tag/name. There are multiple tests for each method and multiple assertions in each test, including assertion before applying the methods and after applying the methods to test the change in the state of the boards/cards.

The GUI is tested manually after creating each component to check the alignment between different components in the program. Different use cases such as dragging and dropping cards between columns, clicking the button, adding/removing cards

to/from the board are extensively tested in the GUI to test the functionality of those components and the effect of them on program objects and some core methods. The logic side and the interface is also tested together in the GUI. There was a button called Test which used the function “println” to test the change of state of Card/Board objects. However, this button was deleted in the final GUI.

9. Known bugs and missing features

According to the requirement for the intermediate level, there is just only one minor missing feature related to the cards. The program is designed to drag and drop or remove only one card at a time, not many cards (or card lists) as mentioned in the small part of the requirement.

The file seems to take a lot of memory (about 400 MB) when I first zipped and tested the program. This happens because something wrong happened when I first used VCS in Scala to commit and push this programming into Gitlab. However, the problem was solved.

10. 3 best sides and 3 weaknesses

3 best sides:

The GUI is well-designed, with many icons and effects added to increase the ease of use and the aesthetic of the program.

This program supports all the basic requirements and functionalities of the Kanban App.

The logic code of this program is readable with a simple class structure and variable naming, which make it easy to follow and modify later on.

3 weaknesses:

The class structure of the GUI is not completely based on the logic side in some parts. This is mainly because the card object doesn’t take a UICard object as a parameter, which will later cause a thread when running the program.

This structure makes the code a little longer in some parts; however, all the logic variables of the Kanban are updated whenever any change happens in the GUI, which makes the program still functional.

The program still misses one minor feature mentioned above.

11. Deviations from the plan, realized process, and schedule

The first three weeks were for the implementation of the core logic methods and the unit test. The GUI seemed to be harder than I planned, so I put more time and effort into implementing this. The remaining time was mainly for the implementation of the GUI and file handling. One of the main functions of the GUI but also the hardest part to implement is the drag and drop function. The most considerable effort was put to implement this function, about 1.5 weeks.

I possibly learned a lot during this process. There are not many useful resources about ScalaFX, so I learned to do the GUI by learning the materials in JavaFX and implementing it in the language Scala. After overcoming many difficulties during the project, including debugging my code and figuring out how to implement some methods, I took away a lot of valuable lessons.

Overall, it is a worthwhile learning experience to apply the knowledge I have learned in Programming 1 and this course with my first hands-on project to know how one program with the combination of logic methods and the user interface is made and what steps/phases including in creating one program.

12. Final evaluation

I would say that the project is acceptable in terms of its functionality. In the end, I had managed to implement almost all the requirements mentioned in the project plan. The program interface is natural and easy to understand. However, this program is nowhere compared to some actual online Kanban Board. The project can be extended and improved in various ways with more features added to enhance the functionality and usability of the Kanban. The GUI is natural and easy to use but can be more attractive to improve its aesthetic. If I were to start the project again, I would plan the structure of the GUI more tightly with the core logic methods.

13. References

Scala API Library Document

Link: https://www.scala-lang.org/api/current/

ScalaFX API Library Document

Link:https://oss.sonatype.org/service/local/repositories/releases/archive/org/scalafx/scalafx_2.13/16.0.0-R22/scalafx_2.13-16.0.0-R22-javadoc.jar/!/index.html

ScalaFX Documentation Dialog and Alert

Link: http://www.scalafx.org/docs/dialogs_and_alerts/

JavaFX Drag and Drop

Link: https://docs.oracle.com/javafx/2/drag_drop/jfxpub-drag_drop.htm

FlatIcon

Link: https://www.flaticon.com/

14. Appendices