Onni Tammi
Computer Science
1. year
25.4.2023

Football dashboard: Project document

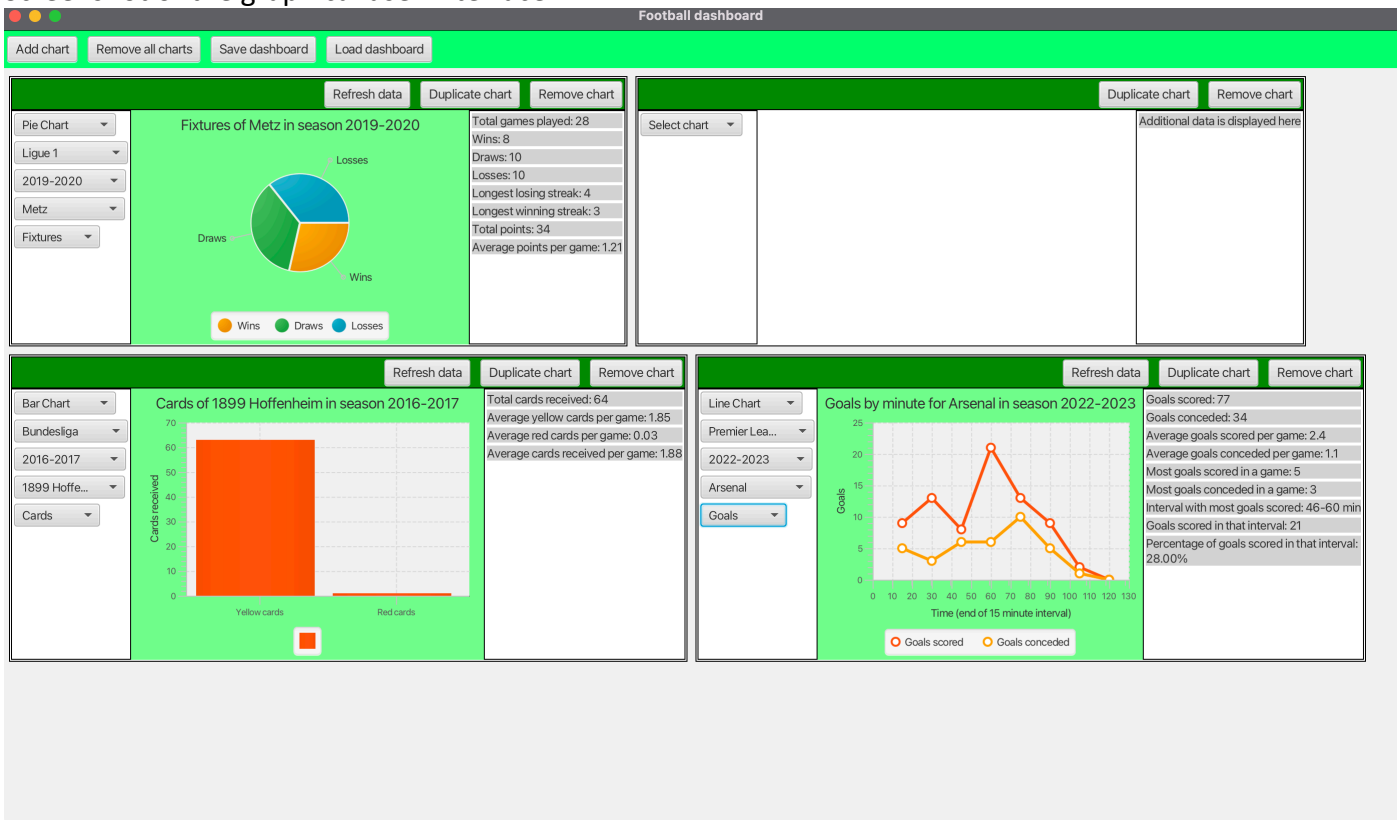
General description

The football dashboard is a data dashboard that displays football data retrieved from the API-football API. The desired data can be selected from various leagues, seasons, and clubs. The exact data set can also be selected from fixtures, goals, and cards. The data is displayed using pie charts, bar charts, line charts, and cards with additional data. The data displayed in a chart and card varies based on the selected chart type. The user can display multiple chart boxes. One chart box always contains a data selection panel, a chart, and a card with additional data. The size of the chart boxes can be changed. The chart boxes can also be duplicated and removed. The user can save the dashboard to a file and load it from a file. The general appearance of the final project doesn't have any substantial changes compared to the plan.

I planned to complete the demanding level project. However, my project is missing some of the demanding level requirements, like ability to specify colors for individual components and selecting data points/components by using a rectangle selection tool. My project does still fill all the moderate level project requirements, except the ability to hide components. Therefore, I think the project was finally completed on the moderate level.

User interface

Screenshot of the graphical user interface:



When the program is launched, some buttons show up at the top of the window. There is an empty area under the buttons, where new chart boxes can be added. The user can add a chart box to the area by clicking the “Add chart” -button. The user can add as many chart boxes as they want to the chart area. All chart boxes in the chart area can be deleted by clicking the “Remove all charts” -button. All the chart boxes in the dashboard can be saved to a file by clicking the “Save dashboard” -button. The chart boxes can be loaded from a file by clicking the “Load dashboard” -button. If loading the file fails, a pop-up window with more information about the error shows up.

The chart boxes in the chart area can be resized by dragging them from the bottom or right border. When the mouse hovers on a draggable area, the cursor icon changes to a diagonal double-headed arrow. The chart area is a flow pane. Therefore, when a chart box is resized, the chart boxes next to it are moved so that the distance between them is maintained. When the height of a chart box is changed, the heights of all other chart boxes on the same row are also changed. If the rightmost chart box on a row can’t fully fit in the window, it goes to the left side of the next row, and the same thing happens also on that row if all chart boxes can’t fit on it and so on. The first chart box added to the chart area has a preset height. After that, the height of a new chart box is initially set to the same height as the previously added chart box. All chart boxes added to the chart have initially same preset width. The chart boxes also go on to the next row if they can’t fully fit into the window when the size of the window is changed.

When a chart box is added to the chart area, it appears as the one in the top right corner in the screenshot. It doesn't have any chart or card yet, but only a list box for selecting chart type. When the "Select chart" -list box is clicked, a list of chart types shows up. The list contains pie chart, bar chart and line chart. The chart type can be selected by clicking an item on the list. When the chart type is selected, the selected chart with sample data shows up in the center of the chart box. On the same click a new list box with a "Select league" -prompt text shows up under the chart selection box list. The new league selection list box works in a similar way with the chart list box. The league box contains a list of five leagues: Bundesliga, La Liga, Ligue 1, Premier League, and Serie A. When the league is selected, a new list box with list of seasons appears under the league box. The list has seasons from 2015-2016 to 2022-2023. When the season is selected, a new list box and a loading icon next to it show up under the season box. When the clubs from the selected league in a selected season are loaded from the Internet, the loading icon disappears, and the club can be selected. If loading the data fails, the prompt text in the club selection box changes to "Error loading data". When the club is selected, the selected chart in the center is updated with fixture data of the selected club in a selected season. If loading the data fails, the title of the chart changes to "Error loading data". At the same time a new list box for selecting the data set shows up under the club box. The initial value of the data set list box is "Fixtures". The user can also select "Goals" and "Cards" data sets from the data set selection. The data selection panel of the chart box takes up quite a lot of space. I thought about adding a button that can be clicked to hide and show the selection panel but I didn't have enough time to implement that.

The selections can be changed afterwards, and the data in the chart and card is updated based on the new selections. If all the selection boxes have a selected value, and the user changes the chart type or data set selection, the chart and card are updated immediately (league, season and club are unchanged). If the user selects a new league or season, the clubs are updated. The chart and card are updated only after the club is selected from the new league and season. The chart type and data set selection remain unchanged. New data is loaded from the Internet every time a new club is selected.

When a new chart box is added, there are initially two buttons at its top area. The "Remove chart" -button removes the chart box when clicked. The "Duplicate chart" -button duplicates the chart box so that the selected data, chart, and card are retained. When a club is selected from the club selection box, a "Refresh data" -button shows up at the top of the chart box. By clicking the refresh button, the user can update the data in the chart and the card. If the club is not selected, the title of the chart changes to "Select club before refreshing", and the chart is updated only after the club is selected.

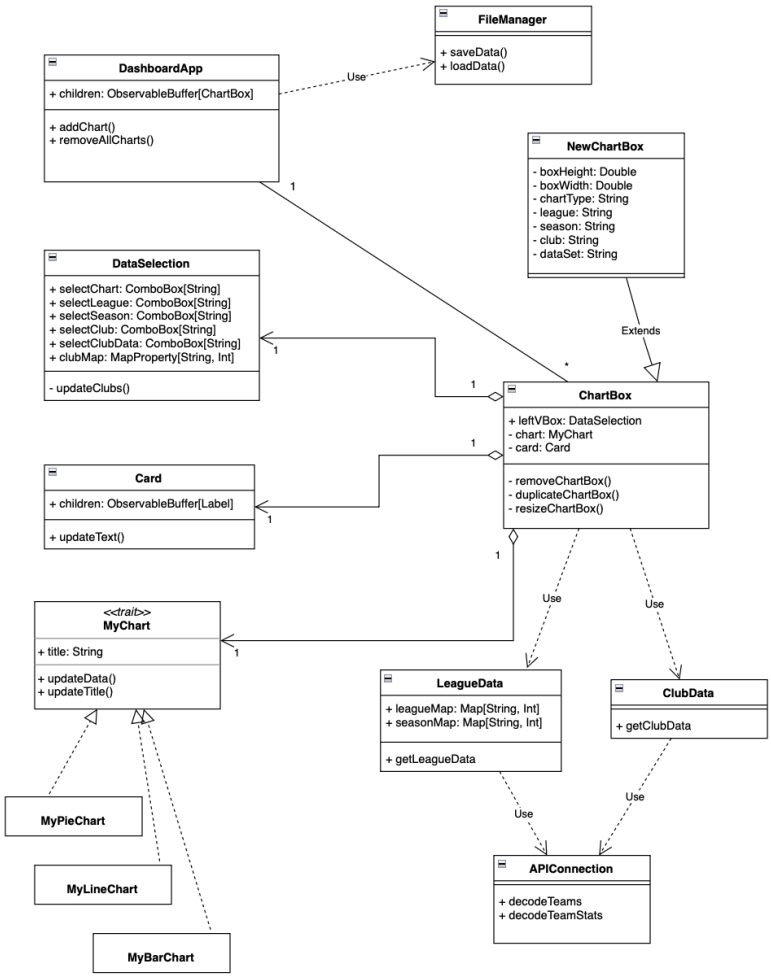
The chart in the middle of the chart box changes based on the selections made on the left side of the chart box. The chart has a title, which tells the data set, club, and season displayed in the chart. The chart has also legends and axis titles that provide more information about the data. The user can hover their mouse on data points to show more information about them in tooltips. The pie chart and the bar chart display the same discrete data when they have the same data set selected. For example, when the "Fixtures" -data set is selected, they display wins, draws and losses of a club in a season. The line chart displays a bit different data because it wouldn't make much sense to display discrete data

on it. For example, when “Fixtures” -data set is selected, it displays how many points the club has received from every game during the season.

The card on the right side of a chart box displays some additional data about the selected data set. Pie chart and bar chart have the same additional data shown in the card when they have the same data set selected. With line chart, there is also some more data displayed in the card in addition to the data displayed with pie chart and bar chart. For example, when the “Fixtures” -data set is selected with a bar chart or pie chart, the card displays information like total games played and average points per game. When a line chart displays the same data set, the card has some more additional information like standard deviation. I could have added a lot more additional data to the cards, but I thought it would take up too much space from the chart area, in which case the user wouldn’t be able to add so many chart boxes to the area. The card could have been implemented as a separate component that can be added to chart area, but I thought it would be more convenient to connect the card with the chart and selections.

Program structure

Final UML structure:



This UML showcases the final class structure of the football dashboard project. Some classes, methods and parameters have been left out of the UML to improve readability. Some of the methods in the UML don't really have a name in the code because they are written directly as a parameter for some ScalaFX method.

The DashboardApp class is responsible for displaying the initial window when the program is launched. It handles the presses of the buttons at the top of the window. The class has its own methods for adding new chart boxes and removing all chart boxes from the chart area. When the save or load button is pressed, the class calls the saveData and loadData methods in the FileManager object. Before calling the saveData method, the DashboardApp class converts the data in the chart boxes into a suitable form. The class also has a method for showing a pop-up error when loading a dashboard fails.

The ChartBox class represents chart boxes added to the chart area. This class extends ScalaFX StackPane class. The class has some methods for handling the resizing of a chart box. It also has a method for removing the chart box from the chart area. This method calls the parent of the chart box and removes this chart box from the children of the parent. The duplicate method of this class creates a new chart box with the same data and size as this one and adds it to the chart area. This is implemented by creating a new instance of NewChartBox class with the current parameters in this chart box. The class also has a method for refreshing the data in the chart of the chart box. The main components of this class are data selection (DataSelection class), chart (MyChart trait), and card (Card class). This class also listens to some properties in the data selection part and updates the data in the chart and card when these properties are changed. For example, when the club selection is changed, this class calls the getClubData method in the ClubData object and updates the chart and card. The new data is wrapped in a Response case class and the Response case class is saved to this class, so it can be easily accessed when other parameters in the selection area change. I could have made a method that is used in both updating the data when a club is changed and updating the data when the refresh button is pressed. Now the code for those separate methods looks very similar but it is still repeated in both.

The DataSelection class extends ScalaFX VBox class. It is responsible for displaying the data selection in a chart box. This class has a vertical list of ComboBoxes, which are used to select the chart, league, season, club, and data set displayed in the chart box. Initially only the chart selection box is visible, and other combo boxes become visible when a value is selected in a combo box above them. This class has listeners for all the combo boxes to perform some actions when their value is changed. The club selection combo box differs a bit from other combo boxes. It is in a HBox container because when a new season or league is selected, the new clubs are loaded from the Internet and a loading icon shows up next to the combo box for the duration of the loading process. The updateClubs method is responsible for this functionality. The method calls the getLeagueData method in the LeagueData object, creates a new combo box with a list of the received clubs and adds a new listener for the new combo box so that the selected club can be used by the chart box. The updateClub method is called every time the value of the season or league selection is changed.

The MyChart trait extends ScalaFX Chart. This trait is a supertype of the various charts that can be displayed in a chart box. This trait has two abstract methods: updateData and updateChart. The updateData method updates the data in the chart, and the updateTitle method updates the title of the chart. The updateData method receives as a parameter a Response class which contains all the processed data used in the chart. The method also has a parameter, which tells the data set so that the right data can be selected in the Response class. The updateTitle method has as parameter the selected club, season, and data set so that the right title can be displayed. These methods are implemented in the subtype classes MyLineChart (extends ScalaFX LineChart), MyPieChart (extends SFX PieChart) and MyBarChart (extends SFX BarChart). Making a common trait for these classes was a good idea and made working with different charts quite simple. All the chart classes also have a method for displaying tooltips in the chart.

The Card class extends SFX VBox class. It is responsible for displaying additional information about the selected data in a chart box. It has an updateText method which is called from the ChartBox class when the selected data is changed. This method has as parameter a Response class, which contains some statistics of a club. These statistics can be easily accessed. The method also has a data set and a chart parameter for selecting the desired data in the Response class. The method displays a vertical list of additional information about the selected data. The text is created by using a simple MyLabel class which extends SFX Label class. The MyLabel class has some preset properties to display the text as desired.

The NewChartBox class extends ChartBox class. The class is used to create new chart boxes with preset size and data selections. For example, when a chart box is duplicated, this class is used to create the new chart box with same selections and size as the one where the action was performed. This class is also used for creating the chart boxes when a dashboard is loaded from a file. I came up with this class when I tried to implement the duplicating method in the ChartBox class. The ChartBox class had such a structure that I couldn't implement the method solely in the class, so I created the NewChartBox class which takes parameters. I also could have changed the ChartBox class so that it takes parameters and initializes the ChartBox with those parameters. However, I didn't want to break the structure of the ChartBox class because it was working just fine, except for the duplicating method. Creating the NewChartBox class turned out to be a good idea because it is also used for loading the dashboard.

The APIConnection object is responsible for connecting to the API-Football API and fetching data. This object reads the API-key from a file. The API-key is then used in the fetch method. The fetch method is used for fetching data from the given URL. The method uses Requests-Scala library to connect to the API and retrieves a JSON as a string. The object also has decodeTeams and decodeTeamStats methods. They use the circe library to decode the retrieved JSON string into a case class. The case class structures for these methods are defined in the LeagueData and ClubData objects.

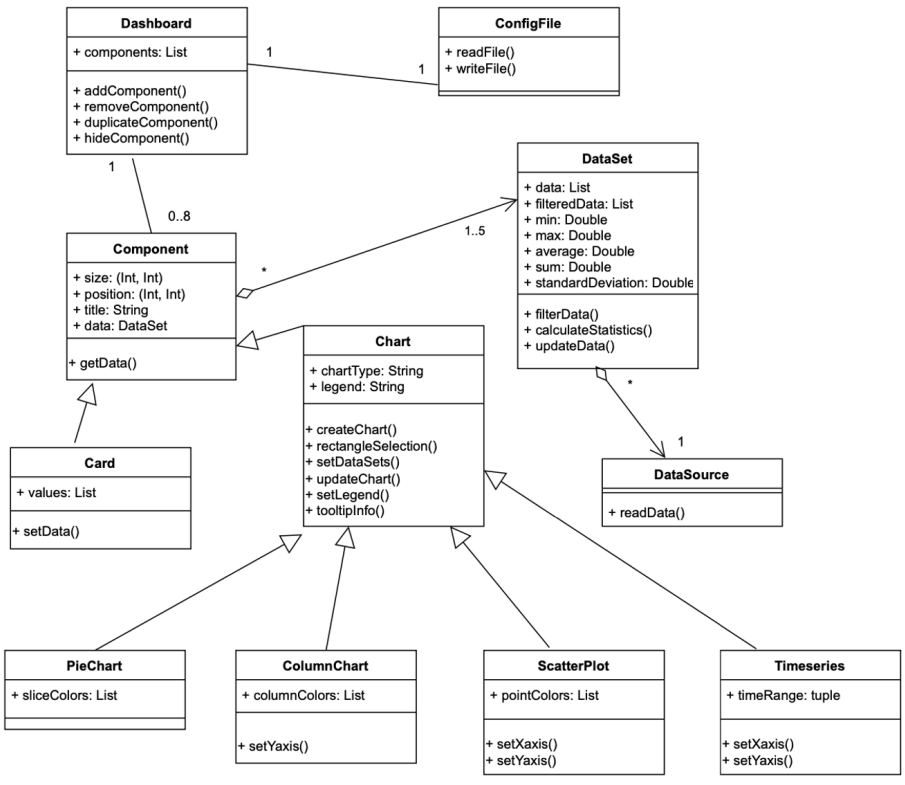
The LeagueData and ClubData objects have a quite similar structure with each other. They have getLeagueData and getClubData methods that construct the desired URL based on the given parameters. The URL is then used in the methods to call the fetch method in APIConnection object and to decode the JSON into their own Response case classes using

the decode methods in APIConnection. Both classes have multiple case classes to structure their Response case classes. Especially the Response case class structure in the ClubData object is quite complex. The LeagueData Response class is only used for listing the clubs in a selected league in a selected season. In contrast, the ClubData Response class has a lot of different variables that can be accessed by other classes that use this Response class. Some of the variables are directly read from the JSON, whereas some variables are calculated based on other variables in the JSON. These variables are used to display the data in a chart box.

The FileManager object is used to save and load a file containing information about chart boxes. The saveData method in the object writes the heights, widths, and data selections of the chart boxes in the chart area into a file in a human readable format. The loadData method is responsible for reading the data in a saved file and creating new chart boxes based on the data. There is also a FileManagerException class which extends Exception class. This class is used to specify the errors that may occur when loading the data.

The final class structure is quite similar with the one I planned. The final class structure has some classes, like the NewChartBox class, that are not in the planned class structure. There are also multiple classes for processing data instead of just one DataSource class. Some of the classes in the final project have different names and a bit different functionality compared to the plan. Also, the card class is not a separate component, but is included in a chart box.

Planned UML structure:



Algorithms

The program uses some algorithms, like sum, average, and standard deviations, for calculating various data displayed in the dashboard. The sum is calculated just by summing up all the values in a list. The average is calculated by dividing the sum of a list with the number of data points in the list. Standard deviation measures how spread out the data point values are. The formula for standard deviation is $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$, where N is the number of data points, x_i is the value of a data point and μ is the average of the data set. I could have also implemented some more complex algorithms for processing data, but I didn't have time to think about these implementations. The program also uses algorithms for processing strings. For example, the form of a club is initially stored in a string with different chars. The char list is converted into a list of points corresponding to the chars. The initially retrieved JSON string is also modified because it has some value names like numbers and "for" that can't be used as value names in Scala.

Data structures

The most important data structures in the program are the InitialResponse case classes in ClubData and LeagueData objects. They represent the JSON strings that are retrieved from the API. The InitialResponse case classes consist of other case classes that match the structure of the JSON. The different fields in a InitialResponse can be accessed by giving the class as a parameter for a Response case class. The paths to the InitialResponse values used in other parts of the program are written in the Response case class so they can be easily accessed with short commands. The Response case class also contains some predefined calculations on the data in InitialResponse case class. This data structure is immutable.

I have also used a lot of Map collections in the program. They are needed to bind the names of leagues, seasons, and clubs with their corresponding IDs. The Maps used in this program are immutable. Some sequential club data is stored using Lists. I chose List as a collection because it is immutable, and it is faster for sequential access than a Vector for instance. Mutable Buffers are also used in some methods to save the values when looping through some data.

Files and Internet access

The program uses a human readable format to save the data into a file. The file contains information about the height, width, selected chart type, selected league, selected season,

selected club, and selected data set of every chart box in the chart area. Here is an example of a saved file:

```
CHARTBOXES:
BOX0:
HEIGHT: 286.0
WIDTH: 612.0
CHART: Line Chart
LEAGUE: Bundesliga
SEASON: 2022-2023
CLUB: Bayern Munich
DATASET: Cards
*
BOX1:
HEIGHT: 286.0
WIDTH: 693.0
CHART: Pie Chart
LEAGUE: La Liga
SEASON: 2018-2019
CLUB: Barcelona
DATASET: Fixtures
*
BOX2:
HEIGHT: 220.0
WIDTH: 570.0
CHART: Bar Chart
LEAGUE: Bundesliga
SEASON: null
CLUB: null
DATASET: Fixtures
#
ENDOFFILE
```

The height and width of a chart box are saved as doubles. Other values are saved as strings. If some selection doesn't have a value, it is written as "null" into the file.

The program accesses the Internet to retrieve data from the API. This functionality is already described in the Program structure and Data structures sections of this document.

Testing

I mostly tested the program just by using the GUI. The GUI is an important part of the program, so it was intuitive to test it that way. Every time, I implemented a new functionality that affected the GUI, I tested that it works as it should in different scenarios. I also used the REPL for testing some simple functionalities. For example, I used the REPL to test that the club data in the Response case class is what it should be. This was easy because the API had a website that showed the correct results for different API requests with different parameters. I also built few unit tests to test that connecting to the API works properly and it returns the correct data.

Known bugs and missing features

The program has a few minor bugs that I haven't been able to solve or didn't have enough time to address. For example, if all the values in the data selection of a chart box are selected and the user picks a new league or season, the clubs are updated. In this case, if the user duplicates the chart box before selecting a new club, the duplicated chart box won't have the same chart as the original chart box. Instead, the chart shows the sample data which is usually shown in a new chart box before a club is selected. This happens because the chart isn't changed when a new league or season is selected, but only when a new club, chart type, or data set is selected. However, the data selections are remained, except that the data set selection is not visible. This bug could be fixed by also saving the league, season, and club displayed in the chart, and giving these values as parameters to the NewChartBox class. Now only the data selections are given as parameter to the constructor class. The same bug also occurs when the user loads a saved file with some club data displayed in the chart, but no club selected.

The program has also another little bug. This bug also occurs when all the values in the data selection part of a chart box are selected, and the user selects a new season. In this case, if the user selects the same club as displayed in the chart again, the chart won't update. This happens because the chart is updated when a new club is selected, and in this case the club is not changed. The chart updates only after a new chart type, club, or data set is selected. This could be fixed by updating the chart when the value of the club selection box is changed. Now the chart update is triggered only when the saved club ID is changed.

I had planned that the user would have an ability to select the colors of charts. However, I didn't implement that although I tried. I managed to change the color of the series and data points in a chart, but the colors in the legends didn't change to correspond to the new data point colors. Therefore, I gave up with implementing that functionality because I had some other more important functionalities that I hadn't implemented at that moment. I didn't implement the planned rectangle selection tool either because I had no idea how I would implement it. In addition, the selectable data in the dashboard is not as extensive as I had planned. Adding new data sets to the dashboard is quite easy but it takes some time to write the functions for handling different data, and I didn't have the time. I could implement this functionality after the course.

3 best sides and 3 weaknesses

Making a common trait for all charts was a good idea. I didn't know much about traits and how they work before the implementation. However, it turned out to be a very good idea and made working with different charts quite easy.

The structure of APIConnection, LeagueData, and ClubData classes is also convenient. It is easy to add new data points and calculations that can be accessed by other parts of the program.

On the other hand, the structures of ChartBox and DataSelection classes are quite bad and incoherent. It is quite difficult to add new functionalities to these classes and the new functionalities may have unexpected side effects in other classes. The main purposes of the classes are mixed. For example, there are some listeners for the same data selection boxes in both classes. There is also a lot of repetition in these classes. Also, when a value is changed in a data selection box, the new value is saved to multiple different variables, which makes the code quite complex.

Deviations from the plan, realized process and schedule

I started the project by implementing the basic layout for the GUI. Next, I implemented the API connection and data fetching functionalities. At this point, I didn't have a clear view on how I would construct the GUI. I realized the GUI I had in the project plan would be difficult to implement and not very intuitive to use. Eventually, I decided to use a SFX FlowPane to layout the chart boxes.

Then I implemented the chart boxes which were initially empty. The resizing of the chart boxes turned out to be quite difficult. This was because the changed heights of the boxes weren't saved automatically if they were changed when some other chart box was resized. I struggled for a long time with the resizing functionality but eventually came up with a solution to the problem; I had to write some code to change the height of all chart boxes on the same row. I also used a lot of time to think of a good way to deal with the user selections. I decided to place the user selections in the chart boxes, so that every chart box has its own selection and data displayed.

After implementing the data selection in a chart box, I started working with the data and displaying it in a chart. I used a lot of time on these functionalities because they were an essential part of the program. Once I was satisfied with these functionalities, I started implementing the cards. I had planned that the cards would be separate components in the dashboard and could be added in the same way as chart boxes. However, I decided to bind them with the chart boxes because I thought it would be easier to implement and I didn't have much time left at this point. I'm quite happy with the result and think it was a good choice to add the cards to chart boxes because they display some nice additional data in a clear manner.

Then, I started implementing the duplicating functionality of a chart box. I had to make a new constructor class for chart boxes because I couldn't add the same chart box instance twice to the dashboard. After that I started to implement the saving and loading functionalities. Loading new chart boxes was quite easy with the new chart box constructor class. These were the last large functionalities that I implemented.

I managed to follow the schedule for the first two weeks. After that I didn't work as much on the project as I had planned and wasn't therefore able to implement the things written in the schedule. During the last two weeks I used a lot more time on the project that I had planned and was also able to achieve more goals in the plan. Almost all the components in a chart box and the data processing functionalities were implemented during the last two weeks. Also, the saving and loading functionalities were implemented in that time. The order in which the project was finally implemented, was roughly the same as I had planned. The charts and cards were eventually implemented after the data selection but otherwise there were no big changes to the plan. In total, I have used approximately 80 hours on the project which is 20 hours less than I had planned. If I had used the additional 20 hours on the project, maybe I could have implemented all the things I had planned.

Final evaluation

I'm quite happy with the outcome of the project. It can be used to display and compare various football data, which was my personal goal for the project. It fills all the moderate level requirements, except the hiding functionality of components, and it also fills some of the demanding level requirements, like loading and refreshing data from the Internet. The program is also convenient to use.

The GUI of the dashboard isn't very appealing because I decided not to use too much time on it since the look of GUI isn't an essential part of the evaluation. However, I could improve the GUI, if I continue to develop this project after the course. I could also add more comprehensive data to the dashboard because implementing that should be quite easy. On the other hand, adding new components to the DataSelection and ChartBox classes could be quite frustrating because they have a lot of functionalities that are dependent on each other.

In general, the quality of the code could be better. It has some repetition, and some methods could be divided to smaller auxiliary methods. The classes could be more independent and the interfaces between them could be designed better.

If I started the project now, I would change the structure of the ChartBox and DataSelection classes and add parameters to the ChartBox class. I would also use more auxiliary methods in the code. I would have also used more time on the project at the beginning, so that I wouldn't be so busy at the end of the project.

References

API-Football: https://www.api-football.com/
Rapidapi: https://rapidapi.com/api-sports/api/api-football/
Circe: https://circe.github.io/circe/
Requests-Scala: https://github.com/com-lihaoyi/requests-scala

ScalaFX API: https://javadoc.io/doc/org.scalafx/scalafx_2.13/latest/index.html
ScalaFX Ensamble: https://github.com/scalafx/scalafx-ensemble
Mark Lewis Youtube videos:
https://www.youtube.com/playlist?list=PLLMXbkbDbVt9MIJ9DV4ps-_trOzWtphYO