École Centrale Électronique

Report Projet mobile

Mobile Programming Project

Smits Victor
Snyers Harold

Professor : Christian Khoury

1er juin 2020

Table des matières

Hockey Trainer

1 Function

1.1 Present

Picture

Ability to take pictures within the app during the recording, to store them locally on the phone and being able to have a preview of these pictures in *detailed match page under the picture tab*.

Save statistics

Ability to save statistics of the all matches in a MySQL database running outside the app on a hibernate server.

Save statistics locally

Ability to save statistics of the 5 previous match in a SQLite database running within the app.

Ergonomic Interface

User-friendly app interface using fragments, a menu drawer, 2 languages, and adaptive orientation.

Location

Ability to localise the match and returning the address.

Match interface

Interface allowing the user to see a resume of each recorded match.

1.2 Missing

2 Improvements

For the improvement, we thought about building a complete API. With the API you could be able to add, delete, or even edit a recorded match. We also thought about cancelling a recorded action, for example, a goal which is cancelled after. we have also planned to add a time reference to the recorded actions.

The other improvement is to link a user with a match. People can create an account for a single person or even a team. With the team account, the trainer can follow the team match stat and see directly where the team need to work further.

We also thought about a link between a player and an action. With this link, we could develop player statistics and evolution during the season and between each match.

The last improvement is an overall scoreboard for the season. With this scoreboard, everyone will be able to follow the current season's score. We can also implement other type of ranking such as ranking base on fault or card.

3 Architecture & Implementation

3.1 Data Architecture

Our application is using 2 databases, an external database and an internal database. The external database is a MySQL database. The application is communicating with this database through a hibernate server. This external database is used to store the general match details as the team's names, the location of the match, the data of the match and finally the image path for the pictures taken during the match. It stores then also the details of the match in an other table called Quarters (further explication below).

The internal database is a SQLite database. This database is used to store the last 5 matches recorded, and allows a faster access to the data without the need for internet access by the application. The application is requesting data from the database by the *DatabaseHelper* which is extending *SQLiteOpenHelper*.

External database

The MySQL database stores details about each recorded match (see figure 1.3.1) and also a *Quarter* table which will store details about each quarter of a match. Each match is composed of several quarters, we have so a relation one-to-many between the matches and the quarters.

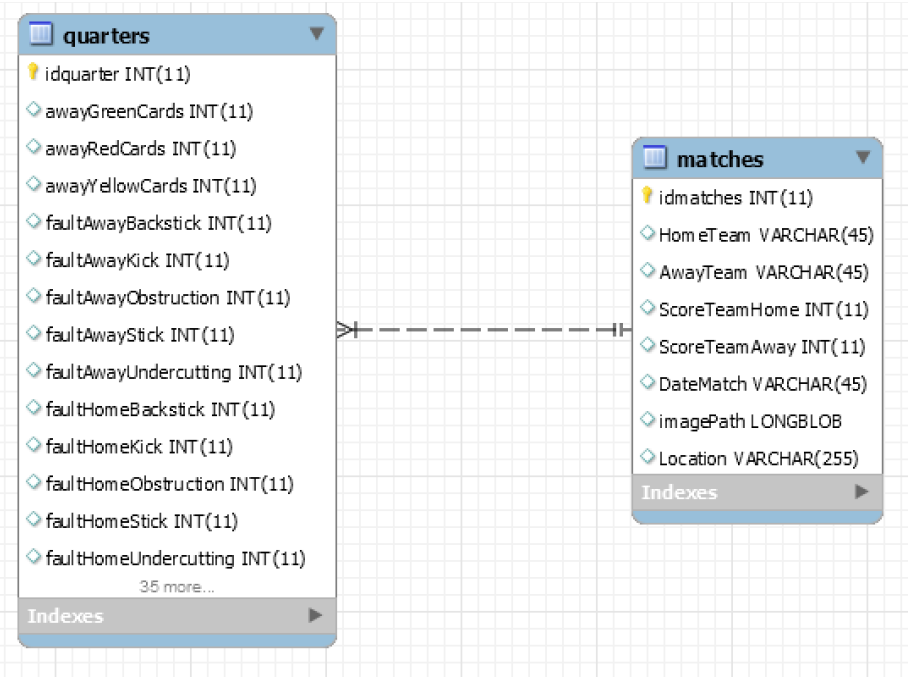

FIGURE 1.3.1 – MySQL database structure

To communicate with the database, an Asynctask class is used while inserting in or fetching from the database the needed data. As this fetch can take some time, we are using this Asynctask in order to not freeze the UI during this process. Our application is then communicating with the hibernate server through a socket. The data communicated between the two is in JSON form as it makes it easy to send object with it.

The hibernate server is fetching through a socket the JSON string. This JSON string can have 3 different actions : *INSERT*, *GETALL*, *GETMATCH*. Those action are implemented inside the *DownloadModel*. The *INSERT* method is used to create a new entry inside the database by sending JSON strings composed of the match summary and quarter details. This is done with an objectmapper to convert the object as a JSON String. The *GETALL* action sends a JSON object with a key-value pair requesting all past matches. The last method, *GETMATCH*, requests a specific match by communicating the match id inside the JSON object.

To communicate with the MySQL database, we developed a server based on the Hibernate library. We connect our application to the server through a socket, and write the JSON object as a string inside the output stream. In the server we collect the data from the stream. First we request the mode key to identify in which mode we are (*INSERT*, *GETALL*, *GETMATCH*) then if we are in *INSERT* mode we ask for all the value corresponding to a key and build an object Match and Quarter with those values.

We have developed 2 class Match and Quarter, to make the use of data easier in the server. Those classes are base on the database table and each one represent one table. When we request data from the database we transform the result data into an object Match and 4 object Quarter.

internal database

The internal database is build around a SqLite database. As we use this database to store the 5 previous matches locally, the data stored inside is the same as the data store inside the MySQL database. Although there is no explicit relation between the 2 tables, the quarter table does contain the id of the match it is connected to.

To interact with the database we have implemented a class called *DatabaseHelper*, which extends the *SQLiteOpenHelper*, and allows us to request data via the same 3 methods described earlier (see section 3.1). The *DatabaseHelper* helps us build the SQLite database and interacting with it. When we create the app for the first time we are generating both tables.

We are using a cursor to go trough all the data set and transforming each row into a MatchModel to be used by the app later. The cursor contain the query result made against the SQLite database. It allows an app to read the columns that were returned from the query or iterate over the rows of the result set.

The *DatabaseHelper* can also add Match and Quarters in the SQLite database. To accomplish that we create a *ContentValue* object and insert sets on key value pair to fill the tables.

3.2 Interface

The application interface can be compared with a one page application where we are navigating between the corresponding fragments via the navigation drawer (see figure 1.3.2 and figure 1.3.3).

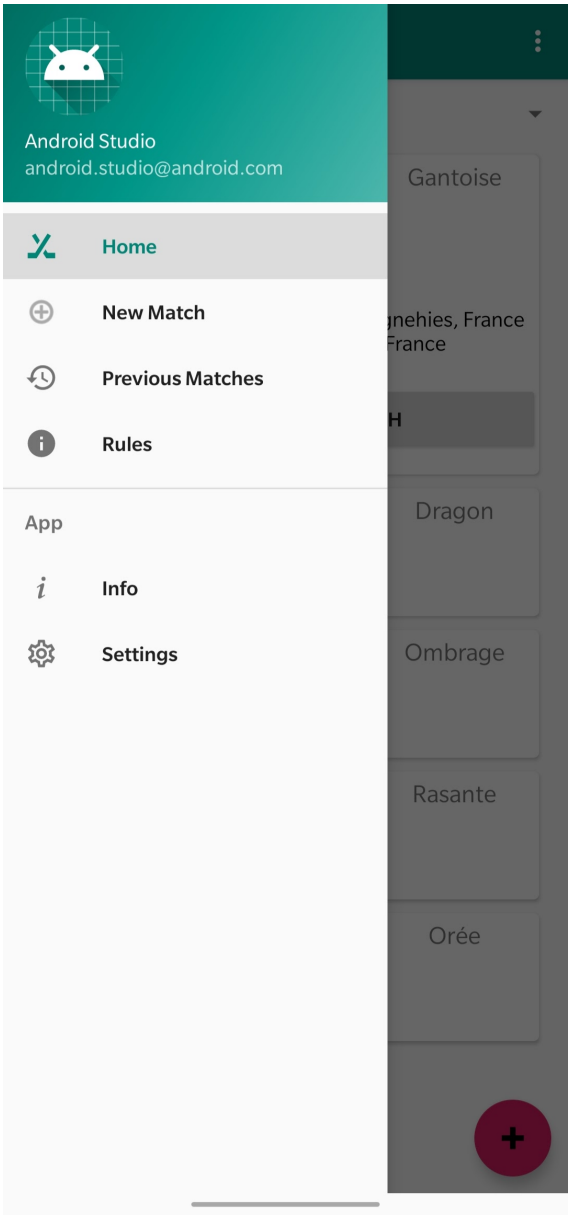

FIGURE 1.3.2 – Navigation drawer

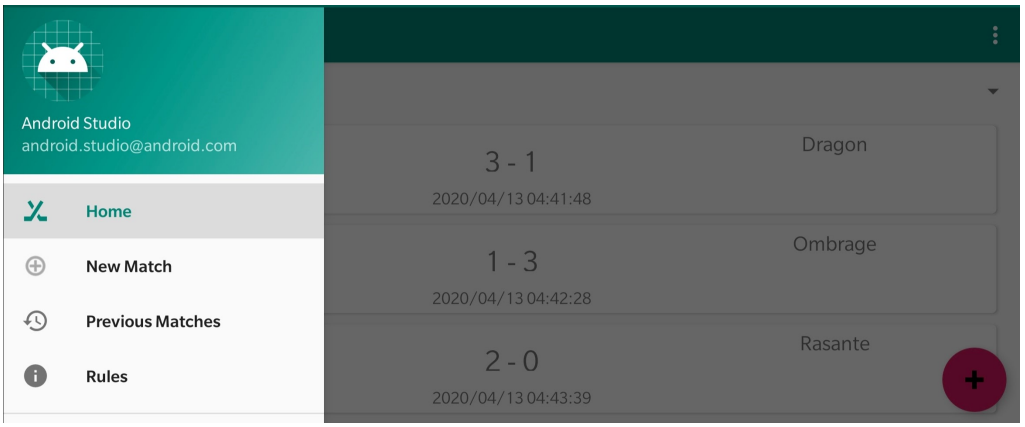

FIGURE 1.3.3 – Navigation drawer horizontal view

homepage

The front page is composed of a recycler view which enables us to display the 5 last recorded matches stored in the SQLite database (see figure 1.3.4) (see section 3.1).

The recycler view is composed of cards (see figure 1.3.4), each card is clickable and expands (see figure 1.3.7 to show more detail about the match and a button to navigate the detailed match page (see figure 1.3.14). When you navigate to a match, a new activity is started. This interface is displaying the full recorded match with all the information and statistics about it.

The recycler view uses an adaptater, the *HomePageMatchAdapter*, to pass on the data given as an array to the *viewholder* it is connected to. The data from this array is downloaded from the database SQLite. In this case the array is an array object Match and contains all the matches downloaded fetched from the database. This adapter is added to the recyclerview and will automatically generate the needed viewgroups on the interface.

We have decided to use a recycler view as it optimises the UI because it generate only a set a views. What we mean by that is that after generating enough view groups, it will only recycle the data. When the user scrolls down for example, one item that was visible is not visible anymore. The adapter will then use this viewgroup again for the new item that appeared after scrolling down. It will then only replace the data inside it. This recycler view is then also very practical to display data dynamically from a database.

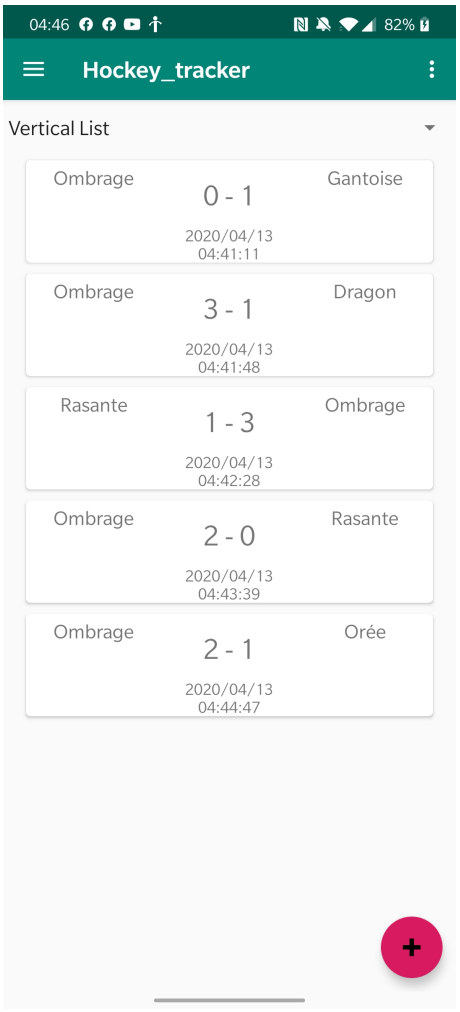

FIGURE 1.3.4 – Home Page

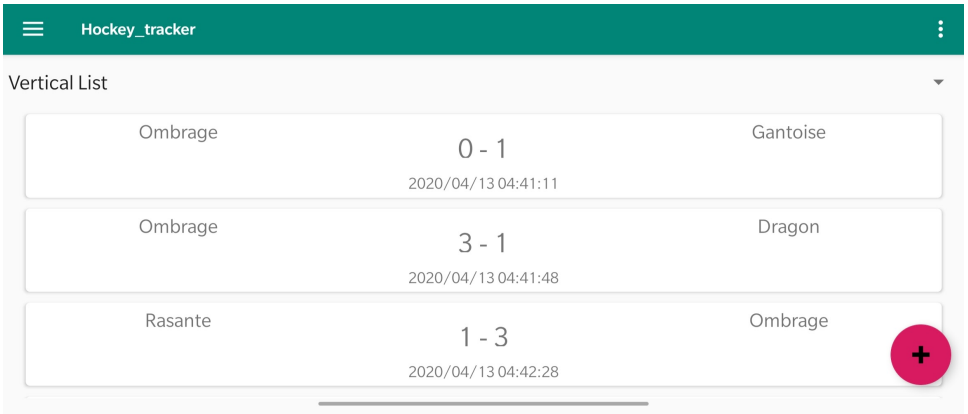

FIGURE 1.3.5 – Home Page horizontal view

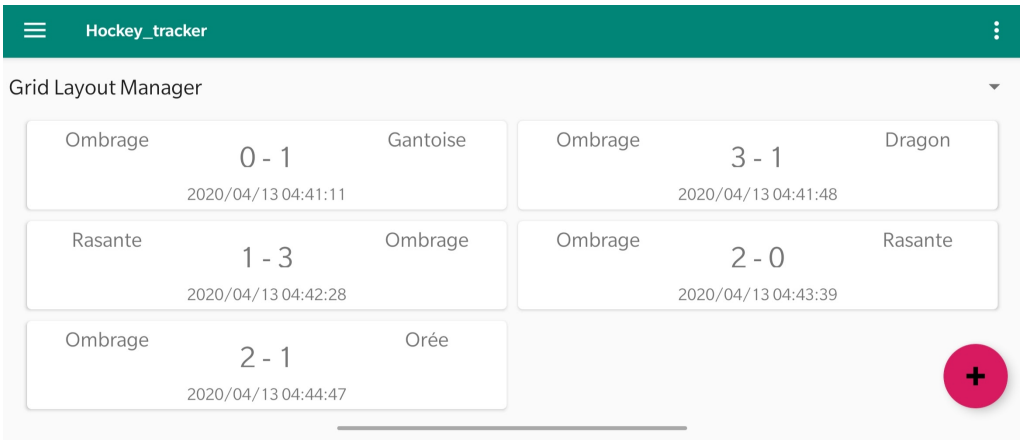

FIGURE 1.3.6 – Home Page horizontal view, gridlayout

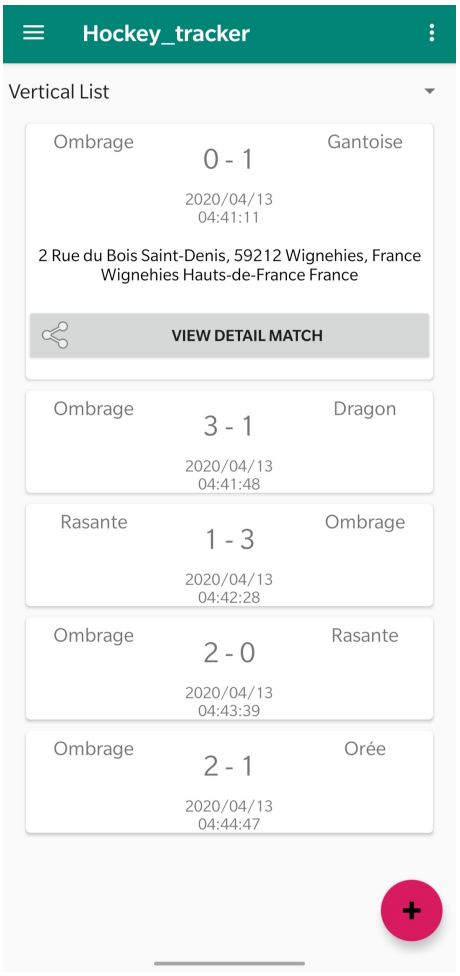

FIGURE 1.3.7 – Home Page expanded card

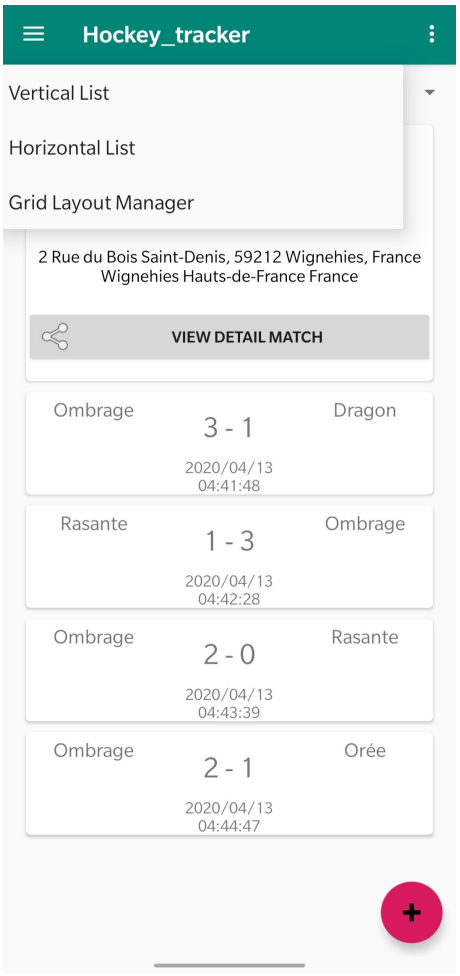

FIGURE 1.3.8 – Home Page spinner

The *viewholder* contains all the references and other items of view group that the adapter needs to fill with the data passed on to it. As said above, the viewholder is in this case a cardview that can be expanded. When the card is not expanded it displays the teams names, the score, the date and the time of the match (see figure 1.3.4), and when expanded it displays the location and a button to access the full match details (see figure 1.3.14).

previous match

The previous match page works in the same way as the *homepage* but the passed on array is different and contains all the matches of the SQL database (see figure 1.3.9). To pass the data from the *DownloadModel* to the fragment trough a string extra when the fragment is loaded. The the adapter is dealing with the transformation of the data into an array and link the array to the corresponding viewholder.

FIGURE 1.3.9 – Previous match page

new match

To record a new match, the user can either click on the plus button on the bottom right of the homepage or previous match page. Or he can navigate to it via the navigation drawer and click on the new match menu item. This opens a new fragment to start recording the match, the user needs to enter the 2 team's names. The location of the user is taken as the current position of the user, the location will be display on the map (see figure 1.3.10). If the user uses the app for the first time, a popup window will appear requesting permission to use the location. When the user is done, he can click on the V icon on the top right to go to the next stage of the recording.

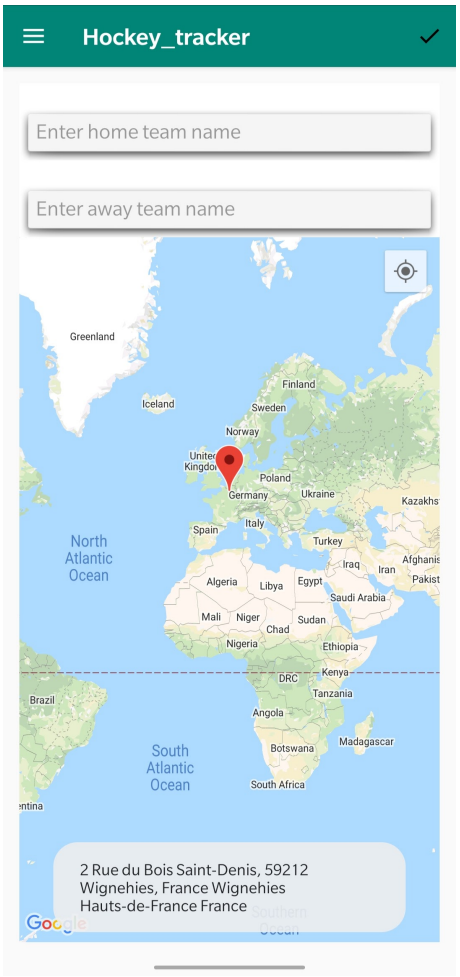

FIGURE 1.3.10 – New match page

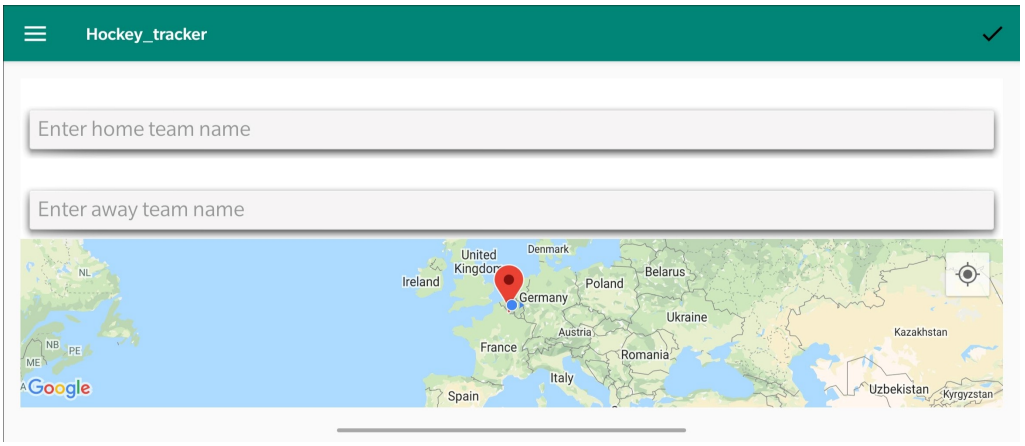

FIGURE 1.3.11 – New match page horizontal view

After creating the new match, the user will be send to a new activity, the *RecordActivity* (see figure 1.3.12). On this page the user is able to record any action that happened on the field. He uses specific buttons to add the different actions happening during the match. As a hockey match is made out of 4 quarters, 2 navigation buttons are available to switch between the different quarters. For each team a set of buttons is made available, each opening a popup window when clicked on. They are 5 different buttons that each have their own popup window.

We have the goal button (see figure 1.3.13 (a) for popup), the pc button (see figure 1.3.13 (b) for popup), the shot button (see figure 1.3.13 (c) for popup), the stroke button (see figure 1.3.13 (d) for popup), the outside button (see figure 1.3.13 (e) for popup), the fault button (see figure 1.3.13 (f) for popup)

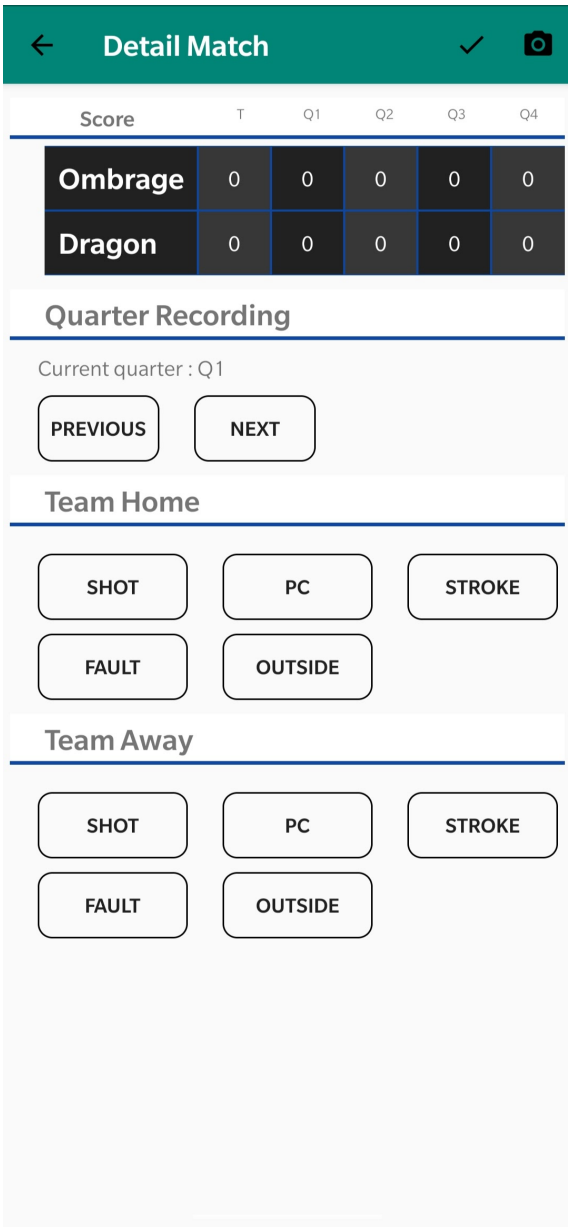

FIGURE 1.3.12 – record match page

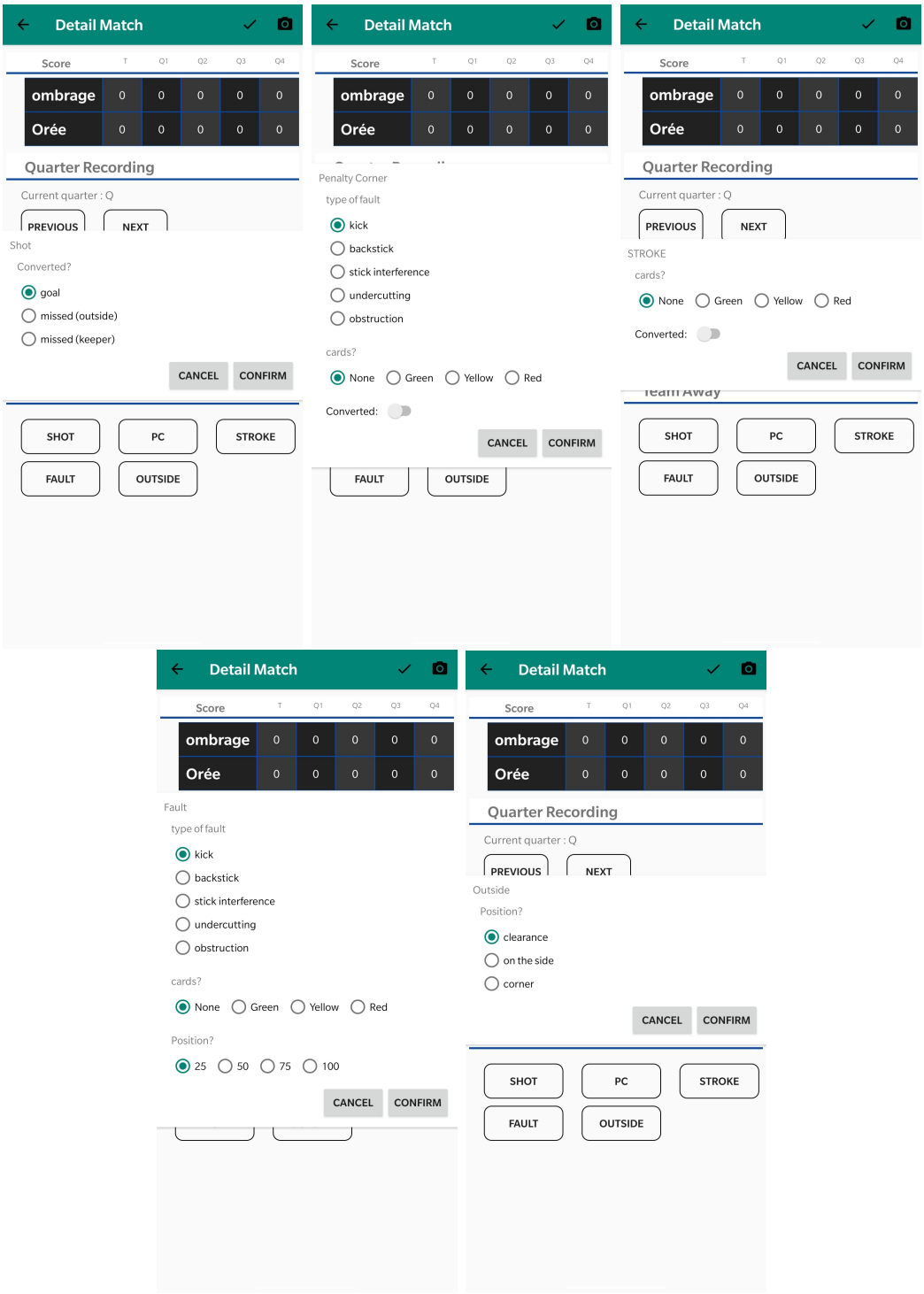

FIGURE 1.3.13 – (a) shot popup – (b) PC popup – (c) stroke popup – (d) fault popup

– (e) outside popup

The user has also the possibility to take pictures during the match. the application will store them locally and reference the file path inside the database.

When finished, the user has to press the confirmation button on the top right of the application to save the current match. If everything goes well, the user should be redirected to the home page and a toast should appear confirming that the match has been added. If not a toast appears saying there was problem during the process.

detail match

When the user clicks on the *View detail match* button in cardviews explained beforehand, a new activity, called *DetailActivity*, will be started. This activity is separated in 3 parts : The resume of the match, the gallery of the images taken during the match and finally a last part containing info about the match (not very useful for the moment as it only contains the date and location of the match.

The different pages of this activity can be accessed through a bottom navigation view that will load a different fragment depending on the button clicked. Before going into details about the different pages, the resume page will load a different fragment depending on the fragment the detailed activity was started from. As the home page uses a different database than the previous match page, the fetching method is different.

When using the internal database, the data from the database is directly downloaded from the fragment *onCreateView* method via the database helper. However when using the external database, the data is downloaded from the detail activity via the asynctask and passed on as a string object to the fragment when started. Beside this, these fragment are practically the same and use the same layouts.

The resume of the match is composed of a table displaying the score of the match and a recycler view displaying all the statistic about the match such as faults, shots, cards and other events that have been recorded during the match (see figure 1.3.14). The recycler view is composed of different part : the overall match statistic and quarter's statistics. You navigate between the different part by sliding the table on the left or on the right (see figure 1.3.15). To create each table we created a *DetailMatchAdapter* which will fill the view holder based on the data array passed on.

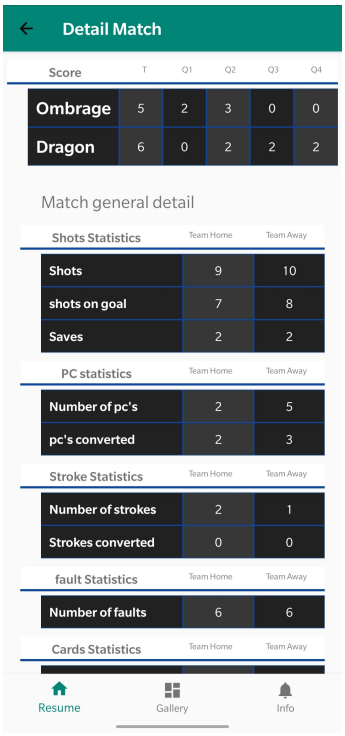

FIGURE 1.3.14 – Match Score

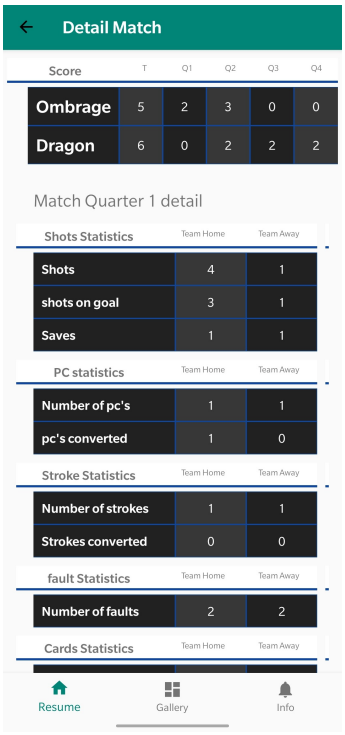

FIGURE 1.3.15 – Quarter detail

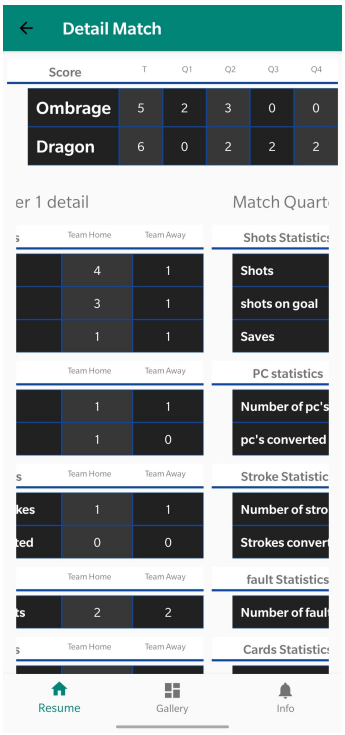

FIGURE 1.3.16 – Quarter detail- scrolling right/left

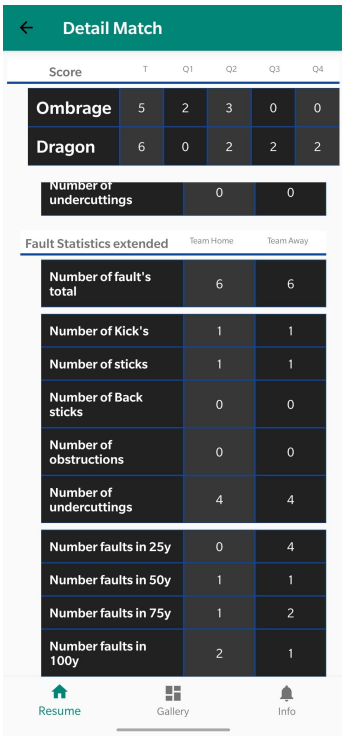

FIGURE 1.3.17 – Quarter detail- scrolling up/down

The imageview fragment is used to display all the pictures taken during the match (see figure 1.3.18). Again, we are using an adapter, *ImageRecyclerViewAdapter* to fill the recycle view it is connected to. This adapter will pass on the image path from the array is received to convert the image file it refers to into a bitmap which will then be displayed inside an *ImageView* component. Again this page includes a spinner that enables the user to choose between scrolling up/down or left/right.

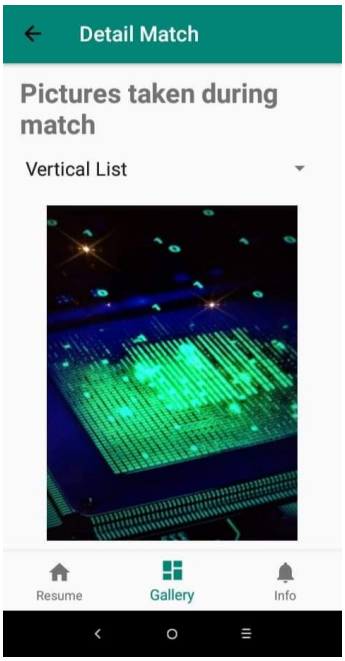

FIGURE 1.3.18 – ImageView Page

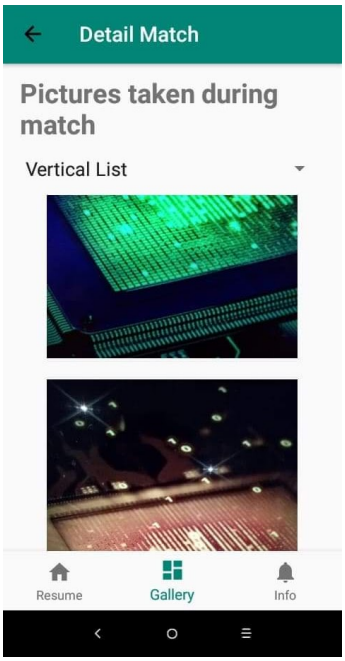

FIGURE 1.3.19 – ImageView Page - scroll down/up

The match info fragment contains information as the date and location of the match and could in the future contain information as the players names of each team.

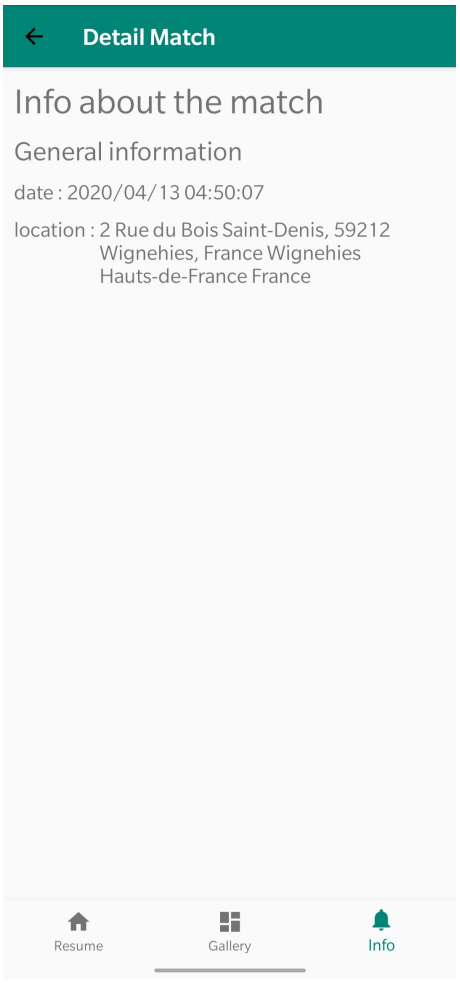

FIGURE 1.3.20 – Match Info

Others

As the application can be in English or French we added a *Settings* page. This page is build as a fragment and allows the user to switch the language of the app (see figure 1.3.21 and 1.3.22). The language switch is base around a switch of the *String.xml*, we have translate all the app inside an second *string.xml* and we select the corresponding XML.

We define the application language via the *LocalManager* which helps us switch between French and English. Inside the *LocalManager* class we define several methods, one of them is used to update the resource of the context's application after changing the language of the local. To change the language you do not directly switch the XML file, you are changing the local's language code of the app (fr : French, en : English). After updating the locals the application is dealing with file switch by itself.

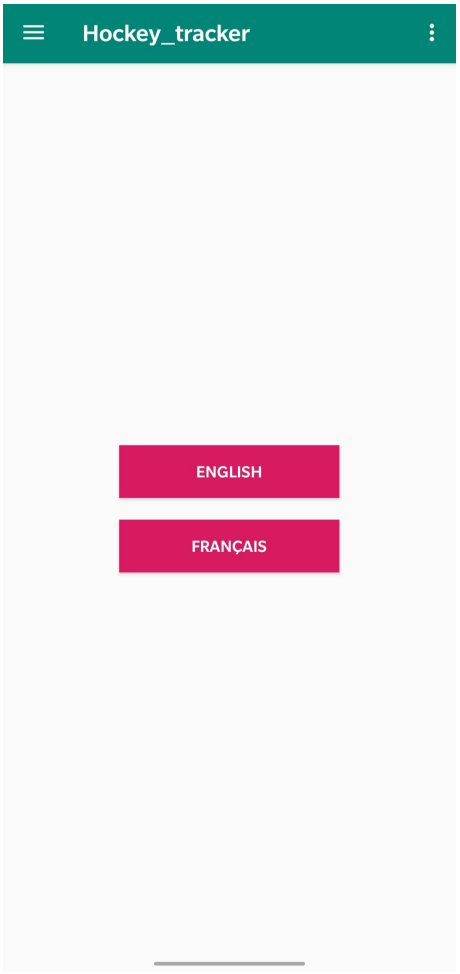

FIGURE 1.3.21 – Settings page

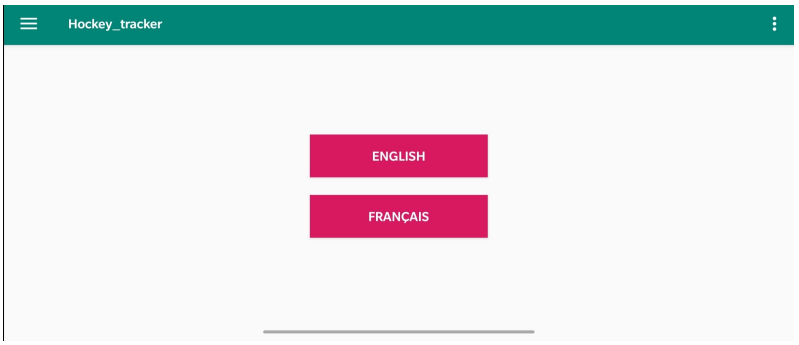

FIGURE 1.3.22 – Settings page

We also included a page explaining the rules of grass field hockey (see figure 1.3.23 and 1.3.24) and a page with more information about the app (see figure 1.3.25 and 1.3.26).

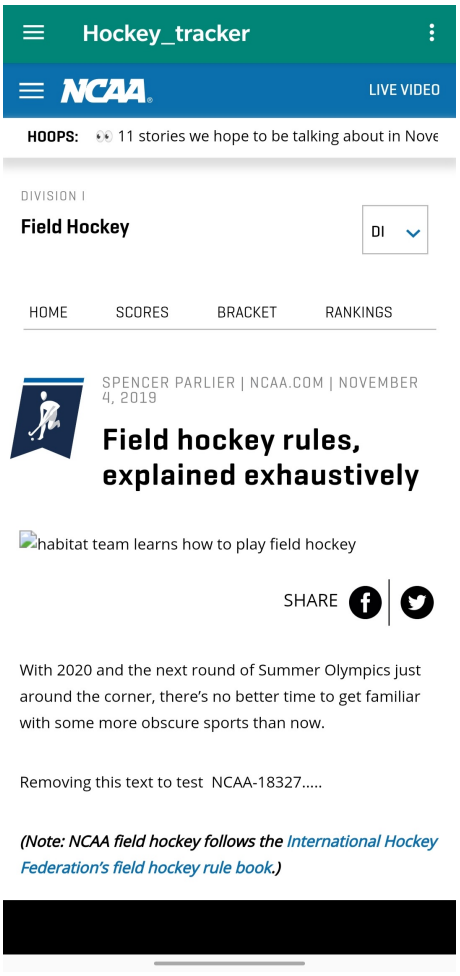

FIGURE 1.3.23 – Rules page

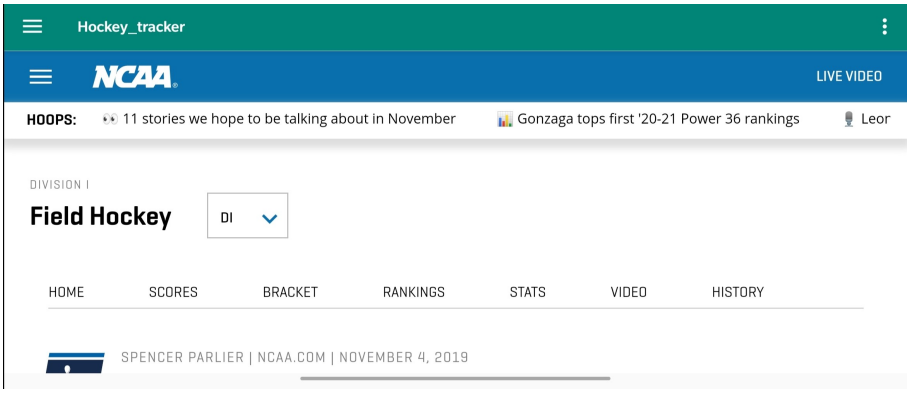

FIGURE 1.3.24 – Rules page

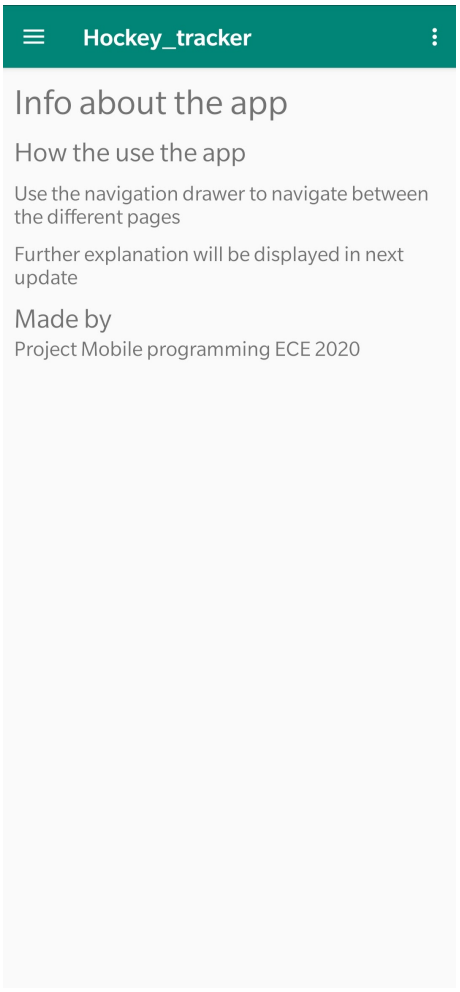

FIGURE 1.3.25 – Info page

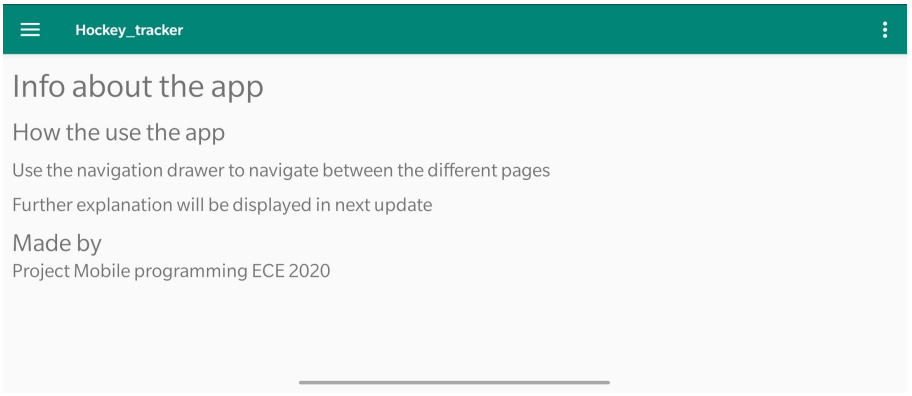

FIGURE 1.3.26 – Info page

3.3 File Organisation

MainActivity

- Java : Mainactivity.java (extends BaseActivity.java for language feature)
- xml :
  - Layout : activity_main.xml, app_bar_main.xml
  - Menu : main.xml
- Drawer
  - Layout : nav_header_main.xml
  - Menu : activity_main_drawer.xml
  - Drawables : icons menu
- Fragments
  - HomePage.java
    - Java : HomeFragment.java
      - Database : DatabaseHelper.java, MatchModel.java
      - Adapter : HomePageMatchAdapter.java
    - Xml :
      - Layout : fragment_home.xml, layout_match_view_model.xml,
      - Spinner : spinner_list.xml
  - New Match
    - Java : newMatchFragment.java
      - Database : DownloadModel.java, MainActivity.java, MatchModel.java
    - Xml :
      - Layout : fragment_new_match.xml
      - Menu : main.xml
      - Values : google_maps_api.xml
  - Previous match
    - Java : matchFragment.java
      - Database : DownloadModel.java, MainActivity.java, MatchModel.java
      - Adapter : HomePageMatchAdapter.java
    - Xml :
      - Layout : fragment_match_list.xml, layout_match_view_model.xml
      - Spinner : spinner_list.xml

- Rules
  - Java : RulesFragment.java
  - Xml :
    - Layout : fragment_rules.xml
- Info
  - Java : InfoFragment.java
  - Xml :
    - Layout : fragment_info.xml
- Settings
  - Java : settingsFragment.java
    - Preferences : Baseactivity.java, LocaleManager.java
  - Xml :
    - Layout : fragment_settings.xml
    - Values : strings.xml, strings-fr.xml

RecordActivity

- Java : RecordActivity.java (extends BaseActivity for language feature)
- XML :
  - layout
    - MainLayout : actitivity_record.xml
    - Sublayout : table_layout.xml, table_buttons_quarters.xml, table_buttons_team_a.xml, table_buttons_team_b.xml
    - Layout_popup_windows : layout_shot.xml, layout_pc.xml, layout_stroke.xml, layout_fault.xml, layout_outside.xml
    - Menu : main
  - Xml : file_path.xml (path of image file storage)
  - Drawable : design buttons and table

DetailActivity

- Java : DetailActivity.java (extends BaseActivity.java for language feature)
- Xml :
  - Layout : activity_record.xml
  - Bottom_menu : navigation.xml
- Fragments :
  - Detail match page
    - Java : DetailFragment.java
      - Database : QuarterModel
        Internal : databaseHelper
        External : DownloadModel
      - Adapter : DetailMatchAdapter
    - XML :
      - Layout : fragment_detailfragment.xml
      - Sublayout : table_layout.xml
      - recyclerView : statistics_detail_xml
        sub : statistics.shots.xml, statistics_pc.xml, statistics_stroke.xml, statistics_cards.xml, statistics_fault.xml, statistics_outside.xml, statistics_fault_detail.xml, statistics_pc_fault_and_cards.xml
  - Gallery page
    - Java : ImageViewFragment.java
      - Database : MatchModel
      - Adapter : ImageRecyclerViewAdapter
    - Xml :
      - Layout : fragment_item_list
      - recyclerView : fragment_item
  - Info page
    - Java : InfoMatchFragment.java
    - Xml :
      - Layout : fragment_info_match.xml

Annexe

1 Code

https://github.com/snihar2/hockey$_t racker$