Code Documentation

7th June 2016

Introduction

This documentation lists all relevant methods used in the Football App. All classes, except for GUI classes, are listed below. The corresponding methods had been described by the developer who implemented them. You can see who implemented which method by having a look at the description or directly at the code. Unrelevant methods are not described here.

Table of Contents

1 ControllerMatch

1.1 `protected String doInBackground(Object… command)`

POST - Wutti

This method gets a list of commands and merges them together to form a query string which is then sent to the webservice. Then the webservices adds a match to the database and returns the added match. The added match is then returned to the calling method.

PUT – Lagger

The method comes in action when a match is updated. The match that ought to be updated is passed as an actual Match object and then parsed to a json format. The match is converted into a byte-array and then passed to an output stream. If everything worked without any problems, a 200 response code is returned.

2 ControllerParticipation

2.1 `protected String doInBackground(Object… command)`

POST and PUT - Wutti

This method gets a list of commands and merges them together according to the the functions selected (POST OR PUT).in order to form a query string which is then sent to the webservice. The wevservice treats the query accordingly, adding a Participation (POST) or updaten a Participation (UPDATE)

2.2 `private ArrayList<Participation> prepeareParticipations(Match m)`

Wutti

This method takes a Match as input and returns an ArrayList of type Participation. This function gets the data from the Match and converts it into Participations such that the query strings can be created correctly.

3 EditTeamActivity

3.1 `private void initOthers()`

Wutti

This method lets the database load the players from the Database.

3.2 `private void initViews()`

Wutti

This method finds all the views by ID which will be needed by this class. Furthermore, it registers an OnClickListener

3.3 `private void doTableStuff()`

Wutti

This method deals with the TableLayout. It dynamically creates all the controls for the table for each user.

3.4 `private void setTeamsAccordingToUsersChoice(Player p, TeamEnum team)`

Wutti

This method decides where to add and remove the player from Team A or B

4 SQLDateHelper

4.1 `public static Date getDate(String dateString, String format) throws Exception`

Wutti

This method converts a String to a java.sql.Date with the corresponding format.

4.2 `public static Date getDate(String date) throws Exception`

Wutti

This method converts the string into a java.sql.Date with the date format :“dd-mm-yyyy”

4.3 `public static String dateToString(Date date, String format) throws Exception`

Wutti

This method converts a java.sql.Date into a String with the corresponding format.

4.4 `public static String dateToString(Date date) throws Exception`

Wutti

This method converts a java.sql.Date into a String with the date format: “dd-mm-yyyy”

5 PlayerRadioButton

Wutti

This class extends the RadioButton. It is needed to remember which players Radio Button has been clicked (Team choice)

6 UpdateMatchActivity

6.1 `private void getViews()`

Wutti

This method finds all Views by Id which will be needed by this class.

6.2 `private void prepare()`

Wutti

This method is needed because this GUI is getting recycled. And for the adding we need some changes to the GUI (Some view elements are gone etc.)

6.3 `public void onBtnSave(View view)`

isAdd – Wutti

This class deals with the date picker. Furthermore it prepears all the data and tells the Database to add it to the Webservice.

6.4 `private void displayDateIsInTheFutureError(Date dateWhichIsInTheFuture)`

Wutti

This class deals with the date picker. Furthermore, it prepares all the data and tells the Database to add it to the Webservice.

6.5 `private void displayDateIsInTheFutureError(Date dateWhichIsInTheFuture)`

Wutti

This method displays a Toast when the date which was selected is in the future.

7 Database

7.1 `public void addMatch(Match match) throws Exception`

Wutti

This method sets the Parameter for the MatchController such the Controller can insert it into the DB.

7.2 `public void addOrUpdateParticipation(Match m, boolean FirstTime)`

Wutti

This method adds or update. This is decided by the FirstTime parameter if it is true the participation will be added if it is false the participation will be updated .

7.3 `public void updateMatch(Match m) throws Exception`

Lagger

The match which ought to be updated is passed. If the match controller returns 200 , the match is removed and then added. This might appear odd, but since only the values change and the id (which is compared in the equals() method of every match) evidently remains the same, it is a way of easily updating the match in the local collection. The local collection is later needed for the offline mode of the app.

7.4 `public void addPlayer(Player player)throws Exception`

Lagger

Works similar to the updateMatch. In general, each method that communicates with a controller has the same flow. The new player is immediately added to the local collection.

7.5 `public boolean authUser(Player p)throws Exception`

Lagger

Is used to verify whether a user entered a correct username and password. If the id of the returned Player object equals -1, the inserted credentials are not valid.

7.6 pubblic void removePlayer(int id, String name) throws Exception

Weiler

Instantiates a new ControllerPlayer and uses it to perform a DELETE with the delivered player-id. It also removes the specific player from the local ArrayList “allPlayers”. It throws an exception if the webservice does not return the code “200”.

7.7 public void addOccupation(int playerId, String positionName)

Weiler

Instantiates a new ControllerOccupation and uses it to perform a POST with the delivered player-id and position-name (occupation). It throws an exception if the webservice does not return the code “200”.

7.8 public void removeOccupation(int playerId, String positionName)

Weiler

Instantiates a new ControllerOccupation and uses it to perform a DELETE with the delivered player-id and position-name. It throws an exception if the webservice does not return the code “200”.

7.9 public void loadAllPlayers() throws Exception

Weiler

Instantiates a new ControllerPlayer and uses it to perform a GET. It adds all players it has got from the webservice to the local ArrayList “allPlayers” in a new thread. It throws an exception if the webservice returns “null”.

7.10 public void loadAllOccupations() throws Exception

Weiler

Calls the “loadOccupations” method for every item of the local ArrayList “allPlayers”.

7.11 private void loadOccupations(Player player) throws Exception

Weiler

Instantiates a new ControllerOccupation and uses it to perform a GET for the delivered player. It calls the “addPosition” method of the delivered player for every occupation it has got from the webservice with the particular position-name.

8 ControllerPlayer

8.1 protected String doInBackground(Object... command)

Lagger and Weiler

The two possible methods that work with POST are the functionality to add a new player and to verify whether the passed player exists or not (Login).

If a GET-request was sent, all players will be returned as a json string.

DELETE: Defines the necessary URL with the player-id as query parameter and sends a request with the DELETE-method to the webservice. As return value you get the response code of the webservice.

GET: Defines the necessary URL and sends a request with the GET-method to the webservice. As return value you get a string in JSON-format which contains all players of the database.

9 AddPlayerActivity

9.1 `public void onBtnConfirm(View view)`

Lagger

The entered data is validated and an appropriate error message is shown in a toast.

10 ControllerOccupation

10.1 `protected String doInBackground(Object... command)`

Weiler

POST: Defines the necessary URL and sends a request with the POST-method and the delivered occupation object as content to the webservice. As return value you get the response code of the webservice.

DELETE: Defines the necessary URL with the player-id and the position-name as query parameters and sends a request with the DELETE-method to the webservice. As return value you get the response code of the webservice.

GET: Defines the necessary URL with the player-id as path parameter and sends a request with the GET-method to the webservice. As return value you get a string in JSON-format which contains all occupations concerning the delivered player.

11 Player

11.1 `public void addPosition(String positionName, boolean calledFromGet)`

Weiler

Checks whether it has been called from the get method or not (calledFromGet-parameter), if it has not been called from get and if it's not already contained in the ArrayList "positions" it calls the "addOccupation" method of the database to deliver it to the webservice (this is used by the ProfileActivity to add occupations to a specific player).

If it has been called from get and is not already contained in "positions" it adds the delivered position to "positions".

11.2 `public void removePosition(String positionName) throws Exception`

Weiler

After checking whether the delivered position already exists for this player it calls the "removeOccupation" method of the database and removes the delivered position from the ArrayList "positions".

11.3 `public void resetPositions() throws Exception`

Weiler

Calls "removePosition" for all four possible positions.

I2 ProfileActivity

I2.2 `protected void onCreate(Bundle savedInstanceState)`

Weiler

Loads the player of whom it has to show the profile (if an admin is logged in: the player selected in the spinner of the MainActivity (name delivered as an extra of the intent), otherwise: currentPlayer from the database). Calls “getViews”. Checks the checkboxes if the position is used by the player.

I2.3 `private void getViews()`

Weiler

Gets all checkboxes with “findViewById”.

I2.4 `public void onBtnSave(View view)`

Weiler

Calls “addPosition” of the player if the specific checkbox is cheked and “removePosition” if it is not checked. At the end it closes the activity.

I2.5 `public void onBtnReset(View view)`

Weiler

Calls “resetPositions” of the player and unchecks all checkboxes. At the end it closes the activity.

I3 RemoveDialogActivity

I3.I `protected void onCreate(Bundle savedInstanceState)`

Weiler

Fills the textview with the “confirmation”-message.

I3.2 `public void onBtnYes(View view)`

Weiler

Calls the “removePlayer” method of the database with the delivered player name (as an extra of the intent) and the “handlePlayerRemoved” of the listener if any is registered. At the end it closes the activity.

I3.3 `public void onBtnNo(View view)`

Weiler

Closes the activity.

14 RemovePlayerActivity

14.1 `protected void onCreate(Bundle savedInstanceState)`

Weiler

Calls “setAdapterLvPlayer” and registers a “setOnItemClickListener” for the listview “lvPlayer” which opens the “RemovePlayerActivity” and delivers the id, name and goalDifference to it.

14.2 `private void setAdapterLvPlayer()`

Weiler

Fills the listview “lvPlayer” with the return value of “getAllPlayers” of the database.

14.3 `public void onResume()`

Weiler

Calls “setAdapterLvPlayer” to guarantee “lvPlayers” is up to date after the user removed a certain player.