**Mobile Devices Homework**

1. **Teamwork**

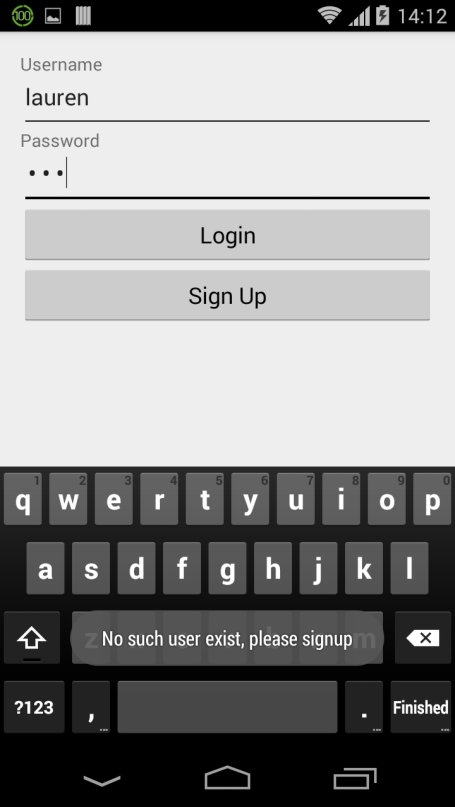
This homework was done by Codreanu Florin Laurentiu and Pirjol Adina-Florina, from group 1231E. The implementation of the application was done for Android mobile devices, with a targeted minimum OS of Android 4.1 JellyBean. We chose this Android build version because, as of Android Studio, around 78% of the users currently using the PlayStore have this or an above version. The project was uploaded on a Git repository for better teamwork and an efficient sharing of ideas.

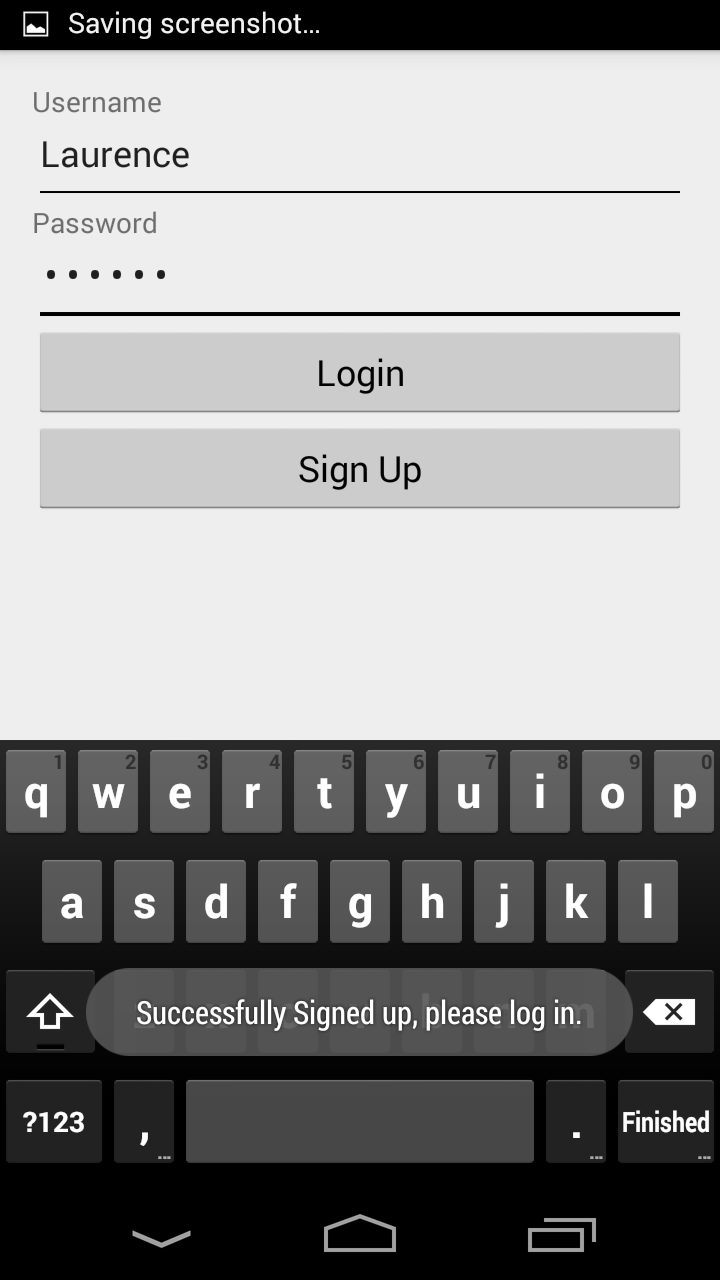
1. **Specifications**

* When the app is opened, there is a Login interface displayed
* User can Sign In/Sign Up in the application
* As an authenticated user, user can see his homepage
* User can see a weather widget and information regarding his interests on his homepage
* On his homepage, user can change his password
* On his homepage, user can choose/change his interests
* On his homepage, user can acces/ edit his Favorite links
* On the interests screen, user can change his interests
* Based on the chosen interests, they will appear on his homepage
* On the favorite links screen, user can edit his previously save links
* Based on the edited links, they will appear on his homepage

1. **Functions and implementation**

The user login was done using the Parse.com tool. Parse provides prowerful backend tools and services for developers. The Parse library was created and instantiated in the onCreate() method of the LoginActivity. In order to use Parse, we firstly needed to identify ourselves using the clientId and

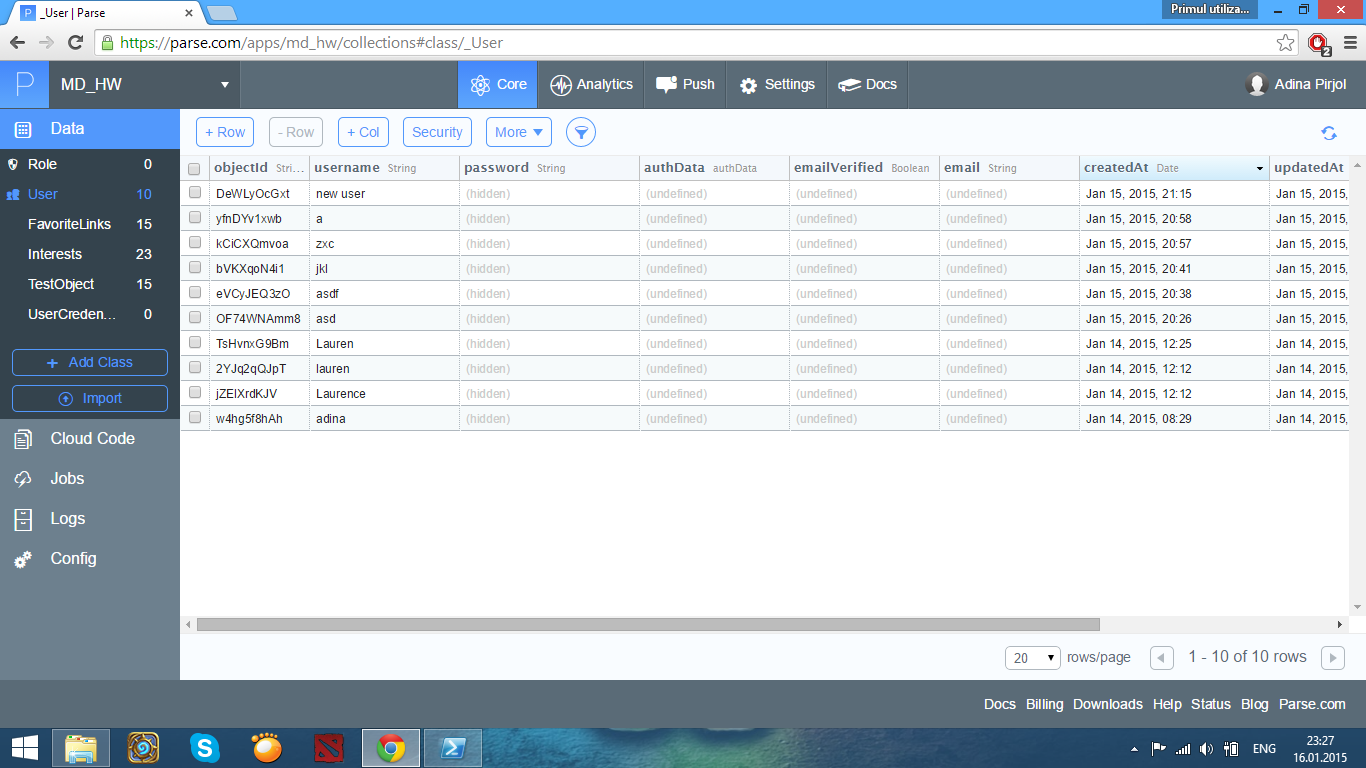


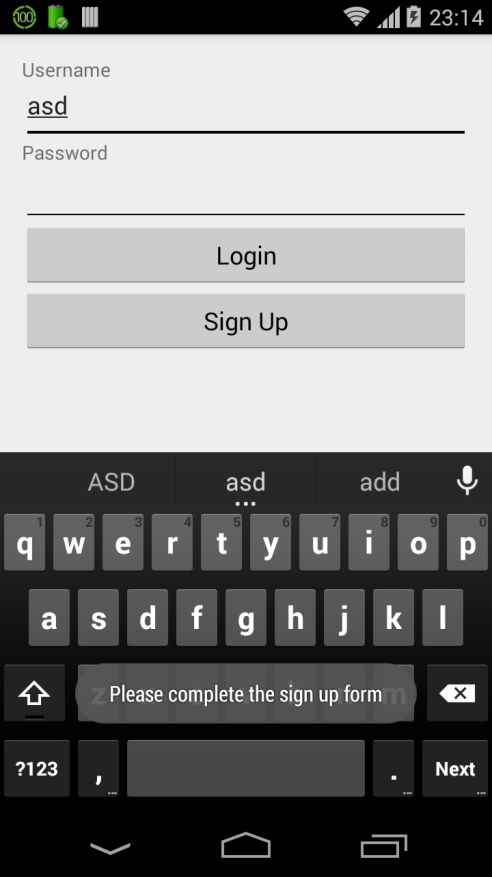
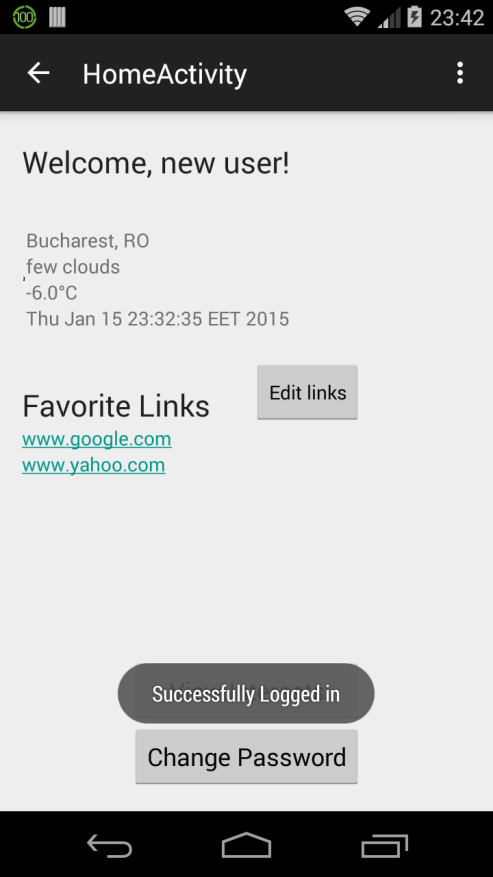


applicationId we received with the creation of our Parse account.

The login activity uses the Parse service. This way, the user credentials data for anyone that wants to login is stored safely online. This, however, requires constant internet connection; otherwise, data might be lost and, of course, the user cannot login. The first time, we used a “UserCredentials” class for storing the credentials on the cloud, which is similar with a table with two columns for username and password. Then, we decided that it is better to use objects of type ParseUser, already defined by the Parse library in order to create the login. We added validations for creating new users/extracting existing users. To make sure we don’t store corrupted data we put a validator on the signup form to check if either the user or the password are null.

Below we have a printscreen with the data stored in the database when a new user is created.



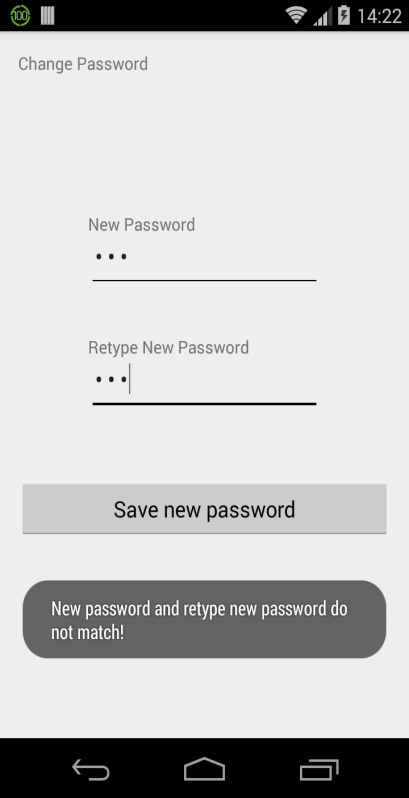
****

Once the user logs in successfully, he/she can see his homepage. In the top right corner, there is an welcome TextView of the format “Welcome, {currentUser}!” for ensuring and remembering to the user that he/she logged in with the right username.

Then, the is some weather information displayed, using the OpenWeatherMap api, available for free use online. All information regarding the api, weather parsing and weather info are in the package com.app.md\_hw.OpenWeatherMap. The connection to the weather api is done by establishing a http connection via a HttpURLConnection object in the com.app.md\_hw.OpenWeatherMap.OpenWeatherMapClient.getWeatherMessage, from where a JSON response is read regarding weather information in “Bucharest,RO”. The response received is converted into a JSONObject, the JSON is parsed accordingly and the needed information is extracted in the com.app.md\_hw.OpenWeatherMap.OpenWeatherMapClient.parseWeatherMessage() method, which is then stored in an object of type Weather. The Weather class stores information regarding the location, date, temperature, weather conditions and description from the place/time the homepage was accessed.

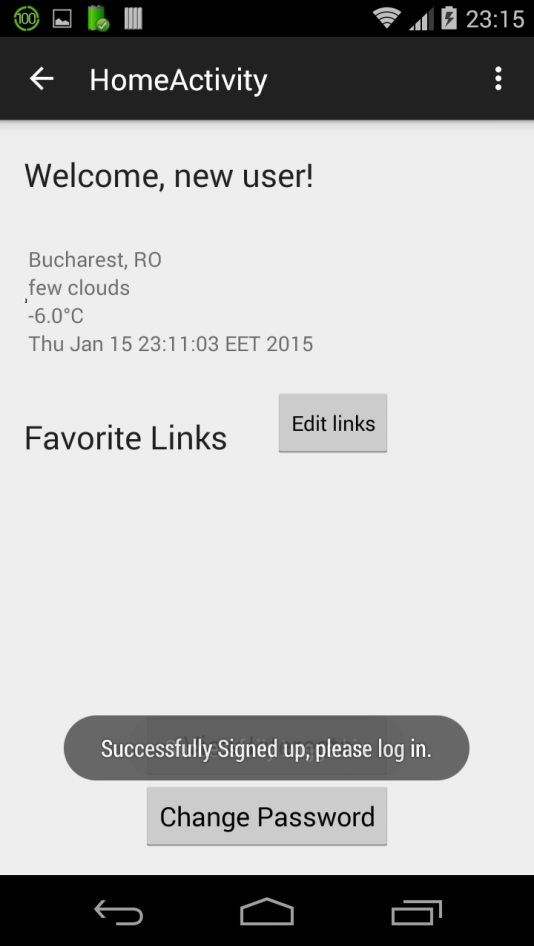
Back on the homepage, this information is processed and displayed using the local WeatherAsyncTask class, of type AsyncClass, that allows background operations to be performed and then displayed using the UI. Our class has three generic types, <String, Void, Weather>, where String is the return type of the data extracted from the local api using a location parameter, void is the progress result that does not affect the homepage layout, and Weather is the result returned by the onPostExecute overridden function, that sets the TextViews and the ImageView on the home\_activity layout

The user interface also provides a change password button that calls a new ChangePasswordActivity to store the new Password. This prompts you to a new layout where the user can change the password if it passes the validations. After changing the password the user is prompted back to the homepage.

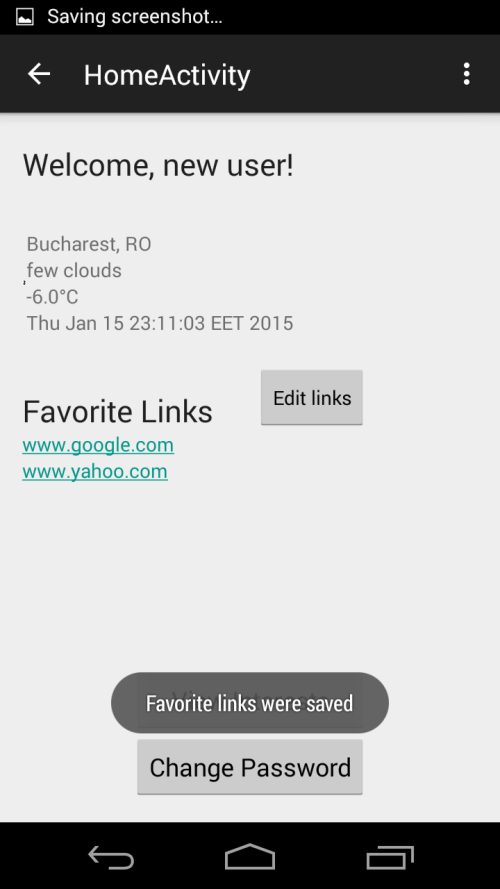


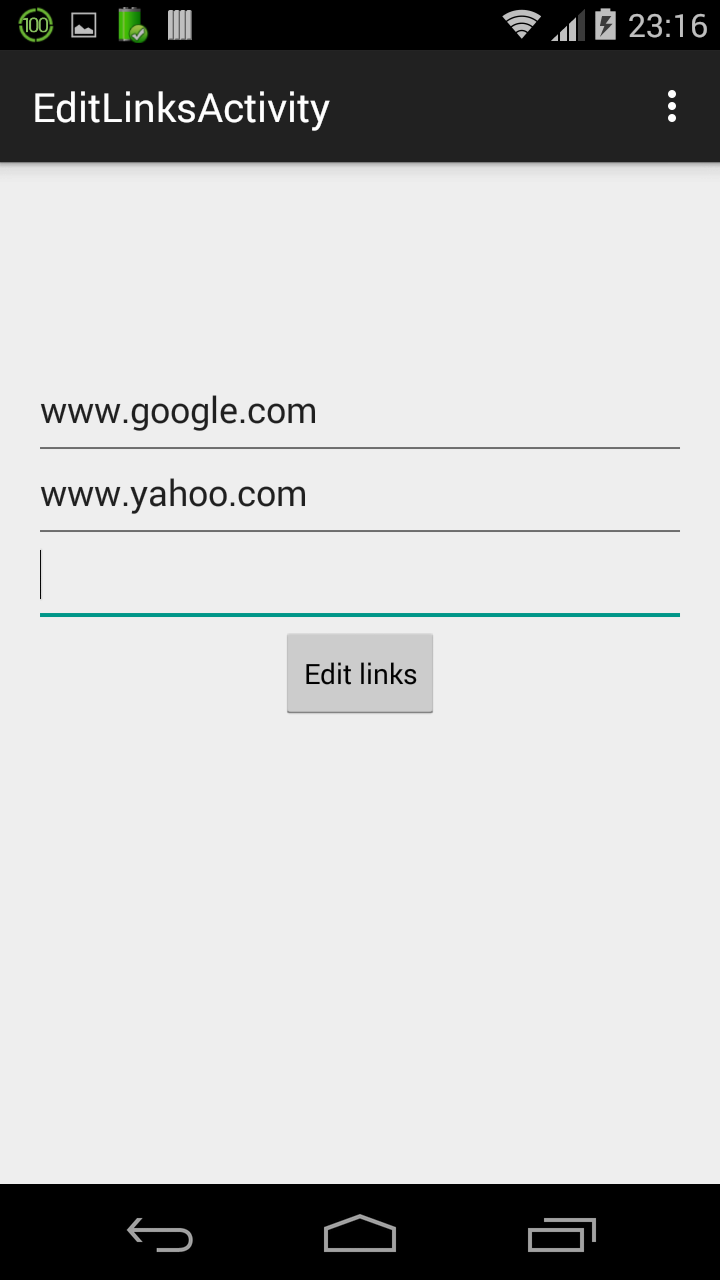
The layout consists of two EditTexts and two TextViews; one to type the new password and one to retype it. On top of that, there is a save new password button that has an onClick event to trigger the validation course. If the two passwords are null the user is prompted with a toast that says that the two passwords must not be null. Also the two passwords must match in order for them to be saved. If the passwords correspond to the regulation then the new password is set and the user can continue with his normal activity in the homepage.

Another implemented feature is FavoriteLinks section, where the user can edit 3 favorite links that he can access anytime he wants by a simple click on the URL. For the implementation we used a FavoriteLinks table where we stored the currentUser and the set of links he/she implements by querying the database. On the first iteration, the links section is empty and awaits the first iteration. On press of the Edit Links button, a new layout is opened. Here, we create a new pair of ParseObjects (username, favorite\_links) pair.



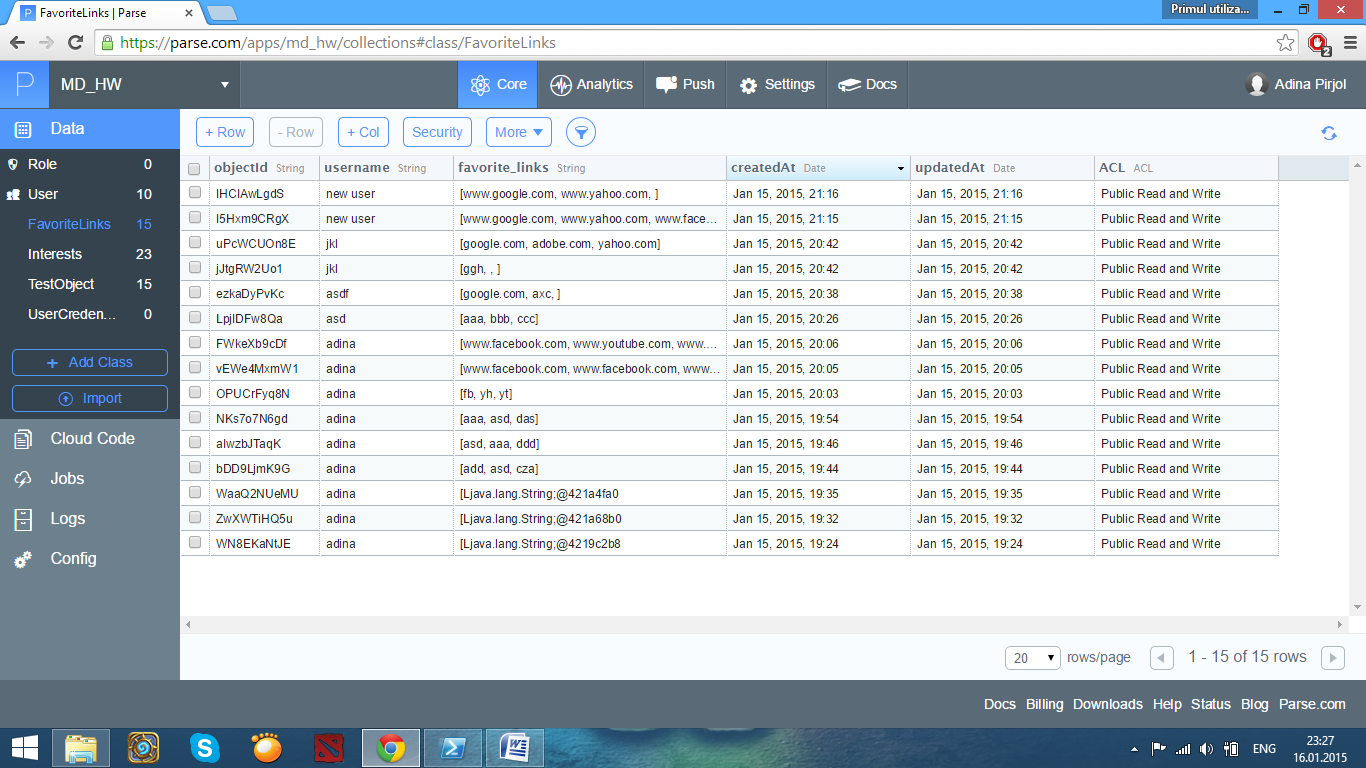
If there are any previous links, the corresponding editText is filled with the information gained through a ParseQuery object that retrieves a String with all the previous links. If the database was empty then we put in it through the query.put() objects in the database on the Save Links button click.

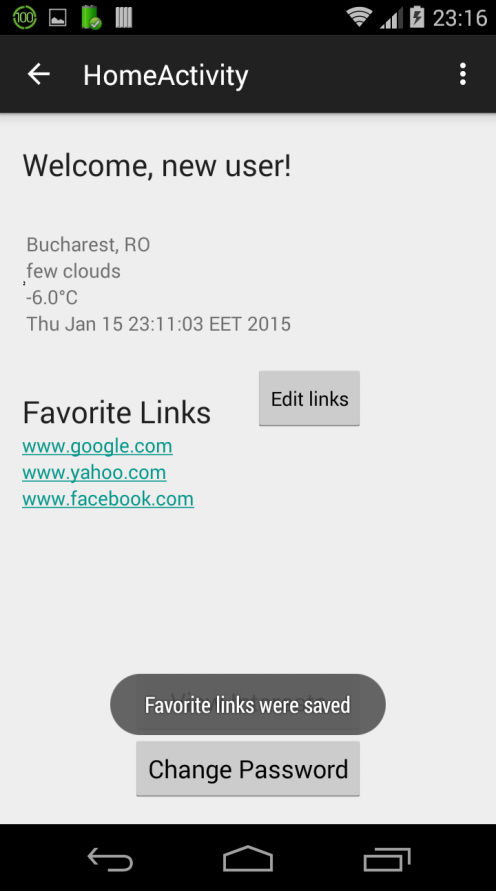


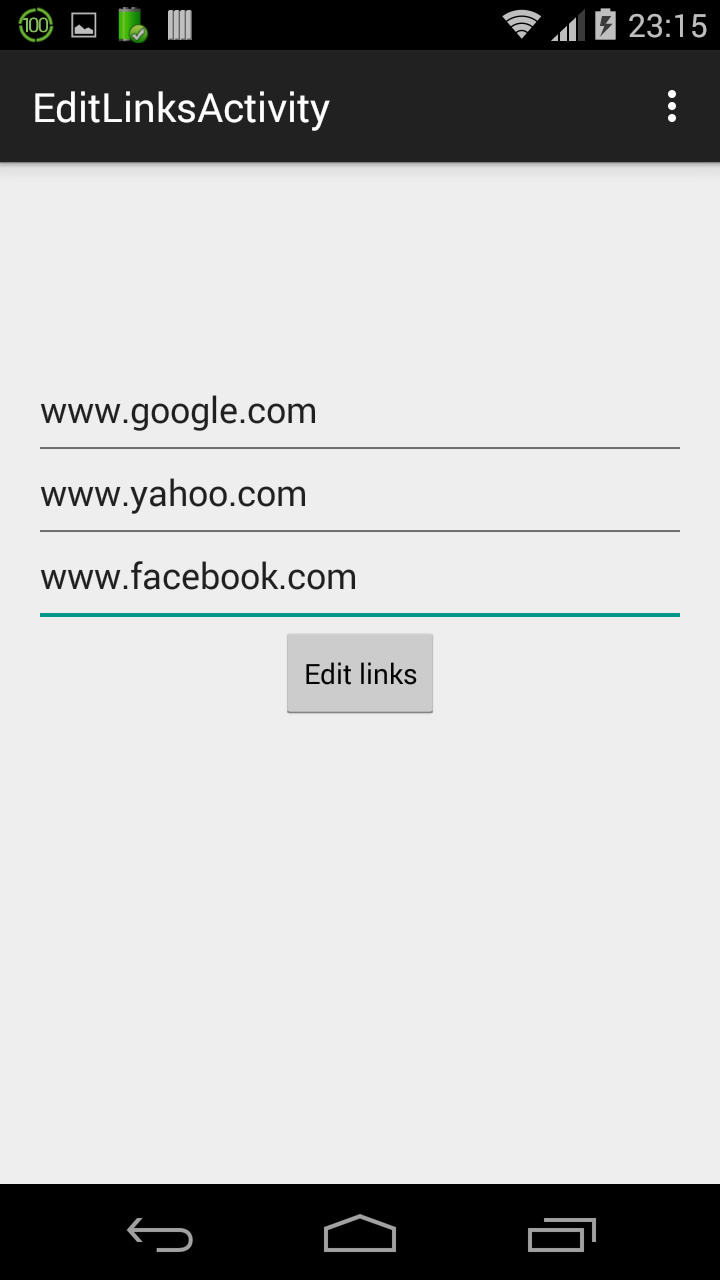


After that, we prompt the user with a toast informing him/her that the links were saved and return to the homepage to update the fields through a new query to check for the database elements. Here after the array has been parsed we create the ediTexts dynamically by setting their text as the text received from the database. Afterwards we use a Pattern object to parse a new url that will be accessed when the user clicks the link.

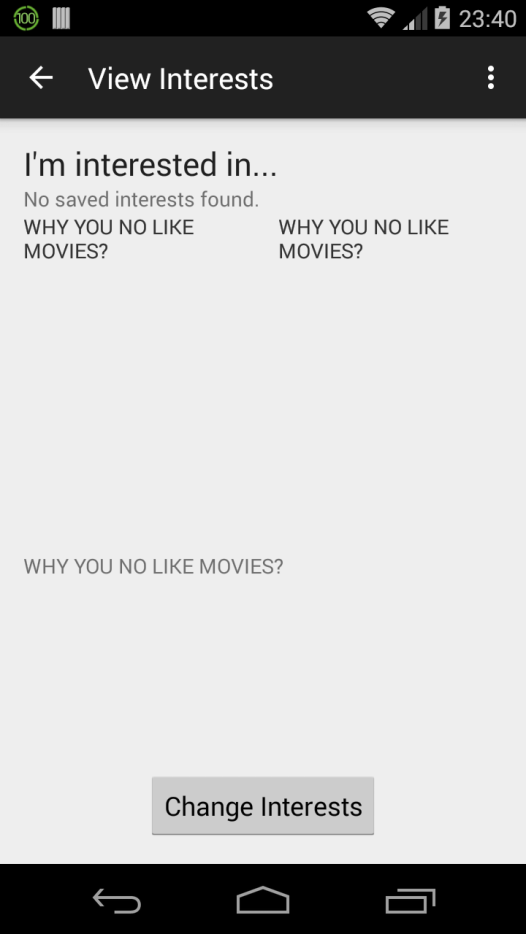
Below we have a printscreen with how the table was modified when we tried to insert various data in the edit link section.







Another feature on the homepage is the “INTERESTS” section. On the first iteration the I’m interested in button has an onClicked activity that prompts the user to an InterestsActivity where he/she can set a list of things he/ she is interested in and would like to see on his/her homepage.

****

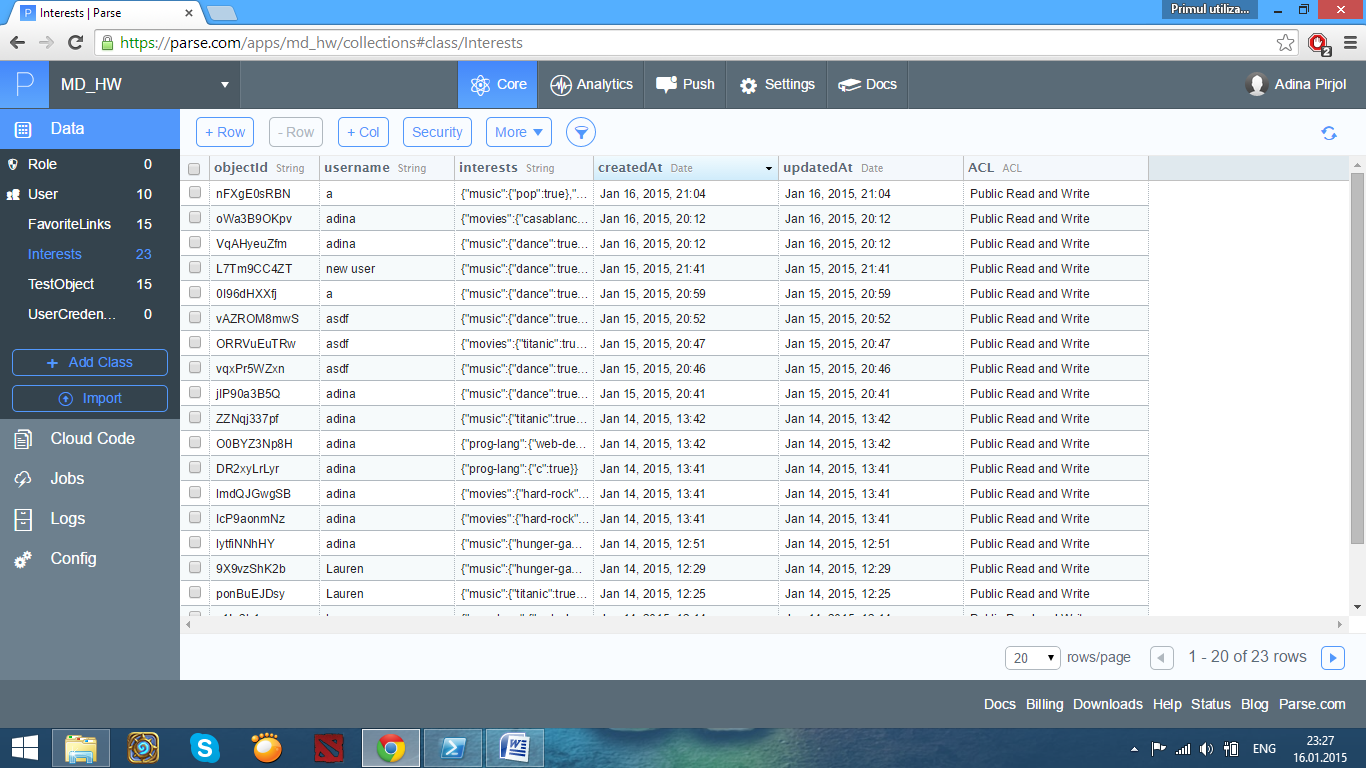
The interests layout consists of 3 fragments: MoviesFragment, MusicFragment and ProgrammingLanguagesFragment. Each of them has is connected to an individual layout and for each calss with a fragment there is a dynamical request to retrieve the users’ data from the database. These views offer a set of checkboxes with options such as: Titanic, Hunger Games, Casablanca or Hard Rock, R&B, Dance and What is that?, C/C++/C#/anything C, Pything/Perl/Ruby/PHP/Web/web/WEB, Java?Java.JAVA . The options the user selects are then sent via a JSON back to the Homepage using the “SEND INTERESTS” button.

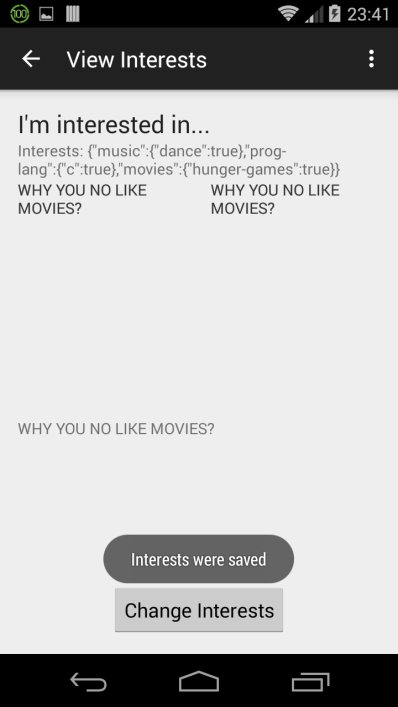
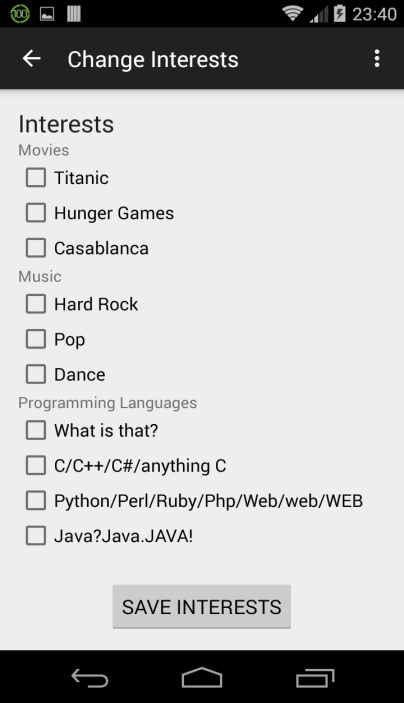
The ViewInterests layout is generated using a JSON stored internally in a text file named interests.txt in the assets folder. When the ChangeInterests activity is accessed, the layout is generated automatically based on the information found in that file. This way, the interest information can be changed easily and stored dynamically in different fragments of the screen.

After the layout is generated, the user can select his/her interests by using the CheckBoxes provided for each category. When the “Save Interests” button is pressed, the onClickButtonListener saves the choices the user made in the “Interests” table on Parse in the column interests of type String, and is redirected back to the Home activity.

If there were changes made in the interests section, the interests data is retrieved from the Parse table using a ParseQuery. The general pattern is to create a ParseQuery, put conditions on it, and then retrieve a List of matching ParseObjects using the findInBackground method with a FindCallback. For example, we retrieved the interests column, by using the whereEqualTo method to constrain the value for the username.

Below we have a printscreen with the data stored over time when we inserted new interests for several users.



****

When the SAVE INTERESTS button is clicked, the onSave method is called and it checks if either of the checkboxes is clicked. When one is clicked a new JSONObject of type X is created and it receives the required information. For instance if Hunger Games is clicked we find out because movies.2ischkec() will return true and we create a JSONObject movies where we put “Hunger Games”. If any change has made we create a ParseObject and for the current user we put in the “interest” column the new data.

On the homepage activity, there are also 2 more buttons, for changing the password and for selecting/changing the user’s interests.

There is no need for a save profile button as the information stored on the homepage is generated dynamically (for the weather info) or saved previously (in the Interests activity).

1. **Conclusions & perspectives**

We spend some time debugging several crashes of the homepage because we didn’t exactly understand the JSON response received from the external API. This led us to further improve some validation tests, like checking for the internet connection, and draw some awareness on throwing and catching exceptions in order to inform the user about what a possible cause of the problem might be. A conclusion would be that it can be tricky to process information received from third parties.

From the point of view of the tools that we had in order to develop the application, we had some difficulties with using either Eclipse/Android Studio because they consume a lot of memory. Also, companies should invest into improving virtual machines emulators.

Concerning the usability of our application, it can be said that this project can be used by a wide variety of people, from young students to seniors, because it provides functionalities that any human might need (anyone is interested in knowing the weather outside). Also, it has to be taken into consideration that popular web services like Facebook started from the idea of sharing for finding people with mutual interests, so the functionality of our application of saving interests can be a very powerful tool if further developed.

1. **References**

* <http://developer.android.com/>
* <https://parse.com/docs>
* <http://openweathermap.org/API>
* Skills developed during the Java faculty courses (Object-Oriented Programming)
* <http://json.parser.online.fr/> for helping us better take a grasp of the response received from the OpenWeatherMap api