Lab11 – Theory

**SharedPreferences:**

* Java class that is used to manage SharedPreferences files. These files work similar to a HashMap, on the principle of key-value. As a result, the form of a SharedPreferences file is a table:
  + the key -> a String variable that represents the name of the stored value
  + value -> the primitive value (String/ Intger/Float/Double/Set/Boolean)

These files represent a way to persistently store data. The lifecycle of the file is equivalent to the existence of the application on the mobile device.

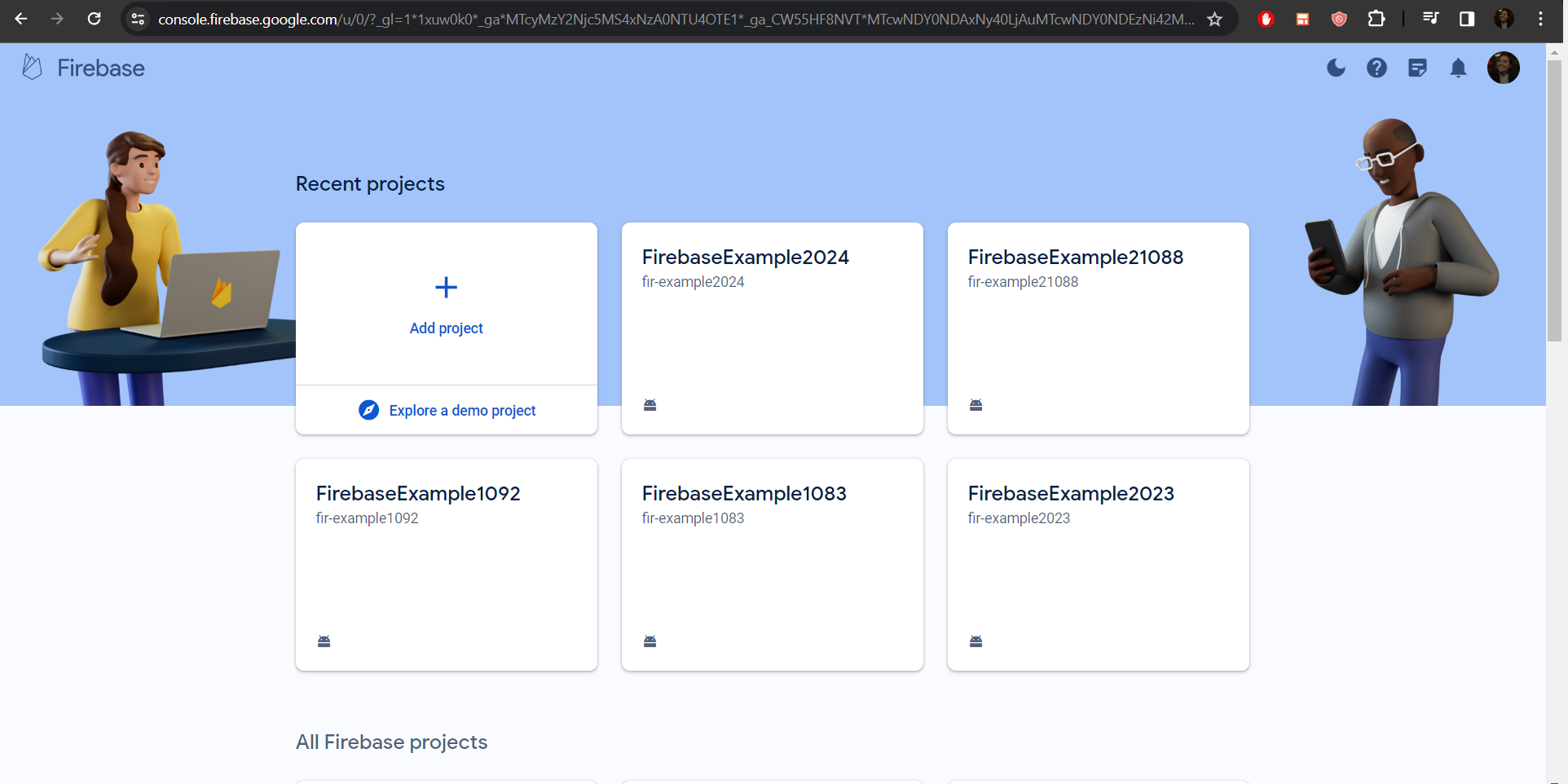
* Editor -> java class used to write in the SharedPreferences file. It presents methods of type put, that allows writing in the file
* Reading data from the file is done via get methods.
* getSharedPreferences -> method that available at activity level and it assures creating the SharedPreferences objects. It has two parameters:
  + a String parameter equivalent to the name of the file
  + the access mode -> default it should be MODE\_PRIVATE -> to allow the access to the file only to the application that created it
* apply -> method at Editor class that assures the saving of the data is effectively done.

**Firebase**:

The official site of the Firebase documentation: <https://firebase.google.com/>

By pressing the *Go to console* button (top corner right), the user should be able to see all their Firebase project.

Before being able to use Firebase in an Android application, it is necessary to create a new Firebase project and to integrate it in the Android project:



Moreover, some Gradle dependencies should be added in the project:

* Project-level:

id("com.google.gms.google-services") *version* "4.4.0" *apply* false

* App-level:
  + In plugins:

id("com.google.gms.google-services")

* + In dependencies:

*implementation*(platform("com.google.firebase:firebase-bom:32.7.0"))  
*implementation*("com.google.firebase:firebase-analytics")  
*implementation*("com.google.firebase:firebase-database")

Other information:

* FirebaseDatabase -> class that is used to create a connection between the mobile application and the realtime Firebase database. To instantiate it, *FirebaseDatabase.getInstance()* is used.
* DatabaseReference -> it creates the connection to a parent node from the database, so that access to data is granted and CRUD operations can be executed
* getReference() -> method at FirebaseDatabase that is used to initialize an object of DatabaseReference. It has only one parameter, a String, that represents the path where the operations will be applied to.
* push() -> method at DatabaseReference level that generates a new node and to add it in the DatabaseReference instance.
* getKey() -> method that is used to get the value generated via push() method
* setValue() -> method that assures sending information to a node. Usually, it is used together with child() method for adding the new data at the level of a node that was generated via push() method. It this chain is not realized, the information sent via setValue() method will be added at parent node level. This method has only a parameter, an Object type one, that represent the type of object that should be saved in the database
* getValue() -> the opposite of the setValue() method. It assures the conversion of a Firebase node to a Java instance. The method belongs to the DataSnapshot class and it has only one parameter, that is the type of the Java object used to deserialize the information. The java class needs to have a default constructor, getters and setters.
* removeValue() -> used for DELETE operation. Usually, it is used together with child() method. If it is used at the root level, then all the child nodes will be deleted.
* child() -> method at DatabaseReference level that assures getting a child node within a parent node. The type of returned node is DatabaseReference. The method has only one parameter, a String, that represents the name of the node.
* addValueEventListener() -> the method from DatabaseReference that assures that the application is notified when CRUD operations are executed at the Firebase database level. The method can be added at child or at parent level.
* addListenerForSingleValueEvent() -> it has the same properties as addValueListener. The only difference is that it assures the application is notified only when a single CRUD operation is executed on the database.
* DataSnapshot -> this class is used to create the conversion of a JSON to a Java object in the context of Firebase database.