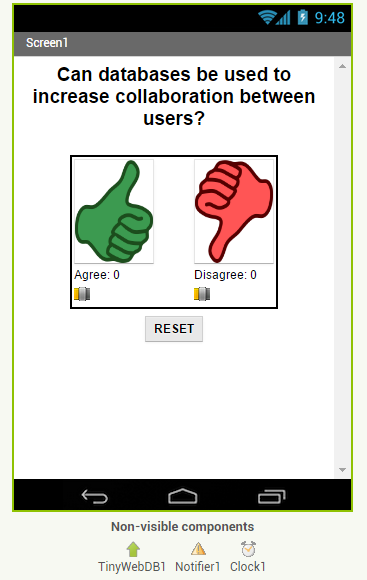
|  |  |
| --- | --- |
| In the first version of the Clicker app that we built using TinyWebDB, the clock timer requested data from the database every 3 seconds to remain in synch with the latest data. However, in that version of the app, there remains the possibility that some other user might update the data in the database immediately after we fetch it. Thus, we would end up with stale (outdated) data.    In this newer version of the Clicker app, we will switch from using TinyWebDB to Firebase as our underlying database. There are several advantages to using the latter. For this Clicker app, the main advantage that we draw from using Firebase is its “PUSH” capability - its ability to inform an app autonomously that data inside the database has changed. Objectives: In this lesson you will learn to:  * grasp the difference between ***PUSH***  and ***PULL*** *database* operations; * understand the efficiencies of a **PUSH** model; * use a **Firebase** database; | FirebaseUI.png  ***[Click to watch video](https://youtu.be/VNnWBgSM3uw)*** |

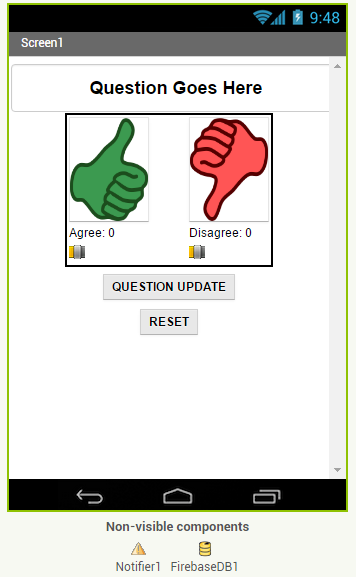
## 

# Getting Ready

To build the Firebase version of the Clicker app, you can start with your finished Clicker app from the previous lesson or you can start with [Clicker App Firebase Template](http://ai2.appinventor.mit.edu/?repo=templates.appinventor.mit.edu/trincoll/csp/unit6/templates/ClickerApp/ClickerFBtemplate.asc). Once the project opens use *Save As* to rename your project *ClickerFBstudent.*

## Comparing the User Interfaces: TinyWebDB vs Firebase





Regardless of whether you are using your own app or the official template as a starter, you will need to convert the User Interface to be similar to the one shown on the right. The template you are starting with will still have a TinyWebDB and a Clock component in the Non-visible components section of the UI as shown in the figure on the left. These are leftovers from the previous (TinyWebDB) version of the app. Make the following changes:

* Delete Clock1 and its associated blocks.
* Add a Firebase component from the Experimental drawer.
* Do not delete TinyWebDB1 yet. Once we change its blocks to Firebase blocks, we will come back to delete this component.

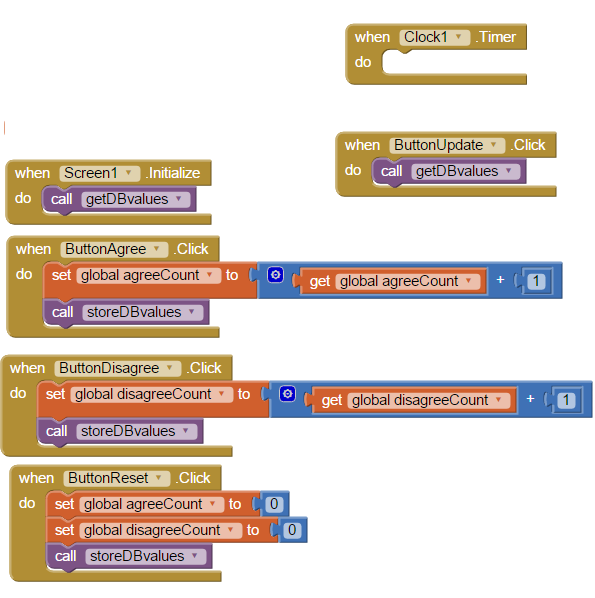
# Advantages of Using Firebase

Whereas the TinyWebDB database was public and, thus, shared among your classmates, each user gets his or her own private instance of storage in the Firebase database. This means that you no longer need to worry about generating unique tags - you can use simple tags like “agree” and “disagree”. The use of a private database also greatly reduces the chance of your data being maliciously damaged or destroyed.

However, by far the biggest advantage to using the Firebase database is its PUSH feature, which frees the client from having the responsibility of having to implement a polling algorithm to update its display of the data. Instead, the database can alert the app whenever its data changes. It does this by triggering the ***DataChanged*** event handler inside App Inventor. This event is triggered not only when another user updates data in the central database but also when we update the data, or when our app initializes.

The implications of this feature are far-reaching - it can make our app much more efficient: We no longer need to fetch data manually to make sure we have the latest. (We do not need to fetch it even when the app initializes because the DataChanged event handler is triggered at startup). When we write new data to the database, we can count on the DataChanged event handler to be triggered which in turn will signal us to update our display. This means that the same event handler that handles data changes due to other users can be used to handle our own data changes.

# Button Handlers: Mostly Unchanged, Some No Longer Needed

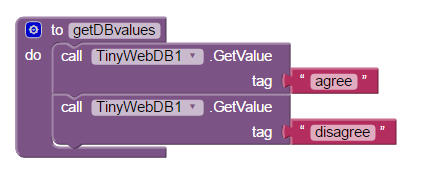
The image, below shows the button handlers left over from the TinyWebDB version of the app. Since we will be using FB’s PUSH capability to update our app whenever data in the Firebase database changes, we will no longer need to poll the database by using the Timer. Thus, this version of the app will not need a Clock/Timer. Furthermore, since all data is pushed to us whenever our app initializes, the Screen.Initialize event handler can also be discarded. For these reasons, the Timer and Screen.Initialize event handlers are no longer part of this version of the app and, thus, not included in the image, below.****

# Code Walkthrough

We will now walk through the important blocks in this app.

# Receiving Changed Data Asynchronously from Firebase

The purple block, below, shows the code we used to query the TinyWebDB database in the previous version of the Clicker app.



In this newer version of the app built with Firebase, we will instead use the following event handler block, ***Firebase.DataChanged*** to eliminate the getDBvalues procedure, above. The reason that the GetValue blocks in this procedure are no longer needed is that the *Firebase.DataChanged* event automatically updates our local copy of the data in our app whenever it changes in the centralized database. Thus, we never have to do a manual query of the database!

## You can delete the following code blocks which will be replaced by the Firebase.dataChanged event handler:

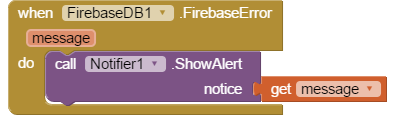
* Delete the **getDBvalues** procedure.
* Delete **When Screen.Initialize** event handler.
* Change **storeDBvalues** procedure to use **Firebase.StoreValue** blocks instead of the TinyWebDB.store blocks.
* Take code out of *when TinyWebDB.GotValue* and put it in *Firebase.DataChanged* like below*.* Change the parameter variable names.

# FBdatachanged.png

You can now go back to Design view to delete the TinyWebDB component from the User Interface.

## Error Handling is Identical

The Error Handling event handler for Firebase is nearly identical to its TinyWebDB counterpart:



# Testing the App

You should test this app in a similar manner to how you tested the TinyWebDB version - in groups. Note that the Update button will no longer be needed to keep the data in synch as the advantages of Firebase’s PUSH feature should be prominent in this newer version of the app.

**Testing this App**  
  
Firebase, unlike TinyWebDB, provides a different database for every app by providing a new Firebase Token (take a look at this property by clicking on the Firebase component in the UI). If you want to share a Firebase database among different apps, they will need to have the same Firebase token and the same tag names. You may want to share this token and tag names by email with a group of students so that your apps all share the same Firebase database. Or you can take turns testing your apps by using Build/App (provide QR code for .apk) in App Inventor, and then have each student scan that QR code from your screen. Make sure that everyone in the group is running the same version of the app from one member of the group, so that they will all use the same database tags and the same Firebase database. Test to see if the race condition has been reduced. If your app does not update its values correctly, check the tag names (capitalization matters!).

# Enhancement: Allow a Teacher to Update the Question

Add a feature that will create a special version of the app, the “Teacher” version, to update the question displayed on the screen in real time.

**Adding the “Update Questions” feature:**

1. Build a separate version of the app called “ClickerFBteacher” (use File/Save As). Only this version will be allowed to change the questions that appear on the original student version of the app.
2. Replace the Question Label in the teacher version of the app with a TextBox to allow the teacher to update the question field in real time. Add an “Update Question” button to the teacher app that will store the new question into the Firebase database from where it will eventually be pushed to all the users. Also, reset the counters and store them in the database too.
3. Change the student version of the app to accept new questions while the app is running. This will involve adding code to the Firebase.DataChanged event handler to see if the question was changed in the database and changing the question label accordingly and re-enabling the voting buttons. Note that the *Question* data will consist of a string whereas the *agree* and *disagree* data were numbers.
4. Remove the reset button from the UI of the student side so that only the teacher can reset the counters. Test with your group with one student using the teacher app and the rest using the corresponding student apps.

# 

***Nice work! Complete the Self-Check Exercises and Portfolio Reflection Questions as directed by your instructor.***