Basic web application with Firebase

Web based technologies

Alfonso López Ruiz

How to create a new project in Firebase

Basic operations: CRUD

Try out a basic web application with Firebase

What is firebase?

In this tutorial we'll use Firebase as a **Realtime Database**, which means we'll get real time data.

That's achieved by using **WebSockets** instead of HTTP calls. This is what lets us get **real time data**, since we're not following a Request-Response pattern as HTTP does.

Also, because we don't have to make a request before we get any data, **WebSockets are much faster**.
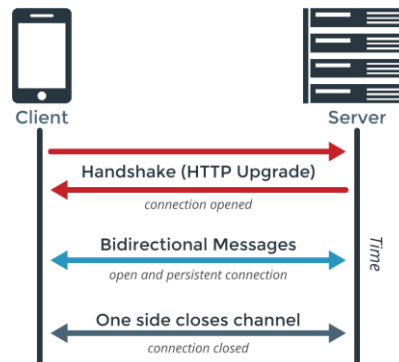
What is firebase?

Actually, Firebase is much more than a Realtime database:

- We can upload files, mostly images, directly from the client **(Storage)**
- Email / Password authentication system. It also supports authentication for Google, Github... **(Authentication)**
- Notifications to users **(Cloud Messaging)**

...

Realtime Database
Authentication
Cloud Messaging
Storage
Hosting
Remote Config
Test Lab
Crash Reporting

The page where we can manage our projects is https://console.firebase.google.com (we can access with our Google account).

Once our project is created we get a piece of code like the following one:

```html
<script src="https://www.gstatic.com/firebasejs/4.10.1/firebase.js"></script>
<script>
    var config = {
        apiKey: "<API_KEY>",
        authDomain: "<PROJECT_ID>.firebaseapp.com",
        databaseURL: "https://<DATABASE_NAME>.firebaseio.com",
        storageBucket: "<BUCKET>.appspot.com",
        messagingSenderId: "<SENDER_ID>",
    };
    firebase.initializeApp(config);
</script>
```
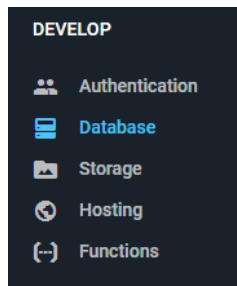
We must place this Javascript code in our application: in the HTML code or in a Javascript file referenced by our application.

This piece of code references our database, so we can initialize and manipulate it.

From now on, we'll use the tab **Database** at the Firebase console.

DEVELOP

Authentication

Database

Storage

Hosting

Functions

In this tab we can access the **Rules** of our database, which determine the **Read and Write rights** of the users, as well as the **structure of our database and indexes**.

For **developing** purposes we must mark the Read and Write properties as true. It means that anyone can read or write in our database, so we must set it as necessary once the developing process is over.

```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

In this tab we can also access the **data from our database**...

In contrast to a SQL database, Firebase has no tables or registers. Firebase uses a JSON tree to save the data, where the values of our entities will be nodes at that tree structure.

Our tree structure must be as flat as possible. This way, if we want to get the chat 'one', we don't get all the messages related to that chat.

```json
{
    "chats": {
        "one": {
            "title": "Historical Tech Pioneers",
            "messages": {
                "m1": {"message": "Relay malfunction found. Cause: moth."},
                "m2": {...},
            }
        },
        "two": {...}
    }
}
```

```json
{
    "chats": {
        "one": {
            "title": "Historical Tech Pioneers",
        },
        "two": {...},
    },

    "messages": {
        "one": {
            "m1": {
                "message": "The relay seems to be malfunctioning.",
            },
            "m2": {...},
        },
        "two": {...},
    }
}
```
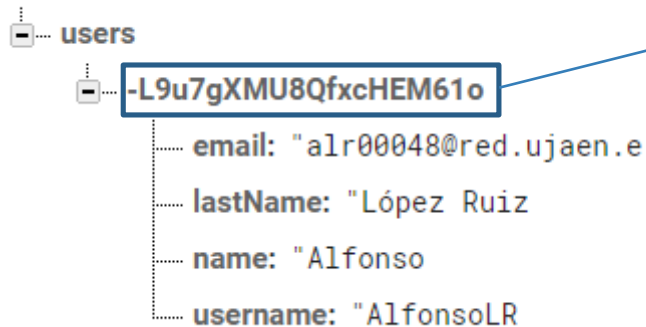
From now on, our database consist of users, and we'll see how to read and modify the database (insert, update and delete).

web-based-technologies
- **users**
  - **-L9u7gXMU8QfxcHEM61o**
    - email: "alr00048@red.ujaen.e
    - lastName: "López Ruiz
    - name: "Alfonso
    - username: "AlfonsoLR

We don't have auto increment properties, but we got strings that we can use as a key for our users.

Once we copy the previous piece of code in our application, **we can save an instance of our database:**

var db = firebase.database()

How can we insert users?

In case we know the tree path, we can do it as follows:

If the user wants to modify the username, this approach is not the best, since we need to delete the user and create a new one.

```
var users = db.ref('users');
var data = {
    username: username,
    name: name,
    lastName: lastName,
    email: email
}
users.push(data);
```

Database path

```
db.ref('users/' + username).set({
    name: name,
    lastName: lastName,
    email: email,
});
```

We can automatically generate a key for the new user with **'push'**.

How can we read users?

Because of that bi-directional communication we've already talked about, our queries can be triggered more than once, so we can get events like updates, insertions or deletions.

As we're used to see, queries are only 'executed' once, and we can get this behaviour in Firebase too.

```javascript
var userRef = db.ref('users/' + userId);
userRef.once('value', function(snapshot) {
  // Do something
});
```

We could also make a query that listens to events, and remove that listener once our query is executed.

Once we know how to listen to events, we'll see how to remove those listeners.

Let's suppose the next situation: we need to show all our users in our page in realtime, so we can listen for insertions, updates and deletions.

What events can we associate to a query?

Event	Description
child_added	This event is triggered once per existing element and once again when a new element is inserted.
child_changed	This event detects any change in a secondary node, including the descendants from those nodes.
child_removed	This even is triggered when a secondary node is removed.
child_moved	This event detects changes in the order of an ordered list.

We'll see how to insert, update and remove users.

So, how can we read users and display them?

We want to show all our users in our page, and we want this list to be updated in realtime, so we can accomplish that with the event 'child_added'.

For example, in jQuery we can execute this piece of code once the document is ready.

```
db.ref().child('users').on('child_added', function(snapshot) {
    addUser(snapshot.key, snapshot.val().name + ' ' + snapshot.val().lastName, snapshot.val().username, snapshot.val().email);
});
```

Automatically generated key

User associated to that key (only the attributes of that user (username, email...) – The behavior is not the same in 'once' queries!!

This will let us show the users that already exists as soon as the page is loaded, and will let us listen to insertions.

How can we update users?

In our application (we'll see later) we can update the users we've already created, including their username (that's why we don't use it as a key). We can update an user with **'update',** taking into account we must have a reference to one of them.
We just want to update the information of an specific user once, that's why we'll use the 'once' query we saw previously.

```
var usernameRef = db.ref().child('users').orderByChild('username').equalTo('newUsername').once('value',
            function(snapshot) {
                var update = {
                    username: username
                    name: name,
                    lastName: lastName,
                    email: email
                };
                var userRef = db.ref('users/' + Object.keys(snapshot.val())[0]);
                userRef.update(update);
            });
```

We're searching by the column 'username'

Node from our database ~ User 'table'

Reference to an user.
That's an special case, 'once'
queries returns the whole
user: key + attributes, so we
need to access only its key.

How can we delete users?

The way to delete users is similar to the update process. We need to get those users with a certain username, and then execute the 'remove' order.

```javascript
var usernameRef = db.ref().child('users').orderByChild('username').equalTo('newUsername').once('value',
            function(snapshot) {
                    db.ref('users/' + Object.keys(snapshot.val())[0]).remove();
            });
```

In order to make queries we've used methods like orderByChild or equalTo but there's a lot of methods we can use:
- limitToLast
- limitToFirst
- startAt
- orderByKey
- ...

[Reference to the Firebase query guide!](#)

We can insert, update and delete users, even listen to insert events, but **if an update or a deletion occurs, our page won't get updated...**

As we did with 'child_added', we still need to listen for those events:

<div id="keyIdentifier"> ... </div>

```
db.ref().child('users').on('child_removed', function(snapshot) {
    removeUser(snapshot.key);
});
```
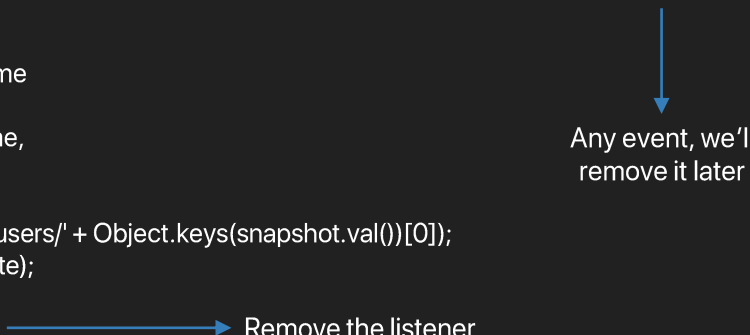
Removes

Updates

```
db.ref().child('users').on('child_changed', function(snapshot) {
    updateUser(snapshot.key, snapshot.val().name + ' ' + snapshot.val().lastName, snapshot.val().username, snapshot.val().email);
});
```

Alfonso López Ruiz

Username: AlfonsoLR

Email: alr00048@red.ujaen.es

We said that instead of using the 'once' query we could also make a query and then remove the listener associated to that query. Obviously it is not recommended to use if we can use the 'once' query, but it could be interesting to learn how to remove a listener.

```javascript
var usernameRef = db.ref().child('users').orderByChild('username').equalTo('newUsername').on('child_added',
          function(snapshot) {
                var update = {
                      username: username
                      name: name,
                      lastName: lastName,
                      email: email
                };
                var userRef = db.ref('users/' + Object.keys(snapshot.val())[0]);
                userRef.update(update);
          });
    db.ref().off('child_added', usernameRef);
```

Any event, we'll remove it later

Remove the listener

We've created a web application in order to show an example of realtime data.

It consists only of users with username, name and email. Take into account that the username must be unique, even though it is not the key for the entity user.

You can check this project in Github!

Firebase project - AlfonsoLRz

Links to keep learning about Firebase:

- Firebase web documentation
- Firebase web queries
- Getting started with Firebase on the Web (Video)
- Firebase database for SQL developers (List of videos)
- Firebase channel (Youtube)

Other resources:

- Javascript library of particles animations. 😀