FIREBASE

Firebase – Home

Firebase is a backend platform for building Web, Android and IOS applications. It offers real time database, different APIs, multiple authentication types and hosting platform. This is a tutorial, which covers the basics of the Firebase platform and explains how to deal with its various components and sub-components.

As per firebase official website –

Firebase can power your app's backend, including data storage, user authentication, static hosting, and more. Focus on creating extraordinary user experiences. We will take care of the rest. Build cross-platform native mobile and web apps with our Android, iOS, and JavaScript SDKs. You can also connect Firebase to your existing backend using our server-side libraries or our REST API.

Firebase - Overview

Firebase Features

- **Real-time Database** − Firebase supports JSON data and all users connected to it receive live updates after every change.
- **Authentication** − We can use anonymous, password or different social authentications.
- **Hosting** − The applications can be deployed over secured connection to Firebase servers.

Today there are many other cloud API's available on firebase like the ML kit

Firebase Advantages

- It is simple and user friendly. No need for complicated configuration.
- The data is real-time, which means that every change will automatically update connected clients.
- Firebase offers simple control dashboard.
- There are a number of useful services to choose.

Firebase Limitations

- Firebase free plan is limited to 50 Connections and 100 MB of storage.

Firebase - Environment Setup

We will see how to add Firebase to the existing application. We will need **NodeJS**. Check the link from the following table, if you do not have it already.

Step 1 - Create Firebase Account

You can create a Firebase account https://console.firebase.google.com

Step 2 - Create Firebase App

You can create new app from the dashboard page. The following image shows the app we created. We can click the **Manage App** button to enter the app.

Create basic HTML/js App

You just need to create a folder where your app will be placed. Inside that folder, we will need **index.html** and **index.js** files. We will add Firebase to the header of our app.

index.html

```
<html>
   <head>
      <script src = "https://cdn.firebase.com/js/client/2.4.2/firebase.js"></script>
      <script type = "text/javascript" src = "index.js"></script>
   </head>

   <body>

   </body>
</html>
```

Step 3b - Use NPM or Bower

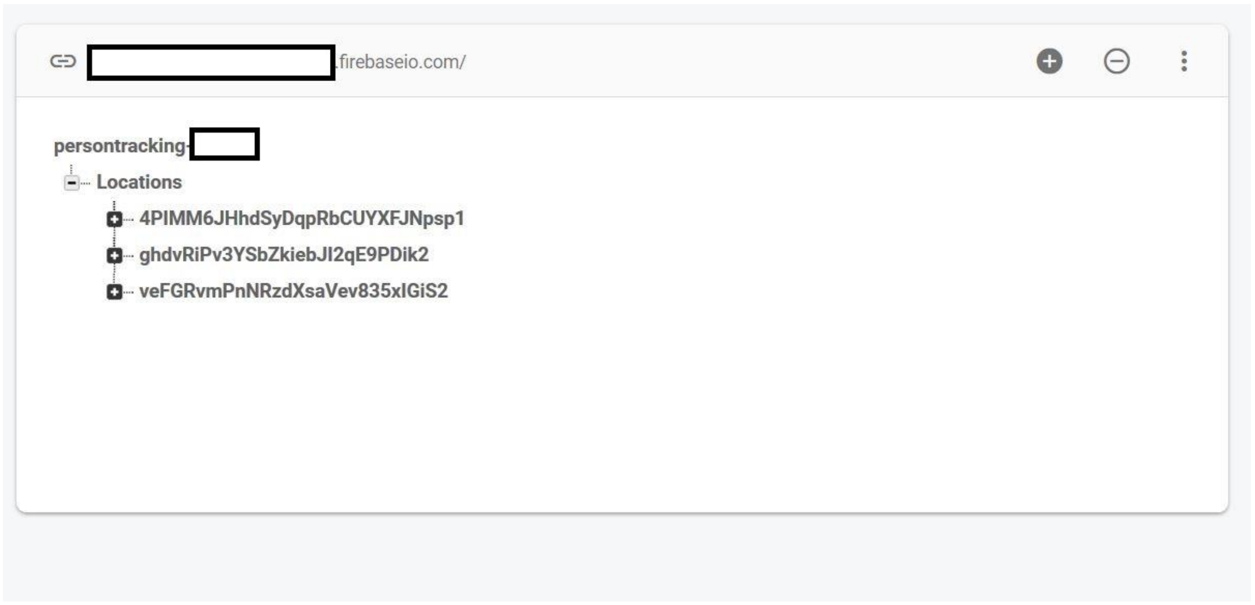
If you want to use your existing app, you can use Firebase NPM or Bowers packages. Run one of the following command from your apps root folder.

```
npm install firebase --save
bower install firebase
```

Firebase – Data

The Firebase data is representing JSON objects. If you open your app from Firebase dashboard, you can add data manually by clicking on the + sign.

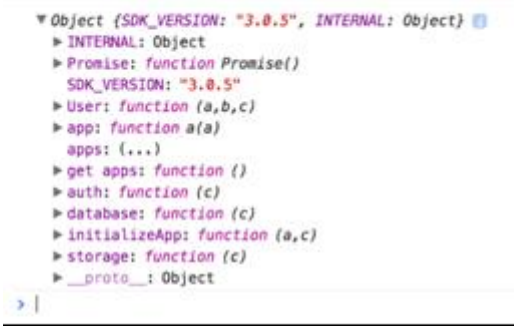
We will create a simple data structure. You can check the image below.



```
[redacted].firebaseio.com/

persontracking-[redacted]
  Locations
    4PIMM6JHhdSyDqpRbCUYXFJNpsp1
    ghdvRiPv3YSbZkiebJl2qE9PDik2
    veFGRvmPnNRzdXsaVev835xIGiS2
```

In the previous chapter, we connected Firebase to our app. Now, we can log Firebase to the console.

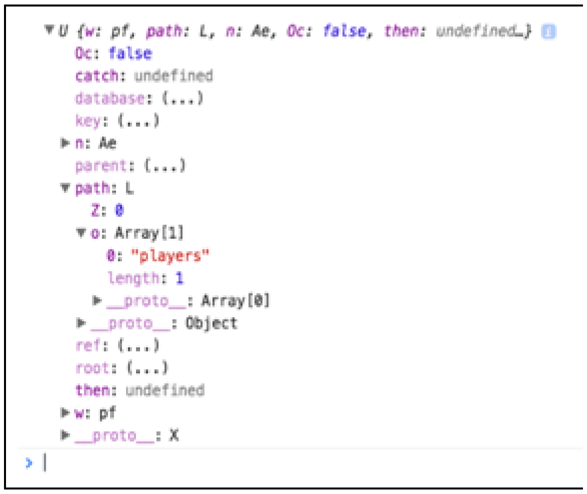
```
console.log(firebase)
```



```
▼Object {SDK_VERSION: "3.0.5", INTERNAL: Object}
  ▶INTERNAL: Object
  ▶Promise: function Promise()
   SDK_VERSION: "3.0.5"
  ▶User: function (a,b,c)
  ▶app: function a(a)
   apps: (...)
  ▶get apps: function ()
  ▶auth: function (c)
  ▶database: function (c)
  ▶initializeApp: function (a,c)
  ▶storage: function (c)
  ▶__proto__: Object
> |
```

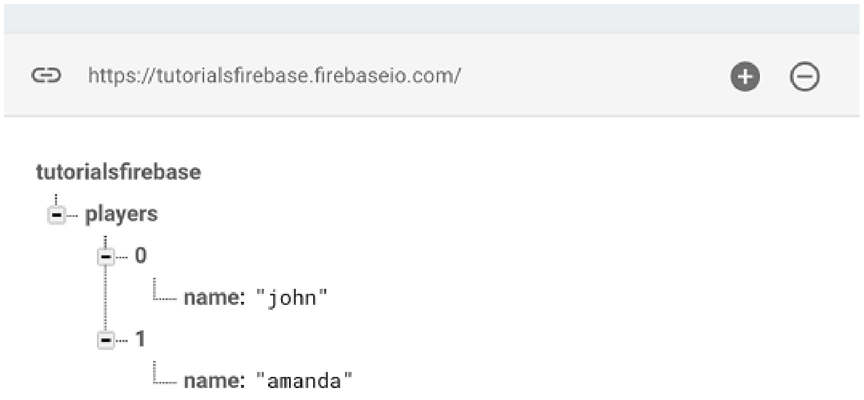
We can create a reference to our player’s collection.

```
var ref = firebase.database().ref('players');

console.log(ref);
```



```
▼U {w: pf, path: L, n: Ae, Oc: false, then: undefined…}
   Oc: false
   catch: undefined
   database: (...)
   key: (...)
 ▶n: Ae
   parent: (...)
 ▼path: L
     Z: 0
   ▼o: Array[1]
       0: "players"
       length: 1
     ▶__proto__: Array[0]
   ▶__proto__: Object
   ref: (...)
   root: (...)
   then: undefined
 ▶w: pf
 ▶__proto__: X
> |
```

Firebase – Arrays



```
https://tutorialsfirebase.firebaseio.com/

tutorialsfirebase
  players
    0
      name: "john"
    1
      name: "amanda"
```

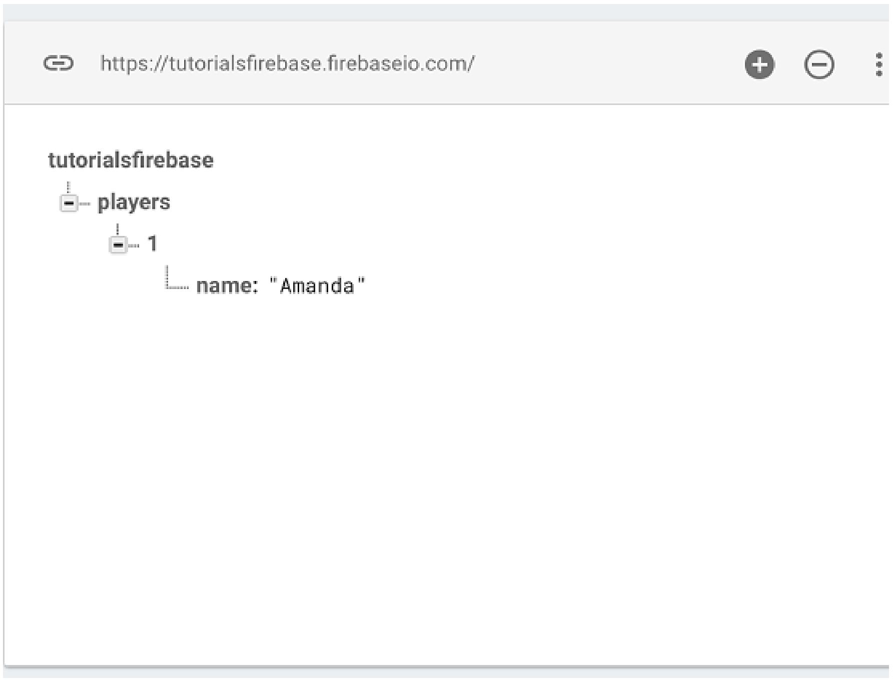
We could create this data by sending the following JSON tree to the player's collection.

['john', 'amanda']

This is because Firebase does not support Arrays directly, but it creates a list of objects with integers as key names.

The reason for not using arrays is because Firebase acts as a real time database and if a couple of users were to manipulate arrays at the same time, the result could be problematic since array indexes are constantly changing.

The way Firebase handles it, the keys (indexes) will always stay the same. We could delete **john** and **amanda** would still have the key (index) 1.



https://tutorialsfirebase.firebaseio.com/

```
tutorialsfirebase
  players
    1
      name: "Amanda"
```

Firebase - Write Data

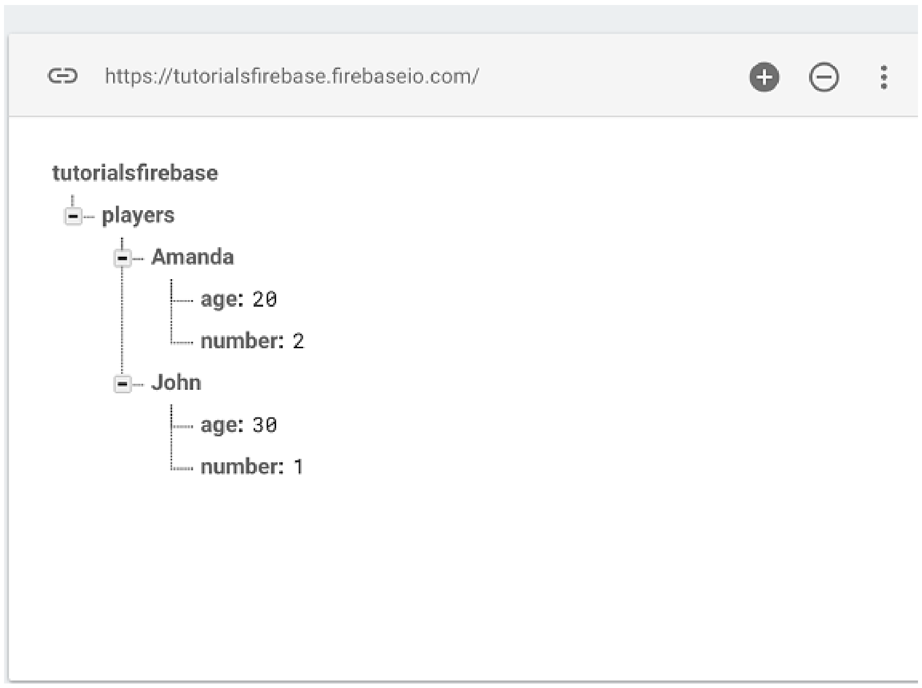
We will see how to save your data to Firebase.

Set

The **set** method will write or replace data on a specified path. Let us create a reference to the player's collection and set two players.

```
var playersRef = firebase.database().ref("players/");

playersRef.set ({
  John: {
    number: 1,
    age: 30
  },

  Amanda: {
    number: 2,
    age: 20
  }
});
```

We will see the following result.



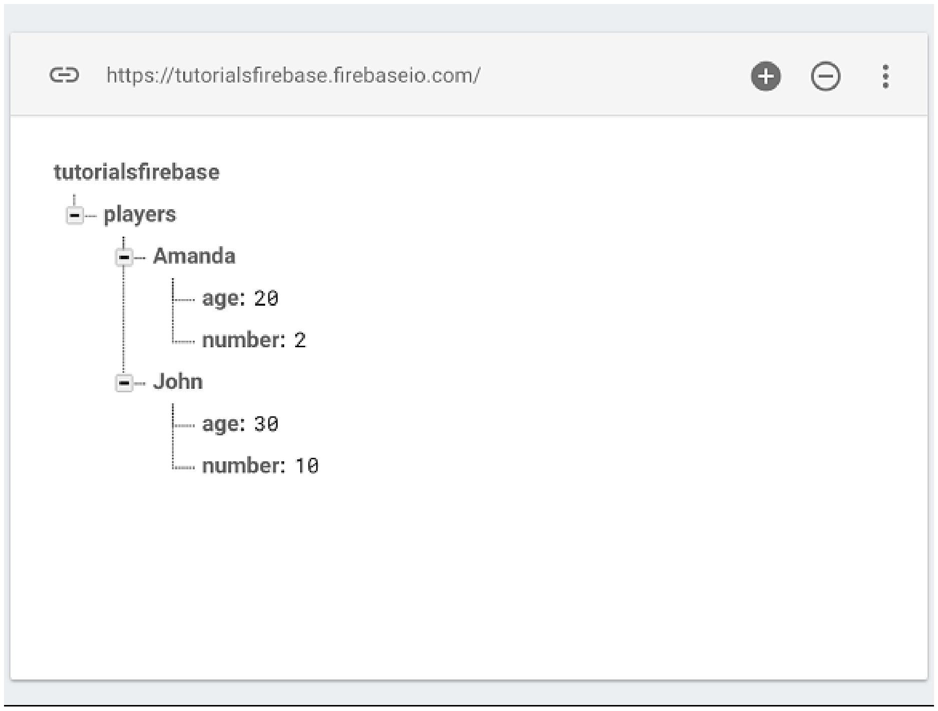
```
https://tutorialsfirebase.firebaseio.com/

tutorialsfirebase
 players
    Amanda
       age: 20
       number: 2
    John
       age: 30
       number: 1
```

Update

We can update the Firebase data in a similar fashion. Notice how we are using the **players/john** path.

```
var johnRef = firebase.database().ref("players/John");


johnRef.update ({
   "number": 10
});
```

When we refresh our app, we can see that the Firebase data is updating.



https://tutorialsfirebase.firebaseio.com/

```
tutorialsfirebase
  players
    Amanda
      age: 20
      number: 2
    John
      age: 30
      number: 10
```

Firebase - Write List Data

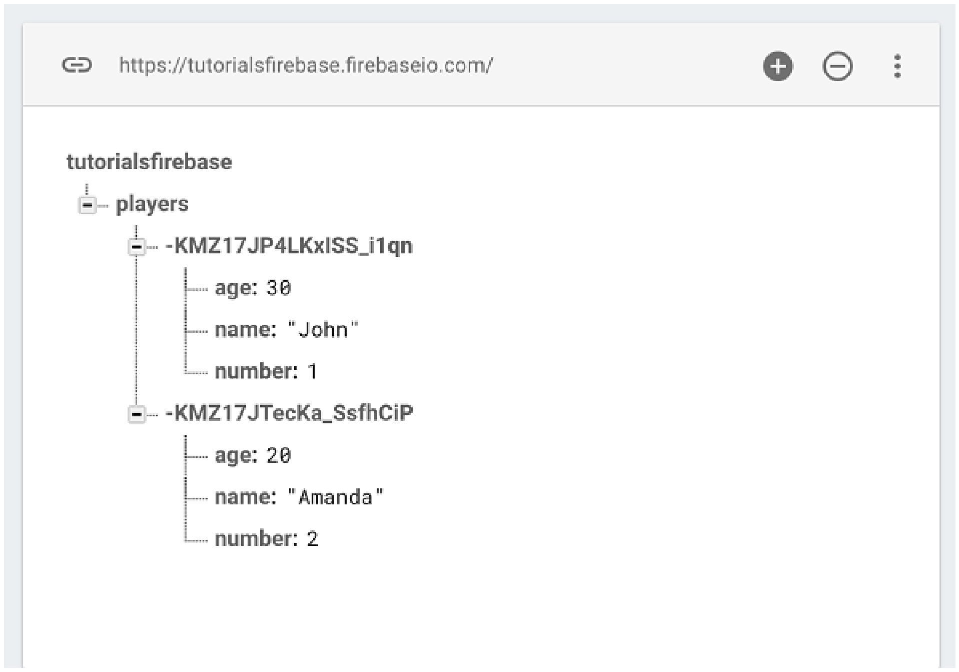
Sometimes you need to have a unique identifier for your data. When you want to create unique identifiers for your data, you need to use the push method instead of the set method.

The Push Method

The **push()** method will create a unique id when the data is pushed. If we want to create our players from the previous chapters with a unique id, we could use the code snippet given below.

```
var ref = new Firebase('https://tutorialsfirebase.firebaseio.com');

var playersRef = ref.child("players");
playersRef.push ({
   name: "John",
   number: 1,
   age: 30
});

playersRef.push ({
   name: "Amanda",
   number: 2,
   age: 20
});
```

Now our data will look differently. The name will just be a name/value pair like the rest of the properties.

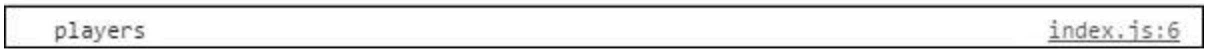

```
https://tutorialsfirebase.firebaseio.com/

tutorialsfirebase
  players
    -KMZ17JP4LKxISS_i1qn
      age: 30
      name: "John"
      number: 1
    -KMZ17JTecKa_SsfhCiP
      age: 20
      name: "Amanda"
      number: 2
```

The Key Method

We can get any key from Firebase by using the **key()** method. For example, if we want to get our collection name, we could use the following snippet.

```
var ref = new Firebase('https://tutorialsfirebase.firebaseio.com');


var playersRef = ref.child("players");


var playersKey = playersRef.key();
console.log(playersKey);
```

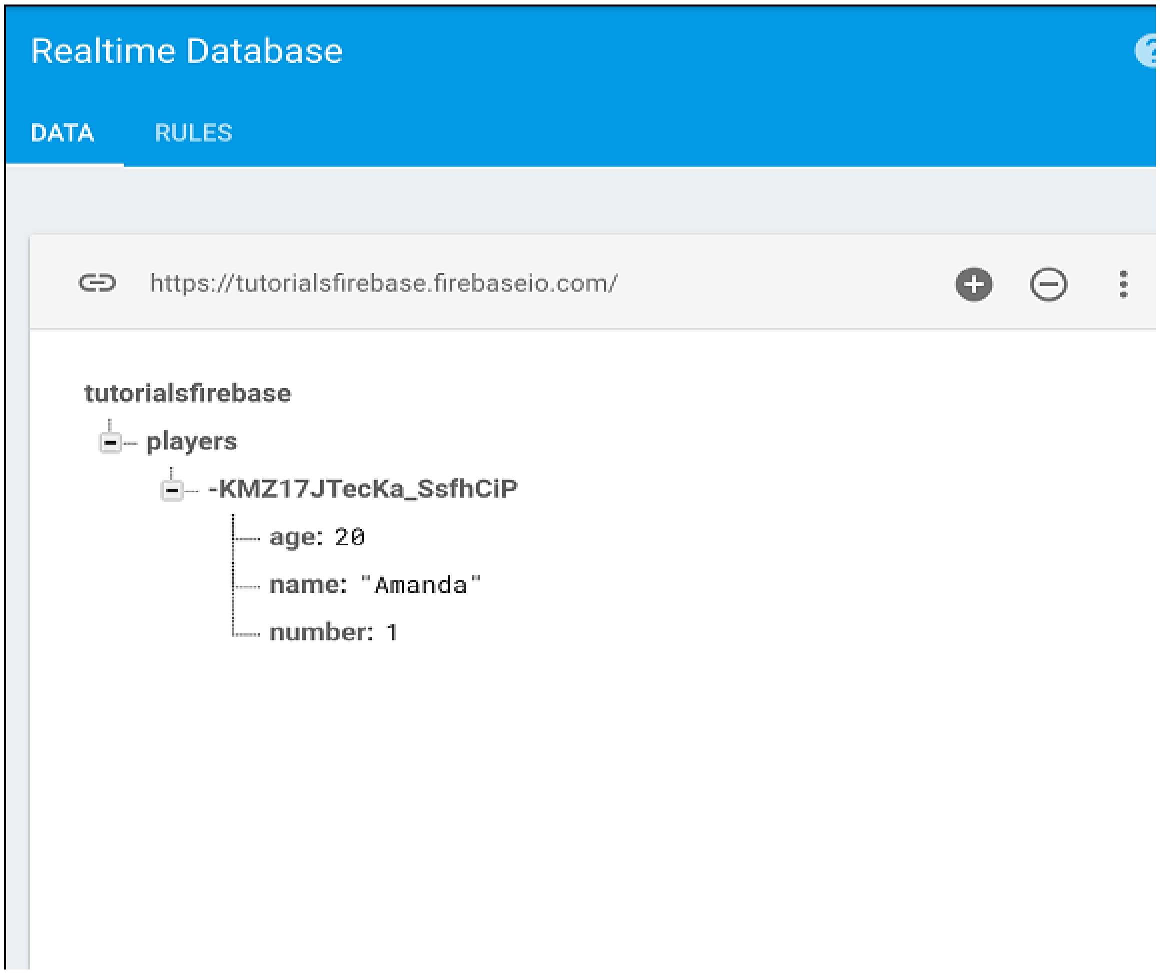
The console will log our collection name (players).


```
players                                                    index.js:6
```

Firebase - Write Transactional Data

Transactional data is used when you need to return some data from the database then make some calculation with it and store it back.

Let us say we have one player inside our player list.



```
tutorialsfirebase
  players
    -KMZ17JTecKa_SsfhCiP
      age: 20
      name: "Amanda"
      number: 1
```

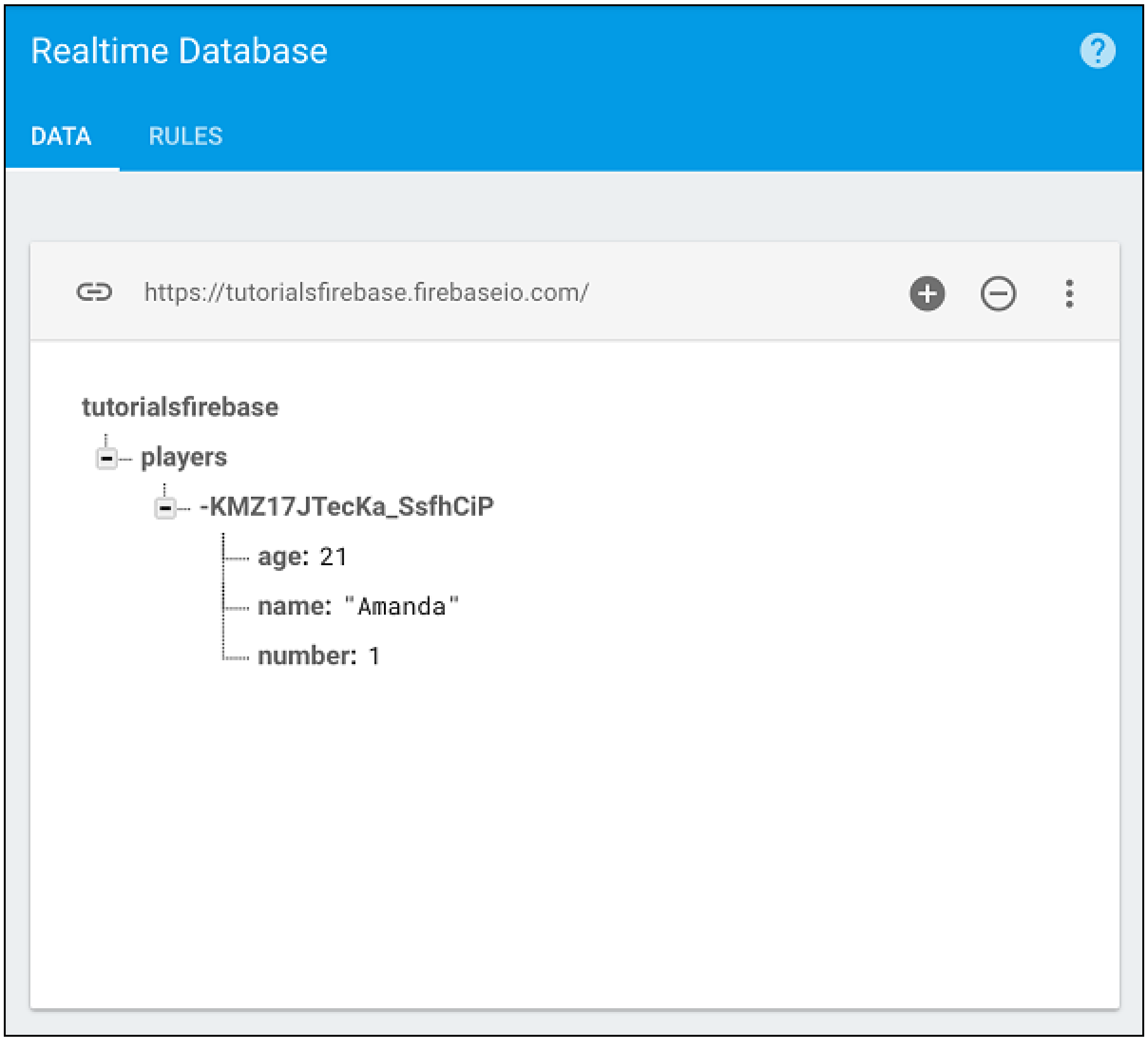
We want to retrieve property, add one year of age and return it back to Firebase.

The **amandaRef** is retrieving the age from the collection and then we can use the transaction method. We will get the current age, add one year and update the collection.

var ref = new Firebase('https://tutorialsfirebase.firebaseio.com');

```
var amandaAgeRef = ref.child("players").child("-KGb1Ls-gEErWbAMMnZC").child('age');

amandaAgeRef.transaction(function(currentAge) {
  return currentAge + 1;
});
```

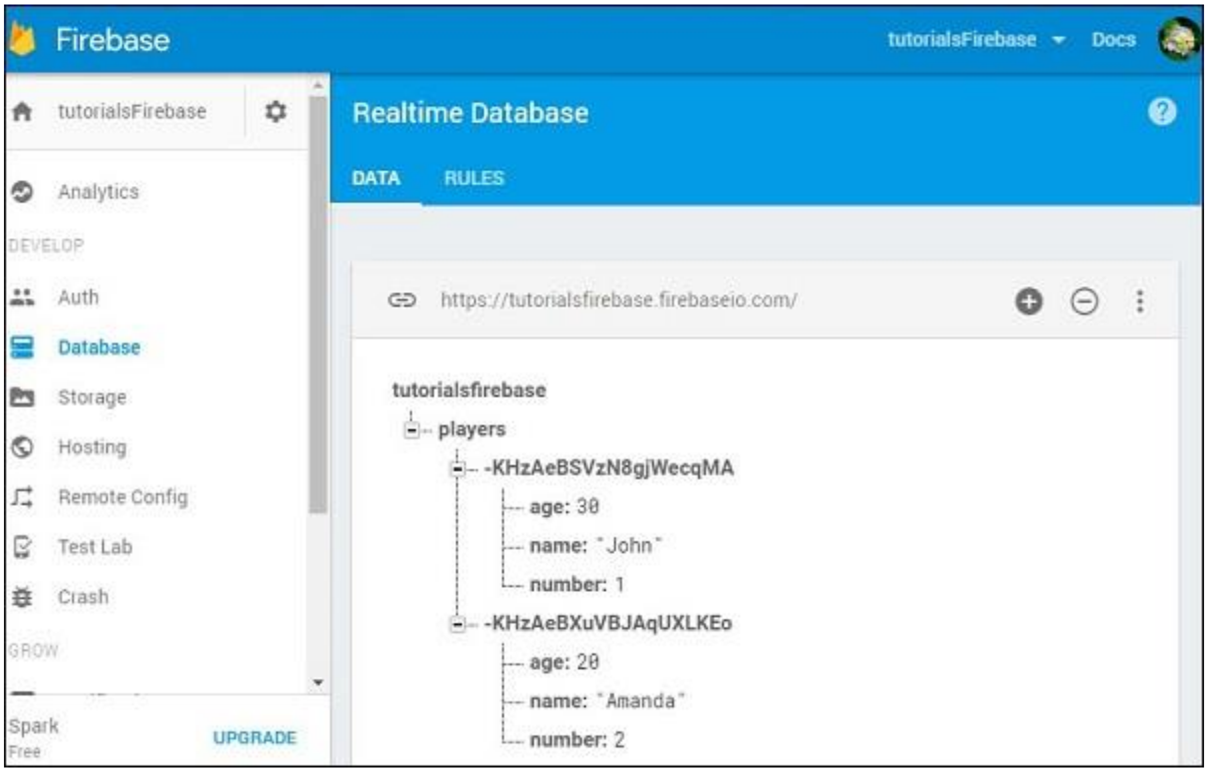
If we run this code, we can see that the age value is updated to **21**.



```
tutorialsfirebase
  players
    -KMZ17JTecKa_SsfhCiP
      age: 21
      name: "Amanda"
      number: 1
```

Firebase - Read Data

We will see how to read Firebase data. The following image shows the data we want to read.



https://tutorialsfirebase.firebaseio.com/

```
tutorialsfirebase
  players
    -KHzAeBSVzN8gjWecqMA
      age: 30
      name: "John"
      number: 1
    -KHzAeBXuVBJAqUXLKEo
      age: 20
      name: "Amanda"
      number: 2
```

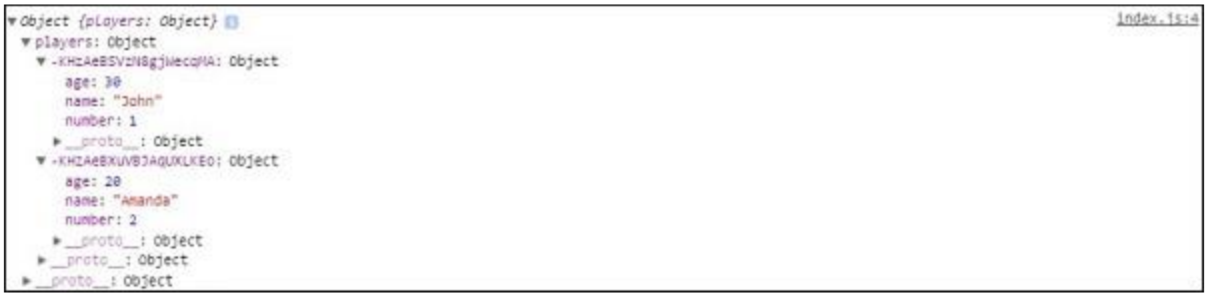
We can use the **on()** method to retrieve data. This method is taking the event type as **"value"** and then retrieves the **snapshot** of the data. When we add **val()** method to the snapshot, we will get the JavaScript representation of the data.

Example

Let us consider the following example.

```
var ref = firebase.database().ref();

ref.on("value", function(snapshot) {
   console.log(snapshot.val());
}, function (error) {
   console.log("Error: " + error.code);
});
```

If we run the following code, our console will show the data.



```
▼Object {players: Object}                                      index.js:4
  ▼players: Object
    ▼ -KHzAeBSVzN8gjWecqMA: Object
        age: 30
        name: "John"
        number: 1
      ▶__proto__: Object
    ▼ -KHzAeBXUVBJAQUXLKEo: Object
        age: 20
        name: "Amanda"
        number: 2
      ▶__proto__: Object
    ▶__proto__: Object
  ▶__proto__: Object
```

Firebase - Event Types

Firebase offers several different event types for reading data. Some of the most commonly used ones are described below.

value

The first event type is **value**. We showed you how to use value in our last chapter. This event type will be triggered every time the data changes and it will retrieve all the data including children.

child_added

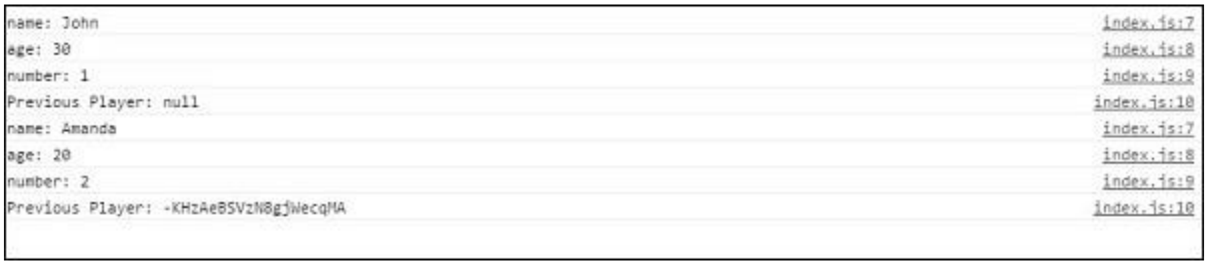
This event type will be triggered once for every player and every time a new player is added to our data. It is useful for reading list data because we get access of the added player and previous player from the list.

Example

Let us consider the following example.

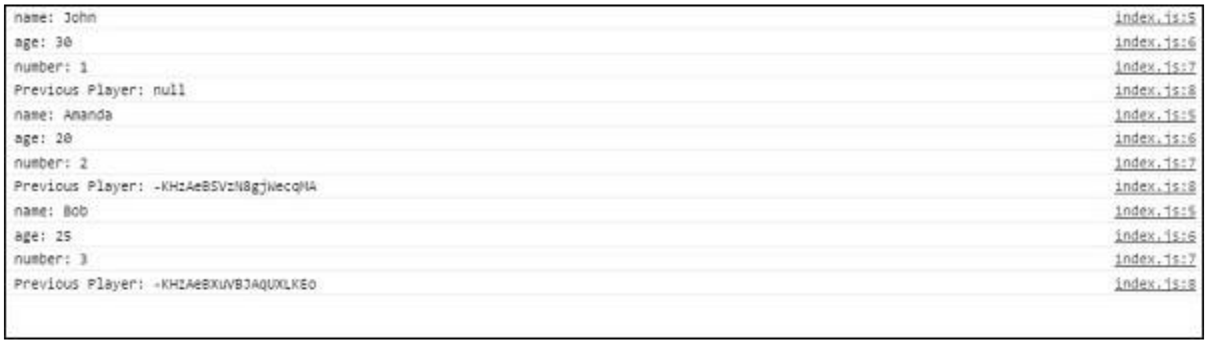
```
var playersRef = firebase.database().ref("players/");


playersRef.on("child_added", function(data, prevChildKey) {
   var newPlayer = data.val();
   console.log("name: " + newPlayer.name);
   console.log("age: " + newPlayer.age);
   console.log("number: " + newPlayer.number);
   console.log("Previous Player: " + prevChildKey);
});
```

We will get the following result.



```
name: John                                                    index.js:7
age: 30                                                       index.js:8
number: 1                                                     index.js:9
Previous Player: null                                         index.js:10
name: Amanda                                                  index.js:7
age: 20                                                       index.js:8
number: 2                                                     index.js:9
Previous Player: -KHzAeBSVzN8gjWecqMA                         index.js:10
```

If we add a new player named Bob, we will get the updated data.



```
name: John                                                    index.js:5
age: 30                                                       index.js:6
number: 1                                                     index.js:7
Previous Player: null                                         index.js:8
name: Amanda                                                  index.js:5
age: 20                                                       index.js:6
number: 2                                                     index.js:7
Previous Player: -KHzAeBSVzN8gjWecqMA                         index.js:8
name: Bob                                                     index.js:5
age: 25                                                       index.js:6
number: 3                                                     index.js:7
Previous Player: -KHZAeBXUVBJAQUXLKEo                         index.js:8
```

child_changed

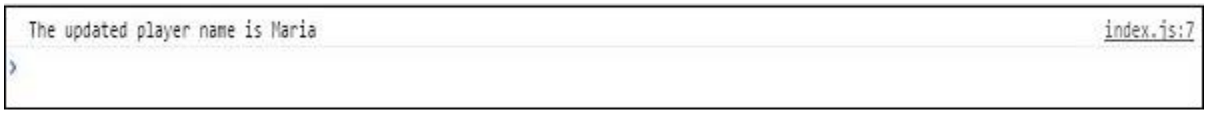
This event type is triggered when the data has changed.

Example

Let us consider the following example.

```
var playersRef = firebase.database().ref("players/");


playersRef.on("child_changed", function(data) {
   var player = data.val();
   console.log("The updated player name is " + player.name);
});
```

We can change **Bob** to **Maria** in Firebase to get the update.



```
The updated player name is Maria                                    index.js:7
>
```

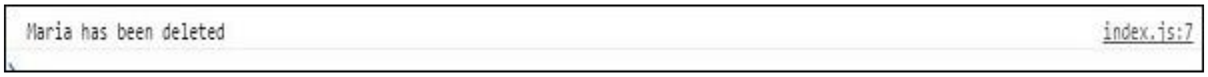
child_removed

If we want to get access of deleted data, we can use **child_removed** event type.

Example

```
var playersRef = firebase.database().ref("players/");


playersRef.on("child_removed", function(data) {
   var deletedPlayer = data.val();
   console.log(deletedPlayer.name + " has been deleted");
});
```

Now, we can delete Maria from Firebase to get notifications.



```
Maria has been deleted                                              index.js:7
```

Firebase - Detaching Callbacks

Let us say we want to detach a callback for a function with **value** event type.

Example

```
var playersRef = firebase.database().ref("players/");


ref.on("value", function(data) {
   console.log(data.val());
}, function (error) {
   console.log("Error: " + error.code);
});
```

We need to use **off()** method. This will remove all callbacks with **value** event type.

```
playersRef.off("value");
```

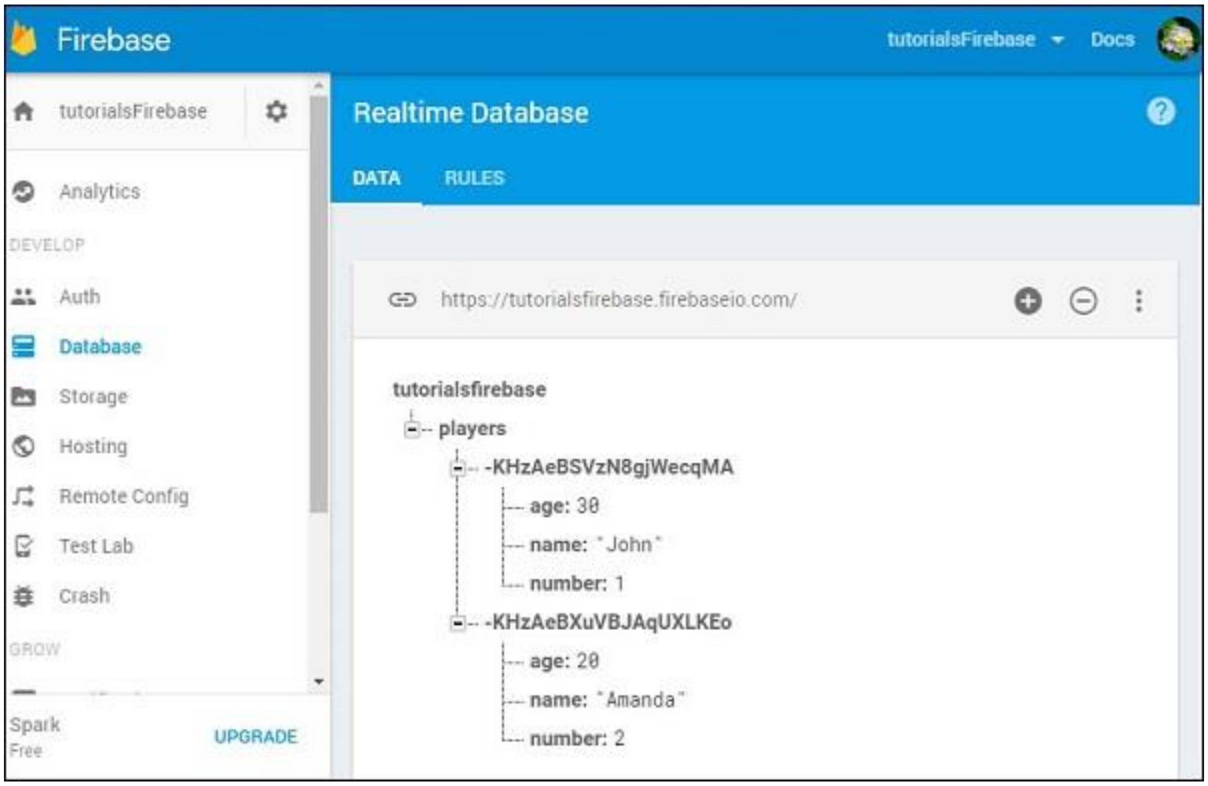
Detach All Callbacks

When we want to detach all callbacks, we can use −

```
playersRef.off();
```

Firebase – Queries

Firebase offers various ways of ordering data. We will see simple query examples. We will use the same data from our previous chapters.



```
tutorialsfirebase
  players
    -KHzAeBSVzN8gjWecqMA
      age: 30
      name: "John"
      number: 1
    -KHzAeBXuVBJAqUXLKEo
      age: 20
      name: "Amanda"
      number: 2
```

Order by Child

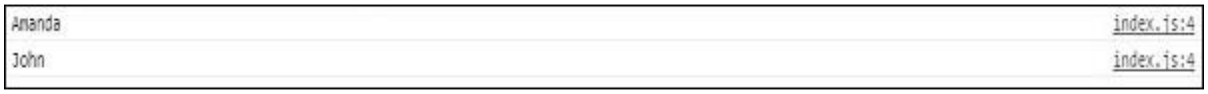
To order data by name, we can use the following code.

Example

Let us consider the following example.

```
var playersRef = firebase.database().ref("players/");
playersRef.orderByChild("name").on("child_added", function(data) {
  console.log(data.val().name);
});
```

We will see names in the alphabetic order.



```
Amanda                                                        index.js:4
John                                                          index.js:4
```

Order by Key

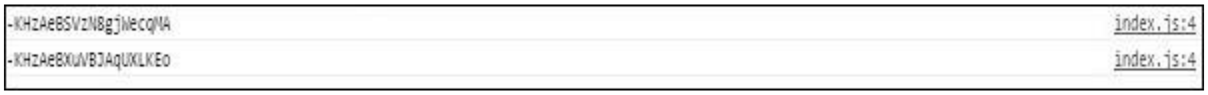
We can order data by key in a similar fashion.

Example

Let us consider the following example.

```
var playersRef = firebase.database().ref("players/");


playersRef.orderByKey().on("child_added", function(data) {
   console.log(data.key);
});
```

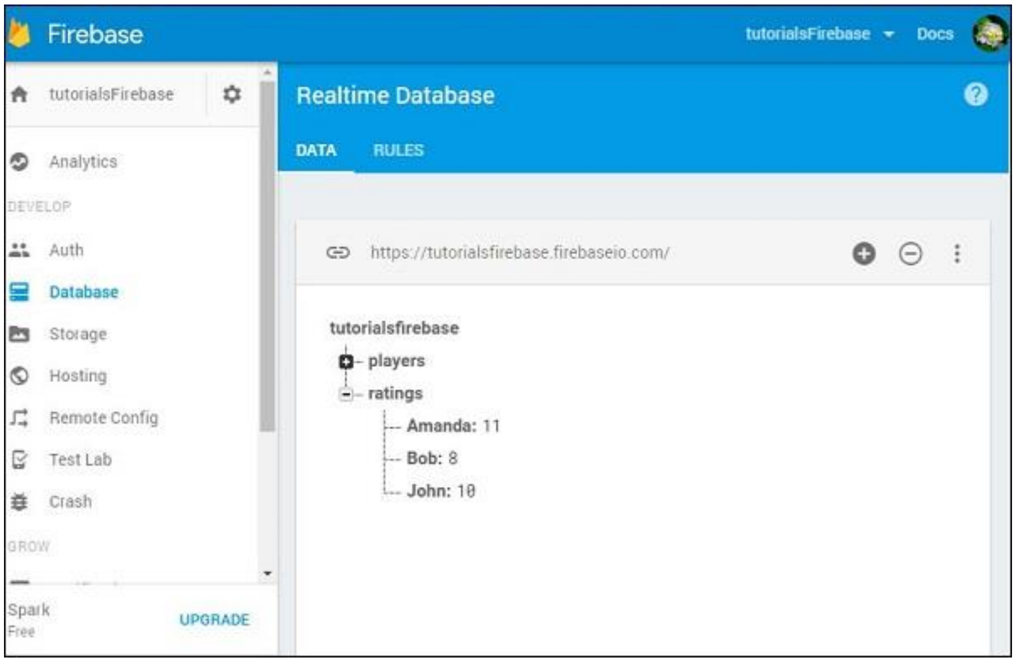
The output will be as shown below.



```
-KHzAeBSVzN8gjWecqNA                                          index.js:4
-KHzAeBXuVBJAqUXLKEo                                          index.js:4
```

Order by Value

We can also order data by value. Let us add the ratings collection in Firebase.



```
tutorialsfirebase
  players
  ratings
    Amanda: 11
    Bob: 8
    John: 10
```

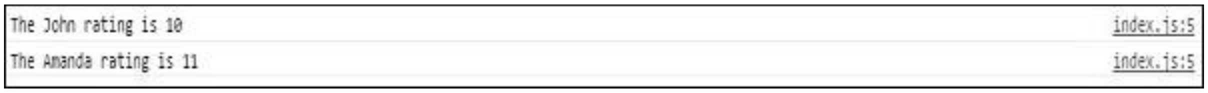
Now we can order data by value for each player.

Example

Let us consider the following example.

```
var ratingRef = firebase.database().ref("ratings/");


ratingRef.orderByValue().on("value", function(data) {


   data.forEach(function(data) {
      console.log("The " + data.key + " rating is " + data.val());
   });


});
```

The output will be as shown below.



```
The John rating is 10                                        index.js:5
The Amanda rating is 11                                      index.js:5
```

Firebase - Filtering Data

Firebase offers several ways to filter data.

Limit to First and Last

Let us understand what limit to first and last is.

- **limitToFirst** method returns the specified number of items beginning from the first one.
- **limitToLast** method returns a specified number of items beginning from the last one.

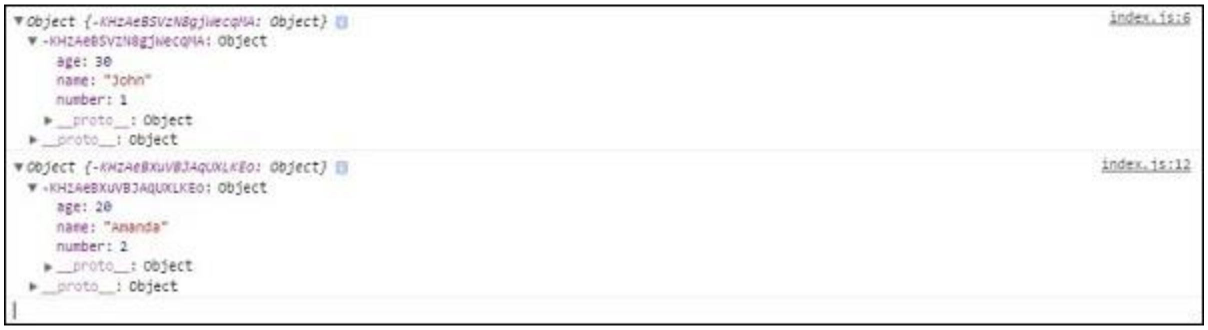
Our example is showing how this works. Since we only have two players in database, we will limit queries to one player.

Example

Let us consider the following example.

```
var firstPlayerRef = firebase.database().ref("players/").limitToFirst(1);

var lastPlayerRef = firebase.database().ref('players/').limitToLast(1);

firstPlayerRef.on("value", function(data) {
   console.log(data.val());
}, function (error) {
   console.log("Error: " + error.code);
});

lastPlayerRef.on("value", function(data) {
   console.log(data.val());
}, function (error) {
   console.log("Error: " + error.code);
});
```

Our console will log the first player from the first query, and the last player from the second query.



```
▼Object {-KHZAeBSVzN8gjWecQMA: Object}                                      index.js:6
  ▼-KHZAeBSVzN8gjWecQMA: Object
      age: 30
      name: "John"
      number: 1
    ▶__proto__: Object
  ▶__proto__: Object
▼Object {-KHZAeBXuVBJAQUXLKEo: Object}                                      index.js:12
  ▼-KHZAeBXuVBJAQUXLKEo: Object
      age: 20
      name: "Amanda"
      number: 2
    ▶__proto__: Object
  ▶__proto__: Object
```

Other Filters

We can also use other Firebase filtering methods. The **startAt()**, **endAt()** and the **equalTo()** can be combined with ordering methods. In our example, we will combine it with the **orderByChild()** method.

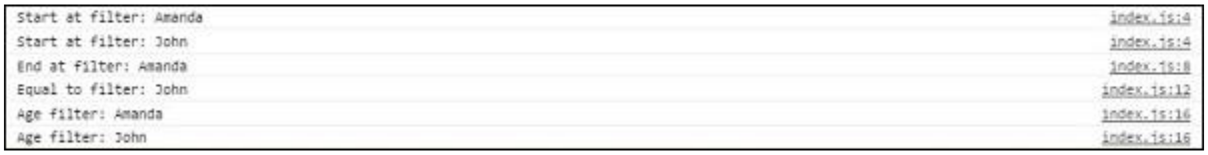
Example

Let us consider the following example.

```
var playersRef = firebase.database().ref("players/");

playersRef.orderByChild("name").startAt("Amanda").on("child_added", function(data) {
   console.log("Start at filter: " + data.val().name);
});

playersRef.orderByChild("name").endAt("Amanda").on("child_added", function(data) {
   console.log("End at filter: " + data.val().name);
});

playersRef.orderByChild("name").equalTo("John").on("child_added", function(data) {
    console.log("Equal to filter: " + data.val().name);
});

playersRef.orderByChild("age").startAt(20).on("child_added", function(data) {
    console.log("Age filter: " + data.val().name);
});
```

The first query will order elements by name and filter from the player with the name **Amanda**. The console will log both players. The second query will log "Amanda" since we are ending query with this name. The third one will log "John" since we are searching for a player with that name.

The fourth example is showing how we can combine filters with "age" value. Instead of string, we are passing the number inside the **startAt()** method since age is represented by a number value.



```
Start at filter: Amanda                                   index.js:4
Start at filter: John                                     index.js:4
End at filter: Amanda                                     index.js:8
Equal to filter: John                                     index.js:12
Age filter: Amanda                                        index.js:16
Age filter: John                                          index.js:16
```

Firebase - Best Practices

We will see the best practices of Firebase.

Avoid Nesting Data

When you fetch the data from Firebase, you will get all of the child nodes. This is why deep nesting is not said to be the best practice.

Denormalize Data

When you need deep nesting functionality, consider adding couple of different collections; even when you need to add some data duplication and use more than one request to retrieve what you need.

Firebase - Email Authentication

We will see how to use Firebase Email/Password authentication.

Create user

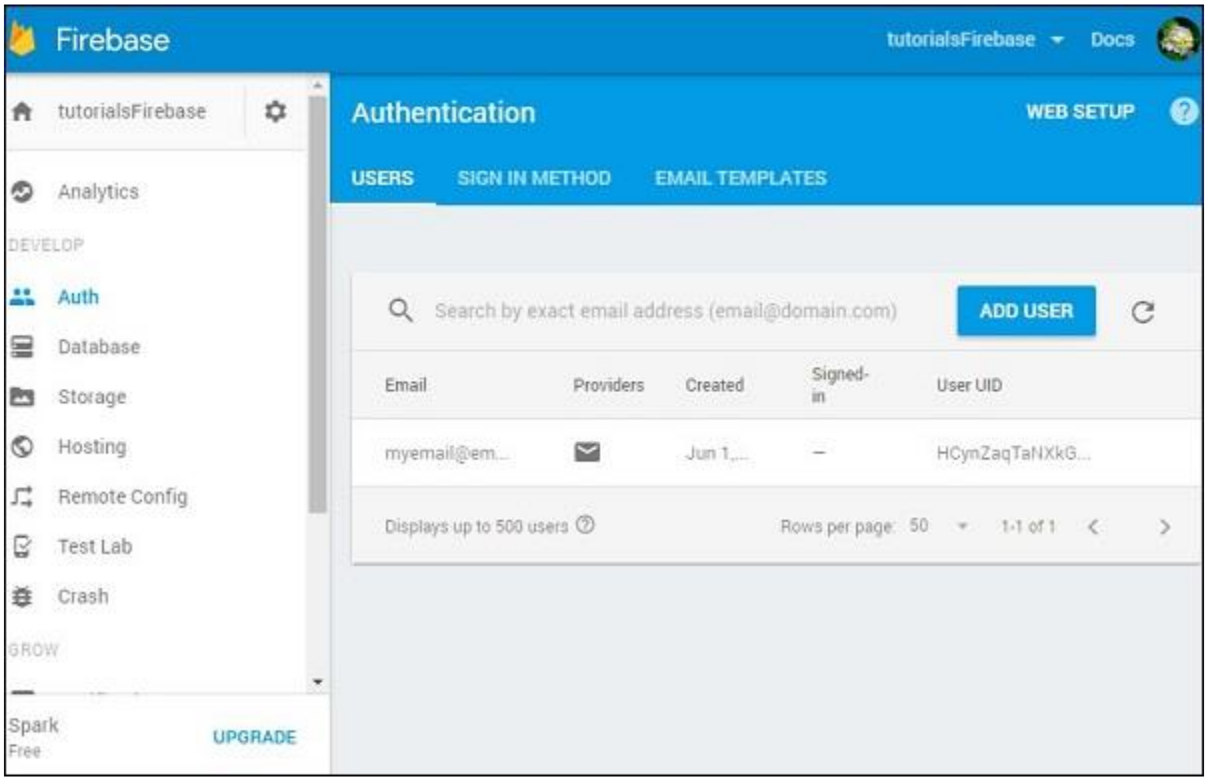
To authenticate a user, we can use the **createUserWithEmailAndPassword(email, password)** method.

Example

Let us consider the following example.

```
var email = "myemail@email.com";
var password = "mypassword";

firebase.auth().createUserWithEmailAndPassword(email, password).catch(function(error) {
   console.log(error.code);
   console.log(error.message);
});
```

We can check the Firebase dashboard and see that the user is created.



Email	Providers	Created	Signed-in	User UID
myemail@em...	✉	Jun 1,...	–	HCynZaqTaNXkG...

Sign In

The Sign-in process is almost the same. We are using the **signInWithEmailAndPassword(email, password)** to sign in the user.

Example

Let us consider the following example.

```
var email = "myemail@email.com";
var password = "mypassword";

firebase.auth().signInWithEmailAndPassword(email, password).catch(function(error) {
   console.log(error.code);
   console.log(error.message);
});
```

Signout

And finally we can logout the user with the **signOut()** method.

Example

Let us consider the following example.

```
firebase.auth().signOut().then(function() {
   console.log("Logged out!")
}, function(error) {
   console.log(error.code);
   console.log(error.message);
});
```

Firebase - Google Authentication

We will see how to set up Google authentication in Firebase.

Step 1 - Enable Google Authentication

Open Firebase dashboard and click **Auth** on the left side menu. To open the list of available methods, you need to click on **SIGN_IN_METHODS** in the tab menu.

Now you can choose **Google** from the list, enable it and save it.

Step 2 - Create Buttons

Inside our **index.html**, we will add two buttons.

index.html

```
<button onclick = "googleSignin()">Google Signin</button>
<button onclick = "googleSignout()">Google Signout</button>
```

Step 3 - Signin and Signout

In this step, we will create Signin and Signout functions. We will use **signInWithPopup()** and **signOut()** methods.

Example

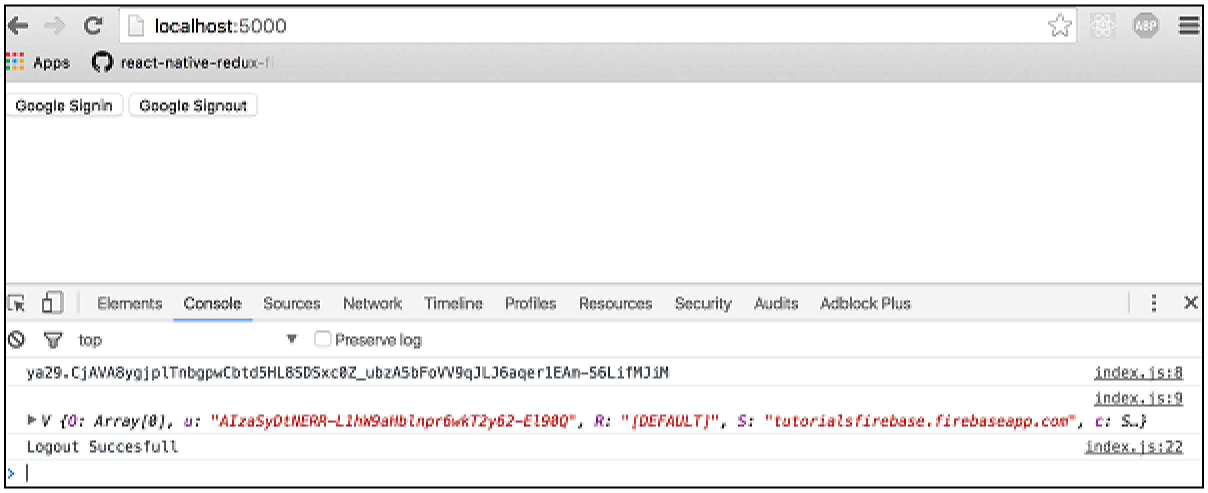
Let us consider the following example.

```
var provider = new firebase.auth.GoogleAuthProvider();

function googleSignin() {
   firebase.auth()
```

```
  .signInWithPopup(provider).then(function(result) {
    var token = result.credential.accessToken;
    var user = result.user;

    console.log(token)
    console.log(user)
  }).catch(function(error) {
    var errorCode = error.code;
    var errorMessage = error.message;

    console.log(error.code)
    console.log(error.message)
  });
}

function googleSignout() {
  firebase.auth().signOut()

  .then(function() {
    console.log('Signout Succesfull')
  }, function(error) {
    console.log('Signout Failed')
  });
}
```

After we refresh the page, we can click on the **Google Signin** button to trigger the Google popup. If signing in is successful, the developer console will log in our user.

We can also click on the **Google Signout** button to logout from the app. The console will confirm that the logout was successful.



```
ya29.CjAVA8ygjplTnbgpwCbtd5HL8SDSxc0Z_ubzA5bFoVV9qJLJ6aqer1EAm-S6LifMJiM        index.js:8
                                                                                  index.js:9
▶ V {O: Array[0], u: "AIzaSyDtNERR-L1hW9aHblnpr6wkT2y62-E198Q", R: "[DEFAULT]", S: "tutorialsfirebase.firebaseapp.com", c: S…}
Logout Succesfull                                                                 index.js:22
```

Firebase - Facebook Authentication

we will authenticate users with Firebase Facebook authentication.

Step 1 - Enable Facebook Auth

We need to open Firebase dashboard and click **Auth** in side menu. Next, we need to choose **SIGN-IN-METHOD** in tab bar. We will enable Facebook auth and leave this open since we need to add **App ID** and **App Secret** when we finish step 2.

Step 2 - Create Facebook App

To enable Facebook authentication, we need to create the Facebook app. Click on https://developers.facebook.com/ to start. Once the app is created, we need to copy **App ID** and **App Secret** to the Firebase page, which we left open in step 1. We also need to copy **OAuth Redirect URI** from this window into the Facebook app. You can find **+ Add Product** inside side menu of the Facebook app dashboard.

Choose **Facebook Login** and it will appear in the side menu. You will find input field **Valid OAuth redirect URIs** where you need to copy the **OAuth Redirect URI** from Firebase.

Step 3 - Connect to Facebook SDK

Copy the following code at the beginning of the **body** tag in **index.html**. Be sure to replace the **'APP_ID'** to your app id from Facebook dashboard.

Example

Let us consider the following example.

```
<script>
  window.fbAsyncInit = function() {
    FB.init ({
      appId      : 'APP_ID',
      xfbml      : true,
      version    : 'v2.6'
    });
  };

  (function(d, s, id) {
    var js, fjs = d.getElementsByTagName(s)[0];
    if (d.getElementById(id)) {return;}
    js = d.createElement(s); js.id = id;
    js.src = "//connect.facebook.net/en_US/sdk.js";
    fjs.parentNode.insertBefore(js, fjs);
  } (document, 'script', 'facebook-jssdk'));

</script>
```

Step 4 - Create Buttons

We set everything in first three steps, now we can create two buttons for login and logout.

index.html

```
<button onclick = "facebookSignin()">Facebook Signin</button>
<button onclick = "facebookSignout()">Facebook Signout</button>
```

Step 5 - Create Auth Functions

This is the last step. Open **index.js** and copy the following code.

index.js

```
var provider = new firebase.auth.FacebookAuthProvider();

function facebookSignin() {
   firebase.auth().signInWithPopup(provider)

   .then(function(result) {
      var token = result.credential.accessToken;
      var user = result.user;

      console.log(token)
      console.log(user)
   }).catch(function(error) {
      console.log(error.code);
      console.log(error.message);
   });
}

function facebookSignout() {
   firebase.auth().signOut()

   .then(function() {
      console.log('Signout successful!')
   }, function(error) {
      console.log('Signout failed')
   });
}
```

Firebase - Twitter Authentication

we will explain how to use Twitter authentication.

Step 1 - Create Twitter App

You can create Twitter app on this https://developers.facebook.com/ . Once your app is created click on **Keys and Access Tokens** where you can find **API Key** and **API Secret**. You will need this in step 2.

Step 2 - Enable Twitter Authentication

In your Firebase dashboard side menu, you need to click **Auth**. Then open **SIGN-IN-METHOD** tab. Click on Twitter to enable it. You need to add **API Key** and **API Secret** from the step 1.

Then you would need to copy the **callback URL** and paste it in your Twitter app. You can find the Callback URL of your Twitter app when you click on the **Settings** tab.

Step 3 - Add Buttons

In this step, we will add two buttons inside the **body** tag of **index.html**.

index.html

```
<button onclick = "twitterSignin()">Twitter Signin</button>
<button onclick = "twitterSignout()">Twitter Signout</button>
```

Step 4 - Authentication Functions

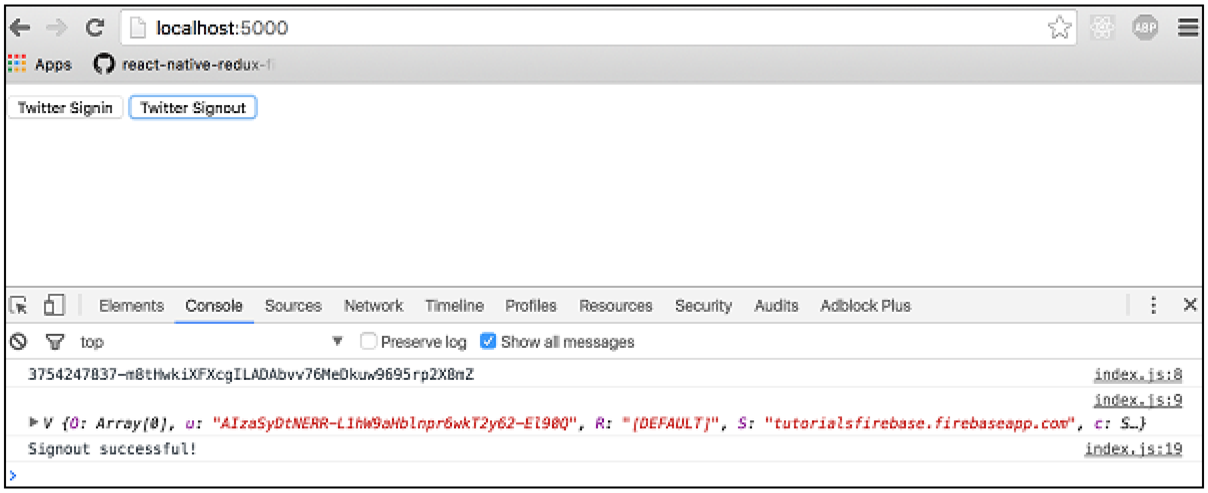
Now we can create functions for Twitter authentication.

index.js

```
var provider = new firebase.auth.TwitterAuthProvider();
```

```
function twitterSignin() {
   firebase.auth().signInWithPopup(provider)

  .then(function(result) {
      var token = result.credential.accessToken;
      var user = result.user;

      console.log(token)
      console.log(user)
   }).catch(function(error) {
      console.log(error.code)
      console.log(error.message)
   });
}

function twitterSignout() {
   firebase.auth().signOut()

   .then(function() {
      console.log('Signout successful!')
   }, function(error) {
      console.log('Signout failed!')
   });
}
```

When we start our app, we can sigin or signout by clicking the two buttons. The console will confirm that the authentication is successful.



```
3754247837-m8tHwkiXFXcgILADAbvv76MeDkuw9695rp2X8mZ                                              index.js:8
                                                                                                 index.js:9
▶ V {O: Array(0), u: "AIzaSyDtNERR-L1hW9aHblnpr6wkT2y62-El98Q", R: "[DEFAULT]", S: "tutorialsfirebase.firebaseapp.com", c: S…}
Signout successful!                                                                              index.js:19
>
```

Firebase - Github Authentication

We will see how to authenticate users using the GitHub API.

Step 1 - Enable GitHub Authentication

Open Firebase dashboard and click **Auth** from the side menu and then **SIGN-IN-METHOD** in tab bar. You need enable GitHub authentication and copy the **Callback URL**. You will need this in step 2. You can leave this tab open since you will need to add **Client ID** and **Client Secret** once you finish step 2.

Step 2 - Create Github App

Follow this https://github.com/settings/developers to create the GitHub app. You need to copy the **Callback URL** from Firebase into the **Authorization callback URL** field. Once your app is created, you need to copy the **Client Key** and the **Client Secret** from the GitHub app to Firebase.

Step 3 - Create Buttons

We will add two buttons in the **body** tag.

index.html

```
<button onclick = "githubSignin()">Github Signin</button>
<button onclick = "githubSignout()">Github Signout</button>
```

Step 4 - Create Auth Functions

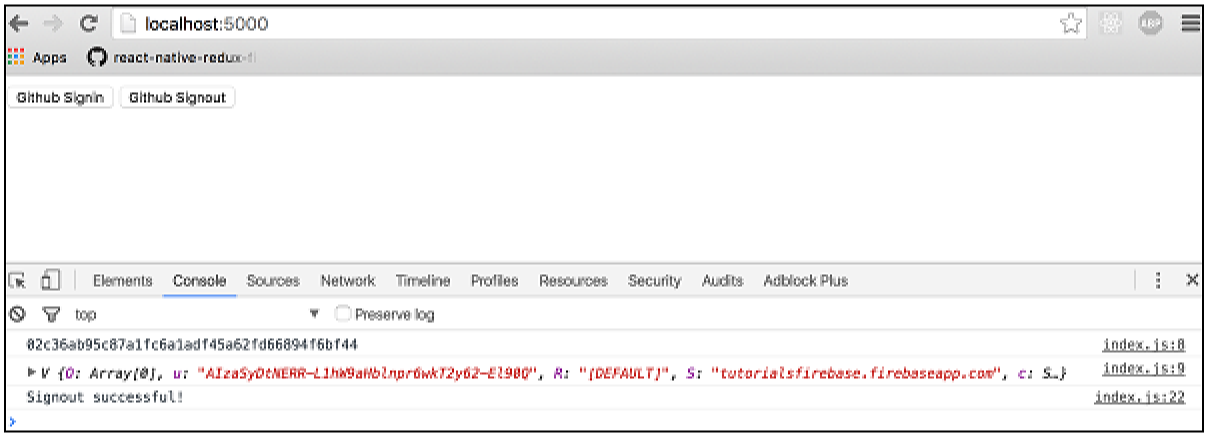
We will create functions for signin and signout inside the **index.js** file.

index.js

```
var provider = new firebase.auth.GithubAuthProvider();
```

```
function githubSignin() {
  firebase.auth().signInWithPopup(provider)

  .then(function(result) {
    var token = result.credential.accessToken;
    var user = result.user;

    console.log(token)
    console.log(user)
  }).catch(function(error) {
    var errorCode = error.code;
    var errorMessage = error.message;

    console.log(error.code)
    console.log(error.message)
  });
}

function githubSignout(){
  firebase.auth().signOut()

  .then(function() {
    console.log('Signout successful!')
  }, function(error) {
    console.log('Signout failed')
  });
}
```

Now we can click on buttons to trigger authentication. Console will show that the authentication is successful.



```
02c36ab95c87a1fc6a1adf45a62fd66894f6bf44                                                                                  index.js:8
▶ V {O: Array[0], u: "AIzaSyDtNERR-L1hW9aHblnpr6wkT2y62-El90Q", R: "[DEFAULT]", S: "tutorialsfirebase.firebaseapp.com", c: S…}   index.js:9
Signout successful!                                                                                                      index.js:22
```

Anonymous Authentication

We will authenticate users anonymously.

Step 1 - Enable Anonymous Auth

This is the same process as in our earlier chapters. You need to open the Firebase dashboard, click on **Auth** from the side menu and **SIGN-IN-METHOD** inside the tab bar. You need to enable anonymous authentication.

Step 2 - Signin Function

We can use **signInAnonymously()** method for this authentication.

Example

Let us consider the following example.

```
firebase.auth().signInAnonymously()
.then(function() {
   console.log('Logged in as Anonymous!')


   }).catch(function(error) {
   var errorCode = error.code;
   var errorMessage = error.message;
   console.log(errorCode);
   console.log(errorMessage);
});
```

Firebase - Offline Capabilities

We will see how to handle the Firebase connection state.

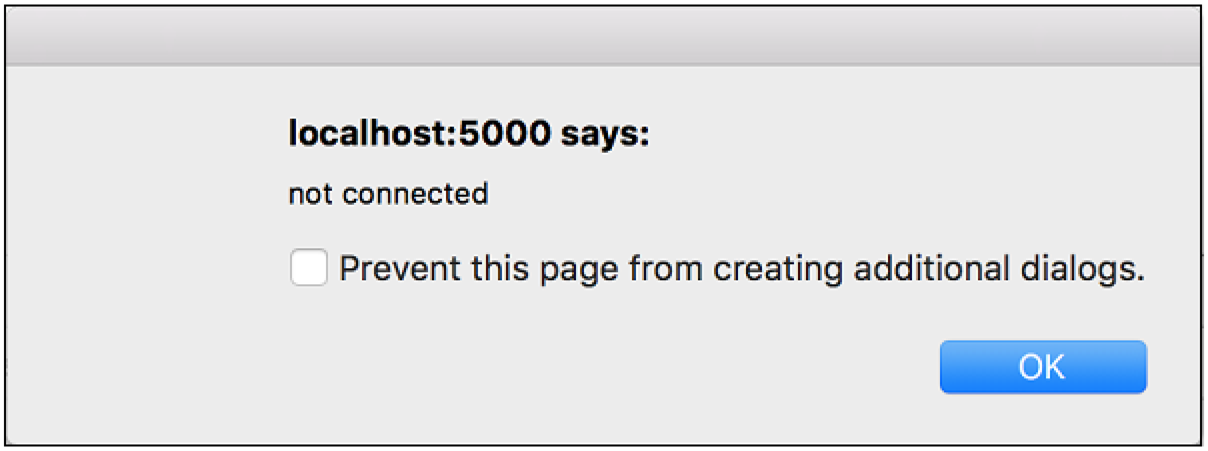
Check Connection

We can check for connection value using the following code.

index.js

```
var connectedRef = firebase.database().ref(".info/connected");

connectedRef.on("value", function(snap) {
  if (snap.val() === true) {
    alert("connected");
  } else {
    alert("not connected");
  }
});
```

When we run the app, the pop up will inform us about the connection.



localhost:5000 says:

not connected

By using the above given function, you can keep a track of the connection state and update your app accordingly.

Firebase – Security

Security in Firebase is handled by setting the JSON like object inside the security rules. Security rules can be found when we click on **Database** inside the side menu and then **RULES** in tab bar.

In this chapter, we will go through a couple of simple examples to show you how to secure the Firebase data.

Read and Write

The following code snippet defined inside the Firebase security rules will allow writing access to **/users/'$uid'/** for the authenticated user with the same **uid**, but everyone could read it.

Example

Let us consider the following example.

```
{
  "rules": {
    "users": {

      "$uid": {
        ".write": "$uid === auth.uid",
        ".read": true
      }

    }
  }
}
```

Validate

We can enforce data to string by using the following example.

Example

```
{
  "rules": {

    "foo": {
      ".validate": "newData.isString()"
    }

  }
}
```

This chapter only grabbed the surface of Firebase security rules. The important thing is to understand how these rules work, so you can combine it inside the app.

Firebase – Deploying

We will see how to host your app on the Firebase server.

Before we begin, let us just add some text to **index.html** body tag. In this example, we will add the following text.

```
<h1>WELCOME TO FIREBASE TUTORIALS APP</h1>
```

Step 1 - Install Firebase Tools

We need to install firebase tools globally in the **command prompt** window.

```
npm install -g firebase-tools
```

Step 2 - Initialize the Firebase App

First we need to login to Firebase in the **command prompt**.

```
firebase login
```

Open the root folder of your app in the **command prompt** and run the following command.

```
firebase init
```

This command will initialize your app.

NOTE − If you have used a default configuration, the **public** folder will be created and the **index.html** inside this folder will be the starting point of your app. You can copy your app file inside the public folder as a workaround.

Step 3 - Deploy Firebase App

This is the last step . Run the following command from the **command prompt** to deploy your app.

```
firebase deploy
```

Firebase Useful Resources

The following resources contain additional information on Firebase. Please use them to get more in-depth knowledge on this topic.

Useful Links on Firebase

Firebase Wiki – https://en.wikipedia.org/wiki/Firebase Wikipedia Reference for Firebase.

Firebase Official Site – **https://firebase.google.com/** Official Site of Firebase.

Firebase - Interview question and answer

1) What is Firebase?

Firebase is a platform which is used for building Web, IOS and Android applications.

It offers:

- Real time database
- Different APIs
- Multiple authentication types
- and Hosting platform

2) What are the features of Firebase?

Features of firebase are:

- Hosting
- Authentication
- Real-time Database

3) Who is the Founder of Firebase?

James Tamplin is the founder of Firebase.

4) What is Push Method in Firebase?

The **push() method** is used, When the data is pushed.

The push() method will create a unique id.

5) What is Key Method in Firebase?

By using the **key() method**, We can get any key from Firebase.

6) What is Set Method in Firebase?

The Set Method is used to write or replace data on a specified method.

7) What is Transactional data in Firebase?

The Transactional data is used to return some data from the database.

8) What is limitToFirst method in Firebase?

It is used to returns the specified number of items beginning from the first one.

9) What is limitToLast method in Firebase?

It is used to returns a specified number of items beginning from the last one.

10) What are the advantages of Firebase?

Features of Firebase are:

- It offers simple control dashboard
- Number of useful services to choose
- Here, the data is real time which means that every change will update automatically
- It is user friendly and simple

11) What are the filtering methods in Firebase?

Firebase filtering methods are:

- startAt() method
- endAt() method
- equalTo() method etc

12) What are the different types of event types for reading data in Firebase?

There are several types of event types are:

value

child_added

child_changed

child_removed

13) What is off() method in Firebase?

The **off() method** is used to remove all callbacks with value event type.

14) Which method is used to create a user in Firebase Email/Password authentication?

createUserWithEmailAndPassword(email, password) method.

15) Which method is used to Sign-in the user in Firebase Email/Password authentication?

signInWithEmailAndPassword(email, password) method.

16) What is the use of on() method in Firebase?

on() method: This method is used to retrieve data.

17) What is the use of child_changed method?

child_changed: This is an event type which is triggered when the data has changed.

18) What is the use of child_removed method?

child_removed: This is an event type which is triggered when we want to remove data.

19) What is a Firebase Project?

Firebase Project: Firebase project is a container for apps.

Firebase project supports many features such as Database, Config and Notifications between your cross-platform apps.

20) What is the use of Sha-1 in Firebase?

Sha-1: Sha-1 is only required if you are using either Firebase Dynamic Links or Firebase Invites. It is used to simplify the Google Sign-In configuration with Firebase Authentication.