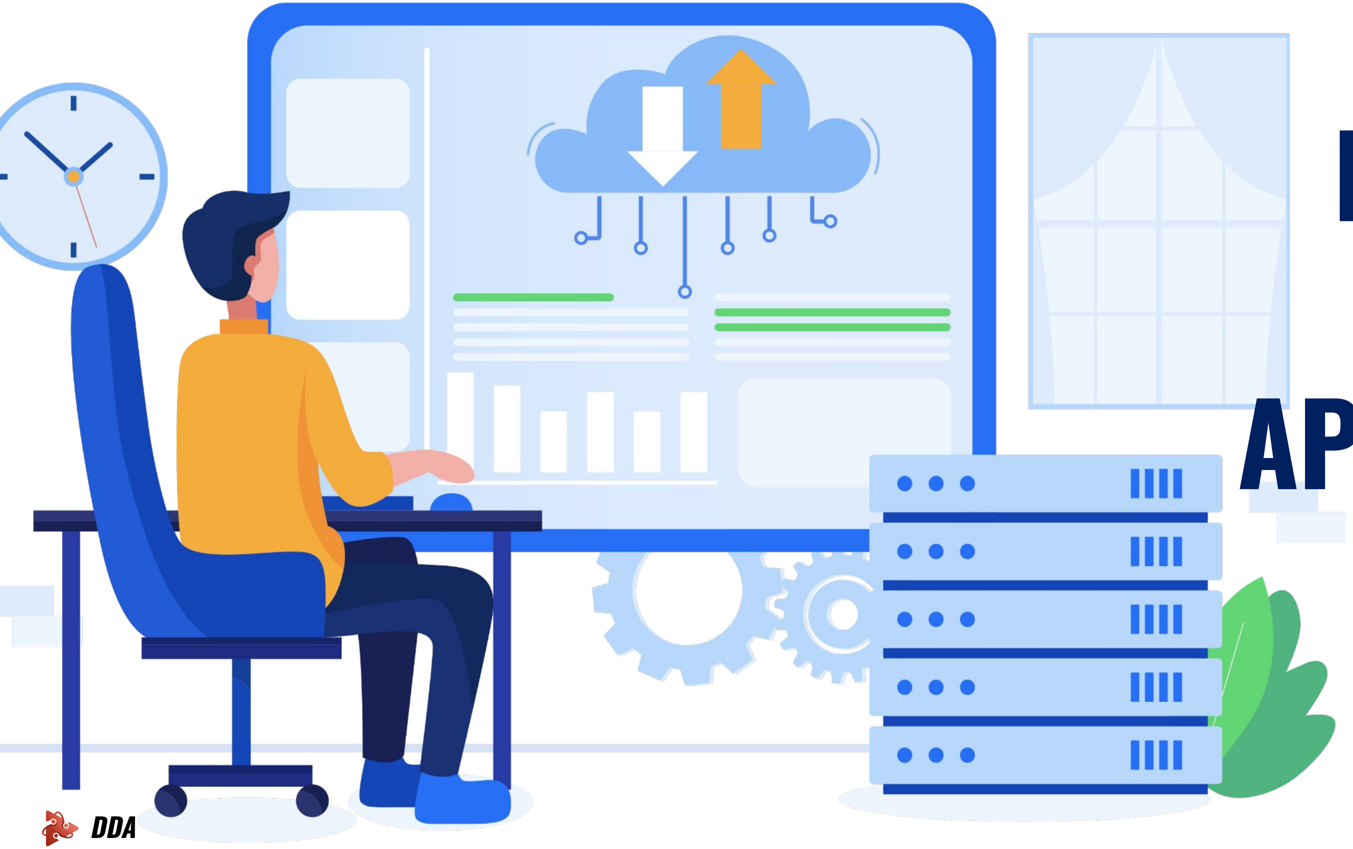
WEEK 7



(DDA) 2.2

DEVELOPING DYNAMIC APPLICATIONS

2021

DDA

Lesson 6: Advanced Firebase w/ Web & Dashboarding

DDA

COVERAGE

Week 7

Learning Objectives

1. OOP Recap & Web Recap
2. CRUD via Web & Unity
3. Indexing in Firebase for Efficiency
4. Web Dashboarding w/ Firebase & CRUD

*Note: We are **using Firebase v9.***

We are NOT using Firebase v8

OOP

Object Oriented Programming

● OBJECT
● KEY / NAME
● VALUE

LITERAL OBJECTS

```
let hotel  = {

  name: 'Raffles Hotel',
  rooms:  100,
  booked: 24,
  gym: true,
  roomTypes:
['twin','suite','delux'],

  checkAvailability: function() {
    return this.rooms - this.booked;
  }
};
```

PROPERTIES
These are variables

METHOD
This is a function

Object is the curly braces {… } and its contents
Object stored in a variable **hotel**

Separate each key from its value using a colon
Separate each property and method with a comma

The this keyword in checkAvailability() method,
References the rooms and **booked** projects of the Object (hotel)

https://repl.it/@malcolmyam/wk06-objects#script.js

ACCESSING AN OBJECT

OBJECT

PROPERTY/METHOD NAME

```
let hotelName = hotel.name;
let roomsFree = hotel.checkAvailability();
```

MEMBER OPERATOR

```
let hotelName = hotel['name'];
let roomsFree = hotel['checkAvailability']();
```

https://repl.it/@malcolmyam/wk06-objects#script.js

UPDATING AN OBJECT

OBJECT PROPERTY NAME PROPERTY VALUE

```
hotel.name = 'Favcho Royale Hotel';
```

MEMBER OPERATOR ASSIGNMENT OPERATOR

*** Note:*** If the object does not have the property you are trying to update, it will be added to the object

```
hotel['name'] = 'Favcho Royale Hotel';
```

```
delete hotel.name;
hotel.name = '';
```

// Delete a property using the **delete** keyword

// Clear the value of a property by assigning a blank string

● OBJECT
● KEY / NAME
● VALUE

CONSTRUCTOR NOTATION

```
let hotel = new Object();

hotel.name = 'Raffles Hotel';
hotel.rooms = 100;
hotel.booked = 24;
hotel.gym = true;
hotel.roomTypes =
['twin','suite','delux'];

hotel.checkAvailability: function() {
  return this.rooms - this.booked;
}
```

PROPERTIES

METHOD

Create an object using the "new" keyword and the **Object()** constructor (Blank object)

Add properties, methods to the newly created blank object

● OBJECT
● KEY / NAME
● VALUE

FUNCTION BASED OBJECTS

PARAMETERS

```
function Hotel(name, rooms, booked){

  this.name = name;
  this.rooms = rooms;
  this.booked = 24;

  this.checkAvailability = function()
{
    return this.rooms - this.booked;
  };
}
```

PROPERTIES

METHOD

Creating a function Hotel allows it to be used as a template for creating multiple objects

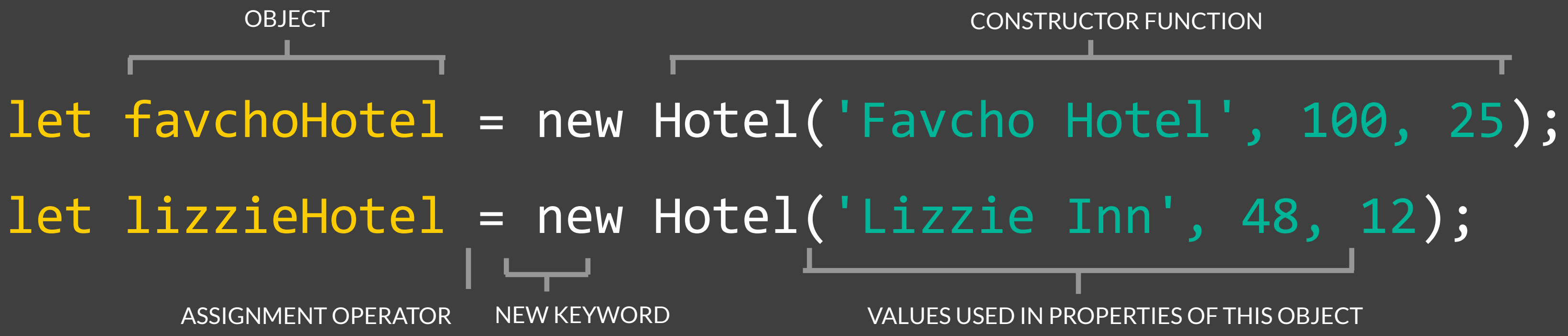
This is called a function based object.

The **this** keyword is used instead of the object name to indicate the property/method belongs to the object that **this** function creates.

Each statement creates a new property or method. Uses **semi-colon** instead of comma (literal object syntax)

https://repl.it/@malcolmyam/wk06-objects#script.js

MULTIPLE OBJECT INSTANCES



```
let favchoHotel = new Hotel('Favcho Hotel', 100, 25);
let lizzieHotel = new Hotel('Lizzie Inn', 48, 12);
```

The first object **favchoHotel**. Name is "Favcho Hotel".
100 rooms, 25 booked

The second object **lizzieHotel**. Name is "Lizzie Inn".
48 rooms, 12 booked

*** *Note:*** Even when multiple objects are created using the same constructor function, the methods stay the same.

https://repl.it/@malcolmyam/wk06-objects#script.js

ACTIVITY

"Objects" (30min) - GROUP Work

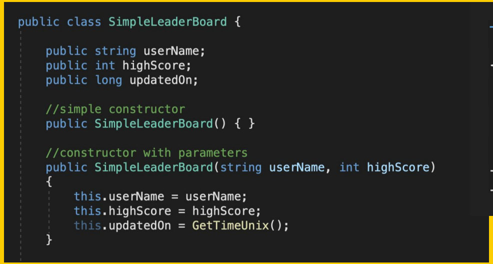
Q1. Working with Literal Objects
Based on your assignment 1, create literal objects of your class models.

Create appropriate literal objects and for each property, indicate the appropriate data types to use.

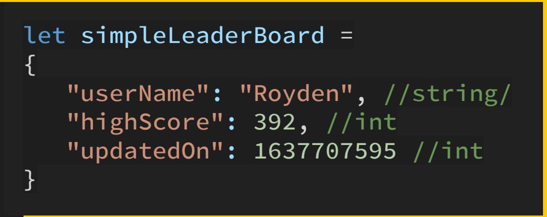
Q2. Working with Function Based Objects
Based on your assignment 1, create function based objects of your class models, **indicate the appropriate data types to use.**

Create sample object variables based on your models
You may assume your own property values.

Eg. converting SimpleLeaderBoard



```
public class SimpleLeaderBoard {

    public string userName;
    public int highScore;
    public long updatedOn;

    //simple constructor
    public SimpleLeaderBoard() { }

    //constructor with parameters
    public SimpleLeaderBoard(string userName, int highScore)
    {
        this.userName = userName;
        this.highScore = highScore;
        this.updatedOn = GetTimeUnix();
    }
```



```
let simpleLeaderBoard =
{
    "userName": "Royden", //string/
    "highScore": 392, //int
    "updatedOn": 1637707595 //int
}
```

Select the best structured data in your team
Submit your JavaScript files and your C# class models
//save as DDAWk06-studentid-studentname-objects.zip
Upload to Google Classroom

https://replit.com/@malcolmyam/firebase-json-example

Firebase & Web

#FirebaseWeb

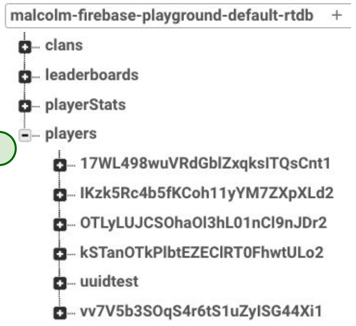
Connect with Firebase via JS (Read Data)

1. Create a .html file
2. Create a JS file and link it up. Place in your imports and get the database reference, path reference
3. Start with the skeleton JS script that Firebase provides
4. Let's retrieve data using **get** and **ref**

Note: How we reference nodes/data depends on how you structure your data inside Firebase Database

Do not have function names that are the same as your imports

```
import {
 getDatabase, ref, child, get
} from "https://www.gstatic.com/firebasejs/9.5.0/firebase-database.js";

const db = getDatabase();                    1
const playerRef = ref(db, "players"); //refers to the path in db


getPlayerData();
function getPlayerData(){
   //const playerRef = ref(db, "players");
   //PlayerRef is declared at the top using a constant
   //get(child(db,`players/`))
   get(playerRef)
   .then((snapshot) => {//retrieve a snapshot of the data using a callback
   if (snapshot.exists()) {//if the data exist
       try {
           var content = "";
           snapshot.forEach((childSnapshot) => {//looping through each snapshot

//https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forE
ach
               console.log("GetPlayerData: childkey " + childSnapshot.key);
           });
       }catch(error){
           console.log("Error getPlayerData" + error);
       }
   }
   });
}//end getPlayerData
```



malcolm-firebase-playground-default-rtdb +
- clans
- leaderboards
- playerStats
- players 1
  - 17WL498wuVRdGblZxqksITQsCnt1
  - IKzk5Rc4b5fKCoh11yYM7ZXpXLd2
  - OTLyLUJCSOhaOl3hL01nCl9nJDr2
  - kSTanOTkPlbtEZEClRT0FhwtULo2
  - uuidtest
  - vv7V5b3SOqS4r6tS1uZyISG44Xi1

Connect with Firebase via JS (Create User)

1. Start with the skeleton JS script that Firebase provides
2. Create a form using inputs and a button
3. Create your button listener
4. Test retrieve your form values
5. Let's create auth data using **createUserWithEmailAndPassword** and **userCredential**

Note: How we reference nodes/data depends on how you structure your data inside Firebase Database

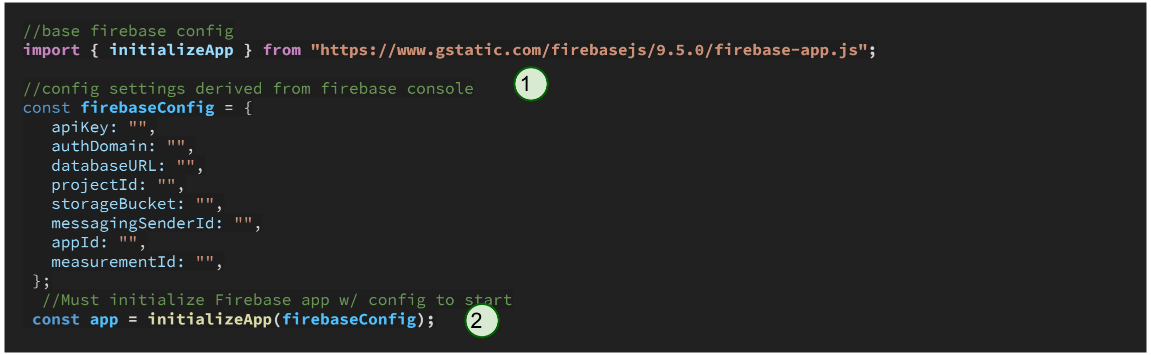
Do not have function names that are the same as your imports

```
import {
 getAuth,
 createUserWithEmailAndPassword,
 signInWithEmailAndPassword,
} from "https://www.gstatic.com/firebasejs/9.5.0/firebase-auth.js";

//Working with Auth
const auth = getAuth();
//retrieve element from form
var frmCreateUser = document.getElementById("frmCreateUser");
//we create a button listener to listen when someone clicks
frmCreateUser.addEventListener("submit", function(e){
 e.preventDefault();
  var email = document.getElementById("email").value;
 var password = document.getElementById("password").value;
 createUser(email, password);
 console.log("email" + email + "password" + password);
});

//create a new user based on email n password into Auth service
//user will get signed in
//userCredential is an object that gets
function createUser(email, password){
 console.log("creating the user");
 createUserWithEmailAndPassword(auth, email, password)
 .then((userCredential)=>{
   //signedin
   const user = userCredential.user;
   console.log("created user ... " + JSON.stringify(userCredential));
   console.log("User is now signed in ");
 }).catch((error)=>{
   const errorCode = error.code;
   const errorMessage = error.message;
   console.log(`ErrorCode: ${errorCode} -> Message: ${errorMessage}`);
 });
}
```

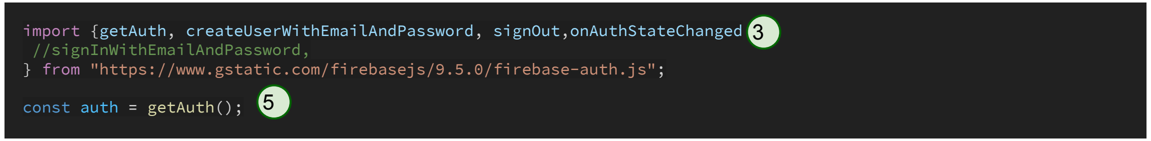
How to Setup your Firebase & Web + Forms

1. Start with the Firebase config settings and place into a "**config.js**" JS script that can be found in your Firebase console -> project settings (Web)
2. **Initialize your app** based on the **firebase config**
3. Create another **"main.js" JS file that contains your necessary Firebase imports**
4. Create your UI using HTML
5. Add on logic to your "main.js" to r**etrieve the auth object or database object from Firebase**
6. Add Event Listeners that listen to your form
   Remember: to place in necessary error handlers

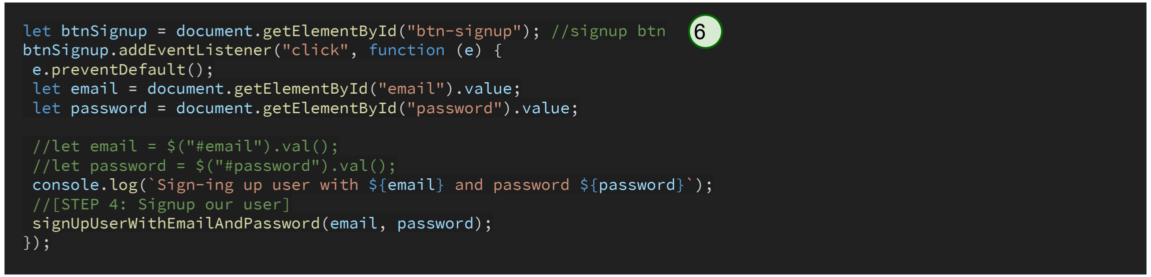
```
/project
--/js            1
  |-- config.js
  |-- main.js    3
signup.html
```



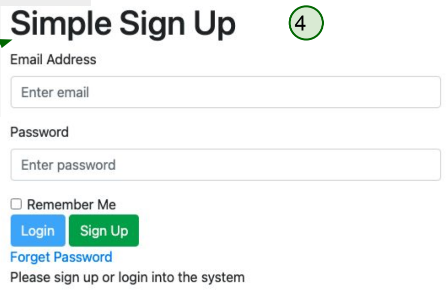
```
//base firebase config
import { initializeApp } from "https://www.gstatic.com/firebasejs/9.5.0/firebase-app.js";

//config settings derived from firebase console          1
const firebaseConfig = {
   apiKey: "",
   authDomain: "",
   databaseURL: "",
   projectId: "",
   storageBucket: "",
   messagingSenderId: "",
   appId: "",
   measurementId: "",
 };
  //Must initialize Firebase app w/ config to start
 const app = initializeApp(firebaseConfig);   2
```



```
import {getAuth, createUserWithEmailAndPassword, signOut,onAuthStateChanged 3
 //signInWithEmailAndPassword,
} from "https://www.gstatic.com/firebasejs/9.5.0/firebase-auth.js";

const auth = getAuth(); 5
```



```
let btnSignup = document.getElementById("btn-signup"); //signup btn    6
btnSignup.addEventListener("click", function (e) {
 e.preventDefault();
 let email = document.getElementById("email").value;
 let password = document.getElementById("password").value;

 //let email = $("#email").val();
 //let password = $("#password").val();
 console.log(`Sign-ing up user with ${email} and password ${password}`);
 //[STEP 4: Signup our user]
 signUpUserWithEmailAndPassword(email, password);
});
```



Simple Sign Up 4

Email Address

Enter email

Password

Enter password

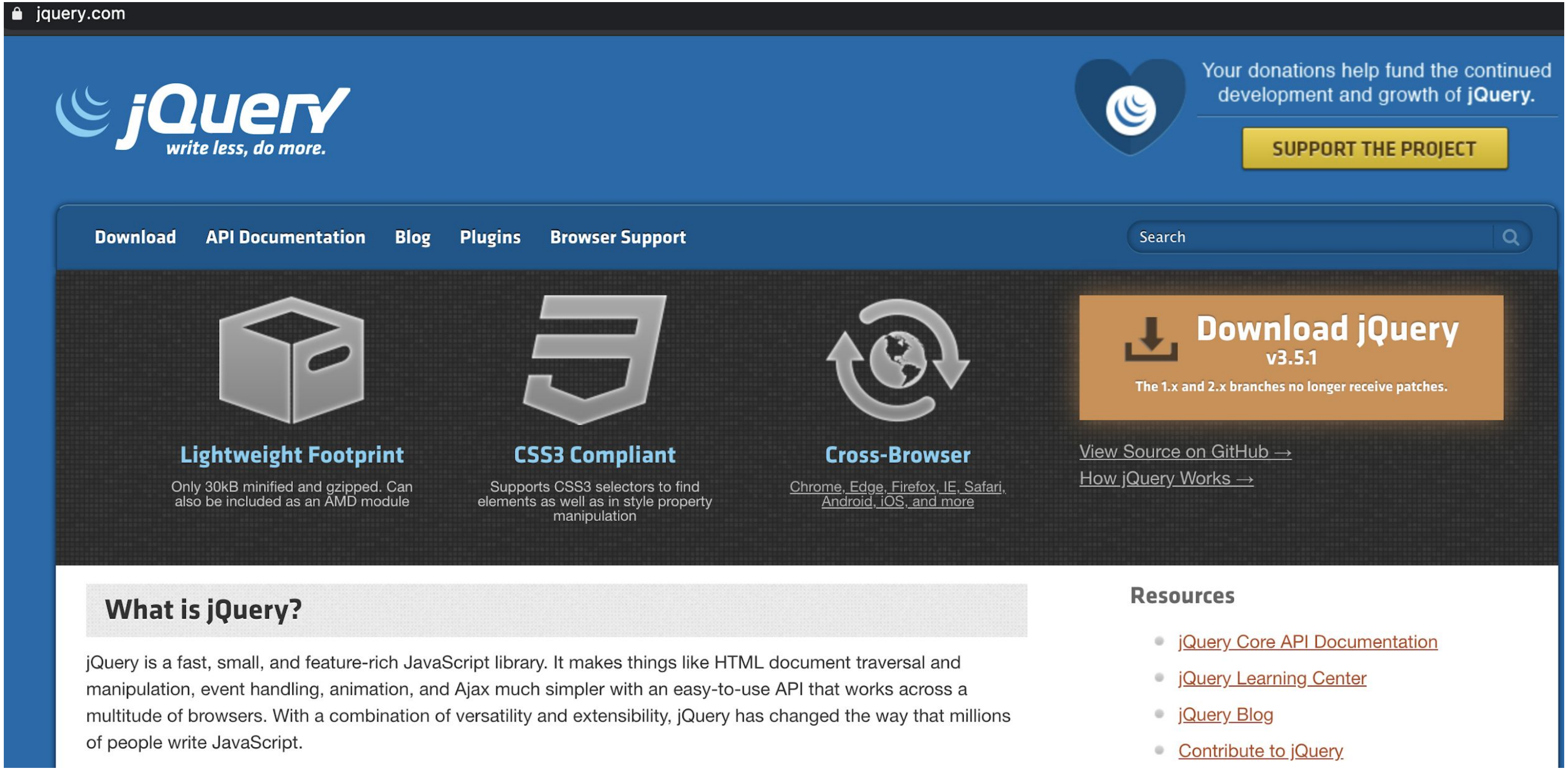
Remember Me

Login | Sign Up

Forget Password

Please sign up or login into the system

Working with #jQuery #Recap



Lightweight Footprint

Only 30kB minified and gzipped. Can also be included as an AMD module

CSS3 Compliant

Supports CSS3 selectors to find elements as well as in style property manipulation

Cross-Browser

Chrome, Edge, Firefox, IE, Safari, Android, iOS, and more

Download jQuery
v3.5.1
The 1.x and 2.x branches no longer receive patches.

View Source on GitHub →
How jQuery Works →

What is jQuery?

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

Resources

- jQuery Core API Documentation
- jQuery Learning Center
- jQuery Blog
- Contribute to jQuery

https://jquery.com/

CSS-STYLE SELECTOR

```
jQuery('li.hot');
```

FUNCTION

SHORTHAND FOR JQUERY FUNCTION

```
$('li.hot');
```

NORMAL JS

```
document.querySelectorAll('li.hot');
```

https://repl.it/@malcolmyam/simple-jquery-demo#script.js

HIDE ALL MATCHING LIST ITEMS

$$\texttt{\$('li.hot')}\underbrace{\texttt{.hide()}}_{\text{METHOD OF THE JQUERY OBJECT}}\texttt{;}$$

This is roughly equivalent to calling `.css( "display", "none" )`

https://api.jquery.com/hide/

Selecting by HTML Tags

$("div") – selects all <div> elements
$("li") – selects all <li> elements
$("p") – selects all <p> elements

We have to select by CSS to narrow down

NORMAL JS

```
document.getElementsByTagName('div');
document.querySelectorAll('div');
```

https://repl.it/@malcolmyam/simple-jquery-demo#script.js

Selecting by CSS/ ID

$("#page") = select any with id of 'page'
$(".hot") = select any with class of 'hot'
$("li.cool") = select **only** li with class of 'cool'
$("li.cool a") = select **only** <a> that are inside <li> with class of 'cool'

NORMAL JS

```
document.getElementById('page');
document.querySelectorAll('li.cool a');
```

https://repl.it/@malcolmyam/simple-jquery-demo#script.js

SETTING HTML CONTENT

This example will replace the content of each list item with the word **Updated** in <b> tags. This includes HTML markup

```
$('li').html('<b>Updated</b>');
```

NORMAL JS

```
document.getElementsByTagName('li').innerHTML
```

SETTING TEXT CONTENT

This example will replace the text content of each list item with the word **Updated**.

```
$('li').text('Updated');
```

NORMAL JS

```
document.getElementsByTagName('li').textContent
```

https://repl.it/@malcolmyam/simple-jquery-demo#script.js

Get & Set

Attributes are those "x=y" inside tags
eg: <a **href**="…">, href is an attribute

```
$('a').attr('href') -- retrieve the value of href
$('a').attr('href', 'newvalue') -- set the value of
the href attribute
```

https://repl.it/@malcolmyam/simple-jquery-demo#script.js

Loading JS when page is ready

Use $(document).ready();

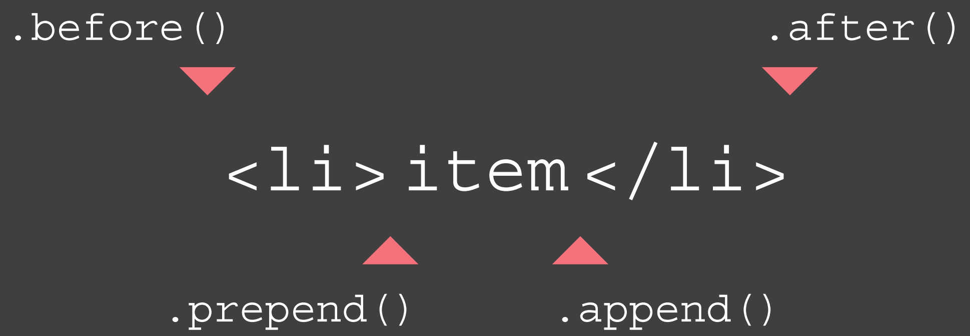
```
$(document).ready(function() {
   console.log("Hello World!");
});
```

```
$(function() {  //shorthand version
   console.log( "ready!" );
});
```

Why we are doing this?
HTML elements must be loaded and “safe” before we can safely manipulate them
$(document).ready() allows us to safely detect that the page is all ready for us to use, then we can execute our JS code

Further Reading:
https://learn.jquery.com/using-jquery-core/document-ready/

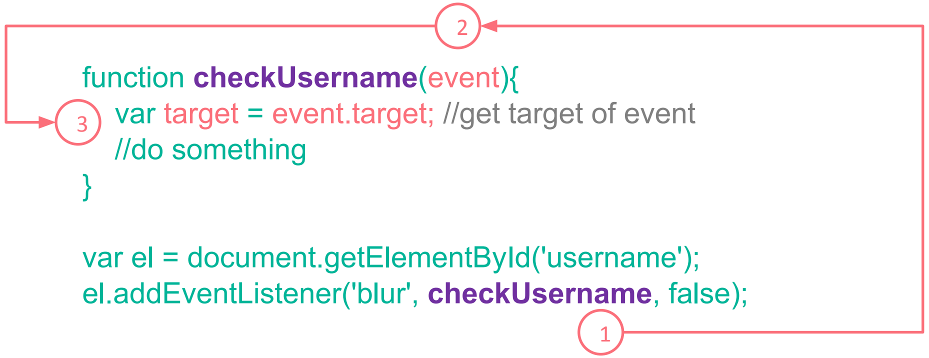
INSERT ELEMENTS



DEMO: https://repl.it/@malcolmyam/wk08-jquery-insertelements#script.js

Working with #EventHandlers

EVENT LISTENER WITH NO PARAMETERS (A)

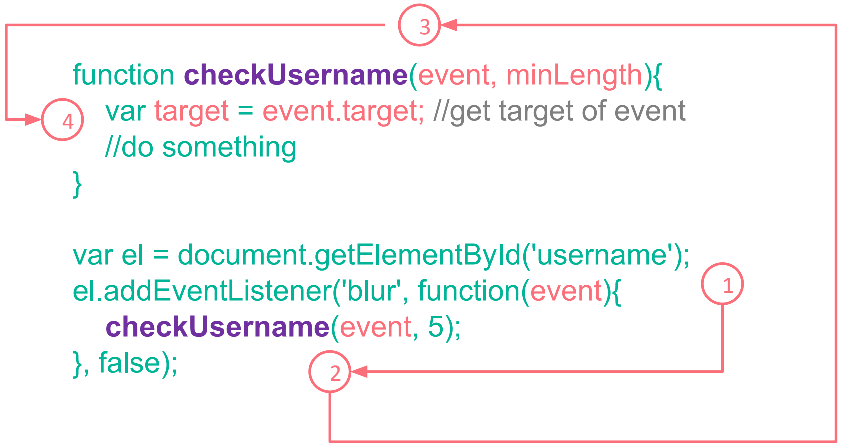


```
function checkUsername(event){
   var target = event.target; //get target of event
   //do something
}

var el = document.getElementById('username');
el.addEventListener('blur', checkUsername, false);
```

jQuery

```
$("#username").on("blur", function(event){
   checkUsername(event)
});
```

EVENT LISTENER WITH PARAMETERS (B)



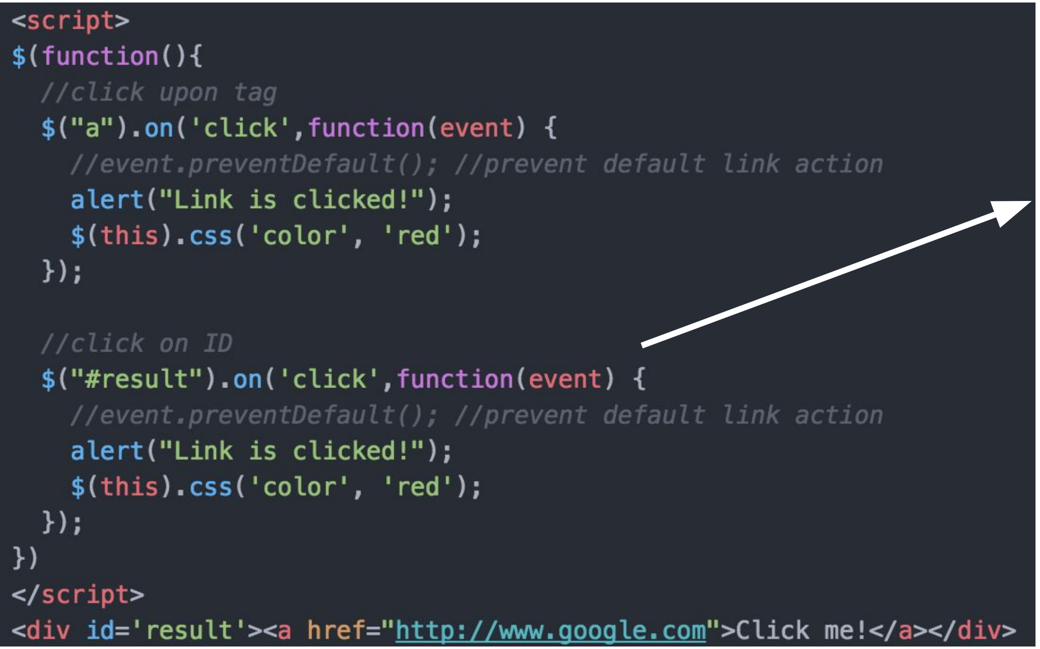
```
function checkUsername(event, minLength){
   var target = event.target; //get target of event
   //do something
}

var el = document.getElementById('username');
el.addEventListener('blur', function(event){
   checkUsername(event, 5);
}, false);
```

jQuery

```
$("#username").on("blur", function(e) {
      checkUsername(e, 5)
});
```

https://repl.it/@malcolmyam/wk07-sample-username-check#index.html

Refer to activity/js/event-listener-with-event-object.js
https://repl.it/@malcolmyam/wk07-activity-materials#index.html



```
<script>
$(function(){
  //click upon tag
  $("a").on('click',function(event) {
    //event.preventDefault(); //prevent default link action
    alert("Link is clicked!");
    $(this).css('color', 'red');
  });

  //click on ID
  $("#result").on('click',function(event) {
    //event.preventDefault(); //prevent default link action
    alert("Link is clicked!");
    $(this).css('color', 'red');
  });
})
</script>
<div id='result'><a href="http://www.google.com">Click me!</a></div>
```

jQuery (Equivalent)

```
$("#result").click(function(e) {
    alert("Link is clicked!");
    $(this).css('color','red');
});
```

Sample source of event click using jQuery:
https://repl.it/@malcolmyam/wk08-simpleclick-on#script.js

https://api.jquery.com/category/events/mouse-events/
https://api.jquery.com/on/
https://api.jquery.com/click/
https://api.jquery.com/on/#additional-notes

SELECTING FORM ELEMENT

By assigning a id or a class (like normal HTML)
By selecting by name:

```
// select all input with the name "username"
$("input[name=username]")
```

RETRIEVE FORM VALUE

The value is what is stored in the input element

Use `.val()` to retrieve:

```
$("#username").val()
```

Use the same method, but pass in a parameter to set

```
$("#username").val("leeroy");
```

Detect change in select dropdown

Use the `.change()` event

```
$("#select").change(function(e){
 console.log("You have changed to " + $(this).value());
})
```

If you want to get the text of the option

```
$("#select").change(function(e){
 console.log("You have changed to " +
  $(this).children(":selected").text());
})
```

Working with #jQueryEffects

EFFECTS Used to enhance page with transitions and movements

Basic Effects

METHOD	DESCRIPTION
.show()	Displays selected elements
.hide()	Hides selected elements
.toggle()	Toggles between showing and hiding selected elements

Fading Effects

METHOD	DESCRIPTION
.fadeIn()	Fades in selected elements making them opaque
.fadeOut()	Fades out selected elements making them transparent
.fadeTo()	Changes opacity of selected element
.fadeToggle()	Hides or shows selected elements by changing their opacity (opposite of current state)

Further Reference Reading:
https://api.jquery.com/category/effects/
https://api.jquery.com/animate/

EFFECTS Used to enhance page with transitions and movements

Sliding Effects

METHOD	DESCRIPTION
.slideUp()	Hides selected elements with a sliding motion
.slideDown()	Shows selected elements with a sliding motion
.slideToggle()	Hides or show selected elements with a sliding motion (opposite direction of current state)

Custom Effects

METHOD	DESCRIPTION
.delay()	Delays execution of subsequent items in queue
.stop()	Stops an animation if it is currently running
.animate()	Creates custom animation

Further Reference Reading:
https://api.jquery.com/category/effects/
https://api.jquery.com/animate/

CRUD
#READ

CRUD - **READING** Data

Dealing with CRUD is similar to how it is done in Unity.

Some differences includes dealing with the Firebase imports from the SDK and the HTML interface.

Retrieve number of child nodes

```
Snapshot.size;
```

Convert Timestamp to Dates

```
new Date(childSnapshot.child("lastLoggedIn").val() * 1000)
```

https://coderrocketfuel.com/article/convert-a-unix-timestamp-to-a-date-in-vanilla-javascript

Reading Reference
https://firebase.google.com/docs/reference/js/database
https://firebase.google.com/docs/reference/js/database.datasnapshot.md#datasnapshotsize

```
import {getDatabase, ref, child, get, set, onValue, orderByChild} from
"https://www.gstatic.com/firebasejs/9.5.0/firebase-database.js";

//[STEP 1] Get our database reference
const db = getDatabase();

//[STEP 2] Setup our node/path reference
const playerRef = ref(db, "players");

//[STEP 3] Setup our event listener
var readBtn = document
 .getElementById("btn-read")
 .addEventListener("click", getPlayerData);

//[STEP 4] Setup our player function to display info
function getPlayerData(e) {
 e.preventDefault();
 //playerRef is declared at the top using a constant
 //const playerRef = ref(db, "players");
 //get(child(db,`players/`))
 get(playerRef).then((snapshot) => {   //retrieve a snapshot of the data using a callback
   if (snapshot.exists()) {
     //if the data exist
     try {
       //let's do something about it
       var playerContent = document.getElementById("player-content");
       var content = "";

       snapshot.forEach((childSnapshot) => {
         //looping through each snapshot
         //https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach
         console.log("User key: " + childSnapshot.key);
         console.log("Username: " + childSnapshot.child("username").val());
         content += `<tr>
         <td>${childSnapshot.child("active").val()}</td>
         //======= insert your own place to update UI
         </tr>`;
       });
       //update our table content
       playerContent.innerHTML = content;
     } catch (error) {
       console.log("Error getPlayerData" + error);
     }
   }else{
     //@TODO what if no data ?
   }
 });
} //end getPlayerData
```

Note the usage of **.val()** to retrieve the property value

ACTIVITY

“READ + HTML” (30 min - GROUP)

Create your READ data and display in HTML
Based on your previous CAs, add in new functionalities to deal with READ data.

Place your configs in another folder
Optimise your code to show what happens when there’s no data.
Optional: Use Bootstrap/jQuery for nicer UI/easier JS handling

You may use one main node reference e.g players or playerStats depending on your own data structure

CRUD
#CREATE

CRUD - **CREATING** Data

When we write information, there's no callback coming back. Hence we can update the UI once it is set properly

Unix Timestamp is used to provide us a numeric value for date. This timestamp can be converted to any kind of date format later for display purposes.

Set new info

```
set(ref(<database>, `<path>`), <data>);
```

```
import {
 getDatabase,
 ref,
 set,
} from "https://www.gstatic.com/firebasejs/9.5.0/firebase-database.js";

//https://firebase.google.com/docs/reference/js/database

//[STEP 1] Get our database reference
//===================================================
const db = getDatabase();

//[STEP 2] Setup our Create using "set"
//===================================================
 var currentTimestamp = new Date().getTime();
 var playerData = {
   active: true,
   createdOn: currentTimestamp,
   displayName: "testPlayer",
   email: "someemail@email.com",
   lastLoggedIn: currentTimestamp,
   updatedOn: currentTimestamp,
   userName: "some user name",
 };

 set(ref(db, `players/${userId}`), playerData);
```

Reading Reference
https://firebase.google.com/docs/reference/js/database
https://firebase.google.com/docs/reference/js/database.datasnapshot

CRUD - **READ/WRITE** Data (Listening for Data)

We can listen for new data updates, changes using the **onValue** property and provide a reference path to listen to

Listening for Information

```
onValue(playerRef, (snapshot) => {
   //do whatever you want
   //eg. Update UI when data is changed
   updatePlayerContent(snapshot);
 });
```

https://firebase.google.com/docs/database/web/read-and-write#web_value_events

Event	**Typical usage**
value	Read and listen for changes to the entire contents of a path.

You can use the value event to read a static snapshot of the contents at a given path, as they existed at the time of the event. This method is triggered once when the listener is attached and again every time the data, including children, changes. The event callback is passed a snapshot containing all data at that location, including child data. If there is no data, the snapshot will return false when you call exists() and null when you call val() on it.

```
import {
 getDatabase,
 ref,
 set,
} from "https://www.gstatic.com/firebasejs/9.5.0/firebase-database.js";

//https://firebase.google.com/docs/reference/js/database

//[STEP 1] Get our database reference
//==================================================
const db = getDatabase();

//[STEP 2] Setup our Create using "set"
//==================================================
 var currentTimestamp = new Date().getTime();
 var playerData = {
   active: true,
   createdOn: currentTimestamp,
   displayName: "testPlayer",
   email: "someemail@email.com",
   lastLoggedIn: currentTimestamp,
   updatedOn: currentTimestamp,
   userName: "some user name",
 };

 set(ref(db, `players/${userId}`), playerData);

onValue(playerRef, (snapshot) => {
   //const data = snapshot.val();
   updatePlayerContent(snapshot);
 });
```

ACTIVITY

"CREATE + HTML" (30 min)

Add on your Create function into your script, update the UI when you are done.
Based on your previous CAs, add in new functionalities to deal with **CREATE** using Firebase **set**.

Have a form to accept user inputs
Once form is submitted, process it and add the data to Firebase

Optimise your code to show what happens when there's no data, think about edge cases and perform validation
Optional: Use Bootstrap/jQuery for nicer UI/easier JS handling

You may use one main node reference e.g players or playerStats depending on your own data structure

Working with #Auth.CurrentUser

Auth.CurrentUser

The Authentication in Firebase is very powerful and packed with features.

Once we are logged in, we can use our authentication reference to retrieve the current user session and get user's details (userId, DisplayName, ProfilePic, etc)

```
const auth = getAuth();
//Auth refers to our auth object that is derived from the Firebase auth
service
//onAuthStateChanged is an observer that provides a Promise return
//currentUser is our Promise (the naming doesn't matter)
onAuthStateChanged(auth, (currentUser) => {
   if (currentUser) {
     // User is signed in, see docs for a list of available properties
     // https://firebase.google.com/docs/reference/js/firebase.User
     const uid = currentUser.uid;
     statusMsg.innerHTML = `(OnAuthStateChanged) Welcome back:
${currentUser.email} :: ${currentUser.uid}`;
     console.log(`(OnAuthStateChanged) Current user is logged in:
${currentUser.email} ::

   } else {
     statusMsg.innerHTML = `Please sign up or login into the system`;

   }
 });
```

Reading Reference *https://firebase.google.com/docs/reference/js/firebase.User*

Additional Reading
https://firebase.google.com/docs/database/security/indexing-data

Firebase Auth in Web

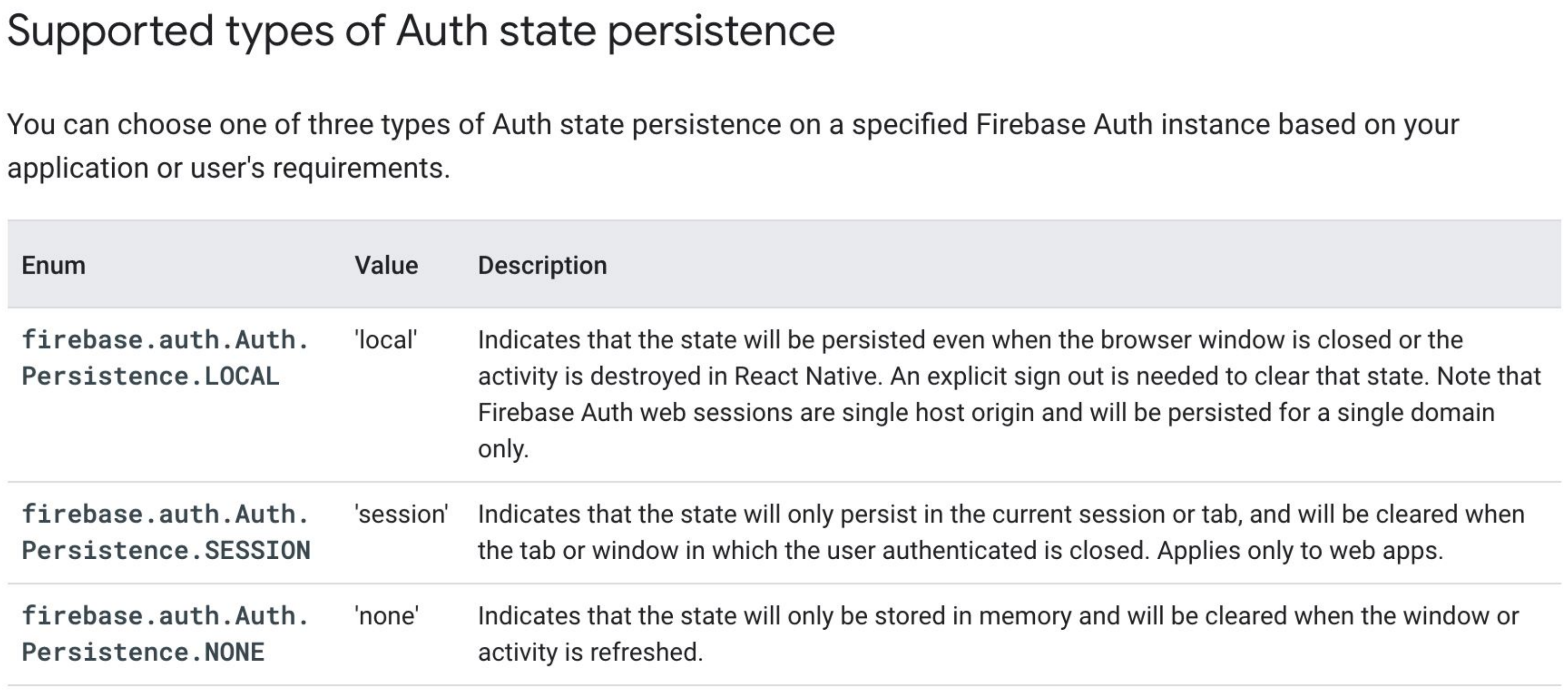
You can specify how the Authentication state persists when using the Firebase JS SDK. This includes the ability to specify whether a **signed in user should be indefinitely persiste**d until explicit sign out, cleared when the window is closed or cleared on page reload.

For a web application, the **default behavior** is to **persist a user's session even after the user closes the browser.**

This is convenient as the user is not required to continuously sign-in every time the web page is visited on the same device. This could require the user having to re-enter their password, send an SMS verification, etc, which could add a lot of friction to the user experience.

Reference
https://firebase.google.com/docs/auth/web/auth-state-persistence#supported_types_of_auth_state_persistence

Firebase Auth in Web



Supported types of Auth state persistence

You can choose one of three types of Auth state persistence on a specified Firebase Auth instance based on your application or user's requirements.

Enum	Value	Description
`firebase.auth.Auth.Persistence.LOCAL`	'local'	Indicates that the state will be persisted even when the browser window is closed or the activity is destroyed in React Native. An explicit sign out is needed to clear that state. Note that Firebase Auth web sessions are single host origin and will be persisted for a single domain only.
`firebase.auth.Auth.Persistence.SESSION`	'session'	Indicates that the state will only persist in the current session or tab, and will be cleared when the tab or window in which the user authenticated is closed. Applies only to web apps.
`firebase.auth.Auth.Persistence.NONE`	'none'	Indicates that the state will only be stored in memory and will be cleared when the window or activity is refreshed.

Reference
https://firebase.google.com/docs/auth/web/auth-state-persistence#supported_types_of_auth_state_persistence

Modifying Auth Persistence

You can specify how the Authentication state persists when using the Firebase JS SDK. This includes the ability to specify whether a **signed in user should be indefinitely persiste**d until explicit sign out, cleared when the window is closed or cleared on page reload.

For a web application, the **default behavior** is to **persist a user's session even after the user closes the browser.**

This is convenient as the user is not required to continuously sign-in every time the web page is visited on the same device. This could require the user having to re-enter their password, send an SMS verification, etc, which could add a lot of friction to the user experience.

Reference
https://firebase.google.com/docs/auth/web/auth-state-persistence#supported_types_of_auth_state_persistence

Working with #Imports

Working with Imports

Imports used in Firebase are meant to keep things as modular as possible. In order to load faster, and bring about better efficiency

```
import {
 getAuth,
 setPersistence,
 signInWithEmailAndPassword,
 browserSessionPersistence,
 inMemoryPersistence,
 browserLocalPersistence, //default
} from
"https://www.gstatic.com/firebasejs/9.5.0/firebase-auth.js";
```

```
import {
   getDatabase,
   ref,
   child,
   get,
   set,
   onValue,
   orderByChild,
 } from
"https://www.gstatic.com/firebasejs/9.5.0/firebase-database.js";
```

```
//base firebase config
import { initializeApp } from
"https://www.gstatic.com/firebasejs/9.5.0/firebase-app.js";

//config settings derived from firebase console
const firebaseConfig = {
 apiKey: "A ",
 authDomain: " ",
 databaseURL: " ",
 projectId: " ",
 storageBucket: " ",
 messagingSenderId: " ",
 appId: " ",
 measurementId: " "
 };
  //Must initialize Firebase app w/ config to start
 const app = initializeApp(firebaseConfig);
```

Lost? Watch this

How to import Firebase with JavaScript modules - Firecasts

https://www.youtube.com/watch?v=lGqKYpvLkhE

Additional Reading

e.com/docs/database/security/indexing-data

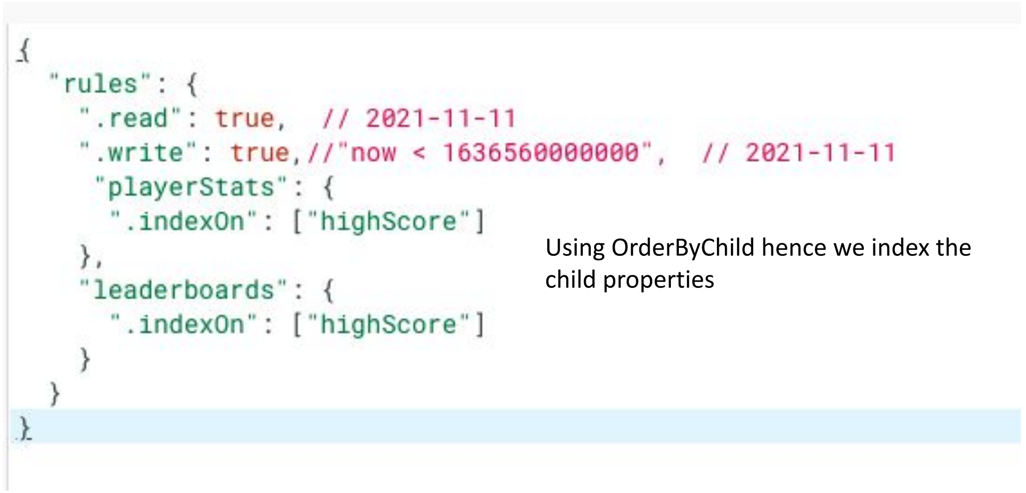
Working with #Indexes

What is an Index (Optimisation)

An index is a powerful tool. Say for example, our NRICs are unique and we know that it is unique. So it is treated like a key in our database. So once we know exactly the key we can retrieve the data

In databases, an index works by “compiling” that data nicely. So that we can sort our data efficiently. In Firebase terms we use .OrderByChild(“somechildproperty”) or .OrderByKey(“somekey”)

When we index, the database will query and find the data much more efficiently. However, having said so, firebase is pretty efficient. So it depends on how much data you have, and how you want to manipulate the data.



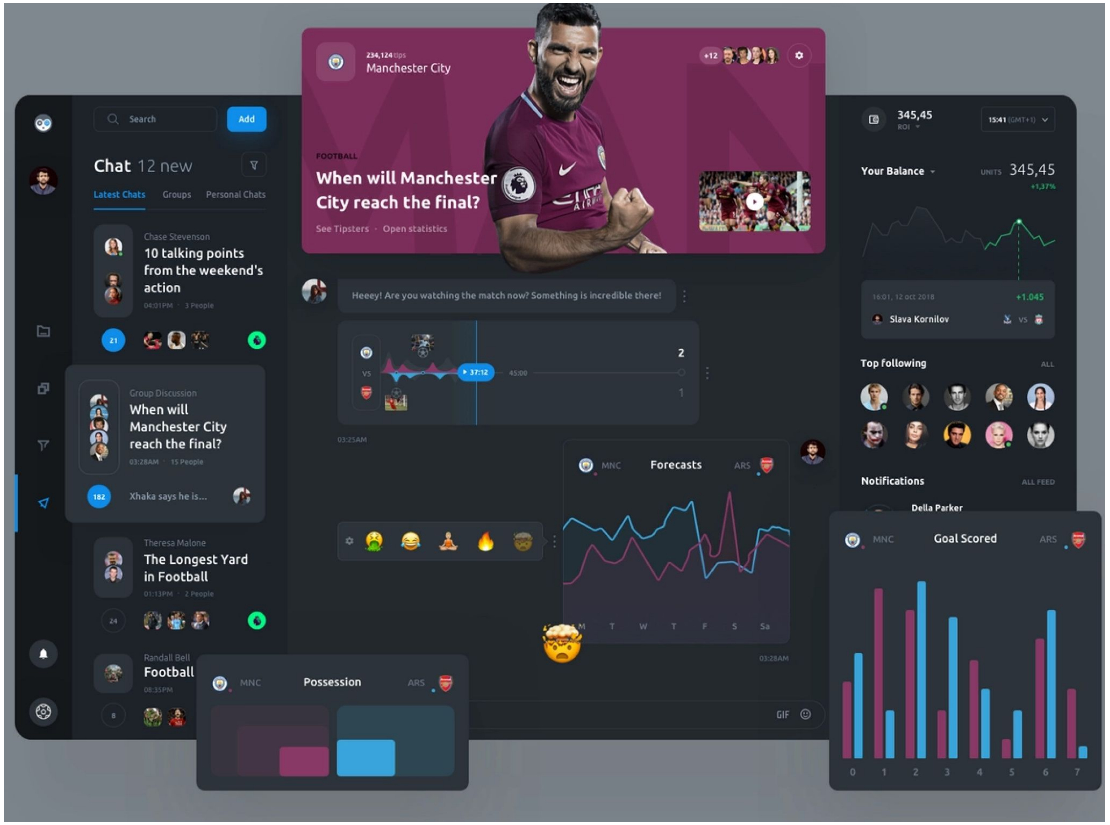
```
{
  "rules": {
    ".read": true,  // 2021-11-11
    ".write": true,//"now < 1636560000000",  // 2021-11-11
     "playerStats": {
      ".indexOn": ["highScore"]
    },
    "leaderboards": {
      ".indexOn": ["highScore"]
    }
  }
}
```

Using OrderByChild hence we index the child properties

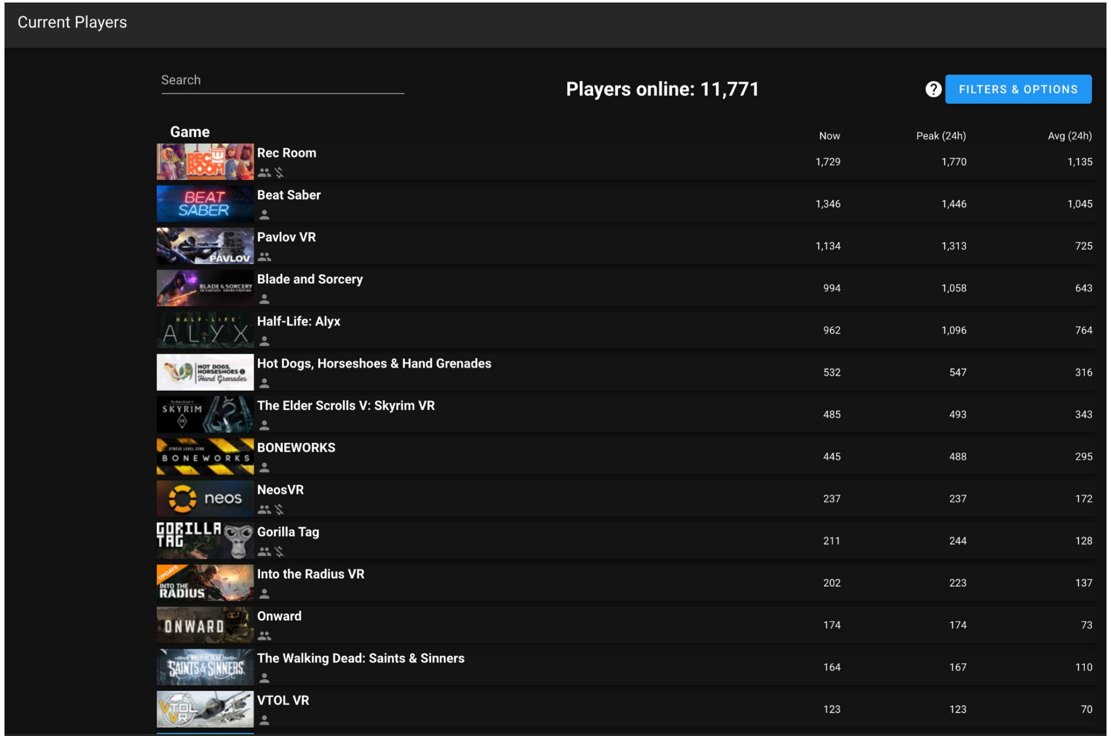
Additional Reading
https://firebase.google.com/docs/database/security/indexing-data

Importance of #Data #Dashboarding

Displaying Data in Dashboard + Charts (WEB)



Displaying Data in Dashboard + Charts (Web)



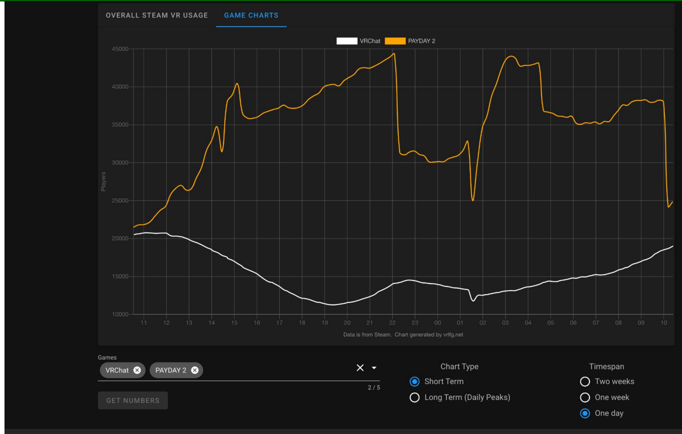
Current Players

Search

Players online: 11,771

Game	Now	Peak (24h)	Avg (24h)
Rec Room	1,729	1,770	1,135
Beat Saber	1,346	1,446	1,045
Pavlov VR	1,134	1,313	725
Blade and Sorcery	994	1,058	643
Half-Life: Alyx	962	1,096	764
Hot Dogs, Horseshoes & Hand Grenades	532	547	316
The Elder Scrolls V: Skyrim VR	485	493	343
BONEWORKS	445	488	295
NeosVR	237	237	172
Gorilla Tag	211	244	128
Into the Radius VR	202	223	137
Onward	174	174	73
The Walking Dead: Saints & Sinners	164	167	110
VTOL VR	123	123	70

Comparing Game activity



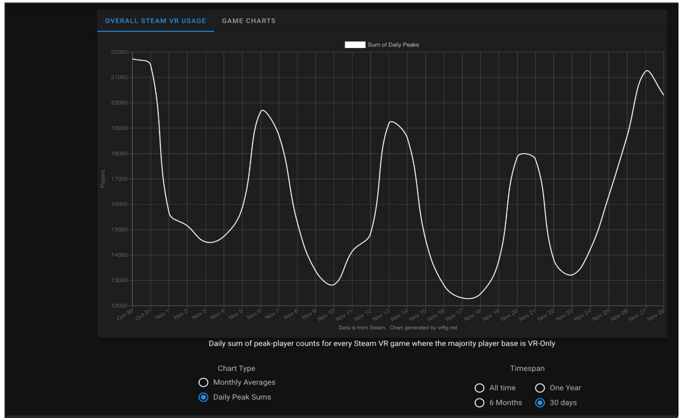
Games: VRChat, PAYDAY 2 (2 / 5)

GET NUMBERS

Chart Type: [✓ Short Term] Long Term (Daily Peaks)

Timespan: Two weeks One week [✓ One day]

Displaying Game data over time period w/ comparison



Daily sum of peak-player counts for every Steam VR game where the majority player base is VR-Only

Chart Type: Monthly Averages [✓ Daily Peak Sums]

Timespan: All time One Year 6 Months [✓ 30 days]

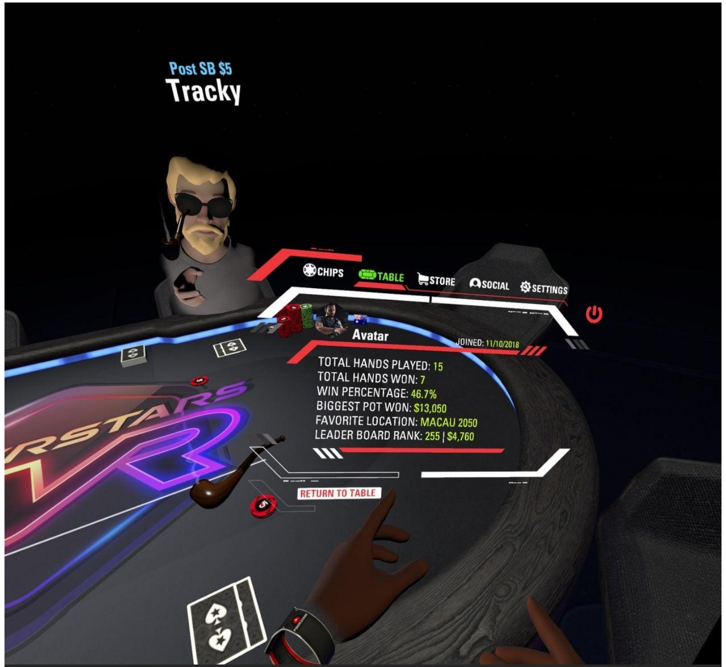
Daily Player Activity Charting

DDA

Lesson 6: Advanced Firebase w/ Web & Dashboarding

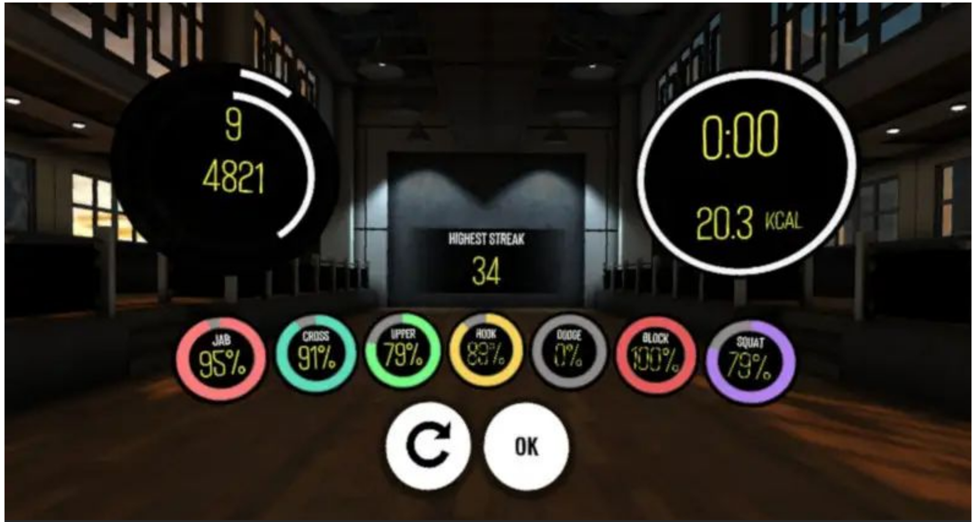
Reference
https://vrlfg.net/Charts

Displaying Data in Dashboard + Charts (VR)



TOTAL HANDS PLAYED: 15
TOTAL HANDS WON: 7
WIN PERCENTAGE: 46.7%
BIGGEST POT WON: $13,050
FAVORITE LOCATION: MACAU 2050
LEADER BOARD RANK: 255 | $4,760

PokerStarsVR Player Stats
https://www.oculus.com/experiences/quest/2370815932930055/



BoxVR Player Stats
https://store.steampowered.com/app/641960/BOXVR/

DDA

Displaying Data in Dashboard + Charts (Chart.js)

```javascript
import { getDatabase, ref, child, get} from
"https://www.gstatic.com/firebasejs/9.5.0/firebase-database.js";

//[STEP 1] Get our database reference
//===================================================
const db = getDatabase();

//[STEP 2] Setup our node/path reference
//===================================================
const dailyActiveUsers = ref(db, "dailyActiveUsers");

//[STEP 3] Retrieve our data
//===================================================
get(dailyActiveUsers).then((snapshot) => {
 //retrieve a snapshot of the data using a callback
 if (snapshot.exists()) {
   //if the data exist
   try {
     var content = "";
     //setup our temp arrays
     var dates = [];
     var logs = [];
     snapshot.forEach((childSnapshot) => {
       //push data to our arrays for our X/Y axes later
       dates.push(childSnapshot.key);
       logs.push(childSnapshot.size);
       console.log(`Number of Players: ${childSnapshot.key}`);
       console.log(`looping child size:  ${childSnapshot.size}`);
     });
     makeChart(dates, logs);
   } catch (error) {}
 }
}); //end get
```

```javascript
//[STEP 4] Make our chart
//===================================================
function makeChart(dates, logData) {
 console.log(logData);
 //based on the canvas ID
 const ctx = document.getElementById("myChart").getContext("2d");
 const myChart = new Chart(ctx, {
   type: "line",
   data: {
     labels: dates,//xaxis
     datasets: [
       {
         label: "# of Active Users",
         data: logData, //yaxis
         borderWidth: 1,
         borderColor: "#8e5ea2",
         backgroundColor: "rgb(142, 94, 162, 0.2)",
         lineTension: 0.4,
         fill: true,
         borderWidth: 3
       },
     ],
   },
   options: {
     scales: {
       y: {
         ticks:{
             stepSize:1,
             beginAtZero: true,
         }
       }
     },
   },
 });
 //charting size
 myChart.canvas.parentNode.style.height = '800px';
 myChart.canvas.parentNode.style.width = '800px';
}
```

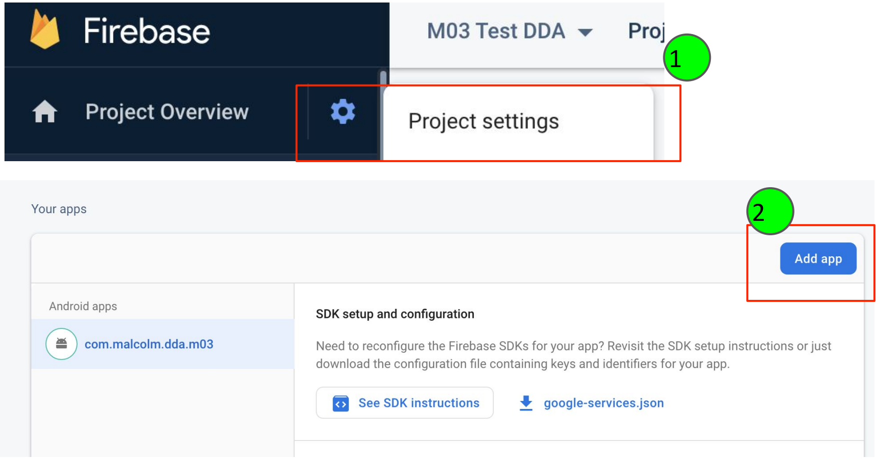
Additional #References

Additional Reading

Additional Reference CRUD with Firebase RealTimeDB & Web
https://www.youtube.com/watch?v=oxqVnWPg0So
Simple Firebase DB Setup with Web
https://www.youtube.com/watch?v=S8D9Cxb2ILA

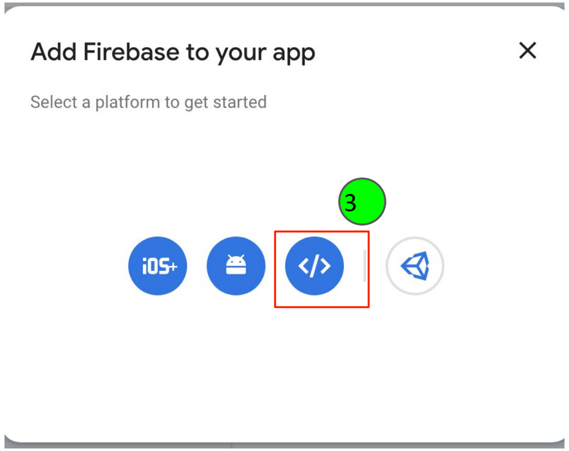
Setting up Firebase for Web

1. Go to your Firebase Console
2. Select your project
3. Click on Project Overview -> Project Settings
4. Select Add App
5. A popup will appear -> Choose web



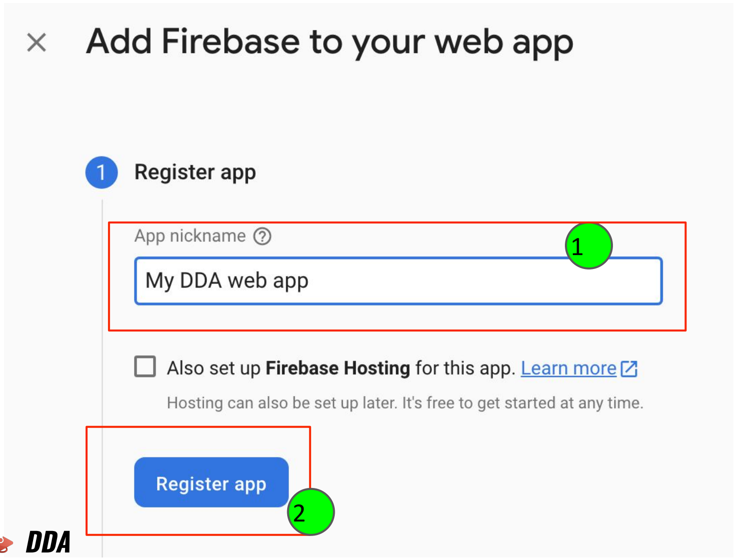
SDK setup and configuration

Need to reconfigure the Firebase SDKs for your app? Revisit the SDK setup instructions or just download the configuration file containing keys and identifiers for your app.



Setting up Firebase for Web

1. Key in an nickname
2. Click on Register app
3. Select “Use a <script> tag”
4. Create a js file, copy and paste the script from Firebase



Add Firebase to your web app

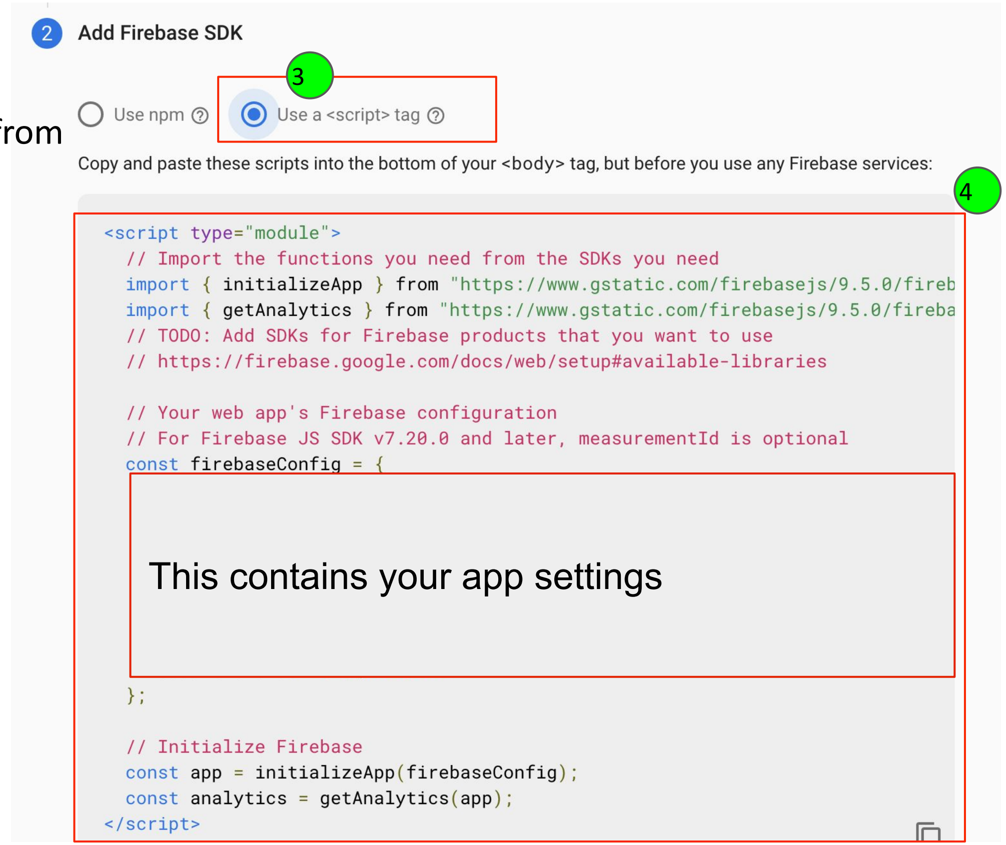
1 Register app

App nickname

My DDA web app

☐ Also set up **Firebase Hosting** for this app. Learn more

Hosting can also be set up later. It's free to get started at any time.



2 Add Firebase SDK

Use npm [✓ Use a <script> tag]

Copy and paste these scripts into the bottom of your <body> tag, but before you use any Firebase services:

```
<script type="module">
  // Import the functions you need from the SDKs you need
  import { initializeApp } from "https://www.gstatic.com/firebasejs/9.5.0/fireb
  import { getAnalytics } from "https://www.gstatic.com/firebasejs/9.5.0/fireba
  // TODO: Add SDKs for Firebase products that you want to use
  // https://firebase.google.com/docs/web/setup#available-libraries

  // Your web app's Firebase configuration
  // For Firebase JS SDK v7.20.0 and later, measurementId is optional
  const firebaseConfig = {

  };

  // Initialize Firebase
  const app = initializeApp(firebaseConfig);
  const analytics = getAnalytics(app);
</script>
```

This contains your app settings