API for firebase

2.0.0B - Code once & deploy everywhere

Getting Started

The API for Firebase unity asset provides a unified & simple API to work with Firebase in your unity project, on all the supported platforms

First steps

Install Unity IOS & Android build tools
Create a Firebase Project
Switch to .NET 4.x api level

> Both the android & ios build tools required!

Standalone | **Mobile** (required)
Setup your Apps for the project
Get your Firebase generated configuration files
Place your IOS & Android configuration files into your assets folder

> Both the android & ios configuration files required!

WebGL (optional)
Setup your WebApp for the project
Get your Firebase generated configuration object
Select the Tamarin WebGLTemplate (you can modify the styles, html, etc! You have to keep the code, and the directory)

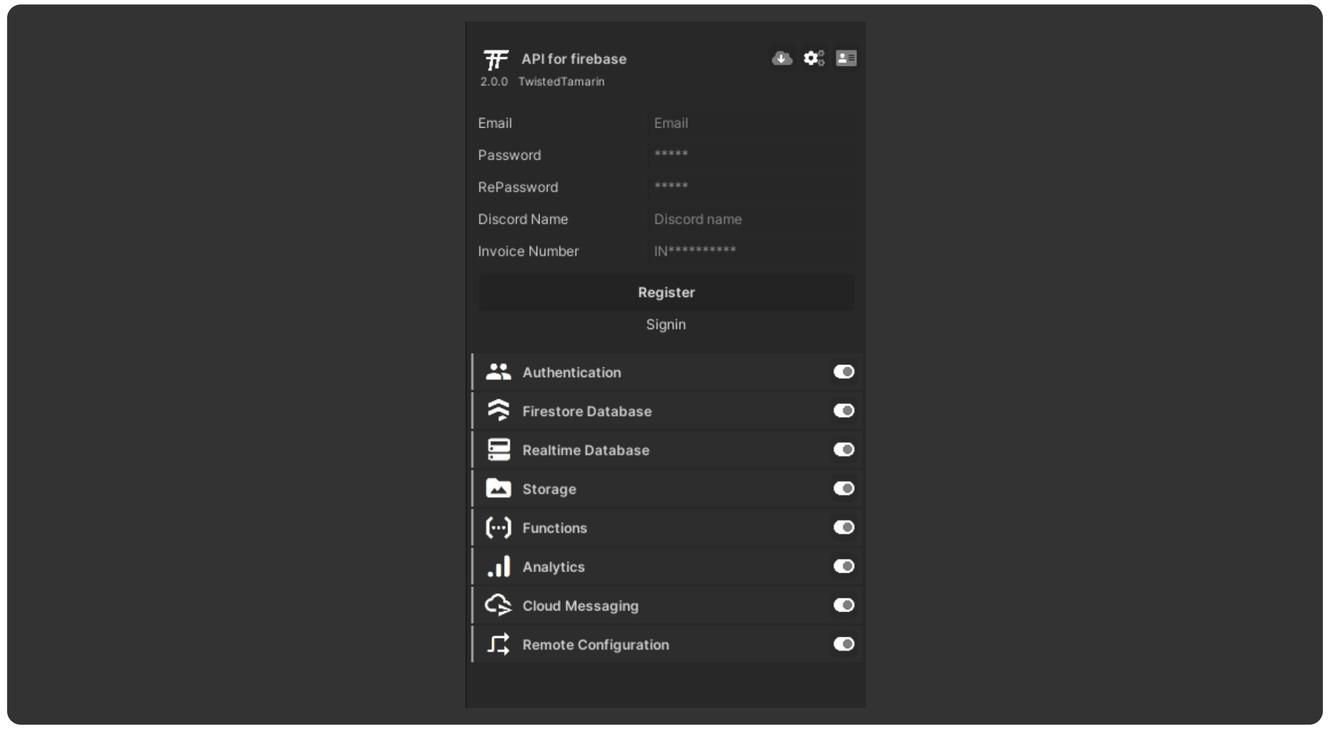
> Both the android & ios configuration files required!

> Compression settings (gzip | brotli) require proper server setup, disable it for development.

> The asset depends on the official apis for each respective platform, with the same limitations and quirks!

Follow the steps to install the API

Open the Manager via Twistedtamarin -> Firebase -> Manage
Register & verify your license
The discord username is optional, but you cannot request support without one!

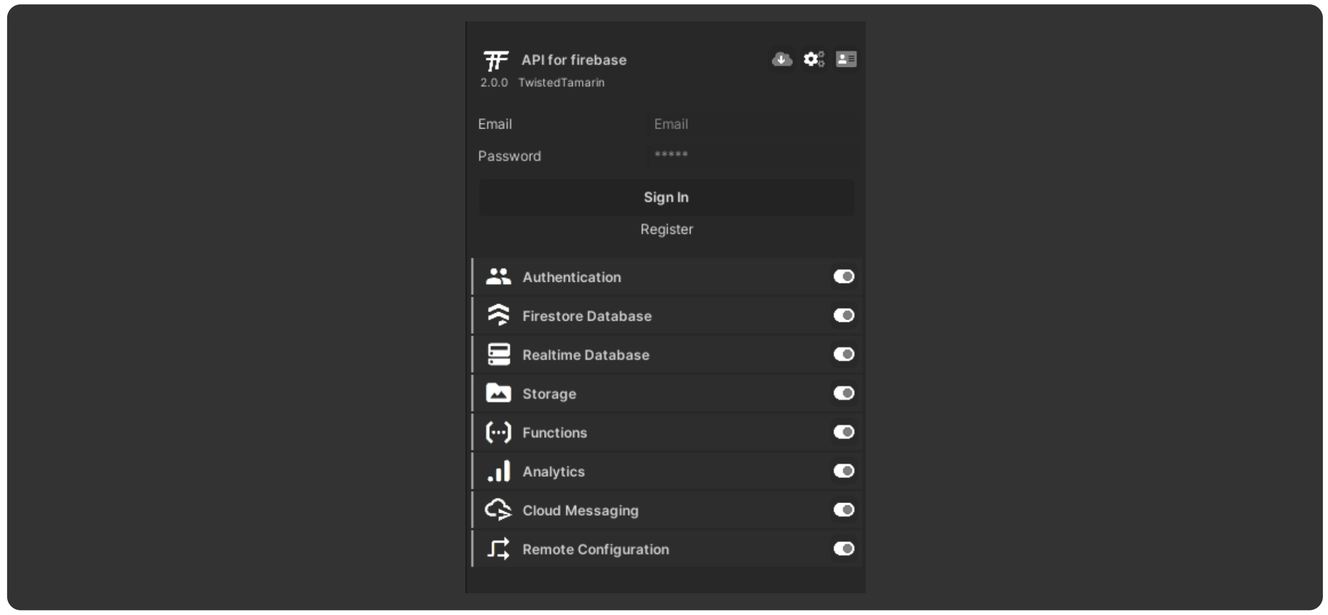


> Close than re-open the manager

> Please remember that the review section is not for support requests! join our discord!

Sign in if you have registered already

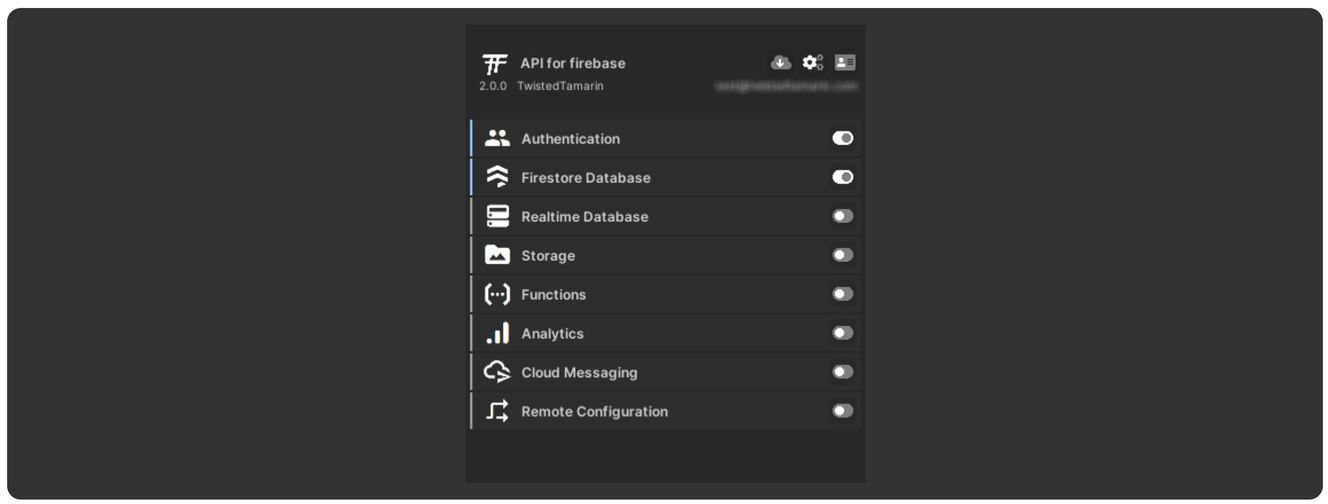
```
await Waiter.Until(()=>FirebaseAPI.Instance.ready == true);
FirebaseAPI firebase = Firebase.Instance;
```



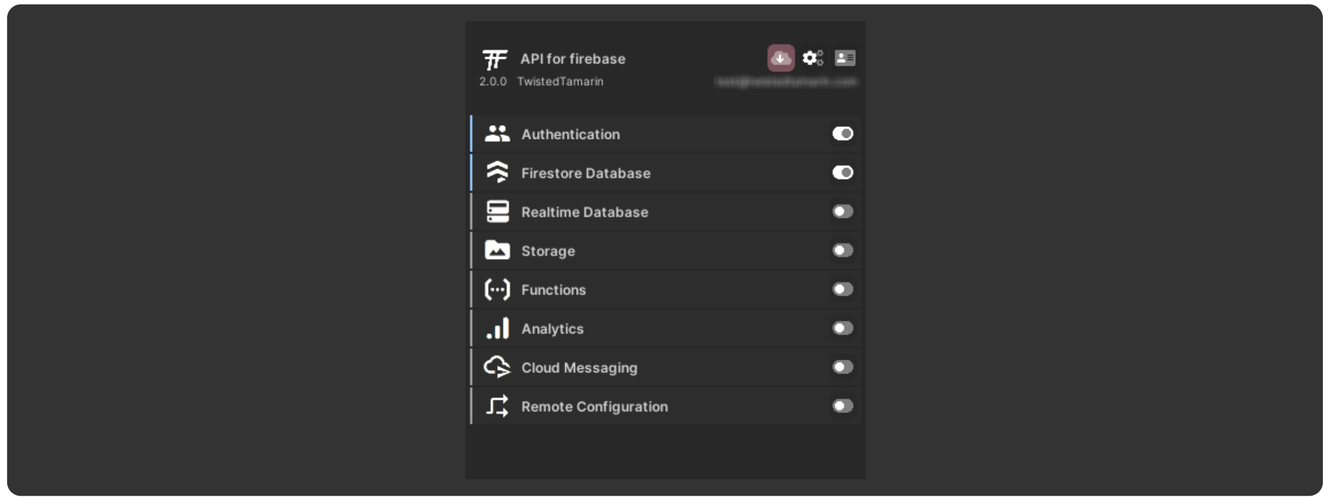
Close than re-open the manager

Please remember that the review section is not for support requests! join our discord!

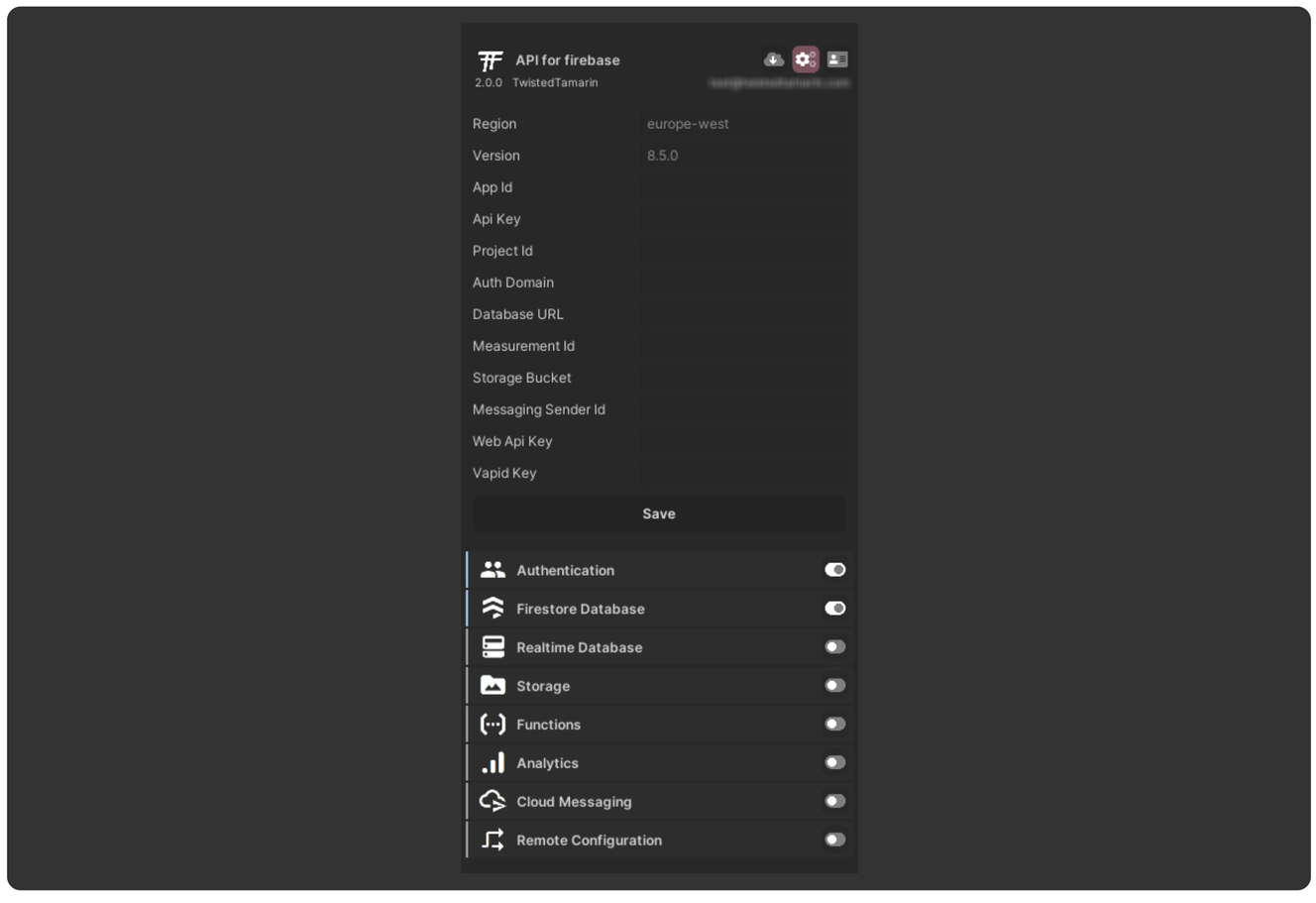
Enable the apis you'd like to use (if available on your license)



Press the update button
Update every time you enable or disable an api
The Update button will install & update all the required apis, be patient it could take a while depending on your system



WebGL (optional)
Open the webGL config panel with the settings button
Find your configs in your firebase console (everything is there)
You only need to fill in the configs required by the apis in use



Everything should be ready. Lets code away!

Authentication

```
await Waiter.Until(()=>FirebaseAPI.Instance.ready == true);
```

To authenticate a user you can use the following authentication methods out of the box.

SignInWithAnon

Param	Type	Description
		No params

SignInWithToken

Param	Type	Description
token	string	Custom auth token

RegisterWithEmail

Param	Type	Description
email	string	User's email
password	string	User's password

SignInWithEmail

Param	Type	Description
email	string	User's email
password	string	User's password
link	bool	Optional, link instead of sign in

SignOut

Param	Type	Description
		No params

VerifyEmail

Param	Type	Description
		No Params

ResetPassword

Param	Type	Description
email	string	Password reset email

UpdateEmail

Param	Type	Description
email	string	Updated email

UpdatePassword

Param	Type	Description
email	string	Updated password

To authenticate a user with 3rd party providers, you can use the following authentication methods

3rd party configurations required for WebGL

3rd party plugins requred for Mobile & Standalone

SignInWithApple

Param	Type	Description
token	string	From apple authentication plugin
nonce	string	From apple authentication plugin
code	string	From apple authentication plugin
link	bool	Optional, link instead of sign in
redirect	bool	Optional, redirect instead of popup

SignInWithAppleProvider

Param	Type	Description

```
FirebaseAPI firebase = Firebase.Instance;
FirebaseUser user;

user = await firebase.auth.SignInWithAnon();
user = await firebase.auth.SignInWithToken(token);
user = await firebase.auth.RegisterWithEmail(email, password);
user = await firebase.auth.SignInWithEmail(email, password);
user = await firebase.auth.SignInWithEmail(email, password, link);

await firebase.auth.signOut();
await firebase.auth.VerifyEmail();
await firebase.auth.ResetPassword(email);
await firebase.auth.UpdateEmail(email);
await firebase.auth.UpdatePassword(password);

// 3rdparty providers
// Redirect only available for WebGL
// 3rd party plugins are required for Mobile & Standalone

user = await firebase.auth.SignInWithApple(token, nonce, code);
user = await firebase.auth.SignInWithApple(token, nonce, code, link, redirect);
user = await firebase.auth.SignInWithAppleProvider(scope, custom, link);

user = await firebase.auth.SignInWithGoogle(idToken);
user = await firebase.auth.SignInWithGoogle(idToken, link, redirect);
user = await firebase.auth.SignInWithGoogleProvider(scope, custom, link);

user = await firebase.auth.SignInWithFacebook(accessToken);
user = await firebase.auth.SignInWithFacebook(accessToken, link, redirect);
user = await firebase.auth.SignInWithFacebookProvider(scope, custom, link);

user = await firebase.auth.SignInWithMicrosoftProvider(scope, custom, link);
```

Param	Type	Description
scope	List	Scopes
custom	Dictionary	Custom parameters
link	bool	Link current user to the auth method

SignInWithGoogle

Param	Type	Description
idToken	string	From google authentication plugin
authCode	string	From google authentication plugin (*optional)
link	bool	Optional, link instead of sign in
redirect	bool	Optional, redirect instead of popup

SignInWithGoogleProvider

Param	Type	Description
scope	List	Scopes
custom	Dictionary	Custom parameters
link	bool	Link current user to the auth method

SignInWithFacebook

Param	Type	Description
accessToken	string	From facebook SDK
link	bool	Optional, link instead of sign in
redirect	bool	Optional, redirect instead of popup

SignInWithFacebookProvider

Param	Type	Description
scope	List	Scopes
custom	Dictionary	Custom parameters
link	bool	Link current user to the auth method

SignInWithMicrosoftProvider

Param	Type	Description
scope	List	Scopes
custom	Dictionary	Custom parameters
link	bool	Link current user to the auth method

Firestore Database

Cloud Firestore is the successor of Realtime Database, with powerful query operations & realtime listeners!

Firestore DataModels require a special attribute [FirestoreData]

Properties won't be parsed without a [FirestoreProperty] attribute and {get; set;}

To ignore properties on WebGL use the [JsonIgnore] attribute

To retreive data from firestore, you can use QueryAs and its overloads
Which will return the DataModel specified!

QueryAs

Param	Type	Description
path	string	Firestore collection

```
await Waiter.Until(()=>FirebaseAPI.Instance.ready == true);
FirebaseAPI firebase = Firebase.Instance;

var res = await firebase.firestore.QueryAs<T>(collection);
var res = await firebase.firestore.QueryAs<T>(collection, query);
var res = await firebase.firestore.QueryAs<T>(collection, documentId);

firebase.firestore.QueryListenAs<T>(collection, documentId, callback);
firebase.firestore.QueryListenAs<T>(collection, query, callback);


var res = await firebase.firestore.SetAsync(collection, data);
var res = await firebase.firestore.SetAsync(collection, documentId, data);

var res = await firebase.firestore.UpdateAsync(collection, documentId, data);

var res = await firebase.firestore.RemoveAsync(collection, documentId);
```

QueryAs

Param	Type	Description
path	string	Firestore collection
docId	string	Firestore document id

QueryAs

Param	Type	Description
path	string	Firestore collection
search	List	List of query operations
limit	Int	Firestore query size limit

To listen to data changes in firstore, you can use QueryListenAs and its overloads Which will return the DataModel specified!

QueryListenAs

Param	Type	Description
path	string	Firestore collection
docId	string	Firestore document id
callback	System.Action	Callback action

QueryListenAs

Param	Type	Description
path	string	Firestore collection
search	List	List of query operations
callback	System.Action	Callback action
limit	Int	Firestore query limit (*optional)

To Create | Update | Delete firestore documents you can use the following methods

SetAsync

Param	Type	Description
path	string	Firestore collection
data	object	data

SetAsync

Param	Type	Description
path	string	Firestore collection
docId	string	Firestore document id
data	object	data

UpdateAsync

Param	Type	Description
path	string	Firestore collection
docId	string	Firestore document id
data	object	data

RemoveAsync

Param	Type	Description
path	string	Firestore collection
docId	string	Firestore document id

FirestoreQuery Operations

Operation
==
<=

Operation
>=
<
>
in
array-contains
array-contains-any

Realtime Database

Realtime database is a json-like no-sql database, with powerful data listeners

> We recommend using DataModels for Realtime Database, yet its not required

You can use the following methods and their overloads to work with realtime database

SetAsync

Param	Type	Description
path	string	The path to the data
data	object	The data to set

SetRawAsync

Param	Type	Description
path	string	The path to the data
data	object	The data to set

UpdateAsync

Param	Type	Description
path	string	The path to the data
data	Dictionary	The data to updatet

QueryAsync

Param	Type	Description
path	string	The path to the data
query	List	Query operations

QueryListenAsync

Param	Type	Description
path	string	The path to the data
callback	callback	Listener callback
query	List	Query operations

```
await Waiter.Until(()=>FirebaseAPI.Instance.ready == true);
FirebaseAPI firebase = Firebase.Instance;

await firebase.database.SetAsync(path, data);
await firebase.database.SetRawAsync(path, data);
await firebase.database.UpdateAsync(path, data);

var res = await firebase.database.QueryAsync(path);
var res = await firebase.database.QueryAsync<T>(path);
var res = await firebase.database.QueryAsync(path, query);
var res = await firebase.database.QueryAsync<T>(path, query);

firebase.database.QueryListenAsync(path, (result)=>{
...JSONDATA
}, query);

firebase.database.QueryListenAsync<T>(path, (result)=>{
...MODELDATA
}, query);
```

Storage

Cloud Storage for Firebase is a powerful, simple, and cost-effective object storage service

> Files for upload must be in byte[] array

StorageUpload

Params	Type	Description
bucket	string	Optional, custom bucket url
path	string	The path and file name
data	byte[]	The file for upload
type	string	Mime type

StorageDownloadUrl

Params	Type	Description
bucket	string	Optional, custom bucket url
path	string	The path and file name

```
```

```
await Waiter.Until(()=>FirebaseAPI.Instance.ready == true);
FirebaseAPI firebase = Firebase.Instance;

var upload = await firebase.storage.Upload(path, buffer, type);
var downloadUrl = await firebase.storage.DownloadUrl(path);

var upload = await firebase.storage.Upload(bucket, path, buffer, type);
var downloadUrl = await firebase.storage.DownloadUrl(bucket, path);
```

Functions

Cloud functions are the best way to do backend operations

Don't forget to set the region in the FirebaseAPI & FirebaseWebHelper

Send data as a Dictionary<string, object>

Results require a DataModel <T>

HttpsCall

Params	Type	Description
fn	string	The function name to call
data	object	Function payload

```
await Waiter.Until(()=>FirebaseAPI.Instance.ready == true);
FirebaseAPI firebase = Firebase.Instance;

var res = await firebase.functions.HttpsCall<T>(fn, data);
```

Analytics

You can use the following methods & their overloads for Analytics

Setup

Param	Type	Description
status	bool	Set analytics collection enabled | disabled
timeout	int	Set analytics session timeout seconds

SetCollection

Param	Type	Description
status	bool	Set analytics collection enabled | disabled

SetTimeout

Param	Type	Description
timeout	int	Set analytics session timeout seconds

SetUserId

Param	Type	Description
userId	string	Analytics user id

SetUserProperty

Param	Type	Description
prop	string	Analytics user property name
value	string	Analytics user property value

LogEvent

Param	Type	Description
name	string	Event name
param	string	Event param name
value	variable	Event param value

```
await Waiter.Until(()=>FirebaseAPI.Instance.ready == true);
FirebaseAPI firebase = Firebase.Instance;

firebase.analytics.Setup(status);
firebase.analytics.Setup(status, timeout);

firebase.analytics.SetCollection(status);
firebase.analytics.SetTimeout(timeout);

firebase.analytics.SetUserId(userId);
firebase.analytics.SetUserProperty(prop, value);

firebase.analytics.LogEvent(event);
firebase.analytics.LogEvent(name, param, value);
```

Messaging

With Cloud messaging you can listen & subscribe to push notifications

> Push notification setup required

> WebGL notifications are limited & require a VapidKey

> WebGL background notifications are handled via: firebase-messaging-sw.js service worker

Setup & Listening to incoming messages

Setup

Param	Type	Description
callback	callback	callback for push token

OnReceived

Param	Type	Description
callback	callback	callback for FirebaseMessage

Topic subscribe & unsubscribe Not available on WebGL

Subscribe

Param	Type	Description
topic	string	Topic name

Unsubscribe

Param	Type	Description
topic	string	Topic name

```
await Waiter.Until(()=>FirebaseAPI.Instance.ready == true);
FirebaseAPI firebase = Firebase.Instance;

firebase.messaging.Setup((token) =>
{
...token
});

firebase.messaging.OnReceived((message) =>
{
...message
});

firebase.messaging.Subscribe(topic);
firebase.messaging.Unsubscribe(topic);
```

Remote Configuration

You can use the following methods for Remote Configuration

Setup

Param	Type	Description
data	Dictionary	Default config
interval	long	Minimum check interval

FetchAndActivate

Param	Type	Description
		no params

GetValue

Param	Type	Description
name	string	Property name

GetAll

Param	Type	Description
		no params

```
await Waiter.Until(()=>FirebaseAPI.Instance.ready == true);
FirebaseAPI firebase = Firebase.Instance;

await firebase.config.Setup(config, interval);
await firebase.config.FetchActivate();
var result = await firebase.config.GetValue(name);
var result = await firebase.config.GetAll();
```