Firebase Admin SDK for PHP

Jun 23, 2019

Contents

Interact with Google Firebase from your PHP application.

Note: If you are interested in using the PHP Admin SDK as a client for end-user access (for example, in a web application), as opposed to admin access from a privileged environment (like a server), you should instead follow the instructions for setting up the client JavaScript SDK.

The source code can be found at https://github.com/kreait/firebase-php/ .

CHAPTER 1

User Guide

1.1 Overview

1.1.1 Requirements

- PHP >= 7.0
- The mbstring PHP extension
- A Firebase project - create a new project in the Firebase console, if you don't already have one.
- A Google service account, follow the instructions in the official Firebase Server documentation and place the JSON configuration file somewhere in your project's path.

1.1.2 Installation

The recommended way to install the Firebase Admin SDK is with Composer. Composer is a dependency management tool for PHP that allows you to declare the dependencies your project needs and installs them into your project.

```
composer require kreait/firebase-php ^4.18
```

Alternatively, you can specify the Firebase Admin SDK as a dependency in your project's existing composer.json file:

```
 {
   "require": {
      "kreait/firebase-php": "^4.18"
   }
}
```

After installing, you need to require Composer's autoloader:

```
<?php

require __DIR__.'/vendor/autoload.php';
```

You can find out more on how to install Composer, configure autoloading, and other best-practices for defining dependencies at getcomposer.org.

Please continue to the *Setup section* to learn more about connecting your application to Firebase.

1.1.3 Usage example

You can find more usage examples at https://github.com/jeromegamez/firebase-php-examples and in the tests directory of this project's GitHub repository.

```php
<?php

require __DIR__.'/vendor/autoload.php';

use Kreait\Firebase\Factory;
use Kreait\Firebase\ServiceAccount;

// This assumes that you have placed the Firebase credentials in the same directory
// as this PHP file.
$serviceAccount = ServiceAccount::fromJsonFile(__DIR__.'/google-service-account.json
↪');

$firebase = (new Factory)
    ->withServiceAccount($serviceAccount)
    // The following line is optional if the project id in your credentials file
    // is identical to the subdomain of your Firebase project. If you need it,
    // make sure to replace the URL with the URL of your project.
    ->withDatabaseUri('https://my-project.firebaseio.com')
    ->create();

$database = $firebase->getDatabase();

$newPost = $database
    ->getReference('blog/posts')
    ->push([
        'title' => 'Post title',
        'body' => 'This should probably be longer.'
    ]);

$newPost->getKey(); // => -KVr5eu8gcTv7_AHb-3-
$newPost->getUri(); // => https://my-project.firebaseio.com/blog/posts/-KVr5eu8gcTv7_
↪AHb-3-

$newPost->getChild('title')->set('Changed post title');
$newPost->getValue(); // Fetches the data from the realtime database
$newPost->remove();
```

1.1.4 Issues/Support

- For bugs, feature requests and past issues: Github issue tracker
- For help with and discussion about the PHP SDK: Discord channel dedicated to this library
- For questions about Firebase in general: Stack Overflow and the Firebase Slack Community.

1.1.5 License

Licensed using the MIT license.

> Copyright (c) Jérôme Gamez <https://github.com/jeromegamez> <jerome@gamez.name>
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.1.6 Contributing

Guidelines

1. The SDK utilizes PSR-1, PSR-2, PSR-4, and PSR-7.
2. This SDK has a minimum PHP version requirement of PHP 7.0. Pull requests must not require a PHP version greater than PHP 7.0 unless the feature is only utilized conditionally.
3. All pull requests must include unit tests to ensure the change works as expected and to prevent regressions.

Running the tests

The SDK is unit tested with PHPUnit. Run the tests using the Makefile:

```
make tests
```

Coding standards

The SDK uses the PHP Coding Standars Fixer to ensure a uniform coding style. Apply coding standard fixed using the Makefile:

```
make cs
```

from the root of the project.

1.2 Setup

1.2.1 Google Service Account

In order to access a Firebase project using a server SDK, you must authenticate your requests to Firebase with a Service Account.

Follow the steps described in the official Firebase documentation to create a Service Account for your Firebase application (see Add the Firebase Admin SDK to your Server) and make sure the Service Account has the *Project -> Editor* or *Project -> Owner* role.

With autodiscovery

By default, the SDK is able to autodiscover the Service Account for your project in the following conditions:

1. Your application runs on Google Cloud Engine.
2. The path to the JSON key file is defined in one of the following environment variables
   - `FIREBASE_CREDENTIALS`
   - `GOOGLE_APPLICATION_CREDENTIALS`
3. The JSON Key file is located in Google's "well known path"
   - on Linux/MacOS: `$HOME/.config/gcloud/application_default_credentials.json`
   - on Windows: `$APPDATA/gcloud/application_default_credentials.json`

If one of the conditions above is met, creating a new Firebase instance is as easy as this:

```
use Kreait\Firebase\Factory;

$firebase = (new Factory)->create();
```

A more explicit alternative:

```
use Kreait\Firebase\Factory;
use Kreait\Firebase\ServiceAccount;

$serviceAccount = ServiceAccount::discover();

$firebase = (new Factory)
    ->withServiceAccount($serviceAccount)
    ->create();
```

Manually

You can also pass the path to the Service Account JSON file explicitly:

```
use Kreait\Firebase\Factory;
use Kreait\Firebase\ServiceAccount;

$serviceAccount = ServiceAccount::fromJsonFile(__DIR__.'/firebase_credentials.json');
$firebase = (new Factory)
    ->withServiceAccount($serviceAccount)
    ->create();
```

Use your own autodiscovery

You can use your own, custom autodiscovery methods as well:

```
use Kreait\Firebase\Factory;
use Kreait\Firebase\ServiceAccount\Discoverer

$discoverer = new Discoverer([
    function () {
        $serviceAccount = ...; // Instance of Kreait\Firebase\ServiceAccount

        return $serviceAccount;
    }
]);

$firebase = (new Factory)
    ->withServiceAccountDiscoverer($myDiscoverer)
    ->create();
```

Disabling the autodiscovery

You can also disable the autodiscovery. This can be useful to ensure that it will not be triggered in case an explicitely given service account is invalid.

```
use Kreait\Firebase\Factory;

$firebase = (new Factory)
    ->withServiceAccount($serviceAccount)
    ->withDisabledAutoDiscovery()
    ->create();
```

1.2.2 Custom Database URI

If the project ID in the JSON file does not match the URL of your Firebase application, or if you want to be explicit, you can configure the Factory like this:

```
use Kreait\Firebase\Factory;

$firebase = (new Factory)
    ->withDatabaseUri('https://my-project.firebaseio.com')
    ->create();
```

1.2.3 HTTP Client Options and middlewares

If you want to extend or change the behaviour of the underlying HTTP client, you can pass options to it while creating your Firebase instance.

See Guzzle Request Options for the available options and Guzzle Middlewares for information on how to use middlewares.

```
use Kreait\Firebase\Factory;

$httpConfig = [
```

(continues on next page)

(continued from previous page)

```
    // see http://docs.guzzlephp.org/en/stable/request-options.html
];

$httpMiddlewares = [
    // see http://docs.guzzlephp.org/en/stable/handlers-and-middleware.html#middleware
];

$firebase = (new Factory)
    ->withHttpClientConfig($httpConfig)
    ->withHttpClientMiddlewares($httpMiddlewares)
    ->create();
```

1.3 Realtime Database

Note: The Realtime Database API currently does not support realtime event listeners.

You can work with your Firebase application's Realtime Database by invoking the `getDatabase()` method of your Firebase instance:

```
use Kreait\Firebase;

$firebase = (new Firebase\Factory())->create();
$database = $firebase->getDatabase();
```

1.3.1 Retrieving data

Every node in your database can be accessed through a Reference:

```
$reference = $database->getReference('path/to/child/location');
```

Note: Creating a reference does not result in a request to your Database. Requests to your Firebase applications are executed with the `getSnapshot()` and `getValue()` methods only.

You can then retrieve a Database Snapshot for the Reference or its value directly:

```
$snapshot = $reference->getSnapshot();

$value = $snapshot->getValue();
// or
$value = $reference->getValue();
```

Database Snapshots

Database Snapshots are immutable copies of the data at a Firebase Database location at the time of a query. The can't be modified and will never change.

```
$snapshot = $reference->getSnapshot();
$value = $snapshot->getValue();

$value = $reference->getValue(); // Shortcut for $reference->getSnapshot()->
↪getValue();
```

Snapshots provide additional methods to work with and analyze the contained value:

- `exists()` returns true if the Snapshot contains any (non-null) data.
- `getChild()` returns another Snapshot for the location at the specified relative path.
- `getKey()` returns the key (last part of the path) of the location of the Snapshot.
- `getReference()` returns the Reference for the location that generated this Snapshot.
- `getValue()` returns the data contained in this Snapshot.
- `hasChild()` returns true if the specified child path has (non-null) data.
- `hasChildren()` returns true if the Snapshot has any child properties, i.e. if the value is an array.
- `numChildren()` returns the number of child properties of this Snapshot, if there are any.

Queries

You can use Queries to filter and order the results returned from the Realtime Database. Queries behave exactly like References. That means you can execute any method on a Query that you can execute on a Reference.

Note: You can combine every filter query with every order query, but not multiple queries of each type. Shallow queries are a special case: they can not be combined with any other query method.

Shallow queries

This is an advanced feature, designed to help you work with large datasets without needing to download everything. Set this to true to limit the depth of the data returned at a location. If the data at the location is a JSON primitive (string, number or boolean), its value will simply be returned.

If the data snapshot at the location is a JSON object, the values for each key will be truncated to true.

Detailed information can be found on the official Firebase documentation page for shallow queries

```
$db->getReference('currencies')
    // order the reference's children by their key in ascending order
    ->shallow()
    ->getSnapshot();
```

A convenience method is available to retrieve the key names of a reference's children:

```
$db->getReference('currencies')->getChildKeys(); // returns an array of key names
```

Ordering data

The official Firebase documentation explains How data is ordered.

Data is always ordered in ascending order.

You can only order by one property at a time - if you try to order by multiple properties, e.g. by child and by value, an exception will be thrown.

By key

```
$db->getReference('currencies')
    // order the reference's children by their key in ascending order
    ->orderByKey()
    ->getSnapshot();
```

By value

Note: In order to order by value, you must define an index, otherwise the Firebase API will refuse the query.

```
{
    "currencies": {
        ".indexOn": ".value"
    }
}
```

```
$db->getReference('currencies')
    // order the reference's children by their value in ascending order
    ->orderByValue()
    ->getSnapshot();
```

By child

Note: In order to order by a child value, you must define an index, otherwise the Firebase API will refuse the query.

```
{
    "people": {
        ".indexOn": "height"
    }
}
```

```
$db->getReference('people')
    // order the reference's children by the values in the field 'height' in␣
↪ascending order
    ->orderByChild('height')
    ->getSnapshot();
```

Filtering data

To be able to filter results, you must also define an order.

limitToFirst

```
$db->getReference('people')
    // order the reference's children by the values in the field 'height'
    ->orderByChild('height')
    // limits the result to the first 10 children (in this case: the 10 shortest␣
↪persons)
    // values for 'height')
    ->limitToFirst(10)
    ->getSnapshot();
```

limitToLast

```
$db->getReference('people')
    // order the reference's children by the values in the field 'height'
    ->orderByChild('height')
    // limits the result to the last 10 children (in this case: the 10 tallest␣
↪persons)
    ->limitToLast(10)
    ->getSnapshot();
```

startAt

```
$db->getReference('people')
    // order the reference's children by the values in the field 'height'
    ->orderByChild('height')
    // returns all persons taller than or exactly 1.68 (meters)
    ->startAt(1.68)
    ->getSnapshot();
```

endAt

```
$db->getReference('people')
    // order the reference's children by the values in the field 'height'
    ->orderByChild('height')
    // returns all persons shorter than or exactly 1.98 (meters)
    ->endAt(1.98)
    ->getSnapshot();
```

equalTo

```
$db->getReference('people')
    // order the reference's children by the values in the field 'height'
    ->orderByChild('height')
    // returns all persons being exactly 1.98 (meters) tall
    ->equalTo(1.98)
    ->getSnapshot();
```

1.3.2 Saving data

Set/replace values

For basic write operations, you can use set() to save data to a specified reference, replacing any existing data at that path. For example a configuration array for a website might be set as follows:

```
$db->getReference('config/website')
   ->set([
       'name' => 'My Application',
       'emails' => [
           'support' => 'support@domain.tld',
           'sales' => 'sales@domain.tld',
       ],
       'website' => 'https://app.domain.tld',
      ]);

$db->getReference('config/website/name')->set('New name');
```

Note: Using `set()` overwrites data at the specified location, including any child nodes.

Update specific fields[1]

To simultaneously write to specific children of a node without overwriting other child nodes, use the update() method.

When calling `update()`, you can update lower-level child values by specifying a path for the key. If data is stored in multiple locations to scale better, you can update all instances of that data using data fan-out.

For example, in a blogging app you might want to add a post and simultaneously update it to the recent activity feed and the posting user's activity feed using code like this:

```
$uid = 'some-user-id';
$postData = [
    'title' => 'My awesome post title',
    'body' => 'This text should be longer',
];

// Create a key for a new post
$newPostKey = $db->getReference('posts')->push()->getKey();

$updates = [
    'posts/'.$newPostKey => $postData,
    'user-posts/'.$uid.'/'.$newPostKey => $postData,
];

$db->getReference() // this is the root reference
   ->update($updates);
```

Writing lists[2]

[1] This example and its description is the same as in the official documentation: Update specific fields.
[2] This example and its description is the same as in the official documentation: Append to a list of data.

Use the `push()` method to append data to a list in multiuser applications. The `push()` method generates a unique key every time a new child is added to the specified Firebase reference. By using these auto-generated keys for each new element in the list, several clients can add children to the same location at the same time without write conflicts. The unique key generated by `push()` is based on a timestamp, so list items are automatically ordered chronologically.

You can use the reference to the new data returned by the `push()` method to get the value of the child's auto-generated key or set data for the child. The `getKey()` method of a `push()` reference contains the auto-generated key.

```
$postData = [...];
$postRef = $db->getReference('posts')->push($postData);

$postKey = $postRef->getKey(); // The key looks like this: -KVquJHezVLf-lSye6Qg
```

Server values

Server values can be written at a location using a placeholder value which is an object with a single *.sv* key. The value for that key is the type of server value you wish to set.

Firebase currently supports only one server value: `timestamp`. You can either set it manually in your write operation, or use a constant from the `Firebase\Database` class.

The following to usages are equivalent:

```
$ref = $db->getReference('posts/my-post')
          ->set('created_at', ['.sv' => 'timestamp']);

$ref = $db->getReference('posts/my-post')
          ->set('created_at', Database::SERVER_TIMESTAMP);
```

Delete data[3]

The simplest way to delete data is to call remove() on a reference to the location of that data.

```
$db->getReference('posts')->remove();
```

You can also delete by specifying null as the value for another write operation such as *set()* or *update()*.

```
$db->getReference('posts')->set(null);
```

You can use this technique with *update()* to delete multiple children in a single API call.

1.3.3 Database transactions

Note: Support for database transactions has been added in release 4.21.0

You can use transaction to update data according to its existing state. For example, if you want to increase an upvote counter, and want to make sure the count accurately reflects multiple, simultaneous upvotes, use a transaction to write the new value to the counter. Instead of two writes that change the counter to the same number, one of the write requests fails and you can then retry the request with the new value.

[3] This example and its description is the same as in the official documentation: Delete data.

Replace data inside a transaction

```
use Kreait\Firebase\Database\Transaction;

$counterRef = $db->getReference('counter');

$db->runTransaction(function (Transaction $transaction) use ($counterRef) {

    // You have to snapshot the reference in order to change its value
    $counterSnapshot = $transaction->snapshot($counterRef);

    // Get the existing value from the snapshot
    $counter = $counterSnapshot->getValue() ?: 0;
    $newCounter = ++$counter;

    // If the value hasn't changed in the Realtime Database while we are
    // incrementing it, the transaction will be a success.
    $transaction->set($counterRef, $newCounter);
});
```

Delete data inside a transaction

Likewise, you can wrap the removal of a reference in a transaction as well: you can remove the reference only if it hasn't changed in the meantime.

```
use Kreait\Firebase\Database\Transaction;

$toBeDeleted = $db->getReference('to-be-deleted');

$db->runTransaction(function (Transaction $transaction) use ($toBeDeleted) {

    $transaction->snapshot($toBeDeleted);

    $transaction->remove($toBeDeleted);
});
```

Handling transaction failures

If you haven't snapshotted a reference before trying to change it, the operation will fail with a `\Kreait\Firebase\Exception\Database\ReferenceHasNotBeenSnapshotted` error.

If the reference has changed in the Realtime Database after you started the transaction, the transaction will fail with a `\Kreait\Firebase\Exception\Database\TransactionFailed` error.

```
use Kreait\Firebase\Database\Transaction;
use Kreait\Firebase\Exception\Database\ReferenceHasNotBeenSnapshotted;
use Kreait\Firebase\Exception\Database\TransactionFailed;

$ref = $db->getReference('my-ref');

try {
    $db->runTransaction(function (Transaction $transaction) use ($ref) {

        // $transaction->snapshot($ref);
```

(continues on next page)

(continued from previous page)

```
        $ref->set('value change without a transaction');

        $transaction->set($ref, 'this will fail');
    });

} catch (ReferenceHasNotBeenSnapshotted $e) {

    $referenceInQuestion = $e->getReference();

    echo $e->getReference()->getUri().': '.$e->getMessage();

} catch (TransactionFailed $e) {

    $referenceInQuestion = $e->getReference();
    $failedRequest = $e->getRequest();
    $failureResponse = $e->getResponse();

    echo $e->getReference()->getUri().': '.$e->getMessage();

}
```

1.3.4 Debugging API exceptions

When a request to Firebase fails, the SDK will throw a `\Kreait\Firebase\Exception\ApiException` that includes the sent request and the received response object:

```
try {
    $db->getReference('forbidden')->getValue();
} catch (ApiException $e) {
    /** @var \Psr\Http\Message\RequestInterface $request */
    $request = $e->getRequest();
    /** @var \Psr\Http\Message\ResponseInterface|null $response */
    $response = $e->getResponse();

    echo $request->getUri().PHP_EOL;
    echo $request->getBody().PHP_EOL;

    if ($response) {
        echo $response->getBody();
    }
}
```

1.3.5 Database rules

Learn more about the usage of Firebase Realtime Database Rules in the official documentation.

```
use Kreait\Firebase\Database\RuleSet;

// The default rules allow full read and write access to authenticated users of your␣
↪app
$ruleSet = RuleSet::default();
```

(continues on next page)

(continued from previous page)

```
// This level of access means anyone can read or write to your database. You should
// configure more secure rules before launching your app.
$ruleSet = RuleSet::public();

// Private rules disable read and write access to your database by users.
// With these rules, you can only access the database through the
// Firebase console and the Admin SDKs.
$ruleSet = RuleSet::private();

// You can of course define custom rules
$ruleSet = RuleSet::fromArray(['rules' => [
    '.read' => true,
    '.write' => false,
    'users' => [
        '$uid' => [
            '.read' => '$uid === auth.uid',
            '.write' => '$uid === auth.uid',
        ]
    ]
]]);

$db->updateRules($ruleSet);

$freshRuleSet = $db->getRules(); // Returns a new RuleSet instance
$actualRules = $ruleSet->getRules(); // returns an array
```

1.4 Authentication[1]

Before you can access the Firebase Realtime Database from a server using the Firebase Admin SDK, you must authenticate your server with Firebase. When you authenticate a server, rather than sign in with a user account's credentials as you would in a client app, you authenticate with a service account which identifies your server to Firebase.

You can get two different levels of access when you authenticate using the Firebase Admin SDK:

Administrative privileges: Complete read and write access to a project's Realtime Database. Use with caution to complete administrative tasks such as data migration or restructuring that require unrestricted access to your project's resources.

Limited privileges: Access to a project's Realtime Database, limited to only the resources your server needs. Use this level to complete administrative tasks that have well-defined access requirements. For example, when running a summarization job that reads data across the entire database, you can protect against accidental writes by setting a read-only security rule and then initializing the Admin SDK with privileges limited by that rule.

1.4.1 Authenticate with admin privileges

When you initialize the Firebase Admin SDK with the credentials for a service account with the Editor role on your Firebase project, that instance has complete read and write access to your project's Realtime Database.

```
use Kreait\Firebase\Factory;
use Kreait\Firebase\ServiceAccount;

```

(continues on next page)

[1] Google: Introduction to the Admin Database API

(continued from previous page)

```
$serviceAccount = ServiceAccount::fromJsonFile(__DIR__.'/google-service-account.json
↪');

$firebase = (new Factory)
    ->withServiceAccount($serviceAccount)
    ->create();
```

Note: Your service only has as much access as the service account used to authenticate it. For example, you can limit your service to read-only by using a service account with the Reader role on your project. Similarly, a service account with no role on the project is not able to read or write any data.

1.4.2 Authenticate with limited privileges

As a best practice, a service should have access to only the resources it needs.

To get more fine-grained control over the resources a Firebase app instance can access, use a unique identifier in your Security Rules to represent your service.

Then set up appropriate rules which grant your service access to the resources it needs. For example:

```
{
  "rules": {
    "public_resource": {
      ".read": true,
      ".write": true
    },
    "some_resource": {
      ".read": "auth.uid === 'my-service-worker'",
      ".write": false
    },
    "another_resource": {
      ".read": "auth.uid === 'my-service-worker'",
      ".write": "auth.uid === 'my-service-worker'"
    }
  }
}
```

Then, on your server, when you initialize the Firebase app, use the `asUser($uid)` method with the identifier you used to represent your service in your Security Rules.

```
use Kreait\Firebase\Factory;
use Kreait\Firebase\ServiceAccount;

$serviceAccount = ServiceAccount::fromJsonFile(__DIR__.'/google-service-account.json
↪');

$firebase = (new Factory)
    ->withServiceAccount($serviceAccount)
    ->asUser('my-service-worker')
    ->create();
```

1.4.3 Create custom tokens[2]

The Firebase Admin SDK has a built-in method for creating custom tokens. At a minimum, you need to provide a uid, which can be any string but should uniquely identify the user or device you are authenticating. These tokens expire after one hour.

```
$uid = 'some-uid';

$customToken = $firebase->getAuth()->createCustomToken($uid);
```

You can also optionally specify additional claims to be included in the custom token. For example, below, a premiumAccount field has been added to the custom token, which will be available in the auth / request.auth objects in your Security Rules:

```
$uid = 'some-uid';
$additionalClaims = [
    'premiumAccount' => true
];

$customToken = $firebase->getAuth()->createCustomToken($uid, $additionalClaims);

$customTokenString = (string) $customToken;
```

Note: This library uses lcobucci/jwt to work with JSON Web Tokens (JWT). You can find the usage instructions at https://github.com/lcobucci/jwt/blob/3.2/README.md.

1.4.4 Verify a Firebase ID Token[3]

If a Firebase client app communicates with your server, you might need to identify the currently signed-in user. To do so, verify the integrity and authenticity of the ID token and retrieve the uid from it. You can use the uid transmitted in this way to securely identify the currently signed-in user on your server.

Note: Many use cases for verifying ID tokens on the server can be accomplished by using Security Rules for the Firebase Realtime Database and Cloud Storage. See if those solve your problem before verifying ID tokens yourself.

Warning: The ID token verification methods included in the Firebase Admin SDKs are meant to verify ID tokens that come from the client SDKs, not the custom tokens that you create with the Admin SDKs. See Auth tokens for more information.

Use `Auth::verifyIdToken()` to verify an ID token:

```
use Firebase\Auth\Token\Exception\InvalidToken;

$idTokenString = '...';

try {
    $verifiedIdToken = $firebase->getAuth()->verifyIdToken($idTokenString);
```

(continues on next page)

[2] Google: Create custom tokens
[3] Google: Verify ID Tokens

(continued from previous page)

```
} catch (InvalidToken $e) {
    echo $e->getMessage();
}

$uid = $verifiedIdToken->getClaim('sub');
$user = $firebase->getAuth()->getUser($uid);
```

`Auth::verifyIdToken()` accepts up to three parameters:

Parameter	Type	Description
`idToken`	string|Token	**(required)** The ID token to verify
`checkIfRevoked`	boolean	(optional, default: `false`) check if the ID token is revoked
`allowTimeInconsistencies`	boolean	(optional, default: `false`) allow a token even if it's timestamps are invalid

Warning: Allowing time inconsistencies might impose a security risk. Do this only when you are not able to fix your environment's time to be consistent with Google's servers. This parameter is here for backwards compatibility reasons, and will be removed in the next major version. You should not rely on it.

Note: A leeway of 5 minutes is applied when verifying time based claims starting with release 4.25.0

Note: This library uses lcobucci/jwt to work with JSON Web Tokens (JWT). You can find the usage instructions at https://github.com/lcobucci/jwt/blob/3.2/README.md.

Caching Google's public keys

In order to verify ID tokens, the verifier makes a call to fetch Firebase's currently available public keys. The keys are cached in memory by default.

If you want to cache the public keys more effectively, you can provide any implementation of psr/simple-cache to the Firebase factory when creating your Firebase instance.

Here is an example using the Symfony Cache Component:

```
use Kreait\Firebase\Factory;
use Kreait\Firebase\ServiceAccount;
use Symfony\Component\Cache\Simple\FilesystemCache;

$cache = new FilesystemCache();

$serviceAccount = ServiceAccount::fromJsonFile(__DIR__.'/google-service-account.json
↪');

$firebase = (new Factory)
    ->withServiceAccount($serviceAccount)
    ->withVerifierCache($cache)
    ->create();
```

References

1.5 User management

The Firebase Admin SDK for PHP provides an API for managing your Firebase users with elevated privileges. The admin user management API gives you the ability to programmatically retrieve, create, update, and delete users without requiring a user's existing credentials and without worrying about client-side rate limiting.

```
use Kreait\Firebase\Factory;
use Kreait\Firebase\ServiceAccount;

$serviceAccount = ServiceAccount::fromJsonFile(__DIR__.'/google-service-account.json
↪');

$firebase = (new Factory)
    ->withServiceAccount($serviceAccount)
    ->create();

$auth = $firebase->getAuth();
```

1.5.1 User Records

`UserRecord` s returned by methods from the `Kreait\Firebase\Auth` class have the following signature:

```
{
    "uid": "jEazVdPDhqec0tnEOG7vM5wbDyU2",
    "email": "user@domain.tld",
    "emailVerified": true,
    "displayName": null,
    "photoUrl": null,
    "phoneNumber": null,
    "disabled": false,
    "metadata": {
        "createdAt": "2018-02-14T15:41:32+00:00",
        "lastLoginAt": "2018-02-14T15:41:32+00:00"
    },
    "providerData": [
        {
            "uid": "user@domain.tld",
            "displayName": null,
            "email": "user@domain.tld",
            "photoUrl": null,
            "providerId": "password",
            "phoneNumber": null
        }
    ],
    "passwordHash": "UkVEQUNURUQ=",
    "customClaims": null,
    "tokensValidAfterTime": "2018-02-14T15:41:32+00:00"
}
```

1.5.2 List users

To enhance performance and prevent memory issues when retrieving a huge amount of users, this methods returns a Generator.

```
$users = $auth->listUsers($defaultMaxResults = 1000, $defaultBatchSize = 1000);

foreach ($users as $user) {
    /** @var \Kreait\Firebase\Auth\UserRecord $user */
    // ...
}
// or
array_map(function (\Kreait\Firebase\Auth\UserRecord $user) {
    // ...
}, iterator_to_array($users));
```

1.5.3 Get information about a specific user

```
$user = $auth->getUser('some-uid');
$user = $auth->getUserByEmail('user@domain.tld');
```

1.5.4 Create a user

The Admin SDK provides a method that allows you to create a new Firebase Authentication user. This method accepts an object containing the profile information to include in the newly created user account:

```
$userProperties = [
    'email' => 'user@example.com',
    'emailVerified' => false,
    'phoneNumber' => '+15555550100',
    'password' => 'secretPassword',
    'displayName' => 'John Doe',
    'photoUrl' => 'http://www.example.com/12345678/photo.png',
    'disabled' => false,
];

$createdUser = $auth->createUser($userProperties);

// This is equivalent to:

$request = \Kreait\Auth\Request\CreateUser::new()
    ->withUnverifiedEmail('user@example.com')
    ->withPhoneNumber('+15555550100')
    ->withClearTextPassword('secretPassword')
    ->withDisplayName('John Doe')
    ->withPhotoUrl('http://www.example.com/12345678/photo.png');

$createdUser = $auth->createUser($request);
```

By default, Firebase Authentication will generate a random uid for the new user. If you instead want to specify your own uid for the new user, you can include in the properties passed to the user creation method:

```
$properties = [
    'uid' => 'some-uid',
```

(continues on next page)

(continued from previous page)

```
    // other properties
];

$request = \Kreait\Auth\Request\CreateUser::new()
    ->withUid('some-uid')
    // with other properties
;
```

Any combination of the following properties can be provided:

Property	Type	Description
`uid`	string	The uid to assign to the newly created user. Must be a string between 1 and 128 characters long, inclusive. If not provided, a random uid will be automatically generated.
`email`	string	The user's primary email. Must be a valid email address.
`emailVerified`	boolean	Whether or not the user's primary email is verified. If not provided, the default is false.
`phoneNumber`	string	The user's primary phone number. Must be a valid E.164 spec compliant phone number.
`password`	string	The user's raw, unhashed password. Must be at least six characters long.
`displayName`	string	The users' display name.
`photoURL`	string	The user's photo URL.
`disabled`	boolean	Whether or not the user is disabled. true for disabled; false for enabled. If not provided, the default is false.

Note: All of the above properties are optional. If a certain property is not specified, the value for that property will be empty unless a default is mentioned in the above table.

Note: If you provide none of the properties, an anonymous user will be created.

1.5.5 Update a user

Updating a user works exactly as creating a new user, except that the `uid` property is required:

```
$uid = 'some-uid';
$properties = [
    'displayName' => 'New display name'
];

$updatedUser = $auth->updateUser($uid, $properties);

$request = \Kreait\Auth\Request\UpdateUser::new()
    ->withDisplayName('New display name');

$updatedUser = $auth->updateUser($uid, $request);
```

In addition to the properties of a create request, the following properties can be provided:

Property	Type	Description
`deletePhotoUrl`	boolean	Whether or not to delete the user's photo.
`deleteDisplayName`	boolean	Whether or not to delete the user's display name.
`deletePhoneNumber`	boolean	Whether or not to delete the user's phone number.
`deleteProvider`	stringlarray	One or more identity providers to delete.
`customAttributes`	array	A list of custom attributes which will be available in a User's ID token.

1.5.6 Change a user's password

```
$uid = 'some-uid';

$updatedUser = $auth->changeUserPassword($uid, 'new password');
```

1.5.7 Change a user's email

```
$uid = 'some-uid';

$updatedUser = $auth->changeUserEmail($uid, 'user@domain.tld');
```

1.5.8 Disable a user

```
$uid = 'some-uid';

$updatedUser = $auth->disableUser($uid);
```

1.5.9 Enable a user

```
$uid = 'some-uid';

$updatedUser = $auth->enableUser($uid);
```

1.5.10 Update custom attributes

```
$uid = 'some-uid';
$customAttributes = [
    'admin' => true,
    'groupId' => '1234'
];

$updatedUser = $auth->setCustomUserAttributes($uid, $customAttributes);
$userWithDeletedCustomAttributes = $auth->deleteCustomUserAttributes($uid);
```

Note: Learn more about custom attributes/claims in the official documentation: Control Access with Custom Claims and Security Rules

1.5.11 Delete a user

```
$uid = 'some-uid';

$auth->deleteUser($uid);
```

1.5.12 Verify a password

Warning: This method has the side effect of changing the last login timestamp of the given user. The recommended way to authenticate users in a client/server environment is to use a Firebase Client SDK to authenticate the user and to send an ID Token generated by the client back to the server.

```
try {
    $user = $auth->verifyPassword($email, $password);
} catch (Kreait\Firebase\Exception\Auth\InvalidPassword $e) {
    echo $e->getMessage();
}
```

1.5.13 Verify an email address

Unless a user verifies the email address assigned to them, their email address will be markes as `unverified`.

You can send a verification email to a user with the following method:

```
$auth->sendEmailVerification($uid);

// The method has an optional second parameter to specify where the user should be
↪redirected
// to after they have have verified their email address
$auth->sendEmailVerification($uid, 'https//my-application.com/email-verified');

// The method has an optional third parameter to specifiy the locale of the sent email
// Without it, your Firebase project's configured default language will be used
$auth->sendEmailVerification('user@domain.tld', null, 'de');
```

1.5.14 Send a password reset email

You can send an email allowing a user to reset their password with the following method:

```
$auth->sendPasswordResetEmail('user@domain.tld');

// The method has an optional second parameter to specify where the user should be
↪redirected
// to after they have have reset their password
$auth->sendPasswordResetEmail('user@domain.tld', 'https//my-application.com/password-
↪reset');

// The method has an optional third parameter to specifiy the locale of the sent email
// Without it, your Firebase project's configured default language will be used
$auth->sendPasswordResetEmail('user@domain.tld', null, 'fr');
```

1.5.15 Invalidate user sessions[1]

This will revoke all sessions for a specified user and disable any new ID tokens for existing sessions from getting minted. **Existing ID tokens may remain active until their natural expiration (one hour).** To verify that ID tokens are revoked, use `Auth::verifyIdToken()` with the second parameter set to `true`.

If the check fails, a `RevokedIdToken` exception will be thrown.

```
use Kreait\Firebase\Exception\Auth\RevokedIdToken;

$idTokenString = '...';

$verifiedIdToken = $firebase->getAuth()->verifyIdToken($idTokenString);

$uid = $verifiedIdToken->getClaim('sub');

$firebase->getAuth()->revokeRefreshTokens($uid);

try {
    $verifiedIdToken = $firebase->getAuth()->verifyIdToken($idTokenString, true);
} catch (RevokedIdToken $e) {
    echo $e->getMessage();
}
```

Note: Because Firebase ID tokens are stateless JWTs, you can determine a token has been revoked only by requesting the token's status from the Firebase Authentication backend. For this reason, performing this check on your server is an expensive operation, requiring an extra network round trip. You can avoid making this network request by setting up Firebase Rules that check for revocation rather than using the Admin SDK to make the check.

For more information, please visit Google: Detect ID token revocation in Database Rules

---

References

1.6 Storage

Cloud Storage for Firebase stores your data in Google Cloud Storage, an exabyte scale object storage solution with high availability and global redundancy.

Before you start, please read about Firebase Cloud Storage in the official documentation:

- Firebase Cloud Storage
- Introduction to the Admin Cloud Storage API

You can work with your Firebase application's storage by invoking the `getStorage()` method of your Firebase instance:

```
use Kreait\Firebase;

$firebase = (new Firebase\Factory())->create();
$storage = $firebase->getStorage();
```

[1] Google: Revoke refresh tokens

1.6.1 Default Storage bucket

The SDK assumes that your project's default storage bucket name has the format `<project-id>.appspot.com` and will configure the `$firebase` instance accordingly.

If you want to change the default bucket your instance works with, you can specify the name when using the factory:

```
use Kreait\Firebase;

$firebase = (new Firebase\Factory())
    ->withDefaultStorageBucket('another-default-bucket')
    ->create();
```

You can access the files on your storage in the following ways:

- via Google Cloud Storage APIs
- via the Filesystem abstraction provided by the PHP League's league/flysystem and superbalist/flysystem-google-storage

1.6.2 Google Cloud Storage API

Read about the usage of the Google Cloud Storage API in the official documentation.

```
// Get the default bucket
$bucket = $storage->getBucket();

// Get the bucket with the given name
$bucket = $storage->getBucket('another-bucket');
```

1.6.3 The PHP League's Flysystem

Read about the usage of the PHP League's Flysystem at http://flysystem.thephpleague.com/api/.

```
// Get the default filesystem
$filesystem = $storage->getFilesystem()

// Get the bucket with the given name
$filesystem = $storage->getFilesystem('another-bucket');
```

1.7 Remote Config

Change the behavior and appearance of your app without publishing an app update.

Firebase Remote Config is a cloud service that lets you change the behavior and appearance of your app without requiring users to download an app update. When using Remote Config, you create in-app default values that control the behavior and appearance of your app.

Before you start, please read about Firebase Remote Config in the official documentation:

- Firebase Remote Config

1.7.1 Before you begin

For Firebase projects created before the March 7, 2018 release of the Remote Config REST API, you must enable the API in the Google APIs console.

1. Open the Firebase Remote Config API page in the Google APIs console.
2. When prompted, select your Firebase project. (Every Firebase project has a corresponding project in the Google APIs console.)
3. Click Enable on the Firebase Remote Config API page.

You can work with your Firebase application's Remote Config by invoking the `getRemoteConfig()` method of your Firebase instance:

```
use Kreait\Firebase;

$firebase = (new Firebase\Factory())->create();
$remoteConfig = $firebase->getRemoteConfig();
```

1.7.2 Get the Remote Config

```
$template = $remoteConfig->get();
echo json_encode($template, JSON_PRETTY_PRINT);
```

1.7.3 Create a new Remote Config

```
use Kreait\Firebase\RemoteConfig;

$template = RemoteConfig\Template::new();
```

1.7.4 Add a condition

```
use Kreait\Firebase\RemoteConfig;

$germanLanguageCondition = RemoteConfig\Condition::named('lang_german')
    ->withExpression("device.language in ['de', 'de_AT', 'de_CH']")
    ->withTagColor(TagColor::ORANGE); // The TagColor is optional

$template = $template->withCondition($germanLanguageCondition);
```

1.7.5 Add a parameter

```
use Kreait\Firebase\RemoteConfig;

$welcomeMessageParameter = Parameter::named('welcome_message')
        ->withDefaultValue('Welcome!')
        ->withDescription('This is a welcome message') // optional
;
```

1.7.6 Conditional values

```
use Kreait\Firebase\RemoteConfig;

$germanLanguageCondition = RemoteConfig\Condition::named('lang_german')
    ->withExpression("device.language in ['de', 'de_AT', 'de_CH']");

$germanWelcomeMessage = RemoteConfig\ConditionalValue::basedOn(
↪$germanLanguageCondition, 'Willkommen!');

$welcomeMessageParameter = Parameter::named('welcome_message')
        ->withDefaultValue('Welcome!')
        ->withConditionalValue($germanWelcomeMessage);

$template = $template
    ->withCondition($germanLanguageCondition)
    ->withParameter($welcomeMessageParameter);
```

Note: When you use a conditional value, make sure to add the corresponding condition to the template first.

1.7.7 Validation

Usually, the SDK will protect you from creating an invalid Remote Config template in the first place. If you want to be sure, you can validate the template with a call to the Firebase API:

```
use Kreait\Firebase\Exception\RemoteConfig\ValidationFailed;

try {
    $remoteConfig->validate($template);
} catch (ValidationFailed $e) {
    echo $e->getMessage();
}
```

Note: The `ValidationFailed` exception extends `Kreait\Firebase\Exception\RemoteConfigException`, so you can safely use the more generic exception type as well.

1.7.8 Publish the Remote Config

```
use Kreait\Firebase\Exception\RemoteConfigException

try {
    $remoteConfig->publish($template);
} catch (RemoteConfigException $e) {
    echo $e->getMessage();
}
```

1.7.9 Change history

Since August 23, 2018, Firebase provides a change history for your published Remote configs.

Versions

The following properties are available from a `Kreait\Firebase\RemoteConfig\Version` object:

```
$version->versionNumber();
$version->user(); // The user/service account the performed the change
$version->description();
$version->updatedAt();
$version->updateOrigin();
$version->updateType();
$version->rollBackSource();
```

List versions

To enhance performance and prevent memory issues when retrieving a huge amount of users, this methods returns a Generator.

```
$users = ;

foreach ($auth->listVersions() as $version) {
    /** @var \Kreait\Firebase\RemoteConfig\Version $user */
    // ...
}
// or
array_map(function (\Kreait\Firebase\RemoteConfig\Version $user) {
    // ...
}, iterator_to_array($auth->listVersions()));
```

Get a specific version

```
$version = $remoteConfig->getVersion($versionNumber);
```

Rollback to a version

```
$template = $remoteConfig->rollbackToVersion($versionNumber);
```

1.8 Cloud Messaging

You can use the Firebase Admin SDK for PHP to send Firebase Cloud Messaging messages to end-user devices. Specifically, you can send messages to individual devices, named topics, or condition statements that match one or more topics.

Note: Sending messages to Device Groups is only possible with legacy protocols which are not supported by this SDK.

Before you start, please read about Firebase Remote Config in the official documentation:

- Introduction to Firebase Cloud Messaging
- Introduction to Admin FCM API

1.8.1 Getting started

After having initialized your Firebase project instance, you can access the Cloud Messaging component with `$firebase->getMessaging()`.

```
use Kreait\Firebase;
use Kreait\Firebase\Messaging\CloudMessage;

$firebase = (new Firebase\Factory())->create();
$messaging = $firebase->getMessaging();

$message = CloudMessage::withTarget(/* see sections below */)
    ->withNotification(Notification::create('Title', 'Body'))
    ->withData(['key' => 'value']);

$messaging->send($message);
```

A message must be an object implementing `Kreait\Firebase\Messaging\Message` or an array that can be parsed to a `Kreait\Firebase\Messaging\CloudMessage`.

1.8.2 Send messages to topics

Based on the publish/subscribe model, FCM topic messaging allows you to send a message to multiple devices that have opted in to a particular topic. You compose topic messages as needed, and FCM handles routing and delivering the message reliably to the right devices.

For example, users of a local weather forecasting app could opt in to a "severe weather alerts" topic and receive notifications of storms threatening specified areas. Users of a sports app could subscribe to automatic updates in live game scores for their favorite teams.

Some things to keep in mind about topics:

- Topic messaging supports unlimited topics and subscriptions for each app.
- Topic messaging is best suited for content such as news, weather, or other publicly available information.
- Topic messages are optimized for throughput rather than latency. For fast, secure delivery to single devices or small groups of devices, target messages to registration tokens, not topics.

You can create a message to a topic in one of the following ways:

```
use Kreait\Firebase\Messaging\CloudMessage;

$topic = 'a-topic';

$message = CloudMessage::withTarget('topic', $topic)
    ->withNotification($notification) // optional
    ->withData($data) // optional
;

$message = CloudMessage::fromArray([
    'topic' => $topic,
    'notification' => [/* Notification data as array */], // optional
    'data' => [/* data array */], // optional
]);

$messaging->send($message);
```

1.8.3 Send conditional messages

Sometimes you want to send a message to a combination of topics. This is done by specifying a condition, which is a boolean expression that specifies the target topics. For example, the following condition will send messages to devices that are subscribed to `TopicA` and either `TopicB` or `TopicC`:

```
"'TopicA' in topics && ('TopicB' in topics || 'TopicC' in topics)"
```

FCM first evaluates any conditions in parentheses, and then evaluates the expression from left to right. In the above expression, a user subscribed to any single topic does not receive the message. Likewise, a user who does not subscribe to TopicA does not receive the message. These combinations do receive it:

- `TopicA` and `TopicB`
- `TopicA` and `TopicC`

```
use Kreait\Firebase\Messaging\CloudMessage;

$condition = "'TopicA' in topics && ('TopicB' in topics || 'TopicC' in topics)";

$message = CloudMessage::withTarget('condition', $condition)
    ->withNotification($notification) // optional
    ->withData($data) // optional
;

$message = CloudMessage::fromArray([
    'condition' => $condition,
    'notification' => [/* Notification data as array */], // optional
    'data' => [/* data array */], // optional
]);

$messaging->send($message);
```

1.8.4 Send messages to specific devices

The Admin FCM API allows you to send messages to individual devices by specifying a registration token for the target device. Registration tokens are strings generated by the client FCM SDKs for each end-user client app instance.

Each of the Firebase client SDKs are able to generate these registration tokens: iOS, Android, Web, C++, and Unity.

```
use Kreait\Firebase\Messaging\CloudMessage;

$deviceToken = '...';

$message = CloudMessage::withTarget('token', $deviceToken)
    ->withNotification($notification) // optional
    ->withData($data) // optional
;

$message = CloudMessage::fromArray([
    'token' => $deviceToken,
    'notification' => [/* Notification data as array */], // optional
    'data' => [/* data array */], // optional
]);

$messaging->send($message);
```

1.8.5 Send messages to multiple devices (Multicast)

Note: Support for multicast messages has been added in release 4.24.0

You can send send one message to multiple devices:

```
use Kreait\Firebase\Messaging\CloudMessage;

$deviceTokens = ['...', '...' /* ... */];

$message = CloudMessage::new(); // Any instance of Kreait\Messaging\Message

$sendReport = $messaging->sendMulticast($message, $deviceTokens);
```

The returned value is an instance of `Kreait\Firebase\Messaging\MulticastSendReport` and provides you with methods to determine the successes and failures of the multicasted message:

```
$report = $messaging->sendMulticast($message, $deviceTokens);

echo 'Successful sends: '.$report->successes()->count().PHP_EOL;
echo 'Failed sends: '.$report->failures()->count().PHP_EOL;

if ($report->hasFailures()) {
    foreach ($report->failures()->getItems() as $failure) {
        echo $failure->error()->getMessage().PHP_EOL;
    }
}
```

1.8.6 Adding a notification

A notification is an instance of `Kreait\Firebase\Messaging\Notification` and can be created in one of the following ways. The title and the body of a notification are both optional.

```
use Kreait\Firebase\Messaging\Notification;

$title = 'My Notification Title';
$body = 'My Notification Body';

$notification = Notification::fromArray([
    'title' => $title,
    'body' => $body
]);

$notification = Notification::create($title, $body);

$notification = Notification::create()
    ->withTitle($title)
    ->withBody($body);
```

Once you have created a message with one of the methods described below, you can attach the notification to it:

```
$message = $message->withNotification($notification);
```

1.8.7 Adding data

The data attached to a message must be an array of key-value pairs where all keys and values are strings.

Once you have created a message with one of the methods described below, you can attach data to it:

```
$data = [
    'first_key' => 'First Value',
    'second_key' => 'Second Value',
];

$message = $message->withData($data);
```

1.8.8 Changing the message target

You can change the target of an already created message with the `withChangedTarget()` method.

```
use Kreait\Firebase\Messaging\CloudMessage;

$deviceToken = '...';
$anotherDeviceToken = '...';

$message = CloudMessage::withTarget('token', $deviceToken)
    ->withNotification(['title' => 'My title', 'body' => 'My Body'])
;

$messaging->send($message);

$sameMessageToDifferentTarget = $message->withChangedTarget('token',
↪$anotherDeviceToken);
```

1.8.9 Adding target platform specific configuration

You can target platforms specific configuration to your messages.

Android

You can find the full Android configuration reference in the official documentation: REST Resource: projects.messages.AndroidConfig

```
use Kreait\Firebase\Messaging\AndroidConfig;

// Example from https://firebase.google.com/docs/cloud-messaging/admin/send-messages
↪#android_specific_fields
$config = AndroidConfig::fromArray([
    'ttl' => '3600s',
    'priority' => 'normal',
    'notification' => [
        'title' => '$GOOG up 1.43% on the day',
        'body' => '$GOOG gained 11.80 points to close at 835.67, up 1.43% on the day.
↪',
        'icon' => 'stock_ticker_update',
        'color' => '#f45342',
```

(continues on next page)

(continued from previous page)

```
    ],
]);

$message = $message->withAndroidConfig($config);
```

APNs

You can find the full APNs configuration reference in the official documentation: REST Resource: projects.messages.ApnsConfig

```
use Kreait\Firebase\Messaging\ApnsConfig;

// Example from https://firebase.google.com/docs/cloud-messaging/admin/send-messages
↪#apns_specific_fields
$config = ApnsConfig::fromArray([
    'headers' => [
        'apns-priority' => '10',
    ],
    'payload' => [
        'aps' => [
            'alert' => [
                'title' => '$GOOG up 1.43% on the day',
                'body' => '$GOOG gained 11.80 points to close at 835.67, up 1.43% on␣
↪the day.',
            ],
            'badge' => 42,
        ],
    ],
]);

$message = $message->withApnsConfig($config);
```

WebPush

You can find the full WebPush configuration reference in the official documentation: REST Resource: projects.messages.Webpush

```
use Kreait\Firebase\Messaging\WebPushConfig;

// Example from https://firebase.google.com/docs/cloud-messaging/admin/send-messages
↪#webpush_specific_fields
$config = ApnsConfig::fromArray([
    'notification' => [
        'title' => '$GOOG up 1.43% on the day',
        'body' => '$GOOG gained 11.80 points to close at 835.67, up 1.43% on the day.
↪',
        'icon' => 'https://my-server/icon.png',
    ],
    'fcm_options' => [
        'link' => 'https://my-server/some-page',
    ],
]);

$message = $message->withWebPushConfig($config);
```

1.8.10 Sending a fully configured raw message

Note: The message will be parsed and validated by the SDK.

```
$firebase
    ->getMessaging()
    ->send([
        'topic' => 'my-topic',
        // 'condition' => "'TopicA' in topics && ('TopicB' in topics || 'TopicC' in␣
↪topics)",
        // 'token' => '...',
        'notification' => [
            'title' => 'Notification title',
            'body' => 'Notification body',
        ],
        'data' => [
            'key_1' => 'Value 1',
            'key_2' => 'Value 2',
        ],
        'android' => [
            'ttl' => '3600s',
            'priority' => 'normal',
            'notification' => [
                'title' => '$GOOG up 1.43% on the day',
                'body' => '$GOOG gained 11.80 points to close at 835.67, up 1.43% on␣
↪the day.',
                'icon' => 'stock_ticker_update',
                'color' => '#f45342',
            ],
        ],
        'apns' => [
            'headers' => [
                'apns-priority' => '10',
            ],
            'payload' => [
                'aps' => [
                    'alert' => [
                        'title' => '$GOOG up 1.43% on the day',
                        'body' => '$GOOG gained 11.80 points to close at 835.67, up 1.
↪43% on the day.',
                    ],
                    'badge' => 42,
                ],
            ],
        ],
        'webpush' => [
            'notification' => [
                'title' => '$GOOG up 1.43% on the day',
                'body' => '$GOOG gained 11.80 points to close at 835.67, up 1.43% on␣
↪the day.',
                'icon' => 'https://my-server/icon.png',
            ],
            'fcm_options' => [
                'link' => 'https://my-server/some-page',
            ],
        ],
```

(continues on next page)

(continued from previous page)

```
    ])
```

1.8.11 Validating messages

You can validate a message by sending a validation-only request to the Firebase REST API. If the message is invalid, a *KreaitFirebaseExceptionMessagingInvalidMessage* exception is thrown, which you can catch to evaluate the raw error message(s) that the API returned.

```
use Kreait\Firebase\Exception\Messaging\InvalidMessage;

try {
    $firebase->getMessaging()->validate($message);
} catch (InvalidMessage $e) {
    print_r($e->errors());
}
```

1.8.12 Topic management

Subscribe to a topic

You can subscribe one or multiple devices to a topic by passing registration tokens to the `subscribeToTopic()` method.

```
$topic = 'my-topic';
$registrationTokens = [
    // ...
];

$firebase
    ->getMessaging()
    ->subscribeToTopic($topic, $registrationTokens);
```

Note: You can subscribe up to 1,000 devices in a single request. If you provide an array with over 1,000 registration tokens, the operation will fail with an error.

Unsubscribe from a topic

You can unsubscribe one or multiple devices from a topic by passing registration tokens to the `unsubscribeFromTopic()` method.

```
$topic = 'my-topic';
$registrationTokens = [
    // ...
];

$firebase
    ->getMessaging()
    ->unsubscribeFromTopic($topic, $registrationTokens);
```

Note: You can unsubscribe up to 1,000 devices in a single request. If you provide an array with over 1,000 registration tokens, the operation will fail with an error.

1.9 Tutorials

You can find an example project implementing the Firebase Admin SDK for PHP at https://github.com/jeromegamez/firebase-php-examples .

In addition, the SDK has been featured in the following tutorials:

- How to integrate Laravel with Google Firebase by Javier Núñez (April 2019)
- Integrate Firebase With PHP and Optimize Your Real Time Communication by Shahroze Nawaz (November 2018)
- Firebase and PHP Video Series by Arthur Mann (August 2018)
- Connect Laravel with Firebase Real Time Database by Pardeep Kumar (March 2018)

Note: Do you know another tutorial that is not featured in this list? Then please consider adding it by creating a Pull Request in the GitHub Repository of this project.

1.10 Troubleshooting

1.10.1 PHP Parse Error/PHP Syntax Error

If you're getting an error in the likes of

```
PHP Parse error: syntax error, unexpected ':', expecting ';' or '{' in ...
```

the environment you are running the script in does not use PHP 7.x. You can check this by adding the line

```
echo phpversion(); exit;
```

somewhere in your script.

1.10.2 Class 'Kreait\Firebase\ . . . ' not found

You are not using the latest release of the SDK, please update your composer dependencies.

1.10.3 Call to undefined function `openssl_sign()`

You need to install the OpenSSL PHP Extension: http://php.net/openssl

1.10.4 cURL error XX: ...

If you receive a `cURL error XX: ...`, make sure that you have a current CA Root Certificates bundle on your system and that PHP uses it.

To see where PHP looks for the CA bundle, check the output of the following command:

```
var_dump(openssl_get_cert_locations());
```

which should lead to an output similar to this:

```
array(8) {
    'default_cert_file' =>
    string(32) "/usr/local/etc/openssl/cert.pem"
    'default_cert_file_env' =>
    string(13) "SSL_CERT_FILE"
    'default_cert_dir' =>
    string(29) "/usr/local/etc/openssl/certs"
    'default_cert_dir_env' =>
    string(12) "SSL_CERT_DIR"
    'default_private_dir' =>
    string(31) "/usr/local/etc/openssl/private"
    'default_default_cert_area' =>
    string(23) "/usr/local/etc/openssl"
    'ini_cafile' =>
    string(0) ""
    'ini_capath' =>
    string(0) ""
}
```

Now check if the file given in the `default_cert_file` field actually exists. Create a backup of the file, download the current CA bundle from https://curl.haxx.se/ca/cacert.pem and put it where `default_cert_file` points to.

If the problem still occurs, another possible solution is to configure the `curl.cainfo` setting in your `php.ini`:

```
[curl]
curl.cainfo = /absolute/path/to/cacert.pem
```

1.10.5 ID Tokens are issued in the future

When ID Token verification fails because of an `IssuedInTheFuture` exception, this is an indication that the system time in your environment is not set correctly.

If you chose to ignore the issue, you can catch the exception and return the ID token nonetheless:

```
use Firebase\Auth\Token\Exception\InvalidToken;
use Firebase\Auth\Token\Exception\IssuedInTheFuture;

try {
    return $firebase->getAuth()->verifyIdToken($idTokenString);
} catch (IssuedInTheFuture $e) {
    return $e->getToken();
} catch (InvalidIdToken $e) {
    echo $e->getMessage();
    exit;
}
```

1.10.6 "403 Forbidden" Errors

Under the hood, a Firebase project is actually a Google Cloud project with pre-defined and pre-allocated permissions and resources.

When Google adds features to its product line, it is possible that you have to manually configure your Firebase/Google Cloud Project to take advantage of those new features.

When a request to the Firebase APIs fails, please make sure that the according Google Cloud API is enabled for your project:

- Firebase Services: https://console.cloud.google.com/apis/library/firebase.googleapis.com
- Cloud Messaging (FCM): https://console.cloud.google.com/apis/library/fcm.googleapis.com
- Dynamic Links: https://console.cloud.google.com/apis/library/firebasedynamiclinks.googleapis.com
- Firestore: https://console.cloud.google.com/apis/library/firestore.googleapis.com
- Realtime Database Rules: https://console.cloud.google.com/apis/library/firebaserules.googleapis.com
- Remote Config: https://console.cloud.google.com/apis/library/firebaseremoteconfig.googleapis.com
- Storage: https://console.cloud.google.com/apis/library/storage-component.googleapis.com