Firebase Admin SDK for PHP

Jun 03, 2023

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

Quick Start

```php
use Kreait\Firebase\Factory;

$factory = (new Factory)
    ->withServiceAccount('/path/to/firebase_credentials.json')
    ->withDatabaseUri('https://my-project-default-rtdb.firebaseio.com');

$auth = $factory->createAuth();
$realtimeDatabase = $factory->createDatabase();
$cloudMessaging = $factory->createMessaging();
$remoteConfig = $factory->createRemoteConfig();
$cloudStorage = $factory->createStorage();
$firestore = $factory->createFirestore();
```

CHAPTER 2

User Guide

2.1 Overview

2.1.1 Requirements

- PHP 8.1.x or 8.2.x
- 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.

2.1.2 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.

If you want to use the SDK within a Framework, please follow the installation instructions here:

- **Laravel**: kreait/laravel-firebase
- **Symfony**: kreait/firebase-bundle

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

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.

2.1.3 Usage examples

You can find usage examples in the tests directory of this project's GitHub repository.

2.1.4 Issues/Support

- For bugs and past issues: Github issue tracker
- For questions about Firebase in general: Stack Overflow and the Firebase Slack Community.

2.1.5 License

Licensed using the MIT license.

2.1.6 Contributing

Guidelines

1. The SDK utilizes PSR-4, PSR-7 and PSR-12.
2. This SDK has a minimum PHP version requirement of PHP 8.1.
3. All pull requests should 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 Standards Fixer to ensure a uniform coding style. Apply coding standard fixed using the Makefile:

```
make cs
```

from the root of the project.

2.2 Setup

2.2.1 Google Service Account

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

The SDK is able to auto-discover 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 or a JSON string (not recommended) is provided by a `GOOGLE_APPLICATION_CREDENTIALS` variable.
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 auto-discovery is not wanted, you can generate a private key file in JSON format and provide it to the factory directly. To generate a private key file for your service account:

1. Open https://console.firebase.google.com/project/_/settings/serviceaccounts/adminsdk and select the project you want to generate a private key file for.
2. Click **Generate New Private Key**, then confirm by clicking **Generate Key**
3. Securely store the JSON file containing the key.

Note: You should store the JSON file outside of your code repository to avoid accidentally exposing it to the outside world.

You can then configure the SDK to use this Service Account:

With the SDK

```php
use Kreait\Firebase\Factory;

$factory = (new Factory)->withServiceAccount('/path/to/firebase_credentials.json');
```

With the Symfony Bundle

Please see https://github.com/kreait/firebase-bundle#configuration

With the Laravel/Lumen Package

Please see https://github.com/kreait/laravel-firebase#configuration

2.2.2 Project ID

Note: It is not necessary to explicitly configure the project ID in most cases.

Service Account credentials usually include the ID of the Google Cloud Project your Firebase project belongs to.

If you use another type of credential, it might be necessary to provide it manually to the Firebase Factory.

```
use Kreait\Firebase\Factory;

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

You can also set a `GOOGLE_CLOUD_PROJECT=<project-id>` environment variable before instantiating a component with the factory.

2.2.3 Realtime Database URI

Note: You can find the URI for your Realtime Database at https://console.firebase.google.com/project/_/database. For recently created Firebase projects the default database URI usually has the format `https://<project-id>-default-rtdb.firebaseio.com`. Databases in projects created before September 2020 had the default database URI `https://<project-id>.firebaseio.com`.

For backward compatibility reasons, if you don't specify a database URI, the SDK will use the project ID defined in the Service Account JSON file to automatically generate it.

```
use Kreait\Firebase\Factory;

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

2.2.4 Caching

Authentication tokens

Before connecting to the Firebase APIs, the SDK fetches an authentication token for your credentials. This authentication token is cached in-memory so that it can be re-used during the same process.

If you want to cache authentication tokens more effectively, you can provide any implementation of psr/cache to the Firebase factory when creating your Firebase instance.

Note: Authentication tokens are cached in-memory by default. For Symfony and Laravel, the Framework's cache will automatically be used.

For Symfony and Laravel, the Framework's cache will automatically be used.

Here is an example using the Symfony Cache Component:

```
use Symfony\Component\Cache\Simple\FilesystemCache;

$factory = $factory->withAuthTokenCache(new FilesystemCache());
```

ID Token Verification

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.

Note: Public keys tokens are cached in-memory by default. For Symfony and Laravel, the Framework's cache will automatically be used.

Here is an example using the Symfony Cache Component:

```
use Symfony\Component\Cache\Simple\FilesystemCache;

$factory = $factory->withVerifierCache(new FilesystemCache());
```

2.2.5 End User Credentials

Note: While theoretically possible, it's not recommended to use end user credentials in the context of a Server-to-Server backend application.

When using End User Credentials (for example if you set you application default credentials locally with `gcloud auth application-default login`), you need to provide the ID of the project you want to access directly and suppress warnings triggered by the Google Auth Component:

```
use Kreait\Firebase\Factory;

putenv('SUPPRESS_GCLOUD_CREDS_WARNING=true');

// This will use the project defined in the Service Account
// credentials files by default
$base = (new Factory())->withProjectId('firebase-project-id');
```

2.2.6 HTTP Client Options

You can configure the behavior of the Guzzle HTTP Client performing the API requests by passing an instance of *KreaitFirebaseHttpHttpClientOptions* to the factory before creating a service.

```
use Kreait\Firebase\Http\HttpClientOptions;

$options = HttpClientOptions::default();

// Set the maximum amount of seconds (float) that can pass before
// a request is considered timed out
```

(continues on next page)

(continued from previous page)

```
// (default: indefinitely)
$options = $options->withTimeOut(3.5);

// Use a proxy that all API requests should be passed through.
// (default: none)
$options = $options->withProxy('tcp://<host>:<port>');

$factory = $factory->withHttpClientOptions($options);

// Newly created services will now use the new HTTP options
$realtimeDatabase = $factory->createDatabase();
```

Setting Guzzle Config Options

In addition to the explicit settings above, you can fully customize the configuration of the Guzzle HTTP Client:

```
use Kreait\Firebase\Http\HttpClientOptions;

$options = HttpClientOptions::default()
    ->withGuzzleConfigOption('single', 'value')
    ->withGuzzleConfigOptions([
        'first' => 'value',
        'second' => 'value,
    ]);
```

Note: You can find all Guzzle Config Options at Guzzle: Request Options

Adding Guzzle Middlewares

You can also add middlewares to the Guzzle HTTP Client:

```
use Kreait\Firebase\Http\HttpClientOptions;

$options = HttpClientOptions::default();

# Adding a single middleware
$options = $options->withGuzzleMiddleware($myMiddleware, 'my_middleware'); // The␣
↪name can be omitted

# Adding multiple middlewares
$options = $options->withGuzzleMiddlewares([
    # Just providing the middleware
    $myMiddleware,
    # Alternative notation:
    ['middleware' => $myMiddleware]
    # Providing a named middleware
    ['middleware' => $myMiddleware, 'name' => 'my_middleware],
]);
```

Note: You can find more information about Guzzle Middlewares at Guzzle: Handlers and Middleware

2.2.7 Logging

In order to log API requests to the Firebase APIs, you can provide the factory with loggers implementing `Psr\Log\LoggerInterface`.

The following examples use the Monolog logger, but work with any PSR-3 log implementation.

```
use GuzzleHttp\MessageFormatter;
use Kreait\Firebase\Factory;
use Monolog\Logger;
use Monolog\Handler\StreamHandler;

$httpLogger = new Logger('firebase_http_logs');
$httpLogger->pushHandler(new StreamHandler('path/to/firebase_api.log', Logger::INFO));

// Without further arguments, requests and responses will be logged with basic
// request and response information. Successful responses will be logged with
// the 'info' log level, failures (Status code >= 400) with 'notice'
$factory = $factory->withHttpLogger($httpLogger);

// You can configure the message format and log levels individually
$messageFormatter = new MessageFormatter(MessageFormatter::SHORT);
$factory = $factory->withHttpLogger(
    $httpLogger, $messageFormatter, $successes = 'debug', $errors = 'warning'
);

// You can provide a separate logger for detailed HTTP message logs
$httpDebugLogger = new Logger('firebase_http_debug_logs');
$httpDebugLogger->pushHandler(
    new StreamHandler('path/to/firebase_api_debug.log',
    Logger::DEBUG)
);

// Logs will include the full request and response headers and bodies
$factory = $factory->withHttpDebugLogger($httpDebugLogger)
```

2.3 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

2.3.1 Initializing the Messaging component

With the SDK

```
$messaging = $factory->createMessaging();
```

With Dependency Injection (Symfony Bundle/Laravel/Lumen Package)

```
use Kreait\Firebase\Contract\Messaging;

class MyService
{
    public function __construct(Messaging $messaging)
    {
        $this->messaging = $messaging;
    }
}
```

With the Laravel `app()` helper (Laravel/Lumen Package)

```
$messaging = app('firebase.messaging');
```

2.3.2 Getting started

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

$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`.

You can use `Kreait\Firebase\Messaging\RawMessageFromArray` to create a message without the SDK checking it for validity before sending it. This gives you full control over the sent message, but also means that you have to send/validate a message in order to know if it's valid or not.

Note: If you notice that a field is not supported by the SDK yet, please open an issue on the issue tracker, so that others can benefit from it as well.

2.3.3 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);
```

2.3.4 Send conditional messages

Warning: OR-conditions are currently not processed correctly by the Firebase Rest API, leading to undelivered messages. This can be resolved by splitting up a message to an OR-condition into multiple messages to AND-conditions. So one conditional message to `'a' in topics || 'b' in topics` should be sent as two messages to the conditions `'a' in topics && !('b' in topics)` and `'b' in topics && !('a' in topics)`

References:

- https://github.com/firebase/quickstart-js/issues/183
- https://stackoverflow.com/a/52302136/284325

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
```

(continues on next page)

(continued from previous page)

```
    ->withData($data) // optional
;

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

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

2.3.5 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);
```

2.3.6 Send messages to multiple devices (Multicast)

You can send one message to up to 500 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;
```

(continues on next page)

(continued from previous page)

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

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

// The following methods return arrays with registration token strings
$successfulTargets = $report->validTokens(); // string[]

// Unknown tokens are tokens that are valid but not know to the currently
// used Firebase project. This can, for example, happen when you are
// sending from a project on a staging environment to tokens in a
// production environment
$unknownTargets = $report->unknownTokens(); // string[]

// Invalid (=malformed) tokens
$invalidTargets = $report->invalidTokens(); // string[]
```

2.3.7 Send multiple messages at once

You can send up to 500 prepared messages (each message has a token, topic or condition as a target) in one go:

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

$messages = [
    // Up to 500 items, either objects implementing Kreait\Firebase\Messaging\Message
    // or arrays that can be used to create valid to␣
↪Kreait\Firebase\Messaging\Cloudmessage instances
];

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

/** @var Kreait\Firebase\Messaging\MulticastSendReport $sendReport **/
$sendReport = $messaging->sendAll($messages);
```

2.3.8 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';
$imageUrl = 'http://lorempixel.com/400/200/';

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

(continues on next page)

(continued from previous page)

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

$changedNotification = $notification
    ->withTitle('Changed title')
    ->withBody('Changed body')
    ->withImageUrl('http://lorempixel.com/200/400/');
```

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

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

2.3.9 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);
```

2.3.10 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);
```

2.3.11 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

```php
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',
        'sound' => 'default',
    ],
]);

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

APNs

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

```php
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,
            'sound' => 'default',
        ],
    ],
]);

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

WebPush

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

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

(continues on next page)

(continued from previous page)

```
// Example from https://firebase.google.com/docs/cloud-messaging/admin/send-messages
↪#webpush_specific_fields
$config = WebPushConfig::fromArray([
    'notification' => [
        'title' => '$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);
```

2.3.12 Adding platform independent FCM options

You can find the full FCM Options configuration reference in the official documentation: REST Resource: projects.messages.fcm_options

```
use Kreait\Firebase\Messaging\FcmOptions;

$fcmOptions = FcmOptions::create()
    ->withAnalyticsLabel('my-analytics-label');
// or
$fcmOptions = [
    'analytics_label' => 'my-analytics-label';
];

$message = $message->withFcmOptions($fcmOptions);
```

2.3.13 Notification Sounds

The SDK provides helper methods to add sounds to messages:

- `CloudMessage::withDefaultSounds()`
- `AndroidConfig::withDefaultSound()`
- `AndroidConfig::withSound($sound)`
- `ApnsConfig::withDefaultSound()`
- `ApnsConfig::withSound($sound)`

Note: WebPush notification don't support the inclusion of sounds.

```
$message = CloudMessage::withTarget('token', $token)
    ->withNotification(['title' => 'Notification title', 'body' => 'Notification body
↪'])
    ->withDefaultSounds() // Enables default notifications sounds on iOS and Android␣
↪devices.
```

(continues on next page)

(continued from previous page)

```
    ->withApnsConfig(
        ApnsConfig::new()
            ->withSound('bingbong.aiff')
            ->withBadge(1)
    )
;
```

2.3.14 Message Priority

The SDK provides helper methods to define the priority of a message.

Note: You can learn more about message priorities for the different target platforms at Setting the priority of a message in the official Firebase documentation.

Note: Setting a message priority is optional. If you don't set a priority, the Firebase backend or the target platform uses their defined defaults.

Android

- `AndroidConfig::withNormalPriority()`
- `AndroidConfig::withHighPriority()`
- `AndroidConfig::withPriority(string $priority)`

iOS (APNS)

- `ApnsConfig::withPowerConservingPriority()`
- `ApnsConfig::withImmediatePriority()`
- `ApnsConfig::withPriority(string $priority)`

Web

- `WebPushConfig::withVeryLowUrgency()`
- `WebPushConfig::withLowUrgency()`
- `WebPushConfig::withNormalUrgency()`
- `WebPushConfig::withHighUrgency()`
- `WebPushConfig::withUrgency(string $urgency)`

Combined

- `CloudMessage::withLowestPossiblePriority()`
- `CloudMessage::withHighestPossiblePriority()`

Example

```
$message = CloudMessage::withTarget('token', $token)
    ->withNotification([
        'title' => 'If you had an iOS device...',
        'body' => '... you would have received a very important message'
    ])
    ->withLowestPossiblePriority()
    ->withApnsConfig(
        ApnsConfig::new()
            ->withImmediatePriority()
            ->withNotification([
                'title => 'A very important message...',
                'body' => '... that requires your immediate attention.'
            ])
    )
;
```

2.3.15 Using Emojis

Firebase Messaging supports Emojis in Messages.

Note: You can find a full list of all currently available Emojis at https://www.unicode.org/emoji/charts/full-emoji-list.html

```
// You can copy and paste an emoji directly into you source code
$text = "This is an emoji ";
$text = "This is an emoji \u{1F600}";
```

2.3.16 Sending a raw/custom messages

Instead of composing messages with the help of the `CloudMessage` builder, you can use `RawMessageFromArray` as a wrapper for a pre-compiled message payload. Alternatively, you can implement custom messages by implementing the `Kreait\Firebase\Messaging\Message` interface.

```
use Kreait\Firebase\Messaging\RawMessageFromArray;

$message = new RawMessageFromArray([
        'notification' => [
            // https://firebase.google.com/docs/reference/fcm/rest/v1/projects.
↪messages#notification
            'title' => 'Default title',
            'body' => 'Default body',
        ],
        'data' => [
            'key' => 'Value',
        ],
        'android' => [
            // https://firebase.google.com/docs/reference/fcm/rest/v1/projects.
↪messages#androidconfig
            'notification' => [
                'title' => 'Android Title',
```

(continues on next page)

(continued from previous page)

```
                'body' => 'Android Body',
            ],
        ],
        'apns' => [
            // https://firebase.google.com/docs/reference/fcm/rest/v1/projects.
↪messages#apnsconfig
            'payload' => [
                'aps' => [
                    'alert' => [
                        'title' => 'iOS Title',
                        'body' => 'iOS Body',
                    ],
                ],
            ],
        ],
        'webpush' => [
            // https://firebase.google.com/docs/reference/fcm/rest/v1/projects.
↪messages#webpushconfig
            'notification' => [
                'title' => 'Webpush Title',
                'body' => 'Webpush Body'
            ],
        ],
        'fcm_options' => [
            // https://firebase.google.com/docs/reference/fcm/rest/v1/projects.
↪messages#fcmoptions
            'analytics_label' => 'some-analytics-label'
        ]
    ]);

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

2.3.17 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 {
    $messaging->validate($message);
    // or
    $messaging->send($message, $validateOnly = true);
} catch (InvalidMessage $e) {
    print_r($e->errors());
}
```

You can also use the `send*` methods with an additional parameter:

```
$validateOnly = true;

$messaging->send($message, $validateOnly);
$messaging->sendMulticast($message, $tokens, $validateOnly);
$messaging->sendAll($messages, $validateOnly);
```

2.3.18 Validating Registration Tokens

If you have a set of registration tokens that you want to check for validity or if they are still registered to your project, you can use the *validateTokens()* method:

```
$tokens = [...];

$result = $messaging->validateRegistrationTokens($tokens);
```

The result is an array with three keys containing the checked tokens:

- `valid` contains all tokens that are valid and registered to the current Firebase project
- `unknown` contains all tokens that are valid, but **not** registered to the current Firebase project
- `invalid` contains all invalid (=malformed) tokens

2.3.19 Topic management

You can subscribe one or multiple devices to one or multiple messaging topics with the following methods:

```
$result = $messaging->subscribeToTopic($topic, $registrationTokenOrTokens);
$result = $messaging->subscribeToTopics($topics, $registrationTokenOrTokens);

$result = $messaging->unsubscribeFromTopic($topic, $registrationTokenOrTokens);
$result = $messaging->unsubscribeFromTopics($topics, $registrationTokenOrTokens);

$result = $messaging->unsubscribeFromAllTopics($registrationTokenOrTokens);
```

The result will return an array win which the keys are the topic names, and the values are the operation results for the individual tokens.

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.

2.3.20 App instance management

A registration token is related to an application that generated it. You can retrieve current information about an app instance by passing a registration token to the `getAppInstance()` method.

```
$registrationToken = '...';

$appInstance = $messaging->getAppInstance($registrationToken);
// Return the full information as provided by the Firebase API
$instanceInfo = $appInstance->rawData();

/* Example output for an Android application instance:
    [
      "applicationVersion" => "1060100"
      "connectDate" => "2019-07-21"
      "attestStatus" => "UNKNOWN"
      "application" => "com.vendor.application"
      "scope" => "*"
      "authorizedEntity" => "..."
```

(continues on next page)

(continued from previous page)

```
      "rel" => array:1 [
        "topics" => array:3 [
          "test-topic" => array:1 [
            "addDate" => "2019-07-21"
          ]
          "test-topic-5d35b46a15094" => array:1 [
            "addDate" => "2019-07-22"
          ]
          "test-topic-5d35b46b66c31" => array:1 [
            "addDate" => "2019-07-22"
          ]
        ]
      ]
      "connectionType" => "WIFI"
      "appSigner" => "..."
      "platform" => "ANDROID"
    ]
*/

/* Example output for a web application instance
    [
      "application" => "webpush"
      "scope" => ""
      "authorizedEntity" => "..."
      "rel" => array:1 [
        "topics" => array:2 [
          "test-topic-5d35b445b830a" => array:1 [
            "addDate" => "2019-07-22"
          ]
          "test-topic-5d35b446c0839" => array:1 [
            "addDate" => "2019-07-22"
          ]
        ]
      ]
      "platform" => "BROWSER"
    ]
*/
```

Note: As the data returned by the Google Instance ID API can return differently formed results depending on the application or platform, it is currently difficult to add reliable convenience methods for specific fields in the raw data.

Working with topic subscriptions

You can retrieve all topic subscriptions for an app instance with the `topicSubscriptions()` method:

```
$appInstance = $messaging->getAppInstance('<registration token>');

/** @var \Kreait\Firebase\Messaging\TopicSubscriptions $subscriptions */
$subscriptions = $appInstance->topicSubscriptions();

foreach ($subscriptions as $subscription) {
    echo "{$subscription->registrationToken()} is subscribed to {$subscription->
↪topic()}\n";
}
```

2.4 Cloud Firestore

This SDK provides a bridge to the google/cloud-firestore package. You can enable the component in the SDK by adding the package to your project dependencies:

```
composer require google/cloud-firestore
```

Note: The `google/cloud-firestore` package requires the gRPC PHP extension to be installed. You can find installation instructions for gRPC at github.com/grpc/grpc. The following projects aim to provide support for Firestore without the need to install the gRPC PHP extension, but have to be set up separately:

- bensontrent/firestore-php
- morrislaptop/firestore-php

Before you start, please read about Firestore in the official documentation:

- Official Documentation
- google/cloud-firestore on GitHub
- PHP API Documentation
- PHP Usage Examples

2.4.1 Initializing the Firestore component

With the SDK

```php
$firestore = $factory->createFirestore();
```

With Dependency Injection (Symfony Bundle/Laravel/Lumen Package)

```php
use Kreait\Firebase\Contract\Firestore;

class MyService
{
    public function __construct(Firestore $firestore)
    {
        $this->firestore = $firestore;
    }
}
```

With the Laravel `app()` helper (Laravel/Lumen Package)

```php
$firestore = app('firebase.firestore');
```

2.4.2 Getting started

```php
$database = $firestore->database();
```

`$database` is an instance of `Google\Cloud\Firestore\FirestoreClient`. Please refer to the links above for guidance on how to proceed from here.

2.5 Cloud Storage

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

This SDK provides a bridge to the google/cloud-storage package. You can enable the component in the SDK by adding the package to your project dependencies:

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

- Firebase Cloud Storage
- Introduction to the Admin Cloud Storage API
- PHP API Documentation
- PHP Usage examples

2.5.1 Initializing the Storage component

With the SDK

```
$storage = $factory->createStorage();
```

With Dependency Injection (Symfony Bundle/Laravel/Lumen Package)

```
use Kreait\Firebase\Contract\Storage;

class MyService
{
    public function __construct(Storage $storage)
    {
        $this->storage = $storage;
    }
}
```

With the Laravel `app()` helper (Laravel/Lumen Package)

```
$storage = app('firebase.storage');
```

2.5.2 Getting started

```
$storageClient = $storage->getStorageClient();
$defaultBucket = $storage->getBucket();
$anotherBucket = $storage->getBucket('another-bucket');
```

2.5.3 Default Storage bucket

Note: It is not necessary to change the default storage bucket in most cases.

The SDK assumes that your project's default storage bucket name has the format `<project-id>.appspot.com` and will configure the storage 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\Factory;

$storage = (new Factory())
    ->withDefaultStorageBucket('another-default-bucket')
    ->createStorage();
```

2.6 Realtime Database

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

2.6.1 Initializing the Realtime Database component

With the SDK

```
$database = $factory->createDatabase();
```

With Dependency Injection (Symfony Bundle/Laravel/Lumen Package)

```
use Kreait\Firebase\Contract\Database;

class MyService
{
    public function __construct(Database $database)
    {
        $this->database = $database;
    }
}
```

With the Laravel `app()` helper (Laravel/Lumen Package)

```
$database = app('firebase.database');
```

2.6.2 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();
```

(continues on next page)

(continued from previous page)

```
// 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

```
$database->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:

```
$database->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

```
$database->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"
    }
}
```

```
$database->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"
    }
}
```

```
$database->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

```
$database->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

```
$database->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

```
$database->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();
```

startAfter

Note: The `startAfter` query filter has been added to the Firebase JS SDK on 2021-02-11. This PHP SDK implements this in the same way as the other filters, but `startAfter` does not seem to have an effect when used with the Firebase REST API. If you happen to know why or you tried it and it does indeed work, please let me know via the SDK's git repo.

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

endAt

```
$database->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();
```

endBefore

Note: The `endBefore` query filter has been added to the Firebase JS SDK on 2021-02-11. This PHP SDK implements this in the same way as the other filters, but `endBefore` does not seem to have an effect when used with the Firebase REST API. If you happen to know why or you tried it and it does indeed work, please let me know via the SDK's git repo.

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

equalTo

```
$database->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();
```

2.6.3 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:

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

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

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

Update specific fields

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 = $database->getReference('posts')->push()->getKey();

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

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

Writing lists

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 = $database->getReference('posts')->push($postData);
```

(continues on next page)

(continued from previous page)

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

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

Delete data

You can delete a reference, including all data it contains, with the `remove()` method:

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

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

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

You can also delete in bulk using the function `removeChildren()`:

```
$data->getReference()->removeChildren([
    'posts/post1',
    'user-posts/f55dee9a-dd67-4e78-bd7d-f1b4be157a53/post1',
    'users/f55dee9a-dd67-4e78-bd7d-f1b4be157a53'
]);
```

2.6.4 Database transactions

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.

Replace data inside a transaction

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

$counterRef = $database->getReference('counter');
```

(continues on next page)

(continued from previous page)

```
$result = $database->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);

    return $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 = $database->getReference('to-be-deleted');

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

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

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

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

(continues on next page)

(continued from previous page)

```
        $transaction->set($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();

}
```

2.6.5 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 {
    $database->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();
    }
}
```

2.6.6 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();

// 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();
```

(continues on next page)

(continued from previous page)

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

$database->updateRules($ruleSet);

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

2.6.7 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, when instantiating the database component of the SDK, use the `withDatabaseAuthVariableOverride()` method to override the auth object used by your database rules. In this custom auth object, set the `uid` field to the identifier you used to represent your service in your Security Rules.

```
use Kreait\Firebase\Factory;

$factory = (new Factory)
```

(continues on next page)

(continued from previous page)

```
    ->withServiceAccount('/path/to/firebase_credentials.json')
    ->withDatabaseUri('https://my-project-default-rtdb.firebaseio.com');

$database = $factory
    ->withDatabaseAuthVariableOverride('my-service-worker')
    ->createDatabase();

// $database now only has access as defined in the Security Rules
```

In some cases, you may want to downscope the Admin SDKs to act as an unauthenticated client. You can do this by providing a value of `null` for the database auth variable override.

```
$database = $factory
    ->withDatabaseAuthVariableOverride(null)
    ->createDatabase();

// $database now only has access to public resources
```

2.7 Authentication

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

- Introduction to the Admin Database API
- Create custom tokens
- Verify ID Tokens
- Manage Session Cookies
- Revoke refresh tokens

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.

2.7.1 Initializing the Auth component

With the SDK

```
$auth = $factory->createAuth();
```

With Dependency Injection (Symfony Bundle/Laravel/Lumen Package)

```php
use Kreait\Firebase\Contract\Auth;

class MyService
{
    public function __construct(Auth $auth)
    {
        $this->auth = $auth;
    }
}
```

With the Laravel `app()` helper (Laravel/Lumen Package)

```php
$auth = app('firebase.auth');
```

2.7.2 Create custom tokens

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.

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

$customToken = $auth->createCustomToken($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:

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

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

$customTokenString = $customToken->toString();
```

Note: This library uses lcobucci/jwt to work with JSON Web Tokens (JWT). You can find the usage instructions at https://lcobucci-jwt.readthedocs.io/.

2.7.3 Verify a Firebase ID Token

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 Kreait\Firebase\Exception\Auth\FailedToVerifyToken;

$idTokenString = '...';

try {
    $verifiedIdToken = $auth->verifyIdToken($idTokenString);
} catch (FailedToVerifyToken $e) {
    echo 'The token is invalid: '.$e->getMessage();
}

$uid = $verifiedIdToken->claims()->get('sub');

$user = $auth->getUser($uid);
```

`Auth::verifyIdToken()` accepts the following 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
`leewayInSeconds`	positive-int|null	(optional, default: `null`) number of seconds to allow a token to be expired, in case that there is a clock skew between the signing and the verifying server.

Note: This library uses lcobucci/jwt to work with JSON Web Tokens (JWT). You can find the usage instructions at https://lcobucci-jwt.readthedocs.io/.

2.7.4 Custom Authentication Flows

Warning: It is recommended that you use the Firebase Client SDKs to perform user authentication. Once signed in via a client SDK, you should pass the logged-in user's current ID token to your PHP endpoint and *verify the ID token* with each request to your backend.

Each of the methods documented below will return an instance of `Kreait\Firebase\Auth\SignInResult\SignInResult` with the following accessors:

```
$signInResult->idToken(); // string|null
$signInResult->firebaseUserId(); // string|null
$signInResult->accessToken(); // string|null
$signInResult->refreshToken(); // string|null
$signInResult->data(); // array
$signInResult->asTokenResponse(); // array
```

`SignInResult::data()` returns the full payload of the response returned by the Firebase API, `SignInResult::asTokenResponse()` returns the Sign-In result in a format that can be returned to clients:

```
$tokenResponse = [
    'token_type' => 'Bearer',
    'access_token' => '...',
    'id_token' => '...',
    'refresh_token' => '...',
    'expires_in' => 3600,
];
```

Note: Not all sign-in methods return all types of tokens.

Anonymous Sign In

Note: This method will create a new user in the Firebase Auth User Database each time it is invoked

```
$signInResult = $auth->signInAnonymously();
```

Sign In with Email and Password

```
$signInResult = $auth->signInWithEmailAndPassword($email, $clearTextPassword);
```

Sign In with Email and Oob Code

```
$signInResult = $auth->signInWithEmailAndOobCode($email, $oobCode);
```

Sign In with a Custom Token

```
$signInResult = $auth->signInWithCustomToken($customToken);
```

Sign In with a Refresh Token

```
$signInResult = $auth->signInWithRefreshToken($refreshToken);
```

Sign In with IdP credentials

IdP (Identity Provider) credentials are credentials provided by authentication providers other than Firebase, for example Facebook, Github, Google or Twitter. You can find the currently supported authentication providers in the official Firebase documentation.

This could be useful if you already have "Sign in with X" implemented in your application, and want to authenticate the same user with Firebase.

Once you have received those credentials, you can use them to sign a user in with them:

```
$signInResult = $auth->signInWithIdpAccessToken($provider, string $accessToken,
↪$redirectUrl = null, ?string $oauthTokenSecret = null, ?string $linkingIdToken =␣
↪null, ?string $rawNonce = null);

$signInResult = $auth->signInWithIdpIdToken($provider, $idToken, $redirectUrl = null,␣
↪?string $linkingIdToken = null, ?string $rawNonce = null);
```

Sign In without a token

```
$signInResult = $auth->signInAsUser($userOrUid, array $claims = null);
```

Linking and Unlinking Identity Providers

For linking IdP you can add use any of above methods for signing in with IdP credentials, by providing the ID token of a user to link to as an additional parameter:

```
$signInResult = $auth->signInWithIdpAccessToken($provider, $accessToken, $redirectUrl␣
↪= null, $oauthTokenSecret = null, $linkingIdToken);
```

You can unlink a provider from a given user with the `unlinkProvider()` method:

```
$auth->unlinkProvider($uid, $provider)
```

2.7.5 Invalidate user sessions

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;

$auth->revokeRefreshTokens($uid);

try {
    $verifiedIdToken = $auth->verifyIdToken($idTokenString, $checkIfRevoked = 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

---

2.7.6 Session Cookies

Firebase Auth provides server-side session cookie management for traditional websites that rely on session cookies. This solution has several advantages over client-side short-lived ID tokens, which may require a redirect mechanism each time to update the session cookie on expiration:

- Improved security via JWT-based session tokens that can only be generated using authorized service accounts.
- Stateless session cookies that come with all the benefit of using JWTs for authentication. The session cookie has the same claims (including custom claims) as the ID token, making the same permissions checks enforceable on the session cookies.
- Ability to create session cookies with custom expiration times ranging from 5 minutes to 2 weeks.
- Flexibility to enforce cookie policies based on application requirements: domain, path, secure, httpOnly, etc.
- Ability to revoke session cookies when token theft is suspected using the existing refresh token revocation API.
- Ability to detect session revocation on major account changes.

You can learn more about Firebase Session Cookies in the official documentation:

- Manage Session Cookies

Warning: Creating and verifying session cookies when using tenants is currently not possible. Please follow this issue on GitHub or in the Google Issue Tracker for updates.

Create session cookie

Given an ID token sent to your server application from a client application, you can convert it to a session cookie:

```
use Kreait\Firebase\Auth\CreateSessionCookie\FailedToCreateSessionCookie;

$idToken = '...';

// The TTL must be between 5 minutes and 2 weeks and can be provided as
// an integer value in seconds or a DateInterval

$fiveMinutes = 300;
$oneWeek = new \DateInterval('P7D');

try {
    $sessionCookieString = $auth->createSessionCookie($idToken, $oneWeek);
} catch (FailedToCreateSessionCookie $e) {
    echo $e->getMessage();
}
```

Verify a Firebase Session Cookie

Use `Auth::verifySessionCookie()` to verify a Session Cookie:

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

$sessionCookieString = '...';

try {
```

(continues on next page)

(continued from previous page)

```
    $verifiedSessionCookie = $auth->verifySessionCookie($sessionCookieString);
} catch (FailedToVerifySessionCookie $e) {
    echo 'The Session Cookie is invalid: '.$e->getMessage();
}

$uid = $verifiedSessionCookie->claims()->get('sub');

$user = $auth->getUser($uid);
```

`Auth::verifySessionCookie()` accepts the following parameters:

Parameter	Type	Description
`sessionCookie`	string	**(required)** The Session Cookie to verify
`checkIfRevoked`	boolean	(optional, default: `false`) check if the ID token is revoked
`leewayInSeconds`	positive-int|null	(optional, default: `null`) number of seconds to allow a Session Cookie to be expired, in case that there is a clock skew between the signing and the verifying server.

Note: This library uses lcobucci/jwt to work with JSON Web Tokens (JWT). You can find the usage instructions at https://lcobucci-jwt.readthedocs.io/.

2.7.7 Tenant Awareness

Note: Multi-tenancy support requires Google Cloud's Identity Platform (GCIP). To learn more about GCIP, including pricing and features, see the GCIP documentation.

Before multi-tenancy can be used on a Google Cloud Identity Platform project, tenants must be allowed on that project via the Cloud Console UI.

In order to manage users, create custom tokens, verify ID tokens and sign in users in the scope of a tenant, you can configure the factory with a tenant ID:

```
$tenantUnawareAuth = $factory->createAuth();

$tenantAwareAuth = $factory
    ->withTenantId('my-tenant-id')
    ->createAuth();
```

2.8 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.

2.8.1 User Records

`UserRecord` s returned by methods from `Kreait\Firebase\Contract\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",
        "passwordUpdatedAt": "2018-02-14T15:42:19+00:00",
        "lastRefreshAt": "2018-02-14T15:42:19+00:00"
    },
    "providerData": [
        {
            "uid": "user@domain.tld",
            "displayName": null,
            "screenName": null,
            "email": "user@domain.tld",
            "photoUrl": null,
            "providerId": "password",
            "phoneNumber": null
        }
    ],
    "passwordHash": "UkVEQUNURUQ=",
    "customClaims": null,
    "tokensValidAfterTime": "2018-02-14T15:41:32+00:00"
}
```

2.8.2 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));
```

2.8.3 Query users

Listing all users with `listUser()` is fast an memory-efficient if you want to process a large number of users. However, if you prefer paginating over subsets of users with more parameters, you can use the `queryUsers()` method.

User queries can be created in two ways: by building a `UserQuery` object or by passing an array.

The following two snippets show all possible query modifiers with both ways:

```php
use Kreait\Firebase\Auth\UserQuery;

# Building a user query object
$userQuery = UserQuery::all()
    ->sortedBy(UserQuery::FIELD_USER_EMAIL)
    ->inDescendingOrder()
    // ->inAscendingOrder() # this is the default
    ->withOffset(1)
    ->withLimit(499); # The maximum supported limit is 500

# Using an array
$userQuery = [
    'sortBy' => UserQuery::FIELD_USER_EMAIL,
    'order' => UserQuery::ORDER_DESC,
    // 'order' => UserQuery::ORDER_DESC # this is the default
    'offset' => 1,
    'limit' => 499, # The maximum supported limit is 500
];
```

It is possible to sort by the following fields:

- `UserQuery::FIELD_CREATED_AT`
- `UserQuery::FIELD_LAST_LOGIN_AT`
- `UserQuery::FIELD_NAME`
- `UserQuery::FIELD_USER_EMAIL`
- `UserQuery::FIELD_USER_ID`

```php
$users = $auth->queryUsers($userQuery);
```

You can also filter by email, phone number or uid:

```php
use Kreait\Firebase\Auth\UserQuery;

$userQuery = UserQuery::all()->withFilter(UserQuery::FILTER_EMAIL, '<email>');
$userQuery = UserQuery::all()->withFilter(UserQuery::FILTER_PHONE_NUMBER, '<phone␣
↪number>');
$userQuery = UserQuery::all()->withFilter(UserQuery::FILTER_UID, '<uid>');

$userQuery = ['filter' => [UserQuery::FILTER_EMAIL => '<email>'];
$userQuery = ['filter' => [UserQuery::FILTER_PHONE_NUMBER => '<email>'];
$userQuery = ['filter' => [UserQuery::FILTER_UID => '<email>'];
```

A user query will always return an array of `UserRecord` s. If none could be found, the array will be empty.

Note: Filters don't support partial matches, and only one filter can be applied at the same time. If you specify multiple filters, only the last one will be submitted.

2.8.4 Get information about a specific user

```php
try {
    $user = $auth->getUser('some-uid');
```

(continues on next page)

(continued from previous page)

```
    $user = $auth->getUserByEmail('user@domain.tld');
    $user = $auth->getUserByPhoneNumber('+49-123-456789');
} catch (\Kreait\Firebase\Exception\Auth\UserNotFound $e) {
    echo $e->getMessage();
}
```

2.8.5 Get information about multiple users

You can retrieve multiple user records by using `$auth->getUsers()`. When a user doesn't exist, no exception is thrown, but its entry in the result set is null:

```
$users = $auth->getUsers(['some-uid', 'another-uid', 'non-existing-uid']);
```

Result:

```
[
   'some-uid' => <UserRecord>,
   'another-uid' => <UserRecord>,
   'non-existing-uid' => null
]
```

2.8.6 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',
    // 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.

2.8.7 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
`deleteEmail`	boolean	Whether or not to delete the user's email.
`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`	string|array	One or more identity providers to delete.
`customAttributes`	array	A list of custom attributes which will be available in a User's ID token.

Note: When deleting the email from an existing user, the password authentication provider will be disabled (the user can't log in with an email and password combination anymore). After adding a new email to the same user, the previously set password might be restored. If you just want to change a user's email, consider updating the email field directly.

2.8.8 Change a user's password

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

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

2.8.9 Change a user's email

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

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

2.8.10 Disable a user

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

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

2.8.11 Enable a user

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

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

2.8.12 Custom user claims

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

```
// The new custom claims will propagate to the user's ID token the
// next time a new one is issued.
$auth->setCustomUserClaims($uid, ['admin' => true, 'key1' => 'value1']);

// Retrieve a user's current custom claims
$claims = $auth->getUser($uid)->customClaims;

// Remove a user's custom claims
$auth->setCustomUserClaims($uid, null);
```

The custom claims object should not contain any OIDC reserved key names or Firebase reserved names. Custom claims payload must not exceed 1000 bytes.

2.8.13 Delete a user

The Firebase Admin SDK allows deleting users by their `uid`:

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

try {
    $auth->deleteUser($uid);
catch (\Kreait\Firebase\Exception\Auth\UserNotFound $e) {
    echo $e->getMessage();
} catch (\Kreait\Firebase\Exception\AuthException $e) {
    echo 'Deleting
}
```

This method returns nothing when the deletion completes successfully. If the provided `uid` does not correspond to an existing user or the user cannot be deleted for any other reason, the delete user method throws an error.

2.8.14 Delete multiple users

The Firebase Admin SDK can also delete multiple (up to 1000) users at once:

```
$uid = ['uid-1', 'uid-2', 'uid-3'];
$forceDeleteEnabledUsers = true; // default: false

$result = $auth->deleteUsers($uids, $forceDeleteEnabledUsers);
```

By default, only disabled users will be deleted. If you want to also delete enabled users, use `true` as the second argument.

This method always returns an instance of `Kreait\Firebase\Auth\DeleteUsersResult`:

```
$successCount = $result->successCount();
$failureCount = $result->failureCount();
$errors = $result->rawErrors();
```

Note: Using this method to delete multiple users at once will not trigger `onDelete()` event handlers for Cloud Functions for Firebase. This is because batch deletes do not trigger a user deletion event on each user. Delete users one at a time if you want user deletion events to fire for each deleted user.

2.8.15 Duplicate/Unregistered email addresses

Some Firebase Authentication methods that take email addresses as parameters throw specific errors if the email address is unregistered when it must be registered (for example, when signing in with an email address and password), or registered when it must be unused (for example, when changing a user's email address).

If you try to create a new user, but the given email address has already been used before, the Firebase API returns an `EMAIL_EXISTS` error. On the other hand, if you try to sign in a user with an email address that hasn't been registered, the API returns an `EMAIL_NOT_FOUND` error.

You can handle both cases with the SDK:

```
try {
    $user = $auth->createUser([
        'email' => $email,
    ]);
} catch (\Kreait\Firebase\Exception\Auth\EmailExists $e) {
    echo $e->getMessage(); // "The email address is already in use by another account"
}

try {
    $signInResult = $auth->signInWithEmailAndPassword($email, $password);
} catch (\Kreait\Firebase\Exception\Auth\EmailNotFound $e) {
    echo $e->getMessage(); // "There is no user record corresponding to this␣
↪identifier. The user may have been deleted."
}
```

Note: Checking for existing/non-existing email addresses can be helpful for suggesting specific remedies to users, but it can also be abused by malicious actors to discover the email addresses registered by your users.

To mitigate this risk, Firebase recommends you enable email enumeration protection for your project using the Google Cloud `gcloud` tool. Note that enabling this feature changes Firebase Authentication's error reporting behavior: be sure your app doesn't rely on the more specific errors.

2.8.16 Using Email Action Codes

The Firebase Admin SDK provides the ability to send users emails containing links they can use for password resets, email address verification, and email-based sign-in. These emails are sent by Google and have limited customizability.

If you want to instead use your own email templates and your own email delivery service, you can use the Firebase Admin SDK to programmatically generate the action links for the above flows, which you can include in emails to your users.

Action Code Settings

Note: Action Code Settings are optional.

Action Code Settings allow you to pass additional state via a continue URL which is accessible after the user clicks the email link. This also provides the user the ability to go back to the app after the action is completed. In addition, you can specify whether to handle the email action link directly from a mobile application when it is installed or from a browser.

For links that are meant to be opened via a mobile app, you'll need to enable Firebase Dynamic Links and perform some tasks to detect these links from your mobile app. Refer to the instructions on how to configure Firebase Dynamic Links for email actions.

Parameter	Type	Description
`continueUrl`	string|null	Sets the continue URL
`url`	string|null	Alias for `continueUrl`
`handleCodeInApp`	bool|null	Whether the email action link will be opened in a mobile app or a web link first. The default is false. When set to true, the action code link will be be sent as a Universal Link or Android App Link and will be opened by the app if installed. In the false case, the code will be sent to the web widget first and then on continue will redirect to the app if installed.
`androidPackageName`	string|null	Sets the Android package name. This will try to open the link in an android app if it is installed.
`androidInstallApp`	bool|null	Whether to install the Android app if the device supports it and the app is not already installed. If this field is provided without a `androidPackageName`, an error is thrown explaining that the packageName must be provided in conjunction with this field.
`androidMinimumVersion`	string|null	If specified, and an older version of the app is installed, the user is taken to the Play Store to upgrade the app. The Android app needs to be registered in the Console.
`iOSBundleId`	string|null	Sets the iOS bundle ID. This will try to open the link in an iOS app if it is installed. The iOS app needs to be registered in the Console.

Example:

```
$actionCodeSettings = [
    'continueUrl' => 'https://www.example.com/checkout?cartId=1234',
    'handleCodeInApp' => true,
    'dynamicLinkDomain' => 'coolapp.page.link',
    'androidPackageName' => 'com.example.android',
    'androidMinimumVersion' => '12',
    'androidInstallApp' => true,
    'iOSBundleId' => 'com.example.ios',
];
```

Email verification

To generate an email verification link, provide the existing user's unverified email and optional Action Code Settings. The email used must belong to an existing user. Depending on the method you use, an email will be sent to the user, or you will get an email action link that you can use in a custom email.

```
$link = $auth->getEmailVerificationLink($email);
$link = $auth->getEmailVerificationLink($email, $actionCodeSettings);
$link = $auth->getEmailVerificationLink($email, $actionCodeSettings, $locale);

$auth->sendEmailVerificationLink($email);
$auth->sendEmailVerificationLink($email, $actionCodeSettings);
$auth->sendEmailVerificationLink($email, null, $locale);
$auth->sendEmailVerificationLink($email, $actionCodeSettings, $locale);
```

Password reset

To generate a password reset link, provide the existing user's email and optional Action Code Settings. The email used must belong to an existing user. Depending on the method you use, an email will be sent to the user, or you will get an email action link that you can use in a custom email.

```
$link = $auth->getPasswordResetLink($email);
$link = $auth->getPasswordResetLink($email, $actionCodeSettings);
$link = $auth->getPasswordResetLink($email, $actionCodeSettings, $locale);

$auth->sendPasswordResetLink($email);
$auth->sendPasswordResetLink($email, $actionCodeSettings);
$auth->sendPasswordResetLink($email, null, $locale);
$auth->sendPasswordResetLink($email, $actionCodeSettings, $locale);
```

Email link for sign-in

Note: Before you can authenticate users with email link sign-in, you will need to enable email link sign-in for your Firebase project.

Note: Unlike password reset and email verification, the email used does not necessarily need to belong to an existing user, as this operation can be used to sign up new users into your app via email link.

Note: The ActionCodeSettings object is required in this case to provide information on where to return the user after the link is clicked for sign-in completion.

To generate a sign-in link, provide the user's email and Action Code Settings. Depending on the method you use, an email will be sent to the user, or you will get an email action link that you can use in a custom email.

```
$link = $auth->getSignInWithEmailLink($email, $actionCodeSettings);

$auth->sendSignInWithEmailLink($email, $actionCodeSettings);
$auth->sendSignInWithEmailLink($email, $actionCodeSettings, $locale);
```

Confirm a password reset

Note: Out of the box, Firebase handles the confirmation of password reset requests. You can use your own server to handle account management emails by following the instructions on Customize account management emails and SMS messages

```
$oobCode = '...'; // Extract the OOB code from the request url (not scope of the SDK␣
↪(yet :)))
$newPassword = '...';
$invalidatePreviousSessions = true; // default, will revoke current user refresh␣
↪tokens

try {
    $auth->confirmPasswordReset($oobCode, $newPassword, $invalidatePreviousSessions);
} catch (\Kreait\Firebase\Exception\Auth\ExpiredOobCode $e) {
    // Handle the case of an expired reset code
} catch (\Kreait\Firebase\Exception\Auth\InvalidOobCode $e) {
    // Handle the case of an invalid reset code
} catch (\Kreait\Firebase\Exception\AuthException $e) {
    // Another error has occurred
}
```

2.9 Dynamic Links

You can create short Dynamic Links with the Firebase Admin SDK for PHP. Dynamic Links can be

- a long Dynamic Link
- an array containing Dynamic Link parameters
- an action created with builder methods

and will return a URL like `https://example.page.link/wXYZ`.

Note: Short Dynamic Links created via the REST API or this SDK do not show up in the Firebase console. Such Dynamic Links are intended for user-to-user sharing. For marketing use cases, continue to create your links directly through the Dynamic Links page of the Firebase console.

Before you start, please read about Dynamic Links in the official documentation:

- Dynamic Links Product Page
- Create Dynamic Links with the REST API
- Long Dynamic Links
- Dynamic Link API Reference

2.9.1 Getting started

- In the Firebase console, open the Dynamic Links section.
- If you have not already accepted the terms of service and set a domain for your Dynamic Links, do so when prompted.
- If you already have a Dynamic Links domain, take note of it. You need to provide a Dynamic Links Domain when you programmatically create Dynamic Links.

2.9.2 Initializing the Dynamic Links component

With the SDK

```
$dynamicLinksDomain = 'https://example.page.link';
$dynamicLinks = $factory->createDynamicLinksService($dynamicLinksDomain);
```

With Dependency Injection (Symfony Bundle/Laravel/Lumen Package)

To define the default Dynamic Links Domain for **Laravel**, configure the `FIREBASE_DYNAMIC_LINKS_DEFAULT_DOMAIN` environment variable.

```
use Kreait\Firebase\Contract\DynamicLinks;

class MyService
{
    public function __construct(DynamicLinks $dynamicLinks)
    {
        $this->dynamicLinks = $dynamicLinks;
    }
}
```

With the Laravel `app()` helper (Laravel/Lumen Package)

To define the default Dynamic Links Domain, configure the `FIREBASE_DYNAMIC_LINKS_DEFAULT_DOMAIN` environment variable.

```
$dynamicLinks = app('firebase.dynamic_links');
```

2.9.3 Create a Dynamic Link

You can create a Dynamic Link by using one of the methods below. Each method will return an instance of `Kreait\Firebase\DynamicLink`.

```
use use Kreait\Firebase\DynamicLink\CreateDynamicLink\FailedToCreateDynamicLink;

$url = 'https://www.example.com/some/path';

```

(continues on next page)

(continued from previous page)

```
try {
    $link = $dynamicLinks->createUnguessableLink($url);
    $link = $dynamicLinks->createDynamicLink($url, CreateDynamicLink::WITH_
↪UNGUESSABLE_SUFFIX);

    $link = $dynamicLinks->createShortLink($url);
    $link = $dynamicLinks->createDynamicLink($url, CreateDynamicLink::WITH_SHORT_
↪SUFFIX);
} catch (FailedToCreateDynamicLink $e) {
    echo $e->getMessage(); exit;
}
```

If `createDynamicLink()` is called without a second parameter, the Dynamic Link is created with an unguessable suffix.

Unguessable suffixes have a length of 17 characters, short suffixes a length of 4 characters. You can learn more about the length of Dynamic Links in the official documentation.

The returned object will be an instance of `Kreait\Firebase\DynamicLink` with the following accessors:

```
$link->uri();          // Psr\Http\Message\UriInterface
$link->previewUri();   // Psr\Http\Message\UriInterface
$link->domain();       // string
$link->suffix();       // string
$link->hasWarnings(); // bool
$link->warnings();     // array

$uriString = (string) $link;
```

2.9.4 Create a short link from a long link

If you have a manually constructed link, you can convert it to a short link:

```
use Kreait\Firebase\DynamicLink\ShortenLongDynamicLink\FailedToShortenLongDynamicLink;

$longLink = 'https://example.page.link?link=https://domain.tld/some/path';

try {
    $link = $dynamicLinks->shortenLongDynamicLink($longLink);
    $link = $dynamicLinks->shortenLongDynamicLink($longLink,␣
↪ShortenLongDynamicLink::WITH_UNGUESSABLE_SUFFIX);
    $link = $dynamicLinks->shortenLongDynamicLink($longLink,␣
↪ShortenLongDynamicLink::WITH_SHORT_SUFFIX);
} catch (FailedToShortenLongDynamicLink $e) {
    echo $e->getMessage(); exit;
}
```

If `shortenLongDynamicLink()` is called without a second parameter, the Dynamic Link is created with an unguessable suffix.

2.9.5 Get link statistics

You can use this REST API to get analytics data for each of your short Dynamic Links, whether created in the console or programmatically.

Note: These statistics might not include events that have been logged within the last 36 hours.

```
use␣
↪Kreait\Firebase\DynamicLink\GetStatisticsForDynamicLink\FailedToGetStatisticsForDynamicLink;
↪

try {
    $stats = $dynamicLinks->getStatistics('https://example.page.link/wXYZ');
    $stats = $dynamicLinks->getStatistics('https://example.page.link/wXYZ', 14); //␣
↪duration in days
} catch (FailedToGetStatisticsForDynamicLink $e) {
    echo $e->getMessage(); exit;
}
```

If `getStatistics()` is called without a second parameter, stats will include the statistics of the past 7 days.

The returned object will be an instance of `Kreait\Firebase\DynamicLink\DynamicLinkStatistics`, which currently only includes event statistics. You can access the raw returned data with *$stats->rawData()*.

Event Statistics

Firebase Dynamic Links tracks the number of times each of your short Dynamic Links have been clicked, as well as the number of times a click resulted in a redirect, app install, app first-open, or app re-open, including the platform on which that event occurred.

Each of the following methods returns a (filtered) instance of `Kreait\Firebase\DynamicLink\EventStatistics` which supports any combination of filters and is countable with `count()` or `->count()` as shown below:

```
$eventStats = $stats->eventStatistics();

$allClicks = $eventStats->clicks();
$allRedirects = $eventStats->redirects();
$allAppInstalls = $eventStats->appInstalls();
$allAppFirstOpens = $eventStats->appFirstOpens();
$allAppReOpens = $eventStats->appReOpens();

$allAndroidEvents = $eventStats->onAndroid();
$allDesktopEvents = $eventStats->onDesktop();
$allIOSEvents = $eventStats->onIOS();

$clicksOnDesktop = $eventStats->clicks()->onDesktop();
$appInstallsOnAndroid = $eventStats->onAndroid()->appInstalls();
$appReOpensOnIOS = $eventStats->appReOpens()->onIOS();

$totalAmountOfClicks = count($eventStats->clicks());
$totalAmountOfAppFirstOpensOnAndroid = $eventStats->appFirstOpens()->onAndroid()->
↪count();

$custom = $eventStats->filter(function (array $eventGroup) {
    return $eventGroup['platform'] === 'CUSTOM_PLATFORM_THAT_THE_SDK_DOES_NOT_KNOW_YET
↪';
});
```

2.9.6 Advanced usage

Using actions

You can fully customize the creation of Dynamic Links by building up a `Kreait\Firebase\DynamicLink\CreateDynamicLink` action. The following code shows all available building components:

```php
use Kreait\Firebase\DynamicLink\CreateDynamicLink;

$action = CreateDynamicLink::forUrl($url)
    ->withDynamicLinkDomain('https://example.page.link')
    ->withUnguessableSuffix() // default
    // or
    ->withShortSuffix()
    ->withAnalyticsInfo(
        AnalyticsInfo::new()
            ->withGooglePlayAnalyticsInfo(
                GooglePlayAnalytics::new()
                    ->withGclid('gclid')
                    ->withUtmCampaign('utmCampaign')
                    ->withUtmContent('utmContent')
                    ->withUtmMedium('utmMedium')
                    ->withUtmSource('utmSource')
                    ->withUtmTerm('utmTerm')
            )
            ->withItunesConnectAnalytics(
                ITunesConnectAnalytics::new()
                    ->withAffiliateToken('affiliateToken')
                    ->withCampaignToken('campaignToken')
                    ->withMediaType('8')
                    ->withProviderToken('providerToken')
            )
    )
    ->withNavigationInfo(
        NavigationInfo::new()
            ->withoutForcedRedirect() // default
            // or
            ->withForcedRedirect()
    )
    ->withIOSInfo(
        IOSInfo::new()
            ->withAppStoreId('appStoreId')
            ->withBundleId('bundleId')
            ->withCustomScheme('customScheme')
            ->withFallbackLink('https://fallback.domain.tld')
            ->withIPadBundleId('iPadBundleId')
            ->withIPadFallbackLink('https://ipad-fallback.domain.tld')
    )
    ->withAndroidInfo(
        AndroidInfo::new()
            ->withFallbackLink('https://fallback.domain.tld')
            ->withPackageName('packageName')
            ->withMinPackageVersionCode('minPackageVersionCode')
    )
    ->withSocialMetaTagInfo(
        SocialMetaTagInfo::new()
            ->withDescription('Social Meta Tag description')
            ->withTitle('Social Meta Tag title')
            ->withImageLink('https://domain.tld/image.jpg')
```

(continues on next page)

(continued from previous page)

```
    );

$link = $dynamicLinks->createDynamicLink($action);
```

Using parameter arrays

If you prefer using a parameter array to configure a Dynamic Link, or if this SDK doesn't yet have support for a given new option, you can pass an array to the `createDynamicLink()` method. As the parameters will not be processed or validated by the SDK, you have to make sure that the parameter structure matches the one described in the API Reference Documentation

```
use use Kreait\Firebase\DynamicLink\CreateDynamicLink\FailedToCreateDynamicLink;

$parameters = [
    'dynamicLinkInfo' => [
        'domainUriPrefix' => 'https://example.page.link',
        'link' => 'https://domain.tld/some/path',
    ],
    'suffix' => ['option' => 'SHORT'],
];

try {
    $link = $dynamicLinks->createDynamicLink($parameters);
} catch (FailedToCreateDynamicLink $e) {
    echo $e->getMessage(); exit;
}
```

2.10 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

2.10.1 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.

2.10.2 Initializing the Realtime Database component

With the SDK

```
$remoteConfig = $factory->createRemoteConfig();
```

With Dependency Injection (Symfony Bundle/Laravel/Lumen Package)

```
use Kreait\Firebase\Contract\RemoteConfig;

class MyService
{
    public function __construct(RemoteConfig $remoteConfig)
    {
        $this->remoteConfig = $remoteConfig;
    }
}
```

With the Laravel `app()` helper (Laravel/Lumen Package)

```
$remoteConfig = app('firebase.remote_config');
```

2.10.3 Get the Remote Config

```
$template = $remoteConfig->get(); // Returns a Kreait\Firebase\RemoteConfig\Template
$version = $template->version(); // Returns a Kreait\Firebase\RemoteConfig\Version
```

2.10.4 Create a new Remote Config

```
use Kreait\Firebase\RemoteConfig;

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

2.10.5 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

$frenchLanguageCondition = Condition::named('lang_french')
    ->withExpression("device.language in ['fr', 'fr_CA', 'fr_CH']")
    ->withTagColor(TagColor::GREEN);

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

$conditionNames = $template->conditionNames();
// Returns ['lang_german', 'lang_french']
```

2.10.6 Add a parameter

```
use Kreait\Firebase\RemoteConfig;

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

2.10.7 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)->withValue('Willkommen!');

$welcomeMessageParameter = RemoteConfig\Parameter::named('welcome_message')
        ->withDefaultValue('Welcome!')
        ->withConditionalValue($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.

2.10.8 Parameter Groups

```
use Kreait\Firebase\RemoteConfig;

$uiColors = RemoteConfig\ParameterGroup::named('UI Colors')
    ->withDescription('Remote configurable UI colors')
    ->withParameter(RemoteConfig\Parameter::named('Primary Color')->withDefaultValue(
↪'blue'))
    ->withParameter(RemoteConfig\Parameter::named('Secondary Color')->
↪withDefaultValue('red'))
;

$template = $template->withParameterGroup($parameterGroup);
```

2.10.9 Removing Remote Config Elements

You can remove elements from a Remote Config template with the following methods:

```
$template = Template::new()
    ->withCondition(Condition::named('condition'))
    ->withParameter(Parameter::named('parameter'))
```

(continues on next page)

(continued from previous page)

```
    ->withParameterGroup(ParameterGroup::named('group'))

$template = $template
    ->withRemovedCondition('condition')
    ->withRemovedParameter('parameter')
    ->withRemovedParameterGroup('group');
```

2.10.10 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.

2.10.11 Publish the Remote Config

```
use Kreait\Firebase\Exception\RemoteConfigException

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

2.10.12 Remote Config history

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

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 versions, this methods returns a Generator.

```
foreach ($auth->listVersions() as $version) {
    /** @var \Kreait\Firebase\RemoteConfig\Version $version */
    // ...
}

// or

array_map(function (\Kreait\Firebase\RemoteConfig\Version $version) {
    // ...
}, iterator_to_array($auth->listVersions()));
```

Filtering

You can filter the results of `RemoteConfig::listVersions()`:

```
use Kreait\Firebase\RemoteConfig\FindVersions;

$query = FindVersions::all()
    // Versions created/updated after August 1st, 2019 at midnight
    ->startingAt(new DateTime('2019-08-01 00:00:00'))
    // Versions created/updated before August 7th, 2019 at the end of the day
    ->endingAt(new DateTime('2019-08-06 23:59:59'))
    // Versions with version numbers smaller than 3464
    ->upToVersion(VersionNumber::fromValue(3463))
    // Setting a page size can results in faster first results,
    // but results in more request
    ->withPageSize(5)
    // Stop querying after the first 10 results
    ->withLimit(10)
;

// Alternative array notation

$query = [
    'startingAt' => '2019-08-01',
    'endingAt' => '2019-08-07',
    'upToVersion' => 9999,
    'pageSize' => 5,
    'limit' => 10,
];

foreach ($remoteConfig->listVersions($query) as $version) {
    echo "Version number: {$version->versionNumber()}\n";
    echo "Last updated at {$version->updatedAt()->format('Y-m-d H:i:s')}\n";
    // ...
    echo "\n---\n";
}
```

Get a specific version

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

Rollback to a version

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

2.11 App Check

The Firebase Admin SDK for PHP provides an API for verifying custom backends using Firebase App Check.

Before you start, please read about Firebase App Check in the official documentation:

- Introduction to Firebase App Check
- Verify App Check tokens from a custom backend (Client-side)
- Implement a custom App Check provider

2.11.1 Initializing the App Check component

With the SDK

```
$appCheck = $factory->createAppCheck();
```

With Dependency Injection (Symfony Bundle/Laravel/Lumen Package)

```
use Kreait\Firebase\Contract\AppCheck;

class MyService
{
    public function __construct(AppCheck $appCheck)
    {
        $this->appCheck = $appCheck;
    }
}
```

With the Laravel `app()` helper (Laravel/Lumen Package)

```
$appCheck = app('firebase.app_check');
```

2.11.2 Verify App Check Tokens

The Firebase Admin SDK has a built-in method for validating App Check tokens.

See https://firebase.google.com/docs/app-check/custom-resource-backend for more information.

```
use Kreait\Firebase\Exception\AppCheck\FailedToVerifyAppCheckToken;

$appCheckTokenString = '...';

```

(continues on next page)

(continued from previous page)

```
try {
    $appCheck->verifyToken($appCheckTokenString);
} catch (FailedToVerifyAppCheckToken $e) {
    // The token is invalid
}
```

2.11.3 Create a Custom Provider

The Firebase Admin SDK has a built-in method for creating custom provider of Firebase App Check tokens. It creates a custom token and then exchanges it for Firebase App Check token that can be sent back to the client.

See https://firebase.google.com/docs/app-check/custom-provider for more information.

```
$token = $appCheck->createToken("com.example.app-id");
```

2.12 Framework Integrations

kreait provides and maintains the following framework integrations for the Firebase Admin SDK for PHP:

2.12.1 Laravel

kreait/laravel-firebase

2.12.2 Symfony

kreait/firebase-bundle

2.12.3 CodeIgniter

tatter/firebase

2.13 Testing and Local Development

2.13.1 Integration Tests

The most reliable way of testing your project is to create a separate Firebase project and configure your tests to use it instead of the your production project. For example, you could have multiple Firebase projects depending on your use-case:

- `my-project-dev`: used for developers while they develop new features
- `my-project-int`: use by CI/CD pipelines
- `my-project-staging`: used to present upcoming to stake holders
- `my-project`: used for production

2.13.2 Using the Firebase Emulator Suite

For an introduction to the Firebase Emulator suite, please visit the official documentation: https://firebase.google.com/docs/emulator-suite

> **Warning:** Only the Auth and Realtime Database Emulators are currently supported in this PHP SDK.

To use the Firebase Emulator Suite, you must first install it.

See the official documentation for instructions how to work with it.

The emulator suite must be running otherwise the PHP SDK can't connect to it.

Auth Emulator

If not already present, create a *firebase.json* file in the root of your project and make sure that at least the following fields are set (the port number can be changed to your requirements):

```
{
  "emulators": {
    "auth": {
      "port": 9099
    }
  }
}
```

Firebase Admin SDKs automatically connect to the Authentication emulator when the `FIREBASE_AUTH_EMULATOR_HOST` environment variable is set.

```
$ export FIREBASE_AUTH_EMULATOR_HOST="localhost:9099"
```

With the environment variable set, Firebase Admin SDKs will accept unsigned ID Tokens and session cookies issued by the Authentication emulator (via `verifyIdToken` and `createSessionCookie` methods respectively) to facilitate local development and testing. Please make sure not to set the environment variable in production.

When connecting to the Authentication emulator, you will need to specify a project ID. You can pass a project ID to the Factory directly or set the `GOOGLE_CLOUD_PROJECT` environment variable. Note that you do not need to use your real Firebase project ID; the Authentication emulator will accept any project ID.

Realtime Database Emulator

If not already present, create a *firebase.json* file in the root of your project and make sure that at least the following fields are set (the port number can be changed to your requirements):

```
{
  "emulators": {
    "database": {
      "port": 9100
    }
  }
}
```

Note: The Realtime Database Emulator uses port `9000` by default. This port is also used by PHP-FPM, so it is recommended to chose one that differs to not run into conflicts.

Firebase Admin SDKs automatically connect to the Realtime Database emulator when the `FIREBASE_DATABASE_EMULATOR_HOST` environment variable is set.

```
$ export FIREBASE_DATABASE_EMULATOR_HOST="localhost:9100"
```

2.14 Troubleshooting

Note: This SDK works with immutable objects until noted otherwise. You can recognize these objects when they have a 'with*''method. In that case, please keep in mind that in order to get hold of the changes you made, you will have to use the result of that method, e.g. `$changedObject = $object->withChangedProperty();`.

2.14.1 Error handling

In general, if executing a method from the SDK doesn't throw an error, it is safe to assume that the requested operation has worked according to the motto "no news is good news". If you do get an error, it is good practice to wrap the problematic code in a try/catch (*try* an operation and handle possible errors by *catch* ing them):

```
use Kreait\Firebase\Exception\FirebaseException;
use Throwable;

try {
    // The operation you want to perform
    echo 'OK';
} catch (FirebaseException $e) {
    echo 'An error has occurred while working with the SDK: '.$e->getMessage;
} catch (Throwable $e) {
    echo 'A not-Firebase specific error has occurred: '.$e->getMessage;
}
```

This is especially useful when you encounter `Fatal error: Uncaught GuzzleHttp\Exception\ClientException` errors which are caused by the Google/Firebase APIs rejecting a request. Those errors are handled by the SDK and should be converted to instances of `Kreait\Firebase\Exception\FirebaseException`.

If you want to be sure to catch *any* error, catch `Throwable`.

2.14.2 Call to private/undefined method ...

If you receive an error like

```
Fatal error: Uncaught Error: Call to private method␣
↪Kreait\Firebase\ServiceAccount::fromJsonFile()
```

you have most likely followed a tutorial that is targeted at Version 4.x of this release and have code that looks like this:

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

$database = $firebase->getDatabase();
```

Change it to the following:

```
$factory = (new Factory)->withServiceAccount(__DIR__.'/google-service-account.json');

$database = $factory->createDatabase();
```

2.14.3 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.

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

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

2.14.5 Call to undefined function `openssl_sign()`

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

2.14.6 Default sound not played on message delivery

If you specified `'sound' => 'default'` in the message payload, try chaning it to `'sound' => "default"` - although single or double quotes shouldn't™ make a difference, it has been reported that this can solve the issue.

2.14.7 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
```

2.14.8 "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
- FCM Registration API: https://console.cloud.google.com/apis/library/fcmregistrations.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

Please also make sure that the Service Account you are using for your project has all necessary roles and permissions as described in the official documentation at Manage project access with Firebase IAM.

2.14.9 MultiCast SendReports are empty

This is an issue seen in XAMPP/WAMP environments and seems related to the cURL version shipped with the current PHP installation. Please ensure that cURL is installed with at least version **7.67** (preferably newer, version 7.70 is known to work).

You can check the currently installed cURL version by adding the following line somewhere in your code:

```
echo curl_version()['version']; exit;
```

To install a newer version of cURL, download the latest release from https://curl.haxx.se/ . From the unpacked archive in the `bin` folder, use the file ending with `libcurl*.dll` to overwrite the existing `libcurl*.dll` in the `ext` folder of your PHP installation and restart the environment.

If this issue occurs in other environments (e.g. Linux or MacOS), please ensure that you have the latest (minor) versions of PHP and cURL installed. If the problem persists, please open an issue in the issue tracker.

2.14.10 Proxy configuration

If you need to access the Firebase/Google APIs through a proxy, you can specify an according HTTP Client option while configuring the service factory: *HTTP Client Options*

2.14.11 Debugging

In order to debug HTTP requests to the Firebase/Google APIs, you can enable the factory's debug mode and provide an instance of `Psr\Log\LoggerInterface`. HTTP requests and responses will then be pushed to this logger with their full headers and bodies.

```
$factory = $factory->withHttpDebugLogger($logger);
```

If you want to make sure that the Factory has the configuration you expect it to have, call the `getDebugInfo()` method:

```
$factoryInfo = $factory->getDebugInfo();
```

The output will be something like this:

The private key of a service account will be redacted.